aws

Entwicklerhandbuch

Deep-Learning-AMI

Deep-Learning-AMI: Entwicklerhandbuch

Table of Contents

Was ist das AWS Deep Learning AMI?

Willkommen zum Benutzerhandbuch für das AWS Deep Learning AMI.

DasAWS Deep Learning AMI (DLAMI) ist Ihre zentrale Anlaufstelle für Deep Learning in der Cloud. Diese maßgeschneiderte Maschineninstanz ist in den meisten Amazon EC2 EC2-Regionen für eine Vielzahl von Instance-Typen verfügbar, von einer kleinen Instance nur für die CPU bis hin zu den neuesten leistungsstarken Multi-GPU-Instances. Es ist mit NVIDIA CUDA und NVIDIA cuDNN sowie den neuesten Versionen der beliebtesten Deep-Learning-Frameworks vorkonfiguriert.

Informationen zu diesem Handbuch

Diese Anleitung hilft Ihnen beim Starten und Verwenden des DLAMI. Er deckt mehrere häufige Anwendungsfälle für Deep-Learning ab – sowohl für Training als auch für Inferenz. Auch die Auswahl des richtigen AMIs für Ihren Zweck und des Instance-Typs werden besprochen. Das DLAMI enthält mehrere Tutorials für jedes der Frameworks. Es enthält auch Tutorials zu verteiltem Training, Debugging, Verwendung vonAWS Inferentia und anderen wichtigen Konzepten. Es enthält Anweisungen zur Konfiguration von Jupyter zum Ausführen der Tutorials in Ihrem Browser.

Voraussetzungen

Sie sollten mit Befehlszeilentools und Python-Grundkenntnissen vertraut sein, um den DLAMI erfolgreich auszuführen. Tutorials zur Verwendung der einzelnen Frameworks werden von den Frameworks selbst zur Verfügung gestellt. Dieser Leitfaden zeigt Ihnen jedoch, wie Sie die einzelnen Frameworks aktivieren und die entsprechenden Tutorials für den Einstieg finden.

Beispiele DLAMI DLAMI-Verwendungen

Erfahren Sie mehr über Deep Learning: Das DLAMI ist eine gute Wahl für das Lernen oder Unterrichten von maschinellem Lernen und Deep-Learning-Frameworks. Es vereinfacht die Fehlersuche bei den Installationen der einzelnen Frameworks und sorgt für einen reibungslosen Betrieb auf einem einzelnen Computer. Das DLAMI wird mit einem Jupyter-Notebook geliefert und erleichtert die Ausführung der von den Frameworks bereitgestellten Tutorials für Personen, die neu im Bereich maschinelles Lernen und Deep Learning sind.

App-Entwicklung: Wenn Sie ein App-Entwickler sind und daran interessiert sind, Deep Learning zu nutzen, damit Ihre Apps die neuesten Fortschritte in der KI nutzen, ist das DLAMI das perfekte

Testfeld für Sie. Jedes Framework enthält Tutorials zum Einstieg in Deep-Learning. Viele bieten Modellszenarien, mit denen Deep-Learning einfach ausprobiert werden kann, ohne selbst neuronale Netze erstellen zu müssen oder eines der Modelltrainings durchzuführen. Einige Beispiele zeigen Ihnen, wie Sie eine Bilderkennungsanwendung in wenigen Minuten erstellen können oder wie Sie eine Spracherkennungsanwendung für Ihren eigenen Chatbot erstellen können.

Maschinelles Lernen und Datenanalytik: Wenn Sie ein Datenwissenschaftler sind oder daran interessiert sind, Ihre Daten mit Hilfe von Deep-Learning zu verarbeiten, stellen Ihnen viele der Frameworks eine Unterstützung für R und Spark bereit. Sie umfassen Tutorials von einfachen Regressionen bis hin zum Aufbau skalierbarer Datenverarbeitungssysteme für Personalisierungs- und Prognosesysteme.

Forschung: Wenn Sie ein Forscher sind und ein neues Framework ausprobieren, ein neues Modell testen oder neue Modelle trainieren möchten, können das DLAMI und dieAWS Skalierungsmöglichkeiten die mühsame Installation und Verwaltung mehrerer Trainingsknoten erleichtern.

> **Note**
>
> Ihre erste Wahl könnte zwar darin bestehen, Ihren Instanztyp auf eine größere Instance mit mehr GPUs (bis zu 8) zu aktualisieren, Sie können jedoch auch horizontal skalieren, indem Sie einen Cluster von DLAMI-Instances erstellen. Weitere Informationen zu Cluster-Builds finden Sie in den entsprechenden Verwandte Informationen.

Funktionen von DLAMI

Vorinstallierte Frameworks

Derzeit gibt es zwei Hauptvarianten des DLAMI mit weiteren Variationen, die sich auf das Betriebssystem (OS) und die Softwareversionen beziehen:

- -Deep-Learning-AMI – Frameworks, die separat mit `conda`-Paketen und separaten Python-Umgebungen installiert werden.
- Deep Learning Learning Learning Learning Learning Learning Learning Learning- keine Frameworks installiert; nur NVIDIA CUDA und andere Abhängigkeiten

Das Deep Learning-AMI mit Conda verwendet`conda` Umgebungen, um jedes Framework zu isolieren, sodass Sie nach Belieben zwischen ihnen wechseln können, ohne sich Gedanken über widersprüchliche Abhängigkeiten machen zu müssen.

Dies ist die vollständige Liste der von Deep Learning AMI mit Conda unterstützten Frameworks:

- Apache MXNet (Inkubation)
- PyTorch
- TensorFlow 2

> **Note**
>
> Die Umgebungen CNTK, Caffe, Caffe2, Theano, Chainer oder Keras Conda sind zuAWS Deep Learning AMI Beginn der Version v28 nicht mehr enthalten. Frühere Versionen von AWS Deep Learning AMI, die diese Umgebungen enthalten, sind weiterhin verfügbar. Wir werden jedoch nur Updates für diese Umgebungen bereitstellen, wenn von der Open-Source-Community Sicherheitskorrekturen für diese Frameworks veröffentlicht werden.

Vorinstallierte GPU-Software

Selbst wenn Sie eine reine CPU-Instanz verwenden, verfügt der DLAMI über NVIDIA CUDA und NVIDIA cuDNN. Die installierte Software ist unabhängig vom Instance-Typ identisch. Beachten Sie, dass GPU-spezifische Tools nur in einer Instance funktionieren, die mindestens eine GPU umfasst. Weitere Informationen hierzu finden Sie in Auswahl des Instance-Typs für DLAMI.

Weitere Informationen zur CUDA-Installation finden Sie unterCUDA-Installationen und Framework-Bindungen.

Model-Bereitstellung und Visualisierung

Deep Learning AMI mit Conda ist mit zwei Arten von Modellservern vorinstalliert, einer für TensorFlow MXNet und einer für sowie TensorBoard für Modellvisualisierungen.

- Modellserver für Apache MXNet (MMS)
- TensorFlow Servieren
- TensorBoard

Erste Schritte

Wie fange ich mit dem DLAMI an

Dieser Leitfaden enthält Tipps zur Auswahl des für Sie richtigen DLAMI, zur Auswahl eines Instanztyps, der zu Ihrem Anwendungsfall und Budget passtVerwandte Informationen, und beschreibt benutzerdefinierte Setups, die von Interesse sein könnten.

Wenn Sie mit der VerwendungAWS oder Verwendung von Amazon EC2 noch nicht vertraut sind, beginnen Sie mit dem-Deep-Learning-AMI. Wenn Sie mit Amazon EC2 und anderenAWS Diensten wie Amazon EMR, Amazon EFS oder Amazon S3 vertraut sind und daran interessiert sind, diese Services für Projekte zu integrieren, die verteiltes Training oder Inferenz erfordern, schauen Sie sichVerwandte Informationen an, ob einer für Ihren Anwendungsfall geeignet ist.

Wir empfehlen Ihnen, sich zuerst über Wählen Sie Ihr DLAMI zu informieren und so den am besten geeigneten Instance-Typ zu bestimmen.

Eine weitere Option ist dieses kurze Tutorial: Starten Sie einenAWS Deep Learning AMI (in 10 Minuten).

Nächster Schritt

Wählen Sie Ihr DLAMI

Wählen Sie Ihr DLAMI

Wir bieten eine Reihe von DLAMI-Optionen an. Um Ihnen bei der Auswahl des richtigen DLAMI für Ihren Anwendungsfall zu helfen, gruppieren wir Images nach dem Hardwaretyp oder der Funktionalität, für die sie entwickelt wurden. Unsere Top-Gruppierungen sind:

- DLAMI-Typ: CUDA versus Base versus Single-Framework versus Multi-Framework (Conda DLAMI)
- Rechenarchitektur: x86-basiertes versus ARM-basiertes AWSGraviton
- Prozessortyp: GPU gegen CPU gegen Inferentia gegen Habana
- SDK: CUDA gegen AWSNeuron gegen SynapsesAI
- Betriebssystem: Amazon Linux gegen Ubuntu

Die restlichen Themen in diesem Handbuch helfen Ihnen, sich weiter zu informieren und gehen näher darauf ein.

Themen

Nächstes Thema

-Deep-Learning-AMI

CUDA-Installationen und Framework-Bindungen

Deep Learning ist zwar ziemlich modern, aber jedes Framework bietet „stabile“ Versionen. Diese stabilen Versionen funktionieren unter Umständen nicht mit den neuesten CUDA- oder cuDNN-Implementierungen und -Funktionen. Ihr Anwendungsfall und die Funktionen, die Sie benötigen, können Ihnen bei der Auswahl eines Frameworks helfen. Wenn Sie sich nicht sicher sind, verwenden Sie das neueste Deep Learning AMI mit Conda. Es hat offizielle`pip` Binärdateien für alle Frameworks mit CUDA 10, wobei die neueste Version verwendet wird, die von jedem Framework unterstützt wird. Wenn Sie die neuesten Versionen benötigen und Ihre Deep-Learning-Umgebung anpassen möchten, verwenden Sie das Deep Learning Base-AMI.

Weitere Informationen finden Sie in unserem Handbuch in Stabile und Release-Kandidaten.

Wählen Sie ein DLAMI mit CUDA

DerDeep Learning Learning Learning Learning Learning Learning Learning Learning verfügt über alle verfügbaren CUDA 11-Serien, einschließlich 11.0, 11.1 und 11.2.

Der-Deep-Learning-AMI verfügt über alle verfügbaren CUDA 11-Serien, einschließlich 11.0, 11.1 und 11.2.

> **Note**
>
> Die Umgebungen CNTK, Caffe, Caffe2, Theano, Chainer oder Keras Conda sind zuAWS Deep Learning AMI Beginn der Version v28 nicht mehr enthalten. Frühere Versionen

vonAWS Deep Learning AMI, die diese Umgebungen enthalten, sind weiterhin verfügbar. Wir stellen jedoch nur Updates für diese Umgebungen bereit, wenn von der Open-Source-Community Sicherheitskorrekturen für diese Frameworks veröffentlicht wurden.

Informationen zu bestimmten Framework-Versionsnummern finden Sie unterVersionshinweise für DLAMI

Wählen Sie diesen DLAMI-Typ oder erfahren Sie mit der Option Next Up mehr über die verschiedenen DLAMIs.

Wählen Sie eine der CUDA-Versionen aus und sehen Sie sich die vollständige Liste der DLAMIs mit dieser Version im Anhang an, oder erfahren Sie mehr über die verschiedenen DLAMIs mit der Option Next Up.

Nächstes Thema

Deep Learning Learning Learning Learning Learning Learning Learning Learning

Verwandte Themen

- Anweisungen zum Wechseln zwischen CUDA-Versionen finden Sie im Verwenden des Deep Learning Base AMI-Tutorial.

Deep Learning Learning Learning Learning Learning Learning Learning Learning

Das Deep Learning Base AMI ist wie eine leere Leinwand für Deep Learning. Es enthält alles, was Sie bis zur Installation eines bestimmten Frameworks benötigen, und bietet Ihnen die Wahl zwischen CUDA-Versionen.

Warum sollten Sie sich für den Basis-DLAMI entscheiden

Diese AMI-Gruppe ist nützlich für Projektbeteiligte, die ein Deep-Learning-Projekt aufspalten und das aktuelle erstellen möchten. Es ist für Entwickler gedacht, die ihre eigene Umgebung mit der Gewissheit bereitstellen möchten, dass die neueste NVIDIA-Software installiert ist und funktioniert, damit sie ungestört auswählen können, welche Frameworks und Versionen sie installieren möchten.

Wählen Sie diesen DLAMI-Typ oder erfahren Sie mit der Option Next Up mehr über die verschiedenen DLAMIs.

Nächstes Thema

DLAMI mit Conda

Verwandte Themen

- Verwenden des Deep Learning Base-AMI

-Deep-Learning-AMI

Der Conda DLAMI verwendetconda virtuelle Umgebungen. Diese Umgebungen sind so konfiguriert, dass die verschiedenen Framework-Installationen getrennt bleiben und das Umschalten zwischen Frameworks optimiert wird. Dies eignet sich hervorragend zum Lernen und Experimentieren mit allen Frameworks, die der DLAMI zu bieten hat. Die meisten Benutzer finden, dass das neue Deep Learning AMI mit Conda perfekt für sie ist.

Diese AMIs sind die primären DLAMIs. Sie werden häufig mit den neuesten Versionen der Frameworks aktualisiert und verfügen über die neuesten GPU-Treiber und Software. Sie werden in den meisten Dokumenten allgemein als dieAWS Deep Learning AMI bezeichnet.

- Der Ubuntu 18.04 DLAMI hat die folgenden Frameworks: Apache MXNet (Incubating) PyTorch, und TensorFlow 2.
- Das Amazon Linux 2-DLAMI verfügt über die folgenden Frameworks: Apache MXNet (Incubating) PyTorch, und TensorFlow 2.

> **Note**
>
> Wir schließen die Umgebungen CNTK, Caffe, Caffe2, Theano, Chainer und Keras CondaAWS Deep Learning AMI ab Version 28 nicht mehr mit ein. Frühere Versionen vonAWS Deep Learning AMI, die diese Umgebungen enthalten, sind weiterhin verfügbar. Wir stellen jedoch nur Updates für diese Umgebungen bereit, wenn von der Open-Source-Community Sicherheitskorrekturen für diese Frameworks veröffentlicht wurden.

Stabile und Release-Kandidaten

Die Conda AMIs verwenden optimierte Binärdateien der neuesten formellen Versionen jedes Frameworks. Release-Kandidaten und experimentelle Funktionen sind nicht zu erwarten. Die

Optimierungen hängen von der Unterstützung des Frameworks für Beschleunigungstechnologien wie Intels MKL DNN ab, das das Training und die Inferenz von C5- und C4-CPU-Instanztypen beschleunigt. Die Binärdateien wurden auch so kompiliert, dass sie erweiterte Intel-Befehlssätze unterstützen, einschließlich, aber nicht beschränkt auf AVX, AVX-2, SSE4.1 und SSE4.2. Diese beschleunigen Vektor- und Gleitkommaoperationen auf Intel-CPU-Architekturen. Darüber hinaus werden CUDA und cuDNN für GPU-Instanztypen mit der Version aktualisiert, die die neueste offizielle Version unterstützt.

Das Deep Learning AMI mit Conda installiert bei der ersten Aktivierung des Frameworks automatisch die optimierteste Version des Frameworks für Ihre Amazon EC2 EC2-Instance. Weitere Informationen finden Sie unter Verwenden des Deep Learning-AMI mit Conda.

Wenn Sie mit benutzerdefinierten oder optimierten Build-Optionen aus dem Quellcode installieren möchten,Deep Learning Learning Learning Learning Learning Learning Learning Learning ist das s möglicherweise die bessere Option für Sie.

Beendigung von Python 2

Die Python-Open-Source-Community hat die Unterstützung für Python 2 am 1.Januar 2020 offiziell beendet. Die TensorFlow und PyTorch Community haben angekündigt, dass die Versionen TensorFlow 2.1 und PyTorch 1.4 die letzten sind, die Python 2 unterstützen. Frühere Versionen des DLAMI (v26, v25 usw.), die Python 2 Conda-Umgebungen enthalten, sind weiterhin verfügbar. Wir stellen Updates für die Python 2 Conda-Umgebungen für zuvor veröffentlichte DLAMI-Versionen jedoch nur bereit, wenn von der Open-Source-Community für diese Versionen Sicherheitskorrekturen veröffentlicht wurden. DLAMI-Versionen mit den neuesten Versionen der TensorFlow PyTorch AND-Frameworks enthalten nicht die Python-2-Conda-Umgebungen.

CUDA Support

Spezifische CUDA-Versionsnummern finden Sie in den GPU-DLAMI-Versionshinweisen.

Nächstes Thema

Optionen für die DLAMI-CPU-Architektur

Verwandte Themen

- Ein Tutorial zur Verwendung eines Deep Learning-AMI mit Conda finden Sie imVerwenden des Deep Learning-AMI mit Conda Tutorial.

Optionen für die DLAMI-CPU-Architektur

AWS Deep Learning AMIs werden entweder mit x86-basierten oder ARM-basierten AWSGraviton2-CPU-Architekturen angeboten.

Wählen Sie eines der Graviton-GPU-DLAMIs, um mit einer ARM-basierten CPU-Architektur zu arbeiten. Alle anderen GPU-DLAMIs sind derzeit x86-basiert.

- AWSDeep-Learning-AMI Graviton-GPU CUDA 11.4 (Ubuntu 20.04)
- AWSDeep-Learning-AMI Graviton-GPU TensorFlow 2.6 (Ubuntu 20.04)
- AWSDeep-Learning-AMI Graviton-GPU PyTorch 1.10 (Ubuntu 20.04)

Weitere Informationen zu den ersten Schritten mit dem Graviton GPU DLAMI finden Sie unterDas Graviton DLAMI. Weitere Informationen zu verfügbaren Instanztypen finden Sie unterAuswahl des Instance-Typs für DLAMI.

Nächstes Thema

DLAMI Betriebssysteme

DLAMI Betriebssysteme

DLAMs werden in den folgenden Betriebssystemen.

- Amazon Linux 2
- Ubuntu 20.04
- Ubuntu 18.04

Ältere Versionen von Betriebssystemen sind auf veralteten DLAMIs verfügbar. Weitere Informationen zur Deprecation von DLAMI finden Sie unter Deprecations for DLAMI

Bevor Sie sich für einen DLAMI entscheiden, sollten Sie prüfen, welchen Instanztyp Sie benötigen, und identifizieren Sie IhreAWS Region.

Nächstes Thema

Auswahl des Instance-Typs für DLAMI

Auswahl des Instance-Typs für DLAMI

Im AWS Deep Learning AMIKatalog finden Sie empfohlene Amazon EC2 EC2-Instance-Familien, die mit bestimmten DLAMIs kompatibel sind.

Generell sollten Sie bei der Auswahl eines Instance-Typs für einen DLAMI Folgendes berücksichtigen.

- Wenn Deep Learning für Sie neu ist, könnte eine Instanz mit einer einzigen GPU Ihren Anforderungen entsprechen.
- Wenn Sie budgetbewusst sind, können Sie reine CPU-Instances verwenden.
- Wenn Sie die hohe Leistung und Kosteneffizienz für die Deep-Learning-Modellinferenz optimieren möchten, können Sie Instances mit AWS Inferentia-Chips verwenden.
- Wenn Sie die hohe Leistung und Kosteneffizienz für das Deep-Learning-Modelltraining optimieren möchten, können Sie Instances mit Habana-Beschleunigern verwenden.
- Wenn Sie nach einer leistungsstarken GPU-Instance mit einer ARM-basierten CPU-Architektur suchen, können Sie den G5G-Instance-Typ verwenden.
- Wenn Sie daran interessiert sind, ein vortrainiertes Modell für Inferenz und Vorhersagen auszuführen, können Sie Ihrer Amazon EC2-Instance eine Amazon Elastic Inference zuordnen. Amazon Elastic Inference bietet Ihnen Zugriff auf einen Beschleuniger mit einem Bruchteil einer GPU.
- Für Inferenzdienste mit hohem Volumen ist eine einzelne CPU-Instanz mit viel Speicher oder ein Cluster solcher Instanzen möglicherweise die bessere Lösung.
- Wenn Sie ein großes Modell mit vielen Daten oder einer hohen Batchgröße verwenden, benötigen Sie eine größere Instanz mit mehr Speicher. Sie können Ihr Modell auch auf einen Cluster von GPUs verteilen. Möglicherweise stellt auch eine Instance mit weniger Speicher eine bessere Lösung für Sie dar, wenn Sie Ihre Batchgröße verringern. Dies kann sich auf die Genauigkeit und Schulungsgeschwindigkeit auswirken.
- Wenn Sie daran interessiert sind, Anwendungen für maschinelles Lernen mit der NVIDIA Collective Communications Library (NCCL) auszuführen, die eine hohe Kommunikation zwischen den Knoten erfordern Elastic Fabric Adapter (EFA) von

Weitere Informationen zu Instances finden Sie unter .

Die folgenden Themen enthalten Informationen zu den Überlegungen zu den Instance-Typen.

> **Important**
>
> Die Deep Learning AMIs beinhalten Treiber, Software oder Toolkits, die von der NVIDIA Corporation entwickelt wurden, ihr Eigentum sind oder von ihr bereitgestellt werden. Sie erklären, diese NVIDIA-Treiber, -Software oder -Toolkits nur auf Amazon EC2-Instances zu verwenden, die NVIDIA-Hardware enthalten.

Themen

Preise für den DLAMI

Die im DLAMI enthaltenen Deep-Learning-Frameworks sind kostenlos und jedes hat seine eigenen Open-Source-Lizenzen. Obwohl die im DLAMI enthaltene Software kostenlos ist, müssen Sie dennoch für die zugrunde liegende Amazon EC2 EC2-Instance-Hardware bezahlen.

Einige Amazon EC2 EC2-Instance-Typen sind als kostenlos gekennzeichnet. Es ist möglich, den DLAMI auf einer dieser kostenlosen Instanzen auszuführen. Das bedeutet, dass die Nutzung des DLAMI völlig kostenlos ist, wenn Sie nur die Kapazität dieser Instanz nutzen. Wenn Sie eine leistungsfähigere Instanz mit mehr CPU-Kernen, mehr Festplattenspeicher, mehr RAM oder einer oder mehreren GPUs benötigen, benötigen Sie eine Instanz, die nicht zur Free-Tier-Instance-Klasse gehört.

Weitere Informationen zur Auswahl und den Preisen finden Sie unter Amazon EC2 — Preise.

Verfügbarkeit der DLAMI-Region

Jede Region unterstützt unterschiedliche Instance-Typen und oft fallen für einen Instance-Typ in verschiedenen Regionen leicht unterschiedliche Kosten an. DLAMIs sind nicht in jeder Region

verfügbar, aber es ist möglich, DLAMIs in die Region Ihrer Wahl zu kopieren. Weitere Informationen finden Sie unter AMI kopieren. Beachten Sie die Auswahlliste für Regionen und stellen Sie sicher, dass Sie eine Region auswählen, die sich in Ihrer Nähe oder Ihren Kunden befindet. Wenn Sie planen, mehr als einen DLAMI zu verwenden und möglicherweise einen Cluster zu erstellen, stellen Sie sicher, dass Sie dieselbe Region für alle Knoten im Cluster verwenden.

Weitere Informationen zu Regionen finden Sie unter .

Nächstes Thema

Empfohlene GPU-Instances

Empfohlene GPU-Instances

Wir empfehlen für die meisten Deep-Learning-Zwecke eine GPU-Instanz. Das Training neuer Modelle ist auf einer GPU-Instanz schneller als auf einer CPU-Instanz. Sie können sublinear skalieren, wenn Sie über mehrere GPU-Instances verfügen oder wenn Sie eine Schulung für viele Instances mit GPUs verwenden. Informationen zum Einrichten verteilter Schulungen finden Sie unterVerteilte Schulungen.

Die folgenden Instance-Typen unterstützen den DLAMI. Informationen zu den Optionen für GPU-Instance-Typen und deren Verwendung finden Sie unter und wählen Sie Accelerated Computing aus.

> **Note**
>
> Bei der Auswahl einer Instance sollte die Größe Ihres Modells berücksichtigt werden. Wenn Ihr Modell den verfügbaren Arbeitsspeicher einer Instance überschreitet, wählen Sie für Ihre Anwendung einen anderen Instance-Typ mit ausreichend Speicher.

- Amazon EC2 P3-Instances verfügen über bis zu 8 NVIDIA Tesla V100-GPUs.
- Amazon EC2 P4-Instances verfügen über bis zu 8 NVIDIA Tesla A100-GPUs.
- Amazon EC2 G3-Instances verfügen über bis zu 4 NVIDIA Tesla M60-GPUs.
- Amazon EC2 G4-Instances verfügen über bis zu 4 NVIDIA-T4-GPUs.
- Amazon EC2 G5 Instances verfügen über bis zu 8 NVIDIA A10G-GPUs.
- Amazon EC2 G5G-Instances verfügen über AWS ARM-basierte Graviton2-Prozessoren.

DLAMI-Instances bieten Tools zur Überwachung und Optimierung Ihrer GPU-Prozesse. Weitere Informationen zur Überwachung Ihrer GPU-Prozesse finden Sie unterGPU-Überwachung und -Optimierung.

Spezifische Tutorials zur Arbeit mit G5G-Instances finden Sie unterDas Graviton DLAMI.

Nächstes Thema

Empfohlene CPU-Instances

Empfohlene CPU-Instances

Egal, ob Sie nur ein begrenztes Budget haben, etwas über Deep Learning lernen oder einfach nur einen Vorrausage-Service betreiben möchten, Sie haben viele günstige Optionen in der CPU-Kategorie. Einige Frameworks nutzen Intels MKL DNN, wodurch das Training und die Inferenz auf den CPU-Instance-Typen C5 (nicht in allen Regionen verfügbar), C4 und C3 beschleunigt werden. Informationen zu CPU-Instance-Typen finden Sie unter und wählen Sie Compute Optimized aus.

> **Note**
>
> Bei der Auswahl einer Instance sollte die Größe Ihres Modells berücksichtigt werden. Wenn Ihr Modell den verfügbaren Arbeitsspeicher einer Instance überschreitet, wählen Sie für Ihre Anwendung einen anderen Instance-Typ mit ausreichend Speicher.

- Amazon EC2 C5 Instances verfügen über bis zu 72 Intel vCPUs. C5-Instances zeichnen sich durch wissenschaftliche Modellierung, Stapelverarbeitung, verteilte Analytik, Hochleistungsrechnen (HPC) sowie Inferenz für maschinelles und tiefes Lernen aus.
- Amazon EC2 C4-Instances haben bis zu 36 Intel-vCPUs.

Nächstes Thema

Empfohlener Instances

Empfohlener Instances

AWSInstances von Inferentia-Instances sind so konzipiert, dass sie hohe Leistung und Kosteneffizienz für Deep-Learning-Model-Inference-Workloads bieten. Insbesondere verwenden Inf2-Instanztypen AWS Inferentia-Chips und das AWSNeuron SDK, das in beliebte Frameworks für maschinelles Lernen wie und integriert ist. TensorFlow PyTorch

Kunden können Inf2-Instances verwenden, um umfangreiche Inferenzanwendungen für maschinelles Lernen wie Suchen, Empfehlungsmaschinen, Computer Vision, Spracherkennung, Verarbeitung natürlicher Sprache, Personalisierung und Betrugserkennung zu den niedrigsten Kosten in der Cloud auszuführen.

> **Note**
>
> Bei der Auswahl einer Instance sollte die Größe Ihres Modells berücksichtigt werden. Wenn Ihr Modell den verfügbaren Arbeitsspeicher einer Instance überschreitet, wählen Sie für Ihre Anwendung einen anderen Instance-Typ mit ausreichend Speicher.

- Amazon EC2 Inf2-Instances verfügen über bis zu 16 AWS Inferentia-Chips und einen Netzwerkdurchsatz von 100 Gbit/s.

Weitere Informationen zu den ersten Schritten mit AWS Inferentia dLamis finden Sie unter. Der AWS Inferentia-Chip mit DLAMI

Nächstes Thema

Empfohlener Trainium-Instances

Empfohlener Trainium-Instances

AWSTrainium-Instances sind so konzipiert, dass sie hohe Leistung und Kosteneffizienz für Deep-Learning-Modell-Inference-Workloads bieten. Insbesondere verwenden Trn1-Instanztypen AWS Trainium-Chips und das AWSNeuron SDK, das in beliebte Frameworks für maschinelles Lernen wie und integriert ist. TensorFlow PyTorch

Kunden können Trn1-Instances verwenden, um umfangreiche Inferenzanwendungen für maschinelles Lernen wie Suchen, Empfehlungsmaschinen, Computer Vision, Spracherkennung, Verarbeitung natürlicher Sprache, Personalisierung und Betrugserkennung auszuführen, und das zu den niedrigsten Kosten in der Cloud.

> **Note**
>
> Bei der Auswahl einer Instance sollte die Größe Ihres Modells berücksichtigt werden. Wenn Ihr Modell den verfügbaren Arbeitsspeicher einer Instance überschreitet, wählen Sie für Ihre Anwendung einen anderen Instance-Typ mit ausreichend Speicher.

- Amazon EC2 Trn1-Instances verfügen über bis zu 16 AWS Trainium-Chips und einen Netzwerkdurchsatz von 100 Gbit/s.

Nächstes Thema

Empfohlener Habana-Instances

Empfohlener Habana-Instances

Instances mit Habana-Beschleunigern sind so konzipiert, dass sie hohe Leistung und Kosteneffizienz für Deep-Learning-Modell-Training-Workloads bieten. Insbesondere DL1-Instance-Typen verwenden Habana-Gaudi-Beschleuniger von Habana Labs, einem Unternehmen von Intel, verwenden. DL1-Instances sind ideal für das Training von Modellen des maschinellen Lernens, die in Anwendungen wie Verarbeitung natürlicher Sprache, Objekterkennung und -klassifizierung, Autonome Fahrzeugerkennung und -klassifizierung, Autonome Fahrzeugerkennung und -klassifizierung, Fahrzeugerkennung und -klassifizierung, Auton

Instanzen mit Habana-Beschleunigern werden mit der Habana SynapseAI-Software konfiguriert und in gängige Frameworks für maschinelles Lernen wie und vorintegriert. TensorFlow PyTorch Wenn Sie nach einer optimalen Kombination aus Leistung und Preis für das Training von Deep-Learning-Modellen suchen, sollten Sie Instances mit Habana-Beschleunigern in Betracht ziehen, da die Trainingskosten am niedrigsten sind.

> **Note**
>
> Bei der Auswahl einer Instance sollte die Größe Ihres Modells berücksichtigt werden. Wenn Ihr Modell den verfügbaren Arbeitsspeicher einer Instance überschreitet, wählen Sie für Ihre Anwendung einen anderen Instance-Typ mit ausreichend Speicher.

- Amazon EC2 DL1-Instances verfügen über bis zu acht Habana Gaudi-Beschleuniger, 256 GB Beschleunigerspeicher, 4 TB lokalen NVMe-Speicher und 400 Gbit/s Netzwerkdurchsatz.

Weitere Informationen zu den ersten Schritten mit Habana dLamis finden Sie unter. Das Habana DLAMI

Richtlinie zur Support von Frameworks

AWS Deep Learning AMIs (DLAMIs) vereinfachen die Image-Konfiguration für Deep-Learning-Workloads und sind für die neuesten Frameworks, Hardware, Treiber, Bibliotheken und Betriebssysteme optimiert. Auf dieser Seite werden die Richtlinien zur Framework-Supportunterstützung für DLAMIs beschrieben. Eine Liste der verfügbaren DLAMIs finden Sie in den Versionshinweisen für DLAMI.

Unterstützte Frameworks

In der folgenden Tabelle mit den AWS Deep Learning AMI Framework-Supportrichtlinien können Sie überprüfen, welche Frameworks und Versionen aktiv unterstützt werden.

Unter Ende des Patches erfahren Sie, wie lange aktuelle Versionen AWS unterstützt werden, die aktiv vom Wartungsteam des Origin-Frameworks unterstützt werden. Frameworks und Versionen sind als DLAMIs mit einem Framework oder als DLAMIs mit mehreren Frameworks verfügbar.

> **Note**
>
> In der Framework-Version x.y.z bezieht sich x auf die Hauptversion, y auf die Nebenversion und z auf die Patch-Version. Für TensorFlow 2.6.5 ist die Hauptversion beispielsweise 2, die Nebenversion ist 6 und die Patch-Version ist 5.

Weitere Informationen zu bestimmten Bildern finden Sie in den Versionshinweisen:

- Versionshinweise zu DLAMI für ein einzelnes Framework
- Versionshinweise zu DLAMI für mehrere Frameworks

Häufig gestellte Fragen

- Welche Framework-Versionen erhalten Sicherheitspatches?
- Welche Images werden AWS veröffentlicht, wenn neue Framework-Versionen veröffentlicht werden?
- Welche Bilder erhalten neue SageMaker AWS /Funktionen?

Welche Framework-Versionen erhalten Sicherheitspatches?

Wenn die Framework-Version in der Tabelle der AWS Deep Learning AMI Framework-Support-Richtlinien als Unterstützt gekennzeichnet ist, erhält sie Sicherheitspatches.

Welche Images werden AWS veröffentlicht, wenn neue Framework-Versionen veröffentlicht werden?

Wir veröffentlichen neue DLAMIs kurz nach der Veröffentlichung neuer Versionen von TensorFlow und PyTorch. Dazu gehören Hauptversionen, Haupt-Nebenversionen und major-minor-patch Versionen von Frameworks. Wir aktualisieren auch Images, wenn neue Versionen von Treibern und Bibliotheken verfügbar werden. Weitere Informationen zur Image-Wartung finden Sie unter Wann endet der aktive Support für meine Framework-Version?

Welche Bilder erhalten neue SageMaker AWS /Funktionen?

Neue Funktionen werden normalerweise in der neuesten Version von DLAmis für PyTorch und veröffentlicht. TensorFlow Einzelheiten zu neuen SageMaker Funktionen finden Sie in den Versionshinweisen für ein bestimmtes Bild. AWS Eine Liste der verfügbaren DLAMIs finden Sie in

den Versionshinweisen für DLAMI. Weitere Informationen zur Image-Wartung finden Sie unter Wann endet der aktive Support für meine Framework-Version?

Wie ist die aktuelle Version in der Tabelle Unterstützte Frameworks definiert?

Die aktuelle Version in der Tabelle AWS Deep Learning AMI Framework Support Policy bezieht sich auf die neueste Framework-Version, die AWS am verfügbar ist GitHub. Jede neueste Version enthält Updates für die Treiber, Bibliotheken und relevanten Pakete im DLAMI. Informationen zur Image-Wartung finden Sie unter Wann endet der aktive Support für meine Framework-Version?

Was ist, wenn ich eine Version verwende, die nicht in der Tabelle Unterstützte Frameworks aufgeführt ist?

Wenn Sie eine Version ausführen, die nicht in der Tabelle der AWS Deep Learning AMI Framework-Supportrichtlinien aufgeführt ist, verfügen Sie möglicherweise nicht über die aktuellsten Treiber, Bibliotheken und relevanten Pakete. Für eine weitere up-to-date Version empfehlen wir Ihnen, auf eines der unterstützten Frameworks zu aktualisieren, das das neueste DLAMI Ihrer Wahl verwendet. Eine Liste der verfügbaren DLAMIs finden Sie in den Versionshinweisen für DLAMI.

Unterstützen DLAMIs frühere Versionen von? TensorFlow

Nein. Wir Support die neueste Patch-Version der neuesten Hauptversion jedes Frameworks, die 365 Tage nach der ersten GitHub Veröffentlichung veröffentlicht wurde, wie in der Tabelle mit den AWS Deep Learning AMI Framework-Supportrichtlinien angegeben. Weitere Informationen finden Sie unter Was ist, wenn ich eine Version verwende, die nicht in der Tabelle Unterstützte Frameworks aufgeführt ist?

Wie finde ich das neueste gepatchte Image für eine unterstützte Framework-Version?

Um ein DLAMI mit der neuesten Framework-Version zu verwenden, rufen Sie die DLAMI-ID ab und verwenden Sie sie, um das DLAMI über die EC2-Konsole zu starten. Beispiele für AWS CLI-Befehle zum Abrufen der AWS Deep Learning AMI ID finden Sie im Abschnitt Deep Learning Frameworks im AWS Deep Learning AMIKatalog. AWS CLI-AMI-ID-Abfragen sind auch in den DLAMI-Versionshinweisen für ein einzelnes Framework enthalten. Die von Ihnen gewählte Framework-Version muss in der Tabelle AWS Deep Learning AMI Framework-Support-Richtlinien als Unterstützt gekennzeichnet sein.

Wie oft werden neue Images veröffentlicht?

Die Bereitstellung aktualisierter Patch-Versionen hat für uns höchste Priorität. Wir erstellen routinemäßig zum frühestmöglichen Zeitpunkt gepatchte Images. Wir suchen nach neu gepatchten Framework-Versionen (z. B. TensorFlow 2.9 bis TensorFlow 2.9.1) und neue Nebenversionen (z. B. TensorFlow 2.9 bis TensorFlow 2.10) und stellen sie so bald wie möglich zur Verfügung. Wenn eine vorhandene Version von mit einer neuen Version von CUDA veröffentlicht TensorFlow wird, veröffentlichen wir ein neues DLAMI für diese Version von TensorFlow mit Unterstützung für die neue CUDA-Version.

Wird meine Instance an Ort und Stelle gepatcht, während mein Workload läuft?

Nein. Patch-Updates für DLAMI sind keine „direkten“ Updates.

Sie müssen eine neue EC2-Instance einschalten, Ihre Workloads und Skripts migrieren und dann Ihre vorherige Instanz ausschalten.

Was passiert, wenn eine neue gepatchte oder aktualisierte Framework-Version verfügbar ist?

Suchen Sie regelmäßig auf der Seite mit den Versionshinweisen nach Ihrem Image. Wir empfehlen Ihnen, auf neue gepatchte oder aktualisierte Frameworks zu aktualisieren, sobald diese verfügbar sind. Eine Liste der verfügbaren DLAMIs finden Sie in den Versionshinweisen für DLAMI.

Werden Abhängigkeiten aktualisiert, ohne die Framework-Version zu ändern?

Wir aktualisieren Abhängigkeiten, ohne die Framework-Version zu ändern. Wenn ein Abhängigkeitsupdate jedoch zu einer Inkompatibilität führt, erstellen wir ein Image mit einer anderen Version. Aktuelle Informationen zu Abhängigkeiten finden Sie in den Versionshinweisen für DLAMI.

Wann endet der aktive Support für meine Framework-Version?

DLAMI-Bilder sind unveränderlich. Sobald sie erstellt wurden, ändern sie sich nicht. Es gibt vier Hauptgründe, warum der aktive Support für eine Framework-Version endet:

- Upgrades der Framework-Version (Patch)

- AWSSicherheitspatches
- Ende des Patch-Datums (Alterung nicht mehr verfügbar)
- Abhängigkeit end-of-support

> **Note**
>
> Aufgrund der Häufigkeit von Versionspatch-Upgrades und Sicherheitspatches empfehlen wir, die Seite mit den Versionshinweisen für Ihr DLAMI häufig zu überprüfen und ein Upgrade durchzuführen, wenn Änderungen vorgenommen werden.

Upgrades der Framework-Version (Patch)

Wenn Sie einen DLAMI-Workload haben, der auf TensorFlow 2.7.0 basiert und Version 2.7.1 TensorFlow veröffentlicht GitHub, dann AWS veröffentlicht Sie ein neues DLAMI mit 2.7.1. TensorFlow Die vorherigen Images mit 2.7.0 werden nicht mehr aktiv gepflegt, sobald das neue Image mit 2.7.1 veröffentlicht wird. TensorFlow Das DLAMI mit TensorFlow 2.7.0 erhält keine weiteren Patches. Die Seite mit den DLAMI-Versionshinweisen für TensorFlow 2.7 wird dann mit den neuesten Informationen aktualisiert. Es gibt keine eigene Seite mit Versionshinweisen für jeden kleineren Patch.

Neue DLAMIs, die aufgrund von Patch-Upgrades erstellt wurden, werden mit einer neuen AMI-ID gekennzeichnet.

AWSSicherheitspatches

Wenn Sie einen Workload haben, der auf einem Image mit TensorFlow 2.7.0 basiert und AWS Sie einen Sicherheitspatch erstellen, wird eine neue Version des DLAMI für 2.7.0 veröffentlicht. TensorFlow Die vorherige Version der Images mit TensorFlow 2.7.0 wird nicht mehr aktiv gepflegt. Weitere Informationen finden Sie unter Schritte Wird meine Instance an Ort und Stelle gepatcht, während mein Workload läuft? zur Suche nach der neuesten DLAMI-Version finden Sie unter Wie finde ich das neueste gepatchte Image für eine unterstützte Framework-Version?

Neue DLAMIs, die aufgrund von Patch-Upgrades erstellt wurden, werden mit einer neuen AMI-ID gekennzeichnet.

Ende des Patch-Datums (Alterung nicht mehr verfügbar)

DLAMIs haben das Ende des Patches 365 Tage nach dem GitHub Veröffentlichungsdatum erreicht.

Für Multi-Framework-DLAMIs ist bei der Aktualisierung einer der Framework-Versionen ein neues DLAMI mit der aktualisierten Version erforderlich. Das DLAMI mit der alten Framework-Version wird nicht mehr aktiv gepflegt.

> **Important**
>
> Wir machen eine Ausnahme, wenn es ein größeres Framework-Update gibt. Wenn beispielsweise TensorFlow 1.15 auf TensorFlow 2.0 aktualisiert wird, unterstützen wir weiterhin die neueste Version von TensorFlow 1.15 für einen Zeitraum von zwei Jahren ab dem Datum der GitHub Veröffentlichung oder sechs Monate, nachdem das Origin-Framework-Wartungsteam den Support eingestellt hat, je nachdem, welches Datum früher liegt.

Abhängigkeit end-of-support

Wenn Sie einen Workload auf einem TensorFlow 2.7.0 DLAMI-Image mit Python 3.6 ausführen und diese Version von Python für markiert ist end-of-support, werden alle auf Python 3.6 basierenden DLAMI-Images nicht mehr aktiv verwaltet. Ebenso werden alle DLAMI-Images, die von Ubuntu 16.04 abhängig sind end-of-support, nicht mehr aktiv verwaltet, wenn eine Betriebssystemversion wie Ubuntu 16.04 markiert ist.

Werden Images mit Framework-Versionen, die nicht mehr aktiv verwaltet werden, gepatcht?

Nein. Für Bilder, die nicht mehr aktiv gepflegt werden, wird es keine neuen Versionen geben.

Wie verwende ich eine ältere Framework-Version?

Um ein DLAMI mit einer älteren Framework-Version zu verwenden, rufen Sie die DLAMI-ID ab und verwenden Sie sie, um das DLAMI über die EC2-Konsole zu starten. AWSCLI-Befehle zum Abrufen der AMI-ID finden Sie im Abschnitt Deep Learning Frameworks im AWSDeep Learning AMI-Katalog. AWS CLI-AMI-ID-Abfragen sind auch in den DLAMI-Versionshinweisen für ein einzelnes Framework enthalten.

Wie bleibe ich up-to-date bei Support-Änderungen an Frameworks und deren Versionen?

Bleiben Sie up-to-date bei den DLAMI-Frameworks und -Versionen, indem Sie die Tabelle mit den AWS Deep Learning AMI Framework Support Policys und die DLAMI-Versionshinweise verwenden.

Benötige ich eine kommerzielle Lizenz, um das Anaconda Repository nutzen zu können?

Anaconda hat für bestimmte Benutzer auf ein kommerzielles Lizenzmodell umgestellt. Aktiv gepflegte DLAMIs wurden vom Anaconda-Kanal auf die öffentlich verfügbare Open-Source-Version von Conda (Conda-Forge) migriert.

Einen DLAMI starten und konfigurieren

Wenn Sie hier sind, sollten Sie bereits wissen, welches AMI Sie starten möchten. Wenn nicht, suchen Sie im AWS Deep Learning AMIKatalog nach einem DLAMI und der zugehörigen Hardware, Frameworks und IDs oder sehen Sie sich die aktuellen und historischen DLAMI-Versionshinweise unter anVersionshinweise für DLAMI.

Sie sollten auch wissen, welchen Instance-Typ und welche Region Sie wählen werden. Wenn dies nicht der Fall ist, durchsuchen Sie Auswahl des Instance-Typs für DLAMI.

> **Note**
>
> In den Beispielen verwenden wir p3.16xlarge als Standard-Instanztyp. Ersetzen Sie dies einfach durch den von Ihnen gewünschten Instance-Typ.

> **Important**
>
> Wenn Sie Elastic Inference verwenden möchten, verfügen Sie über das Elastic Inference Inference-Setup, das vor dem Start Ihres DLAMI abgeschlossen sein muss.

Themen

Schritt 1: Starten eines DLAMI

> **Note**
>
> Für diese Komplettlösung könnten wir spezifische Verweise auf das Deep Learning-AMI (Ubuntu 18.04) ab Version 18.04). Auch wenn Sie ein anderes DLAMI wählen, sollten Sie in der Lage sein, dieser Anleitung zu folgen.

1. Finden Sie die ID Ihres DLAMI
2. Starten Sie eine Amazon EC2 EC2-Instance von Ihrem DLAMI

   Sie werden die Amazon EC2 EC2-Konsole verwenden. Folgen Sie den Anweisungen unterStarten von Amazon EC2 Console

   > **Tip**
   >
   > CLI-Option: Wenn Sie sich dafür entscheiden, einen DLAMI über dieAWS CLI einzurichten, benötigen Sie die ID des AMI, die Region und den Instance-Typ sowie Ihre Sicherheitstoken-Informationen. Stellen Sie sicher, dass Ihre AMI- und Instance-IDs bereit sind. Wenn Sie dieAWS CLI noch nicht eingerichtet haben, tun Sie dies zunächst anhand der Anleitung zur Installation derAWS Befehlszeilenschnittstelle.

3. Nachdem Sie die Schritte von einer dieser Optionen ausgeführt haben, warten Sie, bis die Instance bereit ist. Dieser Vorgang dauert einige Minuten. Sie können den Status der Instance in der EC2-Konsole überprüfen.

Rufen Sie die DLAMI-ID ab

Jedes AMI besitzt eine eindeutige Kennung (ID). Sie können die ID für den DLAMI Ihrer Wahl mit derAWS Befehlszeilenschnittstelle (AWSCLI) abfragen. Wenn Sie das noch nichtAWS CLI installiert haben, finden Sie weitere Informationen unter Erste Schritte mit derAWS CLI.

> **Note**
>
> Erinnerung: Im AWS Deep Learning AMIKatalog finden Sie alle DLAMIs und die zugehörigen Prozessor/Beschleuniger, Betriebssysteme, Rechenarchitektur, empfohlene Amazon

EC2 EC2-Instance-Familien, den Support-Status und die Abfragen zum Abrufen von IDs. Weitere Informationen (Treiber, Python-Versionen, Amazon EBS-Typ) finden Sie auch inVersionshinweise für DLAMI den DLAMI-Versionshinweisen.

1. Stellen Sie sicher, dass IhreAWS Anmeldeinformationen konfiguriert sind.

```
aws configure
```

2. Verwenden Sie den folgenden Befehl, um die ID Ihres DLAMI abzurufen, oder suchen Sie nach der imAWS Deep Learning AMI Katalog bereitgestellten Abfrage.

```
aws ec2 describe-images --region us-east-1 --owners amazon \
--filters 'Name=name,Values=Deep Learning AMI (Ubuntu 18.04) Version ??.?'
 'Name=state,Values=available' \
--query 'reverse(sort_by(Images, &CreationDate))[:1].ImageId' --output text
```

> **Note**
>
> Sie können eine Release-Version für ein bestimmtes Framework angeben oder die neueste Version abrufen, indem Sie die Versionsnummer durch ein Fragezeichen ersetzen.

3. Die Ausgabe sollte folgendermaßen oder ähnlich aussehen:

```
ami-094c089c38ed069f2
```

Kopieren Sie diese DLAMI-ID und drücken Sie,q um die Eingabeaufforderung zu verlassen.

Nächster Schritt

Starten von Amazon EC2 Console

Starten von Amazon EC2 Console

> **Note**
>
> Gehen Sie wie folgt vor, um eine Instance mit Elastic Fabric Adapter (EFA) zu

1. Öffnen Sie die EC2-Konsole.
2. Notieren Sie Ihre aktuelle Region in der obersten Navigation. Wenn dies nicht Ihren Wünschen entsprichtAWS-Region, ändern Sie diese Option, bevor Sie fortfahren. Weitere Informationen finden Sie unter EC2-Regionen.
3. Wählen Sie Launch Instance aus.
4. Geben Sie einen Namen für Ihre Instance ein und wählen Sie das für Sie passende DLAMI aus.

   a. Suchen Sie unter Meine AMIs nach einem vorhandenen DLAMI oder wählen Sie Quick Start.

   b. Suche nach DLAMI ID. Durchsuchen Sie die Optionen und wählen Sie dann Ihre Wahl aus.
5. Wählen Sie einen Instance-Type. Die empfohlenen Instanzfamilien für Ihren DLAMI finden Sie imAWS Deep Learning AMI Katalog. Allgemeine Empfehlungen zu DLAMI-Instanztypen finden Sie unter Instanzauswahl.

   > **Note**
   >
   > Wenn Sie Elastic Inference (EI) verwenden möchten, klicken Sie auf Instanzdetails konfigurieren, wählen Sie Amazon EI-Beschleuniger hinzufügen und wählen Sie dann die Größe des Amazon EI-Beschleunigers aus.
6. Wählen Sie Launch Instance aus.

> **Tip**
>
> Eine Anleitung mit Screenshots finden Sie unter Erste Schritte mitAWS Deep Learning mithilfe des Deep Learning-AMI.

Nächster Schritt

Schritt 2: Herstellen der Verbindung zum DLAMI

Schritt 2: Herstellen der Verbindung zum DLAMI

Stellen Sie eine Connect zu dem DLAMI her, das Sie von einem Client aus gestartet haben (Windows, macOS oder Linux). Weitere Informationen finden Sie unter Connect Linux-Instances im Amazon EC2 EC2-Benutzerhandbuch für Linux-Instances.

Halten Sie eine Kopie des SSH-Login-Befehls griffbereit, wenn Sie das Jupyter-Setup nach dem Einloggen durchführen möchten. Sie werden eine Variante davon verwenden, um sich mit der Jupyter-Webseite zu verbinden.

Nächster Schritt

Schritt 3: Testen des DLAMI

Schritt 3: Testen des DLAMI

Abhängig von Ihrer DLAMI-Version haben Sie unterschiedliche Testmöglichkeiten:

- -Deep-Learning-AMI— gehe zuVerwenden des Deep Learning-AMI mit Conda.
- Deep Learning Learning Learning Learning Learning Learning Learning Learning— lesen Sie die Installationsdokumentation Ihres gewünschten Frameworks.

