aws

Guida per gli sviluppatori

Deep Learning AMI

Deep Learning AMI: Guida per gli sviluppatori

Table of Contents

Cos'è l'AWS Deep Learning AMI?

Benvenuto nella Guida per l'utente della AWS Deep Learning AMI.

AWS Deep Learning AMI(DLAMI) è il tuo punto di riferimento per il deep learning nel cloud. Questa istanza di macchina personalizzata è disponibile nella maggior parte delle regioni Amazon EC2 per una varietà di tipi di istanze, da una piccola istanza solo per CPU alle più recenti istanze multi-GPU ad alta potenza. È preconfigurato con NVIDIA CUDA e NVIDIA CuDNN, nonché con le ultime versioni dei framework di deep learning più diffusi.

Informazioni sulla guida

Questa guida ti aiuterà a lanciare e utilizzare il DLAMI. Copre vari casi d'uso comuni per l'apprendimento profondo, sia per il training che per l'inferenza. Descrive inoltre come scegliere l'AMI più adatta al tuo scopo e al tipo di istanze scelto. Il DLAMI viene fornito con diversi tutorial per ciascuno dei framework. Contiene anche tutorial sulla formazione distribuita, il debugging, l'uso diAWS Inferentia e altri concetti chiave. Troverai inoltre istruzioni su come configurare Jupyter per eseguire i tutorial nel tuo browser.

Prerequisiti

Dovresti avere familiarità con gli strumenti della riga di comando e con Python di base per eseguire correttamente il DLAMI. I tutorial su come utilizzare ogni framework sono forniti dai framework stessi; tuttavia, questa guida descrive come attivarli e come trovare i tutorial appropriati per iniziare.

Esempi di DLAMI

Informazioni sul deep learning: il DLAMI è un'ottima scelta per l'apprendimento o l'insegnamento dei framework di machine learning e deep learning. Semplifica la risoluzione dei problemi relativi alle installazioni di ogni framework e il funzionamento delle stesse su un unico computer. Il DLAMI viene fornito con un notebook Jupyter e semplifica l'esecuzione dei tutorial forniti dai framework per le persone che non conoscono l'apprendimento automatico e il deep learning.

Sviluppo di app: se sei uno sviluppatore di app e sei interessato a utilizzare il deep learning per far sì che le tue app utilizzino gli ultimi progressi dell'intelligenza artificiale, il DLAMI è il banco di prova perfetto per te. Ogni framework dispone di tutorial su come iniziare a utilizzare l'apprendimento

profondo e molti di questi includono serie di modelli che ne semplificano l'utilizzo eliminando la necessità di creare personalmente reti neurali o eseguire il training dei modelli. Alcuni esempi mostrano come creare un'applicazione di rilevamento delle immagini in pochi minuti oppure un'app di riconoscimento vocale per un servizio di chatbot.

Machine learning e analisi dei dati: se sei un data scientist interessato all'elaborazione dei dati con l'apprendimento profondo, scoprirai che molti framework supportano R e Spark. Troverai tutorial su come eseguire semplici regressioni, fino alla creazione di sistemi di elaborazione dati scalabili per sistemi di personalizzazione e di stima.

Ricerca: se sei un ricercatore e desideri provare un nuovo framework, testare un nuovo modello o addestrare nuovi modelli, il DLAMI eAWS le funzionalità di scalabilità possono alleviare il problema delle noiose installazioni e della gestione di più nodi di formazione.

> **Note**
>
> Sebbene la scelta iniziale possa essere quella di aggiornare il tipo di istanza fino a un'istanza più grande con più GPU (fino a 8), puoi anche scalare orizzontalmente creando un cluster di istanze DLAMI. Per ulteriori informazioni sulle build di cluster, consulta Informazioni correlate.

Caratteristiche di DLAMI

Framework preinstallati

Attualmente esistono due versioni principali del DLAMI con altre varianti relative al sistema operativo (OS) e alle versioni del software:

- AMI di deep learning di – Framework installati separatamente utilizzando pacchetti `conda` e ambienti Python distinti
- Deep Learning Learning Learning AMI- nessun framework installato; solo NVIDIA CUDA e altre dipendenze

L'AMI di Deep Learning con Conda utilizza`conda` gli ambienti per isolare ogni framework, in modo da poter passare da uno all'altro a piacimento e non preoccuparti che le loro dipendenze siano in conflitto.

Questo è l'elenco completo dei framework supportati da Deep Learning AMI con Conda:

- Apache MXNet (incubazione)
- PyTorch
- TensorFlow 2

> **Note**
>
> Non includiamo più gli ambienti CNTK, Caffe, Caffe2, Theano, Chainer o Keras Conda all'AWS Deep Learning AMIinizio della versione v28. Le versioni precedenti di AWS Deep Learning AMI che contengono questi ambienti continueranno a essere disponibili. Tuttavia, verranno forniti aggiornamenti a questi ambienti solo se sono disponibili correzioni di protezione pubblicate dalla comunità open source per questi framework.

Software GPU preinstallato

Anche se utilizzi un'istanza che utilizza solo CPU, il DLAMI avrà NVIDIA CUDA e NVIDIA cuDNN. Il software installato è lo stesso indipendentemente dal tipo di istanza. Tieni presente che gli strumenti specifici alle GPU sono compatibili solo con le istanze con almeno una GPU. Per ulteriori informazioni, consulta Selezione del tipo di istanza per DLAMI.

Per ulteriori informazioni sull'installazione di CUDA, vedere ilInstallazioni CUDA e binding di framework.

Model serving e visualizzazione

L'AMI di Deep Learning con Conda è preinstallato con due tipi di server di modelli TensorFlow, uno per MXNet e uno per la visualizzazione dei modelli. TensorBoard

- Model Server for Apache MXNet (MMS)
- TensorFlow Servire
- TensorBoard

Nozioni di base

Come iniziare con il DLAMI

Questa guida include suggerimenti su come scegliere il DLAMI più adatto a te, selezionare un tipo di istanza adatto al tuo caso d'uso e al tuo budget eInformazioni correlate descrive le configurazioni personalizzate che potrebbero interessarti.

Se sei nuovo nell'usoAWS o nell'utilizzo di Amazon EC2, inizia conAMI di deep learning di. Se conosci Amazon EC2 e altriAWS servizi come Amazon EMR, Amazon EFS o Amazon S3 e sei interessato a integrare tali servizi per progetti che richiedono formazione o inferenza distribuite, verifica se uno si adatta al tuo caso d'uso.Informazioni correlate

Ti consigliamo di consultare dapprima Scelta del tuo DLAMI per avere un'idea del tipo di istanza più adatto per la tua applicazione.

Un'altra opzione è questo breve tutorial: Avvia unAWS Deep Learning AMI (in 10 minuti).

Fase successiva

Scelta del tuo DLAMI

Scelta del tuo DLAMI

Offriamo una gamma di opzioni DLAMI. Per aiutarti a selezionare il DLAMI corretto per il tuo caso d'uso, raggruppiamo le immagini in base al tipo di hardware o alla funzionalità per cui sono state sviluppate. I nostri raggruppamenti di primo livello sono:

- Tipo DLAMI: CUDA versus Base versus Single-Framework versus Multi-Framework (Conda DLAMI)
- Architettura di elaborazione: Graviton basato su x86 rispetto a AWSGraviton basato su ARM
- Tipo di processore: GPU contro CPU contro Inferentia contro Habana
- SDK: CUDA contro AWSNeuron contro SynapsesAI
- Sistema operativo: Amazon Linux contro Ubuntu

Gli altri argomenti di questa guida ti aiutano a informarti ulteriormente e ad approfondire i dettagli.

Argomenti

Argomento successivo

AMI di deep learning di

Installazioni CUDA e binding di framework

Sebbene il deep learning sia piuttosto all'avanguardia, ogni framework offre versioni «stabili». Queste versioni stabili potrebbero non funzionare con l'implementazione e le funzionalità più recenti di CUDA o cuDNN. Il tuo caso d'uso e le funzionalità di cui hai bisogno possono aiutarti a scegliere un framework. Se non sei sicuro, usa l'ultima AMI di Deep Learning con Conda. Dispone di`pip` binari ufficiali per tutti i framework con CUDA 10, utilizzando la versione più recente supportata da ciascun framework. Se desideri le versioni più recenti e personalizzare il tuo ambiente di deep learning, usa l'AMI Deep Learning Base.

Per ulteriori informazioni, consulta la nostra guida Stable Versus Release Candidate.

Scegli un DLAMI con CUDA

Deep Learning Learning Learning AMIHa tutte le serie CUDA 11 disponibili, incluse 11.0, 11.1 e 11.2.

AMI di deep learning diHa tutte le serie CUDA 11 disponibili, incluse 11.0, 11.1 e 11.2.

> **Note**
>
> Non includiamo più gli ambienti CNTK, Caffe, Caffe2, Theano, Chainer o Keras Conda all'AWS Deep Learning AMIinizio della versione v28. Le versioni precedenti di quelleAWS Deep Learning AMI che contengono questi ambienti continuano a essere disponibili. Tuttavia, forniamo aggiornamenti a questi ambienti solo se ci sono correzioni di sicurezza pubblicate dalla comunità open source per questi framework.

Per i numeri di versione del framework specifici, consulta ilNote di rilascio per DLAMI

Scegli questo tipo di DLAMI o scopri di più sui diversi DLAMI con l'opzione Next Up.

Scegli una delle versioni CUDA e consulta l'elenco completo dei DLAMI con quella versione nell'Appendice oppure scopri di più sui diversi DLAMI con l'opzione Next Up.

Argomento successivo

Deep Learning Learning Learning AMI

Argomenti correlati

- Per le istruzioni su come passare da una versione CUDA all'altra, consulta il tutorial Utilizzo dell'AMI Deep Learning Base.

Deep Learning Learning Learning AMI

L'AMI Deep Learning Base è come una tela vuota per il deep learning. Viene fornito con tutto il necessario fino al momento dell'installazione di un particolare framework e offre una scelta di versioni CUDA.

Perché scegliere la Base DLAMI

Questo gruppo di AMI è utile per chi collabora ai progetti e intende eseguire il fork di un progetto di apprendimento profondo e compilare la versione più recente. È destinato a chiunque desideri utilizzare il proprio ambiente avendo la certezza che il software NVIDIA più recente sia installato e funzionante, in modo da potersi concentrare sulla scelta dei framework e delle versioni che intende installare.

Scegli questo tipo di DLAMI o scopri di più sui diversi DLAMI con l'opzione Next Up.

Argomento successivo

DLAMI con Conda

Argomenti correlati

- Utilizzo dell'AMI Deep Learning Base

AMI di deep learning di

Il DLAMI Conda utilizza ambienticonda virtuali. Questi ambienti sono configurati per mantenere separate le diverse installazioni del framework e semplificare il passaggio da un framework all'altro. Questo è ottimo per imparare e sperimentare tutti i framework che il DLAMI ha da offrire. La maggior parte degli utenti ritiene che la nuova AMI di Deep Learning con Conda sia perfetta per loro.

Queste AMI sono le DLAMI principali. Vengono aggiornati spesso con le ultime versioni dei framework e dispongono dei driver e del software GPU più recenti. Sono generalmente indicati come i AWS Deep Learning AMInella maggior parte dei documenti.

- Il DLAMI di Ubuntu 18.04 ha i seguenti framework: Apache MXNet (Incubating) e TensorFlow 2. PyTorch
- Il DLAMI di Amazon Linux 2 ha i seguenti framework: Apache MXNet (Incubating) e TensorFlow 2. PyTorch

> **Note**
>
> Non includiamo più gli ambienti CNTK, Caffe, Caffe2, Theano, Chainer e Keras Conda all'AWS Deep Learning AMInizio della versione v28. Le versioni precedenti di quelleAWS Deep Learning AMI che contengono questi ambienti continuano a essere disponibili. Tuttavia, forniamo aggiornamenti a questi ambienti solo se ci sono correzioni di sicurezza pubblicate dalla comunità open source per questi framework.

Stable Versus Release Candidate

Le AMI Conda utilizzano file binari ottimizzati delle versioni ufficiali più recenti di ogni framework. Versioni candidate e funzionalità sperimentali non sono previste. Le ottimizzazioni dipendono dal supporto del framework per tecnologie di accelerazione come MKL DNN di Intel, che velocizza l'addestramento e l'inferenza sui tipi di istanze di CPU C5 e C4. I binari sono inoltre compilati per supportare set di istruzioni Intel avanzati, inclusi, a titolo esemplificativo, AVX, AVX-2, SSE4.1 e SSE4.2. Questi accelerano le operazioni vettoriali e a virgola mobile su architetture CPU di Intel. Inoltre, per i tipi di istanze GPU, CUDA e cuDNN vengono aggiornati con qualsiasi versione supportata dall'ultima versione ufficiale.

L'AMI Deep Learning con Conda installa automaticamente la versione più ottimizzata del framework per la tua istanza Amazon EC2 alla prima attivazione del framework. Per ulteriori informazioni, consulta Utilizzo dell'AMI Deep Learning con Conda.

Se desideri installare dal sorgente, utilizzando opzioni di build personalizzate o ottimizzate,Deep Learning Learning Learning AMI s potrebbe essere l'opzione migliore per te.

Impostare Python 2 come obsoleto

La comunità open source di Python ha ufficialmente interrotto il supporto per Python 2 il 1 gennaio 2020. La PyTorch community TensorFlow e la community hanno annunciato che le versioni TensorFlow 2.1 e PyTorch 1.4 sono le ultime a supportare Python 2. Le versioni precedenti del DLAMI (v26, v25, ecc.) che contengono ambienti Python 2 Conda continuano a essere disponibili. Tuttavia, forniamo aggiornamenti agli ambienti Python 2 Conda su versioni DLAMI pubblicate in precedenza solo se ci sono correzioni di sicurezza pubblicate dalla comunità open source per tali versioni. Le versioni DLAMI con le ultime versioni dei PyTorch framework TensorFlow and non contengono gli ambienti Python 2 Conda.

Supporto per CUDA

I numeri di versione CUDA specifici sono disponibili nelle note di rilascio della GPU DLAMI.

Argomento successivo

Opzioni di architettura CPU DLAMI

Argomenti correlati

- Per un tutorial sull'utilizzo di un'AMI di Deep Learning con Conda, consulta ilUtilizzo dell'AMI Deep Learning con Conda tutorial.

Opzioni di architettura CPU DLAMI

AWS Deep Learning AMIs sono offerti con architetture CPU AWSGraviton2 basate su x86 o basate su ARM.

Scegli una delle DLAMI della GPU Graviton per lavorare con un'architettura CPU basata su ARM. Tutte le altre GPU DLAMI sono attualmente basate su x86.

- AWSGPU CUDA 11.4 per l'apprendimento profondo AMI Graviton (Ubuntu 20.04)

- AWSGPU Graviton AMI Deep Learning TensorFlow 2.6 (Ubuntu 20.04)
- AWSGPU Graviton AMI Deep Learning PyTorch 1.10 (Ubuntu 20.04)

Per informazioni sull'utilizzo della GPS con, consultaIl Graviton DLAMI. Per maggiori dettagli sui tipi di istanze disponibili, consultaSelezione del tipo di istanza per DLAMI.

Argomento successivo

Opzioni del sistema operativo DLAMI

Opzioni del sistema operativo DLAMI

I DLAMI sono disponibili nei seguenti sistemi operativi.

- Amazon Linux 2
- Ubuntu 20.04
- Ubuntu 18.04

Le versioni precedenti dei sistemi operativi sono disponibili su DLAMI obsoleti. Per ulteriori informazioni sulla deprecazione di DLAMI, vedere Obsoleti per DLAMI

Prima di scegliere un DLAMI, valuta il tipo di istanza di cui hai bisogno e identifica la tuaAWS regione.

Argomento successivo

Selezione del tipo di istanza per DLAMI

Selezione del tipo di istanza per DLAMI

Consulta il AWS Deep Learning AMIcatalogo per le famiglie di istanze Amazon EC2 consigliate compatibili con DLAMI specifiche.

Più in generale, considera quanto segue quando selezioni un tipo di istanza per un DLAMI.

- Se non conosci il deep learning, un'istanza con una singola GPU potrebbe soddisfare le tue esigenze.
- Se sei attento al budget, puoi utilizzare istanze solo per CPU.

- Se stai cercando di ottimizzare le alte prestazioni e l'efficienza in termini di costi per l'inferenza di modelli di deep learning, puoi utilizzare istanze con chip AWS Inferentia.
- Se stai cercando di ottimizzare le alte prestazioni e l'efficienza in termini di costi per la formazione sui modelli di deep learning, puoi utilizzare le istanze con acceleratori Habana.
- Se stai cercando un'istanza GPU ad alte prestazioni con un'architettura CPU basata su ARM, puoi utilizzare il tipo di istanza G5G.
- Se sei interessato a eseguire un modello preaddestrato per inferenze e previsioni, puoi collegare Amazon Elastic Inference alla tua istanza Amazon EC2. Amazon Elastic Inference ti dà accesso a un acceleratore con una frazione di GPU.
- Per i servizi di inferenza ad alto volume, una singola istanza di CPU con molta memoria o un cluster di tali istanze potrebbe essere una soluzione migliore.
- Se utilizzi un modello di grandi dimensioni con molti dati o batch di grandi dimensioni, allora hai bisogno di un'istanza più grande con più memoria. È inoltre possibile distribuire il modello a un cluster di GPU. Potresti scoprire che l'utilizzo di un'istanza con meno memoria è una soluzione migliore se riduci la dimensione del batch. Ciò potrebbe influire sulla precisione e sulla velocità di allenamento.
- Se sei interessato a eseguire applicazioni di machine learning utilizzando NVIDIA Collective Communications Library (NCCL) che richiedono livelli elevati di comunicazioni internodale su larga scala in, potresti utilizzare Elastic Fabric Adapter (EFA).

Per maggiori dettagli sulle istanze, vedere Tipi di istanze EC2.

I seguenti argomenti forniscono informazioni sui tipi di istanza.

> **Important**
>
> Le Deep Learning AMI includono driver, software o kit di strumenti sviluppati, di proprietà o forniti da NVIDIA Corporation. A questo proposito, accetti di utilizzare tali driver, software o kit di strumenti NVIDIA solo sulle istanze Amazon EC2 che includono hardware NVIDIA.