Sie können auch ein Jupyter-Notebook erstellen, Tutorials ausprobieren oder in Python programmieren. Weitere Informationen finden Sie unter Einrichten eines Jupyter-Notebook-Servers.

Schritt 4: Verwaltung Ihrer DLAMI-Instance

Halten Sie Ihr Betriebssystem und sonstige installierte Software immer auf dem neuesten Stand, indem Sie Patches und Updates anwenden, sobald diese verfügbar werden.

Wenn Sie Amazon Linux oder Ubuntu verwenden, werden Sie bei der Anmeldung bei Ihrem DLAMI benachrichtigt, wenn Updates verfügbar sind, und es werden Anweisungen für die Aktualisierung angezeigt. Weitere Informationen zur Wartung von Amazon Linux finden Sie unter Instanzsoftware aktualisieren. Für Ubuntu-Instances lesen Sie in der offiziellen Ubuntu-Dokumentation nach.

Unter Windows überprüfen Sie Windows Update regelmäßig auf Software- und Sicherheits-Updates. Wenn Sie möchten, können neue Updates auch automatisch angewendet.

> **Important**
>
> Informationen zu den Sicherheitslücken Meltdown und Spectre und dazu, wie Sie Ihr Betriebssystem patchen können, um diese zu beheben, finden Sie im Security BulletinAWS -2018-013.

Bereinigen

Wenn Sie das DLAMI nicht mehr benötigen, können Sie es beenden oder kündigen, um weitere Gebühren zu vermeiden. Durch das Stoppen einer Instance wird diese beibehalten, sodass Sie sie später weiter verwenden können. Ihre Konfigurationen, Dateien und andere nichtflüchtige Informationen werden in einem Volume auf Amazon S3 gespeichert. Sie werden mit der kleinen S3-Gebühr belastet, um das Volumen beizubehalten, während die Instance gestoppt ist, aber Sie werden nicht mehr für die Rechenressourcen belastet, während sie sich im gestoppten Zustand befindet. Wenn Sie die Instance erneut starten, wird das Volume geladen und Ihre Daten werden dort gespeichert. Wenn Sie eine Instance beenden, ist sie weg und Sie können sie nicht erneut starten. Ihre Daten befinden sich tatsächlich immer noch in S3. Um weitere Gebühren zu vermeiden, müssen Sie das Volume ebenfalls löschen. Weitere Anweisungen finden Sie unter Beenden der Instance im Amazon EC2 EC2-Benutzerhandbuch für Linux-Instances.

Einrichten eines Jupyter-Notebook-Servers

Mit einem Jupyter-Notebookserver können Sie Jupyter-Notebooks von Ihrer DLAMI-Instanz aus erstellen und ausführen. Mit Jupyter-Notebooks können Sie Experimente zum maschinellen Lernen (ML) für Training und Inferenz durchführen, während Sie dieAWS Infrastruktur nutzen und auf im DLAMI integrierte Pakete zugreifen. Weitere Informationen zu Jupyter-Notebooks finden Sie in der Jupyter Notebook-Dokumentation.

Um einen Jupyter-Notebook-Server einzurichten, müssen Sie:

- Konfigurieren Sie den Jupyter-Notebook-Server auf Ihrer Amazon EC2 EC2-DLAMI-Instance.
- Konfigurieren Sie Ihren Client so, dass Sie sich mit dem Jupyter-Notebook-Server verbinden können. Wir stellen Konfigurationsanweisungen für Windows-, MacOS- und Linux-Clients zur Verfügung.
- Testen Sie die Einrichtung, indem Sie sich am Jupyter-Notebook-Server anmelden.

Folgen Sie den Anweisungen in den folgenden Themen, um die Schritte zum Einrichten eines Jupyter abzuschließen. Nachdem Sie einen Jupyter-Notebookserver eingerichtet haben, finden Sie unter InformationenAusführen von Jupyter-Notebook-Tutorials zum Ausführen der Beispiel-Notebooks, die im DLAMI geliefert werden.

Themen

- Sichern Ihres Jupyter-Servers

- Starten des Jupyter-Notebook-Servers
- Konfigurieren des Clients für die Verbindung mit dem Jupyter-Server
- Testen des Jupyter-Notebook-Servers mit einer Anmeldung

Sichern Ihres Jupyter-Servers

Hier richten wir Jupyter mit SSL und einem benutzerdefinierten Passwort ein.

Connect der Amazon EC2 EC2-Instance her, und führen Sie dann das folgende Verfahren aus.

Konfigurieren des Jupyter-Servers

1. Jupyter bietet ein Passwort-Dienstprogramm. Führen Sie den folgenden Befehl aus und geben Sie Ihr bevorzugtes Passwort ein, wenn Sie dazu aufgefordert werden.

   ```
   $ jupyter notebook password
   ```

   Das Ergebnis sieht etwa folgendermaßen aus:

   ```
   Enter password:
   Verify password:
   [NotebookPasswordApp] Wrote hashed password to /home/ubuntu/.jupyter/
   jupyter_notebook_config.json
   ```

2. Erstellen Sie ein selbstsigniertes SSL-Zertifikat Folgen Sie den Anweisungen, um Ihren Ortsnamen entsprechend einzutragen. Geben Sie `.` ein, wenn Sie eine Eingabeaufforderung leer lassen möchten. Ihre Antworten wirken sich nicht auf die Funktionalität des Zertifikats aus.

   ```
   $ cd ~
   $ mkdir ssl
   $ cd ssl
   $ openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout mykey.key -out
    mycert.pem
   ```

> **Note**
>
> Sie können ein reguläres SSL-Zertifikat erstellen, das Drittanbieter-signiert ist und nicht dazu führt, dass der Browser eine Sicherheitswarnung ausgibt. Dieser Prozess ist wesentlich aufwendiger. Besuchen Sie die Jupyter-Dokumentation für weitere Informationen.

Nächster Schritt

Starten des Jupyter-Notebook-Servers

Starten des Jupyter-Notebook-Servers

Nun können Sie den Jupyter-Server starten, indem Sie sich bei der Instance anmelden und den folgenden Befehl ausführen, der das SSL-Zertifikat verwendet, das Sie im vorherigen Schritt erstellt haben.

```
$ jupyter notebook --certfile=~/ssl/mycert.pem --keyfile ~/ssl/mykey.key
```

Wenn der Server gestartet ist, können Sie sich von Ihrem Client-Computer aus über einen SSH-Tunnel mit ihm verbinden. Wenn der Server ausgeführt wird, sehen Sie eine Ausgabe von Jupyter. Diese bestätigt, dass der Server ausgeführt wird. Ignorieren Sie an dieser Stelle den Hinweis, dass Sie über eine Localhost-URL auf den Server zugreifen können. Dies funktioniert erst nach Erstellung des Tunnel.

> **Note**
>
> Jupyter übernimmt das Wechseln von Umgebungen, wenn Sie die Frameworks mithilfe der Jupyter-Web-Schnittstelle wechseln. Weitere Informationen zu diesem Thema finden Sie im Wechseln von Umgebungen mit Jupyter.

Nächster Schritt

Konfigurieren des Clients für die Verbindung mit dem Jupyter-Server

Konfigurieren des Clients für die Verbindung mit dem Jupyter-Server

Nachdem Sie Ihren Client so konfiguriert haben, dass er eine Verbindung zum Jupyter-Notebook-Server herstellt, können Sie Notebooks auf dem Server in Ihrem Arbeitsbereich erstellen und darauf zugreifen und Ihren Deep-Learning-Code auf dem Server ausführen.

Informationen zur Konfiguration finden Sie unter den folgenden Links.

Themen

- Konfigurieren eines Windows-Clients
- Konfigurieren eines Linux- oder macOS-Client

Konfigurieren eines Windows-Clients

Vorbereitung

Die folgenden Informationen benötigen Sie, um den SSH-Tunnel einzurichten:

- Der öffentliche DNS-Name Ihrer Amazon-EC2-Instance. Den öffentlichen DNS-Namen finden Sie in der EC2-Konsole.
- Das Schlüsselpaar für die private Schlüsseldatei. Weitere Informationen zum Zugriff auf Ihr Schlüsselpaar finden Sie unter Amazon EC2-Schlüsselpaare im Amazon EC2-Benutzerhandbuch für Linux-Instances.

Verwenden von Jupyter-Notebooks von einem Windows-Client

Informationen zu Ihrer Amazon-EC2-Instance von einem Windows-Client finden Sie in diesen Anleitungen.

1. Problembehandlung beim Herstellen einer Verbindung zu Ihrer Instance
2. Herstellung einer Verbindung zu Ihrer Linux-Instance von Windows mit PuTTY

Um einen Tunnel für einen laufenden Jupyter-Server zu erstellen, wird empfohlen, Git Bash auf Ihrem Windows-Client zu installieren und anschließend den Linux/macOS-Client-Anweisungen zu folgen. Sie können jedoch auch einen anderen Ansatz für das Öffnen eines SSH-Tunnels mit Port-Zuweisung wählen. Weitere Informationen finden Sie in der Jupyter-Dokumentation.

Nächster Schritt

Konfigurieren eines Linux- oder macOS-Client

Konfigurieren eines Linux- oder macOS-Client

1. Öffnen Sie ein -Terminalfenster.
2. Führen Sie den folgenden Befehl aus, um alle Anfragen auf dem lokalen Port 8888 an Port 8888 auf Ihrer Remote-Amazon-EC2-Instance weiterzuleiten. Aktualisieren Sie den Befehl, indem Sie den Speicherort Ihres Schlüssels für den Zugriff auf die Amazon EC2 EC2-Instance und den öffentlichen DNS-Namen Ihrer Amazon EC2 EC2-Instance ersetzen. Hinweis: Bei einem Amazon Linux-AMI lautet der Benutzername `ec2-user` anstelle von `ubuntu`.

   ```
   $ ssh -i ~/mykeypair.pem -N -f -L 8888:localhost:8888 ubuntu@ec2-###-##-##-
   ###.compute-1.amazonaws.com
   ```

   Dieser Befehl öffnet einen Tunnel zwischen Ihrem Client und der Amazon-EC2-Remote-Instance, auf der der Jupyter-Notebook-Server ausgeführt wird.

Nächster Schritt

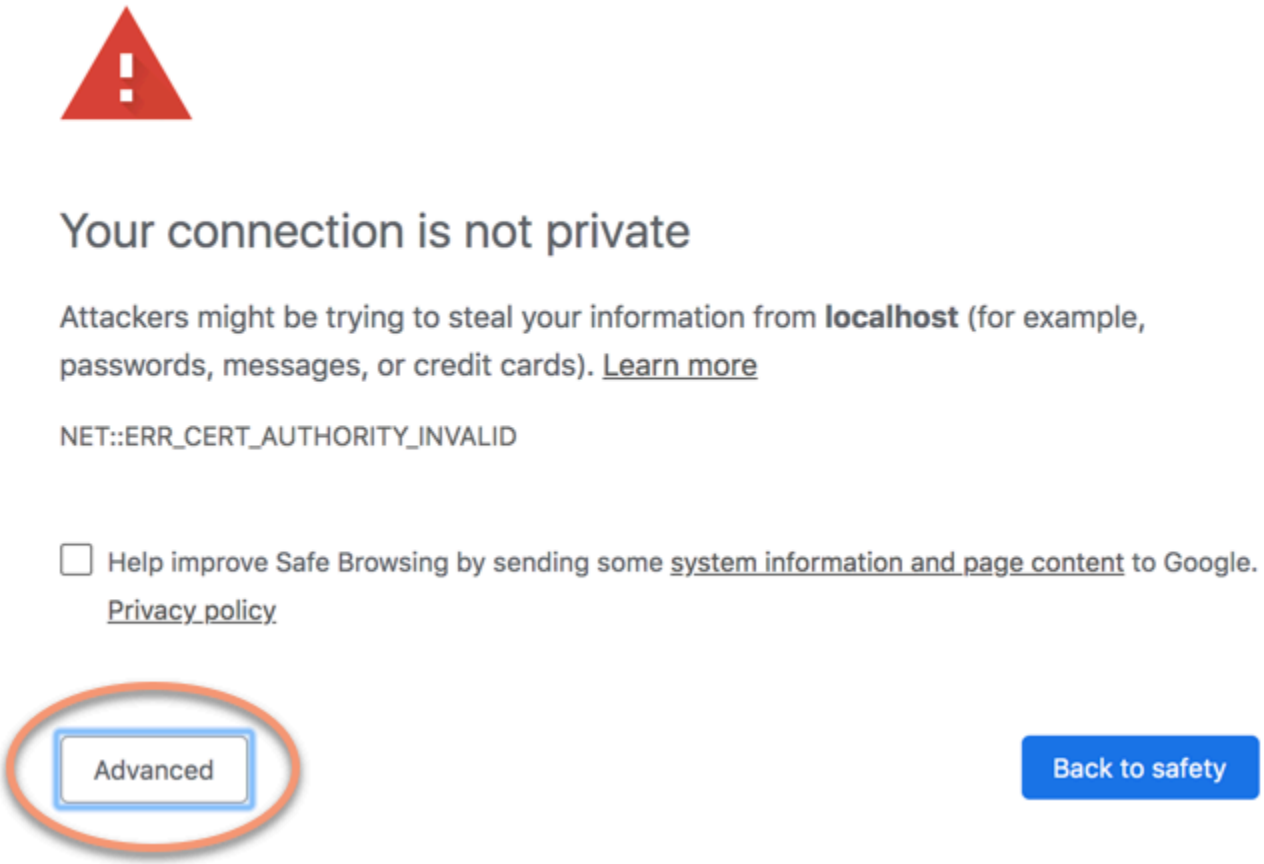
Testen des Jupyter-Notebook-Servers mit einer Anmeldung

Testen des Jupyter-Notebook-Servers mit einer Anmeldung

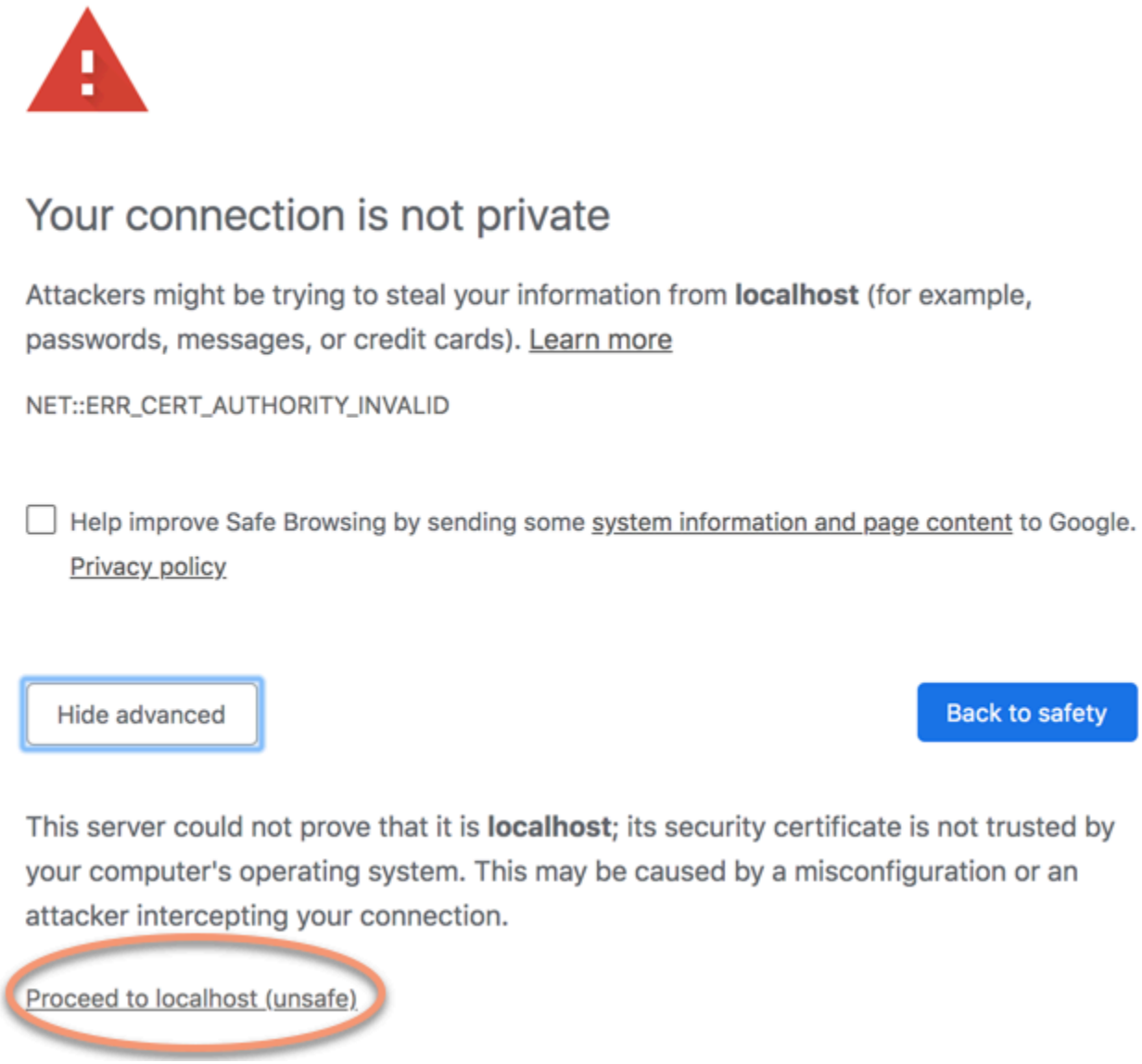
Sie können sich jetzt am Jupyter Notebook-Server anmelden.

Ihr nächster Schritt ist das Testen der Verbindung zum Server über Ihren Browser.

1. Geben Sie in der Adressleiste Ihres Browsers die folgende URL ein, oder klicken Sie auf diesen Link: https://localhost:8888.
2. Mit einem selbstsignierten SSL-Zertifikat warnt Sie Ihr Browser und fordert Sie auf, den Besuch der Website abzubrechen.



Da Sie dies selbst eingerichtet haben, können Sie bedenkenlos fortfahren. Abhängig von Ihrem Browser wird die Schaltfläche "Erweitert", "Details anzeigen" oder eine ähnliche angezeigt.



Klicken Sie darauf und anschließend auf den Link "Weiter zu localhost". Wenn die Verbindung erfolgreich ist, sehen Sie die Webseite des Jupyter-Notebook-Servers. Zu diesem Zeitpunkt werden Sie aufgefordert, das von Ihnen zuvor eingerichtete Passwort einzugeben.

Jetzt haben Sie Zugriff auf den Jupyter-Notebook-Server, der auf dem DLAMI läuft. Sie können nun neue Notebooks erstellen oder die bereitgestellten Tutorials ausführen.

Verwendung eines DLAMI

Themen

In den folgenden Abschnitten wird beschrieben, wie das Deep Learning-AMI mit Conda verwendet werden kann, um Umgebungen zu wechseln, Beispielcode aus jedem der Frameworks auszuführen und Jupyter auszuführen, sodass Sie verschiedene Notebook-Tutorials ausprobieren können.

Verwenden des Deep Learning-AMI mit Conda

Themen

Einführung in das Deep Learning AMI mit Conda

Conda ist ein Open-Source-Paket- und Umgebungsverwaltungssystem, das unter Windows, MacOS und Linux ausgeführt werden kann. Conda installiert, startet und aktualisiert Pakete und deren Abhängigkeiten. Conda erstellt, speichert, lädt und wechselt Umgebungen auf Ihrem lokalen Computer.

Das Deep Learning AMI mit Conda wurde so konfiguriert, dass Sie einfach zwischen Deep-Learning-Umgebungen wechseln können. Die folgenden Anweisungen zeigen Ihnen einige grundlegende Befehle in `conda`. Sie unterstützen Sie bei der Überprüfung der korrekten Funktionsweise des

grundlegenden Imports des Frameworks und der Ausführung einiger einfacher Operationen. Sie können dann zu ausführlicheren Tutorials übergehen, die mit dem DLAMI bereitgestellt werden, oder zu den Frameworks-Beispielen, die Sie auf der Projektseite der einzelnen Frameworks finden.

Loggen Sie sich in Ihr DLAMI ein

Nachdem Sie sich an Ihrem Server angemeldet haben, sehen Sie eine Server-MOTD (Message Of The Day) mit verschiedenen Conda-Befehlen, mit denen Sie zwischen den verschiedenen Deep-Learning-Frameworks wechseln können. Unten folgt eine Beispiel-MOTD. Ihr spezifisches MOTD kann variieren, wenn neue Versionen des DLAMI veröffentlicht werden.

> **Note**
>
> Die Umgebungen CNTK, Caffe, Caffe2, Theano, Chainer und Keras Conda sind ab Version 28 nicht mehr enthalten. AWS Deep Learning AMI Frühere Versionen von, die AWS Deep Learning AMI diese Umgebungen enthalten, werden weiterhin verfügbar sein. Wir werden jedoch nur Updates für diese Umgebungen bereitstellen, wenn von der Open-Source-Community Sicherheitskorrekturen für diese Frameworks veröffentlicht werden.

```
=============================================================================
      __|  __|_  )
      _|  (     /   Deep Learning AMI (Ubuntu 18.04) Version 40.0
     ___|\___|___|
=============================================================================

Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 5.4.0-1037-aws x86_64v)

Please use one of the following commands to start the required environment with the
 framework of your choice:
for AWS MX 1.7 (+Keras2) with Python3 (CUDA 10.1 and Intel MKL-DNN)
 _________________________________ source activate mxnet_p36
for AWS MX 1.8 (+Keras2) with Python3 (CUDA + and Intel MKL-DNN)
 ______________________________ source activate mxnet_latest_p37
for AWS MX(+AWS Neuron) with Python3
 ____________________________________________________ source activate
 aws_neuron_mxnet_p36
for AWS MX(+Amazon Elastic Inference) with Python3
 ________________________________________ source activate amazonei_mxnet_p36
for TensorFlow(+Keras2) with Python3 (CUDA + and Intel MKL-DNN)
 _____________________________ source activate tensorflow_p37
```

```
for Tensorflow(+AWS Neuron) with Python3 ____________________________________________
 source activate aws_neuron_tensorflow_p36
for TensorFlow 2(+Keras2) with Python3 (CUDA 10.1 and Intel MKL-DNN)
 _________________________ source activate tensorflow2_p36
for TensorFlow 2.3 with Python3.7 (CUDA + and Intel MKL-DNN) ____________________________
 source activate tensorflow2_latest_p37
for PyTorch 1.4 with Python3 (CUDA 10.1 and Intel MKL)
 ____________________________________________ source activate pytorch_p36
for PyTorch 1.7.1 with Python3.7 (CUDA 11.0 and Intel MKL)
 ___________________________________ source activate pytorch_latest_p37
for PyTorch (+AWS Neuron) with Python3 ____________________________________________________
 source activate aws_neuron_pytorch_p36
for base Python3 (CUDA 10.0)
 ____________________________________________________________________________ source
 activate python3
```

Ein Conda-Befehl hat das folgende Muster:

`source activate framework_python-version`

Möglicherweise sehen Sie, was bedeutet`for MXNet(+Keras1) with Python3 (CUDA 10.1) _____ source activate mxnet_p36`, dass die Umgebung MXNet, Keras 1, Python 3 und CUDA 10.1 enthält. Und um diese Umgebung zu aktivieren, lautet der Befehl:

```
$ source activate mxnet_p36
```

Starten Sie die Umgebung TensorFlow

> **Note**
>
> Wenn Sie Ihre erste Conda-Umgebung starten, haben Sie bitte etwas Geduld, während diese geladen wird. Das Deep Learning AMI mit Conda installiert bei der ersten Aktivierung des Frameworks automatisch die optimierteste Version des Frameworks für Ihre EC2-Instance. Es sollten keine weiteren Verzögerungen auftreten.

1. Aktivieren Sie die TensorFlow virtuelle Umgebung für Python 3.

   ```
   $ source activate tensorflow_p37
   ```

2. Starten Sie das iPython-Terminal.

```
(tensorflow_37)$ ipython
```

3. Führen Sie ein schnelles TensorFlow Programm aus.

```
import tensorflow as tf
hello = tf.constant('Hello, TensorFlow!')
sess = tf.Session()
print(sess.run(hello))
```

Es sollte "Hello, Tensorflow!" angezeigt werden.

Nächstes Thema

[Ausführen von Jupyter-Notebook-Tutorials]

Wechseln Sie zur PyTorch Python-3-Umgebung

Wenn Sie sich noch in der IPython-Konsole befinden, verwenden Sie und bereiten Sie sich dann darauf vor`quit()`, die Umgebung zu wechseln.

- Aktivieren Sie die PyTorch virtuelle Umgebung für Python 3.

```
$ source activate pytorch_p36
```

Testen Sie etwas PyTorch Code

Um Ihre Installation zu testen, verwenden Sie Python, um PyTorch Code zu schreiben, der ein Array erstellt und ausgibt.

1. Starten Sie das iPython-Terminal.

```
(pytorch_p36)$ ipython
```

2. Importieren PyTorch.

```
import torch
```

Möglicherweise wird eine Warnmeldung zu einem Paket eines Drittanbieters angezeigt. Sie können sie ignorieren.

3. Erstellen Sie eine 5x3-Matrix mit zufällig initialisierten Elementen. Drucken Sie das Array.

```
x = torch.rand(5, 3)
print(x)
```

Überprüfen Sie das Ergebnis.

```
tensor([[0.3105, 0.5983, 0.5410],
        [0.0234, 0.0934, 0.0371],
        [0.9740, 0.1439, 0.3107],
        [0.6461, 0.9035, 0.5715],
        [0.4401, 0.7990, 0.8913]])
```

Wechseln Sie zur MXNet-Python-3-Umgebung

Wenn Sie sich noch in der IPython-Konsole befinden, verwenden Sie und bereiten Sie sich dann darauf vor`quit()`, die Umgebung zu wechseln.

- Aktivieren Sie die virtuelle MXNet-Umgebung für Python 3.

```
$ source activate mxnet_p36
```

Testen Sie etwas MXNet-Code

Um Ihre Installation zu testen, schreiben Sie mit Python MXNet-Code, der mit der `NDArray`-API ein Array erstellt und anzeigt. Weitere Informationen finden Sie unter NDArray-API.

1. Starten Sie das iPython-Terminal.

```
(mxnet_p36)$ ipython
```

2. MXNet importieren.

```
import mxnet as mx
```

Möglicherweise wird eine Warnmeldung zu einem Paket eines Drittanbieters angezeigt. Sie können sie ignorieren.

3. Erstellen Sie eine 5x5-Matrix (eine Instance von `NDArray`) mit Elementen, die mit 0 initialisiert sind. Drucken Sie das Array.

```
mx.ndarray.zeros((5,5)).asnumpy()
```

Überprüfen Sie das Ergebnis.

```
array([[ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.]], dtype=float32)
```

Weitere Beispiele zu MXNet finden Sie im Abschnitt "MXNet-Tutorials".

Entfernen von Umgebungen

Wenn Ihnen der Speicherplatz auf dem DLAMI ausgeht, können Sie Conda-Pakete, die Sie nicht verwenden, deinstallieren:

```
conda env list
conda env remove --name <env_name>
```

Verwenden des Deep Learning Base AMI

Verwenden des Deep Learning Base AMI

Das Basis-AMI wird mit einer grundlegenden Plattform von GPU-Treibern und Beschleunigungsbibliotheken geliefert, um Ihre eigene, angepasste Deep-Learning-Umgebung zu implementieren. Standardmäßig ist das AMI mit der NVIDIA CUDA 11.0-Umgebung konfiguriert. Sie können auch zwischen verschiedenen Versionen von CUDA wechseln. Weitere Informationen zur Vorgehensweise finden Sie in den folgenden Anweisungen.

CUDA-Versionen konfigurieren

Sie können die CUDA-Version überprüfen, indem Sie das NVIDIA-Programm ausführen. `nvcc`

```
nvcc --version
```

Sie können eine bestimmte CUDA-Version mit dem folgenden Bash-Befehl auswählen und überprüfen:

```
sudo rm /usr/local/cuda
sudo ln -s /usr/local/cuda-11.0 /usr/local/cuda
```

Weitere Informationen finden Sie in den Versionshinweisen zu Base DLAMI.

Ausführen von Jupyter-Notebook-Tutorials

Tutorials und Beispiele sind im Quellcode der Deep-Learning-Projekte enthalten und laufen in den meisten Fällen auf jedem DLAMI. Wenn Sie das -Deep-Learning-AMI auswählen, erhalten Sie zusätzlich einige ausgesuchte Tutorials, die bereits eingerichtet sind und sofort ausprobiert werden können.

> **Important**
>
> Um die auf dem DLAMI installierten Jupyter-Notebook-Tutorials auszuführen, müssen Sie Einrichten eines Jupyter-Notebook-Servers

Sobald der Jupyter-Server ausgeführt wird, können Sie die Tutorials über Ihren Webbrowser aufrufen. Wenn Sie das Deep Learning-AMI mit Conda ausführen oder Python-Umgebungen eingerichtet haben, können Sie Python-Kernel über die Jupyter-Notebook-Oberfläche wechseln. Wählen Sie den entsprechenden Kernel aus, bevor Sie versuchen, ein Tutorial für ein bestimmtes Framework auszuführen. Weitere Beispiele hierfür werden Benutzern des Deep Learning AMI mit Conda zur Verfügung gestellt.

> **Note**
>
> Viele Tutorials erfordern zusätzliche Python-Module, die möglicherweise nicht auf Ihrem DLAMI eingerichtet sind. Wenn Sie eine Fehlermeldung erhalten wie`"xyz module`

`not found"`, melden Sie sich bei der DLAMI an, aktivieren Sie die Umgebung wie oben beschrieben und installieren Sie dann die erforderlichen Module.

> **Tip**
>
> Deep-Learning-Tutorials und Beispiele basieren oft auf einer oder mehreren GPUs. Wenn Ihr Instance-Typ keine GPU hat, müssen Sie möglicherweise Codeabschnitte des Beispiels ändern, um dieses ausführen zu können.

Navigation der installierten Tutorials

Sobald Sie beim Jupyter-Server angemeldet sind und das Verzeichnis der Tutorials sehen können (nur auf Deep Learning AMI mit Conda), werden Ihnen Ordner mit Tutorials für jeden Framework-Namen angezeigt. Wenn ein Framework nicht aufgeführt ist, sind auf Ihrem aktuellen DLAMI keine Tutorials für dieses Framework verfügbar. Klicken Sie auf den Namen des Frameworks, um die aufgeführten Tutorials zu sehen. Wählen Sie eines davon per Klick aus, wenn Sie es starten möchten.

Wenn Sie ein Notebook zum ersten Mal auf dem Deep Learning AMI mit Conda ausführen, möchte es wissen, welche Umgebung Sie verwenden möchten. Es wird eine Liste zur Auswahl bereitgestellt. Jede Umgebung hat einen Namen, der diesem Muster entspricht:

```
Environment (conda_framework_python-version)
```

Sie können beispielsweise `Environment (conda_mxnet_p36)` sehen, was bedeutet, dass die Umgebung MXNet und Python 3 enthält. Die andere Variante davon wäre `Environment (conda_mxnet_p27)`, also eine Umgebung mit MXNet und Python 2.

> **Tip**
>
> Wenn Sie sich Sorgen darüber machen, welche Version von CUDA aktiv ist, können Sie sie unter anderem im MOTD sehen, wenn Sie sich zum ersten Mal beim DLAMI anmelden.

Wechseln von Umgebungen mit Jupyter

Wenn Sie ein Tutorial für ein anderes Framework ausprobieren möchten, überprüfen Sie, welcher Kernel aktuell ausgeführt wird. Diese Information finden Sie oben rechts in der Jupyter-Benutzeroberfläche, direkt unter der Schaltfläche zum Abmelden. Sie können den Kernel in einem beliebigen geöffneten Notebook ändern, indem Sie auf die Jupyter-Menüelemente Kernel, Change Kernel und auf die Umgebung klicken, die für das ausgeführte Notebook geeignet ist.

Nach diesem Schritt müssen Sie alle Zellen erneut ausführen, da eine Änderung des Kernels den Zustand von allen Elementen löscht, die Sie zuvor ausgeführt haben.

> **Tip**
>
> Der Wechsel zwischen Frameworks kann Spaß machen und lehrreich sein. Es kann aber vorkommen, dass Sie nicht mehr über genügend Speicher verfügen. Wenn Fehler auftreten, prüfen Sie das Terminal-Fenster, in dem der Jupyter-Server ausgeführt wird. Hier finden Sie hilfreiche Meldungen und Fehlerprotokolle, und möglicherweise wird ein Fehler angezeigt. out-of-memory Zum Beheben dieses Problems können Sie zur Startseites Ihres Jupyter-Servers wechseln, die Registerkarte Running auswählen und für die einzelnen Tutorials, die wahrscheinlich noch im Hintergrund ausgeführt werden und den gesamten Speicher beanspruchen, auf Shutdown klicken.

Nächstes Thema

Klicken Sie für weitere Beispiele und Beispiel-Code von jedem Framework auf Weiter oder fahren Sie mit dem Thema Apache MXNet (Inkubation) fort.

Tutorials

Im Folgenden finden Sie Tutorials zur Verwendung des Deep Learning AMI mit der Software von Conda.

Themen

- 10-Minuten-Tutorials
- Aktivieren von Frameworks
- Debugging und Visualisierung

- Verteilte Schulungen
- Elastic Fabric Adapter
- GPU-Überwachung und -Optimierung
- Der AWS Inferentia-Chip mit DLAMI
- Das Graviton DLAMI
- Das Habana DLAMI
- Inferenz
- Verwenden von Frameworks mit ONNX
- Modellbereitstellung

10-Minuten-Tutorials

- Starten Sie a AWS Deep Learning AMI (in 10 Minuten)
- Trainieren Sie ein Deep-Learning-Modell mit DLC auf Amazon EC2 (in 10 Minuten)

Aktivieren von Frameworks

Im Folgenden sind die Deep-Learning-Frameworks aufgeführt, die auf dem Deep Learning-AMI mit Conda installiert sind. Klicken Sie auf ein Framework, um zu erfahren, wie Sie es aktivieren.

Themen

- Apache MXNet (Inkubation)
- Caffe2
- Chainer
- CNTK
- Keras
- PyTorch
- TensorFlow
- TensorFlow 2
- TensorFlow mit Horovod
- TensorFlow 2 mit Horovod
- Theano

Apache MXNet (Inkubation)

Apache MXNet aktivieren (Inkubation)

Dieses Tutorial zeigt, wie Sie MXNet auf einer Instanz aktivieren, auf der das Deep Learning AMI mit Conda (DLAMI on Conda) ausgeführt wird, und wie Sie ein MXNet-Programm ausführen.

Wenn ein stabiles Conda-Paket eines Frameworks veröffentlicht wird, wird es getestet und auf dem DLAMI vorinstalliert. Wenn Sie den neuesten, nicht getesteten Nightly Build ausführen möchten, können Sie Installieren des Nightly Builds (Test) von MXNet manuell ausführen.

Um MXNet auf dem DLAMI mit Conda auszuführen

1. Um das Framework zu aktivieren, öffnen Sie eine Amazon Elastic Compute Cloud (Amazon EC2) -Instanz des DLAMI mit Conda.
   - Für MXNet und Keras 2 auf Python 3 mit CUDA 9.0 und MKL-DNN führen Sie diesen Befehl aus:

     ```
     $ source activate mxnet_p36
     ```

   - Für MXNet und Keras 2 auf Python 2 mit CUDA 9.0 und MKL-DNN führen Sie diesen Befehl aus:

     ```
     $ source activate mxnet_p27
     ```

2. Starten Sie das iPython-Terminal.

   ```
   (mxnet_p36)$ ipython
   ```

3. Führen Sie ein kurzes MXNet-Programm aus. Erstellen Sie eine 5x5-Matrix (eine Instance von `NDArray`) mit Elementen, die mit 0 initialisiert sind. Drucken Sie das Array.

   ```
   import mxnet as mx
   mx.ndarray.zeros((5,5)).asnumpy()
   ```

4. Überprüfen Sie das Ergebnis.

   ```
   array([[ 0.,  0.,  0.,  0.,  0.],
          [ 0.,  0.,  0.,  0.,  0.],
          [ 0.,  0.,  0.,  0.,  0.],
          [ 0.,  0.,  0.,  0.,  0.],
   ```

```
       [ 0.,  0.,  0.,  0.,  0.]], dtype=float32)
```

Installieren des Nightly Builds (Test) von MXNet

Sie können den neuesten MXNet-Build in einer oder beiden MXNet Conda-Umgebungen auf Ihrem Deep Learning-AMI mit Conda installieren.

So installieren Sie MXNet aus einem Nightly Build

1.
   - In der Python 3 MXNet-Umgebung führen Sie diesen Befehl aus:

     ```
     $ source activate mxnet_p36
     ```

   - In der Python 2 MXNet-Umgebung führen Sie diesen Befehl aus:

     ```
     $ source activate mxnet_p27
     ```

2. Entfernen Sie die derzeit installierte MXNet:

   > **Note**
   >
   > Für die restlichen Schritte wird davon ausgegangen, dass Sie die `mxnet_p36`-Umgebung verwenden.

   ```
   (mxnet_p36)$ pip uninstall mxnet-cu90mkl
   ```

3. Installieren Sie den neuesten Nightly Build von MXNet.

   ```
   (mxnet_p36)$ pip install --pre mxnet-cu90mkl
   ```

4. Um zu überprüfen, ob Sie den neuesten Nightly Build installiert haben, starten Sie das IPython-Terminal und überprüfen Sie die Version von MXNet.

   ```
   (mxnet_p36)$ ipython
   ```

   ```
   import mxnet
   print (mxnet.__version__)
   ```

Die Ausgabe sollte die neueste stabile Version von MXNet enthalten.

Weitere Tutorials

Weitere Tutorials finden Sie im Ordner Deep Learning AMI with Conda tutorials, der sich im Home-Verzeichnis des DLAMI befindet.

1. Verwenden Sie Apache MXNet (Incubating) für Inferenz mit einem 50-Modell ResNet
2. Verwenden Sie Apache MXNet (Incubating) für Inferenzen mit einem ONNX-Modell
3. Modellserver für Apache MXNet (MMS)

Weitere Tutorials und Beispiele finden Sie in der offiziellen Python-Dokumentation des Frameworks, in der Python-API für MXNet oder auf der Apache MXNet-Website.

Caffe2

> **Note**
>
> Wir schließen die Umgebungen CNTK, Caffe, Caffe2 und Theano Conda bei Version v28 nicht mehr in den AWS Deep Learning AMI -Start ein. Frühere Versionen von AWS Deep Learning AMI , die diese Umgebungen enthalten, werden weiterhin verfügbar sein. Wir werden jedoch nur Updates für diese Umgebungen bereitstellen, wenn von der Open-Source-Community Sicherheitskorrekturen für diese Frameworks veröffentlicht werden.

Caffe2-Tutorial

Um das Framework zu aktivieren, folgen Sie diesen Anweisungen auf Ihrem Deep Learning AMI mit Conda.

Es gibt nur die Option Python 2 mit CUDA 9 mit cuDNN 7:

```
$ source activate caffe2_p27
```

Starten Sie das iPython-Terminal.

```
(caffe2_p27)$ ipython
```

Führen Sie ein kurzes Caffe2-Programm aus.

```
from caffe2.python import workspace, model_helper
import numpy as np
# Create random tensor of three dimensions
x = np.random.rand(4, 3, 2)
print(x)
print(x.shape)
workspace.FeedBlob("my_x", x)
x2 = workspace.FetchBlob("my_x")
print(x2)
```

Die initialen, zufälligen numpy-Arrays sollten angezeigt und in einen Caffe2-Blob geladen werden. Beachten Sie, dass sie nach dem Laden gleich sind.

Weitere Tutorials

Weitere Tutorials und Beispiele finden Sie in den offiziellen Python-Dokumenten des Frameworks, der Python-API für Caffe2 und der Caffe2-Website.

Chainer

> **Note**
>
> AWS Deep Learning AMI Ab Version 28 enthalten wir keine Chainer Conda-Umgebungen mehr. Frühere Versionen von AWS Deep Learning AMI , die diese Umgebungen enthalten, werden weiterhin verfügbar sein. Wir werden jedoch nur Updates für diese Umgebungen bereitstellen, wenn von der Open-Source-Community Sicherheitskorrekturen für diese Frameworks veröffentlicht werden.

Chainer ist ein flexibles, auf Python basierendes Framework für die einfache und intuitive Entwicklung komplexer neuronaler Netzwerkarchitekturen. Chainer macht es einfach, Multi-GPU-Instances für das Training zu verwenden. Darüber hinaus protokolliert Chainer automatisch Ergebnisse, Verluste und Genauigkeit von Graphen und erzeugt Ausgaben für die Darstellung des neuronalen Netzwerks mit einem Berechnungsgraphen. Es ist im Deep Learning AMI mit Conda (DLAMI with Conda) enthalten.

Chainer aktivieren

1. Stellen Sie mit Conda Connect zu der Instance her, auf der Deep Learning AMI ausgeführt wird. Informationen zur Auswahl the section called “Auswahl der Instance” oder Verbindung zu einer Instance finden Sie in der oder in der Amazon EC2 EC2-Dokumentation.
2.
   - Aktivieren der Python 3 Chainer-Umgebung:

     ```
     $ source activate chainer_p36
     ```

   - Aktivieren der Python 2 Chainer-Umgebung:

     ```
     $ source activate chainer_p27
     ```

3. Starten Sie das iPython-Terminal:

   ```
   (chainer_p36)$ ipython
   ```

4. Testen Sie das Importieren von Chainer, um sicherzustellen, dass es einwandfrei funktioniert:

   ```
   import chainer
   ```

   Möglicherweise sehen Sie ein paar Warnmeldungen, aber keinen Fehler.

Weitere Infos

- Testen Sie die Tutorials auf Chainer.
- Der `Chainer`-Beispielordner in der Quelle, die Sie vorher heruntergeladen haben, enthält mehrere Beispiele. Probieren Sie sie aus, um zu sehen, wie sie funktionieren.
- Weitere Informationen über Chainer finden Sie auf der Website mit der Chainer-Dokumentation.

CNTK

> **Note**
>
> Wir schließen die Umgebungen CNTK, Caffe, Caffe2 und Theano Conda bei Version v28 nicht mehr in den AWS Deep Learning AMI -Start ein. Frühere Versionen von AWS Deep Learning AMI , die diese Umgebungen enthalten, werden weiterhin verfügbar sein. Wir

werden jedoch nur Updates für diese Umgebungen bereitstellen, wenn von der Open-Source-Community Sicherheitskorrekturen für diese Frameworks veröffentlicht werden.

Aktivieren von CNTK

Dieses Tutorial zeigt, wie Sie CNTK auf einer Instance aktivieren, auf der das Deep Learning AMI mit Conda (DLAMI on Conda) ausgeführt wird, und wie Sie ein CNTK-Programm ausführen.

Wenn ein stabiles Conda-Paket eines Frameworks veröffentlicht wird, wird es getestet und auf dem DLAMI vorinstalliert. Wenn Sie den neuesten, nicht getesteten Nightly Build ausführen möchten, können Sie Installieren Sie den CNTK Nightly Build (Testversion) manuell ausführen.

Um CNTK auf dem DLAMI mit Conda auszuführen

1. Um CNTK zu aktivieren, öffnen Sie eine Amazon Elastic Compute Cloud (Amazon EC2) -Instanz des DLAMI mit Conda.
   - Für Python 3 mit CUDA 9 mit cuDNN 7:

     ```
     $ source activate cntk_p36
     ```

   - Für Python 2 mit CUDA 9 mit cuDNN 7:

     ```
     $ source activate cntk_p27
     ```

2. Starten Sie das iPython-Terminal.

   ```
   (cntk_p36)$ ipython
   ```

3. - Bei einer CPU-Instance führen Sie dieses schnelle CNTK-Programm aus.

     ```
     import cntk as C
     C.__version__
     c = C.constant(3, shape=(2,3))
     c.asarray()
     ```

     Sie sollten die CNTK Version sehen, dann die Ausgabe eines 2x3 Arrays von 3's.
   - Wenn Sie eine GPU-Instance haben, können Sie sie mit dem folgenden Codebeispiel testen. Ein Ergebnis von `True` ist das, was Sie erwarten würden, wenn CNTK auf die GPU zugreifen könnte.

```
from cntk.device import try_set_default_device, gpu
try_set_default_device(gpu(0))
```

Installieren Sie den CNTK Nightly Build (Testversion)

Sie können den neuesten CNTK-Build in einer oder beiden CNTK Conda-Umgebungen auf Ihrem Deep Learning-AMI mit Conda installieren.

So installieren Sie CNTK aus einem Nightly Build

1. • Für CNTK und Keras 2 auf Python 3 mit CUDA 9.0 und MKL-DNN führen Sie diesen Befehl aus:

   ```
   $ source activate cntk_p36
   ```

   • Für CNTK und Keras 2 auf Python 2 mit CUDA 9.0 und MKL-DNN führen Sie diesen Befehl aus:

   ```
   $ source activate cntk_p27
   ```

2. Für die restlichen Schritte wird davon ausgegangen, dass Sie die `cntk_p36`-Umgebung verwenden. Entfernen Sie die derzeit installierte CNTK:

   ```
   (cntk_p36)$ pip uninstall cntk
   ```

3. Um den CNTK-Nightly Build zu installieren, müssen Sie zunächst die zu installierende Version auf der CNTK Nightly-Website finden.

4. • (Option für GPU-Instances): Zur Installation des Nightly Builds verwenden Sie folgende Variante zur Ersetzung im gewünschten Build:

   ```
   (cntk_p36)$ pip install https://cntk.ai/PythonWheel/GPU/latest-nightly-build
   ```

   Ersetzen Sie die URL im vorherigen Befehl durch die GPU-Version für Ihre aktuelle Python-Umgebung.

   • (Option für CPU-Instances): Zur Installation des Nightly Builds verwenden Sie folgende Variante zur Ersetzung im gewünschten Build:

```
(cntk_p36)$ pip install https://cntk.ai/PythonWheel/CPU-Only/latest-nightly-build
```

Ersetzen Sie die URL im vorherigen Befehl durch die CPU-Version für Ihre aktuelle Python-Umgebung.

5. Um zu überprüfen, ob Sie den neuesten Nightly Build installiert haben, starten Sie das IPython-Terminal und überprüfen Sie die Version von CNTK.

```
(cntk_p36)$ ipython
```

```
import cntk
print (cntk.__version__)
```

Die Druckausgabe sollte in etwa wie folgt aussehen: `2.6-rc0.dev20181015`

Weitere Tutorials

Weitere Tutorials und Beispiele finden Sie in den offiziellen Python-Dokumenten des Frameworks, in der Python-API für CNTK oder auf der CNTK-Website.

Keras

Keras-Tutorial

1. Um das Framework zu aktivieren, verwenden Sie die folgenden Befehle auf Ihrer the section called “Conda DLAMI”-CLI.
   - Für Keras 2 mit MXNet-Backend unter Python 3 mit CUDA 9 mit cuDNN 7:

     ```
     $ source activate mxnet_p36
     ```
   - Für Keras 2 mit MXNet-Backend unter Python 2 mit CUDA 9 mit cuDNN 7:

     ```
     $ source activate mxnet_p27
     ```
   - Für Keras 2 mit einem TensorFlow Backend auf Python 3 mit CUDA 9 mit cuDNN 7:

     ```
     $ source activate tensorflow_p36
     ```

- Für Keras 2 mit einem TensorFlow Backend auf Python 2 mit CUDA 9 mit cuDNN 7:

```
$ source activate tensorflow_p27
```

2. Verwenden Sie die folgenden Befehle, um das Importieren von Keras zu testen und zu überprüfen, welches Backend aktiviert ist:

```
$ ipython
import keras as k
```

Folgendes wird auf dem Bildschirm angezeigt:

```
Using MXNet backend
```

Wenn Keras verwendet TensorFlow, wird Folgendes angezeigt:

```
Using TensorFlow backend
```

> **Note**
>
> Wenn Sie eine Fehlermeldung erhalten oder noch das falsche Backend verwendet wird, können Sie Ihre Keras-Konfiguration manuell aktualisieren. Bearbeiten Sie die `~/.keras/keras.json`-Datei und ändern Sie die Backendeinstellung in `mxnet` oder `tensorflow`.

Weitere Tutorials

- Ein Multi-GPU-Tutorial unter Verwendung von Keras mit einem MXNet-Backend erhalten Sie, wenn Sie Tutorial der Keras-MXNet-Multi-GPU-Schulung testen.
- Beispiele für Keras mit einem MXNet-Backend finden Sie im Verzeichnis Deep Learning AMI with Conda. `~/examples/keras-mxnet`
- Beispiele für Keras mit einem TensorFlow Backend finden Sie im Verzeichnis Deep Learning AMI with `~/examples/keras` Conda.
- Weitere Tutorials und Beispiele finden Sie auf der Keras-Website.

PyTorch

Wird aktiviert PyTorch

Wenn ein stabiles Conda-Paket eines Frameworks veröffentlicht wird, wird es getestet und auf dem DLAMI vorinstalliert. Wenn Sie den neuesten, nicht getesteten Nightly Build ausführen möchten, können Sie PyTorchInstall's Nightly Build (experimentell) manuell ausführen.

Um das aktuell installierte Framework zu aktivieren, folgen Sie diesen Anweisungen auf Ihrem Deep Learning AMI mit Conda.

Führen PyTorch Sie für Python 3 mit CUDA 10 und MKL-DNN diesen Befehl aus:

```
$ source activate pytorch_p36
```

Führen PyTorch Sie für Python 2 mit CUDA 10 und MKL-DNN diesen Befehl aus:

```
$ source activate pytorch_p27
```

Starten Sie das iPython-Terminal.

```
(pytorch_p36)$ ipython
```

Führen Sie ein schnelles Programm aus. PyTorch

```
import torch
x = torch.rand(5, 3)
print(x)
print(x.size())
y = torch.rand(5, 3)
print(torch.add(x, y))
```

Das anfängliche zufällige Array sollte angezeigt werden. Danach wird seine Größe angezeigt und dann ein weiteres zufälliges Array.

PyTorchInstall's Nightly Build (experimentell)

Wie installiert man PyTorch von einem Nightly-Build

Sie können den neuesten PyTorch Build in einer oder beiden PyTorch Conda-Umgebungen auf Ihrem Deep Learning-AMI mit Conda installieren.