Argomenti

- Prezzi per il DLAMI
- Disponibilità di Regioni DLAMI
- Istanze GPU consigliate

- Istanze CPU consigliate
- Istanze Inferent
- Istanze di Trainium consigliate
- Istanze consigliate di Habana

Prezzi per il DLAMI

I framework di deep learning inclusi nel DLAMI sono gratuiti e ognuno ha le proprie licenze open source. Sebbene il software incluso nel DLAMI sia gratuito, devi comunque pagare per l'hardware dell'istanza Amazon EC2 sottostante.

Alcuni tipi di istanza Amazon EC2 sono etichettati come gratuiti. È possibile eseguire il DLAMI su una di queste istanze gratuite. Ciò significa che l'utilizzo del DLAMI è completamente gratuito quando si utilizza solo la capacità di quell'istanza. Se hai bisogno di un'istanza più potente con più core CPU, più spazio su disco, più RAM o una o più GPU, allora hai bisogno di un'istanza che non rientri nella classe delle istanze a livello libero.

Per ulteriori informazioni sulla selezione e sui prezzi delle istanze, consulta Prezzi di Amazon EC2.

Disponibilità di Regioni DLAMI

Ogni regione supporta una gamma diversa di tipi di istanza e spesso un tipo di istanza ha un costo leggermente diverso nelle diverse regioni. I DLAMI non sono disponibili in tutte le regioni, ma è possibile copiare i DLAMI nella regione di tua scelta. Per ulteriori informazioni, consulta Copiare un AMI. Prendi nota dell'elenco di selezione delle regioni e assicurati di scegliere una regione più vicina a te o ai tuoi clienti. Se prevedi di utilizzare più di un DLAMI e potenzialmente creare un cluster, assicurati di utilizzare la stessa regione per tutti i nodi del cluster.

Per maggiori informazioni sulle regioni, visita .

Argomento successivo

Istanze GPU consigliate

Istanze GPU consigliate

Consigliamo un'istanza GPU per la maggior parte degli scopi di deep learning. L'addestramento di nuovi modelli è più rapido su un'istanza GPU che su un'istanza CPU. Puoi scalare in modo sublineare

quando hai istanze con più GPU o se utilizzi il training distribuito in molte istanze con GPU. Per configurare una formazione distribuita, vedereTraining distribuito.

I tipi di istanza seguenti supportano il DLAMI. Per informazioni sulle opzioni relative ai tipi di istanza GPU e sui relativi usi, consulta e seleziona Accelerated Computing.

> **Note**
>
> La dimensione del modello dovrebbe essere un elemento di selezione di un'istanza. Se il modello supera la RAM disponibile di un'istanza, selezionare un tipo di istanza diverso con memoria sufficiente per l'applicazione.

- Le istanze Amazon EC2 P3 dispongono di un massimo di 8 GPU NVIDIA Tesla V100.
- Le istanze Amazon EC2 P4 dispongono di un massimo di 8 GPU NVIDIA Tesla A100.
- Le istanze Amazon EC2 G3 dispongono di un massimo di 4 GPU NVIDIA Tesla M60.
- Le istanze Amazon EC2 G4 dispongono di un massimo di 4 GPU NVIDIA T4.
- Le istanze Amazon EC2 G5 dispongono di un massimo di 8 GPU NVIDIA A10G.
- Le istanze G5G di Amazon EC2 sono dotate di processori Graviton2 basati su AWS ARM.

Le istanze DLAMI forniscono strumenti per monitorare e ottimizzare i processi GPU. Per ulteriori informazioni sul monitoraggio dei processi della GPU, consultaMonitoraggio e ottimizzazione GPU.

Per tutorial specifici sull'utilizzo delle istanze G5G, consulta. Il Graviton DLAMI

Argomento successivo

Istanze CPU consigliate

Istanze CPU consigliate

Indipendentemente dal budget, dal livello di conoscenza dell'apprendimento profondo o dall'esigenza di eseguire un servizio di stima, hai a disposizione molte opzioni abbordabili nella categoria CPU. Alcuni framework sfruttano l'MKL DNN di Intel, che velocizza l'addestramento e l'inferenza sui tipi di istanze CPU C5 (non disponibile in tutte le regioni), C4 e C3. Per informazioni sui tipi di istanze CPU, consulta EC2 e seleziona Compute Optimized.

> **Note**
>
> La dimensione del modello dovrebbe essere un elemento di selezione di un'istanza. Se il modello supera la RAM disponibile di un'istanza, selezionare un tipo di istanza diverso con memoria sufficiente per l'applicazione.

- Le istanze C5 Amazon EC2 dispongono di un massimo di 72 vCPUs Intel. Le istanze C5 eccellono nella modellazione scientifica, nell'elaborazione in batch, nell'analisi distribuita, nell'elaborazione ad alte prestazioni (HPC) e nell'inferenza di machine learning e deep learning.
- Le istanze C4 di Amazon EC2 includono fino a 36 vCPU Intel.

Argomento successivo

Istanze Inferent

Istanze Inferent

AWSLe istanze Inferent In particolare, i tipi di istanze Inf2 utilizzano i chip AWS Inferentia e l'SDK AWS Neuron, che è integrato con i più diffusi framework di apprendimento automatico come e. TensorFlow PyTorch

I clienti possono utilizzare le istanze di Inf2 per eseguire applicazioni di inferenza di apprendimento automatico su larga scala come ricerca, motori di raccomandazione, visione artificiale, riconoscimento vocale, elaborazione del linguaggio naturale, personalizzazione e rilevamento delle frodi, al costo più basso del cloud.

> **Note**
>
> La dimensione del modello dovrebbe essere un elemento di selezione di un'istanza. Se il modello supera la RAM disponibile di un'istanza, selezionare un tipo di istanza diverso con memoria sufficiente per l'applicazione.

- Le istanze Amazon EC2 Inf2 dispongono di un massimo di 16 chip AWS Inferentia e 100 Gbps di throughput di rete.

Per ulteriori informazioni sull'utilizzo di AWS Inferent Il chip AWS Inferentia con DLAMI

Argomento successivo

Istanze di Trainium consigliate

Istanze di Trainium consigliate

AWSLe istanze Trainium sono progettate per fornire prestazioni elevate ed efficienza nei costi per i carichi di lavoro di Inferent In particolare, i tipi di istanze Trn1 utilizzano chip AWS Trainium e AWSNeuron SDK, che è integrato con i più diffusi framework di apprendimento automatico come e. TensorFlow PyTorch

I clienti possono utilizzare le istanze Trn1 per eseguire applicazioni di inferenza di machine learning su larga scala come ricerca, motori di raccomandazione, visione artificiale, riconoscimento vocale, elaborazione del linguaggio naturale, personalizzazione e rilevamento delle frodi, al costo più basso del cloud.

> **Note**
>
> La dimensione del modello dovrebbe essere un elemento di selezione di un'istanza. Se il modello supera la RAM disponibile di un'istanza, selezionare un tipo di istanza diverso con memoria sufficiente per l'applicazione.

- Le istanze Trn1 di Amazon EC2 dispongono di un massimo di 16 chip AWS Trainium e 100 Gbps di throughput di rete.

Argomento successivo

Istanze consigliate di Habana

Istanze consigliate di Habana

Le istanze con gli acceleratori Habana sono progettate per fornire prestazioni elevate ed efficienza nei costi per i carichi di lavoro di deep learning di formazione. In particolare, i tipi di istanza DL1 utilizzano gli acceleratori Habana Gaudi di Habana Labs, una società Intel. Le istanze DL1 sono ideali per i modelli di machine learning di machine learning utilizzati in applicazioni quali l'elaborazione del linguaggio naturale, il rilevamento e la classificazione degli oggetti, i motori di raccomandazione e la percezione dei veicoli autonomi.

Le istanze con acceleratori Habana sono configurate con il software Habana SynapseAI e preintegrate con i più diffusi framework di machine learning come e. TensorFlow PyTorch Se stai cercando una combinazione ottimale di prestazioni e prezzo per la formazione di modelli di deep learning, prendi in considerazione le istanze con acceleratori Habana per il minor costo di formazione.

> **Note**
>
> La dimensione del modello dovrebbe essere un elemento di selezione di un'istanza. Se il modello supera la RAM disponibile di un'istanza, selezionare un tipo di istanza diverso con memoria sufficiente per l'applicazione.

- Le istanze Amazon EC2 DL1 dispongono di un massimo di otto acceleratori Habana Gaudi, 256 GB di memoria acceleratrice, 4 TB di storage NVMe locale e 400 Gbps di throughput di rete.

Per ulteriori informazioni sull'utilizzo di Habana DLAmis, consulta. I DLAMI dell'Habana

Politica di supporto del Framework

AWS Deep Learning AMIs (DLAMIS) semplificano la configurazione delle immagini per i carichi di lavoro di deep learning e sono ottimizzati con i framework, l'hardware, i driver, le librerie e i sistemi operativi più recenti. Questa pagina descrive in dettaglio la politica di supporto del framework per DLamis. Per un elenco dei DLAMI disponibili, vedere le note di rilascio per DLAMI.

Framework supportati

Fai riferimento alla seguente tabella AWS Deep Learning AMI Framework Support Policy per verificare quali framework e versioni sono supportati attivamente.

Fate riferimento a Fine della patch per verificare per quanto tempo AWS supporta le versioni correnti che sono supportate attivamente dal team di manutenzione del framework di origine. I framework e le versioni sono disponibili in DLAMI a framework singolo o DLAMI multi-framework.