1. • (Option für Python 3) — Aktiviere die PyTorch Python-3-Umgebung:

```
$ source activate pytorch_p36
```

   • (Option für Python 2) — Aktiviere die Python PyTorch 2-Umgebung:

```
$ source activate pytorch_p27
```

2. Für die restlichen Schritte wird davon ausgegangen, dass Sie die `pytorch_p36`-Umgebung verwenden. Entfernen Sie das aktuell installierte PyTorch:

```
(pytorch_p36)$ pip uninstall torch
```

3. • (Option für GPU-Instanzen) — Installieren Sie den neuesten Nightly-Build von PyTorch mit CUDA 10.0:

```
(pytorch_p36)$ pip install torch_nightly -f https://download.pytorch.org/whl/
nightly/cu100/torch_nightly.html
```

   • (Option für CPU-Instanzen) — Installieren Sie den neuesten nächtlichen Build von PyTorch für Instanzen ohne GPUs:

```
(pytorch_p36)$ pip install torch_nightly -f https://download.pytorch.org/whl/
nightly/cpu/torch_nightly.html
```

4. Um zu überprüfen, ob Sie den neuesten Nightly-Build erfolgreich installiert haben, starten Sie das IPython-Terminal und überprüfen Sie die Version von. PyTorch

```
(pytorch_p36)$ ipython
```

```
import torch
print (torch.__version__)
```

   Die Druckausgabe sollte in etwa wie folgt aussehen: `1.0.0.dev20180922`

5. Um zu überprüfen, ob der PyTorch Nightly-Build gut mit dem MNIST-Beispiel funktioniert, können Sie ein Testskript aus dem Beispiel-Repository ausführen: PyTorch

```
(pytorch_p36)$ cd ~
(pytorch_p36)$ git clone https://github.com/pytorch/examples.git pytorch_examples
```

```
(pytorch_p36)$ cd pytorch_examples/mnist
(pytorch_p36)$ python main.py || exit 1
```

Weitere Tutorials

Weitere Tutorials finden Sie im Ordner Deep Learning AMI with Conda tutorials im Home-Verzeichnis des DLAMI. Weitere Tutorials und Beispiele finden Sie in den offiziellen Dokumenten, der PyTorch Dokumentation und auf der Website des Frameworks. PyTorch

- PyTorch zu ONNX zu MXNet Tutorial
- PyTorch zum ONNX zum CNTK-Tutorial

TensorFlow

Wird aktiviert TensorFlow

Dieses Tutorial zeigt, wie Sie TensorFlow auf einer Instanz, auf der das Deep Learning AMI mit Conda (DLAMI on Conda) ausgeführt wird, aktivieren und ein Programm ausführen. TensorFlow

Wenn ein stabiles Conda-Paket eines Frameworks veröffentlicht wird, wird es getestet und auf dem DLAMI vorinstalliert. Wenn Sie den neuesten, nicht getesteten Nightly Build ausführen möchten, können Sie TensorFlowInstall's Nightly Build (experimentell) manuell ausführen.

Um mit TensorFlow Conda auf dem DLAMI zu laufen

1. Öffnen Sie zur Aktivierung TensorFlow eine Amazon Elastic Compute Cloud (Amazon EC2) - Instanz des DLAMI mit Conda.
   - Führen Sie für TensorFlow und Keras 2 auf Python 3 mit CUDA 9.0 und MKL-DNN diesen Befehl aus:

     ```
     $ source activate tensorflow_p36
     ```

   - Führen Sie für TensorFlow und Keras 2 auf Python 2 mit CUDA 9.0 und MKL-DNN diesen Befehl aus:

     ```
     $ source activate tensorflow_p27
     ```

2. Starten Sie das iPython-Terminal:

```
(tensorflow_p36)$ ipython
```

3. Führen Sie ein TensorFlow Programm aus, um zu überprüfen, ob es ordnungsgemäß funktioniert:

```
import tensorflow as tf
hello = tf.constant('Hello, TensorFlow!')
sess = tf.Session()
print(sess.run(hello))
```

Auf Ihrem Bildschirm sollte `Hello, TensorFlow!` angezeigt werden.

TensorFlowInstall's Nightly Build (experimentell)

Sie können den neuesten TensorFlow Build in einer oder beiden TensorFlow Conda-Umgebungen auf Ihrem Deep Learning-AMI mit Conda installieren.

Um von einem TensorFlow Nightly-Build aus zu installieren

1. • Führen Sie für die TensorFlow Python-3-Umgebung den folgenden Befehl aus:

```
$ source activate tensorflow_p36
```

   • Führen Sie für die TensorFlow Python-2-Umgebung den folgenden Befehl aus:

```
$ source activate tensorflow_p27
```

2. Entfernen Sie den aktuell installierten TensorFlow.

> **Note**
>
> Für die restlichen Schritte wird davon ausgegangen, dass Sie die `tensorflow_p36`-Umgebung verwenden.

```
(tensorflow_p36)$ pip uninstall tensorflow
```

3. Installieren Sie den neuesten Nightly-Build von. TensorFlow

```
(tensorflow_p36)$ pip install tf-nightly
```

4. Um zu überprüfen, ob Sie den neuesten Nightly-Build erfolgreich installiert haben, starten Sie das IPython-Terminal und überprüfen Sie die Version von. TensorFlow

```
(tensorflow_p36)$ ipython
```

```
import tensorflow
print (tensorflow.__version__)
```

   Die Druckausgabe sollte in etwa wie folgt aussehen: `1.12.0-dev20181012`

Weitere Tutorials

TensorFlow mit Horovod

TensorBoard

TensorFlow Servieren

Tutorials finden Sie in dem Ordner namens `Deep Learning AMI with Conda tutorials` im Home-Verzeichnis des DLAMI.

Weitere Tutorials und Beispiele finden Sie in der TensorFlow Dokumentation zur TensorFlow Python-API oder TensorFlowauf der Website.

TensorFlow 2

Dieses Tutorial zeigt, wie Sie TensorFlow 2 auf einer Instance aktivieren, auf der das Deep Learning AMI mit Conda (DLAMI on Conda) ausgeführt wird, und ein 2-Programm ausführen. TensorFlow

Wenn ein stabiles Conda-Paket eines Frameworks veröffentlicht wird, wird es getestet und auf dem DLAMI vorinstalliert. Wenn Sie den neuesten, nicht getesteten Nightly Build ausführen möchten, können Sie Installieren Sie Nightly Build von TensorFlow 2 (experimentell) manuell ausführen.

2 aktivieren TensorFlow

Um mit TensorFlow Conda auf dem DLAMI zu laufen

1. Um TensorFlow 2 zu aktivieren, öffnen Sie eine Amazon Elastic Compute Cloud (Amazon EC2) - Instanz des DLAMI mit Conda.
2. Führen Sie für TensorFlow 2 und Keras 2 auf Python 3 mit CUDA 10.1 und MKL-DNN diesen Befehl aus:

   ```
   $ source activate tensorflow2_p36
   ```

3. Starten Sie das iPython-Terminal:

   ```
   (tensorflow2_p36)$ ipython
   ```

4. Führen Sie ein TensorFlow 2-Programm aus, um zu überprüfen, ob es ordnungsgemäß funktioniert:

   ```
   import tensorflow as tf
   hello = tf.constant('Hello, TensorFlow!')
   tf.print(hello)
   ```

   Auf Ihrem Bildschirm sollte `Hello, TensorFlow!` angezeigt werden.

Installieren Sie Nightly Build von TensorFlow 2 (experimentell)

Sie können den neuesten TensorFlow 2-Build in einer oder beiden der TensorFlow beiden Conda-Umgebungen auf Ihrem Deep Learning-AMI mit Conda installieren.

Um von einem TensorFlow Nightly-Build aus zu installieren

1. Führen Sie für die Python 3 TensorFlow 2-Umgebung den folgenden Befehl aus:

   ```
   $ source activate tensorflow2_p36
   ```

2. Entfernen Sie den aktuell installierten TensorFlow.

> **Note**
>
> Für die restlichen Schritte wird davon ausgegangen, dass Sie die `tensorflow2_p36`-Umgebung verwenden.

```
(tensorflow2_p36)$ pip uninstall tensorflow
```

3. Installieren Sie den neuesten Nightly-Build von. TensorFlow

```
(tensorflow2_p36)$ pip install tf-nightly
```

4. Um zu überprüfen, ob Sie den neuesten Nightly-Build erfolgreich installiert haben, starten Sie das IPython-Terminal und überprüfen Sie die Version von. TensorFlow

```
(tensorflow2_p36)$ ipython
```

```
import tensorflow
print (tensorflow.__version__)
```

Die Druckausgabe sollte in etwa wie folgt aussehen: `2.1.0-dev20191122`

Weitere Tutorials

Tutorials finden Sie in dem Ordner namens `Deep Learning AMI with Conda tutorials` im Home-Verzeichnis des DLAMI.

Weitere Tutorials und Beispiele finden Sie in der TensorFlow Dokumentation zur TensorFlow Python-API oder TensorFlowauf der Website.

TensorFlow mit Horovod

Dieses Tutorial zeigt, wie man TensorFlow mit Horovod auf einem AWS Deep Learning AMI (DLAMI) mit Conda aktiviert. Horovod ist in den Conda-Umgebungen für vorinstalliert. TensorFlow Die Python3-Umgebung wird empfohlen.

> **Note**
>
> Nur Instances vom Typ P3.*, P2.* und G3.* werden unterstützt.

Um Horovod auf dem DLAMI mit Conda zu aktivieren TensorFlow und zu testen

1. Öffnen Sie mit Conda eine Amazon Elastic Compute Cloud (Amazon EC2) -Instanz des DLAMI. Hilfe zu den ersten Schritten mit einem DLAMI finden Sie unter. the section called “Wie fange ich mit dem DLAMI an”
2. (Empfohlen) Führen Sie für TensorFlow 1.15 mit Horovod auf Python 3 mit CUDA 11 den folgenden Befehl aus:

   ```
   $ source activate tensorflow_p37
   ```

3. Starten Sie das iPython-Terminal:

   ```
   (tensorflow_p37)$ ipython
   ```

4. Testen Sie den Import TensorFlow mit Horovod, um sicherzustellen, dass er ordnungsgemäß funktioniert:

   ```
   import horovod.tensorflow as hvd
   hvd.init()
   ```

   Folgendes wird ggf. auf dem Bildschirm angezeigt (Sie können alle Warnmeldungen ignorieren).

   ```
   --------------------------------------------------------------------------
   [[55425,1],0]: A high-performance Open MPI point-to-point messaging module
   was unable to find any relevant network interfaces:

   Module: OpenFabrics (openib)
     Host: ip-172-31-72-4

   Another transport will be used instead, although this may result in
   lower performance.
   --------------------------------------------------------------------------
   ```

Weitere Infos

- TensorFlow mit Horovod
- Tutorials finden Sie im `examples/horovod` Ordner im Home-Verzeichnis des DLAMI.
- Noch mehr Tutorials und Beispiele finden Sie im GitHub Horovod-Projekt.

TensorFlow 2 mit Horovod

Dieses Tutorial zeigt, wie man TensorFlow 2 mit Horovod auf einem AWS Deep Learning AMI (DLAMI) mit Conda aktiviert. Horovod ist in den Conda-Umgebungen für 2 vorinstalliert. TensorFlow Die Python3-Umgebung wird empfohlen.

> **Note**
>
> Nur Instances vom Typ P3.*, P2.* und G3.* werden unterstützt.

Um TensorFlow 2 zu aktivieren und Horovod auf dem DLAMI mit Conda zu testen

1. Öffnen Sie mit Conda eine Amazon Elastic Compute Cloud (Amazon EC2) -Instanz des DLAMI. Hilfe zu den ersten Schritten mit einem DLAMI finden Sie unter. the section called “Wie fange ich mit dem DLAMI an”
   - (Empfohlen) Führen Sie für TensorFlow 2 mit Horovod auf Python 3 mit CUDA 10 diesen Befehl aus:

     ```
     $ source activate tensorflow2_p36
     ```

   - Führen Sie für TensorFlow 2 mit Horovod auf Python 2 mit CUDA 10 diesen Befehl aus:

     ```
     $ source activate tensorflow2_p27
     ```

2. Starten Sie das iPython-Terminal:

```
(tensorflow2_p36)$ ipython
```

3. Testen Sie den Import von TensorFlow 2 mit Horovod, um sicherzustellen, dass er ordnungsgemäß funktioniert:

```
import horovod.tensorflow as hvd
hvd.init()
```

Wenn Sie keine Ausgabe erhalten, funktioniert Horovod ordnungsgemäß. Folgendes wird ggf. auf dem Bildschirm angezeigt (Sie können alle Warnmeldungen ignorieren).

```
--------------------------------------------------------------------------
[[55425,1],0]: A high-performance Open MPI point-to-point messaging module
was unable to find any relevant network interfaces:

Module: OpenFabrics (openib)
  Host: ip-172-31-72-4

Another transport will be used instead, although this may result in
lower performance.
--------------------------------------------------------------------------
```

Weitere Infos

- Tutorials finden Sie im `examples/horovod` Ordner im Home-Verzeichnis des DLAMI.
- Noch mehr Tutorials und Beispiele finden Sie im GitHub Horovod-Projekt.

Theano

Theano-Tutorial

> **Note**
>
> Wir schließen die Umgebungen CNTK, Caffe, Caffe2 und Theano Conda bei Version v28 nicht mehr in den AWS Deep Learning AMI -Start ein. Frühere Versionen von AWS Deep

> Learning AMI , die diese Umgebungen enthalten, werden weiterhin verfügbar sein. Wir werden jedoch nur Updates für diese Umgebungen bereitstellen, wenn von der Open-Source-Community Sicherheitskorrekturen für diese Frameworks veröffentlicht werden.

Um das Framework zu aktivieren, folgen Sie diesen Anweisungen auf Ihrem Deep Learning AMI mit Conda.

Für Theano + Keras in Python 3 mit CUDA 9 mit cuDNN 7:

```
$ source activate theano_p36
```

Für Theano + Keras in Python 2 mit CUDA 9 mit cuDNN 7:

```
$ source activate theano_p27
```

Starten Sie das iPython-Terminal.

```
(theano_p36)$ ipython
```

Führen Sie ein kurzes Theano-Programm aus.

```
import numpy
import theano
import theano.tensor as T
from theano import pp
x = T.dscalar('x')
y = x ** 2
gy = T.grad(y, x)
pp(gy)
```

Theano sollte einen symbolischen Verlauf berechnen.

Weitere Tutorials

Weitere Tutorials und Beispiele finden Sie in den offiziellen Dokumenten des Frameworks, der Theano Python API und der Theano-Website.

Debugging und Visualisierung

Erfahren Sie mehr über die Debugging- und Visualisierungsoptionen für das DLAMI. Klicken Sie auf eine der Optionen, um zu erfahren, wie Sie diese verwenden.

Themen

MXBoard

Mit MxBoard können Sie Ihre MXNet-Läufe und -Grafiken mithilfe der TensorBoard Software visuell überprüfen und interpretieren. Es führt einen Webserver aus, der eine Webseite bereitstellt, die für die Ansicht der MXBoard-Visualisierungen und die Interaktion mit diesen dient.

MXNet TensorBoard, und MXBoard sind mit dem Deep Learning AMI mit Conda (DLAMI mit Conda) vorinstalliert. In diesem Tutorial verwenden Sie eine MxBoard-Funktion, um Protokolle zu generieren, die kompatibel sind mit. TensorBoard

Themen

Verwenden von MXNet mit MXBoard

Generieren Sie MXBoard-Protokolldaten, kompatibel mit TensorBoard

1. Stellen Sie mit Conda Connect zu Ihrer Amazon Elastic Compute Cloud (Amazon EC2) -Instanz des DLAMI her.
2. Aktivieren Sie die Python 3 MXNet-Umgebung.

   ```
   $ source activate mxnet_p36
   ```

3. Bereiten Sie ein Python-Skript zum Speichern von Daten durch den normalen Operator in einer Ereignisdatei vor. Die Daten werden mit abnehmender Standardabweichung zehnmal generiert und dann jedes Mal in eine Ereignisdatei geschrieben. Sie werden sehen, dass sich

die Datenverteilung immer mehr um den Mittelwert herum bewegt. Beachten Sie, dass Sie die Ereignisdatei im Protokollordner angeben. Sie übergeben diesen Ordnerpfad an die Binärdatei. TensorBoard

```
$ vi mxboard_normal.py
```

4. Kopieren Sie folgendes Element in die Datei und speichern Sie es:

```
import mxnet as mx
from mxboard import SummaryWriter


with SummaryWriter(logdir='./logs') as sw:
    for i in range(10):
        # create a normal distribution with fixed mean and decreasing std
        data = mx.nd.normal(loc=0, scale=10.0/(i+1), shape=(10, 3, 8, 8))
        sw.add_histogram(tag='norml_dist', values=data, bins=200, global_step=i)
```

5. Führen Sie das Skript aus. Dadurch werden Protokolle in einem `logs`-Ordner generiert, den Sie für Visualisierungen verwenden können.

```
$ python mxboard_normal.py
```

6. Jetzt müssen Sie zur zu verwendenden TensorFlow Umgebung TensorBoard und zu MxBoard wechseln, um die Protokolle zu visualisieren. Dies ist eine erforderliche Abhängigkeit für MxBoard und. TensorBoard

```
$ source activate tensorflow_p36
```

7. Übergeben Sie `tensorboard` den Speicherort der Protokolle:

```
$ tensorboard --logdir=./logs --host=127.0.0.1 --port=8888
```

TensorBoard startet den Visualisierungs-Webserver auf Port 8888.

8. Für einen einfachen Zugriff von Ihrem lokalen Browser aus können Sie den Webserver-Port auf Port 80 oder einen anderen Port ändern. Welchen Port Sie auch verwenden, Sie müssen diesen Port in der EC2-Sicherheitsgruppe für Ihr DLAMI öffnen. Sie können auch eine Port-Weiterleitung verwenden. Anweisungen zum Ändern Ihrer Sicherheitsgruppeneinstellungen

und für Port-Weiterleitung finden Sie unter Einrichten eines Jupyter-Notebook-Servers. Die Standardeinstellungen sind im nächsten Schritt beschrieben.

> **Note**
>
> Wenn Sie sowohl einen Jupyter- als auch einen MXBoard-Server ausführen müssen, verwenden Sie jeweils einen separaten Port dafür.

9. Öffnen Sie Port 8888 (oder den Port, den Sie dem Webserver für die Visualisierung zugeordnet haben) auf Ihrer EC2-Instance.

   a. Öffnen Sie Ihre EC2-Instance in der Amazon EC2-Konsole unter https://console.aws.amazon.com/ec2/.

   b. Wählen Sie in der Amazon EC2 EC2-Konsole Network & Security und anschließend Security Groups aus.

   c. Für Security Group (Sicherheitsgruppe) wählen Sie die zuletzt erstellte Sicherheitsgruppe (siehe Zeitstempel in der Beschreibung).

   d. Wählen Sie die Registerkarte Inbound (Eingehend) und anschließend Edit (Bearbeiten) aus.

   e. Klicken Sie auf Add Rule (Regel hinzufügen).

   f. Geben Sie in der neuen Zeile Folgendes ein:

      Type (Typ) : Benutzerdefinierte **`TCP Rule`**

      Protocol (Protokoll): **`TCP`**

      Port Range (Port-Bereich): **`8888`** (oder den Port, den Sie dem Visualisierungsserver zugeordnet haben)

      Quelle: **`Custom IP (specify address/range)`**

10. Wenn Sie die Daten aus dem lokalen Browser visualisieren möchten, geben Sie den folgenden Befehl ein, um die auf der EC2-Instance gerenderten Daten an Ihren lokalen Computer weiterzuleiten.

```
$ ssh -Y -L localhost:8888:localhost:8888 user_id@ec2_instance_ip
```

11. Öffnen Sie die Webseite für die MxBoard-Visualisierungen, indem Sie die öffentliche IP- oder DNS-Adresse der EC2-Instanz verwenden, auf der das DLAMI mit Conda ausgeführt wird, und den Port, den Sie für MxBoard geöffnet haben:

http://127.0.0.1:8888

Weitere Infos

Weitere Informationen über MXBoard finden Sie auf der MXBoard-Website.

TensorBoard

TensorBoardermöglicht es Ihnen, Ihre TensorFlow Läufe und Grafiken visuell zu überprüfen und zu interpretieren. Es läuft auf einem Webserver, der eine Webseite für die Anzeige und Interaktion mit den TensorBoard Visualisierungen bereitstellt.

TensorFlow und TensorBoard sind mit dem Deep Learning AMI mit Conda (DLAMI mit Conda) vorinstalliert. Das DLAMI mit Conda enthält auch ein Beispielskript, mit dem ein MNIST-Modell trainiert wird TensorFlow , wobei zusätzliche Protokollierungsfunktionen aktiviert sind. MNIST ist eine Datenbank von Hand geschriebener Zahlen, das häufig verwendet wird, um Bilderkennungsmodelle zu trainieren. In diesem Tutorial verwenden Sie das Skript, um ein MNIST-Modell zu trainieren, TensorBoard und die Protokolle, um Visualisierungen zu erstellen.

Themen

- Trainieren Sie ein MNIST-Modell und visualisieren Sie das Training mit TensorBoard
- Weitere Infos

Trainieren Sie ein MNIST-Modell und visualisieren Sie das Training mit TensorBoard

Visualisieren Sie das MNIST-Modelltraining mit TensorBoard

1. Stellen Sie mit Conda Connect zu Ihrer Amazon Elastic Compute Cloud (Amazon EC2) -Instanz des DLAMI her.
2. Aktivieren Sie die Python TensorFlow 2.7-Umgebung und navigieren Sie zu dem Verzeichnis, das den Ordner mit den TensorBoard Beispielskripten enthält:

   ```
   $ source activate tensorflow_p27
   $ cd ~/examples/tensorboard/
   ```

3. Führen Sie das Skript aus, das ein MNIST-Modell mit aktivierter erweiterter Protokollierung trainiert:

```
$ python mnist_with_summaries.py
```

Das Skript schreibt die Protokolle in `/tmp/tensorflow/mnist`.

4. Übergeben Sie `tensorboard` den Speicherort der Protokolle:

```
$ tensorboard --logdir=/tmp/tensorflow/mnist
```

TensorBoard startet den Visualisierungs-Webserver auf Port 6006.

5. Für einen einfachen Zugriff von Ihrem lokalen Browser aus können Sie den Webserver-Port auf Port 80 oder einen anderen Port ändern. Welchen Port Sie auch verwenden, Sie müssen diesen Port in der EC2-Sicherheitsgruppe für Ihr DLAMI öffnen. Sie können auch eine Port-Weiterleitung verwenden. Anweisungen zum Ändern Ihrer Sicherheitsgruppeneinstellungen und für Port-Weiterleitung finden Sie unter Einrichten eines Jupyter-Notebook-Servers. Die Standardeinstellungen sind im nächsten Schritt beschrieben.

> **Note**
>
> Wenn Sie sowohl den Jupyter-Server als auch einen Server ausführen müssen, verwenden Sie für jeden einen TensorBoard anderen Port.

6. Öffnen Sie Port 6006 (oder den Port, den Sie dem Webserver für die Visualisierung zugeordnet haben) auf Ihrer EC2-Instance.

   a. Öffnen Sie Ihre EC2-Instance in der Amazon EC2-Konsole unter https://console.aws.amazon.com/ec2/.

   b. Wählen Sie in der Amazon EC2 EC2-Konsole Network & Security und anschließend Security Groups aus.

   c. Für Security Group (Sicherheitsgruppe) wählen Sie die zuletzt erstellte Sicherheitsgruppe (siehe Zeitstempel in der Beschreibung).

   d. Wählen Sie die Registerkarte Inbound (Eingehend) und anschließend Edit (Bearbeiten) aus.

   e. Klicken Sie auf Add Rule (Regel hinzufügen).

   f. Geben Sie in der neuen Zeile Folgendes ein:

      Type (Typ) : Benutzerdefinierte **`TCP Rule`**

      Protocol (Protokoll): **`TCP`**

Port Range (Port-Bereich): **`6006`** (oder den Port, den Sie dem Visualisierungsserver zugeordnet haben)

Quelle: **`Custom IP (specify address/range)`**

7. Öffnen Sie die Webseite für die TensorBoard Visualisierungen, indem Sie die öffentliche IP- oder DNS-Adresse der EC2-Instance verwenden, auf der das DLAMI mit Conda ausgeführt wird, und den Port, für den Sie geöffnet haben: TensorBoard

   http:// ***`YourInstancePublicDNS`*****`:6006`**

Weitere Infos

Weitere Informationen finden Sie auf der Website. TensorBoard TensorBoard

Verteilte Schulungen

Erfahren Sie mehr über die Optionen, die das DLAMI für das Training mit mehreren GPUs bietet. Weitere Informationen zur Leistungssteigerung finden Sie unter Elastic Fabric Adapter. Klicken Sie auf eine der Optionen, um etwas über die Verwendung zu erfahren.

Themen

- Chainer
- Keras mit MXNet
- TensorFlow mit Horovod

Chainer

> **Note**
>
> AWS Deep Learning AMI Ab der Version v28 sind Chainer Conda-Umgebungen nicht mehr enthalten. Frühere Versionen von AWS Deep Learning AMI , die diese Umgebungen enthalten, werden weiterhin verfügbar sein. Wir werden jedoch nur Updates für diese Umgebungen bereitstellen, wenn von der Open-Source-Community Sicherheitskorrekturen für diese Frameworks veröffentlicht werden.

Chainer ist ein flexibles, auf Python basierendes Framework für die einfache und intuitive Entwicklung komplexer neuronaler Netzwerkarchitekturen. Chainer macht es einfach, Multi-GPU-Instances für das Training zu verwenden. Darüber hinaus protokolliert Chainer automatisch Ergebnisse, Verluste und Genauigkeit von Graphen und erzeugt Ausgaben für die Darstellung des neuronalen Netzwerks mit einem Berechnungsgraphen. Es ist im Deep Learning AMI mit Conda (DLAMI with Conda) enthalten.

In den folgenden Themen wird erläutert, wie Sie ein Training auf mehreren GPUs, einer einzelnen GPU und einer CPU durchführen, Visualisierungen erstellen und Ihre Chainer-Installation testen.

Themen

Training eines Modells mit Chainer

In diesem Tutorial erfahren Sie, wie Sie Chainer-Beispielskripts verwenden, um ein Modell mit dem MNIST-Dataset zu trainieren. MNIST ist eine Datenbank von Hand geschriebener Zahlen, das häufig verwendet wird, um Bilderkennungsmodelle zu trainieren. Das Tutorial zeigt auch der Unterschied in der Trainingsgeschwindigkeit zwischen dem Training auf einer CPU und einer oder mehreren GPUs.

Verwendung von Chainer für das Training auf mehreren GPUs

Training auf mehreren GPUs

1. Stellen Sie mit Conda Connect zu der Instance her, auf der Deep Learning AMI ausgeführt wird. Informationen zur Auswahl the section called “Auswahl der Instance” oder Verbindung zu einer Instance finden Sie in der oder in der Amazon EC2 EC2-Dokumentation. Zum Ausführen dieses Tutorials können Sie eine Instance mit mindestens zwei GPUs verwenden.
2. Aktivieren der Python 3 Chainer-Umgebung:

```
$ source activate chainer_p36
```

3. Um die neuesten Tutorials zu erhalten, klonen Sie das Chainer-Repository und öffnen den Ordner mit den Beispielen:

```
(chainer_p36) :~$ cd ~/src
(chainer_p36) :~/src$ CHAINER_VERSION=v$(python -c "import chainer;
 print(chainer.__version__)")
(chainer_p36) :~/src$ git clone -b $CHAINER_VERSION https://github.com/chainer/
chainer.git
(chainer_p36) :~/src$ cd chainer/examples/mnist
```

4. Führen Sie das Beispiel im Skript `train_mnist_data_parallel.py` aus. Standardmäßig verwendet das Skript die GPUs, die auf Ihrer Instanz von Deep Learning AMI mit Conda ausgeführt werden. Das Skript kann auf maximal zwei GPUs ausgeführt werden. Es ignoriert alle weiteren GPUs. Es erkennt eine oder beide automatisch. Wenn Sie eine Instance ohne GPUs ausführen, blättern Sie weiter zu Verwendung on Chainer zum Training mit CPUs, später in diesem Tutorial.

```
(chainer_p36) :~/src/chainer/examples/mnist$ python train_mnist_data_parallel.py
```

> **Note**
>
> In diesem Beispiel wird die folgende Fehlermeldung angezeigt, da die Aufnahme einer Beta-Funktion nicht im DLAMI enthalten ist.
> `chainerx ModuleNotFoundError: No module named 'chainerx'`

Während das Chainer-Skript ein Modell mit der MNIST-Datenbank trainiert, sehen Sie die Ergebnisse für jede Epoche.

Dann sehen Sie eine Beispielausgabe, wenn das Skript ausgeführt wird. Die folgende Beispielausgabe wurde auf einer p3.8xlarge-Instance ausgeführt. Die Ausgabe des Skripts zeigt "GPU: 0, 1" an, was bedeutet, dass es die ersten beiden der vier verfügbaren GPUs verwendet. Die Skripte verwenden in der Regel einen Index von GPUs, der mit Null beginnt, statt einer Gesamtzahl.

```
GPU: 0, 1

# unit: 1000
# Minibatch-size: 400
```

```
# epoch: 20

epoch       main/loss   validation/main/loss  main/accuracy  validation/main/
accuracy  elapsed_time
1           0.277561    0.114709              0.919933       0.9654
    6.59261
2           0.0882352   0.0799204             0.973334       0.9752
    8.25162
3           0.0520674   0.0697055             0.983967       0.9786
    9.91661
4           0.0326329   0.0638036             0.989834       0.9805
    11.5767
5           0.0272191   0.0671859             0.9917         0.9796
    13.2341
6           0.0151008   0.0663898             0.9953         0.9813
    14.9068
7           0.0137765   0.0664415             0.995434       0.982
    16.5649
8           0.0116909   0.0737597             0.996          0.9801
    18.2176
9           0.00773858  0.0795216             0.997367       0.979
    19.8797
10          0.00705076  0.0825639             0.997634       0.9785
    21.5388
11          0.00773019  0.0858256             0.9978         0.9787
    23.2003
12          0.0120371   0.0940225             0.996034       0.9776
    24.8587
13          0.00906567  0.0753452             0.997033       0.9824
    26.5167
14          0.00852253  0.082996              0.996967       0.9812
    28.1777
15          0.00670928  0.102362              0.997867       0.9774
    29.8308
16          0.00873565  0.0691577             0.996867       0.9832
    31.498
17          0.00717177  0.094268              0.997767       0.9802
    33.152
18          0.00585393  0.0778739             0.998267       0.9827
    34.8268
19          0.00764773  0.107757              0.9975         0.9773
    36.4819
20          0.00620508  0.0834309             0.998167       0.9834
    38.1389
```

5. Während Ihr Training läuft, ist es sinnvoll, sich Ihre GPU-Auslastung anzusehen. Sie können überprüfen, welche GPUs aktiv sind, und deren Auslastung einsehen. NVIDIA bietet ein Tool dafür, das mit dem Befehl `nvidia-smi` ausgeführt werden kann. Es wird jedoch nur eine Momentaufnahme der Auslastung angezeigt, daher ist es informativer, dies mit dem Linux-Befehl `watch` zu kombinieren. Der folgende Befehl verwendet `watch` mit `nvidia-smi` , um die aktuelle GPU-Auslastung jede Zehntelsekunde zu aktualisieren. Öffnen Sie eine weitere Terminalsitzung auf Ihrem DLAMI und führen Sie den folgenden Befehl aus:

```
(chainer_p36) :~$ watch -n0.1 nvidia-smi
```

Sie sehen eine Ausgabe ähnlich dem folgenden Ergebnis. Verwenden Sie `ctrl-c`, um das Tool zu schließen, oder führen Sie es einfach weiter aus, während Sie die anderen Beispiele in Ihrer ersten Terminalsitzung ausprobieren.

```
Every 0.1s: nvidia-smi                                  Wed Feb 28 00:28:50 2018

Wed Feb 28 00:28:50 2018
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 384.111                Driver Version: 384.111                   |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:00:1B.0 Off |                    0 |
| N/A   46C    P0    56W / 300W |    728MiB / 16152MiB |     10%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100-SXM2...  On   | 00000000:00:1C.0 Off |                    0 |
| N/A   44C    P0    53W / 300W |    696MiB / 16152MiB |      4%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla V100-SXM2...  On   | 00000000:00:1D.0 Off |                    0 |
| N/A   42C    P0    38W / 300W |     10MiB / 16152MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla V100-SXM2...  On   | 00000000:00:1E.0 Off |                    0 |
| N/A   46C    P0    40W / 300W |     10MiB / 16152MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
```

```
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0     54418      C   python                                     718MiB |
|    1     54418      C   python                                     686MiB |
+-----------------------------------------------------------------------------+
```

In diesem Beispiel sind GPU 0 und GPU 1 aktiv, und GPU 2 und 3 nicht aktiv. Sie können auch die Arbeitsspeichernutzung pro GPU anzeigen.

6. Nachdem das Training abgeschlossen ist, notieren Sie die in Ihrer ersten Terminalsitzung abgelaufene Zeit. Im Beispiel beträgt die abgelaufene Zeit 38,1389 Sekunden.

Verwendung von Chainer für das Training auf einer einzelnen GPU

Dieses Beispiel zeigt, wie man das Training auf einer einzelnen GPU ausführt. Sie können dies tun, wenn Sie nur eine GPU zur Verfügung haben, oder nur um zu sehen, wie Multi-GPU-Training mit Chainer skaliert werden kann.

Verwendung von Chainer für das Training auf einer einzelnen GPU

- In diesem Beispiel verwenden Sie ein weiteres Skript, `train_mnist.py`, und weisen es mit dem Argument `--gpu=0` an, nur GPU 0 zu verwenden. Wenn Sie sehen möchten, wie andere GPUs in der `nvidia-smi`-Konsole aktiviert werden, können Sie das Skript mit `--gpu=1` anweisen, GPU 1 zu verwenden.

```
(chainer_p36) :~/src/chainer/examples/mnist$ python train_mnist.py --gpu=0
```

```
GPU: 0
# unit: 1000
# Minibatch-size: 100
# epoch: 20

epoch       main/loss   validation/main/loss  main/accuracy  validation/main/
accuracy  elapsed_time
1           0.192348    0.0909235             0.940934       0.9719
    5.3861
2           0.0746767   0.069854              0.976566       0.9785
    8.97146
3           0.0477152   0.0780836             0.984982       0.976
    12.5596
```

```
4           0.0347092   0.0701098           0.988498       0.9783
    16.1577
5           0.0263807   0.08851             0.991515       0.9793
    19.7939
6           0.0253418   0.0945821           0.991599       0.9761
    23.4643
7           0.0209954   0.0683193           0.993398       0.981
    27.0317
8           0.0179036   0.080285            0.994149       0.9819
    30.6325
9           0.0183184   0.0690474           0.994198       0.9823
    34.2469
10          0.0127616   0.0776328           0.996165       0.9814
    37.8693
11          0.0145421   0.0970157           0.995365       0.9801
    41.4629
12          0.0129053   0.0922671           0.995899       0.981
    45.0233
13          0.0135988   0.0717195           0.995749       0.9857
    48.6271
14          0.00898215  0.0840777           0.997216       0.9839
    52.2269
15          0.0103909   0.123506            0.996832       0.9771
    55.8667
16          0.012099    0.0826434           0.996616       0.9847
    59.5001
17          0.0066183   0.101969            0.997999       0.9826
    63.1294
18          0.00989864  0.0877713           0.997116       0.9829
    66.7449
19          0.0101816   0.0972672           0.996966       0.9822
    70.3686
20          0.00833862  0.0899327           0.997649       0.9835
    74.0063
```

In diesem Beispiel dauerte die Ausführung auf einer einzelnen GPU fast doppelt so lange! Das Training größerer Modelle oder Datasets ergibt unterschiedliche Ergebnisse aus diesem Beispiel, Sie müssen also ein bisschen experimentieren, um die GPU-Leistung auszuwerten.

Verwendung on Chainer zum Training mit CPUs

Jetzt versuchen Sie, ein Training nur mit einer CPU auszuführen. Führen Sie dasselbe Skript aus, `python train_mnist.py`, ohne Argumente:

```
(chainer_p36) :~/src/chainer/examples/mnist$ python train_mnist.py
```

In der Ausgabe zeigt `GPU: -1`, dass keine GPU verwendet wird:

```
GPU: -1
# unit: 1000
# Minibatch-size: 100
# epoch: 20

epoch       main/loss   validation/main/loss  main/accuracy  validation/main/accuracy
 elapsed_time
1           0.192083    0.0918663             0.94195        0.9712
 11.2661
2           0.0732366   0.0790055             0.977267       0.9747
 23.9823
3           0.0485948   0.0723766             0.9844         0.9787
 37.5275
4           0.0352731   0.0817955             0.987967       0.9772
 51.6394
5           0.029566    0.0807774             0.990217       0.9764
 65.2657
6           0.025517    0.0678703             0.9915         0.9814
 79.1276
7           0.0194185   0.0716576             0.99355        0.9808
 93.8085
8           0.0174553   0.0786768             0.994217       0.9809
 108.648
9           0.0148924   0.0923396             0.994983       0.9791
 123.737
10          0.018051    0.099924              0.99445        0.9791
 139.483
11          0.014241    0.0860133             0.995783       0.9806
 156.132
12          0.0124222   0.0829303             0.995967       0.9822
 173.173
13          0.00846336  0.122346              0.997133       0.9769
 190.365
```

```
14          0.011392    0.0982324             0.996383       0.9803
 207.746
15          0.0113111   0.0985907             0.996533       0.9813
 225.764
16          0.0114328   0.0905778             0.996483       0.9811
 244.258
17          0.00900945  0.0907504             0.9974         0.9825
 263.379
18          0.0130028   0.0917099             0.996217       0.9831
 282.887
19          0.00950412  0.0850664             0.997133       0.9839
 303.113
20          0.00808573  0.112367              0.998067       0.9778
 323.852
```

In diesem Beispiel wurde MNIST in 323 Sekunden trainiert, was mehr als 11x länger als ein Training mit zwei GPUs ist. Wenn Sie jemals an der Leistungsfähigkeit von GPUs gezweifelt haben, zeigt dieses Beispiel, wie viel effizienter sie sind.

Ergebnisse grafisch darstellen

Darüber hinaus protokolliert Chainer automatisch Ergebnisse, Verluste und Genauigkeit von Graphen und erzeugt Ausgaben für die Darstellung des Berechnungsgraphen.

Den Berechnungsgraphen erzeugen

1. Nachdem ein Trainingslauf abgeschlossen ist, gehen Sie zu dem Verzeichnis `result` und zeigen die Präzision und den Verlust des Laufs in Form zwei automatisch generierter Bilder an. Öffnen Sie es jetzt und listen Sie den Inhalt auf:

   ```
   (chainer_p36) :~/src/chainer/examples/mnist$ cd result
   (chainer_p36) :~/src/chainer/examples/mnist/result$ ls
   ```

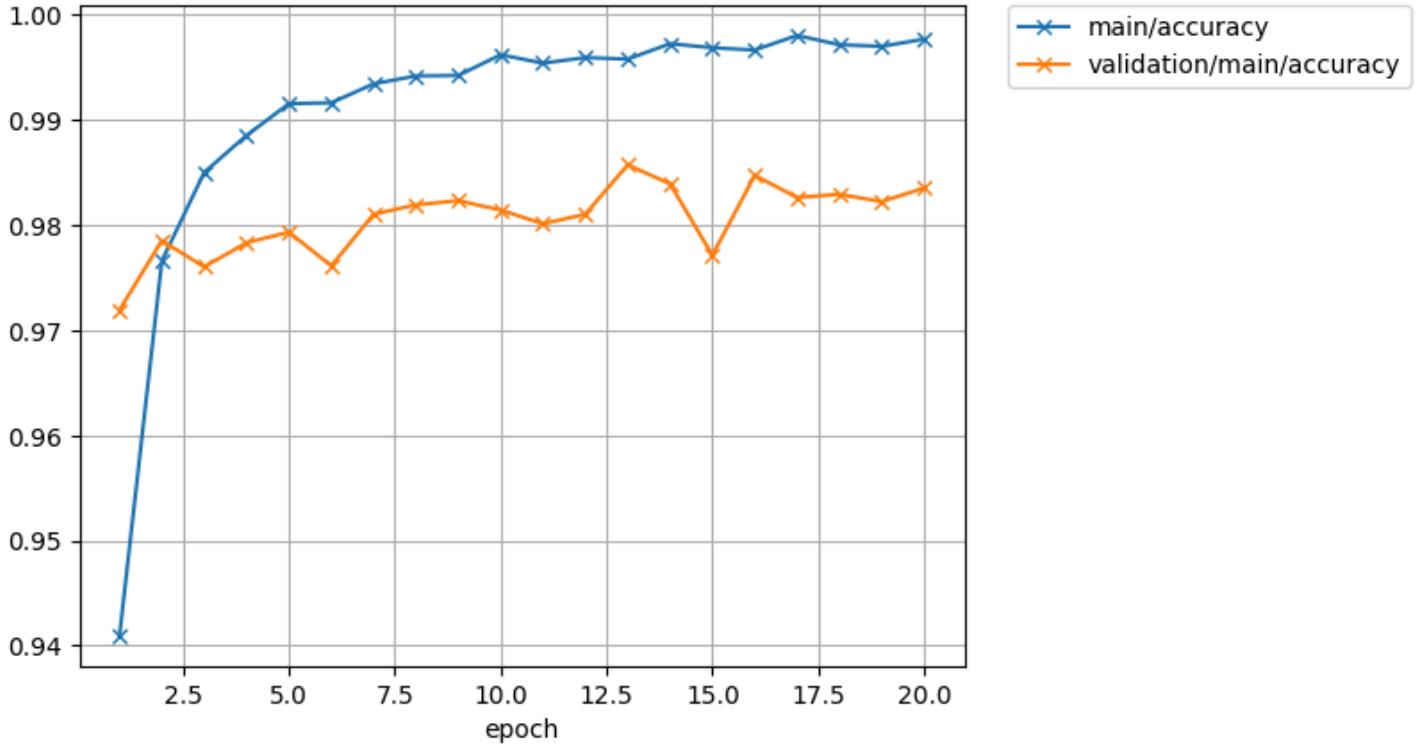
   Das Verzeichnis `result` enthält zwei Dateien im .png-Format: `accuracy.png` and `loss.png`.
2. Um die Graphen anzuzeigen, verwenden Sie den Befehl `scp`, um sie auf Ihren lokalen Computer zu kopieren.

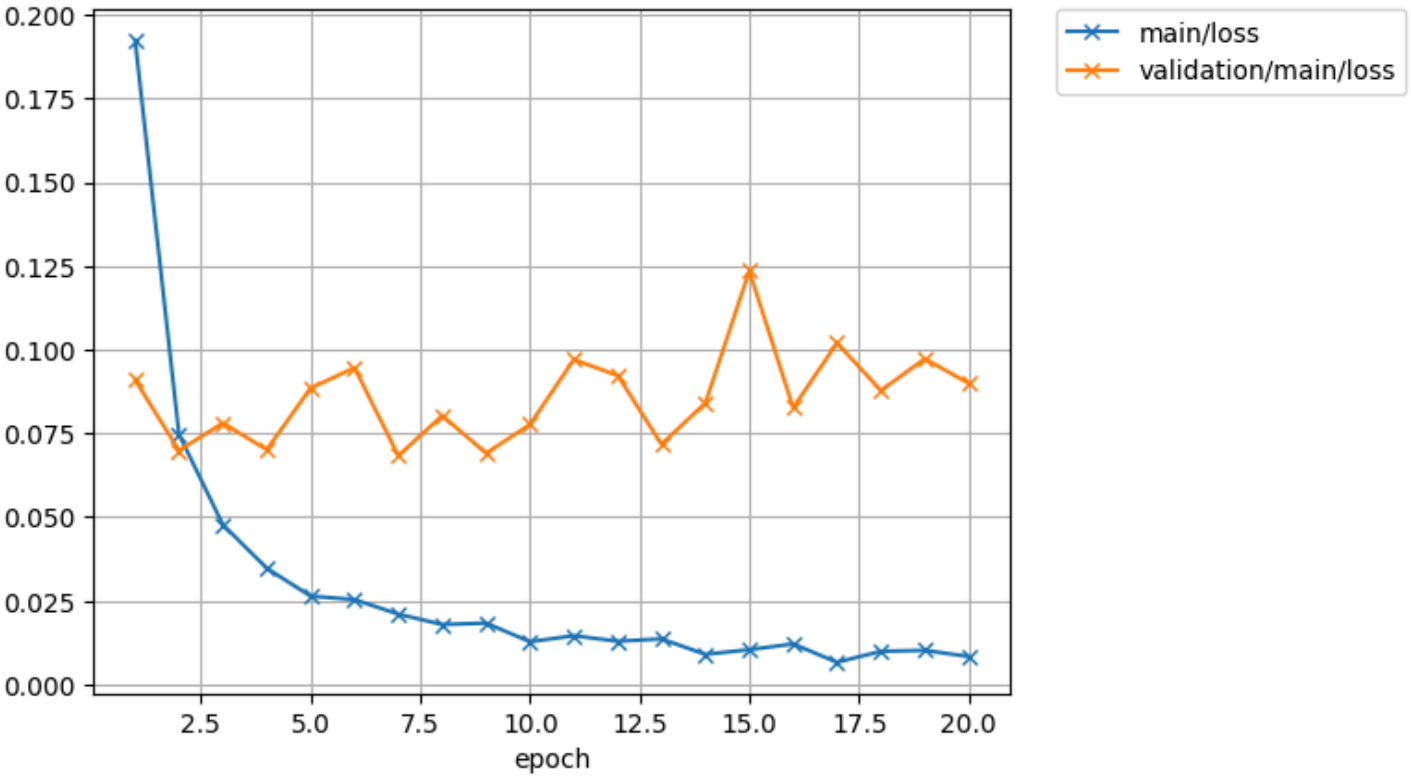
   Bei einem macOS-Terminal werden durch die Ausführung des folgenden `scp`-Befehls alle drei Dateien in Ihren `Downloads`-Ordner heruntergeladen. Ersetzen Sie die Platzhalter für

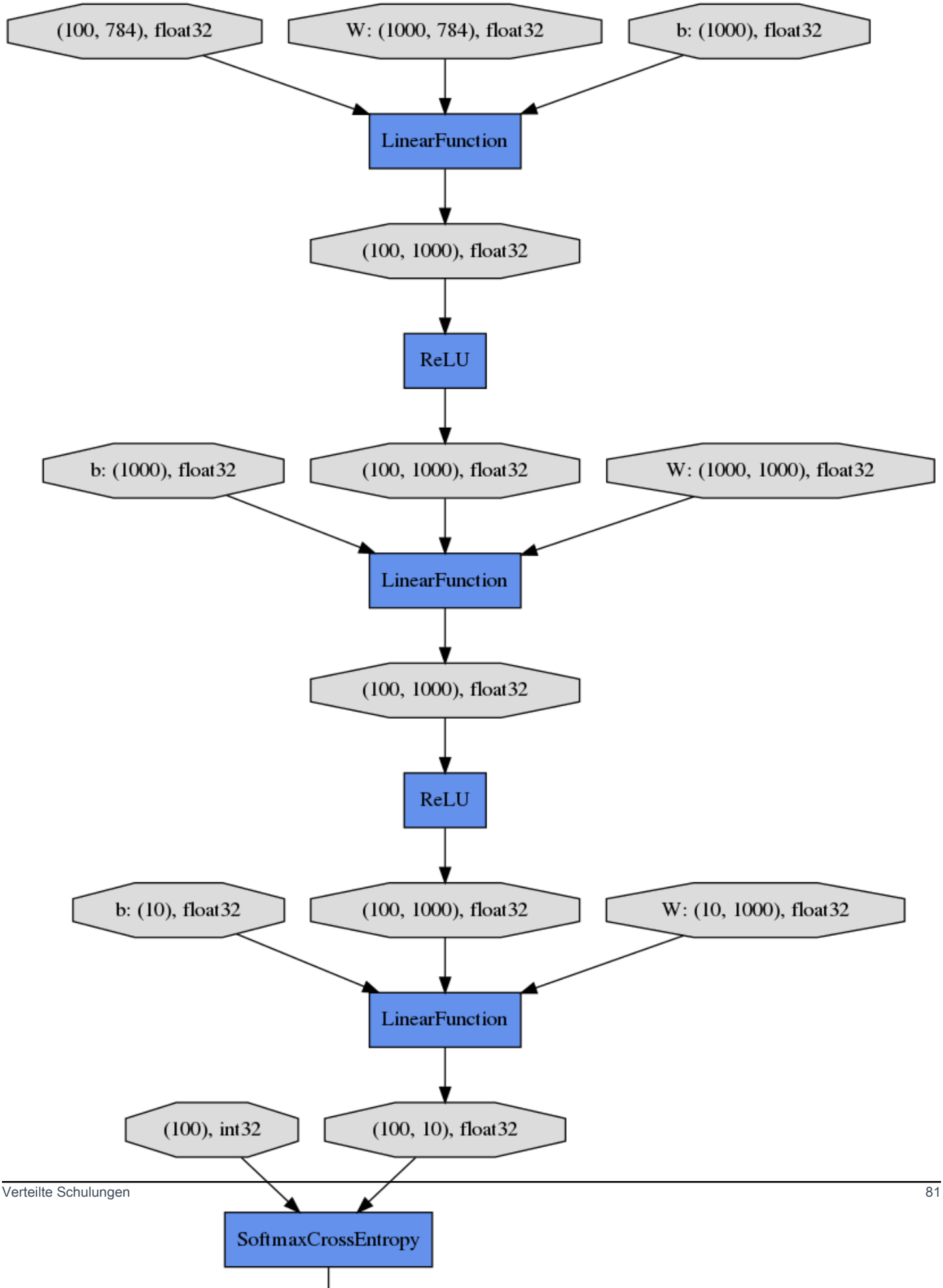
den Speicherort der Schlüsseldatei und die Server-Adresse durch Ihre Daten. Für andere Betriebssysteme verwenden Sie das entsprechende `scp`-Befehlsformat. Hinweis: Bei einem Amazon Linux-AMI lautet der Benutzername ec2-user.

```
(chainer_p36) :~/src/chainer/examples/mnist/result$ scp -i "your-key-file.pem"
 ubuntu@your-dlami-address.compute-1.amazonaws.com:~/src/chainer/examples/mnist/
result/*.png ~/Downloads
```

Die folgenden Abbildungen zeigen Beispiele für Genauigkeit, Verlust bzw. Berechnungsgraphen.




0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
epoch
main/loss
validation/main/loss


(100, 784), float32
W: (1000, 784), float32
b: (1000), float32
LinearFunction
(100, 1000), float32
ReLU
b: (1000), float32
(100, 1000), float32
W: (1000, 1000), float32
LinearFunction
(100, 1000), float32
ReLU
b: (10), float32
(100, 1000), float32
W: (10, 1000), float32
LinearFunction
(100), int32
(100, 10), float32
SoftmaxCrossEntropy

Chainer testen

Um Chainer zu testen und die GPU-Unterstützung mit einem vorinstallierten Testskript zu überprüfen, führen Sie den folgenden Befehl aus:

```
(chainer_p36) :~/src/chainer/examples/mnist/result$ cd ~/src/bin
(chainer_p36) :~/src/bin$ ./testChainer
```

Dieser lädt den Chainer-Quellcode herunter und führt das Beispiel Chainer multi-GPU MNIST aus.

Weitere Infos

Weitere Informationen über Chainer finden Sie auf der Website mit der Chainer-Dokumentation. Der Ordner mit den `Chainer`-Beispielen enthält weitere Beispiele. Probieren Sie sie aus, um zu sehen, wie sie funktionieren.

Keras mit MXNet

Dieses Tutorial zeigt, wie Sie Keras 2 mit dem MXNet-Backend auf einem Deep Learning-AMI mit Conda aktivieren und verwenden.

Aktivieren Sie Keras mit dem MXNet-Backend und testen Sie es auf dem DLAMI mit Conda

1. Um Keras mit dem MXNet-Backend zu aktivieren, öffnen Sie eine Amazon Elastic Compute Cloud (Amazon EC2) -Instanz des DLAMI mit Conda.
   - Führen Sie diesen Befehl für Python 3 aus:

     ```
     $ source activate mxnet_p36
     ```
   - Führen Sie diesen Befehl für Python 2 aus:

     ```
     $ source activate mxnet_p27
     ```
2. Starten Sie das iPython-Terminal:

   ```
   (mxnet_p36)$ ipython
   ```
3. Testen Sie das Importieren von Keras mit MXNet, um sicherzustellen, dass es korrekt funktioniert:

```
import keras as k
```

Folgendes wird auf dem Bildschirm angezeigt (möglicherweise nach ein paar Warnmeldungen).

```
Using MXNet backend
```

> **Note**
>
> Wenn Sie eine Fehlermeldung erhalten oder wenn das TensorFlow Backend immer noch verwendet wird, müssen Sie Ihre Keras-Konfiguration manuell aktualisieren. Bearbeiten Sie die `~/.keras/keras.json`-Datei und ändern Sie die Backendeinstellung in `mxnet`.

Tutorial der Keras-MXNet-Multi-GPU-Schulung

Schulen eines Convolutional Neural Network (CNN)

1. Öffnen Sie ein Terminal und stellen Sie eine SSH-Verbindung zu Ihrem DLAMI her.
2. Navigieren Sie zum Verzeichnis `~/examples/keras-mxnet/`.
3. Führen Sie `nvidia-smi` in Ihrem Terminal-Fenster aus, um die Anzahl der verfügbaren GPUs in DLAMI zu ermitteln. Im nächsten Schritt führen Sie das Skript unverändert aus, wenn Sie über vier GPUs verfügen.
4. (Optional) Führen Sie den folgenden Befehl aus, um das Skript zur Bearbeitung zu öffnen.

   ```
   (mxnet_p36)$ vi cifar10_resnet_multi_gpu.py
   ```

5. (Optional) Das Skript hat die folgende Zeile, die die Anzahl der GPUs definiert. Aktualisieren Sie sie bei Bedarf.

   ```
   model = multi_gpu_model(model, gpus=4)
   ```

6. Führen Sie jetzt die Schulung durch.

   ```
   (mxnet_p36)$ python cifar10_resnet_multi_gpu.py
   ```

> **Note**
>
> Keras-MXNet wird mit dem `channels_first` image_data_format-Satz bis zu zwei Mal schneller ausgeführt. Zum Ändern in `channels_first` bearbeiten Sie Ihre Keras-Konfigurationsdatei (`~/.keras/keras.json`) und legen Sie Folgendes fest: `"image_data_format": "channels_first"`.
> Weitere Techniken zur Leistungsoptimierung finden Sie im Handbuch zur Leistungsoptimierung für Keras-MXNet.

Weitere Infos

- Beispiele für Keras mit einem MXNet-Backend finden Sie im Verzeichnis Deep Learning AMI with Conda. `~/examples/keras-mxnet`
- Noch mehr Tutorials und Beispiele finden Sie im GitHub Keras-MXNet-Projekt.

TensorFlow mit Horovod

Dieses Tutorial zeigt die Verwendung TensorFlow mit Horovod auf einem Deep Learning-AMI mit Conda. Horovod ist in den Conda-Umgebungen für vorinstalliert. TensorFlow Die Python 3-Umgebung wird empfohlen. In diesen Anweisungen wird davon ausgegangen, dass Sie über eine funktionierende DLAMI-Instance mit mindestens einer GPU verfügen. Weitere Informationen finden Sie unter Wie fange ich mit dem DLAMI an.

> **Note**
>
> Nur Instances vom Typ P3.*, P2.* und G3.* werden unterstützt.

> **Note**
>
> Es gibt zwei Orte, an denen mpirun (via OpenMPI) verfügbar ist. Es ist in `/usr/bin` und `/home/ubuntu/anaconda3/envs/<env>/bin` verfügbar. `env` ist eine Umgebung, die dem Framework entspricht, wie Rahmenbedingungen und Apache MXNet. Die neueren OpenMPI-Versionen sind in den conda-Umgebungen verfügbar. Es wird empfohlen, den absoluten Pfad der mpirun-Binärdatei oder des -Präfix-Fags zu verwenden, um mpi-Workloads auszuführen. Beispiel: In der python36-Umgebung von Tensorflow verwenden sie entweder:

```
/home/ubuntu/anaconda3/envs/tensorflow_p36/bin/mpirun <args>

or

mpirun --prefix /home/ubuntu/anaconda3/envs/tensorflow_p36/bin <args>
```

Mit Horovod aktivieren und testen TensorFlow

1. Stellen Sie sicher, dass Ihre Instance über aktive GPUs verfügt. NVIDIA bietet dafür ein Tool:

   ```
   $ nvidia-smi
   ```

2. Aktiviere die TensorFlow Python-3-Umgebung:

   ```
   $ source activate tensorflow_p36
   ```

3. Starten Sie das iPython-Terminal:

   ```
   (tensorflow_p36)$ ipython
   ```

4. Testen Sie den Import TensorFlow mit Horovod, um sicherzustellen, dass er ordnungsgemäß funktioniert:

   ```
   import horovod.tensorflow as hvd
   hvd.init()
   ```

   Folgendes wird ggf. auf dem Bildschirm angezeigt (möglicherweise nach ein paar Warnmeldungen).

   ```
   --------------------------------------------------------------------------
   [[55425,1],0]: A high-performance Open MPI point-to-point messaging module
   was unable to find any relevant network interfaces:

   Module: OpenFabrics (openib)
     Host: ip-172-31-72-4

   Another transport will be used instead, although this may result in
   lower performance.
   ```

```
----------------------------------------------------------------------
```

Konfigurieren Ihrer Horovod Hosts-Datei

Sie können Horovod für GPU-Mehrfachschulungen von Einzelknoten oder für GPU-Mehrfachschulungen von mehreren Knoten verwenden. Wenn Sie mehrere Knoten für verteiltes Training verwenden möchten, müssen Sie jede private DLAMI-IP-Adresse zu einer Hosts-Datei hinzufügen. Das DLAMI, bei dem Sie gerade angemeldet sind, wird als Leader bezeichnet. Andere DLAMI-Instanzen, die Teil des Clusters sind, werden als Mitglieder bezeichnet.

Bevor Sie mit diesem Abschnitt beginnen, starten Sie ein oder mehrere DLAMI und warten Sie, bis sie sich im Status Bereit befinden. Die Beispielskripten erwarten eine Hosts-Datei. Selbst wenn Sie nur ein DLAMI verwenden möchten, erstellen Sie eine Hosts-Datei mit nur einem Eintrag. Wenn Sie die Hosts-Datei nach dem Beginn der Schulung bearbeiten, müssen Sie die Schulung neu starten, damit hinzugefügte oder entfernte Hosts wirksam werden.

So konfigurieren Sie Horovod für Schulungen

1. Ändern Sie die Verzeichnisse, in denen sich die Schulungs-Skripts befinden.

   ```
   cd ~/examples/horovod/tensorflow
   ```

2. Verwenden Sie vim, um eine Datei im Stammverzeichnis des Leiters zu bearbeiten.

   ```
   vim hosts
   ```

3. Wählen Sie eines der Mitglieder in der Amazon Elastic Compute Cloud-Konsole aus, und der Beschreibungsbereich der Konsole wird angezeigt. Suchen Sie das Feld Private IPs (Private IPs), kopieren Sie die IP und fügen Sie diese in eine Textdatei ein. Kopieren Sie die private IP jedes Mitglieds in eine neue Zeile. Anschließend fügen Sie neben jeder IP ein Leerzeichen und den Text `slots=8` hinzu (siehe unten). Dies stellt die Menge der GPUs in jeder Instance dar. Die p3.16xlarge-Instances verfügen über 8 GPUs. Wenn Sie also einen anderen Instance-Typ ausgewählt haben, geben Sie die tatsächliche Anzahl der GPUs für jede Instance an. Für den Leiter können Sie `localhost` verwenden. Ein Cluster mit 4 Knoten sollte wie folgt aussehen:

   ```
   172.100.1.200 slots=8
   172.200.8.99 slots=8
   172.48.3.124 slots=8
   localhost slots=8
   ```

Speichern Sie die Datei, und kehren Sie durch Beenden zum Terminal des Leiters zurück.

4. Fügen Sie den von den Mitglieds-Instances verwendeten SSH-Schlüssel zum SSH-Agent hinzu.

```
eval `ssh-agent -s`
ssh-add <key_name>.pem
```

5. Jetzt kann Ihr Leiter jedes Mitglied erreichen. All dies geschieht auf den Schnittstellen des privaten Netzwerks. Verwenden Sie anschließend eine kurze bash-Funktion, die Ihnen hilft, Befehle an jedes Mitglied zu senden.

```
function runclust(){ while read -u 10 host; do host=${host%% slots*}; ssh -o
 "StrictHostKeyChecking no" $host ""$2""; done 10<$1; };
```

6. Sagen Sie den anderen Mitgliedern, dass sie „StrickHostKeyChecking“ nicht ausführen sollen, da dies dazu führen kann, dass das Training nicht mehr reagiert.

```
runclust hosts "echo \"StrictHostKeyChecking no\" >> ~/.ssh/config"
```

Schulung mit synthetischen Daten

Ihr DLAMI wird mit einem Beispielskript geliefert, um ein Modell mit synthetischen Daten zu trainieren. Dadurch wird getestet, ob Ihr Leiter mit den Mitgliedern des Clusters kommunizieren kann. Eine Hosts-Datei ist erforderlich. Anweisungen finden Sie unter Konfigurieren Ihrer Horovod Hosts-Datei.

So testen Sie eine Horovod-Schulung mit Beispieldaten.

1. Standardmäßig verfügt `~/examples/horovod/tensorflow/train_synthetic.sh` über 8 GPUs, aber Sie können sie GPU-Anzahl bereitstellen, die Sie ausführen möchten. Im folgenden Beispiel wird ein Skript ausgeführt, mit dem 4 als ein Parameter für 4 GPUs übergeben wird.

```
$ ./train_synthetic.sh 4
```

Nach einigen Warnmeldungen sehen Sie die folgende Ausgabe, mit der bestätigt wird, dass Horovod 4 GPUs verwendet.

```
PY3.6.5 |Anaconda custom (64-bit)| (default, Apr 29 2018, 16:14:56) [GCC
 7.2.0]TF1.11.0Horovod size: 4
```

Nach weiteren Warnungen sehen Sie dann den Anfang einer Tabelle und einige Datenpunkte. Wenn Sie sich keine 1.000 Stapel ansehen möchten, brechen Sie die Schulung ab.

```
Step Epoch  Speed  Loss   FinLoss LR
0   0.0   105.6  6.794  7.708 6.40000
1   0.0   311.7  0.000  4.315 6.38721
100   0.1  3010.2  0.000 34.446 5.18400
200   0.2  3013.6  0.000 13.077 4.09600
300   0.2  3012.8  0.000  6.196 3.13600
400   0.3  3012.5  0.000  3.551 2.30401
```

2. Horovod verwendet alle lokalen GPUs, bevor versucht wird, die GPUs, der Cluster-Mitglieder zu verwenden. Um also sicherzustellen, dass eine Schulung für den Cluster funktioniert, testen Sie alle GPUs, die Sie verwenden möchten. Angenommen, Sie verfügen über vier Mitglieder des Instance-Typs p3.16xlarge mit 32 GPUs in Ihrem Cluster. An dieser Stelle sollten Sie alle 32GPUs testen.

   ```
   ./train_synthetic.sh 32
   ```

   Ihre Ausgabe ähnelt dann dem vorherigen Test. Horovod hat eine Größe von 32 und ist ungefähr viermal so schnell. Nach Abschluss dieses Experiments haben Sie Ihren Leiter und seine Fähigkeit zur Kommunikation mit den Mitgliedern getestet. Bei Problemen überprüfen Sie den Abschnitt Fehlerbehebung.

Bereiten Sie den Datensatz vor ImageNet

In diesem Abschnitt laden Sie den ImageNet Datensatz herunter und generieren dann einen Datensatz im TFRecord-Format aus dem Rohdatensatz. Auf der DLAMI steht eine Reihe von Vorverarbeitungsskripten für den ImageNet Datensatz zur Verfügung, die Sie entweder für einen Datensatz ImageNet oder als Vorlage für einen anderen Datensatz verwenden können. Die wichtigsten Trainingsskripte, für ImageNet die konfiguriert sind, werden ebenfalls bereitgestellt. Im folgenden Abschnitt wird davon ausgegangen, dass Sie ein DLAMI mit einer EC2-Instance mit 8 GPUs gestartet haben. Wir empfehlen den Instance-Typ p3.16xlarge.

Im `~/examples/horovod/tensorflow/utils` Verzeichnis auf Ihrem DLAMI finden Sie die folgenden Skripte:

- `utils/preprocess_imagenet.py`- Verwenden Sie dies, um den ImageNet Rohdatensatz in das Format zu konvertieren. `TFRecord`
- `utils/tensorflow_image_resizer.py`- Verwenden Sie dies, um die Größe des `TFRecord` Datensatzes wie für das ImageNet Training empfohlen zu ändern.

Bereiten Sie den Datensatz vor ImageNet

1. Besuchen Sie image-net.org, um ein Konto zu erstellen, einen Zugriffsschlüssel zu akquirieren und das Dataset herunterzuladen. image-net.org hostet das Raw-Dataset. Um es herunterzuladen, benötigen Sie ein ImageNet Konto und einen Zugangsschlüssel. Das Konto ist kostenlos, und um den kostenlosen Zugangsschlüssel zu erhalten, müssen Sie der ImageNet Lizenz zustimmen.
2. Verwenden Sie das Bildvorverarbeitungsskript, um aus dem ImageNet Rohdatensatz einen Datensatz im TFRecord-Format zu generieren. Im `~/examples/horovod/tensorflow/utils`-Verzeichnis:

```
python preprocess_imagenet.py \
         --local_scratch_dir=[YOUR DIRECTORY] \
         --imagenet_username=[imagenet account] \
         --imagenet_access_key=[imagenet access key]
```

3. Verwenden Sie das Bildskalierungsskript. Wenn Sie die Größe der Bilder ändern, läuft das Training schneller ab und passt sich besser an das ResNet Referenzpapier an. Im `~/examples/horovod/utils/preprocess`-Verzeichnis:

```
python tensorflow_image_resizer.py \
          -d imagenet \
          -i [PATH TO TFRECORD TRAINING DATASET] \
          -o  [PATH TO RESIZED TFRECORD TRAINING DATASET] \
          --subset_name train \
          --num_preprocess_threads 60 \
          --num_intra_threads 2 \
          --num_inter_threads 2
```

Trainieren Sie ein ResNet ImageNet -50-Modell auf einem einzigen DLAMI

> **Note**
>
> - Das Skript in diesem Tutorial sucht die vorverarbeiteten Schulungsdaten im Ordner `~/data/tf-imagenet/`. Anweisungen finden Sie unter Bereiten Sie den Datensatz vor ImageNet .
> - Eine Hosts-Datei ist erforderlich. Anweisungen finden Sie unter Konfigurieren Ihrer Horovod Hosts-Datei.

Verwenden Sie Horovod, um ein 50-CNN auf dem Datensatz zu ResNet trainieren ImageNet

1. Navigieren Sie zum Verzeichnis `~/examples/horovod/tensorflow`.

   ```
   cd ~/examples/horovod/tensorflow
   ```

2. Überprüfen Sie Ihre Konfiguration und legen Sie die Anzahl der GPUs fest, die in Schulungen verwendet werden sollen. Überprüfen Sie zuerst die `hosts`, die sich im selben Ordner wie die Skripts befindet. Diese Datei muss aktualisiert werden, wenn Sie eine Instance mit weniger als 8 GPUs verwenden. Standard: `localhost slots=8`. Aktualisieren Sie die Nummer 8 auf die Anzahl der GPUs, die Sie verwenden möchten.
3. Es wird ein Shell-Skript bereitgestellt, das die Anzahl der GPUs, die Sie verwenden möchten, als einzigen Parameter übernimmt. Führen Sie dieses Skript aus, um die Schulung zu starten. Im folgenden Beispiel wird 4 für vier GPUs verwendet.

   ```
   ./train.sh 4
   ```

4. Dies dauert ein paar Stunden. Es verwendet `mpirun` zur Verteilung der Schulung in Ihren gesamten GPUs.

Trainieren Sie ein ResNet ImageNet -50-Modell auf einem Cluster von DLAMIs

> **Note**
>
> - Das Skript in diesem Tutorial sucht die vorverarbeiteten Schulungsdaten im Ordner ~/data/tf-imagenet/. Anweisungen finden Sie unter Bereiten Sie den Datensatz vor ImageNet .
> - Eine Hosts-Datei ist erforderlich. Anweisungen finden Sie unter Konfigurieren Ihrer Horovod Hosts-Datei.

Dieses Beispiel führt Sie durch das Training eines ResNet -50-Modells mit einem vorbereiteten Datensatz über mehrere Knoten in einem DLAMI-Cluster.

- Wenn Sie eine höhere Leistung erzielen möchten, sollte sich das Dataset lokal auf jedem Mitglied des Clusters befinden.

  Verwenden Sie diese `copyclust`-Funktion, um Daten in andere Elemente zu kopieren.

  ```
  function copyclust(){ while read -u 10 host; do host=${host%% slots*}; rsync -azv
   "$2" $host:"$3"; done 10<$1; };
  ```

Falls sich die Dateien in einem S3-Bucket befinden, laden Sie sie mit der `runclust`-Funktion direkt in jedes Element herunter.

```
runclust hosts "tmux new-session -d \"export AWS_ACCESS_KEY_ID=YOUR_ACCESS_KEY &&
 export AWS_SECRET_ACCESS_KEY=YOUR_SECRET && aws s3 sync s3://your-imagenet-bucket
 ~/data/tf-imagenet/ && aws s3 sync s3://your-imagenet-validation-bucket ~/data/tf-
imagenet/\""
```

Tools, mit denen Sie mehrere Knoten gleichzeitig verwalten können, sparen Ihnen eine Menge Zeit. Sie können entweder auf jeden Schritt warten und jede Instance separat verwalten, oder Tools wie tmux oder screen nutzen, um Sitzungen zu trennen und wieder aufzunehmen.

Nachdem der Kopiervorgang abgeschlossen ist, können Sie die Schulung starten. Führen Sie das Skript aus und übergeben Sie 32 als Parameter für die 32 GPUs, die wir für diese Ausführung verwenden. Verwenden Sie tmux oder ein ähnliches Tool, falls Sie Bedenken wegen einer Trennung und Beendigung Ihrer Sitzung haben, da die Schulungsausführung dadurch unterbrochen würde.

```
./train.sh 32
```

Die folgende Ausgabe sehen Sie, wenn Sie das Training ImageNet mit 32 GPUs ausführen. Zweiunddreißig GPUs benötigen 90—110 Minuten.

```
    Step Epoch  Speed  Loss   FinLoss LR
    0   0.0   440.6  6.935  7.850 0.00100
    1   0.0  2215.4  6.923  7.837 0.00305
    50   0.3 19347.5  6.515  7.425 0.10353
    100   0.6 18631.7  6.275  7.173 0.20606
    150   1.0 19742.0  6.043  6.922 0.30860
    200   1.3 19790.7  5.730  6.586 0.41113
    250   1.6 20309.4  5.631  6.458 0.51366
    300   1.9 19943.9  5.233  6.027 0.61619
    350   2.2 19329.8  5.101  5.864 0.71872
    400   2.6 19605.4  4.787  5.519 0.82126
    ...
    13750  87.9 19398.8  0.676  1.082 0.00217
    13800  88.2 19827.5  0.662  1.067 0.00156
    13850  88.6 19986.7  0.591  0.997 0.00104
    13900  88.9 19595.1  0.598  1.003 0.00064
    13950  89.2 19721.8  0.633  1.039 0.00033
    14000  89.5 19567.8  0.567  0.973 0.00012
    14050  89.8 20902.4  0.803  1.209 0.00002
    Finished in 6004.354426383972
```

Nachdem eine Schulung abgeschlossen ist, führt das Skript eine Bewertungsausführung durch. Diese wird auf dem Leiter durchgeführt, da er schnell genug ausgeführt wird, ohne dass die Aufgabe auf andere Mitglieder verteilt werden muss. Das folgende Beispiel zeigt die Ausgabe der Bewertungsausführung:

```
Horovod size: 32
Evaluating
Validation dataset size: 50000
[ip-172-31-36-75:54959] 7 more processes have sent help message help-btl-vader.txt /
 cma-permission-denied
[ip-172-31-36-75:54959] Set MCA parameter "orte_base_help_aggregate" to 0 to see all
 help / error messages
  step  epoch  top1    top5     loss   checkpoint_time(UTC)
 14075   90.0  75.716   92.91    0.97  2018-11-14 08:38:28
```

Im Folgenden finden Sie ein Beispiel für eine Ausgabe bei Ausführung des Skripts mit 256 GPUs, wobei die Laufzeit zwischen 14 und 15 Minuten lag.

```
  Step Epoch  Speed  Loss   FinLoss LR
  1400  71.6 143451.0  1.189  1.720 0.14850
  1450  74.2 142679.2  0.897  1.402 0.10283
  1500  76.7 143268.6  1.326  1.809 0.06719
  1550  79.3 142660.9  1.002  1.470 0.04059
  1600  81.8 143302.2  0.981  1.439 0.02190
  1650  84.4 144808.2  0.740  1.192 0.00987
  1700  87.0 144790.6  0.909  1.359 0.00313
  1750  89.5 143499.8  0.844  1.293 0.00026
Finished in 860.5105031204224

Finished evaluation
1759   90.0  75.086   92.47    0.99  2018-11-20 07:18:18
```

Fehlerbehebung

Der folgende Befehl hilft möglicherweise Fehler zu beseitigen, die beim Experimentieren mit Horovod auftreten können.

- Falls die Schulung aus irgend einem Grund abstürzt, bereinigt mpirun möglicherweise nicht alle Python-Prozesse auf jeder Maschine. In diesem Fall sollten Sie, bevor Sie den nächsten Job starten, die Python-Prozesse auf allen Computern wie folgt beenden:

  ```
  runclust hosts "pkill -9 python"
  ```

- Wenn der Prozess abrupt ohne Fehler abgeschlossen wird, löschen Sie Ihren Protokollordner.

  ```
  runclust hosts "rm -rf ~/imagenet_resnet/"
  ```

- Bei anderen ungeklärten Problemen überprüfen Sie Ihre Speicherplatz. Wenn Sie nicht vor Ort sind, entfernen Sie den Protokollordner, da dieser voller Checkpoints und Daten ist. Sie können auch die Volumes für jedes Mitglied vergrößern.

  ```
  runclust hosts "df /"
  ```

- Als letztes Mittel können Sie neu starten.

```
runclust hosts "sudo reboot"
```

Möglicherweise wird Ihnen der folgende Fehlercode angezeigt, wenn Sie versuchen, TensorFlow mit Horovod einen Instanztyp zu verwenden, der nicht unterstützt wird:

```
---------------------------------------------------------------------------
NotFoundError Traceback (most recent call last)
<ipython-input-3-e90ed6cabab4> in <module>()
----> 1 import horovod.tensorflow as hvd

~/anaconda3/envs/tensorflow_p36/lib/python3.6/site-packages/horovod/tensorflow/
__init__.py in <module>()
** *34* check_extension('horovod.tensorflow', 'HOROVOD_WITH_TENSORFLOW', __file__,
 'mpi_lib')
** *35*
---> 36 from horovod.tensorflow.mpi_ops import allgather, broadcast, _allreduce
** *37* from horovod.tensorflow.mpi_ops import init, shutdown
** *38* from horovod.tensorflow.mpi_ops import size, local_size, rank, local_rank

~/anaconda3/envs/tensorflow_p36/lib/python3.6/site-packages/horovod/tensorflow/
mpi_ops.py in <module>()
** *56*
** *57* MPI_LIB = _load_library('mpi_lib' + get_ext_suffix(),
---> 58 ['HorovodAllgather', 'HorovodAllreduce'])
** *59*
** *60* _basics = _HorovodBasics(__file__, 'mpi_lib')

~/anaconda3/envs/tensorflow_p36/lib/python3.6/site-packages/horovod/tensorflow/
mpi_ops.py in _load_library(name, op_list)
** *43* """
** *44* filename = resource_loader.get_path_to_datafile(name)
---> 45 library = load_library.load_op_library(filename)
** *46* for expected_op in (op_list or []):
** *47* for lib_op in library.OP_LIST.op:

~/anaconda3/envs/tensorflow_p36/lib/python3.6/site-packages/tensorflow/python/
framework/load_library.py in load_op_library(library_filename)
** *59* RuntimeError: when unable to load the library or get the python wrappers.
** *60* """
---> 61 lib_handle = py_tf.TF_LoadLibrary(library_filename)
** *62*
```

```
** *63* op_list_str = py_tf.TF_GetOpList(lib_handle)

NotFoundError: /home/ubuntu/anaconda3/envs/tensorflow_p36/lib/python3.6/site-packages/
horovod/tensorflow/mpi_lib.cpython-36m-x86_64-linux-gnu.so: undefined symbol:
 _ZN10tensorflow14kernel_factory17OpKernelRegistrar12InitInternalEPKNS_9KernelDefEN4absl11strin
```

Weitere Infos

Dienstprogramme und Beispiele finden Sie im `~/examples/horovod` Ordner im Home-Verzeichnis des DLAMI.

Noch mehr Tutorials und Beispiele finden Sie im GitHub Horovod-Projekt.

Elastic Fabric Adapter

Ein Elastic Fabric Adapter (EFA) ist ein Netzwerkgerät, das Sie an Ihre DLAMI-Instanz anschließen können, um High Performance Computing (HPC) -Anwendungen zu beschleunigen. EFA ermöglicht es Ihnen, die Anwendungsleistung eines lokalen HPC-Clusters mit der Skalierbarkeit, Flexibilität und Elastizität der Cloud zu erreichen. AWS

Die folgenden Themen zeigen Ihnen, wie Sie mit der Verwendung von EFA mit dem DLAMI beginnen können.

> **Note**
>
> Wählen Sie Ihr DLAMI aus dieser Base-GPU-DLAMI-Liste

Themen

- Starten einer Instance mit EFA AWS Deep Learning AMI
- EFA auf dem DLAMI verwenden

Starten einer Instance mit EFA AWS Deep Learning AMI

Das neueste Base DLAMI ist sofort mit EFA einsatzbereit und wird mit den erforderlichen Treibern, Kernelmodulen, libfabric, openmpi und dem NCCL OFI-Plugin für GPU-Instanzen geliefert.

Die unterstützten CUDA-Versionen eines Basis-DLAMI finden Sie in den Versionshinweisen.

Hinweis:

- Wenn Sie eine NCCL-Anwendung `mpirun` auf EFA ausführen, müssen Sie den vollständigen Pfad zu der von EFA unterstützten Installation wie folgt angeben:

```
/opt/amazon/openmpi/bin/mpirun <command>
```

- Damit Ihre Anwendung EFA verwenden kann, fügen Sie `FI_PROVIDER="efa"` dem `mpirun`-Befehl hinzu, wie unter EFA auf dem DLAMI verwenden gezeigt.

Themen

- Bereiten Sie eine EFA-fähige Sicherheitsgruppe vor
- Starten Ihrer Instance
- Überprüfen Sie den EFA-Anhang

Bereiten Sie eine EFA-fähige Sicherheitsgruppe vor

EFA benötigt eine Sicherheitsgruppe, die den gesamten ein- und ausgehenden Datenverkehr zur und von der Sicherheitsgruppe selbst zulässt. Weitere Informationen finden Sie in der EFA-Dokumentation.

1. Öffnen Sie die Amazon EC2-Konsole unter https://console.aws.amazon.com/ec2/.
2. Wählen Sie im Navigationsbereich Security Groups (Sicherheitsgruppen) und anschließend Create Security Group (Sicherheitsgruppe erstellen) aus.
3. Führen Sie im Fenster Create Security Group Folgendes aus:
   - Geben Sie für Security group name (Name der Sicherheitsgruppe) einen beschreibenden Namen für die Sicherheitsgruppe ein, wie etwa `EFA-enabled security group`.
   - (Optional:) Geben Sie unter Description (Beschreibung) eine kurze Beschreibung der Sicherheitsgruppe ein.
   - Wählen Sie bei VPC die VPC aus, in der Sie Ihre EFA-fähigen Instances starten möchten.
   - Wählen Sie Create (Erstellen) aus.
4. Wählen Sie die von Ihnen erstellte Sicherheitsgruppe aus und kopieren Sie dann auf der Registerkarte Description (Beschreibung) die Group ID (Gruppen-ID).
5. Gehen Sie auf den Registerkarten Eingehend und Ausgehend wie folgt vor:
   - Wählen Sie Edit aus.

- Wählen Sie für Type (Typ) die Option All traffic (Gesamter Datenverkehr) aus.
- Wählen Sie für Source (Quelle) die Option Custom (Benutzerdefiniert) aus.
- Fügen Sie die Sicherheitsgruppen-ID, die Sie kopiert haben, in das Feld ein.
- Wählen Sie Speichern.

6. Aktivieren Sie eingehenden Datenverkehr, indem Sie entsprechend der Informationen unter Autorisieren von eingehendem Datenverkehr für Linux-Instances vorgehen. Wenn Sie diesen Schritt überspringen, können Sie nicht mit Ihrer DLAMI-Instanz kommunizieren.

Starten Ihrer Instance

EFA on the AWS Deep Learning AMI wird derzeit von den folgenden Instance-Typen und Betriebssystemen unterstützt:

- P3dn.24xlarge: Amazon Linux 2, Ubuntu 20.04
- P4D.24xlarge: Amazon Linux 2, Ubuntu 20.04
- P5.48xlarge: Amazon Linux 2, Ubuntu 20.04

Der folgende Abschnitt zeigt, wie eine EFA-fähige DLAMI-Instanz gestartet wird. Weitere Informationen zum Starten einer EFA-fähigen Instance finden Sie unter Starten von EFA-fähigen Instances in einer Cluster Placement-Gruppe.

1. Öffnen Sie die Amazon EC2-Konsole unter https://console.aws.amazon.com/ec2/.
2. Wählen Sie Launch Instance aus.
3. Wählen Sie auf der Seite „AMI auswählen“ ein unterstütztes DLAMI aus, das Sie auf der Seite mit den DLAMI-Versionshinweisen finden.
4. Wählen Sie auf der Seite „Instance-Typ auswählen“ einen der folgenden unterstützten Instance-Typen aus und klicken Sie dann auf Weiter: Instance-Details konfigurieren. Unter diesem Link finden Sie eine Liste der unterstützten Instances: Erste Schritte mit EFA und MPI
5. Führen Sie auf der Seite Configure Instance Details (Instance-Details konfigurieren) die folgenden Schritte aus:

   - Geben Sie für Number of instances (Anzahl der Instances) die Anzahl der EFA-aktivierten Instances ein, die gestartet werden sollen.
   - Wählen Sie für Network (Netzwerk) und Subnet (Subnetz) die VPC und das Subnetz an, in das Sie die Instances starten möchten.

- [Optional] Wählen Sie für Placement-Gruppe die Option Instance zur Placement-Gruppe hinzufügen aus. Um eine optimale Leistung zu erzielen, starten Sie die Instances innerhalb einer Platzierungsgruppe.
- [Optional] Wählen Sie für Placement-Gruppenname die Option Zu einer neuen Placement-Gruppe hinzufügen aus, geben Sie einen aussagekräftigen Namen für die Placement-Gruppe ein und wählen Sie dann für Placement-Gruppenstrategie die Option Cluster aus.
- Achten Sie darauf, den „Elastic Fabric Adapter“ auf dieser Seite zu aktivieren. Wenn diese Option deaktiviert ist, ändern Sie das Subnetz in ein Subnetz, das den ausgewählten Instance-Typ unterstützt.
- Wählen Sie im Abschnitt Network Interfaces (Netzwerkschnittstellen für das Gerät eth0 die Option New network interface (Neue Netzwerkschnittstelle) aus. Sie können zudem optional eine primäre IPv4-Adresse und eine oder mehrere sekundäre IPv4-Adressen eingeben. Wenn Sie die Instance in einem Subnetz starten, das einen verknüpften IPv6-CIDR-Block hat, können Sie optional eine primäre IPv6-Adresse und eine oder mehrere sekundäre IPv6-Adressen angeben.
- Wählen Sie Next: Add Storage aus.

6. Auf der Seite Add Storage (Speicher hinzufügen) können Sie neben den durch das AMI festgelegten Volumes (wie z. B. dem Root-Gerät-Volume) auch Datenträger angeben, die an die Instance angefügt werden sollen. Wählen Sie anschließend Next: Add Tags (Weiter: Tags (Markierungen) hinzufügen) aus.
7. Geben Sie auf der Seite Add Tags (Tags (Markierungen) hinzufügen) Tags (Markierungen) für die Instances an, z. B. einen benutzerfreundlichen Namen, und wählen Sie anschließend Next: Configure Security Group (Weiter: Sicherheitsgruppe konfigurieren).
8. Wählen Sie auf der Seite Sicherheitsgruppe konfigurieren für Sicherheitsgruppe zuweisen die Option Eine bestehende Sicherheitsgruppe auswählen und wählen Sie dann die Sicherheitsgruppe aus, die Sie zuvor erstellt haben.
9. Klicken Sie auf Review and Launch.
10. Überprüfen Sie auf der Seite Review Instance Launch (Instance-Start überprüfen) Ihre Einstellungen und wählen Sie anschließend Launch (Starten) aus, um ein Schlüsselpaar auszuwählen und Ihre Instance zu starten.

Überprüfen Sie den EFA-Anhang

Über die Konsole

Überprüfen Sie nach dem Start der Instance die Instance-Details in der AWS Konsole. Wählen Sie dazu die Instance in der EC2-Konsole aus und sehen Sie sich die Registerkarte "Description (Beschreibung)" im unteren Bereich auf der Seite an. Suchen Sie den Parameter "Network Interfaces: eth0" und klicken Sie auf "eth0", wodurch ein Pop-up geöffnet wird. Stellen Sie sicher, dass der 'Elastic Fabric Adapter' aktiviert ist.

Wenn EFA nicht aktiviert ist, können Sie dies wie folgt beheben:

- Beenden der EC2-Instance und Starten einer neuen Instance mit den gleichen Schritten. Stellen Sie sicher, dass die EFA angeschlossen ist.
- Hängen Sie EFA an eine bestehende Instanz an.

  1. Wechseln Sie in der EC2-Konsole zu "Network Interfaces (Netzwerkschnittstellen)".
  2. Klicken Sie auf "Create a Network Interface (Netzwerkschnittstelle erstellen)".
  3. Wählen Sie dasselbe Subnetz aus, in dem sich Ihre Instance befindet.
  4. Stellen Sie sicher, dass der 'Elastic Fabric Adapter' aktiviert ist, und klicken Sie auf Erstellen.
  5. Wechseln Sie zurück zur Registerkarte "EC2 Instances (EC-Instances)" und wählen Sie Ihre Instance aus.
  6. Gehen Sie zu Aktionen: Instanzstatus und beenden Sie die Instance, bevor Sie EFA anhängen.
  7. Wählen Sie unter "Actions (Aktionen)" die Option "Networking: Attach Network Interface (Netzwerk: Netzwerkschnittstelle anfügen)" aus.
  8. Wählen Sie die soeben erstellte Schnittstelle aus und klicken Sie auf "Attach (Anfügen)".
  9. Starten Sie Ihre Instance neu.

Über die Instance

Das folgende Testskript ist bereits auf dem DLAMI vorhanden. Führen Sie es aus, um sicherzustellen, dass die Kernel-Module ordnungsgemäß geladen sind.

```
$ fi_info -p efa
```

Ihre Ausgabe sollte in etwa wie folgt aussehen.

```
provider: efa
    fabric: EFA-fe80::e5:56ff:fe34:56a8
    domain: efa_0-rdm
    version: 2.0
    type: FI_EP_RDM
    protocol: FI_PROTO_EFA
provider: efa
    fabric: EFA-fe80::e5:56ff:fe34:56a8
    domain: efa_0-dgrm
    version: 2.0
    type: FI_EP_DGRAM
    protocol: FI_PROTO_EFA
provider: efa;ofi_rxd
    fabric: EFA-fe80::e5:56ff:fe34:56a8
    domain: efa_0-dgrm
    version: 1.0
    type: FI_EP_RDM
    protocol: FI_PROTO_RXD
```

Überprüfen der Sicherheitsgruppenkonfiguration

Das folgende Testskript ist bereits auf dem DLAMI vorhanden. Führen Sie es aus, um sicherzustellen, dass die von Ihnen erstellte Sicherheitsgruppe ordnungsgemäß konfiguriert ist.

```
$ cd /opt/amazon/efa/test/
$ ./efa_test.sh
```

Ihre Ausgabe sollte in etwa wie folgt aussehen.

```
Starting server...
Starting client...
bytes   #sent   #ack     total       time     MB/sec    usec/xfer   Mxfers/sec
64      10      =10      1.2k        0.02s      0.06    1123.55       0.00
256     10      =10      5k          0.00s     17.66      14.50       0.07
1k      10      =10      20k         0.00s     67.81      15.10       0.07
4k      10      =10      80k         0.00s    237.45      17.25       0.06
64k     10      =10      1.2m        0.00s    921.10      71.15       0.01
1m      10      =10      20m         0.01s   2122.41     494.05       0.00
```

Wenn es nicht mehr reagiert oder nicht vollständig ausgeführt wird, stellen Sie sicher, dass Ihre Sicherheitsgruppe über die richtigen Regeln für eingehende/ausgehende Nachrichten verfügt.

EFA auf dem DLAMI verwenden

Im folgenden Abschnitt wird beschrieben, wie EFA verwendet wird, um Anwendungen mit mehreren Knoten auf dem auszuführen. AWS Deep Learning AMI

Ausführen von Anwendungen mit mehreren Knoten mit EFA

Um eine Anwendung auf einem Knotencluster auszuführen, ist die folgende Konfiguration erforderlich

Themen

Aktivieren von passwortloser SSH

Wählen Sie einen Knoten im Cluster als Führungsknoten aus. Die verbleibenden Knoten werden als Mitgliedsknoten bezeichnet.

1. Generieren Sie auf dem Führungsknoten das RSA-Tastenpaar.

   ```
   ssh-keygen -t rsa -N "" -f ~/.ssh/id_rsa
   ```

2. Ändern Sie die Berechtigungen des privaten Schlüssels auf dem Führungsknoten.

   ```
   chmod 600 ~/.ssh/id_rsa
   ```

3. Kopieren Sie den öffentlichen Schlüssel `~/.ssh/id_rsa.pub` auf die Mitgliedsknoten im Cluster und hängen Sie ihn an. `~/.ssh/authorized_keys`
4. Sie sollten nun in der Lage sein, sich direkt mithilfe der privaten IP bei den Mitgliedsknoten über den Führungsknoten anzumelden.

   ```
   ssh <member private ip>
   ```

5. Deaktivieren Sie die strictHostKey Überprüfung und aktivieren Sie die Agentenweiterleitung auf dem Leader-Knoten, indem Sie der Datei ~/.ssh/config auf dem Leader-Knoten Folgendes hinzufügen:

   ```
   Host *
   ```

```
    ForwardAgent yes
Host *
    StrictHostKeyChecking no
```

6. Führen Sie auf Amazon Linux 2-Instances den folgenden Befehl auf dem Leader-Knoten aus, um die richtigen Berechtigungen für die Konfigurationsdatei bereitzustellen:

```
chmod 600 ~/.ssh/config
```

Erstellen einer Hosts-Datei

Erstellen Sie auf dem Führungsknoten eine Hosts-Datei, um die Knoten im Cluster zu identifizieren. Die Hosts-Datei muss für jeden Knoten im Cluster einen Eintrag aufweisen. Erstellen Sie eine Datei "~/hosts" und fügen Sie jeden Knoten mithilfe der privaten IP wie folgt hinzu:

```
localhost slots=8
<private ip of node 1> slots=8
<private ip of node 2> slots=8
```

NCCL-Tests

> **Note**
>
> Diese Tests wurden mit der EFA-Version 1.30.0 und dem OFI NCCL Plugin 1.7.4 ausgeführt.

Im Folgenden ist eine Teilmenge der von Nvidia bereitgestellten NCCL-Tests aufgeführt, mit denen sowohl Funktionalität als auch Leistung über mehrere Rechenknoten hinweg getestet werden können

Unterstützte Instanzen: P3dn, P4, P5

Funktionstests

NCCL-Nachrichtenübertragung, Test mit mehreren Knoten

"nccl_message_transfer" ist ein einfacher Test, um sicherzustellen, dass das NCCL OFI-Plugin wie erwartet funktioniert. Der Test überprüft die Funktionalität der Verbindungseinrichtungs- und Datentransfer-APIs von NCCL. Stellen Sie sicher, dass Sie den vollständigen Pfad zu mpirun verwenden, wie im Beispiel gezeigt, wenn Sie NCCL-Anwendungen mit EFA ausführen. Ändern Sie

die Parameter np und N basierend auf der Anzahl der Instances und GPUs in Ihrem Cluster. Weitere Informationen finden Sie in der OFI NCCL-Dokumentation.AWS

Der folgende nccl_message_transfer-Test gilt für eine generische CUDA xx.x-Version. Sie können die Befehle für jede verfügbare CUDA-Version in Ihrer Amazon EC2 EC2-Instance ausführen, indem Sie die CUDA-Version im Skript ersetzen.

```
$/opt/amazon/openmpi/bin/mpirun -n 2 -N 1 --hostfile hosts \
-x LD_LIBRARY_PATH=/usr/local/cuda-xx.x/efa/lib:/usr/local/cuda-xx.x/lib:/usr/
local/cuda-xx.x/lib64:/usr/local/cuda-xx.x:$LD_LIBRARY_PATH \
--mca btl tcp,self --mca btl_tcp_if_exclude lo,docker0 --bind-to none \
opt/aws-ofi-nccl/tests/nccl_message_transfer
```

Die Ausgabe sollte wie folgt aussehen. Sie können anhand der Ausgabe überprüfen, ob EFA als OFI-Anbieter verwendet wird.

```
INFO: Function: nccl_net_ofi_init Line: 1069: NET/OFI Selected Provider is efa (found 4
 nics)
INFO: Function: nccl_net_ofi_init Line: 1160: NET/OFI Using transport protocol SENDRECV
INFO: Function: configure_ep_inorder Line: 261: NET/OFI Setting
 FI_OPT_EFA_SENDRECV_IN_ORDER_ALIGNED_128_BYTES not supported.
INFO: Function: configure_nccl_proto Line: 227: NET/OFI Setting NCCL_PROTO to "simple"
INFO: Function: main Line: 86: NET/OFI Process rank 1 started. NCCLNet device used on
 ip-172-31-13-179 is AWS Libfabric.
INFO: Function: main Line: 91: NET/OFI Received 4 network devices
INFO: Function: main Line: 111: NET/OFI Network supports communication using CUDA
 buffers. Dev: 3
INFO: Function: main Line: 118: NET/OFI Server: Listening on dev 3
INFO: Function: main Line: 131: NET/OFI Send connection request to rank 1
INFO: Function: main Line: 173: NET/OFI Send connection request to rank 0
INFO: Function: main Line: 137: NET/OFI Server: Start accepting requests
INFO: Function: main Line: 141: NET/OFI Successfully accepted connection from rank 1
INFO: Function: main Line: 145: NET/OFI Send 8 requests to rank 1
INFO: Function: main Line: 179: NET/OFI Server: Start accepting requests
INFO: Function: main Line: 183: NET/OFI Successfully accepted connection from rank 0
INFO: Function: main Line: 187: NET/OFI Rank 1 posting 8 receive buffers
INFO: Function: main Line: 161: NET/OFI Successfully sent 8 requests to rank 1
INFO: Function: main Line: 251: NET/OFI Got completions for 8 requests for rank 0
INFO: Function: main Line: 251: NET/OFI Got completions for 8 requests for rank 1
```

Leistungstests

NCCL-Leistungstest mit mehreren Knoten auf P4D.24xlarge

Um die NCCL-Leistung mit EFA zu überprüfen, führen Sie den standardmäßigen NCCL-Leistungstest aus, der im offiziellen NCCL-Tests Repo verfügbar ist. Das DLAMI enthält diesen Test, der bereits für CUDA XX.X erstellt wurde. Sie können auf ähnliche Weise Ihr eigenes Skript mit EFA ausführen.

Wenn Sie Ihr eigenes Skript erstellen, beachten Sie dabei die folgenden Anweisungen:

- Verwenden Sie den vollständigen Pfad zu mpirun, wie im Beispiel gezeigt, während Sie NCCL-Anwendungen mit EFA ausführen.
- Ändern Sie die Parameter "np" und "N" basierend auf der Anzahl der Instances und GPUs in Ihrem Cluster.
- Fügen Sie das Flag NCCL_DEBUG=INFO hinzu und stellen Sie sicher, dass in den Protokollen die EFA-Nutzung als „Ausgewählter Anbieter ist EFA“ angegeben ist.
- Legen Sie den Speicherort des Trainingsprotokolls fest, der zur Validierung analysiert werden soll

```
TRAINING_LOG="testEFA_$(date +"%N").log"
```

Verwenden Sie den Befehl `watch nvidia-smi` auf einem der Mitgliedsknoten, um die GPU-Nutzung zu überwachen. Die folgenden `watch nvidia-smi` Befehle gelten für eine generische CUDA xx.x-Version und hängen vom Betriebssystem Ihrer Instanz ab. Sie können die Befehle für jede verfügbare CUDA-Version in Ihrer Amazon EC2 EC2-Instance ausführen, indem Sie die CUDA-Version im Skript ersetzen.