> **Note**
>
> Nella versione del framework x.y.z, x si riferisce alla versione principale, y si riferisce alla versione secondaria e z si riferisce alla versione patch. Ad esempio, per TensorFlow 2.6.5, la versione principale è 2, la versione secondaria è 6 e la versione patch è 5.

Consulta le note di rilascio per maggiori dettagli su immagini specifiche:

- Note di rilascio DLAMI a framework singolo
- Note di rilascio DLAMI multi-framework

Domande frequenti

- A quali versioni del framework vengono applicate le patch di sicurezza?
- Quali immagini vengono AWS pubblicate quando vengono rilasciate nuove versioni del framework?
- Quali immagini sono dotate di nuove SageMaker o nuove AWS funzionalità?
- Come viene definita la versione corrente nella tabella Supported Frameworks?
- Cosa succede se utilizzo una versione che non è inclusa nella tabella Supported Frameworks?

- I DLAMI supportano le versioni precedenti di? TensorFlow
- Come posso trovare l'ultima immagine patchata per una versione del framework supportata?
- Con che frequenza vengono rilasciate nuove immagini?
- La mia istanza verrà aggiornata mentre il mio carico di lavoro è in esecuzione?
- Cosa succede quando è disponibile una nuova versione del framework patchata o aggiornata?
- Le dipendenze vengono aggiornate senza modificare la versione del framework?
- Quando termina il supporto attivo per la mia versione del framework?
- Le immagini con versioni del framework che non sono più gestite attivamente verranno corrette?
- Come posso usare una versione precedente del framework?
- Come posso attenermi alle modifiche up-to-date al supporto dei framework e delle relative versioni?
- Ho bisogno di una licenza commerciale per utilizzare l'Anaconda Repository?

A quali versioni del framework vengono applicate le patch di sicurezza?

Se la versione del framework è etichettata Supported nella tabella AWS Deep Learning AMI Framework Support Policy, ottiene le patch di sicurezza.

Quali immagini vengono AWS pubblicate quando vengono rilasciate nuove versioni del framework?

Pubblichiamo nuovi DLAMI subito dopo il rilascio TensorFlow e PyTorch il rilascio delle nuove versioni di. Ciò include le versioni principali, le versioni principali e secondarie e major-minor-patch le versioni dei framework. Aggiorniamo le immagini anche quando diventano disponibili nuove versioni di driver e librerie. Per ulteriori informazioni sulla manutenzione delle immagini, vedere Quando termina il supporto attivo per la mia versione del framework?

Quali immagini sono dotate di nuove SageMaker o nuove AWS funzionalità?

Le nuove funzionalità vengono in genere rilasciate nell'ultima versione di DLamis per PyTorch e. TensorFlow Per informazioni dettagliate sulle novità SageMaker o AWS sulle funzionalità, consultate le note di rilascio per un'immagine specifica. Per un elenco dei DLAMI disponibili, vedere le note di

rilascio per DLAMI. Per ulteriori informazioni sulla manutenzione delle immagini, vedere Quando termina il supporto attivo per la mia versione del framework?

Come viene definita la versione corrente nella tabella Supported Frameworks?

La versione corrente nella tabella AWS Deep Learning AMI Framework Support Policy si riferisce alla versione più recente del framework disponibile su GitHub. AWS Ogni ultima versione include aggiornamenti ai driver, alle librerie e ai pacchetti pertinenti del DLAMI. Per informazioni sulla manutenzione delle immagini, vedere Quando termina il supporto attivo per la mia versione del framework?

Cosa succede se utilizzo una versione che non è inclusa nella tabella Supported Frameworks?

Se si esegue una versione non inclusa nella tabella AWS Deep Learning AMI Framework Support Policy, è possibile che non si disponga dei driver, delle librerie e dei pacchetti pertinenti più aggiornati. Per un'altra up-to-date versione, ti consigliamo di eseguire l'aggiornamento a uno dei framework supportati disponibili utilizzando il DLAMI più recente di tua scelta. Per un elenco dei DLAMI disponibili, vedere le note di rilascio per DLAMI.

I DLAMI supportano le versioni precedenti di? TensorFlow

No. Supportiamo l'ultima versione patch dell'ultima versione principale di ogni framework rilasciata 365 giorni dalla sua GitHub versione iniziale, come indicato nella tabella AWS Deep Learning AMI Framework Support Policy. Per ulteriori informazioni, consultare Cosa succede se utilizzo una versione che non è inclusa nella tabella Supported Frameworks?

Come posso trovare l'ultima immagine patchata per una versione del framework supportata?

Per utilizzare un DLAMI con l'ultima versione del framework, recupera l'ID DLAMI e usalo per avviare il DLAMI utilizzando la console EC2. Per esempi di comandi AWS CLI per recuperare l'AWS Deep Learning AMIID, consulta la sezione Deep Learning Frameworks nel catalogo. AWS Deep Learning AMI AWS Le interrogazioni CLI AMI ID sono incluse anche nelle note di rilascio DLAMI a framework singolo. La versione del framework scelta deve essere etichettata Supported nella tabella AWS Deep Learning AMI Framework Support Policy.

Con che frequenza vengono rilasciate nuove immagini?

Fornire versioni di patch aggiornate è la nostra massima priorità. Creiamo regolarmente immagini con patch non appena possibile. Monitoriamo la presenza di nuove versioni del framework con patch (es. TensorFlow da 2.9 a TensorFlow 2.9.1) e nuove versioni secondarie (es. TensorFlow da 2.9 a TensorFlow 2.10) e renderli disponibili il prima possibile. Quando TensorFlow viene rilasciata una versione esistente di CUDA, rilasciamo un nuovo DLAMI per quella versione con supporto per la nuova versione TensorFlow di CUDA.

La mia istanza verrà aggiornata mentre il mio carico di lavoro è in esecuzione?

No. Gli aggiornamenti delle patch per DLAMI non sono aggiornamenti «sul posto».

È necessario attivare una nuova istanza EC2, migrare i carichi di lavoro e gli script e quindi disattivare l'istanza precedente.

Cosa succede quando è disponibile una nuova versione del framework patchata o aggiornata?

Controlla regolarmente la pagina delle note di rilascio della tua immagine. Ti consigliamo di eseguire l'aggiornamento a nuovi framework patchati o aggiornati non appena disponibili. Per un elenco dei DLAMI disponibili, vedere le note di rilascio per DLAMI.

Le dipendenze vengono aggiornate senza modificare la versione del framework?

Aggiorniamo le dipendenze senza modificare la versione del framework. Tuttavia, se un aggiornamento delle dipendenze causa un'incompatibilità, creiamo un'immagine con una versione diversa. Assicurati di controllare le Note di rilascio per DLAMI per informazioni aggiornate sulle dipendenze.

Quando termina il supporto attivo per la mia versione del framework?

Le immagini DLAMI sono immutabili. Una volta create, non cambiano. Esistono quattro ragioni principali per cui il supporto attivo per una versione del framework termina:

- Aggiornamenti della versione del framework (patch)

- AWSpatch di sicurezza
- Data di fine della patch (data di scadenza)
- Dipendenza end-of-support

> **Note**
>
> A causa della frequenza degli aggiornamenti delle patch di versione e delle patch di sicurezza, consigliamo di controllare spesso la pagina delle note di rilascio del DLAMI e di eseguire l'aggiornamento quando vengono apportate modifiche.

Aggiornamenti della versione del framework (patch)

Se disponi di un carico di lavoro DLAMI basato sulla versione TensorFlow 2.7.0 e TensorFlow versioni successive alla versione 2.7.1, rilascia un nuovo DLAMI con GitHub la versione 2.7.1. AWS TensorFlow Le immagini precedenti con 2.7.0 non vengono più mantenute attivamente una volta rilasciata la nuova immagine con 2.7.1. TensorFlow Il DLAMI con TensorFlow 2.7.0 non riceve ulteriori patch. La pagina delle note di rilascio di DLAMI per la versione TensorFlow 2.7 viene quindi aggiornata con le informazioni più recenti. Non esiste una pagina delle note di rilascio individuale per ogni patch minore.

I nuovi DLAMI creati a seguito degli aggiornamenti delle patch sono designati con un nuovo ID AMI.

AWSpatch di sicurezza

Se hai un carico di lavoro basato su un'immagine con TensorFlow 2.7.0 e crei una patch AWS di sicurezza, viene rilasciata una nuova versione di DLAMI per la 2.7.0. TensorFlow La versione precedente delle immagini con TensorFlow 2.7.0 non viene più gestita attivamente. Per ulteriori informazioni, consulta La mia istanza verrà aggiornata mentre il mio carico di lavoro è in esecuzione? Per la procedura di ricerca del DLAMI più recente, vedere Come posso trovare l'ultima immagine patchata per una versione del framework supportata?

I nuovi DLAMI creati a seguito degli aggiornamenti delle patch sono designati con un nuovo ID AMI.

Data di fine della patch (data di scadenza)

I DLAMI hanno raggiunto la data di fine della patch 365 giorni dopo la data di GitHub rilascio.

Per i DLAMI multi-framework, quando una delle versioni del framework viene aggiornata, è necessario un nuovo DLAMI con la versione aggiornata. Il DLAMI con la vecchia versione del framework non viene più mantenuto attivamente.