- Amazon Linux 2:

```
 $ /opt/amazon/openmpi/bin/mpirun -n 16 -N 8 \
-x NCCL_DEBUG=INFO -x --mca pml ^cm \
-x LD_LIBRARY_PATH=/usr/local/cuda-xx.x/efa/lib:/usr/local/cuda-xx.x/lib:/usr/
local/cuda-xx.x/lib64:/usr/local/cuda-xx.x:/opt/amazon/efa/lib64:/opt/amazon/openmpi/
lib64:$LD_LIBRARY_PATH \
--hostfile hosts --mca btl tcp,self --mca btl_tcp_if_exclude lo,docker0 --bind-to
 none \
/usr/local/cuda-xx.x/efa/test-cuda-xx.x/all_reduce_perf -x NCCL_PROTO=simple -b 8 -e
 1G -f 2 -g 1 -c 1 -n 100 | tee ${TRAINING_LOG}
```

- Ubuntu 20.04:

```
$ /opt/amazon/openmpi/bin/mpirun -n 16 -N 8 \
-x NCCL_DEBUG=INFO -x --mca pml ^cm \
-x LD_LIBRARY_PATH=/usr/local/cuda-xx.x/efa/lib:/usr/local/cuda-xx.x/lib:/usr/
local/cuda-xx.x/lib64:/usr/local/cuda-xx.x:/opt/amazon/efa/lib:/opt/amazon/openmpi/
lib:$LD_LIBRARY_PATH \
--hostfile hosts --mca btl tcp,self --mca btl_tcp_if_exclude lo,docker0 --bind-to
 none \
/usr/local/cuda-xx.x/efa/test-cuda-xx.x/all_reduce_perf -x NCCL_PROTO=simple-b 8 -e
 1G -f 2 -g 1 -c 1 -n 100 | tee ${TRAINING_LOG}
```

Die Ausgabe sollte folgendermaßen aussehen:

```
# nThread 1 nGpus 1 minBytes 8 maxBytes 1073741824 step: 2(factor) warmup iters: 5
 iters: 100 agg iters: 1 validation: 1 graph: 0
#
# Using devices
#  Rank  0 Group  0 Pid   9591 on ip-172-31-4-37 device  0 [0x10] NVIDIA A100-SXM4-40GB
#  Rank  1 Group  0 Pid   9592 on ip-172-31-4-37 device  1 [0x10] NVIDIA A100-SXM4-40GB
#  Rank  2 Group  0 Pid   9593 on ip-172-31-4-37 device  2 [0x20] NVIDIA A100-SXM4-40GB
#  Rank  3 Group  0 Pid   9594 on ip-172-31-4-37 device  3 [0x20] NVIDIA A100-SXM4-40GB
#  Rank  4 Group  0 Pid   9595 on ip-172-31-4-37 device  4 [0x90] NVIDIA A100-SXM4-40GB
#  Rank  5 Group  0 Pid   9596 on ip-172-31-4-37 device  5 [0x90] NVIDIA A100-SXM4-40GB
#  Rank  6 Group  0 Pid   9597 on ip-172-31-4-37 device  6 [0xa0] NVIDIA A100-SXM4-40GB
#  Rank  7 Group  0 Pid   9598 on ip-172-31-4-37 device  7 [0xa0] NVIDIA A100-SXM4-40GB
#  Rank  8 Group  0 Pid  10216 on ip-172-31-13-179 device  0 [0x10] NVIDIA A100-
SXM4-40GB
#  Rank  9 Group  0 Pid  10217 on ip-172-31-13-179 device  1 [0x10] NVIDIA A100-
SXM4-40GB
#  Rank 10 Group  0 Pid  10218 on ip-172-31-13-179 device  2 [0x20] NVIDIA A100-
SXM4-40GB
#  Rank 11 Group  0 Pid  10219 on ip-172-31-13-179 device  3 [0x20] NVIDIA A100-
SXM4-40GB
#  Rank 12 Group  0 Pid  10220 on ip-172-31-13-179 device  4 [0x90] NVIDIA A100-
SXM4-40GB
#  Rank 13 Group  0 Pid  10221 on ip-172-31-13-179 device  5 [0x90] NVIDIA A100-
SXM4-40GB
#  Rank 14 Group  0 Pid  10222 on ip-172-31-13-179 device  6 [0xa0] NVIDIA A100-
SXM4-40GB
#  Rank 15 Group  0 Pid  10223 on ip-172-31-13-179 device  7 [0xa0] NVIDIA A100-
SXM4-40GB
ip-172-31-4-37:9591:9591 [0] NCCL INFO Bootstrap : Using ens32:172.31.4.37
```

```
ip-172-31-4-37:9591:9591 [0] NCCL INFO NET/Plugin: Failed to find ncclCollNetPlugin_v6
 symbol.
ip-172-31-4-37:9591:9591 [0] NCCL INFO NET/Plugin: Failed to find ncclCollNetPlugin
 symbol (v4 or v5).
ip-172-31-4-37:9591:9591 [0] NCCL INFO cudaDriverVersion 12020
NCCL version 2.18.5+cuda12.2
...
ip-172-31-4-37:9024:9062 [6] NCCL INFO NET/OFI Initializing aws-ofi-nccl 1.7.4-aws
ip-172-31-4-37:9020:9063 [2] NCCL INFO NET/OFI Using CUDA runtime version 11070
ip-172-31-4-37:9020:9063 [2] NCCL INFO NET/OFI Configuring AWS-specific options
ip-172-31-4-37:9024:9062 [6] NCCL INFO NET/OFI Using CUDA runtime version 11070
ip-172-31-4-37:9024:9062 [6] NCCL INFO NET/OFI Configuring AWS-specific options
ip-172-31-4-37:9024:9062 [6] NCCL INFO NET/OFI Setting provider_filter to efa
ip-172-31-4-37:9024:9062 [6] NCCL INFO NET/OFI Setting FI_EFA_FORK_SAFE environment
 variable to 1
ip-172-31-4-37:9024:9062 [6] NCCL INFO NET/OFI Disabling NVLS support due to NCCL
 version 21602
ip-172-31-4-37:9020:9063 [2] NCCL INFO NET/OFI Setting provider_filter to efa
ip-172-31-4-37:9020:9063 [2] NCCL INFO NET/OFI Setting FI_EFA_FORK_SAFE environment
 variable to 1
ip-172-31-4-37:9020:9063 [2] NCCL INFO NET/OFI Disabling NVLS support due to NCCL
 version 21602
ip-172-31-4-37:9020:9063 [2] NCCL INFO NET/OFI Running on p4d.24xlarge platform,
 Setting NCCL_TOPO_FILE environment variable to /opt/aws-ofi-nccl/share/aws-ofi-nccl/
xml/p4d-24xl-topo.xml
...
------------------------------some output truncated-----------------------------------
#                                                              out-of-place
           in-place
#       size         count      type   redop    root     time   algbw   busbw #wrong
  time   algbw   busbw #wrong
#        (B)    (elements)                               (us)  (GB/s)  (GB/s)
  (us)  (GB/s)  (GB/s)
           0             0     float     sum      -1    11.02    0.00    0.00      0
 11.04    0.00    0.00      0
           0             0     float     sum      -1    11.01    0.00    0.00      0
 11.00    0.00    0.00      0
           0             0     float     sum      -1    11.02    0.00    0.00      0
 11.02    0.00    0.00      0
           0             0     float     sum      -1    11.01    0.00    0.00      0
 11.00    0.00    0.00      0
           0             0     float     sum      -1    11.02    0.00    0.00      0
 11.02    0.00    0.00      0
```

```
         256             4     float     sum      -1    632.7    0.00    0.00       0
628.2    0.00    0.00       0
         512             8     float     sum      -1    627.4    0.00    0.00       0
629.6    0.00    0.00       0
        1024            16     float     sum      -1    632.2    0.00    0.00       0
631.7    0.00    0.00       0
        2048            32     float     sum      -1    631.0    0.00    0.00       0
634.2    0.00    0.00       0
        4096            64     float     sum      -1    623.3    0.01    0.01       0
633.6    0.01    0.01       0
        8192           128     float     sum      -1    635.1    0.01    0.01       0
633.5    0.01    0.01       0
       16384           256     float     sum      -1    634.8    0.03    0.02       0
637.0    0.03    0.02       0
       32768           512     float     sum      -1    647.9    0.05    0.05       0
636.8    0.05    0.05       0
       65536          1024     float     sum      -1    658.9    0.10    0.09       0
667.0    0.10    0.09       0
      131072          2048     float     sum      -1    671.9    0.20    0.18       0
662.9    0.20    0.19       0
      262144          4096     float     sum      -1    692.1    0.38    0.36       0
685.1    0.38    0.36       0
      524288          8192     float     sum      -1    715.3    0.73    0.69       0
696.6    0.75    0.71       0
     1048576         16384     float     sum      -1    734.6    1.43    1.34       0
729.2    1.44    1.35       0
     2097152         32768     float     sum      -1    785.9    2.67    2.50       0
794.5    2.64    2.47       0
     4194304         65536     float     sum      -1    837.2    5.01    4.70       0
837.6    5.01    4.69       0
     8388608        131072     float     sum      -1    929.2    9.03    8.46       0
931.4    9.01    8.44       0
    16777216        262144     float     sum      -1   1773.6    9.46    8.87       0
1772.8    9.46    8.87       0
    33554432        524288     float     sum      -1   2110.2   15.90   14.91       0
2116.1   15.86   14.87       0
    67108864       1048576     float     sum      -1   2650.9   25.32   23.73       0
2658.1   25.25   23.67       0
   134217728       2097152     float     sum      -1   3943.1   34.04   31.91       0
3945.9   34.01   31.89       0
   268435456       4194304     float     sum      -1   7216.5   37.20   34.87       0
7178.6   37.39   35.06       0
   536870912       8388608     float     sum      -1    13680   39.24   36.79       0
13676   39.26   36.80       0
```

```
[  1073741824      16777216    float     sum      -1    25645   41.87   39.25      0
 25497   42.11   39.48      0 ] <- Used For Benchmark
...
# Out of bounds values : 0 OK
# Avg bus bandwidth    : 7.46044
```

Validierungstests

Um zu überprüfen, ob die EFA-Tests ein gültiges Ergebnis geliefert haben, verwenden Sie bitte die folgenden Tests zur Bestätigung:

- Rufen Sie den Instanztyp mithilfe der EC2-Instanz-Metadaten ab:

```
TOKEN=$(curl -X PUT "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-metadata-
token-ttl-seconds: 21600")
INSTANCE_TYPE=$(curl -H "X-aws-ec2-metadata-token: $TOKEN" -v http://169.254.169.254/
latest/meta-data/instance-type)
```

- Ausführen des Leistungstests
- Stellen Sie die folgenden Parameter ein

```
CUDA_VERSION
CUDA_RUNTIME_VERSION
NCCL_VERSION
```

- Validieren Sie die Ergebnisse wie gezeigt:

```
RETURN_VAL=`echo $?`
if [ ${RETURN_VAL} -eq 0 ]; then

    # Information on how the version come from logs
    #
    # ip-172-31-27-205:6427:6427 [0] NCCL INFO cudaDriverVersion 12020
    # NCCL version 2.16.2+cuda11.8
    # ip-172-31-27-205:6427:6820 [0] NCCL INFO NET/OFI Initializing aws-ofi-nccl
 1.7.1-aws
    # ip-172-31-27-205:6427:6820 [0] NCCL INFO NET/OFI Using CUDA runtime version
 11060

    # cudaDriverVersion 12020  --> This is max supported cuda version by nvidia
 driver
```

```
    # NCCL version 2.16.2+cuda11.8 --> This is NCCL version compiled with cuda
 version
    # Using CUDA runtime version 11060 --> This is selected cuda version

    # Validation of logs
    grep "NET/OFI Using CUDA runtime version ${CUDA_RUNTIME_VERSION}" ${TRAINING_LOG}
 || { echo "Runtime cuda text not found"; exit 1; }
    grep "NET/OFI Initializing aws-ofi-nccl" ${TRAINING_LOG} || { echo "aws-ofi-nccl
 is not working, please check if it is installed correctly"; exit 1; }
    grep "NET/OFI Configuring AWS-specific options" ${TRAINING_LOG} || { echo "AWS-
specific options text not found"; exit 1; }
    grep "Using network AWS Libfabric" ${TRAINING_LOG} || { echo "AWS Libfabric text
 not found"; exit 1; }
    grep "busbw" ${TRAINING_LOG} || { echo "busbw text not found"; exit 1; }
    grep "Avg bus bandwidth " ${TRAINING_LOG} || { echo "Avg bus bandwidth text not
 found"; exit 1; }
    grep "NCCL version $NCCL_VERSION" ${TRAINING_LOG} || { echo "Text not found: NCCL
 version $NCCL_VERSION"; exit 1; }

    if [[ ${INSTANCE_TYPE} == "p4d.24xlarge" ]]; then
        grep "NET/AWS Libfabric/0/GDRDMA" ${TRAINING_LOG} || { echo "Text not found:
 NET/AWS Libfabric/0/GDRDMA"; exit 1; }
        grep "NET/OFI Selected Provider is efa (found 4 nics)" ${TRAINING_LOG} ||
 { echo "Selected Provider is efa text not found"; exit 1; }
        grep "aws-ofi-nccl/xml/p4d-24xl-topo.xml" ${TRAINING_LOG} || { echo "Topology
 file not found"; exit 1; }
    elif [[ ${INSTANCE_TYPE} == "p4de.24xlarge" ]]; then
        grep "NET/AWS Libfabric/0/GDRDMA" ${TRAINING_LOG} || { echo "Avg bus
 bandwidth text not found"; exit 1; }
        grep "NET/OFI Selected Provider is efa (found 4 nics)" ${TRAINING_LOG} ||
 { echo "Avg bus bandwidth text not found"; exit 1; }
        grep "aws-ofi-nccl/xml/p4de-24xl-topo.xml" ${TRAINING_LOG} || { echo
 "Topology file not found"; exit 1; }
    elif [[ ${INSTANCE_TYPE} == "p5.48xlarge" ]]; then
        grep "NET/AWS Libfabric/0/GDRDMA" ${TRAINING_LOG} || { echo "Avg bus
 bandwidth text not found"; exit 1; }
        grep "NET/OFI Selected Provider is efa (found 32 nics)" ${TRAINING_LOG} ||
 { echo "Avg bus bandwidth text not found"; exit 1; }
        grep "aws-ofi-nccl/xml/p5.48xl-topo.xml" ${TRAINING_LOG} || { echo "Topology
 file not found"; exit 1; }
    elif [[ ${INSTANCE_TYPE} == "p3dn.24xlarge" ]]; then
        grep "NET/OFI Selected Provider is efa (found 4 nics)" ${TRAINING_LOG} ||
 { echo "Selected Provider is efa text not found"; exit 1; }
    fi
```

```
    echo "*************************** check_efa_nccl_all_reduce passed for cuda
 version ${CUDA_VERSION} ***************************"
else
    echo "*************************** check_efa_nccl_all_reduce failed for cuda
 version ${CUDA_VERSION} ***************************"
fi
```

- Um auf die Benchmark-Daten zuzugreifen, können wir die letzte Zeile der Tabellenausgabe aus dem All_Reduce-Test mit mehreren Knoten analysieren:

```
benchmark=$(sudo cat ${TRAINING_LOG} | grep '1073741824' | tail -n1 | awk -F " "
 '{{print $12}}' | sed 's/ //' | sed  's/  5e-07//')
if [[ -z "${benchmark}" ]]; then
  echo "benchmark variable is empty"
  exit 1
fi

echo "Benchmark throughput: ${benchmark}"
```

GPU-Überwachung und -Optimierung

Der folgende Abschnitt führt Sie durch GPU-Optimierungs und -Überwachungsoptionen. Dieser Abschnitt ist wie ein typischer Workflow mit Überwachung, Beaufsichtigung, Vorverarbeitung und Schulung aufgebaut.

- Überwachen
  - Überwachen Sie GPUs mit CloudWatch
- Optimierung
  - Vorverarbeitung
  - Training

Überwachen

Auf Ihrem DLAMI sind mehrere GPU-Überwachungstools vorinstalliert. Diese Anleitung erwähnt auch Tools, die heruntergeladen und installiert werden können.

- Überwachen Sie GPUs mit CloudWatch- ein vorinstalliertes Hilfsprogramm, das Statistiken zur GPU-Nutzung an Amazon CloudWatch meldet.
- nvidia-CLI – ein Dienstprogramm zur Überwachung der allgemeinen GPU-Rechenleistungs- und -Speichernutzung. Dies ist auf Ihrem AWS Deep Learning AMI (DLAMI) vorinstalliert.
- NVML C-Bibliothek - eine auf C basierende API für den direkten Zugriff auf GPU-Überwachungs- und Verwaltungsfunktionen. Dies wird von der nvidia-smi-CLI intern verwendet und ist auf Ihrem DLAMI vorinstalliert. Dazu gehören weiterhin Python- und Perl-Anbindungen zur Unterstützung der Bereitstellung in diesen Sprachen. Das auf Ihrem DLAMI vorinstallierte Hilfsprogramm gpumon.py verwendet das Paket pynvml von. nvidia-ml-py
- NVIDIA DCGM - Ein Cluster-Management-Tool. Besuchen Sie die Entwicklerseite, um zu erfahren, wie Sie dieses Tool installieren und konfigurieren.

> **Tip**
>
> Im Entwickler-Blog von NVIDIA finden Sie die neuesten Informationen zur Verwendung der auf Ihrem DLAMI installierten CUDA-Tools:
>
> - Überwachung TensorCore der Auslastung mit Nsight IDE und nvprof.

Überwachen Sie GPUs mit CloudWatch

Wenn Sie Ihre DLAMI mit einem GPU verwenden, stellen Sie möglicherweise fest, dass Sie auf der Suche nach Möglichkeiten zur Nachverfolgung der Nutzung während der Schulung oder Inferenz sind. Dies kann nützlich sein, um Ihre Datenpipeline zu optimieren und Ihr Deep Learning-Netzwerk zu verfeinern.

Es gibt zwei Möglichkeiten, GPU-Metriken zu konfigurieren mit CloudWatch:

- Metriken mit dem AWS CloudWatch Agenten konfigurieren (empfohlen)
- Konfigurieren Sie Metriken mit dem vorinstallierten Skript `gpumon.py`

Metriken mit dem AWS CloudWatch Agenten konfigurieren (empfohlen)

Integrieren Sie Ihr DLAMI in den Unified CloudWatch Agent, um GPU-Metriken zu konfigurieren und die Nutzung von GPU-Koprozessen in Amazon EC2 EC2-beschleunigten Instances zu überwachen.

Es gibt vier Möglichkeiten, GPU-Metriken mit Ihrem DLAMI zu konfigurieren:

- Konfigurieren Sie minimale GPU-Metriken
- Konfigurieren Sie partielle GPU-Metriken
- Konfigurieren Sie alle verfügbaren GPU-Metriken
- Konfigurieren Sie benutzerdefinierte GPU-Metriken

Informationen zu Updates und Sicherheitspatches finden Sie unter Sicherheitspatches für den Agenten AWS CloudWatch

Voraussetzungen

Zu Beginn müssen Sie IAM-Berechtigungen für Amazon EC2 EC2-Instances konfigurieren, an die Ihre Instance Metriken weiterleiten kann. CloudWatch Ausführliche Schritte finden Sie unter IAM-Rollen und -Benutzer für die Verwendung mit dem Agenten erstellen. CloudWatch

Konfigurieren Sie minimale GPU-Metriken

Konfigurieren Sie minimale GPU-Metriken mithilfe des `dlami-cloudwatch-agent@minimal` `systemd` Dienstes. Dieser Dienst konfiguriert die folgenden Metriken:

- `utilization_gpu`
- `utilization_memory`

Sie finden den `systemd` Dienst für minimale vorkonfigurierte GPU-Metriken an der folgenden Stelle:

```
/opt/aws/amazon-cloudwatch-agent/etc/dlami-amazon-cloudwatch-agent-minimal.json
```

Aktivieren und starten Sie den `systemd` Dienst mit den folgenden Befehlen:

```
sudo systemctl enable dlami-cloudwatch-agent@minimal
sudo systemctl start dlami-cloudwatch-agent@minimal
```

Konfigurieren Sie partielle GPU-Metriken

Konfigurieren Sie partielle GPU-Metriken mithilfe des `dlami-cloudwatch-agent@partial` `systemd` Dienstes. Dieser Dienst konfiguriert die folgenden Metriken:

- `utilization_gpu`
- `utilization_memory`

- memory_total
- memory_used
- memory_free

Sie finden den systemd Dienst für teilweise vorkonfigurierte GPU-Metriken an der folgenden Stelle:

```
/opt/aws/amazon-cloudwatch-agent/etc/dlami-amazon-cloudwatch-agent-partial.json
```

Aktivieren und starten Sie den systemd Dienst mit den folgenden Befehlen:

```
sudo systemctl enable dlami-cloudwatch-agent@partial
sudo systemctl start dlami-cloudwatch-agent@partial
```

Konfigurieren Sie alle verfügbaren GPU-Metriken

Konfigurieren Sie alle verfügbaren GPU-Metriken mithilfe des dlami-cloudwatch-agent@all systemd Dienstes. Dieser Dienst konfiguriert die folgenden Metriken:

- utilization_gpu
- utilization_memory
- memory_total
- memory_used
- memory_free
- temperature_gpu
- power_draw
- fan_speed
- pcie_link_gen_current
- pcie_link_width_current
- encoder_stats_session_count
- encoder_stats_average_fps
- encoder_stats_average_latency
- clocks_current_graphics
- clocks_current_sm
- clocks_current_memory

- `clocks_current_video`

Sie finden den `systemd` Dienst für alle verfügbaren vorkonfigurierten GPU-Metriken an der folgenden Stelle:

```
/opt/aws/amazon-cloudwatch-agent/etc/dlami-amazon-cloudwatch-agent-all.json
```

Aktivieren und starten Sie den `systemd` Dienst mit den folgenden Befehlen:

```
sudo systemctl enable dlami-cloudwatch-agent@all
sudo systemctl start dlami-cloudwatch-agent@all
```

Konfigurieren Sie benutzerdefinierte GPU-Metriken

Wenn die vorkonfigurierten Metriken Ihren Anforderungen nicht entsprechen, können Sie eine benutzerdefinierte CloudWatch Agentenkonfigurationsdatei erstellen.

Erstellen Sie eine benutzerdefinierte Konfigurationsdatei

Informationen zum Erstellen einer benutzerdefinierten Konfigurationsdatei finden Sie in den detaillierten Schritten unter Manuelles Erstellen oder Bearbeiten der CloudWatch Agentenkonfigurationsdatei.

Gehen Sie für dieses Beispiel davon aus, dass sich die Schemadefinition unter befindet`/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json`.

Konfigurieren Sie Metriken mit Ihrer benutzerdefinierten Datei

Führen Sie den folgenden Befehl aus, um den CloudWatch Agenten entsprechend Ihrer benutzerdefinierten Datei zu konfigurieren:

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl \
-a fetch-config -m ec2 -s -c \
file:/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json
```

Sicherheitspatches für den Agenten AWS CloudWatch

Neu veröffentlichte DLAMIs sind mit den neuesten verfügbaren AWS CloudWatch Agenten-Sicherheitspatches konfiguriert. In den folgenden Abschnitten erfahren Sie, wie Sie Ihr aktuelles DLAMI je nach Betriebssystem Ihrer Wahl mit den neuesten Sicherheitspatches aktualisieren können.

Amazon Linux 2

Wird verwendet`yum`, um die neuesten AWS CloudWatch Agenten-Sicherheitspatches für ein Amazon Linux 2-DLAMI zu erhalten.

```
sudo yum update
```

Ubuntu

Um die neuesten AWS CloudWatch Sicherheitspatches für ein DLAMI mit Ubuntu zu erhalten, muss der AWS CloudWatch Agent über einen Amazon S3 S3-Download-Link neu installiert werden.

```
wget https://s3.region.amazonaws.com/amazoncloudwatch-agent-region/ubuntu/arm64/latest/
amazon-cloudwatch-agent.deb
```

Weitere Informationen zur Installation des AWS CloudWatch Agenten mithilfe von Amazon S3 S3-Download-Links finden Sie unter Installation und Ausführung des CloudWatch Agenten auf Ihren Servern.

Konfigurieren Sie Metriken mit dem vorinstallierten Skript **`gpumon.py`**

Das Dienstprogramm gpumon.py ist auf Ihrem DLAMI vorinstalliert. Es lässt sich in die Nutzung pro GPU integrieren CloudWatch und unterstützt deren Überwachung: GPU-Speicher, GPU-Temperatur und GPU-Leistung. Das Skript sendet die überwachten Daten regelmäßig an CloudWatch. Sie können die Granularität für die zu sendenden Daten konfigurieren, CloudWatch indem Sie einige Einstellungen im Skript ändern. Bevor Sie das Skript starten, müssen Sie jedoch einrichten, um die Metriken CloudWatch zu empfangen.

Wie richte ich die GPU-Überwachung ein und führe sie aus CloudWatch

1. Erstellen Sie einen IAM-Benutzer oder ändern Sie einen vorhandenen Benutzer, um eine Richtlinie für die Veröffentlichung der Metrik festzulegen CloudWatch. Wenn Sie einen neuen Benutzer erstellen, notieren Sie sich die Anmeldeinformationen, da Sie diese im nächsten Schritt benötigen.

   Die IAM-Richtlinie, nach der gesucht werden soll, lautet „cloudwatch:“. PutMetricData Die Richtlinie, die hinzugefügt wird, lautet wie folgt:

   ```
   {
      "Version": "2012-10-17",
   ```

```
   "Statement": [
        {
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
   ]
}
```

> **Tip**
>
> Weitere Informationen zum Erstellen eines IAM-Benutzers und zum Hinzufügen von Richtlinien für CloudWatch finden Sie in der Dokumentation. CloudWatch

2. Führen Sie auf Ihrem DLAMI AWS configure aus und geben Sie die IAM-Benutzeranmeldedaten an.

```
$ aws configure
```

3. Möglicherweise müssen Sie einige Änderungen am gpumon-Dienstprogramm vornehmen, bevor Sie es ausführen können. Sie finden das Gpumon-Hilfsprogramm und die README-Datei an dem im folgenden Codeblock definierten Speicherort. Weitere Informationen zum `gpumon.py` Skript finden Sie im Amazon S3 S3-Speicherort des Skripts.

```
Folder: ~/tools/GPUCloudWatchMonitor
Files:  ~/tools/GPUCloudWatchMonitor/gpumon.py
       ~/tools/GPUCloudWatchMonitor/README
```

   Optionen:

   - Ändern Sie die Region in gpumon.py, wenn sich Ihre Instance NICHT in us-east-1 befindet.
   - Ändern Sie andere Parameter wie den CloudWatch `namespace` oder den Berichtszeitraum mit`store_reso`.

4. Derzeit unterstützt das Skript nur Python 3. Aktivieren Sie die Python-3-Umgebung Ihres bevorzugten Frameworks oder aktivieren Sie die allgemeine Python-3-Umgebung von DLAMI.

```
$ source activate python3
```

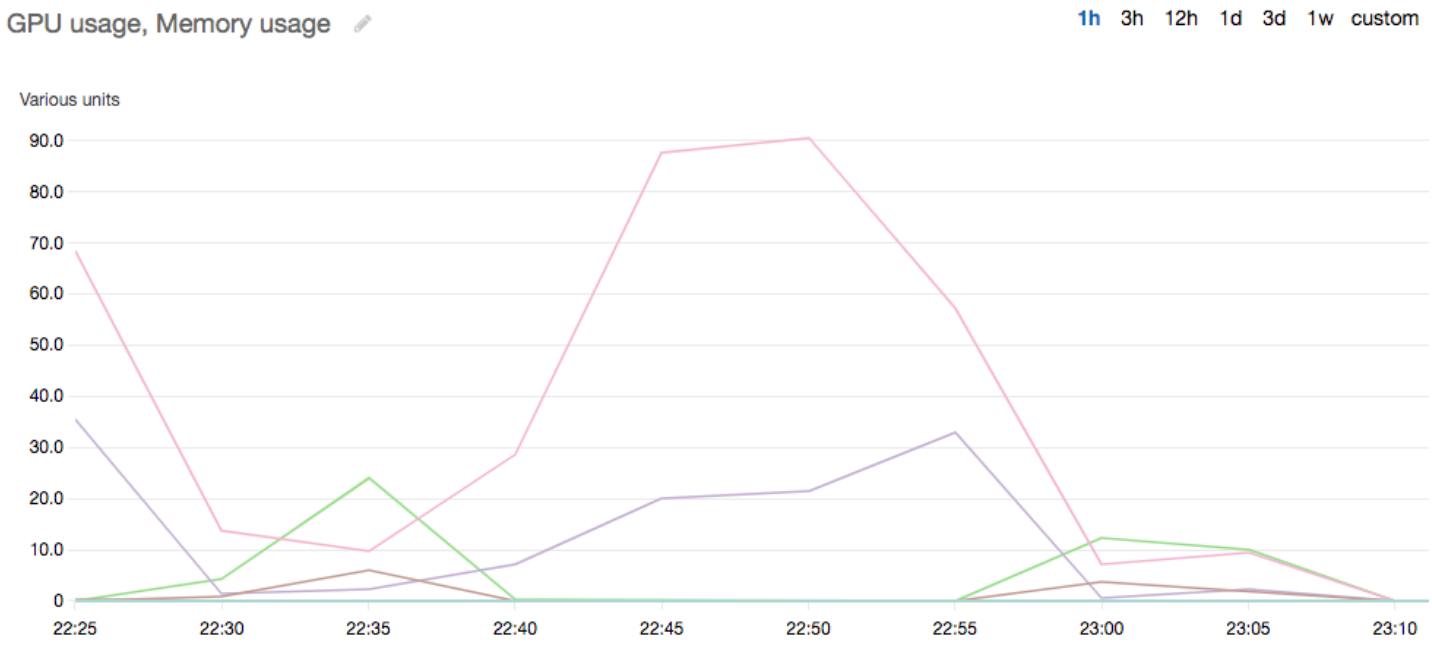
5. Führen Sie das gpumon-Dienstprogramm im Hintergrund aus.

```
(python3)$ python gpumon.py &
```

6. Öffnen Sie in Ihrem Browser https://console.aws.amazon.com/cloudwatch/ und wählen Sie Metrik aus. Es wird einen Namespace 'DeepLearningTrain' haben.

> **Tip**
>
> Sie können den Namespace durch Modifizierung von gpumon.py ändern. Sie können auch das Berichtsintervall durch Anpassung von `store_reso` ändern.

Im Folgenden finden Sie ein CloudWatch Beispieldiagramm, das über einen Lauf von gpumon.py berichtet, der einen Trainingsjob auf der p2.8xlarge-Instance überwacht.



Möglicherweise sind diese weiteren Themen zur GPU-Überwachung und -Optimierung für Sie interessant:

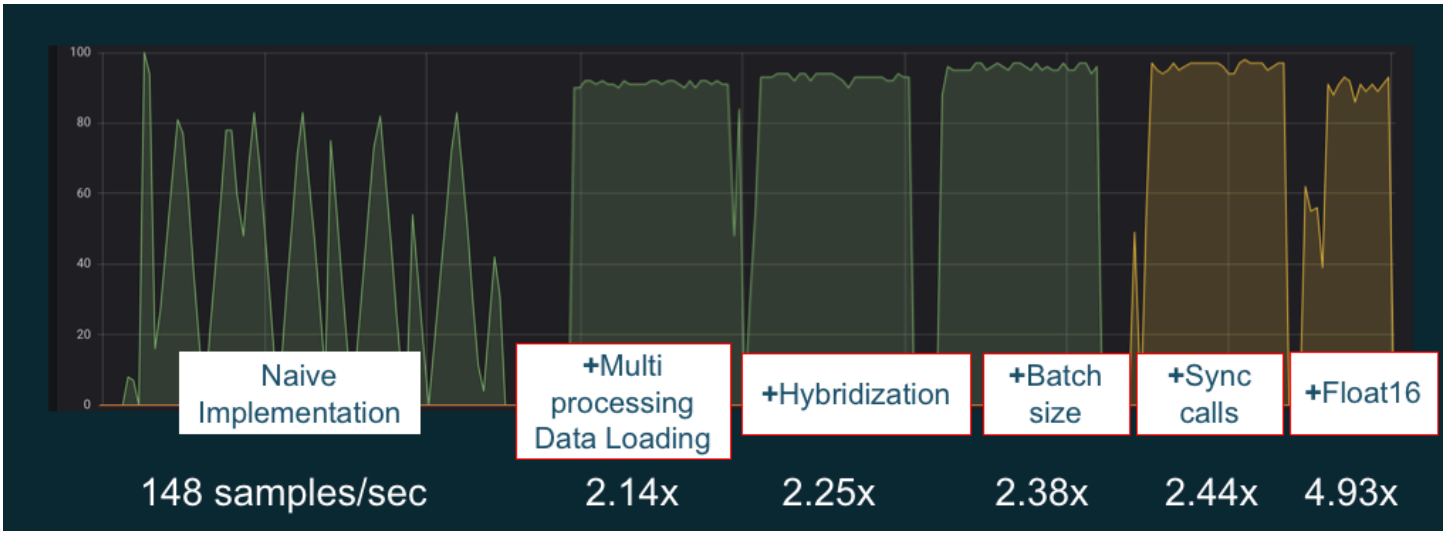
- Überwachen
  - Überwachen Sie GPUs mit CloudWatch
- Optimierung
  - Vorverarbeitung

- Training

Optimierung

Um Ihre GPUs vollständig nutzen zu können, können Sie Ihre Datenpipeline optimieren und Ihr Deep Learning-Netzwerk feineinstellen. Wie das folgende Diagramm zeigt, nutzt eine „naive“ oder einfache Implementierung eines neuronalen Netzwerks die GPU möglicherweise inkonsistent und nicht mit ihrem vollen Potenzial. Wenn Sie Ihre Vorverarbeitung und das Laden der Daten optimieren, können Sie den Engpass von CPU zu GPU reduzieren. Sie können das neuronale Netzwerk selbst anpassen, indem Sie Hybridisierung (wenn vom Framework unterstützt) verwenden, die Stapelgröße anpassen sowie Aufrufe synchronisieren. Sie können in den meisten Frameworks auch Schulungen mit mehreren Präzisionen verwenden, was dramatische Verbesserungen des Durchsatzes mit sich bringen kann.

Das folgende Diagramm zeigt die kumulativen Leistungssteigerungen bei der Anwendung verschiedener Optimierungen. Ihre Ergebnisse hängen von den verarbeiteten Daten und dem optimierten Netzwerk ab.



Beispiel für GPU-Leistungsoptimierungen. Quelle des Diagramms: Performance Tricks with MXNet Gluon

In den folgenden Anleitungen werden Optionen vorgestellt, die mit Ihrem DLAMI funktionieren und Ihnen helfen, die GPU-Leistung zu steigern.

Themen

- Vorverarbeitung

- Training

Vorverarbeitung

Die Vorverarbeitung von Daten durch Umwandlungen oder Erweiterungen ist oft ein CPU-gebundener Prozess, was zu einem Engpass in Ihrer allgemeinen Pipeline führen kann. Frameworks verfügen über integrierte Operatoren für die Abbildverarbeitung, DALI (Data Augmentation Library) zeigt jedoch eine verbesserte Leistung gegenüber den in Frameworks integrierten Optionen.

- NVIDIA Data Augmentation Library (DALI): DALI übergibt die Datenerweiterung an die GPU. Es ist nicht auf dem DLAMI vorinstalliert, aber Sie können darauf zugreifen, indem Sie es installieren oder einen unterstützten Framework-Container auf Ihrem DLAMI oder einer anderen Amazon Elastic Compute Cloud-Instanz laden. Einzelheiten finden Sie auf der DALI-Projektseite auf der NVIDIA-Website. Ein Anwendungsbeispiel und zum Herunterladen von Codebeispielen finden Sie im Beispiel Preprocessing Training Performance. SageMaker
- nvJPEG: eine GPU-beschleunigte JPEG-Decoder-Bibliothek für C-Programmierer. Sie unterstützt die Decodierung einzelner Abbilder oder Stapel sowie Transformationsoperationen, die für Deep Learning verbreitet sind. nvJPEG ist mit DALI integriert. Alternativ können Sie es von der nvjpeg-Seite der NVIDIA-Website herunterladen und separat verwenden.

Möglicherweise sind diese weiteren Themen zur GPU-Überwachung und -Optimierung für Sie interessant:

- Überwachen
  - Überwachen Sie GPUs mit CloudWatch
- Optimierung
  - Vorverarbeitung
  - Training

Training

Mit Schulungen mit gemischter Präzision können Sie größere Netzwerke mit derselben Speichergröße bereitstellen oder die Speichernutzung gegenüber Ihrem Netzwerk mit einzelner oder doppelter Präzision reduzieren - die Rechenleistung wird dadurch sicher gesteigert. Dazu kommt der Vorteil kleinerer und schnellerer Datenübertragungen, ein wichtiger Faktor für verteilte Schulungen mit mehreren Knoten. Um die Schulung mit gemischter Präzision nutzen zu können, müssen Sie das

Data Casting und die Verlustskalierung anpassen. Nachfolgend finden Sie Anleitungen dazu, wie Sie dies für Frameworks tun können, die gemischte Präzisionen unterstützen.

- NVIDIA Deep Learning SDK — Dokumente auf der NVIDIA-Website, die die Mixed-Precision-Implementierung für MXNet beschreiben, und PyTorch. TensorFlow

> **Tip**
>
> Konsultieren Sie die Website für das von Ihnen gewählte Framework und suchen Sie nach „gemischte Präzision“ oder „fp16“ für die jeweils neuesten Optimierungstechniken. Hier finden Sie einige Anleitungen für gemischte Präzisionen, die möglicherweise nützlich für Sie sind:
>
> - Training mit gemischter Präzision mit TensorFlow (Video) — auf der NVIDIA-Blogseite.
> - Mixed-Precision-Schulung unter Verwendung von float16 mit MXNet: Artikel mit häufig gestellten Fragen auf der MXNet-Website.
> - NVIDIA Apex: ein Tool für einfaches Training mit gemischter Präzision mit PyTorch — einem Blogartikel auf der NVIDIA-Website.

Möglicherweise sind diese weiteren Themen zur GPU-Überwachung und -Optimierung für Sie interessant:

- Überwachen
  - Überwachen Sie GPUs mit CloudWatch
- Optimierung
  - Vorverarbeitung
  - Training

Der AWS Inferentia-Chip mit DLAMI

AWS Inferentia ist ein maßgeschneiderter Chip für maschinelles Lernen, der speziell für leistungsstarke AWS Inferenzvorhersagen entwickelt wurde. Um den Chip zu verwenden, richten Sie eine Amazon Elastic Compute Cloud-Instanz ein und verwenden Sie das AWS Neuron Software Development Kit (SDK), um den Inferentia-Chip aufzurufen. Um Kunden das beste Inferentia-Erlebnis zu bieten, wurde Neuron in das AWS Deep Learning AMI (DLAMI) integriert.

Die folgenden Themen zeigen Ihnen, wie Sie mit der Verwendung von Inferentia mit dem DLAMI beginnen können.

Inhalt

Starten einer DLAMI-Instanz mit Neuron AWS

Das neueste DLAMI ist sofort mit AWS Inferentia einsatzbereit und wird mit dem AWS Neuron API-Paket geliefert. Informationen zum Starten einer DLAMI-Instanz finden Sie unter Starten und Konfigurieren einer DLAMI-Instanz. Nachdem Sie ein DLAMI haben, gehen Sie wie folgt vor, um sicherzustellen, dass Ihr AWS Inferentia-Chip und Ihre AWS Neuron-Ressourcen aktiv sind.

Inhalt

Verifizieren der Instance

Bevor Sie Ihre Instance verwenden, stellen Sie sicher, dass sie ordnungsgemäß eingerichtet und mit Neuron konfiguriert ist.

Identifizieren von AWS Inferentia-Geräten

Verwenden Sie den folgenden Befehl, um die Anzahl der Inferentia-Geräte auf Ihrer Instance zu ermitteln:

```
neuron-ls
```

Wenn an Ihre Instance Inferentia-Geräte angeschlossen sind, sieht Ihre Ausgabe in etwa wie folgt aus:

```
+--------+--------+--------+-----------+--------------+
```

```
| NEURON | NEURON | NEURON | CONNECTED |     PCI      |
| DEVICE | CORES  | MEMORY |  DEVICES  |     BDF      |
+--------+--------+--------+-----------+--------------+
| 0      | 4      | 8 GB   | 1         | 0000:00:1c.0 |
| 1      | 4      | 8 GB   | 2, 0      | 0000:00:1d.0 |
| 2      | 4      | 8 GB   | 3, 1      | 0000:00:1e.0 |
| 3      | 4      | 8 GB   | 2         | 0000:00:1f.0 |
+--------+--------+--------+-----------+--------------+
```

Die bereitgestellte Ausgabe stammt aus einer INF1.6XLarge-Instance und umfasst die folgenden Spalten:

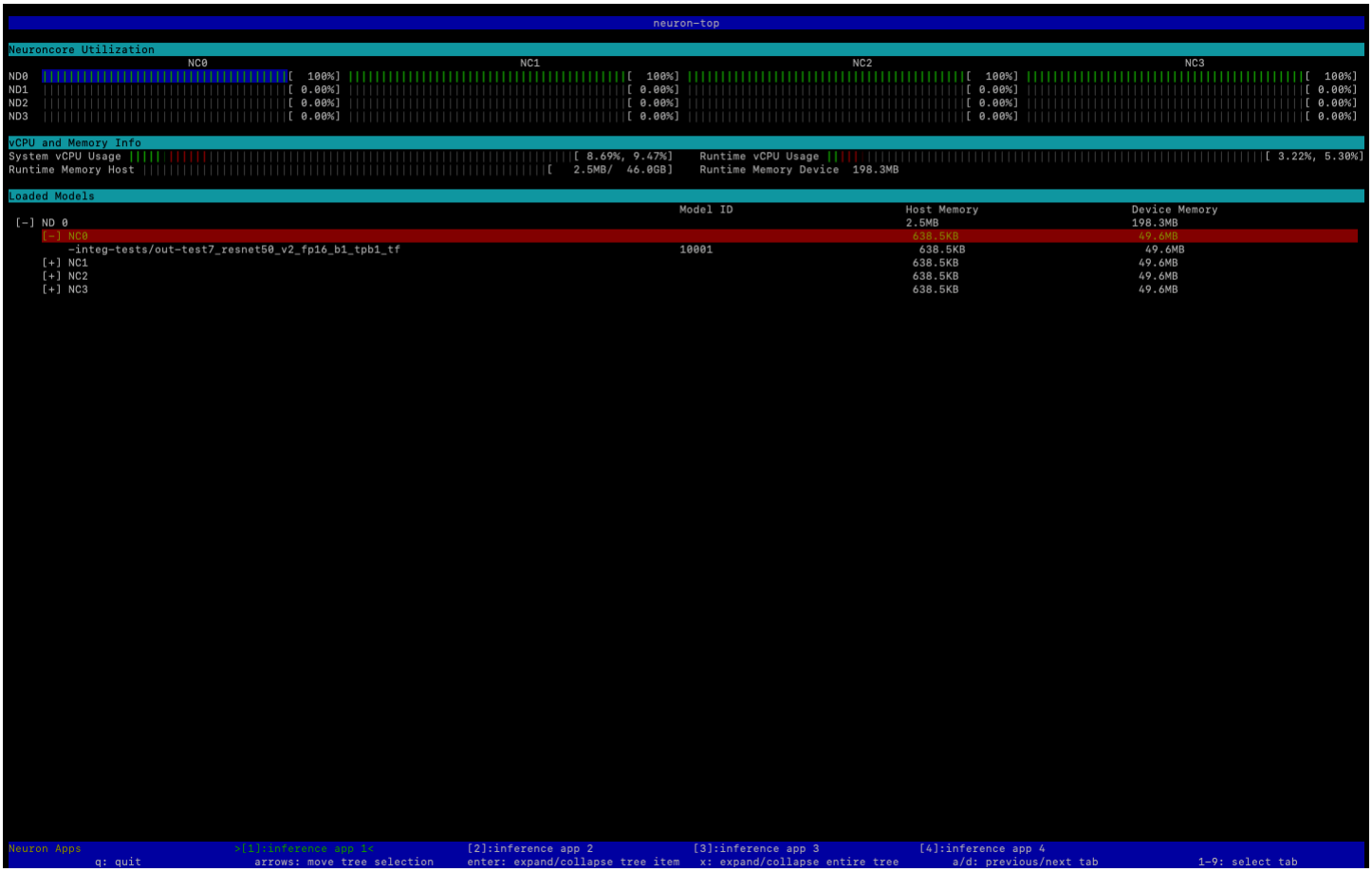
- NEURON DEVICE: Die logische ID, die dem zugewiesen ist. NeuronDevice Diese ID wird verwendet, wenn mehrere Laufzeiten für die Verwendung verschiedener Laufzeiten konfiguriert werden. NeuronDevices
- NEURON-KERNE: Die Anzahl der NeuronCores vorhandenen Kerne in der. NeuronDevice
- NEURONENSPEICHER: Die Menge an DRAM-Speicher im. NeuronDevice
- ANGESCHLOSSENE GERÄTE: Andere, die NeuronDevices mit dem verbunden sind. NeuronDevice
- PCI BDF: Die PCI-Bus-Gerätefunktions-ID (BDF) des. NeuronDevice

Anzeigen des Ressourcenverbrauchs

Zeigen Sie mit dem Befehl nützliche Informationen NeuronCore zur vCPU-Auslastung, zur Speichernutzung, zu geladenen Modellen und Neuron-Anwendungen an`neuron-top`. Wenn Sie `neuron-top` ohne Argumente starten, werden Daten für alle maschinellen Lernanwendungen angezeigt, die dies verwenden. NeuronCores

```
neuron-top
```

Wenn eine Anwendung vier verwendet NeuronCores, sollte die Ausgabe der folgenden Abbildung ähneln:



Weitere Informationen zu Ressourcen zur Überwachung und Optimierung neuronaler Inferenzanwendungen finden Sie unter Neuron Tools.

Verwendung von Neuron Monitor (Neuron-Monitor)

Neuron Monitor sammelt Metriken aus den Neuron-Laufzeiten, die auf dem System laufen, und streamt die gesammelten Daten im JSON-Format auf stdout. Diese Metriken sind in Metrikgruppen organisiert, die Sie konfigurieren, indem Sie eine Konfigurationsdatei bereitstellen. Weitere Informationen zu Neuron Monitor finden Sie im Benutzerhandbuch für Neuron Monitor.

Aktualisierung der Neuron-Software

Informationen zum Aktualisieren der Neuron SDK-Software in DLAMI finden Sie im AWS Neuron Setup Guide.

Nächster Schritt

Verwendung des DLAMI mit Neuron AWS

Verwendung des DLAMI mit Neuron AWS

Ein typischer Arbeitsablauf mit dem AWS Neuron SDK besteht darin, ein zuvor trainiertes Modell für maschinelles Lernen auf einem Kompilierungsserver zu kompilieren. Verteilen Sie anschließend die Artefakte zur Ausführung an die Inf1-Instanzen. AWS Deep Learning AMI (DLAMI) ist mit allem vorinstalliert, was Sie zum Kompilieren und Ausführen von Inferenzen in einer Inf1-Instanz benötigen, die Inferentia verwendet.

In den folgenden Abschnitten wird beschrieben, wie Sie das DLAMI mit Inferentia verwenden.

Inhalt

Verwendung von TensorFlow -Neuron und dem Neuron Compiler AWS

Dieses Tutorial zeigt, wie Sie mit dem AWS Neuron-Compiler das Keras ResNet -50-Modell kompilieren und als gespeichertes Modell im Format exportieren. SavedModel Dieses Format ist ein typisches TensorFlow austauschbares Modellformat. Sie lernen außerdem, wie die Inferenz für eine Inf1-Instance mit Beispieldaten ausgeführt wird.

Weitere Informationen zum Neuron SDK finden Sie in der AWS Neuron SDK-Dokumentation.

Inhalt

Voraussetzungen

Bevor Sie dieses Tutorial verwenden, müssen Sie die Einrichtungsschritte in Starten einer DLAMI-Instanz mit Neuron AWS abgeschlossen haben. Sie sollten auch mit Deep Learning und der Verwendung des DLAMI vertraut sein.

Aktivieren der Conda-Umgebung

Aktivieren Sie die TensorFlow -Neuron Conda-Umgebung mit dem folgenden Befehl:

```
source activate aws_neuron_tensorflow_p36
```

Führen Sie den folgenden Befehl aus, um die aktuelle Conda-Umgebung zu verlassen:

```
source deactivate
```

Resnet50-Kompilierung

Erstellen Sie das Python-Skript **`tensorflow_compile_resnet50.py`** mit folgendem Inhalt. Dieses Python-Skript kompiliert das Keras ResNet 50-Modell und exportiert es als gespeichertes Modell.

```
import os
import time
import shutil
import tensorflow as tf
import tensorflow.neuron as tfn
import tensorflow.compat.v1.keras as keras
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.applications.resnet50 import preprocess_input

# Create a workspace
WORKSPACE = './ws_resnet50'
os.makedirs(WORKSPACE, exist_ok=True)

# Prepare export directory (old one removed)
model_dir = os.path.join(WORKSPACE, 'resnet50')
compiled_model_dir = os.path.join(WORKSPACE, 'resnet50_neuron')
shutil.rmtree(model_dir, ignore_errors=True)
shutil.rmtree(compiled_model_dir, ignore_errors=True)

# Instantiate Keras ResNet50 model
keras.backend.set_learning_phase(0)
model = ResNet50(weights='imagenet')

# Export SavedModel
```

```
tf.saved_model.simple_save(
 session            = keras.backend.get_session(),
 export_dir         = model_dir,
 inputs             = {'input': model.inputs[0]},
 outputs            = {'output': model.outputs[0]})

# Compile using Neuron
tfn.saved_model.compile(model_dir, compiled_model_dir)

# Prepare SavedModel for uploading to Inf1 instance
shutil.make_archive(compiled_model_dir, 'zip', WORKSPACE, 'resnet50_neuron')
```

Kompilieren Sie das Modell mit dem folgenden Befehl:

```
python tensorflow_compile_resnet50.py
```

Der Kompilierungsprozess dauert einige Minuten. Nach Abschluss sollte die Ausgabe so aussehen:

```
...
INFO:tensorflow:fusing subgraph neuron_op_d6f098c01c780733 with neuron-cc
INFO:tensorflow:Number of operations in TensorFlow session: 4638
INFO:tensorflow:Number of operations after tf.neuron optimizations: 556
INFO:tensorflow:Number of operations placed on Neuron runtime: 554
INFO:tensorflow:Successfully converted ./ws_resnet50/resnet50 to ./ws_resnet50/
resnet50_neuron
...
```

Nach der Kompilierung wird das gespeicherte Modell gezippt: **`ws_resnet50/resnet50_neuron.zip`**. Entzippen Sie das Modell und laden Sie das Beispielbild mit den folgenden Befehlen herunter:

```
unzip ws_resnet50/resnet50_neuron.zip -d .
curl -O https://raw.githubusercontent.com/awslabs/mxnet-model-server/master/docs/
images/kitten_small.jpg
```

ResNet50 Inferenz

Erstellen Sie das Python-Skript **`tensorflow_infer_resnet50.py`** mit dem folgenden Inhalt. Dieses Skript führt die Inferenz für das heruntergeladene Modell unter Verwendung eines zuvor kompilierten Inferenzmodells aus.

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications import resnet50

# Create input from image
img_sgl = image.load_img('kitten_small.jpg', target_size=(224, 224))
img_arr = image.img_to_array(img_sgl)
img_arr2 = np.expand_dims(img_arr, axis=0)
img_arr3 = resnet50.preprocess_input(img_arr2)
# Load model
COMPILED_MODEL_DIR = './ws_resnet50/resnet50_neuron/'
predictor_inferentia = tf.contrib.predictor.from_saved_model(COMPILED_MODEL_DIR)
# Run inference
model_feed_dict={'input': img_arr3}
infa_rslts = predictor_inferentia(model_feed_dict);
# Display results
print(resnet50.decode_predictions(infa_rslts["output"], top=5)[0])
```

Führen Sie die Inferenz für das Modell mit dem folgenden Befehl aus:

```
python tensorflow_infer_resnet50.py
```

Die Ausgabe sollte folgendermaßen aussehen:

```
...
[('n02123045', 'tabby', 0.6918919), ('n02127052', 'lynx', 0.12770271), ('n02123159',
 'tiger_cat', 0.08277027), ('n02124075', 'Egyptian_cat', 0.06418919), ('n02128757',
 'snow_leopard', 0.009290541)]
```

Nächster Schritt

Verwenden von AWS Neuron Serving TensorFlow

Verwenden von AWS Neuron Serving TensorFlow

Dieses Tutorial zeigt, wie Sie ein Diagramm erstellen und einen AWS Neuron-Kompilierungsschritt hinzufügen, bevor Sie das gespeicherte Modell zur Verwendung mit TensorFlow Serving exportieren. TensorFlow Serving ist ein Serversystem, mit dem Sie Inferenzen in einem Netzwerk skalieren können. Neuron TensorFlow Serving verwendet dieselbe API wie normales Serving. TensorFlow Der einzige Unterschied besteht darin, dass ein gespeichertes Modell für AWS Inferentia kompiliert werden muss und der Einstiegspunkt eine andere Binärdatei mit dem Namen ist. `tensorflow_model_server_neuron` Die Binärdatei befindet sich im DLAMI `/usr/local/bin/tensorflow_model_server_neuron` und ist dort vorinstalliert.

Weitere Informationen zum Neuron SDK finden Sie in der Neuron SDK-Dokumentation.AWS

Inhalt

Voraussetzungen

Bevor Sie dieses Tutorial verwenden, müssen Sie die Einrichtungsschritte in Starten einer DLAMI-Instanz mit Neuron AWS abgeschlossen haben. Sie sollten auch mit Deep Learning und der Verwendung des DLAMI vertraut sein.

Aktivieren der Conda-Umgebung

Aktivieren Sie die TensorFlow -Neuron Conda-Umgebung mit dem folgenden Befehl:

```
source activate aws_neuron_tensorflow_p36
```

Wenn Sie die aktuelle Conda-Umgebung verlassen müssen, führen Sie Folgendes aus:

```
source deactivate
```

Kompilieren und Exportieren des gespeicherten Modells

Erstellen Sie ein Python-Skript namens `tensorflow-model-server-compile.py` mit dem folgenden Inhalt. Dieses Skript erstellt ein Diagramm und kompiliert es mit Neuron. Anschließend wird das kompilierte Diagramm als gespeichertes Modell exportiert.

```
import tensorflow as tf
import tensorflow.neuron
import os

tf.keras.backend.set_learning_phase(0)
model = tf.keras.applications.ResNet50(weights='imagenet')
sess = tf.keras.backend.get_session()
inputs = {'input': model.inputs[0]}
outputs = {'output': model.outputs[0]}

# save the model using tf.saved_model.simple_save
modeldir = "./resnet50/1"
tf.saved_model.simple_save(sess, modeldir, inputs, outputs)

# compile the model for Inferentia
neuron_modeldir = os.path.join(os.path.expanduser('~'), 'resnet50_inf1', '1')
tf.neuron.saved_model.compile(modeldir, neuron_modeldir, batch_size=1)
```

Kompilieren Sie das Modell mit dem folgenden Befehl:

```
python tensorflow-model-server-compile.py
```

Die Ausgabe sollte folgendermaßen aussehen:

```
...
INFO:tensorflow:fusing subgraph neuron_op_d6f098c01c780733 with neuron-cc
INFO:tensorflow:Number of operations in TensorFlow session: 4638
INFO:tensorflow:Number of operations after tf.neuron optimizations: 556
INFO:tensorflow:Number of operations placed on Neuron runtime: 554
INFO:tensorflow:Successfully converted ./resnet50/1 to /home/ubuntu/resnet50_inf1/1
```

Bereitstellen des gespeicherten Modells

Nachdem das Modell kompiliert wurde, können Sie den folgenden Befehl verwenden, um das gespeicherte Modell mit der Binärdatei tensorflow_model_server_neuron bereitzustellen:

```
tensorflow_model_server_neuron --model_name=resnet50_inf1 \
    --model_base_path=$HOME/resnet50_inf1/ --port=8500 &
```

Die Ausgabe sollte wie folgt aussehen. Das kompilierte Modell wird vom Server im DRAM des Inferentia-Geräts gespeichert, um die Inferenz vorzubereiten.

```
...
2019-11-22 01:20:32.075856: I external/org_tensorflow/tensorflow/cc/saved_model/
loader.cc:311] SavedModel load for tags { serve }; Status: success. Took 40764
 microseconds.
2019-11-22 01:20:32.075888: I tensorflow_serving/servables/tensorflow/
saved_model_warmup.cc:105] No warmup data file found at /home/ubuntu/resnet50_inf1/1/
assets.extra/tf_serving_warmup_requests
2019-11-22 01:20:32.075950: I tensorflow_serving/core/loader_harness.cc:87]
 Successfully loaded servable version {name: resnet50_inf1 version: 1}
2019-11-22 01:20:32.077859: I tensorflow_serving/model_servers/
server.cc:353] Running gRPC ModelServer at 0.0.0.0:8500 ...
```

Generieren von Inferenzanforderungen an den Modell-Server

Erstellen Sie das Python-Skript namens `tensorflow-model-server-infer.py` mit folgendem Inhalt. Dieses Skript führt die Inferenz über gRPC (Service-Framework) aus.

```
import numpy as np
import grpc
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow_serving.apis import predict_pb2
from tensorflow_serving.apis import prediction_service_pb2_grpc
from tensorflow.keras.applications.resnet50 import decode_predictions

if __name__ == '__main__':
    channel = grpc.insecure_channel('localhost:8500')
```

```
    stub = prediction_service_pb2_grpc.PredictionServiceStub(channel)
    img_file = tf.keras.utils.get_file(
        "./kitten_small.jpg",
        "https://raw.githubusercontent.com/awslabs/mxnet-model-server/master/docs/
images/kitten_small.jpg")
    img = image.load_img(img_file, target_size=(224, 224))
    img_array = preprocess_input(image.img_to_array(img)[None, ...])
    request = predict_pb2.PredictRequest()
    request.model_spec.name = 'resnet50_inf1'
    request.inputs['input'].CopyFrom(
        tf.contrib.util.make_tensor_proto(img_array, shape=img_array.shape))
    result = stub.Predict(request)
    prediction = tf.make_ndarray(result.outputs['output'])
    print(decode_predictions(prediction))
```

Führen Sie die Inferenz für das Modell aus, indem Sie gRPC mit dem folgenden Befehl verwenden:

```
python tensorflow-model-server-infer.py
```

Die Ausgabe sollte folgendermaßen aussehen:

```
[[('n02123045', 'tabby', 0.6918919), ('n02127052', 'lynx', 0.12770271), ('n02123159',
 'tiger_cat', 0.08277027), ('n02124075', 'Egyptian_cat', 0.06418919), ('n02128757',
 'snow_leopard', 0.009290541)]]
```

Verwendung von MXNet-Neuron und dem Neuron Compiler AWS

Die MXNet-Neuron-Kompilierungs-API bietet eine Methode zum Kompilieren eines Modelldiagramms, das Sie auf einem AWS Inferentia-Gerät ausführen können.

In diesem Beispiel verwenden Sie die API, um ein ResNet -50-Modell zu kompilieren und damit Inferenzen auszuführen.

Weitere Informationen zum Neuron SDK finden Sie in der Neuron SDK-Dokumentation AWS .

Inhalt

- Resnet50-Kompilierung
- ResNet50 Folgerung

Voraussetzungen

Bevor Sie dieses Tutorial verwenden, müssen Sie die Einrichtungsschritte in Starten einer DLAMI-Instanz mit Neuron AWS abgeschlossen haben. Sie sollten auch mit Deep Learning und der Verwendung des DLAMI vertraut sein.

Aktivieren der Conda-Umgebung

Aktivieren Sie die MXNet-Neuron-Conda-Umgebung mit folgendem Befehl:

```
source activate aws_neuron_mxnet_p36
```

Führen Sie Folgendes aus, um die aktuelle Conda-Umgebung zu verlassen:

```
source deactivate
```

Resnet50-Kompilierung

Erstellen Sie das Python-Skript namens **`mxnet_compile_resnet50.py`** mit folgendem Inhalt. Dieses Skript verwendet die Python-API für die MXNet-Neuron-Kompilierung, um ein ResNet -50-Modell zu kompilieren.

```
import mxnet as mx
import numpy as np

print("downloading...")
path='http://data.mxnet.io/models/imagenet/'
mx.test_utils.download(path+'resnet/50-layers/resnet-50-0000.params')
mx.test_utils.download(path+'resnet/50-layers/resnet-50-symbol.json')
print("download finished.")

sym, args, aux = mx.model.load_checkpoint('resnet-50', 0)

print("compile for inferentia using neuron... this will take a few minutes...")
```

```
inputs = { "data" : mx.nd.ones([1,3,224,224], name='data', dtype='float32') }

sym, args, aux = mx.contrib.neuron.compile(sym, args, aux, inputs)

print("save compiled model...")
mx.model.save_checkpoint("compiled_resnet50", 0, sym, args, aux)
```

Kompilieren Sie das Modell mit dem folgenden Befehl:

```
python mxnet_compile_resnet50.py
```

Die Kompilierung dauert einige Minuten. Wenn die Kompilierung abgeschlossen ist, befinden sich die folgenden Dateien in Ihrem aktuellen Verzeichnis:

```
resnet-50-0000.params
resnet-50-symbol.json
compiled_resnet50-0000.params
compiled_resnet50-symbol.json
```

ResNet50 Folgerung

Erstellen Sie das Python-Skript namens **`mxnet_infer_resnet50.py`** mit folgendem Inhalt. Mit diesem Skript wird ein Beispiel-Image heruntergeladen, das dazu verwendet wird, um die Inferenz für das kompilierte Modell auszuführen.

```
import mxnet as mx
import numpy as np

path='http://data.mxnet.io/models/imagenet/'
mx.test_utils.download(path+'synset.txt')

fname = mx.test_utils.download('https://raw.githubusercontent.com/awslabs/mxnet-model-
server/master/docs/images/kitten_small.jpg')
img = mx.image.imread(fname)

# convert into format (batch, RGB, width, height)
img = mx.image.imresize(img, 224, 224)
# resize
```

```
img = img.transpose((2, 0, 1))
# Channel first
img = img.expand_dims(axis=0)
# batchify
img = img.astype(dtype='float32')

sym, args, aux = mx.model.load_checkpoint('compiled_resnet50', 0)
softmax = mx.nd.random_normal(shape=(1,))
args['softmax_label'] = softmax
args['data'] = img
# Inferentia context
ctx = mx.neuron()

exe = sym.bind(ctx=ctx, args=args, aux_states=aux, grad_req='null')
with open('synset.txt', 'r') as f:
    labels = [l.rstrip() for l in f]

exe.forward(data=img)
prob = exe.outputs[0].asnumpy()
# print the top-5
prob = np.squeeze(prob)
a = np.argsort(prob)[::-1]
for i in a[0:5]:
    print('probability=%f, class=%s' %(prob[i], labels[i]))
```

Führen Sie die Inferenz mit folgendem Befehl mit dem kompilierten Modell aus:

```
python mxnet_infer_resnet50.py
```

Die Ausgabe sollte folgendermaßen aussehen:

```
probability=0.642454, class=n02123045 tabby, tabby cat
probability=0.189407, class=n02123159 tiger cat
probability=0.100798, class=n02124075 Egyptian cat
probability=0.030649, class=n02127052 lynx, catamount
probability=0.016278, class=n02129604 tiger, Panthera tigris
```

Nächster Schritt

Verwenden der MXNet-Neuron-Modellbereitstellung

Verwenden der MXNet-Neuron-Modellbereitstellung

In diesem Tutorial lernen Sie, ein vorgeschultes MXNet-Modell für die Echtzeit-Bildklassifizierung mit MMS (Multi Model Server) zu verwenden. MMS ist ein flexibles easy-to-use Tool zur Bereitstellung von Deep-Learning-Modellen, die mit einem beliebigen Framework für maschinelles Lernen oder Deep Learning trainiert werden. Dieses Tutorial beinhaltet einen Kompilierungsschritt mit AWS Neuron und eine Implementierung von MMS mit MXNet.

Weitere Informationen zum Neuron SDK finden Sie in der Neuron SDK-Dokumentation.AWS

Inhalt

Voraussetzungen

Bevor Sie dieses Tutorial verwenden, müssen Sie die Einrichtungsschritte in Starten einer DLAMI-Instanz mit Neuron AWS abgeschlossen haben. Sie sollten auch mit Deep Learning und der Verwendung des DLAMI vertraut sein.

Aktivieren der Conda-Umgebung

Aktivieren Sie die MXNet-Neuron-Conda-Umgebung mit folgendem Befehl:

```
source activate aws_neuron_mxnet_p36
```

Führen Sie Folgendes aus, um die aktuelle Conda-Umgebung zu verlassen:

```
source deactivate
```

Herunterladen des Beispielcodes

Um dieses Beispiel auszuführen, laden Sie den Beispielcode mit den folgenden Befehlen herunter:

```
git clone https://github.com/awslabs/multi-model-server
cd multi-model-server/examples/mxnet_vision
```

Kompilieren des Modells

Erstellen Sie das Python-Skript namens `multi-model-server-compile.py` mit folgendem Inhalt. Dieses Skript kompiliert das Modell ResNet 50 für das Inferentia-Geräteziel.

```
import mxnet as mx
from mxnet.contrib import neuron
import numpy as np

path='http://data.mxnet.io/models/imagenet/'
mx.test_utils.download(path+'resnet/50-layers/resnet-50-0000.params')
mx.test_utils.download(path+'resnet/50-layers/resnet-50-symbol.json')
mx.test_utils.download(path+'synset.txt')

nn_name = "resnet-50"

#Load a model
sym, args, auxs = mx.model.load_checkpoint(nn_name, 0)

#Define compilation parameters#  - input shape and dtype
inputs = {'data' : mx.nd.zeros([1,3,224,224], dtype='float32') }

# compile graph to inferentia target
csym, cargs, cauxs = neuron.compile(sym, args, auxs, inputs)

# save compiled model
mx.model.save_checkpoint(nn_name + "_compiled", 0, csym, cargs, cauxs)
```

Verwenden Sie den folgenden Befehl, um das Modell zu kompilieren:

```
python multi-model-server-compile.py
```

Die Ausgabe sollte folgendermaßen aussehen:

```
...
[21:18:40] src/nnvm/legacy_json_util.cc:209: Loading symbol saved by previous version
 v0.8.0. Attempting to upgrade...
[21:18:40] src/nnvm/legacy_json_util.cc:217: Symbol successfully upgraded!
[21:19:00] src/operator/subgraph/build_subgraph.cc:698: start to execute partition
 graph.
[21:19:00] src/nnvm/legacy_json_util.cc:209: Loading symbol saved by previous version
 v0.8.0. Attempting to upgrade...
```

```
[21:19:00] src/nnvm/legacy_json_util.cc:217: Symbol successfully upgraded!
```

Erstellen Sie eine Datei namens `signature.json` mit dem folgenden Inhalt, um den Eingabenamen und das Shape zu konfigurieren:

```
{
  "inputs": [
    {
      "data_name": "data",
      "data_shape": [
        1,
        3,
        224,
        224
      ]
    }
  ]
}
```

Sie können die Datei `synset.txt` mit dem folgenden Befehl herunterladen. Diese Datei ist eine Liste mit Namen für ImageNet Vorhersageklassen.

```
curl -O https://s3.amazonaws.com/model-server/model_archive_1.0/examples/
squeezenet_v1.1/synset.txt
```

Erstellen Sie eine benutzerdefinierte Service-Klasse unter Verwendung der Vorlage im Ordner `model_server_template`. Kopieren Sie die Vorlage mit dem folgenden Befehl in das aktuelle Arbeitsverzeichnis:

```
cp -r ../model_service_template/* .
```

Bearbeiten Sie das Modul `mxnet_model_service.py`, indem Sie den Kontext `mx.cpu()` wie folgt durch den Kontext `mx.neuron()` ersetzen. Sie müssen außerdem die nicht benötige Datenkopie für `model_input` auskommentieren, da MXNet-Neuron die NDArray- und die Gluon-API nicht unterstützt.

```
...
self.mxnet_ctx = mx.neuron() if gpu_id is None else mx.gpu(gpu_id)
...
#model_input = [item.as_in_context(self.mxnet_ctx) for item in model_input]
```

Erstellen Sie mit den folgenden Befehlen ein Paket des Modells mit model-archiver:

```
cd ~/multi-model-server/examples
model-archiver --force --model-name resnet-50_compiled --model-path mxnet_vision --
handler mxnet_vision_service:handle
```

Ausführen der Inferenz

Starten Sie den Multimodell-Server und laden Sie das Modell, das die RESTful-API verwendet, mit den folgenden Befehlen. Stellen Sie sicher, dass neuron-rtd mit den Standardeinstellungen ausgeführt wird.

```
cd ~/multi-model-server/
multi-model-server --start --model-store examples > /dev/null # Pipe to log file if you
 want to keep a log of MMS
curl -v -X POST "http://localhost:8081/models?
initial_workers=1&max_workers=4&synchronous=true&url=resnet-50_compiled.mar"
sleep 10 # allow sufficient time to load model
```

Führen Sie die Inferenz mit den folgenden Befehlen mit einem Beispielbild aus:

```
curl -O https://raw.githubusercontent.com/awslabs/multi-model-server/master/docs/
images/kitten_small.jpg
curl -X POST http://127.0.0.1:8080/predictions/resnet-50_compiled -T kitten_small.jpg
```

Die Ausgabe sollte folgendermaßen aussehen:

```
[
  {
    "probability": 0.6388034820556641,
    "class": "n02123045 tabby, tabby cat"
  },
  {
    "probability": 0.16900072991847992,
    "class": "n02123159 tiger cat"
  },
  {
    "probability": 0.12221276015043259,
    "class": "n02124075 Egyptian cat"
  },
  {
```

```
    "probability": 0.028706775978207588,
    "class": "n02127052 lynx, catamount"
  },
  {
    "probability": 0.01915954425930977,
    "class": "n02129604 tiger, Panthera tigris"
  }
]
```

Geben Sie zur Nachbereitung des Tests einen Löschbefehl über die RESTful-API aus und beenden Sie den Modell-Server mit den folgenden Befehlen:

```
curl -X DELETE http://127.0.0.1:8081/models/resnet-50_compiled

multi-model-server --stop
```

Die Ausgabe sollte folgendermaßen aussehen:

```
{
  "status": "Model \"resnet-50_compiled\" unregistered"
}
Model server stopped.
Found 1 models and 1 NCGs.
Unloading 10001 (MODEL_STATUS_STARTED) :: success
Destroying NCG 1 :: success
```

Verwenden von PyTorch -Neuron und dem Neuron Compiler AWS

Die PyTorch -Neuron-Kompilierungs-API bietet eine Methode zum Kompilieren eines Modelldiagramms, das Sie auf einem Inferentia-Gerät ausführen können. AWS

Ein trainiertes Modell muss für ein Inferentia-Ziel kompiliert werden, bevor es auf Inf1-Instances bereitgestellt werden kann. Das folgende Tutorial kompiliert das Torchvision ResNet 50-Modell und exportiert es als gespeichertes Modul. TorchScript Dieses Modell wird dann zum Ausführen der Inferenz verwendet.

Der Einfachheit halber wird in diesem Tutorial eine Inf1-Instance sowohl für die Kompilierung als auch für die Inferenz verwendet. In der Praxis können Sie Ihr Modell mit einem anderen Instance-Typ kompilieren, z. B. mit der c5-Instance-Familie. Anschließend müssen Sie das kompilierte Modell auf dem Inf1-Inferenzserver bereitstellen. Weitere Informationen finden Sie in der AWS Neuron PyTorch SDK-Dokumentation.

Inhalt

Voraussetzungen

Bevor Sie dieses Tutorial verwenden, müssen Sie die Einrichtungsschritte in Starten einer DLAMI-Instanz mit Neuron AWS abgeschlossen haben. Sie sollten auch mit Deep Learning und der Verwendung des DLAMI vertraut sein.

Aktivieren der Conda-Umgebung

Aktivieren Sie die PyTorch -Neuron Conda-Umgebung mit dem folgenden Befehl:

```
source activate aws_neuron_pytorch_p36
```

Führen Sie Folgendes aus, um die aktuelle Conda-Umgebung zu verlassen:

```
source deactivate
```

Resnet50-Kompilierung

Erstellen Sie das Python-Skript namens **`pytorch_trace_resnet50.py`** mit folgendem Inhalt. Dieses Skript verwendet die Python-API zur Kompilierung von PyTorch -Neuron, um ein ResNet -50-Modell zu kompilieren.

> **Note**
>
> Es besteht eine Abhängigkeit zwischen den Versionen von Torchvision und dem Torch-Paket, die Sie beim Kompilieren von Torchvision-Modellen beachten sollten. Diese Abhängigkeitsregeln können über Pip verwaltet werden. Torchvision==0.6.1 entspricht der Version Torch==1.5.1, während Torchvision==0.8.2 der Version Torch==1.7.1 entspricht.

```
import torch
```

```
import numpy as np
import os
import torch_neuron
from torchvision import models

image = torch.zeros([1, 3, 224, 224], dtype=torch.float32)

## Load a pretrained ResNet50 model
model = models.resnet50(pretrained=True)

## Tell the model we are using it for evaluation (not training)
model.eval()
model_neuron = torch.neuron.trace(model, example_inputs=[image])

## Export to saved model
model_neuron.save("resnet50_neuron.pt")
```

Führen Sie das Kompilierungsskript aus.

```
python pytorch_trace_resnet50.py
```

Die Kompilierung wird ein paar Minuten dauern. Nach Abschluss der Kompilierung wird das kompilierte Modell wie `resnet50_neuron.pt` im lokalen Verzeichnis gespeichert.

ResNet50 Inferenz

Erstellen Sie das Python-Skript namens **`pytorch_infer_resnet50.py`** mit folgendem Inhalt. Mit diesem Skript wird ein Beispiel-Image heruntergeladen, das dazu verwendet wird, um die Inferenz für das kompilierte Modell auszuführen.

```
import os
import time
import torch
import torch_neuron
import json
import numpy as np

from urllib import request

from torchvision import models, transforms, datasets

## Create an image directory containing a small kitten
```

```
os.makedirs("./torch_neuron_test/images", exist_ok=True)
request.urlretrieve("https://raw.githubusercontent.com/awslabs/mxnet-model-server/
master/docs/images/kitten_small.jpg",
                    "./torch_neuron_test/images/kitten_small.jpg")


## Fetch labels to output the top classifications
request.urlretrieve("https://s3.amazonaws.com/deep-learning-models/image-models/
imagenet_class_index.json","imagenet_class_index.json")
idx2label = []

with open("imagenet_class_index.json", "r") as read_file:
    class_idx = json.load(read_file)
    idx2label = [class_idx[str(k)][1] for k in range(len(class_idx))]

## Import a sample image and normalize it into a tensor
normalize = transforms.Normalize(
    mean=[0.485, 0.456, 0.406],
    std=[0.229, 0.224, 0.225])

eval_dataset = datasets.ImageFolder(
    os.path.dirname("./torch_neuron_test/"),
    transforms.Compose([
    transforms.Resize([224, 224]),
    transforms.ToTensor(),
    normalize,
    ])
)

image, _ = eval_dataset[0]
image = torch.tensor(image.numpy()[np.newaxis, ...])

## Load model
model_neuron = torch.jit.load( 'resnet50_neuron.pt' )

## Predict
results = model_neuron( image )

# Get the top 5 results
top5_idx = results[0].sort()[1][-5:]

# Lookup and print the top 5 labels
top5_labels = [idx2label[idx] for idx in top5_idx]
```

```
print("Top 5 labels:\n {}".format(top5_labels) )
```

Führen Sie die Inferenz mit folgendem Befehl mit dem kompilierten Modell aus:

```
python pytorch_infer_resnet50.py
```

Die Ausgabe sollte folgendermaßen aussehen:

```
Top 5 labels:
 ['tiger', 'lynx', 'tiger_cat', 'Egyptian_cat', 'tabby']
```

Das Graviton DLAMI

AWS Graviton-GPU-DLAMIs wurden entwickelt, um eine hohe Leistung und Kosteneffizienz für Deep-Learning-Workloads zu bieten. Insbesondere der G5G-Instanztyp verfügt über den ARM-basierten AWS Graviton2-Prozessor, der von Grund auf neu entwickelt AWS und dafür optimiert wurde, wie Kunden ihre Workloads in der Cloud ausführen. AWS Graviton-GPU-DLAMIs sind mit Docker, NVIDIA Docker, NVIDIA Driver, CUDA, cuDNN, NCCL und TensorRT sowie beliebten Frameworks für maschinelles Lernen wie und vorkonfiguriert. TensorFlow PyTorch

Mit dem Instance-Typ G5G können Sie die Preis- und Leistungsvorteile von Graviton2 nutzen, um GPU-beschleunigte Deep-Learning-Modelle zu deutlich geringeren Kosten im Vergleich zu x86-basierten Instances mit GPU-Beschleunigung bereitzustellen.

Wählen Sie ein Graviton DLAMI

Starten Sie eine G5g-Instance mit dem Graviton DLAMI Ihrer Wahl.

step-by-step Anweisungen zum Starten eines DLAMI finden Sie unter Starten und Konfigurieren eines DLAMI.

Eine Liste der neuesten Graviton-DLAMIs finden Sie in den Versionshinweisen für DLAMI.

Erste Schritte

Die folgenden Themen zeigen Ihnen, wie Sie mit dem Graviton DLAMI beginnen können.

Inhalt

Verwenden der Graviton-GPU DLAMI

Der AWS Deep Learning AMI ist sofort einsatzbereit mit Graviton-GPUs auf Arm-Prozessorbasis. Das Graviton GPU DLAMI verfügt über eine grundlegende Plattform aus GPU-Treibern und Beschleunigungsbibliotheken, mit denen Sie Ihre eigene maßgeschneiderte Deep-Learning-Umgebung bereitstellen können. Docker und NVIDIA Docker sind auf dem Graviton-GPU-DLAMI vorkonfiguriert, sodass Sie containerisierte Anwendungen bereitstellen können. Weitere Informationen zur Graviton-GPU DLAMI finden Sie in den Versionshinweisen.

Inhalt

Überprüfen Sie den GPU-Status

Verwenden Sie die NVIDIA System Management Interface, um den Status Ihrer Graviton-GPU zu überprüfen.

```
nvidia-smi
```

Die Ausgabe des `nvidia-smi` Befehls sollte der folgenden ähneln:

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.82.01    Driver Version: 470.82.01    CUDA Version: 11.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA T4G          On   | 00000000:00:1F.0 Off |                    0 |
```

```
| N/A   32C    P8     8W /  70W |      0MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Überprüfen Sie die CUDA-Version

Führen Sie den folgenden Befehl aus, um Ihre CUDA-Version zu überprüfen:

```
/usr/local/cuda/bin/nvcc --version | grep Cuda
```

Ihre Ausgabe sollte wie folgt aussehen:

```
nvcc: NVIDIA (R) Cuda compiler driver
Cuda compilation tools, release 11.4, V11.4.120
```

Überprüfen Sie Docker

Führen Sie einen CUDA-Container von aus, um die Docker-Funktionalität auf Ihrer Graviton-GPU DockerHubzu überprüfen:

```
sudo docker run --platform=linux/arm64 --rm \
    --gpus all nvidia/cuda:11.4.2-base-ubuntu20.04 nvidia-smi
```

Ihre Ausgabe sollte wie folgt aussehen:

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.82.01    Driver Version: 470.82.01    CUDA Version: 11.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA T4G          On   | 00000000:00:1F.0 Off |                    0 |
```

```
| N/A   33C    P8     9W /  70W |      0MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

TensorRT

Verwenden Sie den folgenden Befehl, um auf das TensorRT-Befehlszeilentool zuzugreifen:

```
trtexec
```

Ihre Ausgabe sollte wie folgt aussehen:

```
&&&& RUNNING TensorRT.trtexec [TensorRT v8200] # trtexec
...
&&&& PASSED TensorRT.trtexec [TensorRT v8200] # trtexec
```

Auf Anfrage sind TensorRT-Python-Räder zur Installation erhältlich. Sie finden diese Räder an den folgenden Dateispeicherorten:

```
/usr/local/tensorrt/graphsurgeon/
### graphsurgeon-0.4.5-py2.py3-none-any.whl

/usr/local/tensorrt/onnx_graphsurgeon/
### onnx_graphsurgeon-0.3.12-py2.py3-none-any.whl

/usr/local/tensorrt/python/
### tensorrt-8.2.0.6-cp36-none-linux_aarch64.whl
### tensorrt-8.2.0.6-cp37-none-linux_aarch64.whl
### tensorrt-8.2.0.6-cp38-none-linux_aarch64.whl
### tensorrt-8.2.0.6-cp39-none-linux_aarch64.whl

/usr/local/tensorrt/uff/
### uff-0.6.9-py2.py3-none-any.whl
```

Weitere Informationen finden Sie in der NVIDIA TensorRT-Dokumentation.

Führen Sie CUDA-Beispiele aus

Das Graviton GPU DLAMI bietet vorkompilierte CUDA-Beispiele, mit denen Sie verschiedene CUDA-Funktionen überprüfen können.

```
ls /usr/local/cuda/compiled_samples
```

Führen Sie das Beispiel beispielsweise mit dem `vectorAdd` folgenden Befehl aus:

```
/usr/local/cuda/compiled_samples/vectorAdd
```

Ihre Ausgabe sollte wie folgt aussehen:

```
[Vector addition of 50000 elements]
Copy input data from the host memory to the CUDA device
CUDA kernel launch with 196 blocks of 256 threads
Copy output data from the CUDA device to the host memory
Test PASSED
Done
```

Führen Sie das `transpose` Beispiel aus:

```
/usr/local/cuda/compiled_samples/transpose
```

Ihre Ausgabe sollte wie folgt aussehen:

```
Transpose Starting...

GPU Device 0: "Turing" with compute capability 7.5

> Device 0: "NVIDIA T4G"
> SM Capability 7.5 detected:
> [NVIDIA T4G] has 40 MP(s) x 64 (Cores/MP) = 2560 (Cores)
> Compute performance scaling factor = 1.00

Matrix size: 1024x1024 (64x64 tiles), tile size: 16x16, block size: 16x16

transpose simple copy       , Throughput = 185.1781 GB/s, Time = 0.04219 ms, Size =
 1048576 fp32 elements, NumDevsUsed = 1, Workgroup = 256
```

```
transpose shared memory copy, Throughput = 163.8616 GB/s, Time = 0.04768 ms, Size =
 1048576 fp32 elements, NumDevsUsed = 1, Workgroup = 256
transpose naive             , Throughput = 98.2805 GB/s, Time = 0.07949 ms, Size =
 1048576 fp32 elements, NumDevsUsed = 1, Workgroup = 256
transpose coalesced         , Throughput = 127.6759 GB/s, Time = 0.06119 ms, Size =
 1048576 fp32 elements, NumDevsUsed = 1, Workgroup = 256
transpose optimized         , Throughput = 156.2960 GB/s, Time = 0.04999 ms, Size =
 1048576 fp32 elements, NumDevsUsed = 1, Workgroup = 256
transpose coarse-grained    , Throughput = 155.9157 GB/s, Time = 0.05011 ms, Size =
 1048576 fp32 elements, NumDevsUsed = 1, Workgroup = 256
transpose fine-grained      , Throughput = 158.4177 GB/s, Time = 0.04932 ms, Size =
 1048576 fp32 elements, NumDevsUsed = 1, Workgroup = 256
transpose diagonal          , Throughput = 133.4277 GB/s, Time = 0.05855 ms, Size =
 1048576 fp32 elements, NumDevsUsed = 1, Workgroup = 256
Test passed
```

Nächstes Thema

Verwenden der Graviton-GPU DLAMI TensorFlow

Verwenden der Graviton-GPU DLAMI TensorFlow

Der AWS Deep Learning AMI ist sofort einsatzbereit mit Graviton-GPUs auf Arm-Prozessorbasis und ist für ihn optimiert. TensorFlow Das Graviton GPU TensorFlow DLAMI umfasst eine Python-Umgebung, die mit TensorFlow Serving für Deep-Learning-Inferenz-Anwendungsfälle vorkonfiguriert ist. Weitere Informationen zur Graviton-GPU TensorFlow DLAMI finden Sie in den Versionshinweisen.

Inhalt

Überprüfen Sie die Serververfügbarkeit TensorFlow

Führen Sie den folgenden Befehl aus, um die Verfügbarkeit und Version von Serving zu überprüfen: TensorFlow

```
tensorflow_model_server --version
```

Ihre Ausgabe sollte wie folgt aussehen:

```
TensorFlow ModelServer: 0.0.0+dev.sha.3e05381e
TensorFlow Library: 2.8.0
```

Überprüfen TensorFlow und TensorFlow Bereitstellen der API-Verfügbarkeit

Führen Sie den folgenden Befehl aus, um die Verfügbarkeit TensorFlow und die Serving-API zu überprüfen: TensorFlow

```
python3 -c "import tensorflow, tensorflow_serving"
```

Wenn der Befehl erfolgreich ist, erfolgt keine Ausgabe.

Führen Sie die Beispiel-Inferenz mit TensorFlow Serving aus

Verwenden Sie die folgenden Befehle, um ein vortrainiertes ResNet 50-Modell herunterzuladen und die Inferenz mithilfe von Serving auszuführen: TensorFlow

```
# Clone the TensorFlow Serving repository
git clone https://github.com/tensorflow/serving

# Download pre-trained ResNet50 model
mkdir -p ${HOME}/resnet/1 && cd ${HOME}/resnet/1
wget https://tfhub.dev/tensorflow/resnet_50/classification/1?tf-hub-format=compressed -
O resnet_50_classification_1.tar.gz
tar -xzvf resnet_50_classification_1.tar.gz && rm resnet_50_classification_1.tar.gz

# Start TensorFlow Serving
cd $HOME
tensorflow_model_server \
  --rest_api_port=8501 \
  --model_name="resnet" \
  --model_base_path="${HOME}/resnet" &
```

Ihre Ausgabe sollte wie folgt aussehen:

```
2021-11-10 06:18:51.028341: I tensorflow_serving/model_servers/server_core.cc:486]
 Finished adding/updating models
2021-11-10 06:18:51.028420: I tensorflow_serving/model_servers/server.cc:133] Using
 InsecureServerCredentials
```

```
2021-11-10 06:18:51.028460: I tensorflow_serving/model_servers/server.cc:383] Profiler
 service is enabled
2021-11-10 06:18:51.028889: I tensorflow_serving/model_servers/server.cc:409] Running
 gRPC ModelServer at 0.0.0.0:8500 ...
[evhttp_server.cc : 245] NET_LOG: Entering the event loop ...
2021-11-10 06:18:51.030985: I tensorflow_serving/model_servers/server.cc:430] Exporting
 HTTP/REST API at:localhost:8501 ...
```

Verwenden Sie das TensorFlow `resnet_client` Serving-Beispiel, um die Inferenz auszuführen:

```
python3 serving/tensorflow_serving/example/resnet_client.py
```

Ihre Ausgabe sollte wie folgt aussehen:

```
2021-11-10 06:18:59.335327: I external/org_tensorflow/tensorflow/stream_executor/cuda/
cuda_dnn.cc:368] Loaded cuDNN version 8204
2021-11-10 06:18:59.956156: I external/org_tensorflow/tensorflow/core/platform/default/
subprocess.cc:304] Start cannot spawn child process
Prediction class: 285, avg latency: 111.4673 ms
```

Beenden Sie das TensorFlow Servieren mit dem folgenden Befehl:

```
kill $(pidof tensorflow_model_server)
```

Nächstes Thema

Verwenden der Graviton-GPU DLAMI PyTorch

Verwenden der Graviton-GPU DLAMI PyTorch

Der AWS Deep Learning AMI ist sofort mit Graviton-GPUs auf Arm-Prozessorbasis einsatzbereit und ist optimiert für. PyTorch Das Graviton GPU PyTorch DLAMI umfasst eine Python-Umgebung, die mit PyTorch, TorchVisionund TorchServefür Deep-Learning-Training und Inferenz-Anwendungsfälle vorkonfiguriert ist. Weitere Informationen zur Graviton-GPU PyTorch DLAMI finden Sie in den Versionshinweisen.

Inhalt

- PyTorch Python-Umgebung überprüfen
- Trainingsbeispiel ausführen mit PyTorch
- Führen Sie ein Inferenzbeispiel aus mit PyTorch

PyTorch Python-Umgebung überprüfen

Connect zu Ihrer G5g-Instance her und aktivieren Sie die Conda-Basisumgebung mit dem folgenden Befehl:

```
source activate base
```

Ihre Eingabeaufforderung sollte darauf hinweisen, dass Sie in der Conda-Basisumgebung arbeiten, die PyTorch TorchVision, und andere Bibliotheken enthält.

```
(base) $
```

Überprüfen Sie die Standard-Werkzeugpfade der PyTorch Umgebung:

```
(base) $ which python
/opt/conda/bin/python

(base) $ which pip
/opt/conda/bin/pip

(base) $ which conda
/opt/conda/bin/conda

(base) $ which mamba
/opt/conda/bin/mamba
```

Stellen Sie sicher, dass Torch und Torch verfügbar TorchVersion sind, überprüfen Sie ihre Versionen und testen Sie die grundlegenden Funktionen:

```
(base) $ python
Python 3.8.12 | packaged by conda-forge | (default, Oct 12 2021, 23:06:28)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch, torchvision
>>> torch.__version__
'1.10.0'
>>> torchvision.__version__
'0.11.1'
>>> v = torch.autograd.Variable(torch.randn(10, 3, 224, 224))
>>> v = torch.autograd.Variable(torch.randn(10, 3, 224, 224)).cuda()
>>> assert isinstance(v, torch.Tensor)
```

Trainingsbeispiel ausführen mit PyTorch

Führen Sie einen Beispiel-MNIST-Trainingsjob aus:

```
git clone https://github.com/pytorch/examples.git
cd examples/mnist
python main.py
```

Ihre Ausgabe sollte wie folgt aussehen:

```
...
Train Epoch: 14 [56320/60000 (94%)]    Loss: 0.021424
Train Epoch: 14 [56960/60000 (95%)]    Loss: 0.023695
Train Epoch: 14 [57600/60000 (96%)]    Loss: 0.001973
Train Epoch: 14 [58240/60000 (97%)]    Loss: 0.007121
Train Epoch: 14 [58880/60000 (98%)]    Loss: 0.003717
Train Epoch: 14 [59520/60000 (99%)]    Loss: 0.001729
Test set: Average loss: 0.0275, Accuracy: 9916/10000 (99%)
```

Führen Sie ein Inferenzbeispiel aus mit PyTorch

Verwenden Sie die folgenden Befehle, um ein vortrainiertes densenet161-Modell herunterzuladen und die Inferenz auszuführen mit: TorchServe

```
# Set up TorchServe
cd $HOME
git clone https://github.com/pytorch/serve.git
mkdir -p serve/model_store
cd serve

# Download a pre-trained densenet161 model
wget https://download.pytorch.org/models/densenet161-8d451a50.pth >/dev/null

# Save the model using torch-model-archiver
torch-model-archiver --model-name densenet161 \
    --version 1.0 \
    --model-file examples/image_classifier/densenet_161/model.py \
    --serialized-file densenet161-8d451a50.pth \
    --handler image_classifier \
    --extra-files examples/image_classifier/index_to_name.json  \
    --export-path model_store

# Start the model server
```

```
torchserve --start --no-config-snapshots \
    --model-store model_store \
    --models densenet161=densenet161.mar &> torchserve.log

# Wait for the model server to start
sleep 30

# Run a prediction request
curl http://127.0.0.1:8080/predictions/densenet161 -T examples/image_classifier/
kitten.jpg
```

Ihre Ausgabe sollte wie folgt aussehen:

```
{
  "tiger_cat": 0.4693363308906555,
  "tabby": 0.4633873701095581,
  "Egyptian_cat": 0.06456123292446136,
  "lynx": 0.0012828150065615773,
  "plastic_bag": 0.00023522898778133094
}
```

Verwenden Sie die folgenden Befehle, um die Registrierung des densenet161-Modells aufzuheben und den Server anzuhalten:

```
curl -X DELETE http://localhost:8081/models/densenet161/1.0
torchserve --stop
```

Ihre Ausgabe sollte wie folgt aussehen:

```
{
  "status": "Model \"densenet161\" unregistered"
}
TorchServe has stopped.
```

Das Habana DLAMI

Instanzen mit Habana-Beschleunigern sind so konzipiert, dass sie eine hohe Leistung und Kosteneffizienz für Trainingsworkloads im Deep-Learning-Modell bieten. Insbesondere DL1-Instance-Typen verwenden Habana Gaudi-Beschleuniger von Habana Labs, einem Intel-Unternehmen. Instanzen mit Habana-Beschleunigern werden mit der Habana SynapseAI-Software konfiguriert und in beliebte Frameworks für maschinelles Lernen wie und vorintegriert. TensorFlow PyTorch

Die folgenden Themen zeigen Ihnen, wie Sie mit der Verwendung der Habana Gaudi-Hardware mit dem DLAMI beginnen können.

Inhalt

Markteinführung eines Habana DLAMI

Das neueste DLAMI ist bereit, mit Habana Gaudi-Beschleunigern verwendet zu werden. Gehen Sie wie folgt vor, um Ihr Habana DLAMI zu starten und sicherzustellen, dass Ihre Python- und Framework-spezifischen Ressourcen aktiv sind. Weitere Einrichtungsressourcen finden Sie im Habana Gaudi Setup and Installation Repository.

Inhalt

Wählen Sie ein Habana DLAMI

Starten Sie eine DL1-Instance mit dem Habana DLAMI Ihrer Wahl.

step-by-step Anweisungen zum Starten eines DLAMI finden Sie unter Starten und Konfigurieren eines DLAMI.

Eine Liste der neuesten Habana-DLAMIs finden Sie in den Versionshinweisen für DLAMI.

Python-Umgebung aktivieren

Connect zu Ihrer DL1-Instanz her und aktivieren Sie die empfohlene Python-Umgebung für Ihr Habana DLAMI. Um Ihre empfohlene Python-Umgebung zu überprüfen, wählen Sie Ihr DLAMI in den Versionshinweisen aus.

Framework für Machine Learning importieren

Instanzen mit Habana-Beschleunigern sind in beliebte Frameworks für maschinelles Lernen wie und vorintegriert. TensorFlow PyTorch Importieren Sie das Framework für maschinelles Lernen Ihrer Wahl.

Importieren TensorFlow

Um DLAMI TensorFlow auf Ihrem Habana zu verwenden, navigieren Sie zu dem Ordner der Python-Umgebung, die Sie aktiviert und importiert haben. TensorFlow

```
/usr/bin/$PYTHON_VERSION
import tensorflow
tensorflow.__version__
```

Um zu überprüfen, welche TensorFlow Version mit Ihrem Habana DLAMI kompatibel ist, wählen Sie Ihr DLAMI in den Versionshinweisen aus.

Importieren PyTorch

Um DLAMI PyTorch auf Ihrem Habana zu verwenden, navigieren Sie zu dem Ordner der Python-Umgebung, die Sie aktiviert haben, und importieren Sie die entsprechende Version. PyTorch

```
/usr/bin/$PYTHON_VERSION
import torch
torch.__version__
```

Um zu überprüfen, welche PyTorch Version mit Ihrem Habana DLAMI kompatibel ist, wählen Sie Ihr DLAMI in den Versionshinweisen aus.

Weitere Informationen zum Ausführen und Trainieren von Modellen für maschinelles Lernen in TensorFlow und PyTorch mit Ihrem Habana DLAMI finden Sie im Habana Model References Repository. GitHub Weitere Ressourcen zur Arbeit mit Ihrem Habana DLAMI finden Sie in der Habana Gaudi-Dokumentation.

Inferenz

Dieser Abschnitt enthält Tutorials zum Ausführen von Inferenzen mit den Frameworks und Tools des DLAMI.

Tutorials für die Verwendung von Elastic Inference finden Sie unter Arbeiten mit Amazon Elastic Inference

Inferenz mit Frameworks

- Verwenden Sie Apache MXNet (Incubating) für Inferenzen mit einem ONNX-Modell
- Verwenden Sie Apache MXNet (Incubating) für Inferenz mit einem 50-Modell ResNet

- Verwenden von CNTK für Inferenz mit einem ONNX-Modell

Inferenz-Tools

- Modellserver für Apache MXNet (MMS)
- TensorFlow Servieren

Verwenden Sie Apache MXNet (Incubating) für Inferenzen mit einem ONNX-Modell

So verwenden Sie ein ONNX-Modell für Bildinferenz mit Apache MXNet (Inkubation)

1.
   - (Option für Python 3) — Aktivieren Sie die Python 3 Apache MXNet (Incubating) - Umgebung:

     ```
     $ source activate mxnet_p36
     ```

   - (Option für Python 2) — Aktivieren Sie die Python 2-Umgebung Apache MXNet (Incubating):

     ```
     $ source activate mxnet_p27
     ```

2. Für die restlichen Schritte wird davon ausgegangen, dass Sie die `mxnet_p36`-Umgebung verwenden.
3. Laden Sie das Bild eines Huskys herunter.

   ```
   $ curl -O https://upload.wikimedia.org/wikipedia/commons/b/b5/Siberian_Husky_bi-eyed_Flickr.jpg
   ```

4. Laden Sie eine Liste der Klassen herunter, die mit diesem Modell funktionieren.

   ```
   $ curl -O https://gist.githubusercontent.com/yrevar/6135f1bd8dcf2e0cc683/raw/d133d61a09d7e5a3b36b8c111a8dd5c4b5d560ee/imagenet1000_clsid_to_human.pkl
   ```

5. Laden Sie das vorgeschulte VGG 16-Modell im ONNX-Format herunter.

   ```
   $ wget -O vgg16.onnx https://github.com/onnx/models/raw/master/vision/classification/vgg/model/vgg16-7.onnx
   ```

6. Verwenden Sie den von Ihnen bevorzugten Texteditor, um ein Skript mit folgendem Inhalt zu erstellen. Dieses Skript verwendet das Bild des Huskys, erhält ein Prognoseergebnis

aus dem vorgeschulten Modell, sucht dann in der Datei der Klassen danach und gibt ein Bildklassifizierungsergebnis zurück.

```
import mxnet as mx
import mxnet.contrib.onnx as onnx_mxnet
import numpy as np
from collections import namedtuple
from PIL import Image
import pickle

# Preprocess the image
img = Image.open("Siberian_Husky_bi-eyed_Flickr.jpg")
img = img.resize((224,224))
rgb_img = np.asarray(img, dtype=np.float32) - 128
bgr_img = rgb_img[..., [2,1,0]]
img_data = np.ascontiguousarray(np.rollaxis(bgr_img,2))
img_data = img_data[np.newaxis, :, :, :].astype(np.float32)

# Define the model's input
data_names = ['data']
Batch = namedtuple('Batch', data_names)

# Set the context to cpu or gpu
ctx = mx.cpu()

# Load the model
sym, arg, aux = onnx_mxnet.import_model("vgg16.onnx")
mod = mx.mod.Module(symbol=sym, data_names=data_names, context=ctx,
 label_names=None)
mod.bind(for_training=False, data_shapes=[(data_names[0],img_data.shape)],
 label_shapes=None)
mod.set_params(arg_params=arg, aux_params=aux, allow_missing=True,
 allow_extra=True)

# Run inference on the image
mod.forward(Batch([mx.nd.array(img_data)]))
predictions = mod.get_outputs()[0].asnumpy()
top_class = np.argmax(predictions)
print(top_class)
labels_dict = pickle.load(open("imagenet1000_clsid_to_human.pkl", "rb"))
print(labels_dict[top_class])
```

7. Führen Sie dann das Skript aus. Ein Ergebnis wie das folgende sollte angezeigt werden:

```
248
Eskimo dog, husky
```

Verwenden Sie Apache MXNet (Incubating) für Inferenz mit einem 50-Modell ResNet

So verwenden Sie ein vortrainiertes Apache MXNet-Modell (Inkubation) mit der Symbol-API für Bildinferenz mit MXNet

1. • (Option für Python 3) — Aktivieren Sie die Python 3 Apache MXNet (Incubating) - Umgebung:

   ```
   $ source activate mxnet_p36
   ```

   • (Option für Python 2) — Aktivieren Sie die Python 2-Umgebung Apache MXNet (Incubating):

   ```
   $ source activate mxnet_p27
   ```

2. Für die restlichen Schritte wird davon ausgegangen, dass Sie die `mxnet_p36`-Umgebung verwenden.
3. Verwenden Sie den von Ihnen bevorzugten Texteditor, um ein Skript mit folgendem Inhalt zu erstellen. Dieses Skript lädt die ResNet -50 Modelldateien (resnet-50-0000.params und resnet-50-symbol.json) und die Labelliste (synset.txt) herunter, lädt ein Katzenbild herunter, um ein Vorhersageergebnis aus dem vortrainierten Modell zu erhalten, sucht es dann in der Liste „Ergebnis in Labels“ und gibt ein Vorhersageergebnis zurück.

   ```
   import mxnet as mx
   import numpy as np

   path='http://data.mxnet.io/models/imagenet/'
   [mx.test_utils.download(path+'resnet/50-layers/resnet-50-0000.params'),
    mx.test_utils.download(path+'resnet/50-layers/resnet-50-symbol.json'),
    mx.test_utils.download(path+'synset.txt')]

   ctx = mx.cpu()

   with open('synset.txt', 'r') as f:
       labels = [l.rstrip() for l in f]

   sym, args, aux = mx.model.load_checkpoint('resnet-50', 0)
   ```

```
fname = mx.test_utils.download('https://github.com/dmlc/web-data/blob/master/mxnet/
doc/tutorials/python/predict_image/cat.jpg?raw=true')
img = mx.image.imread(fname)
# convert into format (batch, RGB, width, height)
img = mx.image.imresize(img, 224, 224) # resize
img = img.transpose((2, 0, 1)) # Channel first
img = img.expand_dims(axis=0) # batchify
img = img.astype(dtype='float32')
args['data'] = img

softmax = mx.nd.random_normal(shape=(1,))
args['softmax_label'] = softmax

exe = sym.bind(ctx=ctx, args=args, aux_states=aux, grad_req='null')

exe.forward()
prob = exe.outputs[0].asnumpy()
# print the top-5
prob = np.squeeze(prob)
a = np.argsort(prob)[::-1]
for i in a[0:5]:
    print('probability=%f, class=%s' %(prob[i], labels[i]))
```

4. Führen Sie dann das Skript aus. Ein Ergebnis wie das folgende sollte angezeigt werden:

```
probability=0.418679, class=n02119789 kit fox, Vulpes macrotis
probability=0.293495, class=n02119022 red fox, Vulpes vulpes
probability=0.029321, class=n02120505 grey fox, gray fox, Urocyon cinereoargenteus
probability=0.026230, class=n02124075 Egyptian cat
probability=0.022557, class=n02085620 Chihuahua
```

Verwenden von CNTK für Inferenz mit einem ONNX-Modell

> **Note**
>
> Wir schließen die Umgebungen CNTK, Caffe, Caffe2 und Theano Conda bei Version v28 nicht mehr in den AWS Deep Learning AMI -Start ein. Frühere Versionen von, die diese Umgebungen enthalten, werden weiterhin verfügbar sein. AWS Deep Learning AMI Wir

werden jedoch nur Updates für diese Umgebungen bereitstellen, wenn von der Open-Source-Community Sicherheitskorrekturen für diese Frameworks veröffentlicht werden.

> **Note**
>
> Das in diesem Tutorial verwendete Modell VGG-16 nutzt eine große Menge an Speicher. Bei der Auswahl Ihrer AWS Deep Learning AMI Instanz benötigen Sie möglicherweise eine Instanz mit mehr als 30 GB RAM.

Verwenden eines ONNX-Modells für Inferenzen mit CNTK

1. • (Option für Python 3) – Aktivieren Sie die Python 3-CNTK-Umgebung:

   ```
   $ source activate cntk_p36
   ```

   • (Option für Python 2) – Aktivieren Sie die Python 2-CNTK-Umgebung:

   ```
   $ source activate cntk_p27
   ```

2. Für die restlichen Schritte wird davon ausgegangen, dass Sie die `cntk_p36`-Umgebung verwenden.
3. Erstellen Sie eine neue Datei mit Ihrem Texteditor und verwenden Sie das folgende Programm in einem Skript, um die Datei im ONNX-Format in CNTK zu öffnen.

   ```
   import cntk as C
   # Import the Chainer model into CNTK via the CNTK import API
   z = C.Function.load("vgg16.onnx", device=C.device.cpu(), format=C.ModelFormat.ONNX)
   print("Loaded vgg16.onnx!")
   ```

   Nach dem Ausführen dieses Skripts hat CNTK das Modell geladen.
4. Sie können auch versuchen, Inferenzen mit CNTK auszuführen. Laden Sie zunächst ein Bild eines Huskys herunter.