> **Important**
>
> Facciamo un'eccezione quando c'è un importante aggiornamento del framework. Ad esempio, se la versione TensorFlow 1.15 viene aggiornata alla versione TensorFlow 2.0, continuiamo a supportare la versione più recente della TensorFlow 1.15 per un periodo di due anni dalla data di GitHub rilascio o sei mesi dopo la cessazione del supporto da parte del team di manutenzione del framework di origine, a seconda di quale data sia precedente.

Dipendenza end-of-support

Se stai eseguendo un carico di lavoro su un'immagine DLAMI TensorFlow 2.7.0 con Python 3.6 e quella versione di Python è contrassegnata per end-of-support, tutte le immagini DLAMI basate su Python 3.6 non verranno più gestite attivamente. Allo stesso modo, se una versione del sistema operativo come Ubuntu 16.04 è contrassegnata per end-of-support, tutte le immagini DLAMI che dipendono da Ubuntu 16.04 non verranno più gestite attivamente.

Le immagini con versioni del framework che non sono più gestite attivamente verranno corrette?

No. Le immagini che non vengono più gestite attivamente non avranno nuove versioni.

Come posso usare una versione precedente del framework?

Per utilizzare un DLAMI con una versione precedente del framework, recupera l'ID DLAMI e usalo per avviare il DLAMI utilizzando la console EC2. Per i comandi AWS CLI per recuperare l'ID AMI, consulta la sezione Deep Learning Frameworks nel catalogo delle AMI di Deep AWS Learning. AWS Le interrogazioni CLI AMI ID sono incluse anche nelle note di rilascio DLAMI a framework singolo.

Come posso attenermi alle modifiche up-to-date al supporto dei framework e delle relative versioni?

Resta up-to-date con i framework e le versioni DLAMI utilizzando la tabella Framework AWS Deep Learning AMISupport Policy, le note di rilascio DLAMI.

Ho bisogno di una licenza commerciale per utilizzare l'Anaconda Repository?

Anaconda è passata a un modello di licenza commerciale per determinati utenti. I DLAMI gestiti attivamente sono stati migrati alla versione open source di Conda (conda-forge) disponibile al pubblico dal canale Anaconda.

Avvio e configurazione di un DLAMI

Se stai leggendo questa sezione, hai già una buona idea di quale AMI intendi avviare. In caso contrario, trova un DLAMI e il relativo hardware, framework e recupero degli ID nel AWS Deep Learning AMIcatalogo o visualizza le note di rilascio attuali e storiche di DLAMI inNote di rilascio per DLAMI.

Devi inoltre sapere quale tipo di istanza e regione vuoi utilizzare. Se non hai ancora deciso, consulta Selezione del tipo di istanza per DLAMI.

> **Note**
>
> Useremo p3.16xlarge come tipo di istanza predefinito negli esempi. Sostituisci questo tipo con quello che desideri utilizzare.

> **Important**
>
> Se prevedi di utilizzare Elastic Inference, hai Elastic Inference Setup che deve essere completato prima di avviare il tuo DLAMI.

Argomenti

Fase 1: avvio di un DLAMI

> **Note**
>
> Per questa guida, potremmo fare riferimenti specifici all'AMI Deep Learning (Ubuntu 18.04). Anche se selezioni un DLAMI diverso, dovresti essere in grado di seguire questa guida.

1. Trova l'ID del tuo DLAMI
2. Avvia un'istanza Amazon EC2 dal DLAMI

   Utilizzerai la console Amazon EC2. Segui le istruzioni dettagliate inAvvia dalla console Amazon EC2

   > **Tip**
   >
   > Opzione CLI: se scegli di attivare un DLAMI utilizzando laAWS CLI, avrai bisogno dell'ID dell'AMI, della regione e del tipo di istanza e delle informazioni del token di sicurezza. Assicurarsi di avere l'ID dell'AMI e dell'istanza a portata di mano. Se non hai ancora configurato laAWS CLI, fallo prima usando la guida per l'installazione dell'interfaccia a riga diAWS comando.

3. Una volta completata la procedura di una di queste opzioni, attendere che l'istanza sia pronta. Questo processo richiede in genere soltanto alcuni minuti. Puoi verificare lo stato dell'istanza nella console EC2.

Recupera l'ID DLAMI

Ogni AMI possiede un identificatore univoco (ID). Puoi interrogare l'ID per il DLAMI di tua scelta con l'interfaccia a riga diAWS comando (AWSCLI). Se non lo hai giàAWS CLI installato, consulta Guida introduttiva allaAWS CLI.

> **Note**
>
> Promemoria: nel AWS Deep Learning AMIcatalogo puoi trovare tutti i DLAMI e i relativi processori/acceleratori, sistema operativo, architettura di elaborazione, famiglie di istanze Amazon EC2 consigliate, stato del supporto e domande di recupero degli ID. Consulta anche

le note di rilascio DLAMINote di rilascio per DLAMI per ulteriori informazioni (driver, versioni python, tipo Amazon EBS).

1. Assicurati cheAWS le tue credenziali siano configurate.

```
aws configure
```

2. Usa il seguente comando per recuperare l'ID del tuo DLAMI o trovare la query fornita nelAWS Deep Learning AMI catalogo.

```
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**
>
> È possibile specificare una versione di rilascio per un determinato framework o ottenere l'ultima versione sostituendo il numero di versione con un punto interrogativo.

3. L'output visualizzato dovrebbe essere simile al seguente:

```
ami-094c089c38ed069f2
```

Copia questo ID DLAMI e premiq per uscire dal prompt.

Fase successiva

Avvia dalla console Amazon EC2

Avvia dalla console Amazon EC2

> **Note**
>
> Per avviare un'istanza con Elastic Fabric Adapter (EFA), consulta questi passaggi.

1. Apri la console EC2.

2. Notare la regione corrente nella parte superiore del riquadro di navigazione. Se non è quello che desideriRegione AWS, modifica questa opzione prima di procedere. Per ulteriori informazioni, consulta Regioni EC2.
3. Scegliere Launch Instance (Avvia istanza).
4. Inserisci un nome per la tua istanza e seleziona il DLAMI adatto a te.
   a. Trova un DLAMI esistente in Le mie AMI o scegli Avvio rapido.
   b. Ricerca per ID DLAMI. Sfoglia le opzioni, quindi seleziona la tua scelta.
5. Scegliere un tipo di istanza. Puoi trovare le famiglie di istanze consigliate per il tuo DLAMI nelAWS Deep Learning AMI Catalogo. Per consigli generali sui tipi di istanze DLAMI, consulta Selezione dell'istanza.

   > **Note**
   >
   > Se desideri utilizzare Elastic Inference (EI), fai clic su Configura i dettagli dell'istanza, seleziona Aggiungi un acceleratore Amazon EI, quindi seleziona la dimensione dell'acceleratore Amazon EI.

6. Scegliere Launch Instance (Avvia istanza).

> **Tip**
>
> Dai un'occhiata a Guida introduttiva alAWS deep learning utilizzando l'AMI Deep Learning per una guida dettagliata con schermate.

Fase successiva

Fase 2. Connect al DLAMI

Fase 2. Connect al DLAMI

Connect al DLAMI che hai avviato da un client (Windows, macOS o Linux). Per ulteriori informazioni, consulta Connect all'istanza Linux nella Guida per l'utente di Amazon EC2 per le istanze Linux.

Tieni una copia del comando di accesso SSH a portata di mano se intendi eseguire la configurazione di Jupyter dopo aver effettuato l'accesso. Ne utilizzerai una variante per la connessione alla pagina Web di Jupyter.

Fase successiva

Fase 3: Test del DLAMI

Fase 3: Test del DLAMI

A seconda della versione DLAMI, sono disponibili diverse opzioni di test:

- AMI di deep learning di— vai aUtilizzo dell'AMI Deep Learning con Conda.
- Deep Learning Learning Learning AMI— fare riferimento alla documentazione di installazione del framework desiderato.

Puoi anche creare un notebook Jupyter, provare dei tutorial o scrivere codice in Python. Per ulteriori informazioni, consulta Configurazione di un server di notebook Jupyter.

Passaggio 4: gestione dell'istanza DLAMI

Aggiorna regolarmente il sistema operativo e le altre applicazioni software utilizzate mediante l'installazione di patch e aggiornamenti non appena diventano disponibili.

Se utilizzi Amazon Linux o Ubuntu, quando accedi al tuo DLAMI, ricevi una notifica se sono disponibili aggiornamenti e vedi le istruzioni per l'aggiornamento. Per ulteriori informazioni sulla manutenzione di Amazon Linux, consulta Updating Instance Software. Per le istanze Ubuntu, consulta la documentazione ufficiale di Ubuntu.

In Windows, consulta Windows Update regolarmente per verificare se sono disponibili nuovi aggiornamenti software e di sicurezza. Se lo preferisci, puoi installare gli aggiornamenti automaticamente.

> **Important**
>
> Per informazioni sulle vulnerabilità di Meltdown e Spectre e su come correggere il sistema operativo per risolverle, consulta il Bollettino sulla sicurezzaAWS -2018-013.

Eliminare

Quando non hai più bisogno del DLAMI, puoi interromperlo o terminarlo per evitare di incorrere in addebiti continui. Un'istanza arrestata non viene chiusa e può essere riavviata in seguito. Le configurazioni, i file e altre informazioni non volatili vengono archiviati in un volume su Amazon S3. Per la conservazione del volume durante l'arresto dell'istanza ti verrà addebitato la tariffa S3 minore, ma non incorrerai nei costi relativi alle risorse di calcolo. Quando riavvii l'istanza, questa monterà quel volume e i tuoi dati saranno disponibili. Se termini un'istanza, questa non sarà più disponibile e non potrà essere riavviata. In realtà, i dati sono ancora presenti in S3, quindi per evitare ulteriori costi devi eliminare anche il volume. Per ulteriori istruzioni, consulta Termina l'istanza nella Guida per l'utente di Amazon EC2 per le istanze Linux.

Configurazione di un server di notebook Jupyter

Un server notebook Jupyter consente di creare ed eseguire notebook Jupyter dalla propria istanza DLAMI. Con i notebook Jupyter, è possibile condurre esperimenti di apprendimento automatico (ML) per l'addestramento e l'inferenza utilizzando l'AWSinfrastruttura e accedendo ai pacchetti integrati nel DLAMI. Per ulteriori informazioni sui notebook Jupyter, consulta la documentazione di Jupyter Notebook.

Per configurare un server notebook Jupyter, è necessario:

- Configura il server notebook Jupyter sulla tua istanza DLAMI Amazon EC2.
- Configurare il client in modo da eseguire la connessione al server di notebook Jupyter. Forniamo istruzioni di configurazione per client Windows, macOS e Linux.
- Testare la configurazione accedendo al server di notebook Jupyter.

Per completare i passaggi per configurare un Jupyter, segui le istruzioni nei seguenti argomenti. Dopo aver configurato un server notebook Jupyter, consultaTutorial per l'esecuzione di notebook Jupyter le informazioni sull'esecuzione dei notebook di esempio forniti nel DLAMI.

Argomenti

- Proteggi il tuo server Jupyter
- Avvio del server di notebook Jupyter
- Configurazione del client per la connessione al server Jupyter
- Test mediante l'accesso al server di notebook Jupyter

Proteggi il tuo server Jupyter

Di seguito abbiamo configurato Jupyter con SSL e una password personalizzata.

Connect all'istanza Amazon EC2 e completa la seguente procedura.

Configurazione del server Jupyter

1. Jupyter fornisce una utility per le password. Esegui il comando seguente e inserisci la tua password preferita al prompt.

   ```
   $ jupyter notebook password
   ```

   Il risultato sarà simile al seguente:

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

2. Crea un certificato SSL autofirmato Segui i prompt per compilare la tua località. È necessario immettere `.` se si desidera lasciare vuoto un prompt. Le tue risposte non hanno alcun impatto su queste funzionalità del certificato.

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

> **Note**
>
> Può voler creare un certificato SSL normale firmato da terze parti e che non causi un avviso di protezione da parte del browser. Questo processo è decisamente più compesso. Consulta la documentazione di Jupyter per ulteriori informazioni.

Fase successiva

Avvio del server di notebook Jupyter

Avvio del server di notebook Jupyter

Ora puoi attivare il server Jupyter accedendo all'istanza ed eseguendo il comando seguente che usa il certificato SSL creato nel passaggio precedente.

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

Dopo l'avvio del server, puoi collegarti allo stesso tramite un tunnel SSH dal tuo computer client. Durante l'esecuzione del server, vedrai un messaggio di Jupyter che conferma tale condizione. A questo punto, ignora il messaggio indicante che puoi accedere al server tramite un URL localhost, poiché non funzionerà fino a quando non crei un tunnel.

> **Note**
>
> Jupyter gestirà la commutazione di ambienti quando cambi framework Jupyter utilizzando l'interfaccia Web. Per ulteriori informazioni, consulta Passaggio a un altro ambiente con Jupyter.

Fase successiva

Configurazione del client per la connessione al server Jupyter

Configurazione del client per la connessione al server Jupyter

Dopo la configurazione del client per la connessione al server di notebook Jupyter, puoi creare e accedere ai notebook sul server nel workspace ed eseguire il codice di apprendimento profondo sul server.

Per informazioni sulla configurazione, scegli uno dei collegamenti seguenti.

Argomenti

Configurazione di un client Windows

Preparazione

Assicurati di disporre delle informazioni esposte di seguito, necessarie per configurare il tunnel SSH:

- Il nome DNS pubblico dell'istanza Amazon EC2. Puoi trovare tale nome nella console EC2.
- La coppia di chiavi per il file della chiave privata. Per ulteriori informazioni sull'accesso alla coppia di chiavi, consulta Coppia di chiavi Amazon EC2 nella Guida per l'utente per istanze Linux di Amazon EC2.

Utilizzo dei notebook Jupyter da un client Windows

Consulta queste guide sulla connessione all'istanza Amazon EC2 da un client Windows.

1. Risoluzione dei problemi di connessione all'istanza
2. Connessione all'istanza Linux da Windows tramite PuTTY

Per creare un tunnel su un server con Jupyter in esecuzione, un approccio consigliato è quello di installare Git Bash sul tuo client Windows, quindi seguire le istruzioni del client Linux/macOS . Tuttavia, puoi usare qualsiasi approccio desiderato per aprire un tunnel SSH con mapping di porta. Fai riferimento alla documentazione di Jupyter per ulteriori informazioni.

Fase successiva

Configurazione di un client Linux o macOS

Configurazione di un client Linux o macOS

1. Apri un terminale .
2. Esegui il seguente comando per inoltrare tutte le richieste dalla porta locale 8888 alla porta 8888 della tua istanza Amazon EC2 remota. Aggiorna il comando sostituendo la posizione della chiave per accedere all'istanza Amazon EC2 e il nome DNS pubblico della tua istanza Amazon EC2. Nota, per un'AMI Amazon Linux, il nome utente è `ec2-user` anziché `ubuntu`.

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

Questo comando apre un tunnel tra il client e l'istanza Amazon EC2 remota su cui è in esecuzione il server notebook Jupyter.

Fase successiva

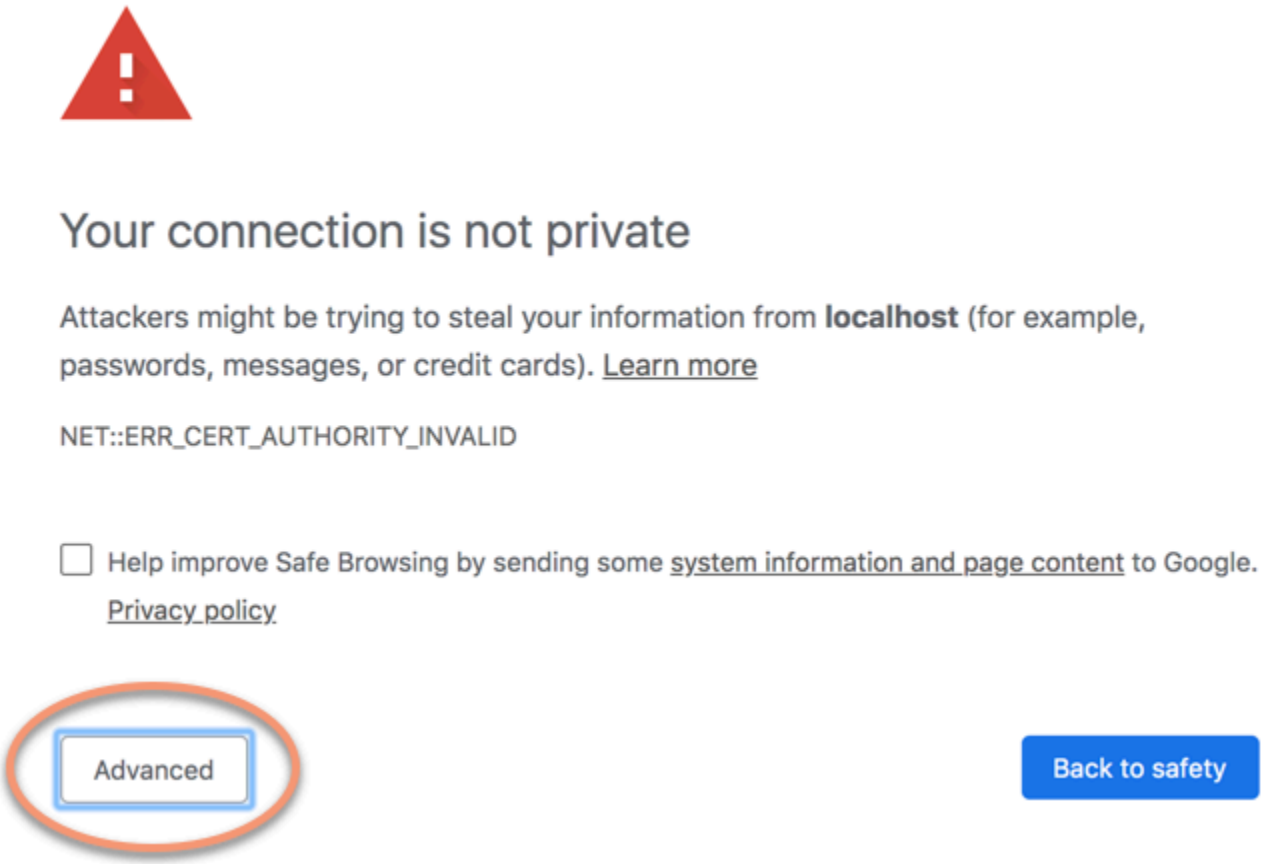
[Test mediante l'accesso al server di notebook Jupyter]

Test mediante l'accesso al server di notebook Jupyter

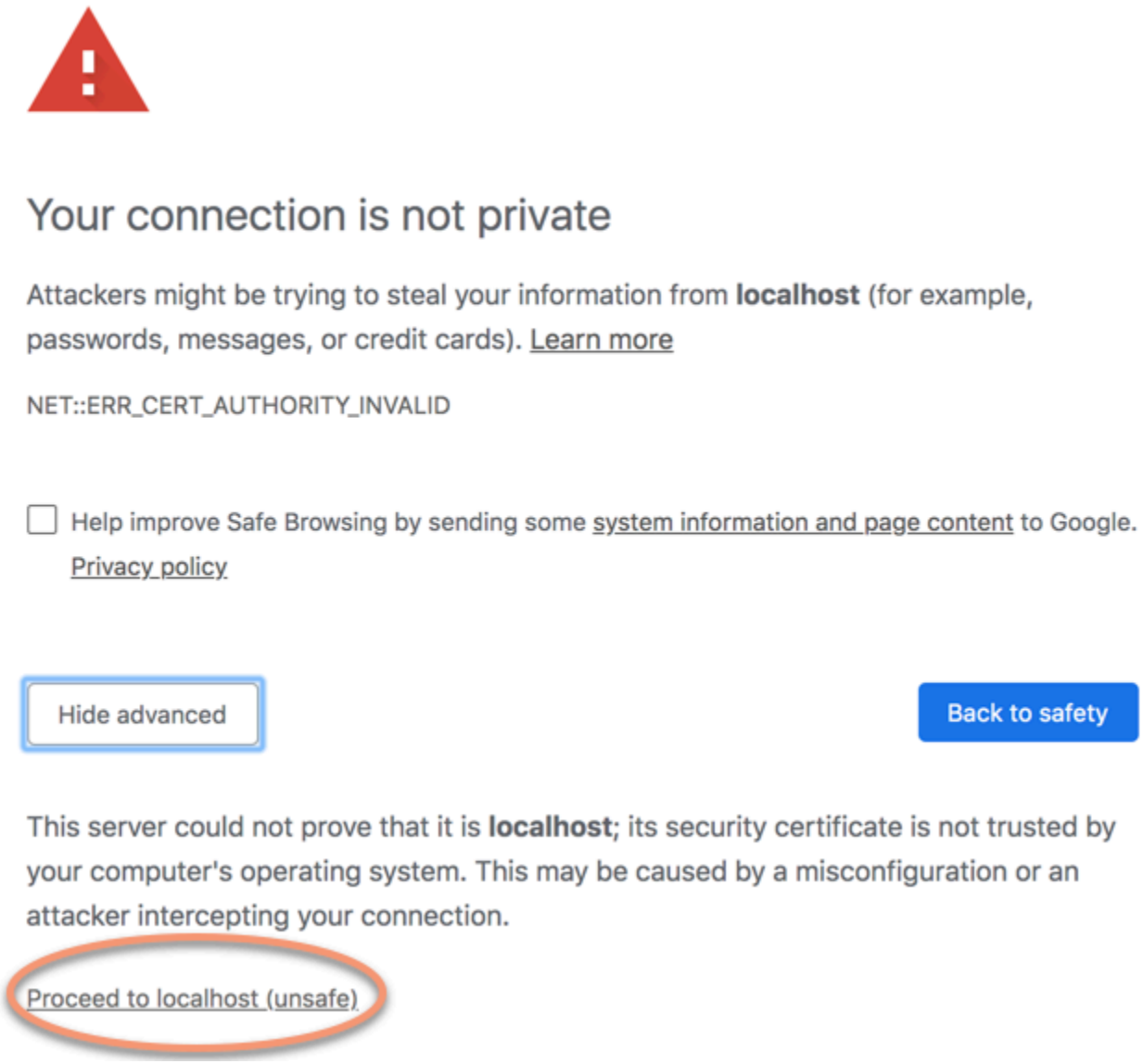
Ora sei pronto a accedere al server di notebook Jupyter.

La fase successiva consiste nel testare la connessione al server tramite il tuo browser.

1. Nella barra degli indirizzi del browser, digita il seguente URL, oppure fai clic su questo collegamento: https://localhost:8888
2. Con un certificato SSL autofirmato, il tuo browser ti mostrerà un avviso e ti verrà richiesto di evitare di proseguire nella visita del sito Web.



Poiché hai impostato tu stesso tale elemento, puoi proseguire in sicurezza.. A seconda del browser verrà visualizzato un pulsante denominato “avanzato”, “mostra dettagli” o simile.



Fai clic su questo elemento, quindi fai clic sul link "procedi verso il localhost". Se la connessione riesce, viene visualizzata la pagina Web del server di notebook Jupyter. A questo punto, ti verrà chiesta la password impostata precedentemente.

Ora hai accesso al server notebook Jupyter in esecuzione sul DLAMI. È possibile creare nuovi notebook o eseguire i Tutorial forniti.

Usare un DLAMI

Argomenti

Le sezioni seguenti descrivono come utilizzare l'AMI Deep Learning con Conda per cambiare ambiente, eseguire codice di esempio da ciascuno dei framework ed eseguire Jupyter in modo da poter provare diversi tutorial per notebook.

Utilizzo dell'AMI Deep Learning con Conda

Argomenti

Introduzione all'AMI Deep Learning con Conda

Conda è un sistema open source per la gestione di pacchetti e di ambienti eseguibile in Windows, macOS e Linux. Conda installa, esegue e aggiorna rapidamente i pacchetti e le relative dipendenze. Conda agevola la creazione, il salvataggio e il caricamento di ambienti sul computer locale nonché il passaggio dall'uno all'altro.

L'AMI Deep Learning con Conda è stata configurata per consentirti di passare facilmente da un ambiente di deep learning all'altro. Le istruzioni seguenti sono relative ad alcuni comandi `conda` di base. Ti consentono inoltre di verificare il corretto funzionamento dell'importazione di base del

framework e che puoi eseguire alcune semplici operazioni con il framework. È quindi possibile passare a tutorial più approfonditi forniti con DLAMI o agli esempi dei framework disponibili sul sito del progetto di ciascun framework.

Accedi al tuo DLAMI

Dopo aver effettuato l'accesso al server, verrà visualizzato un "messaggio del giorno" (MOTD) del server che descrive vari comandi Conda e che puoi utilizzare per passare da un framework di apprendimento profondo all'altro. Di seguito è riportato un esempio di MOTD. Il tuo MOTD specifico può variare man mano che vengono rilasciate nuove versioni di DLAMI.

> **Note**
>
> All'inizio della versione v28 non includiamo più gli ambienti CNTK, Caffe, Caffe2, Theano, Chainer e Keras Conda. AWS Deep Learning AMI Le versioni precedenti di quelle AWS Deep Learning AMI che contengono questi ambienti continueranno a essere disponibili. Tuttavia, verranno forniti aggiornamenti a questi ambienti solo se sono disponibili correzioni di protezione pubblicate dalla comunità open source per questi framework.