   ```
   $ curl -O https://upload.wikimedia.org/wikipedia/commons/b/b5/Siberian_Husky_bi-eyed_Flickr.jpg
   ```

5. Als Nächstes laden Sie eine Liste der Klassen herunter, die mit diesem Modell funktionieren.

```
$ curl -O https://gist.githubusercontent.com/yrevar/6135f1bd8dcf2e0cc683/raw/
d133d61a09d7e5a3b36b8c111a8dd5c4b5d560ee/imagenet1000_clsid_to_human.pkl
```

6. Bearbeiten Sie das zuvor erstellte Skript, um die folgenden Inhalte zu erhalten. Diese neue Version verwendet das Bild des Huskys, erhält ein Prognoseergebnis, sucht dann nach diesem in der Datei der Klassen und gibt ein Prognoseergebnis zurück.

```
import cntk as C
import numpy as np
from PIL import Image
from IPython.core.display import display
import pickle

# Import the model into CNTK via the CNTK import API
z = C.Function.load("vgg16.onnx", device=C.device.cpu(), format=C.ModelFormat.ONNX)
print("Loaded vgg16.onnx!")
img = Image.open("Siberian_Husky_bi-eyed_Flickr.jpg")
img = img.resize((224,224))
rgb_img = np.asarray(img, dtype=np.float32) - 128
bgr_img = rgb_img[..., [2,1,0]]
img_data = np.ascontiguousarray(np.rollaxis(bgr_img,2))
predictions = np.squeeze(z.eval({z.arguments[0]:[img_data]}))
top_class = np.argmax(predictions)
print(top_class)
labels_dict = pickle.load(open("imagenet1000_clsid_to_human.pkl", "rb"))
print(labels_dict[top_class])
```

7. Führen Sie dann das Skript aus. Ein Ergebnis wie das folgende sollte angezeigt werden:

```
248
Eskimo dog, husky
```

Verwenden von Frameworks mit ONNX

Das Deep Learning AMI mit Conda unterstützt jetzt Open Neural Network Exchange (ONNX) - Modelle für einige Frameworks. Wählen Sie eines der unten aufgeführten Themen aus, um zu erfahren, wie Sie ONNX auf Ihrem Deep Learning-AMI mit Conda verwenden können.

Wenn Sie ein vorhandenes ONNX-Modell auf einem DLAMI verwenden möchten, finden Sie weitere Informationen unter. Verwenden Sie Apache MXNet (Incubating) für Inferenzen mit einem ONNX-Modell

Informationen zu ONNX

Open Neural Network Exchange (ONNX) ist ein Open-Source-Format zur Darstellung von Deep-Learning-Modellen. ONNX wird von Amazon Web Services, Microsoft, Facebook und mehreren anderen Partnern unterstützt. Sie können Deep Learning-Modelle mit jedem beliebigen Framework gestalten, schulen und bereitstellen. Der Vorteil von ONNX-Modellen besteht darin, dass sie einfach zwischen Frameworks verschoben werden können.

Das Deep Learning AMI mit Conda beleuchtet derzeit einige der ONNX-Funktionen in der folgenden Sammlung von Tutorials.

- Tutorial für Apache MXNet zu ONNX zu CNTK
- Chainer auf ONNX auf CNTK-Tutorial
- Chainer auf ONNX auf MXNet-Tutorial
- PyTorch zum ONNX zum CNTK-Tutorial
- PyTorch zu ONNX zu MXNet Tutorial

Sie können auch in ONNX-Projektdokumentation und -Tutorials nachsehen:

- ONNX-Projekt auf GitHub
- ONNX-Tutorials

Tutorial für Apache MXNet zu ONNX zu CNTK

> **Note**
>
> Wir schließen die Umgebungen CNTK, Caffe, Caffe2 und Theano Conda bei Version v28 nicht mehr in den AWS Deep Learning AMI -Start ein. Frühere Versionen von AWS Deep Learning AMI , die diese Umgebungen enthalten, werden weiterhin verfügbar sein. Wir werden jedoch nur Updates für diese Umgebungen bereitstellen, wenn von der Open-Source-Community Sicherheitskorrekturen für diese Frameworks veröffentlicht werden.

Übersicht über ONNX

Open Neural Network Exchange (ONNX) ist ein Open-Source-Format zur Darstellung von Deep-Learning-Modellen. ONNX wird von Amazon Web Services, Microsoft, Facebook und mehreren anderen Partnern unterstützt. Sie können Deep Learning-Modelle mit jedem beliebigen Framework gestalten, schulen und bereitstellen. Der Vorteil von ONNX-Modellen besteht darin, dass sie einfach zwischen Frameworks verschoben werden können.

Dieses Tutorial zeigt Ihnen, wie Sie das Deep Learning AMI mit Conda mit ONNX verwenden. Anhand dieser Schritte können Sie ein Modell schulen oder ein vorab geschultes Modell aus einem Framework laden, dieses Modell in ONNX exportieren und das Modell dann in ein anderes Framework importieren.

ONNX-Voraussetzungen

Um dieses ONNX-Tutorial verwenden zu können, benötigen Sie Zugriff auf ein Deep Learning-AMI mit Conda Version 12 oder höher. Weitere Informationen zu den ersten Schritten mit einem Deep Learning-AMI mit Conda finden Sie unter-Deep-Learning-AMI.

> **Important**
>
> In diesen Beispielen werden Funktionen verwendet, die möglicherweise bis zu 8 GB Speicher (oder mehr) benötigen. Wählen Sie unbedingt einen Instance-Typ mit genügend Speicherplatz aus.

Starten Sie eine Terminalsitzung mit Ihrem Deep Learning AMI mit Conda, um mit dem folgenden Tutorial zu beginnen.

Konvertieren eines Apache MXNet (Inkubation)-Modells zu ONNX, anschließend Laden des Modells in CNTK

Exportieren eines Modells aus Apache MXNet (Inkubation)

Sie können den neuesten MXNet-Build in einer oder beiden MXNet Conda-Umgebungen auf Ihrem Deep Learning-AMI mit Conda installieren.

1. • (Option für Python 3) – Aktivieren Sie die Python 3-MXNet-Umgebung:

```
$ source activate mxnet_p36
```

- (Option für Python 2) – Aktivieren Sie die Python 2-MXNet-Umgebung:

```
$ source activate mxnet_p27
```

2. Für die restlichen Schritte wird davon ausgegangen, dass Sie die `mxnet_p36`-Umgebung verwenden.
3. Laden Sie die Modelldateien herunter.

```
curl -O https://s3.amazonaws.com/onnx-mxnet/model-zoo/vgg16/vgg16-symbol.json
curl -O https://s3.amazonaws.com/onnx-mxnet/model-zoo/vgg16/vgg16-0000.params
```

4. Zum Exportieren der Modelldateien aus MXNet in das ONNX-Format erstellen Sie eine neue Datei mit Ihrem Texteditor und verwenden das folgende Programm in einem Skript.

```
import numpy as np
import mxnet as mx
from mxnet.contrib import onnx as onnx_mxnet
converted_onnx_filename='vgg16.onnx'

# Export MXNet model to ONNX format via MXNet's export_model API
converted_onnx_filename=onnx_mxnet.export_model('vgg16-symbol.json',
 'vgg16-0000.params', [(1,3,224,224)], np.float32, converted_onnx_filename)

# Check that the newly created model is valid and meets ONNX specification.
import onnx
model_proto = onnx.load(converted_onnx_filename)
onnx.checker.check_model(model_proto)
```

   Möglicherweise sehen Sie ein paar Warnmeldungen, Sie können sie jedoch derzeit auch einfach ignorieren. Nach dem Ausführen dieses Skripts sehen Sie die neu erstellte .onnx-Datei im selben Verzeichnis.
5. Sie verfügen jetzt über eine ONNX-Datei und können Versuchen, damit im folgenden Beispiel Inferenz auszuführen:

   - Verwenden von CNTK für Inferenz mit einem ONNX-Modell

ONNX-Tutorials

- Tutorial für Apache MXNet zu ONNX zu CNTK

- Chainer auf ONNX auf CNTK-Tutorial
- Chainer auf ONNX auf MXNet-Tutorial
- PyTorch zu ONNX zu MXNet Tutorial
- PyTorch zum ONNX zum CNTK-Tutorial

Chainer auf ONNX auf CNTK-Tutorial

> **Note**
>
> Wir schließen die Umgebungen CNTK, Caffe, Caffe2 und Theano Conda bei Version v28 nicht mehr in den AWS Deep Learning AMI -Start ein. Frühere Versionen von AWS Deep Learning AMI , die diese Umgebungen enthalten, werden weiterhin verfügbar sein. Wir werden jedoch nur Updates für diese Umgebungen bereitstellen, wenn von der Open-Source-Community Sicherheitskorrekturen für diese Frameworks veröffentlicht werden.

Übersicht über ONNX

Open Neural Network Exchange (ONNX) ist ein Open-Source-Format zur Darstellung von Deep-Learning-Modellen. ONNX wird von Amazon Web Services, Microsoft, Facebook und mehreren anderen Partnern unterstützt. Sie können Deep Learning-Modelle mit jedem beliebigen Framework gestalten, schulen und bereitstellen. Der Vorteil von ONNX-Modellen besteht darin, dass sie einfach zwischen Frameworks verschoben werden können.

Dieses Tutorial zeigt Ihnen, wie Sie das Deep Learning AMI mit Conda mit ONNX verwenden. Anhand dieser Schritte können Sie ein Modell schulen oder ein vorab geschultes Modell aus einem Framework laden, dieses Modell in ONNX exportieren und das Modell dann in ein anderes Framework importieren.

ONNX-Voraussetzungen

Um dieses ONNX-Tutorial verwenden zu können, benötigen Sie Zugriff auf ein Deep Learning-AMI mit Conda Version 12 oder höher. Weitere Informationen zu den ersten Schritten mit einem Deep Learning-AMI mit Conda finden Sie unter-Deep-Learning-AMI.

> **Important**
>
> In diesen Beispielen werden Funktionen verwendet, die möglicherweise bis zu 8 GB Speicher (oder mehr) benötigen. Wählen Sie unbedingt einen Instance-Typ mit genügend Speicherplatz aus.

Starten Sie eine Terminalsitzung mit Ihrem Deep Learning AMI mit Conda, um mit dem folgenden Tutorial zu beginnen.

Konvertieren eines Chainer-Modells in ONNX, anschließend Laden des Modells in CNTK

Zuerst aktivieren Sie die Chainer-Umgebung:

```
$ source activate chainer_p36
```

Erstellen Sie eine neue Datei mit Ihrem Texteditor und verwenden Sie das folgende Programm in einem Skript, um ein Modell aus der Modellsammlung von Chainer abzurufen. Exportieren Sie es dann in das ONNX-Format.

```
import numpy as np
import chainer
import chainercv.links as L
import onnx_chainer

# Fetch a vgg16 model
model = L.VGG16(pretrained_model='imagenet')

# Prepare an input tensor
x = np.random.rand(1, 3, 224, 224).astype(np.float32) * 255

# Run the model on the data
with chainer.using_config('train', False):
  chainer_out = model(x).array

# Export the model to a .onnx file
out = onnx_chainer.export(model, x, filename='vgg16.onnx')

# Check that the newly created model is valid and meets ONNX specification.
import onnx
```

```
model_proto = onnx.load("vgg16.onnx")
onnx.checker.check_model(model_proto)
```

Nach dem Ausführen dieses Skripts sehen Sie die neu erstellte .onnx-Datei im selben Verzeichnis.

Sie verfügen jetzt über eine ONNX-Datei und können Versuchen, damit im folgenden Beispiel Inferenz auszuführen:

- Verwenden von CNTK für Inferenz mit einem ONNX-Modell

ONNX-Tutorials

- Tutorial für Apache MXNet zu ONNX zu CNTK
- Chainer auf ONNX auf CNTK-Tutorial
- Chainer auf ONNX auf MXNet-Tutorial
- PyTorch zu ONNX zu MXNet Tutorial
- PyTorch zum ONNX zum CNTK-Tutorial

Chainer auf ONNX auf MXNet-Tutorial

Übersicht über ONNX

Open Neural Network Exchange (ONNX) ist ein Open-Source-Format zur Darstellung von Deep-Learning-Modellen. ONNX wird von Amazon Web Services, Microsoft, Facebook und mehreren anderen Partnern unterstützt. Sie können Deep Learning-Modelle mit jedem beliebigen Framework gestalten, schulen und bereitstellen. Der Vorteil von ONNX-Modellen besteht darin, dass sie einfach zwischen Frameworks verschoben werden können.

Dieses Tutorial zeigt Ihnen, wie Sie das Deep Learning AMI mit Conda mit ONNX verwenden. Anhand dieser Schritte können Sie ein Modell schulen oder ein vorab geschultes Modell aus einem Framework laden, dieses Modell in ONNX exportieren und das Modell dann in ein anderes Framework importieren.

ONNX-Voraussetzungen

Um dieses ONNX-Tutorial verwenden zu können, benötigen Sie Zugriff auf ein Deep Learning-AMI mit Conda Version 12 oder höher. Weitere Informationen zu den ersten Schritten mit einem Deep Learning-AMI mit Conda finden Sie unter-Deep-Learning-AMI.

> **Important**
>
> In diesen Beispielen werden Funktionen verwendet, die möglicherweise bis zu 8 GB Speicher (oder mehr) benötigen. Wählen Sie unbedingt einen Instance-Typ mit genügend Speicherplatz aus.

Starten Sie eine Terminalsitzung mit Ihrem Deep Learning AMI mit Conda, um mit dem folgenden Tutorial zu beginnen.

Konvertieren eines Chainer-Modells in ONNX, anschließend Laden des Modells in MXNet

Zuerst aktivieren Sie die Chainer-Umgebung:

```
$ source activate chainer_p36
```

Erstellen Sie eine neue Datei mit Ihrem Texteditor und verwenden Sie das folgende Programm in einem Skript, um ein Modell aus der Modellsammlung von Chainer abzurufen. Exportieren Sie es dann in das ONNX-Format.

```
import numpy as np
import chainer
import chainercv.links as L
import onnx_chainer

# Fetch a vgg16 model
model = L.VGG16(pretrained_model='imagenet')

# Prepare an input tensor
x = np.random.rand(1, 3, 224, 224).astype(np.float32) * 255

# Run the model on the data
with chainer.using_config('train', False):
  chainer_out = model(x).array

# Export the model to a .onnx file
out = onnx_chainer.export(model, x, filename='vgg16.onnx')

# Check that the newly created model is valid and meets ONNX specification.
import onnx
model_proto = onnx.load("vgg16.onnx")
```

```
onnx.checker.check_model(model_proto)
```

Nach dem Ausführen dieses Skripts sehen Sie die neu erstellte .onnx-Datei im selben Verzeichnis.

Sie verfügen jetzt über eine ONNX-Datei und können Versuchen, damit im folgenden Beispiel Inferenz auszuführen:

- Verwenden Sie Apache MXNet (Incubating) für Inferenzen mit einem ONNX-Modell

ONNX-Tutorials

- Tutorial für Apache MXNet zu ONNX zu CNTK
- Chainer auf ONNX auf CNTK-Tutorial
- Chainer auf ONNX auf MXNet-Tutorial
- PyTorch zu ONNX zu MXNet Tutorial
- PyTorch zum ONNX zum CNTK-Tutorial

PyTorch zum ONNX zum CNTK-Tutorial

> **Note**
>
> Wir schließen die Umgebungen CNTK, Caffe, Caffe2 und Theano Conda bei Version v28 nicht mehr in den AWS Deep Learning AMI -Start ein. Frühere Versionen von AWS Deep Learning AMI , die diese Umgebungen enthalten, werden weiterhin verfügbar sein. Wir werden jedoch nur Updates für diese Umgebungen bereitstellen, wenn von der Open-Source-Community Sicherheitskorrekturen für diese Frameworks veröffentlicht werden.

Übersicht über ONNX

Open Neural Network Exchange (ONNX) ist ein Open-Source-Format zur Darstellung von Deep-Learning-Modellen. ONNX wird von Amazon Web Services, Microsoft, Facebook und mehreren anderen Partnern unterstützt. Sie können Deep Learning-Modelle mit jedem beliebigen Framework gestalten, schulen und bereitstellen. Der Vorteil von ONNX-Modellen besteht darin, dass sie einfach zwischen Frameworks verschoben werden können.

Dieses Tutorial zeigt Ihnen, wie Sie das Deep Learning AMI mit Conda mit ONNX verwenden. Anhand dieser Schritte können Sie ein Modell schulen oder ein vorab geschultes Modell aus

einem Framework laden, dieses Modell in ONNX exportieren und das Modell dann in ein anderes Framework importieren.

ONNX-Voraussetzungen

Um dieses ONNX-Tutorial verwenden zu können, benötigen Sie Zugriff auf ein Deep Learning-AMI mit Conda Version 12 oder höher. Weitere Informationen zu den ersten Schritten mit einem Deep Learning-AMI mit Conda finden Sie unter-Deep-Learning-AMI.

> **Important**
>
> In diesen Beispielen werden Funktionen verwendet, die möglicherweise bis zu 8 GB Speicher (oder mehr) benötigen. Wählen Sie unbedingt einen Instance-Typ mit genügend Speicherplatz aus.

Starten Sie eine Terminalsitzung mit Ihrem Deep Learning AMI mit Conda, um mit dem folgenden Tutorial zu beginnen.

Konvertieren Sie ein PyTorch Modell in ONNX und laden Sie das Modell anschließend in CNTK

Aktivieren Sie zunächst die PyTorch Umgebung:

```
$ source activate pytorch_p36
```

Erstellen Sie eine neue Datei mit Ihrem Texteditor und verwenden Sie das folgende Programm in einem Skript, um ein Scheinmodell zu trainieren PyTorch, und exportieren Sie es dann in das ONNX-Format.

```
# Build a Mock Model in Pytorch with a convolution and a reduceMean layer\
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.autograd import Variable
import torch.onnx as torch_onnx

class Model(nn.Module):
    def __init__(self):
```

```
        super(Model, self).__init__()
        self.conv = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=(3,3),
 stride=1, padding=0, bias=False)

    def forward(self, inputs):
        x = self.conv(inputs)
        #x = x.view(x.size()[0], x.size()[1], -1)
        return torch.mean(x, dim=2)

# Use this an input trace to serialize the model
input_shape = (3, 100, 100)
model_onnx_path = "torch_model.onnx"
model = Model()
model.train(False)

# Export the model to an ONNX file
dummy_input = Variable(torch.randn(1, *input_shape))
output = torch_onnx.export(model,
                           dummy_input,
                           model_onnx_path,
                           verbose=False)
```

Nach dem Ausführen dieses Skripts sehen Sie die neu erstellte .onnx-Datei im selben Verzeichnis. Wechseln Sie jetzt in die CNTK Conda-Umgebung, um das Modell mit CNTK zu laden.

Aktivieren Sie als nächstes die CNTK-Umgebung:

```
$ source deactivate
$ source activate cntk_p36
```

Erstellen Sie eine neue Datei mit Ihrem Texteditor und verwenden Sie das folgende Programm in einem Skript, um die Datei im ONNX-Format in CNTK zu öffnen.

```
import cntk as C
# Import the PyTorch model into CNTK via the CNTK import API
z = C.Function.load("torch_model.onnx", device=C.device.cpu(),
 format=C.ModelFormat.ONNX)
```

Nach dem Ausführen dieses Skripts hat CNTK das Modell geladen.

Sie können es auch mit CNTK in ONNX exportieren, indem Sie Folgendes an Ihr vorheriges Skript anhängen und dieses dann ausführen.

```
# Export the model to ONNX via the CNTK export API
z.save("cntk_model.onnx", format=C.ModelFormat.ONNX)
```

ONNX-Tutorials

- Tutorial für Apache MXNet zu ONNX zu CNTK
- Chainer auf ONNX auf CNTK-Tutorial
- Chainer auf ONNX auf MXNet-Tutorial
- PyTorch zu ONNX zu MXNet Tutorial
- PyTorch zum ONNX zum CNTK-Tutorial

PyTorch zu ONNX zu MXNet Tutorial

Übersicht über ONNX

Open Neural Network Exchange (ONNX) ist ein Open-Source-Format zur Darstellung von Deep-Learning-Modellen. ONNX wird von Amazon Web Services, Microsoft, Facebook und mehreren anderen Partnern unterstützt. Sie können Deep Learning-Modelle mit jedem beliebigen Framework gestalten, schulen und bereitstellen. Der Vorteil von ONNX-Modellen besteht darin, dass sie einfach zwischen Frameworks verschoben werden können.

Dieses Tutorial zeigt Ihnen, wie Sie das Deep Learning AMI mit Conda mit ONNX verwenden. Anhand dieser Schritte können Sie ein Modell schulen oder ein vorab geschultes Modell aus einem Framework laden, dieses Modell in ONNX exportieren und das Modell dann in ein anderes Framework importieren.

ONNX-Voraussetzungen

Um dieses ONNX-Tutorial verwenden zu können, benötigen Sie Zugriff auf ein Deep Learning-AMI mit Conda Version 12 oder höher. Weitere Informationen zu den ersten Schritten mit einem Deep Learning-AMI mit Conda finden Sie unter-Deep-Learning-AMI.

> **Important**
>
> In diesen Beispielen werden Funktionen verwendet, die möglicherweise bis zu 8 GB Speicher (oder mehr) benötigen. Wählen Sie unbedingt einen Instance-Typ mit genügend Speicherplatz aus.

Starten Sie eine Terminalsitzung mit Ihrem Deep Learning AMI mit Conda, um mit dem folgenden Tutorial zu beginnen.

Ein PyTorch Modell in ONNX konvertieren und dann das Modell in MXNet laden

Aktivieren Sie zunächst die PyTorch Umgebung:

```
$ source activate pytorch_p36
```

Erstellen Sie eine neue Datei mit Ihrem Texteditor und verwenden Sie das folgende Programm in einem Skript, um ein Scheinmodell zu trainieren PyTorch, und exportieren Sie es dann in das ONNX-Format.

```
# Build a Mock Model in PyTorch with a convolution and a reduceMean layer
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.autograd import Variable
import torch.onnx as torch_onnx

class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=(3,3),
 stride=1, padding=0, bias=False)

    def forward(self, inputs):
        x = self.conv(inputs)
        #x = x.view(x.size()[0], x.size()[1], -1)
        return torch.mean(x, dim=2)

# Use this an input trace to serialize the model
input_shape = (3, 100, 100)
model_onnx_path = "torch_model.onnx"
model = Model()
model.train(False)

# Export the model to an ONNX file
dummy_input = Variable(torch.randn(1, *input_shape))
```

```
output = torch_onnx.export(model,
                           dummy_input,
                           model_onnx_path,
                           verbose=False)
print("Export of torch_model.onnx complete!")
```

Nach dem Ausführen dieses Skripts sehen Sie die neu erstellte .onnx-Datei im selben Verzeichnis. Wechseln Sie jetzt in die MXNet Conda-Umgebung, um das Modell mit MXNet zu laden.

Aktivieren Sie als nächstes die MXNet-Umgebung:

```
$ source deactivate
$ source activate mxnet_p36
```

Erstellen Sie eine neue Datei mit Ihrem Texteditor und verwenden Sie das folgende Programm in einem Skript, um die Datei im ONNX-Format in MXNet zu öffnen.

```
import mxnet as mx
from mxnet.contrib import onnx as onnx_mxnet
import numpy as np

# Import the ONNX model into MXNet's symbolic interface
sym, arg, aux = onnx_mxnet.import_model("torch_model.onnx")
print("Loaded torch_model.onnx!")
print(sym.get_internals())
```

Nach dem Ausführen dieses Skripts hat MXNet das Modell geladen und druckt einige grundlegende Modellinformationen.

ONNX-Tutorials

- Tutorial für Apache MXNet zu ONNX zu CNTK
- Chainer auf ONNX auf CNTK-Tutorial
- Chainer auf ONNX auf MXNet-Tutorial
- PyTorch zu ONNX zu MXNet Tutorial
- PyTorch zum ONNX zum CNTK-Tutorial

Modellbereitstellung

Im Folgenden sind die Modell-Serving-Optionen aufgeführt, die auf dem Deep Learning-AMI mit Conda installiert sind. Klicken Sie auf eine der Optionen, um zu erfahren, wie Sie diese verwenden.

Themen

- Modellserver für Apache MXNet (MMS)
- TensorFlow Servieren
- TorchServe

Modellserver für Apache MXNet (MMS)

Modellserver für Apache MXNet (MMS) ist ein flexibles Tool für die Bereitstellung von Deep Learning-Modellen, die aus Apache MXNet (Inkubation) oder in ein Open Neural Network Exchange (ONNX)-Modellformat exportiert wurden. MMS ist in dem DLAMI mit Conda vorinstalliert. In diesem Tutorial für MMS wird dargestellt, wie ein Bildklassifikationsmodell bereitgestellt wird.

Themen

- Bereitstellen eines Bildklassifizierungsmodells auf MMS
- Andere Beispiele
- Weitere Infos

Bereitstellen eines Bildklassifizierungsmodells auf MMS

Dieses Tutorial zeigt, wie man ein Bildklassifikationsmodell mit MMS bereitstellt. Das Modell wird über den MMS Model Zoo bereitgestellt und automatisch heruntergeladen, wenn Sie MMS starten. Sobald der Server läuft, nimmt er Prognoseanfragen entgegen. Wenn Sie ein Bild hochladen, in diesem Fall ein Bild eines Kätzchens, gibt der Server eine Prognose der besten 5 übereinstimmenden Klassen der 1.000 Klassen, für die das Modell trainiert wurde, zurück. Weitere Informationen zu den Modellen, wie sie trainiert wurden, und wie Sie diese testen, finden Sie im MMS Model Zoo.

Bereitstellen eines Beispiel-Bildklassifizierungsmodells auf MMS

1. Stellen Sie mit Conda eine Connect zu einer Amazon Elastic Compute Cloud (Amazon EC2) - Instance des Deep Learning AMI her.
2. Aktivieren Sie einen MXNet-Umgebung:

- Für MXNet und Keras 2 auf Python 3 mit CUDA 9.0 und MKL-DNN führen Sie diesen Befehl aus:

```
$ source activate mxnet_p36
```

- Für MXNet und Keras 2 auf Python 2 mit CUDA 9.0 und MKL-DNN führen Sie diesen Befehl aus:

```
$ source activate mxnet_p27
```

3. Führen Sie MMS mit dem folgenden Befehl aus. Durch das Hinzufügen von `> /dev/null` wird die Protokollausgabe während der Durchführung weiterer Tests stillgelegt.

```
$ mxnet-model-server --start > /dev/null
```

MMS wird jetzt auf Ihrem Host ausgeführt und nimmt Inferenz-Anfragen entgegen.

4. Verwenden Sie danach einen curl-Befehl, um MMS-Management-Endpunkte zu verwalten und das bereitzustellende Modell zu definieren.

```
$ curl -X POST "http://localhost:8081/models?url=https%3A%2F%2Fs3.amazonaws.com
%2Fmodel-server%2Fmodels%2Fsqueezenet_v1.1%2Fsqueezenet_v1.1.model"
```

5. MMS muss die Anzahl der Auftragnehmer kennen, die Sie verwenden möchten. Für diesen Test können Sie es mit 3 versuchen.

```
$ curl -v -X PUT "http://localhost:8081/models/squeezenet_v1.1?min_worker=3"
```

6. Laden Sie ein Bild eines Kätzchens herunter und senden Sie es an den MMS-Prognose-Endpunkt:

```
$ curl -O https://s3.amazonaws.com/model-server/inputs/kitten.jpg
$ curl -X POST http://127.0.0.1:8080/predictions/squeezenet_v1.1 -T kitten.jpg
```

Der Prognose-Endpunkt gibt eine Prognose in JSON zurück, ähnlich den folgenden Top-5-Prognosen, wobei das Bild eine Wahrscheinlichkeit von 94 % hat, eine Ägyptische Katze zu enthalten, gefolgt von einer Wahrscheinlichkeit von 5,5 %, dass es einen Luchs oder einen Puma zeigt:

```
{
 "prediction": [
  [{
    "class": "n02124075 Egyptian cat",
    "probability": 0.940
   },
   {
    "class": "n02127052 lynx, catamount",
    "probability": 0.055
   },
   {
    "class": "n02123045 tabby, tabby cat",
    "probability": 0.002
   },
   {
    "class": "n02123159 tiger cat",
    "probability": 0.0003
   },
   {
    "class": "n02123394 Persian cat",
    "probability": 0.0002
   }
  ]
 ]
}
```

7. Testen Sie weitere Bilder, oder wenn Sie die Tests abgeschlossen haben, stoppen Sie den Server:

```
$ mxnet-model-server --stop
```

Dieses Tutorial konzentriert sich auf die Bereitstellung des grundlegenden Modells. MMS unterstützt auch die Verwendung von Elastic Inference mit Modellbereitstellung. Weitere Informationen finden Sie unter Modellbereitstellung mit Amazon Elastic Inference.

Wenn Sie bereit sind, mehr über andere MMS-Funktionen zu erfahren, lesen Sie die MMS-Dokumentation unter. GitHub

Andere Beispiele

MMS bietet eine Vielzahl von Beispielen, die Sie auf Ihrem DLAMI ausführen können. Sie können sie im MMS-Projekt-Repository anzeigen.

Weitere Infos

Weitere MMS-Dokumentation, einschließlich der Einrichtung von MMS mit Docker — oder wie Sie die neuesten MMS-Funktionen nutzen können, finden Sie auf der MMS-Projektseite unter. GitHub

TensorFlow Servieren

TensorFlow Serving ist ein flexibles, leistungsstarkes Serversystem für Modelle des maschinellen Lernens.

Das `tensorflow-serving-api` ist mit Deep Learning AMI mit Conda vorinstalliert! Ein Beispielskript für das Trainieren, Exportieren und Bereitstellen eines MNIST-Modells finden Sie in `~/examples/tensorflow-serving/`.

Um eines dieser Beispiele auszuführen, stellen Sie zunächst mit Conda eine Verbindung zu Ihrem Deep Learning-AMI her und aktivieren Sie die TensorFlow Umgebung.

```
$ source activate tensorflow_p37
```

Jetzt setzen Sie die Verzeichnisse auf den Ordner mit den Beispiel-Skripts.

```
$ cd ~/examples/tensorflow-serving/
```

Bereitstellen eines vorgeschulten Inception-Modells

Im Folgenden finden Sie ein Beispiel, mit dem Sie die Bereitstellung verschiedener Modelle wie Inception testen können. In der Regel benötigen Sie ein servables Modell und Client-Skripte, die bereits auf Ihr DLAMI heruntergeladen wurden.

Bereitstellen und Testen von Inference mit einem Inception-Modell

1. Laden Sie das Modell herunter.

   ```
   $ curl -O https://s3-us-west-2.amazonaws.com/tf-test-models/INCEPTION.zip
   ```

2. Entpacken Sie das Modell.

```
$ unzip INCEPTION.zip
```

3. Laden Sie das Bild eines Huskys herunter.

```
$ curl -O https://upload.wikimedia.org/wikipedia/commons/b/b5/Siberian_Husky_bi-
eyed_Flickr.jpg
```

4. Starten Sie den Server. Beachten Sie, dass Sie in Amazon Linux das Verzeichnis, das für `model_base_path` verwendet wird, von `/home/ubuntu` in `/home/ec2-user` ändern müssen.

```
$ tensorflow_model_server --model_name=INCEPTION --model_base_path=/home/ubuntu/
examples/tensorflow-serving/INCEPTION/INCEPTION --port=9000
```

5. Wenn der Server im Vordergrund läuft, müssen Sie eine weitere Terminalsitzung starten, um fortzufahren. Öffnen Sie ein neues Terminal und aktivieren Sie es TensorFlow mit`source activate tensorflow_p37`. Verwenden Sie anschließend den von Ihnen bevorzugten Texteditor, um ein Skript mit folgendem Inhalt zu erstellen. Geben Sie ihr den Namen `inception_client.py`. Dieses Skript verwendet einen Image-Dateinamen als Parameter und ruft ein Voraussageergebnis von dem vorgeschulten Modell ab.

```
from __future__ import print_function

import grpc
import tensorflow as tf
import argparse

from tensorflow_serving.apis import predict_pb2
from tensorflow_serving.apis import prediction_service_pb2_grpc

parser = argparse.ArgumentParser(
    description='TF Serving Test',
    formatter_class=argparse.ArgumentDefaultsHelpFormatter
)
parser.add_argument('--server_address', default='localhost:9000',
                    help='Tenforflow Model Server Address')
parser.add_argument('--image', default='Siberian_Husky_bi-eyed_Flickr.jpg',
                    help='Path to the image')
args = parser.parse_args()


def main():
```

```
  channel = grpc.insecure_channel(args.server_address)
  stub = prediction_service_pb2_grpc.PredictionServiceStub(channel)
  # Send request
  with open(args.image, 'rb') as f:
    # See prediction_service.proto for gRPC request/response details.
    request = predict_pb2.PredictRequest()
    request.model_spec.name = 'INCEPTION'
    request.model_spec.signature_name = 'predict_images'

    input_name = 'images'
    input_shape = [1]
    input_data = f.read()
    request.inputs[input_name].CopyFrom(
      tf.make_tensor_proto(input_data, shape=input_shape))

    result = stub.Predict(request, 10.0)  # 10 secs timeout
    print(result)

  print("Inception Client Passed")


if __name__ == '__main__':
  main()
```

6. Führen Sie nun das Skript aus und geben Sie Server-Standort sowie Port und Dateinamen des Husky-Bilds als Parameter weiter.

```
$ python3 inception_client.py --server=localhost:9000 --image Siberian_Husky_bi-
eyed_Flickr.jpg
```

Schulen und Bereitstellen eines MNIST-Modells

Für dieses Tutorial exportieren wir ein Modell und stellen es mit der Anwendung `tensorflow_model_server` bereit. Schließlich können Sie den Modell-Server mit einem Beispiel-Client-Skript testen.

Führen Sie das Skript aus, das ein MNIST Modell trainiert und exportiert. Als einziges Argument für das Skript müssen Sie einen Ordnerspeicherort angeben, wo es das Modell speichern kann. Hier können wir es einfach in `mnist_model` ablegen. Das Skript erstellt den Ordner für Sie.

```
$ python mnist_saved_model.py /tmp/mnist_model
```

Haben Sie Geduld, da dieses Skript kann eine Weile brauchen kann, bevor es etwas ausgibt. Wenn das Training abgeschlossen ist und das Modell schließlich exportiert wurde, sollten Sie Folgendes sehen:

```
Done training!
Exporting trained model to mnist_model/1
Done exporting!
```

Ihr nächster Schritt besteht darin, den `tensorflow_model_server` auszuführen, um das exportierte Modell bereitzustellen.

```
$ tensorflow_model_server --port=9000 --model_name=mnist --model_base_path=/tmp/
mnist_model
```

Ein Client-Skript wird bereitgestellt, damit Sie den Server testen können.

Wenn Sie einen Test durchführen, müssen Sie ein neues Terminal-Fenster öffnen.

```
$ python mnist_client.py --num_tests=1000 --server=localhost:9000
```

Weitere Funktionen und Beispiele

Wenn Sie mehr über TensorFlow Serving erfahren möchten, besuchen Sie die TensorFlow Website.

Sie können TensorFlow Serving auch mit Amazon Elastic Inference verwenden. Weitere Informationen finden Sie in der Anleitung zur Verwendung von Elastic Inference with TensorFlow Serving.

TorchServe

TorchServe ist ein flexibles Tool zur Bereitstellung von Deep-Learning-Modellen, aus PyTorch denen exportiert wurden. TorchServe ist mit dem Deep Learning AMI mit Conda ab Version 34 vorinstalliert.

Weitere Informationen zur Verwendung finden Sie in der TorchServe Dokumentation zu Model Server. PyTorch

Topics

Stellen Sie ein Bildklassifizierungsmodell bereit auf TorchServe

Dieses Tutorial zeigt, wie Sie ein Bildklassifizierungsmodell mit bereitstellen TorchServe. Es verwendet ein DenseNet -161-Modell, das von bereitgestellt wird PyTorch. Sobald der Server läuft,

wartet er auf Vorhersageanfragen. Wenn Sie ein Bild hochladen, in diesem Fall das Bild eines Kätzchens, gibt der Server eine Vorhersage der fünf am besten passenden Klassen aus den Klassen zurück, für die das Modell trainiert wurde.

Um ein Beispiel für ein Bildklassifizierungsmodell bereitzustellen TorchServe

1. Stellen Sie eine Connect zu einer Amazon Elastic Compute Cloud (Amazon EC2) -Instance mit Deep Learning AMI mit Conda v34 oder höher her.
2. Aktivieren Sie die Umgebung. `pytorch_latest_p36`

   ```
   source activate pytorch_latest_p36
   ```

3. Klonen Sie das TorchServe Repository und erstellen Sie dann ein Verzeichnis zum Speichern Ihrer Modelle.

   ```
   git clone https://github.com/pytorch/serve.git
   mkdir model_store
   ```

4. Archivieren Sie das Modell mit dem Modellarchiver. Der `extra-files` Parameter verwendet eine Datei aus dem `TorchServe` Repo. Aktualisieren Sie daher den Pfad, falls erforderlich. Weitere Informationen zum Modellarchiver finden Sie unter Torch Model Archiver for. TorchServe

   ```
   wget https://download.pytorch.org/models/densenet161-8d451a50.pth
   torch-model-archiver --model-name densenet161 --version 1.0 --model-file ./
   serve/examples/image_classifier/densenet_161/model.py --serialized-file
    densenet161-8d451a50.pth --export-path model_store --extra-files ./serve/examples/
   image_classifier/index_to_name.json --handler image_classifier
   ```

5. Ausführen TorchServe , um einen Endpunkt zu starten. Durch das `> /dev/null` Hinzufügen wird die Protokollausgabe gestoppt.

   ```
   torchserve --start --ncs --model-store model_store --models densenet161.mar > /dev/
   null
   ```

6. Laden Sie ein Bild eines Kätzchens herunter und senden Sie es an den TorchServe Predict-Endpunkt:

   ```
   curl -O https://s3.amazonaws.com/model-server/inputs/kitten.jpg
   curl http://127.0.0.1:8080/predictions/densenet161 -T kitten.jpg
   ```

Der Vorhersage-Endpunkt gibt eine Vorhersage in JSON zurück, die den folgenden fünf wichtigsten Vorhersagen ähnelt, wobei das Bild mit einer Wahrscheinlichkeit von 47% eine ägyptische Katze enthält, gefolgt von einer Wahrscheinlichkeit von 46%, dass es sich um eine Tabbykatze handelt.

```
{
 "tiger_cat": 0.46933576464653015,
 "tabby": 0.463387668132782,
 "Egyptian_cat": 0.0645613968372345,
 "lynx": 0.0012828196631744504,
 "plastic_bag": 0.00023323058849200606
}
```

7. Wenn Sie mit dem Testen fertig sind, beenden Sie den Server:

```
torchserve --stop
```

Andere Beispiele

TorchServe hat eine Vielzahl von Beispielen, die Sie auf Ihrer DLAMI-Instanz ausführen können. Sie können sie auf der Seite mit den Beispielen für das TorchServe Projekt-Repository einsehen.

Mehr Informationen

Weitere TorchServe Dokumentation, einschließlich der Einrichtung TorchServe mit Docker und der neuesten TorchServe Funktionen, finden Sie auf GitHubder TorchServe Projektseite unter.

Aktualisieren Sie Ihr DLAMI

Hier finden Sie Informationen zum Upgrade Ihres DLAMI und Tipps zur Aktualisierung der Software auf Ihrem DLAMI.

Themen

Durchführen eines Upgrades auf eine neue DLAMI-Version

Die Systemabbilder von DLAMI werden regelmäßig aktualisiert, um die Vorteile neuer Deep-Learning-Framework-Versionen, CUDA- und anderer Softwareupdates sowie der Leistungsoptimierung zu nutzen. Wenn Sie ein DLAMI schon länger verwenden und ein Update nutzen möchten, müssten Sie eine neue Instance starten. Sie müssten auch manuell Datensätze, Kontrollpunkte oder andere wichtige Daten übertragen. Stattdessen können Sie Amazon EBS verwenden, um Ihre Daten zu speichern und sie an einen neuen DLAMI anzuhängen. Auf diese Weise können Sie häufig Aktualisierungen durchführen und dabei die Zeit, die zum Übertragen Ihrer Daten erforderlich ist, minimieren.

> **Note**
>
> Wenn Sie Amazon EBS-Volumes anhängen und zwischen DLAMIs verschieben, müssen Sie sowohl die DLAMIs als auch das neue Volume in derselben Availability Zone haben.

1. Verwenden Sie die Amazon EC2-Konsole, um ein neues Amazon EBS-Volume zu erstellen. Eine detaillierte Anleitung finden Sie unter Erstellen eines Amazon EBS-Volumes.
2. Zuordnen eines Amazon EBS-Volumes zu Ihrem vorhandenen DLAMI Eine ausführliche Anleitung finden Sie unter Anhängen eines Amazon EBS-Volumes.
3. Übertragen Sie Ihre Daten, wie z. B. Datensätze, Kontrollpunkte und Konfigurationsdateien.
4. Starte einen DLAMI. Detaillierte Anweisungen finden Sie unter Einen DLAMI starten und konfigurieren.
5. Trennen Sie das Amazon EBS-Volume von Ihrem alten DLAMI. Eine detaillierte Anleitung finden Sie unter Trennen eines Amazon EBS-Volumes.

6. Zuordnen eines Amazon EBS-Volumes zu Ihrem neuen DLAMI Folgen Sie den Anweisungen aus Schritt 2 zum Verknüpfen des Volumes.
7. Nachdem Sie überprüft haben, dass Ihre Daten auf Ihrem neuen DLAMI verfügbar sind, beenden und beenden Sie Ihr altes DLAMI. Detaillierte Anweisungen zur Bereinigung finden Sie unter Bereinigen.

Tipps für Software-Updates

Von Zeit zu Zeit möchten Sie möglicherweise die Software auf Ihrem DLAMI manuell aktualisieren. Allgemein empfiehlt es sich, `pip` zum Aktualisieren von Python-Paketen zu verwenden. Sie sollten es auch verwendenpip, um Pakete innerhalb einer Conda-Umgebung auf dem Deep Learning-AMI mit Conda zu aktualisieren. Weitere Anweisungen zum Upgrade und zur Installation finden Sie auf der jeweiligen Framework- oder Software-Website.

> **Note**
>
> Wir können nicht garantieren, dass ein Paket-Update erfolgreich sein wird. Der Versuch, ein Paket in einer Umgebung mit inkompatiblen Abhängigkeiten zu aktualisieren, kann zu einem Fehler führen. In einem solchen Fall sollten Sie sich an den Bibliotheksbetreuer wenden, um zu erfahren, ob es möglich ist, die Paketabhängigkeiten zu aktualisieren. Alternativ können Sie versuchen, die Umgebung so zu ändern, dass das Update möglich ist. Diese Änderung wird jedoch wahrscheinlich das Entfernen oder Aktualisieren vorhandener Pakete bedeuten, was bedeutet, dass wir die Stabilität dieser Umgebung nicht mehr garantieren können.

Wenn Sie daran interessiert sind, den neuesten Hauptzweig eines bestimmten Pakets auszuführen, aktivieren Sie die entsprechende Umgebung und fügen`--pre` Sie dann am Ende des`pip install --upgrade` Befehls hinzu. Beispiel:

```
source activate mxnet_p36
pip install --upgrade mxnet --pre
```

DasAWS Deep Learning AMI wird mit vielen Conda-Umgebungen und vielen vorinstallierten Paketen geliefert. Aufgrund der Anzahl der vorinstallierten Pakete ist es schwierig, eine Reihe von Paketen zu finden, die garantiert kompatibel sind. Möglicherweise wird die Warnung „Die Umgebung ist inkonsistent, bitte überprüfen Sie den Paketplan sorgfältig“. DLAMI stellt sicher, dass alle von DLAMI

bereitgestellten Umgebungen korrekt sind, kann jedoch nicht garantieren, dass alle vom Benutzer installierten Pakete korrekt funktionieren.

Sicherheit in AWS Deep Learning AMI

Cloud-Sicherheit hat AWS höchste Priorität. Als AWS Kunde profitieren Sie von einer Rechenzentrums- und Netzwerkarchitektur, die darauf ausgelegt sind, die Anforderungen der sicherheitssensibelsten Unternehmen zu erfüllen.

Sicherheit ist eine gemeinsame Verantwortung von Ihnen AWS und Ihnen. Das Modell der geteilten Verantwortung beschreibt dies als Sicherheit der Cloud selbst und Sicherheit in der Cloud:

- Sicherheit der Cloud — AWS ist verantwortlich für den Schutz der Infrastruktur, die AWS Dienste in der AWS Cloud ausführt. AWS bietet Ihnen auch Dienste, die Sie sicher nutzen können. Externe Prüfer testen und verifizieren regelmäßig die Wirksamkeit unserer Sicherheitsmaßnahmen im Rahmen der AWS . Weitere Informationen zu den Compliance-Programmen, die für DLAMI gelten, finden Sie unter AWS Services in Scope by Compliance Program AWS Program.
- Sicherheit in der Cloud — Ihre Verantwortung richtet sich nach dem AWS Dienst, den Sie nutzen. Sie sind auch für andere Faktoren verantwortlich, etwa für die Vertraulichkeit Ihrer Daten, für die Anforderungen Ihres Unternehmens und für die geltenden Gesetze und Vorschriften.

Diese Dokumentation hilft Ihnen zu verstehen, wie Sie das Modell der gemeinsamen Verantwortung bei der Verwendung von DLAMI anwenden können. In den folgenden Themen erfahren Sie, wie Sie DLAMI konfigurieren, um Ihre Sicherheits- und Compliance-Ziele zu erreichen. Sie lernen auch, wie Sie andere AWS Dienste nutzen können, die Ihnen helfen, Ihre DLAMI-Ressourcen zu überwachen und zu sichern.

Weitere Informationen finden Sie unter Sicherheit in Amazon EC2.

Themen

Datenschutz in AWS Deep Learning AMI

Das AWS Modell der mit gilt für den Datenschutz in AWS Deep Learning AMI. Wie in diesem Modell beschrieben, AWS ist es für den Schutz der globalen Infrastruktur verantwortlich, auf der alle Systeme laufen AWS Cloud. Sie sind dafür verantwortlich, die Kontrolle über Ihre in dieser Infrastruktur gehosteten Inhalte zu behalten. Sie sind auch für die Sicherheitskonfiguration und die Verwaltungsaufgaben für die von Ihnen verwendeten AWS-Services verantwortlich. Weitere Informationen zum Datenschutz finden Sie unter Häufig gestellte Fragen zum Datenschutz. Informationen zum Datenschutz in Europa finden Sie im Blog-Beitrag AWS -Modell der geteilten Verantwortung und in der DSGVO im AWS -Sicherheitsblog.

Aus Datenschutzgründen empfehlen wir, dass Sie AWS-Konto Anmeldeinformationen schützen und einzelne Benutzer mit AWS IAM Identity Center oder AWS Identity and Access Management (IAM) einrichten. So erhält jeder Benutzer nur die Berechtigungen, die zum Durchführen seiner Aufgaben erforderlich sind. Außerdem empfehlen wir, die Daten mit folgenden Methoden schützen:

- Verwenden Sie für jedes Konto die Multi-Faktor Authentifizierung (MFA).
- Verwenden Sie SSL/TLS, um mit Ressourcen zu kommunizieren. AWS Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Richten Sie die API und die Protokollierung von Benutzeraktivitäten mit ein. AWS CloudTrail
- Verwenden Sie AWS Verschlüsselungslösungen zusammen mit allen darin enthaltenen Standardsicherheitskontrollen AWS-Services.
- Verwenden Sie erweiterte verwaltete Sicherheitsservices wie Amazon Macie, die dabei helfen, in Amazon S3 gespeicherte persönliche Daten zu erkennen und zu schützen.
- Wenn Sie für den Zugriff AWS über eine Befehlszeilenschnittstelle oder eine API FIPS 140-2-validierte kryptografische Module benötigen, verwenden Sie einen FIPS-Endpunkt. Weitere Informationen über verfügbare FIPS-Endpunkte finden Sie unter Federal Information Processing Standard (FIPS) 140-2.

Wir empfehlen dringend, in Freitextfeldern, z. B. im Feld Name, keine vertraulichen oder sensiblen Informationen wie die E-Mail-Adressen Ihrer Kunden einzugeben. Dies gilt auch, wenn Sie mit DLAMI oder anderen AWS-Services über die Konsole AWS CLI, API oder AWS SDKs arbeiten. Alle Daten, die Sie in Tags oder Freitextfelder eingeben, die für Namen verwendet werden, können für Abrechnungs- oder Diagnoseprotokolle verwendet werden. Wenn Sie eine URL für einen externen Server bereitstellen, empfehlen wir dringend, keine Anmeldeinformationen zur Validierung Ihrer Anforderung an den betreffenden Server in die URL einzuschließen.

Identity and Access Management in AWS Deep Learning AMI

AWS Identity and Access Management (IAM) hilft einem Administrator AWS-Service , den Zugriff auf Ressourcen sicher zu kontrollieren. AWS IAM-Administratoren kontrollieren, wer authentifiziert (angemeldet) und autorisiert werden kann (über Berechtigungen verfügt), um DLAMI-Ressourcen zu verwenden. IAM ist ein Programm AWS-Service , das Sie ohne zusätzliche Kosten nutzen können.

Weitere Informationen zum Identity and Access Management finden Sie unter Identity and Access Management für Amazon EC2.

Themen

- Authentifizierung mit Identitäten
- Verwalten des Zugriffs mit Richtlinien
- IAM mit Amazon EMR

Authentifizierung mit Identitäten

Authentifizierung ist die Art und Weise, wie Sie sich AWS mit Ihren Identitätsdaten anmelden. Sie müssen als IAM-Benutzer authentifiziert (angemeldet AWS) sein oder eine IAM-Rolle annehmen. Root-Benutzer des AWS-Kontos

Sie können sich AWS als föderierte Identität anmelden, indem Sie Anmeldeinformationen verwenden, die über eine Identitätsquelle bereitgestellt wurden. AWS IAM Identity Center (IAM Identity Center) -Benutzer, die Single Sign-On-Authentifizierung Ihres Unternehmens und Ihre Google- oder Facebook-Anmeldeinformationen sind Beispiele für föderierte Identitäten. Wenn Sie sich als Verbundidentität anmelden, hat der Administrator vorher mithilfe von IAM-Rollen einen Identitätsverbund eingerichtet. Wenn Sie über den Verbund darauf zugreifen AWS , übernehmen Sie indirekt eine Rolle.

Je nachdem, welcher Benutzertyp Sie sind, können Sie sich beim AWS Management Console oder beim AWS Zugangsportal anmelden. Weitere Informationen zur Anmeldung finden Sie AWS unter So melden Sie sich bei Ihrem an AWS-Konto im AWS-Anmeldung Benutzerhandbuch.

Wenn Sie AWS programmgesteuert darauf zugreifen, AWS stellt es ein Software Development Kit (SDK) und eine Befehlszeilenschnittstelle (CLI) bereit, um Ihre Anfragen mithilfe Ihrer Anmeldeinformationen kryptografisch zu signieren. Wenn Sie keine AWS Tools verwenden, müssen Sie Anfragen selbst signieren. Weitere Informationen zur Verwendung der empfohlenen Methode,

um Anfragen selbst zu signieren, finden Sie im IAM-Benutzerhandbuch unter AWS API-Anfragen signieren.

Unabhängig von der verwendeten Authentifizierungsmethode müssen Sie möglicherweise zusätzliche Sicherheitsinformationen angeben. AWS Empfiehlt beispielsweise, die Multi-Faktor-Authentifizierung (MFA) zu verwenden, um die Sicherheit Ihres Kontos zu erhöhen. Weitere Informationen finden Sie unter Multi-Faktor-Authentifizierung im AWS IAM Identity Center - Benutzerhandbuch und Verwenden der Multi-Faktor-Authentifizierung (MFA) in AWS im IAM-Benutzerhandbuch.

AWS-Konto Root-Benutzer

Wenn Sie einen erstellen AWS-Konto, beginnen Sie mit einer Anmeldeidentität, die vollständigen Zugriff auf alle AWS-Services Ressourcen im Konto hat. Diese Identität wird als AWS-Konto Root-Benutzer bezeichnet. Sie können darauf zugreifen, indem Sie sich mit der E-Mail-Adresse und dem Passwort anmelden, mit denen Sie das Konto erstellt haben. Wir raten ausdrücklich davon ab, den Root-Benutzer für Alltagsaufgaben zu verwenden. Schützen Sie Ihre Root-Benutzer-Anmeldeinformationen und verwenden Sie diese, um die Aufgaben auszuführen, die nur der Root-Benutzer ausführen kann. Eine vollständige Liste der Aufgaben, für die Sie sich als Root-Benutzer anmelden müssen, finden Sie unter Aufgaben, die Root-Benutzer-Anmeldeinformationen erfordern im IAM-Benutzerhandbuch.

IAM-Benutzer und -Gruppen

Ein IAM-Benutzer ist eine Identität innerhalb von Ihnen AWS-Konto , die über spezifische Berechtigungen für eine einzelne Person oder Anwendung verfügt. Wenn möglich, empfehlen wir, temporäre Anmeldeinformationen zu verwenden, anstatt IAM-Benutzer zu erstellen, die langfristige Anmeldeinformationen wie Passwörter und Zugriffsschlüssel haben. Bei speziellen Anwendungsfällen, die langfristige Anmeldeinformationen mit IAM-Benutzern erfordern, empfehlen wir jedoch, die Zugriffsschlüssel zu rotieren. Weitere Informationen finden Sie unter Regelmäßiges Rotieren von Zugriffsschlüsseln für Anwendungsfälle, die langfristige Anmeldeinformationen erfordern im IAM-Benutzerhandbuch.

Eine IAM-Gruppe ist eine Identität, die eine Sammlung von IAM-Benutzern angibt. Sie können sich nicht als Gruppe anmelden. Mithilfe von Gruppen können Sie Berechtigungen für mehrere Benutzer gleichzeitig angeben. Gruppen vereinfachen die Verwaltung von Berechtigungen, wenn es zahlreiche Benutzer gibt. Sie könnten beispielsweise einer Gruppe mit dem Namen IAMAdmins Berechtigungen zum Verwalten von IAM-Ressourcen erteilen.

Benutzer unterscheiden sich von Rollen. Ein Benutzer ist einer einzigen Person oder Anwendung eindeutig zugeordnet. Eine Rolle kann von allen Personen angenommen werden, die sie benötigen. Benutzer besitzen dauerhafte Anmeldeinformationen. Rollen stellen temporäre Anmeldeinformationen bereit. Weitere Informationen finden Sie unter Erstellen eines IAM-Benutzers (anstatt einer Rolle) im IAM-Benutzerhandbuch.

IAM-Rollen

Eine IAM-Rolle ist eine Identität innerhalb Ihres Unternehmens AWS-Konto , die über bestimmte Berechtigungen verfügt. Sie ist einem IAM-Benutzer vergleichbar, ist aber nicht mit einer bestimmten Person verknüpft. Sie können vorübergehend eine IAM-Rolle in der übernehmen, AWS Management Console indem Sie die Rollen wechseln. Sie können eine Rolle übernehmen, indem Sie eine AWS CLI oder AWS API-Operation aufrufen oder eine benutzerdefinierte URL verwenden. Weitere Informationen zu Methoden für die Verwendung von Rollen finden Sie unter Verwenden von IAM-Rollen im IAM-Benutzerhandbuch.

IAM-Rollen mit temporären Anmeldeinformationen sind in folgenden Situationen hilfreich:

- Verbundbenutzerzugriff: Um einer Verbundidentität Berechtigungen zuzuweisen, erstellen Sie eine Rolle und definieren Berechtigungen für die Rolle. Wird eine Verbundidentität authentifiziert, so wird die Identität der Rolle zugeordnet und erhält die von der Rolle definierten Berechtigungen. Informationen zu Rollen für den Verbund finden Sie unter Erstellen von Rollen für externe Identitätsanbieter im IAM-Benutzerhandbuch. Wenn Sie IAM Identity Center verwenden, konfigurieren Sie einen Berechtigungssatz. Wenn Sie steuern möchten, worauf Ihre Identitäten nach der Authentifizierung zugreifen können, korreliert IAM Identity Center den Berechtigungssatz mit einer Rolle in IAM. Informationen zu Berechtigungssätzen finden Sie unter Berechtigungssätze im AWS IAM Identity Center -Benutzerhandbuch.
- Temporäre IAM-Benutzerberechtigungen: Ein IAM-Benutzer oder eine -Rolle kann eine IAM-Rolle übernehmen, um vorübergehend andere Berechtigungen für eine bestimmte Aufgabe zu erhalten.
- Kontoübergreifender Zugriff – Sie können eine IAM-Rolle verwenden, um einem vertrauenswürdigen Prinzipal in einem anderen Konto den Zugriff auf Ressourcen in Ihrem Konto zu ermöglichen. Rollen stellen die primäre Möglichkeit dar, um kontoübergreifendem Zugriff zu gewähren. Bei einigen können Sie AWS-Services jedoch eine Richtlinie direkt an eine Ressource anhängen (anstatt eine Rolle als Proxy zu verwenden). Informationen zu den Unterschieden zwischen Rollen und ressourcenbasierten Richtlinien für den kontoübergreifenden Zugriff finden Sie unter So unterscheiden sich IAM-Rollen von ressourcenbasierten Richtlinien im IAM-Benutzerhandbuch.

- Serviceübergreifender Zugriff — Einige AWS-Services verwenden Funktionen in anderen AWS-Services. Wenn Sie beispielsweise einen Aufruf in einem Service tätigen, führt dieser Service häufig Anwendungen in Amazon EC2 aus oder speichert Objekte in Amazon S3. Ein Dienst kann dies mit den Berechtigungen des aufrufenden Prinzipals mit einer Servicerolle oder mit einer serviceverknüpften Rolle tun.
  - Forward Access Sessions (FAS) — Wenn Sie einen IAM-Benutzer oder eine IAM-Rolle verwenden, um Aktionen auszuführen AWS, gelten Sie als Principal. Bei einigen Services könnte es Aktionen geben, die dann eine andere Aktion in einem anderen Service auslösen. FAS verwendet die Berechtigungen des Prinzipals, der einen aufruft AWS-Service, in Kombination mit der Anfrage, Anfragen an AWS-Service nachgelagerte Dienste zu stellen. FAS-Anfragen werden nur gestellt, wenn ein Dienst eine Anfrage erhält, für deren Abschluss Interaktionen mit anderen AWS-Services oder Ressourcen erforderlich sind. In diesem Fall müssen Sie über Berechtigungen zum Ausführen beider Aktionen verfügen. Einzelheiten zu den Richtlinien für FAS-Anfragen finden Sie unter Zugriffssitzungen weiterleiten.
  - Servicerolle: Eine Servicerolle ist eine IAM-Rolle, die ein Service übernimmt, um Aktionen in Ihrem Namen auszuführen. Ein IAM-Administrator kann eine Servicerolle innerhalb von IAM erstellen, ändern und löschen. Weitere Informationen finden Sie unter Erstellen einer Rolle zum Delegieren von Berechtigungen an einen AWS-Service im IAM-Benutzerhandbuch.
  - Dienstbezogene Rolle — Eine dienstbezogene Rolle ist eine Art von Servicerolle, die mit einer verknüpft ist. AWS-Service Der Service kann die Rolle übernehmen, um eine Aktion in Ihrem Namen auszuführen. Servicebezogene Rollen erscheinen in Ihrem Dienst AWS-Konto und gehören dem Dienst. Ein IAM-Administrator kann die Berechtigungen für Service-verknüpfte Rollen anzeigen, aber nicht bearbeiten.
- Auf Amazon EC2 ausgeführte Anwendungen — Sie können eine IAM-Rolle verwenden, um temporäre Anmeldeinformationen für Anwendungen zu verwalten, die auf einer EC2-Instance ausgeführt werden und API-Anfragen stellen AWS CLI . AWS Das ist eher zu empfehlen, als Zugriffsschlüssel innerhalb der EC2-Instance zu speichern. Um einer EC2-Instance eine AWS Rolle zuzuweisen und sie allen ihren Anwendungen zur Verfügung zu stellen, erstellen Sie ein Instance-Profil, das an die Instance angehängt ist. Ein Instance-Profil enthält die Rolle und ermöglicht, dass Programme, die in der EC2-Instance ausgeführt werden, temporäre Anmeldeinformationen erhalten. Weitere Informationen finden Sie unter Verwenden einer IAM-Rolle zum Erteilen von Berechtigungen für Anwendungen, die auf Amazon-EC2-Instances ausgeführt werden im IAM-Benutzerhandbuch.

Informationen dazu, wann Sie IAM-Rollen oder IAM-Benutzer verwenden sollten, finden Sie unter Erstellen einer IAM-Rolle (anstatt eines Benutzers) im IAM-Benutzerhandbuch.

Verwalten des Zugriffs mit Richtlinien

Sie kontrollieren den Zugriff, AWS indem Sie Richtlinien erstellen und diese an AWS Identitäten oder Ressourcen anhängen. Eine Richtlinie ist ein Objekt, AWS das, wenn es einer Identität oder Ressource zugeordnet ist, deren Berechtigungen definiert. AWS wertet diese Richtlinien aus, wenn ein Prinzipal (Benutzer, Root-Benutzer oder Rollensitzung) eine Anfrage stellt. Berechtigungen in den Richtlinien bestimmen, ob die Anforderung zugelassen oder abgelehnt wird. Die meisten Richtlinien werden AWS als JSON-Dokumente gespeichert. Weitere Informationen zu Struktur und Inhalten von JSON-Richtliniendokumenten finden Sie unter Übersicht über JSON-Richtlinien im IAM-Benutzerhandbuch.

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer auf was Zugriff hat. Das bedeutet, welcher Prinzipal kann Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen.

Standardmäßig haben Benutzer, Gruppen und Rollen keine Berechtigungen. Ein IAM-Administrator muss IAM-Richtlinien erstellen, die Benutzern die Berechtigung erteilen, Aktionen für die Ressourcen auszuführen, die sie benötigen. Der Administrator kann dann die IAM-Richtlinien zu Rollen hinzufügen, und Benutzer können die Rollen annehmen.

IAM-Richtlinien definieren Berechtigungen für eine Aktion unabhängig von der Methode, die Sie zur Ausführung der Aktion verwenden. Angenommen, es gibt eine Richtlinie, die Berechtigungen für die `iam:GetRole`-Aktion erteilt. Ein Benutzer mit dieser Richtlinie kann Rolleninformationen von der AWS Management Console AWS CLI, der oder der AWS API abrufen.

Identitätsbasierte Richtlinien

Identitätsbasierte Richtlinien sind JSON-Berechtigungsrichtliniendokumente, die Sie einer Identität anfügen können, wie z. B. IAM-Benutzern, -Benutzergruppen oder -Rollen. Diese Richtlinien steuern, welche Aktionen die Benutzer und Rollen für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen identitätsbasierter Richtlinien finden Sie unter Erstellen von IAM-Richtlinien im IAM-Benutzerhandbuch.

Identitätsbasierte Richtlinien können weiter als Inline-Richtlinien oder verwaltete Richtlinien kategorisiert werden. Inline-Richtlinien sind direkt in einen einzelnen Benutzer, eine einzelne Gruppe oder eine einzelne Rolle eingebettet. Verwaltete Richtlinien sind eigenständige Richtlinien, die Sie

mehreren Benutzern, Gruppen und Rollen in Ihrem System zuordnen können AWS-Konto. Zu den verwalteten Richtlinien gehören AWS verwaltete Richtlinien und vom Kunden verwaltete Richtlinien. Informationen dazu, wie Sie zwischen einer verwalteten Richtlinie und einer eingebundenen Richtlinie wählen, finden Sie unter Auswahl zwischen verwalteten und eingebundenen Richtlinien im IAM-Benutzerhandbuch.

Ressourcenbasierte Richtlinien

Ressourcenbasierte Richtlinien sind JSON-Richtliniendokumente, die Sie an eine Ressource anfügen. Beispiele für ressourcenbasierte Richtlinien sind IAM-Rollen-Vertrauensrichtlinien und Amazon-S3-Bucket-Richtlinien. In Services, die ressourcenbasierte Richtlinien unterstützen, können Service-Administratoren sie verwenden, um den Zugriff auf eine bestimmte Ressource zu steuern. Für die Ressource, an welche die Richtlinie angehängt ist, legt die Richtlinie fest, welche Aktionen ein bestimmter Prinzipal unter welchen Bedingungen für diese Ressource ausführen kann. Sie müssen in einer ressourcenbasierten Richtlinie einen Prinzipal angeben. Zu den Prinzipalen können Konten, Benutzer, Rollen, Verbundbenutzer oder gehören. AWS-Services

Ressourcenbasierte Richtlinien sind Richtlinien innerhalb dieses Diensts. Sie können AWS verwaltete Richtlinien von IAM nicht in einer ressourcenbasierten Richtlinie verwenden.

Zugriffssteuerungslisten (ACLs)

Zugriffssteuerungslisten (ACLs) steuern, welche Prinzipale (Kontomitglieder, Benutzer oder Rollen) auf eine Ressource zugreifen können. ACLs sind ähnlich wie ressourcenbasierte Richtlinien, verwenden jedoch nicht das JSON-Richtliniendokumentformat.

Amazon S3 und Amazon VPC sind Beispiele für Services, die ACLs unterstützen. AWS WAF Weitere Informationen zu ACLs finden Sie unter Zugriffssteuerungsliste (ACL) – Übersicht (Access Control List) im Amazon-Simple-Storage-Service-Entwicklerhandbuch.

Weitere Richtlinientypen

AWS unterstützt zusätzliche, weniger verbreitete Richtlinientypen. Diese Richtlinientypen können die maximalen Berechtigungen festlegen, die Ihnen von den häufiger verwendeten Richtlinientypen erteilt werden können.

- Berechtigungsgrenzen:Eine Berechtigungsgrenze ist ein erweitertes Feature, mit der Sie die maximalen Berechtigungen festlegen können, die eine identitätsbasierte Richtlinie einer IAM-Entität (IAM-Benutzer oder -Rolle) erteilen kann. Sie können eine Berechtigungsgrenze für eine Entität festlegen. Die daraus resultierenden Berechtigungen sind der Schnittpunkt der

identitätsbasierten Richtlinien einer Entität und ihrer Berechtigungsgrenzen. Ressourcenbasierte Richtlinien, die den Benutzer oder die Rolle im Feld `Principal` angeben, werden nicht durch Berechtigungsgrenzen eingeschränkt. Eine explizite Zugriffsverweigerung in einer dieser Richtlinien setzt eine Zugriffserlaubnis außer Kraft. Weitere Informationen über Berechtigungsgrenzen finden Sie unter Berechtigungsgrenzen für IAM-Entitäten im IAM-Benutzerhandbuch.

- Service Control Policies (SCPs) — SCPs sind JSON-Richtlinien, die die maximalen Berechtigungen für eine Organisation oder Organisationseinheit (OU) in festlegen. AWS Organizations AWS Organizations ist ein Dienst zur Gruppierung und zentralen Verwaltung mehrerer Objekte AWS-Konten , die Ihrem Unternehmen gehören. Wenn Sie innerhalb einer Organisation alle Features aktivieren, können Sie Service-Kontrollrichtlinien (SCPs) auf alle oder einzelne Ihrer Konten anwenden. Das SCP schränkt die Berechtigungen für Entitäten in Mitgliedskonten ein, einschließlich der einzelnen Entitäten. Root-Benutzer des AWS-Kontos Weitere Informationen zu Organizations und SCPs finden Sie unter Funktionsweise von SCPs im AWS Organizations -Benutzerhandbuch.
- Sitzungsrichtlinien:Sitzungsrichtlinien sind erweiterte Richtlinien, die Sie als Parameter übergeben, wenn Sie eine temporäre Sitzung für eine Rolle oder einen verbundenen Benutzer programmgesteuert erstellen. Die resultierenden Sitzungsberechtigungen sind eine Schnittmenge der auf der Identität des Benutzers oder der Rolle basierenden Richtlinien und der Sitzungsrichtlinien. Berechtigungen können auch aus einer ressourcenbasierten Richtlinie stammen. Eine explizite Zugriffsverweigerung in einer dieser Richtlinien setzt eine Zugriffserlaubnis außer Kraft. Weitere Informationen finden Sie unter Sitzungsrichtlinien im IAM-Benutzerhandbuch.

Mehrere Richtlinientypen

Wenn mehrere auf eine Anforderung mehrere Richtlinientypen angewendet werden können, sind die entsprechenden Berechtigungen komplizierter. Informationen darüber, wie AWS bestimmt wird, ob eine Anfrage zulässig ist, wenn mehrere Richtlinientypen betroffen sind, finden Sie im IAM-Benutzerhandbuch unter Bewertungslogik für Richtlinien.

IAM mit Amazon EMR

Sie können Amazon EMR verwenden AWS Identity and Access Management , um Benutzer, AWS Ressourcen, Gruppen, Rollen und Richtlinien zu definieren. Sie können auch steuern, auf welche AWS Dienste diese Benutzer und Rollen zugreifen können.

Weitere Informationen zur Verwendung von IAM mit Amazon EMR finden Sie unter AWS Identity and Access Management für Amazon EMR.

Anmeldung und Überwachung AWS Deep Learning AMI

Ihre AWS Deep Learning AMI Instance verfügt über mehrere GPU-Überwachungstools, darunter ein Hilfsprogramm, das Statistiken zur GPU-Nutzung an Amazon meldet CloudWatch. Weitere Informationen finden Sie unter GPU-Überwachung and -Optimierung und Überwachen von Amazon EC2.

Nachverfolgung der Nutzung

Die folgenden AWS Deep Learning AMI Betriebssystemverteilungen enthalten Code, mit dem Instanztyp, Instanz-ID, DLAMI-Typ und Betriebssysteminformationen erfasst werden können AWS . Es werden keine Informationen zu den in der DLAMI verwendeten Befehlen gesammelt oder gespeichert. Es werden keine weiteren Informationen über das DLAMI gesammelt oder gespeichert.

- Ubuntu 16.04
- Ubuntu 18.04
- Ubuntu 20.04
- Amazon Linux 2

Um die Nutzungsverfolgung für Ihr DLAMI zu deaktivieren, fügen Sie Ihrer Amazon EC2 EC2-Instance beim Start ein Tag hinzu. Das Tag sollte den Schlüssel verwenden, dessen `OPT_OUT_TRACKING` zugeordneter Wert auf gesetzt ist. `true` Weitere Informationen finden Sie unter Taggen Ihrer Amazon EC2 EC2-Ressourcen.

Konformitätsvalidierung für AWS Deep Learning AMI

Externe Prüfer bewerten die Sicherheit und Einhaltung von Vorschriften im AWS Deep Learning AMI Rahmen mehrerer AWS Compliance-Programme. Weitere Informationen zu den unterstützten Compliance-Programmen finden Sie unter Compliance-Validierung für Amazon EC2.

Eine Liste der AWS Services im Rahmen bestimmter Compliance-Programme finden Sie unter AWS Services im Umfang nach Compliance-Programmen AWS . Allgemeine Informationen finden Sie unter AWS Compliance-Programme AWS .

Sie können Prüfberichte von Drittanbietern unter herunterladen AWS Artifact. Weitere Informationen finden Sie unter Berichte in AWS Artifact herunterladen Berichte in AWS Artifact .

Ihre Compliance-Verantwortung bei der Verwendung von DLAMI richtet sich nach der Sensibilität Ihrer Daten, den Compliance-Zielen Ihres Unternehmens und den geltenden Gesetzen und Vorschriften. AWS bietet die folgenden Ressourcen, um Sie bei der Einhaltung von Vorschriften zu unterstützen:

- Schnellstartanleitungen für Sicherheit und Compliance – In diesen Bereitstellungsleitfäden werden architektonische Überlegungen erörtert und Schritte für die Bereitstellung von sicherheits- und konformitätsorientierten Basisumgebungen auf AWS angegeben.
- AWS Ressourcen zur AWS von Vorschriften — Diese Sammlung von Arbeitsmappen und Leitfäden kann auf Ihre Branche und Ihren Standort zutreffen.
- Bewertung von Ressourcen anhand von Regeln im AWS Config Entwicklerhandbuch — Der AWS Config Service bewertet, wie gut Ihre Ressourcenkonfigurationen den internen Praktiken, Branchenrichtlinien und Vorschriften entsprechen.
- AWS Security Hub— Dieser AWS Service bietet einen umfassenden Überblick über Ihren Sicherheitsstatus und hilft Ihnen AWS , die Einhaltung der Sicherheitsstandards und bewährten Verfahren der Branche zu überprüfen.

Resilienz in AWS Deep Learning AMI

Die AWS globale Infrastruktur basiert auf AWS Regionen und Availability Zones. AWS Regionen bieten mehrere physisch getrennte und isolierte Availability Zones, die über Netzwerke mit niedriger Latenz, hohem Durchsatz und hoher Redundanz miteinander verbunden sind. Mithilfe von Availability Zones können Sie Anwendungen und Datenbanken erstellen und ausführen, die automatisch Failover zwischen Zonen ausführen, ohne dass es zu Unterbrechungen kommt. Availability Zones sind besser verfügbar, fehlertoleranter und skalierbarer als herkömmliche Infrastrukturen mit einem oder mehreren Rechenzentren.

Weitere Informationen zu AWS Regionen und Availability Zones finden Sie unter AWS Globale Infrastruktur.

Informationen zu Funktionen zur Unterstützung Ihrer Datenresilienz- und Sicherungsanforderungen finden Sie unter Resilience in Amazon EC2.

Sicherheit der Infrastruktur in AWS Deep Learning AMI

Die Infrastruktursicherheit von AWS Deep Learning AMI wird von Amazon EC2 unterstützt. Weitere Informationen finden Sie unter Infrastruktursicherheit in Amazon EC2.

Wichtige Änderungen an DLAMI

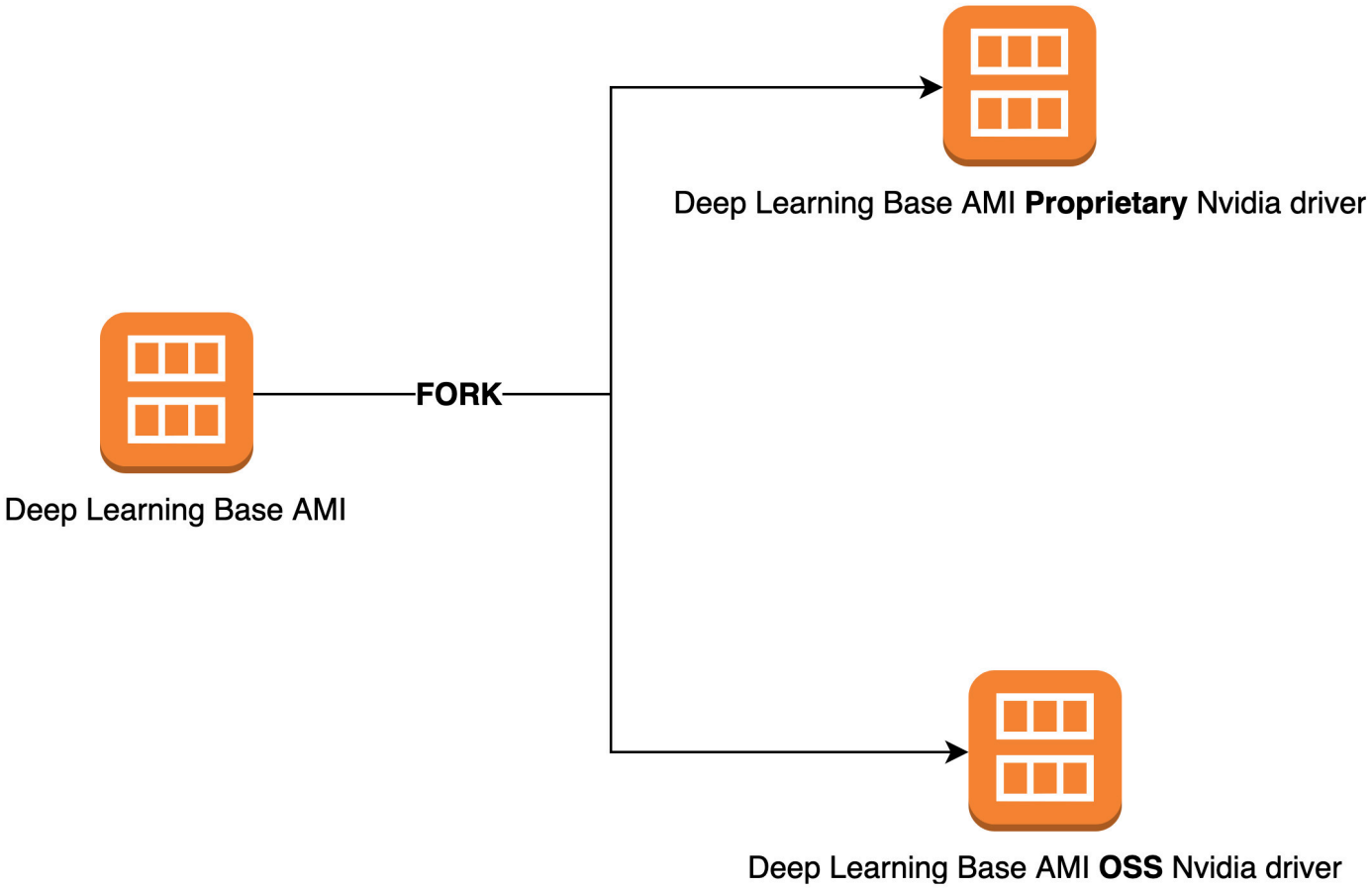
Häufig gestellte Fragen

Was ändert sich?

Am 15.11.2023 wird AWS Deep Learning AMI (DLAMIs) in zwei separate Gruppen aufgeteilt:

- DLAMis, die einen proprietären Treiber von Nvidia verwenden (zur Unterstützung von P3, P3dn, G3).
- DLAMs, die den Nvidia OSS-Treiber verwenden (zur Unterstützung von G4dn, G5, P4, P5).

Infolgedessen werden für jede der beiden Kategorien neue DLAMIs mit neuen Namen und neuen AMI-IDs erstellt. Diese DLAMIs werden nicht austauschbar sein — d. h. DLAMIs aus einer Gruppe unterstützen keine Instances, die von der anderen Gruppe unterstützt werden, z. B. unterstützt das DLAMI, das p5 unterstützt, G3 nicht und umgekehrt.



Warum ist diese Änderung erforderlich?

Derzeit enthalten DLAMIs für NVIDIA-GPUs einen proprietären Kernel-Treiber von NVIDIA. Vor Kurzem hat die Upstream-Linux-Kernel-Community jedoch eine Änderung akzeptiert, die proprietäre Kerneltreiber wie den NVIDIA-GPU-Treiber von der Kommunikation mit anderen Kerneltreibern isoliert. Diese Änderung deaktiviert GPUDirect RDMA auf Instances der P4/P5-Serie. Dies ist der Mechanismus, der es GPUs ermöglicht, EFA effizient für verteilte Schulungen zu nutzen. Aus diesem Grund wird DLAMIS den OpenRM-Treiber (NVIDIA-Open-Source-Treiber) verwenden, der mit den Open-Source-EFA-Treibern verknüpft ist, um G4dn, G5, P4 und P5 zu unterstützen. Dieser OpenRM-Treiber unterstützt jedoch keine älteren Instanzen (P3, G3 usw.) Um sicherzustellen, dass wir weiterhin aktuelle, leistungsstarke und sichere DLAMIs bereitstellen, die beide Instanztypen unterstützen, werden wir DLAMIs in zwei Gruppen aufteilen — eine mit dem OpenRM-Treiber (unterstützt G4dn, G5, P4 und P5) und eine mit dem älteren proprietären Treiber (unterstützt ältere Instanzen P3, P3dn, G3).

Welche DLAMIs sind von dieser Änderung betroffen?

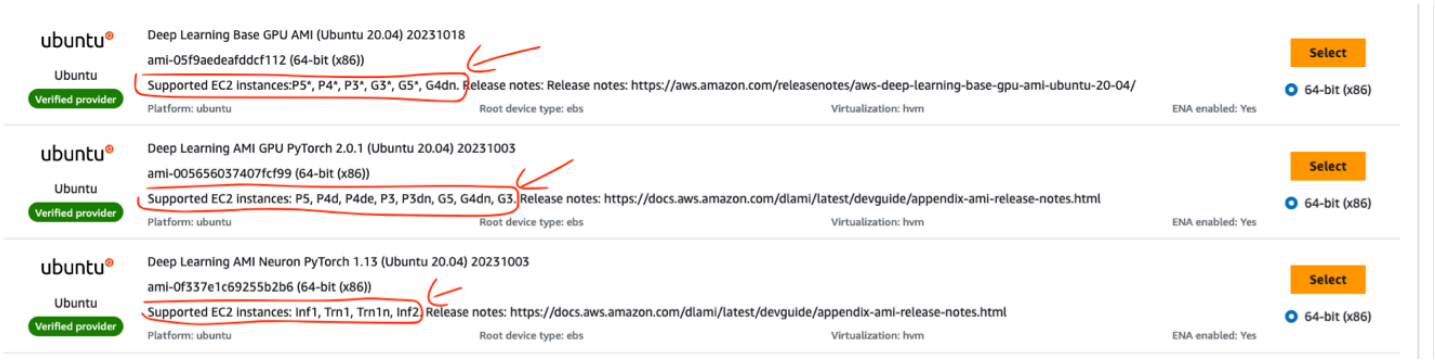
Alle DLAMIs sind von dieser Änderung betroffen.

Was bedeutet das für dich?

Die neuen DLAMIs bieten weiterhin die Funktionalität, Leistung und Sicherheit der aktuellen DLAMIs, solange sie auf einem kompatiblen Instance-Typ ausgeführt werden. Wenn Sie DLAMIs verwenden, müssen Sie sicherstellen, dass ein DLAMI auf einer der kompatiblen Instanzen gestartet wird, die in den Versionshinweisen der einzelnen DLAMIs aufgeführt sind (siehe hier). Beispiel: Sie müssen diese Änderung berücksichtigen, um:

- Rufen Sie DLAMIs mit den richtigen CLI-Abfragen auf (siehe unten)
- Starten Sie DLAMIs über die Konsole und die CLI auf einem kompatiblen Instanztyp

Wenn Sie DLAMIs von der EC2-Konsole aus starten Schnellstart: In jeder DLAMI-Beschreibung sind die Typen von Instances aufgeführt, die in der EC2-Konsole unterstützt werden. Sie sollten die DLAMIs auf kompatiblen Instances starten.



Wenn Sie DLAmis mit CLI starten, müssen Sie Ihre Abfragen ändern. Beispielsweise:

Derzeit wird die folgende CLI-Abfrage für Basis-DLAMIs verwendet, die alle Instanzen unterstützen [P3, P3dn, G3, G4dn, G5, P4, P5]:

```
aws ec2 describe-images --region us-east-1 --owners amazon \
--filters 'Name=name,Values=Deep Learning Base AMI (Amazon Linux 2) ????????'
 'Name=state,Values=available' \
--query 'reverse(sort_by(Images, &CreationDate))[:1].ImageId' --output text
```

Die neuen CLI-Abfragen lauten:

Für Basis-DLAMI, das P3, P3dn und G3 unterstützt:

```
aws ec2 describe-images --region us-east-1 --owners amazon \
--filters 'Name=name,Values=Deep Learning Base Proprietary Nvidia Driver AMI (Amazon
 Linux 2) Version ??.?' 'Name=state,Values=available' \
--query 'reverse(sort_by(Images, &CreationDate))[:1].ImageId' --output text
```

Für Basis-DLAMI, das G4dn, G5, P4 und P5 unterstützt:

```
aws ec2 describe-images --region us-east-1 --owners amazon \
--filters 'Name=name,Values=Deep Learning Base OSS Nvidia Driver AMI (Amazon Linux 2)
 Version ??.?' 'Name=state,Values=available' \
--query 'reverse(sort_by(Images, &CreationDate))[:1].ImageId' --output text
```

Die aktualisierten Versionshinweise für neue AMIs finden Sie hier. Informationen zum Starten von AMIs auf EC2-Instances finden Sie hier.

Wann sollten Sie mit der Verwendung der neuen DLAMIs beginnen?

Sie sollten so schnell wie möglich damit beginnen, die neuen DLAMIs für die neuesten Frameworks, Abhängigkeiten, Patches und Funktionen zu verwenden. Wenn Sie Amazon Linux 2-DLAMIs verwenden, die vor dem 08.11.2023 veröffentlicht wurden, können Sie optional wählen, ihre DLAMIs bis zum 30.11.2023 weiterhin live zu patchen (siehe Anweisungen hier).

Wird es bei den neuen DLAMIs zu Funktionseinbußen kommen?

Nein, bei den neuen DLAmis gibt es keinen Funktionsverlust. Die neuen DLAMIs bieten nach der Aufteilung weiterhin die gesamte Funktionalität, Leistung und Sicherheit der alten DLAMIs vor der Aufteilung, sofern sie auf einer kompatiblen Instanz ausgeführt werden. Wir teilen die DLAMIs in zwei Gruppen auf, sodass wir Ihnen weiterhin aktuelle, performante und sichere DLAMIs für den Einsatz auf einer Vielzahl von Instanzen anbieten können.

Was ist mit DLCs?

DLCs enthalten den NVIDIA-Treiber nicht, sodass sie von dieser Änderung nicht betroffen sind. Sie sollten jedoch sicherstellen, dass die DLCs auf AMIs ausgeführt werden, die mit den zugrunde liegenden Instances kompatibel sind.

Verwandte Informationen

Themen

- Foren
- Verwandte -Blog-Beiträge
- Häufig gestellte Fragen

Foren

- Forum:AWS Deep-Learning-AMIs

Verwandte -Blog-Beiträge

- Aktualisierte Liste von Artikeln zu Deep Learning-AMIs
- Starte einenAWS Deep Learning AMI (in 10 Minuten)
- Schnelleres Training mit Optimized TensorFlow 1.6 auf Amazon EC2 C5- und P3-Instances
- NeueAWS Deep-Learning-AMIs für Praktiker des Machine Learning
- Neue Schulungen verfügbar: Einführung in Machine Learning und Deep Learning amAWS
- Reise ins Deep Learning mitAWS

Häufig gestellte Fragen

- F: Wie behalte ich den Überblick über Produktankündigungen im Zusammenhang mit DLAMI?

  Hier sind zwei Vorschläge dazu:

  - Setzen Sie ein Lesezeichen für diese Blog-Kategorie „AWSDeep Learning AMIs“, die Sie hier finden: Aktualisierte Liste von Artikeln zu Deep Learning-AMIs.
  - Schauen Sie sich das Forum an:AWS Deep Learning-AMIs
- F: Sind die NVIDIA-Treiber und CUDA installiert?

  Ja. Einige DLAMIs haben unterschiedliche Versionen. Der-Deep-Learning-AMI hat die neuesten Versionen aller DLAMI. Dies wird detaillierter in CUDA-Installationen und Framework-Bindungen

beschrieben. Sie können auch in den Versionshinweisen des jeweiligen AMI nachlesen, um zu überprüfen, was installiert ist.

- F: Ist cuDNN installiert?

  Ja.

- F: Wie stelle ich fest, dass die GPUs erkannt werden und sehe ihren aktuellen Status?

  Führen Sie `nvidia-smi`. Dies zeigt, abhängig vom Instance-Typ, eine oder mehrere GPUs und ihren aktuellen Speicherverbrauch an.

- F: Sind virtuelle Umgebungen für mich eingerichtet?

  Ja, aber nur im -Deep-Learning-AMI.

- F: Welche Version von Python ist installiert?

  Jedes DLAMI hat sowohl Python 2 als auch 3. Jedes -Deep-Learning-AMI umfasst Umgebungen für beide Versionen für alle Frameworks.

- F: Ist Keras installiert?

  Das hängt vom AMI ab. Das -Deep-Learning-AMI stellt Keras als Frontend für jedes Framework zur Verfügung. Die Version von Keras hängt von der Unterstützung des Frameworks ab.

- F: Ist es kostenlos?

  Alle DLAMIs sind kostenlos. Je nachdem, welchen Instance-Typ Sie wählen, kann die Instance jedoch kostenpflichtig sein. Weitere Informationen finden Sie unter Preise für den DLAMI.

- F: Ich erhalte CUDA-Fehler oder GPU-bezogene Meldungen von meinem Framework. Was ist los?

  Überprüfen Sie, welchen Instance-Typ Sie verwendet haben. Für viele Beispiele und Tutorials ist eine GPU erforderlich. Wenn beim Ausführen keine GPU`nvidia-smi` angezeigt wird, müssen Sie einen weiteren DLAMI starten, indem Sie eine Instanz mit einer oder mehreren GPUs verwenden. Weitere Informationen finden Sie unter Auswahl des Instance-Typs für DLAMI.

- F: Kann ich Docker verwenden?

  Docker ist seit Version 14 des Deep Learning AMI mit Conda vorinstalliert. Beachten Sie, dass Sie Nvidia-Docker auf GPU-Instanzen verwenden sollten, um die GPU nutzen zu können.

- F: In welchen Regionen sind Linux-DLAMIs verfügbar?

Region	Code
USA Ost (Ohio)	us-east-2
US East (N. Virginia)	us-east-1
GovCloud	us-gov-west-1
USA West (Nordkali fornien)	us-west-1
US West (Oregon)	us-west-2
Peking (China)	cn-north-1
Ningxia (China)	cn-northwest-1
Asien-Pazifik (Mumbai)	ap-south-1
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1
Canada (Central)	ca-central-1
EU (Frankfurt)	eu-central-1
EU (Irland)	eu-west-1
EU (London)	eu-west-2
EU (Paris)	eu-west-3
SA (São Paulo)	sa-east-1

- F: In welchen Regionen sind Windows-DLAMIs verfügbar?

Region	Code
USA Ost (Ohio)	us-east-2
US East (N. Virginia)	us-east-1
GovCloud	us-gov-west-1
USA West (Nordkali fornien)	us-west-1
US West (Oregon)	us-west-2
Peking (China)	cn-north-1
Asien-Pazifik (Mumbai)	ap-south-1
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1
Canada (Central)	ca-central-1
EU (Frankfurt)	eu-central-1
EU (Irland)	eu-west-1
EU (London)	eu-west-2
EU (Paris)	eu-west-3
SA (São Paulo)	sa-east-1

Versionshinweise für DLAMI

> **Note**
>
> AWS Deep Learning AMI Wir haben einen nächtlichen Veröffentlichungsrhythmus für Sicherheitspatches. Diese inkrementellen Sicherheitspatches sind nicht in den offiziellen Versionshinweisen enthalten.

Versionshinweise zu nicht unterstützten Frameworks finden Sie auf der Seite mit den DLAMI-Supportrichtlinien.

Basis DLAMI

GPU

- AWS Deep-Learning-Basis-AMI (Amazon Linux 2)
- AWS Deep-Learning-Basis-AMI (Ubuntu 20.04)

AWS Neuron

- AWS Deep-Learning-Basis AMI Neuron (Amazon Linux 2)
- AWS Deep-Learning-Basis AMI Neuron (Ubuntu 20.04)

Qualcomm

- AWS Deep-Learning-Basis Qualcomm AMI (Amazon Linux 2)

DLAMI mit einem einzigen Framework

PyTorch-Spezifisches AMI

- GPU
  - AWS Deep-Learning-AMI-GPU PyTorch 2.1 (Ubuntu 20.04)
  - AWS Deep-Learning-AMI-GPU PyTorch 1.13 (Amazon Linux 2)

  - AWS Deep-Learning-AMI-GPU PyTorch 1.13 (Ubuntu 20.04)
- AWS Neuron
  - AWS Deep-Learning-AMI Neuron PyTorch 1.13 (Amazon Linux 2)
  - AWS Deep-Learning-AMI Neuron PyTorch 1.13 (Ubuntu 20.04)

TensorFlow-Spezifisches AMI

- GPU
  - AWS Deep-Learning-AMI-GPU TensorFlow 2.15 (Amazon Linux 2)
  - AWS Deep-Learning-AMI-GPU TensorFlow 2.15 (Ubuntu 20.04)
  - AWS Deep-Learning-AMI-GPU TensorFlow 2.13 (Amazon Linux 2)
  - AWS Deep-Learning-AMI-GPU TensorFlow 2.13 (Ubuntu 20.04)
- AWS Neuron
  - AWS Deep-Learning-AMI Neuron TensorFlow 2.10 (Amazon Linux 2)
  - AWS Deep-Learning-AMI Neuron TensorFlow 2.10 (Ubuntu 20.04)

DLAMI mit mehreren Frameworks

GPU

> **Note**
>
> Wenn Sie nur ein Framework für maschinelles Lernen verwenden, empfehlen wir ein DLAMI mit einem einzigen Framework

- AWS Deep-Learning-AMI (Amazon Linux 2)

AWS Neuron

- AWS Deep-Learning-AMI-Neuron (Ubuntu 22.04)

DLAMI bis zur Veraltung bis zur Veraltung bis

In der folgenden Tabelle werden Informationen zu veralteten Funktionen im AWS Deep Learning AMI aufgeführt.

Veraltete Funktion	Datum der Veraltung	Hinweis zur Veraltung
Ubuntu 16.04	10.07.2021	Ubuntu Linux 16.04 LTS erreichte am 30. April 2021 das Ende seines fünfjährigen LTS-Fensters und wird von seinem Anbieter nicht mehr unterstützt. Ab Oktober 2021 gibt es in neuen Versionen keine Updates mehr für das Deep Learning Base AMI (Ubuntu 16.04). Frühere Versionen werden weiterhin verfügbar sein.
Amazon Linux	10.07.2021	Amazon Linux ist end-of-lifeab Dezember 2020 verfügbar. Seit Oktober 2021 gibt es in neuen Versionen keine Updates mehr für das Deep Learning AMI (Amazon Linux). Frühere ab Version (Amazon Linux) ab Version 1 (Amazon Linux) bis zur Veraltung bis zur Veraltung bis zur Veraltung bis zur Veraltung bis zur Veraltung
Chainer	01.07.2020	Chainer hat das Ende der Hauptversionen ab

Veraltete Funktion	Datum der Veraltung	Hinweis zur Veraltung
		Dezember 2019 angekündigt. Folglich werden wir ab Juli 2020 keine Chainer Conda-Umgebungen mehr in den DLAMI aufnehmen. Frühere Versionen des DLAMI, die diese Umgebungen enthalten, werden weiterhin verfügbar sein. Wir werden Updates für diese Umgebungen nur bereitstellen, wenn von der Open-Source-Community Sicherheitskorrekturen für diese Frameworks veröffentlicht werden.
Python 3.6	15.06.2020	Aufgrund von Kundenwünschen wechseln wir für neue TF/MX/PT-Versionen zu Python 3.7.

Veraltete Funktion	Datum der Veraltung	Hinweis zur Veraltung
Python 2	01.01.2020	Die Python-Open-Source-Community hat die Unterstützung für Python 2 offiziell beendet. Die TensorFlow PyTorch, und MXNet-Communities haben außerdem angekündigt, dass die Versionen TensorFlow 1.15, TensorFlow 2.1, PyTorch 1.4 und MXNet 1.6.0 die letzten Versionen sein werden, die Python 2 unterstützen.

Dokumentverlauf für das AWS Deep Learning AMI-Entwicklerhandbuch

Änderung	Beschreibung	Datum
Graviton DLAMI	Der unterstütztAWS Deep Learning AMI jetzt Bilder auf Graviton-GPUs mit Arm-Prozessoren.	29. November 2021
Habana DLAMI	Das unterstütztAWS Deep Learning AMI jetzt Habana Gaudi-Hardware und das Habana SynapseAI SDK.	25. Oktober 2021
TensorFlow 2	Das Deep Learning AMI mit Conda enthält jetzt TensorFlow 2 mit CUDA 10.	3. Dezember 2019
AWSInferenz	Das Deep Learning AMI unterstützt jetztAWS Inferentia-Hardware und dasAWS Neuron SDK.	3. Dezember 2019
TensorFlow Serving mit einem Inception-Modell verwenden	Ein Beispiel für die Verwendung von Inferenz mit einem Inception-Modell wurde für TensorFlow Serving hinzugefügt, sowohl für mit als auch ohne Elastic Inference.	28. November 2018
Training mit 256 GPUs mit TensorFlow und Horovod	Das Tutorial TensorFlow mit Horovod wurde aktualisiert, um ein Beispiel für ein Training mit mehreren Knoten hinzuzufügen.	28. November 2018

Elastic Inference ference ference	Die Voraussetzungen für Elastic Inference sowie zugehörige Informationen wurden zur Einrichtungsanleitung hinzugefügt.	28. November 2018
MMS v1.0 auf dem DLAMI veröffentlicht.	Das MMS-Tutorial wurde aktualisiert. Es nutzt nun das neue Modellarchivformat (.mar) und beinhaltet die neuen Start- und Stopp-Funktionen.	15. November 2018
Installation TensorFlow von einem Nightly Build aus	Es wurde ein Tutorial hinzugefügt, das erklärt TensorFlow, wie Sie mit Conda einen nächtlichen Build von TensorFlow auf Ihrem Deep Learning-AMI deinstallieren und dann installieren können.	16. Oktober 2018
CNTK von einem Nightly Build aus installieren	Es wurde ein Tutorial hinzugefügt, das zeigt, wie Sie CNTK deinstallieren und dann mit Conda einen nächtlichen Build von CNTK auf Ihrem Deep Learning-AMI installieren können.	16. Oktober 2018
Apache MXNet (Incubating) von einem Nightly Build aus installieren	Es wurde ein Tutorial hinzugefügt, das zeigt, wie Sie MXNet deinstallieren und dann mit Conda einen nächtlichen Build von MXNet auf Ihrem Deep Learning-AMI installieren können.	16. Oktober 2018

Installation PyTorch von einem Nightly Build aus	Es wurde ein Tutorial hinzugefügt, das erklärt PyTorch, wie Sie mit Conda einen nächtlichen Build von PyTorch auf Ihrem Deep Learning-AMI deinstallieren und dann installieren können.	25. September 2018
Docker ist jetzt auf Ihrem DLAMI vorinstalliert	Seit Version 14 des Deep Learning AMI mit Conda ist Docker und NVIDIAS Version von Docker für GPUs vorinstal liert.	25. September 2018
TensorBoard Anleitung	Beispiel wurde verschoben nach: ~/examples/tensorboard. Tutorial-Pfade aktualisiert.	23. Juli 2018
MxBoard-Anleitung	Ein Tutorial zur Verwendung von MXBoard für die Visualisi erung von MXNet-Modellen wurde hinzugefügt.	23. Juli 2018
Verteilte Schulungs-Tutorials	Ein Tutorial zum Verwenden von Keras-MXNet für Multi-GPU-Schulungen wurde hinzugefügt. Das Chainer-Tutorial wurde auf v4.2.0 aktualisiert.	23. Juli 2018
Conda-Tutorial	Die Beispiel-MOTD wurde auf eine neuere Version aktualisi ert.	23. Juli 2018
Anleitung für Chainer	Das Tutorial wurde aktualisi ert und verwendet jetzt die neuesten Beispiele von der Chainer-Quelle.	23. Juli 2018

Frühere Aktualisierungen:

In der folgenden Tabelle sind wichtige Änderungen in jeder Version des AWS Deep Learning AMI vor Juli 2018 beschrieben.

Änderung	Beschreibung	Datum
TensorFlow mit Horovod	Ein Tutorial für das Training ImageNet mit TensorFlow und Horovod wurde hinzugefügt.	6. Juni 2018
Aktualisierungsanleitung	Aktualisierungsanleitung wurde hinzugefügt.	15. Mai 2018
Neue Regionen und neues 10-Minuten-Tutorial	Neue Regionen hinzugefügt: US West (Nordkalifornien), Südamerika, Kanada (Zentral) , EU (London) und EU (Paris). Außerdem die erste Version eines 10-Minuten-Tutorials mit dem Titel: „Erste Schritte mit dem Deep Learning AMI“.	26. April 2018
Chainer-Tutorial	Ein Tutorial für die Nutzung von Chainer in den Modi mit mehreren GPUs, einer GPU und CPU wurde hinzugefügt. Die CUDA-Integration wurde für mehrere Frameworks von CUDA 8 auf CUDA 9 aktualisi ert.	28. Februar 2018
Linux AMIs v3.0, plus Einführung von MXNet Model Server, TensorFlow Serving und TensorBoard	Es wurden Tutorials für Conda-AMIs mit neuen Modell- und Visualisierungs-Serverfunktionen unter Verwendung von MXNet Model Server v0.1.5,	25. Januar 2018

Änderung	Beschreibung	Datum
	TensorFlow Serving v1.4.0 und TensorBoard v0.4.0 hinzugefügt. AMI und Framework CUDA-Funktionen, wie in den Conda- und CUDA-Übersichten beschrieben. Neueste Versionshinweise nach https://aws.amazon.com/releasenotes/ verschoben	
Linux-AMIs v2.0	Basis-, Quell- und Conda-AMIs wurden mit NCCL 2.1 aktualisiert. Source- und Conda-AMIs wurden mit MXNet v1.0, PyTorch 0.3.0 und Keras 2.0.9 aktualisiert.	11. Dezember 2017
Zwei Windows-AMI-Optionen hinzugefügt	Windows 2012 R2- und Windows 2016-AMIs veröffentlicht: AMI-Auswahlhilfe ergänzt und Release-Informationen hinzugefügt.	30. November 2017
Erste Dokumentationsversion	Detaillierte Beschreibung der Änderung mit Link zum geänderten Thema/Abschnitt.	15. November 2017

AWS-Glossar

Die neueste AWS-Terminologie finden Sie im AWS-Glossar in der AWS-Glossar-Referenz.

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.