```
=============================================================================
      __|  __|_  )
      _|  (     /   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
```

Ogni comando Conda presenta il seguente modello:

`source activate framework_python-version`

Ad esempio, potresti vedere`for MXNet(+Keras1) with Python3 (CUDA 10.1) _____________________ source activate mxnet_p36`, il che significa che l'ambiente ha MXNet, Keras 1, Python 3 e CUDA 10.1. E per attivare questo ambiente, utilizzi il comando:

```
$ source activate mxnet_p36
```

Avvia l'ambiente TensorFlow

> **Note**
>
> Il caricamento del primo ambiente Conda può risultare alquanto lungo. L'AMI Deep Learning con Conda installa automaticamente la versione più ottimizzata del framework per l'istanza EC2 alla prima attivazione del framework. Non dovrebbero aversi ulteriori ritardi.

1. Attiva l'ambiente TensorFlow virtuale per Python 3.

   ```
   $ source activate tensorflow_p37
   ```

2. Avviare il terminale iPython.

```
(tensorflow_37)$ ipython
```

3. Esegui un TensorFlow programma rapido.

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

Viene visualizzato il messaggio "Hello, Tensorflow!".

Argomento successivo

Tutorial per l'esecuzione di notebook Jupyter

Passa all'ambiente PyTorch Python 3

Se sei ancora nella console IPython, `quit()` usa, quindi preparati a cambiare ambiente.

- Attiva l'ambiente PyTorch virtuale per Python 3.

```
$ source activate pytorch_p36
```

Prova del codice PyTorch

Per testare la tua installazione, usa Python per scrivere PyTorch codice che crea e stampa un array.

1. Avviare il terminale iPython.

```
(pytorch_p36)$ ipython
```

2. Importa PyTorch.

```
import torch
```

È possibile che venga visualizzato un messaggio di avviso su un pacchetto di terze parti. Puoi ignorarla.

3. Crea una matrice 5x3 con gli elementi inizializzati in modo casuale. Stampare la matrice.

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

Verificare il risultato.

```
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]])
```

Passa all'ambiente MXNet Python 3

Se sei ancora nella console IPython, `quit()` usa, quindi preparati a cambiare ambiente.

- Attivare l'ambiente virtuale MXNet per Python 3.

```
$ source activate mxnet_p36
```

Prova del codice MXNet

Per testare l'installazione, utilizza Python per scrivere codice MXNet che crea e stampa una matrice utilizzando l'API `NDArray`. Per ulteriori informazioni, consulta NDArray API.

1. Avviare il terminale iPython.

```
(mxnet_p36)$ ipython
```

2. Importare MXNet.

```
import mxnet as mx
```

È possibile che venga visualizzato un messaggio di avviso su un pacchetto di terze parti. Puoi ignorarla.

3. Creare una matrice 5x5, un'istanza di `NDArray`, con elementi inizializzati a 0. Stampare la matrice.

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

Verificare il risultato.

```
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)
```

Ulteriori esempi di MXNet sono disponibili nella sezione dedicata ai tutorial per MXNet.

Rimozione ambienti

Se esaurisci lo spazio sul DLAMI, puoi scegliere di disinstallare i pacchetti Conda che non stai utilizzando:

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

Utilizzo dell'AMI Deep Learning Base

Utilizzo dell'AMI Deep Learning Base

L'AMI Base include una piattaforma di base di driver GPU e librerie di accelerazione per distribuire l'ambiente di deep learning personalizzato. Per impostazione predefinita, l'AMI è configurata con l'ambiente NVIDIA CUDA 11.0. È anche possibile passare da una versione all'altra di CUDA. Consulta le seguenti istruzioni per eseguire questa operazione.

Configurazione delle versioni CUDA

Puoi verificare la versione CUDA eseguendo il programma NVIDIA. `nvcc`

```
nvcc --version
```

È possibile selezionare e verificare una particolare versione di CUDA con il seguente comando bash:

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

Per ulteriori informazioni, consulta le note di rilascio di Base DLAMI.

Tutorial per l'esecuzione di notebook Jupyter

I tutorial e gli esempi vengono forniti con ogni sorgente dei progetti di deep learning e nella maggior parte dei casi verranno eseguiti su qualsiasi DLAMI. Se scegli la AMI di deep learning di, beneficerai di alcuni tutorial selezionati già configurati e pronti per l'uso.

> **Important**
>
> Per eseguire i tutorial per notebook Jupyter installati sul DLAMI, è necessario. Configurazione di un server di notebook Jupyter

Una volta che il server Jupyter è in esecuzione, puoi eseguire i tutorial mediante il browser web. Se utilizzi l'AMI Deep Learning con Conda o se hai configurato ambienti Python, puoi cambiare i kernel Python dall'interfaccia del notebook Jupyter. Seleziona il kernel appropriato prima di eseguire un tutorial specifico di un framework. Ulteriori esempi di ciò sono forniti agli utenti dell'AMI Deep Learning con Conda.

> **Note**
>
> Molti tutorial richiedono moduli Python aggiuntivi che potrebbero non essere configurati sul DLAMI. Se ricevi un errore del tipo`"xyz module not found"`, accedi al DLAMI, attiva l'ambiente come descritto sopra, quindi installa i moduli necessari.

> **Tip**
>
> I tutorial e gli esempi di apprendimento profondo sono spesso basati su una o più GPU. Se il tuo tipo di istanza non dispone di una GPU, è possibile che sia necessario modificare una parte del codice dell'esempio affinché venga eseguito.

Esplorazione dei tutorial installati

Dopo aver effettuato l'accesso al server Jupyter e aver visualizzato la directory dei tutorial (solo su Deep Learning AMI con Conda), ti verranno presentate cartelle di tutorial per ogni nome di framework. Se non vedi un framework nell'elenco, i tutorial per quel framework non sono disponibili sul tuo DLAMI corrente. Fai clic sul nome del framework per visualizzare i tutorial elencati, quindi fai clic su un tutorial per avviarlo.

La prima volta che esegui un notebook sull'AMI Deep Learning con Conda, vorrà sapere quale ambiente desideri utilizzare. Ti verrà richiesto di selezionarlo da un elenco. Ogni ambiente è denominato in base a questo modello:

`Environment (conda_framework_python-version)`

Ad esempio, è possibile che sia visualizzato `Environment (conda_mxnet_p36)`, il che significa che l'ambiente include MXNet e Python 3. L'altra variazione sarebbe `Environment (conda_mxnet_p27)`, a indicare che l'ambiente include MXNet e Python 2.

> **Tip**
>
> Se sei preoccupato per quale versione di CUDA è attiva, un modo per vederlo è nel MOTD quando accedi per la prima volta al DLAMI.

Passaggio a un altro ambiente con Jupyter

Se decidi di provare un tutorial per un altro framework, assicurati di verificare il kernel attualmente in esecuzione. Questa informazione può essere visualizzata in alto a destra nell'interfaccia Jupyter, sotto il pulsante di disconnessione. Puoi cambiare il kernel su qualsiasi notebook aperto scegliendo la voce del menu Jupyter Kernel, quindi Change Kernel (Cambia kernel) e infine facendo clic sull'ambiente appropriato per il notebook in esecuzione.

A questo punto, devi rieseguire tutte le celle in quanto una modifica nel kernel cancellerà lo stato di quanto eseguito in precedenza.

> **Tip**
>
> Passare da un framework all'altro può risultare divertente e istruttivo, ma esiste il rischio di esaurimento della memoria. Se appaiono degli errori, esamina la finestra del terminale

in cui il server Jupyter è in esecuzione. Qui sono presenti messaggi utili e la registrazione degli errori e potresti visualizzare un errore. out-of-memory Per correggere il problema, puoi accedere alla home page del server Jupyter, fare clic sulla scheda Running (In esecuzione) e quindi su Shutdown (Chiusura) per ogni tutorial che probabilmente è ancora in esecuzione in background e che utilizza tutta la memoria.

Argomento successivo

Per ulteriori esempi e codice di esempio di ogni framework, fai clic su Next (Avanti) oppure passa a Apache MXNet (incubazione).

Tutorial

Di seguito sono riportati dei tutorial su come utilizzare l'AMI Deep Learning con il software di Conda.

Argomenti

- Tutorial di 10 minuti
- Attivazione di framework
- Debug e visualizzazione
- Training distribuito
- Elastic Fabric Adapter
- Monitoraggio e ottimizzazione GPU
- Il chip AWS Inferentia con DLAMI
- Il Graviton DLAMI
- I DLAMI dell'Habana
- Inferenza
- Utilizzo di frameworks con ONNX
- Model serving

Tutorial di 10 minuti

- Avvia un AWS Deep Learning AMI (in 10 minuti)

- Addestra un modello di deep learning con DLC su Amazon EC2 (in 10 minuti)

Attivazione di framework

Di seguito sono riportati i framework di deep learning installati sull'AMI Deep Learning con Conda. Fai clic su un framework per informazioni su come attivarlo.

Argomenti

Apache MXNet (incubazione)

Attivazione di Apache MXNet (incubazione)

Questo tutorial mostra come attivare MXNet su un'istanza che esegue l'AMI Deep Learning con Conda (DLAMI su Conda) ed eseguire un programma MXNet.

Quando viene rilasciato un pacchetto Conda stabile di un framework, viene testato e preinstallato sul DLAMI. Se desideri eseguire la build notturna più recente non testata, puoi eseguire l'Installazione di una build notturna (sperimentale) di MXNet manualmente.

Per eseguire MXNet su DLAMI con Conda

1. Per attivare il framework, apri un'istanza Amazon Elastic Compute Cloud (Amazon EC2) di DLAMI con Conda.
   - Per MXNet e Keras 2 su Python 3 con CUDA 9.0 e MKL-DNN, eseguire questo comando:

```
$ source activate mxnet_p36
```

- Per MXNet e Keras 2 su Python 2 con CUDA 9.0 e MKL-DNN, eseguire questo comando:

```
$ source activate mxnet_p27
```

2. Avviare il terminale iPython.

```
(mxnet_p36)$ ipython
```

3. Eseguire un programma MXNet rapido. Creare una matrice 5x5, un'istanza di `NDArray`, con elementi inizializzati a 0. Stampare la matrice.

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

4. Verificare il risultato.

```
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)
```

Installazione di una build notturna (sperimentale) di MXNet

Puoi installare la build MXNet più recente in uno o entrambi gli ambienti MXNet Conda sulla tua AMI Deep Learning con Conda.

Per installare MXNet da una build notturna

1. • Per l'ambiente MXNet con Python 3, eseguire questo comando:

```
$ source activate mxnet_p36
```

- Per l'ambiente MXNet con Python 2, eseguire questo comando:

```
$ source activate mxnet_p27
```

2. Rimuovere la versione attualmente installata di MXNet.

> **Note**
>
> Per gli altri passaggi, si presuppone che venga utilizzato l'ambiente `mxnet_p36`.

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

3. Installare l'ultima build notturna di MXNet.

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

4. Per verificare l'installazione della build notturna più recente, avviare il terminale IPython e controllare la versione di MXNet.

```
(mxnet_p36)$ ipython
```

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

L'output deve stampare l'ultima versione stabile di MXNet.

Altri tutorial

Puoi trovare altri tutorial nella cartella Deep Learning AMI with Conda tutorials, che si trova nella home directory del DLAMI.

1. Usa Apache MXNet (incubazione) per l'inferenza con un modello 5.0 ResNet
2. Usa Apache MXNet (incubazione) per l'inferenza con un modello ONNX
3. Model Server for Apache MXNet (MMS)

Per ulteriori tutorial ed esempi, consulta la documentazione ufficiale di Python del framework, l'API Python per MXNet o il sito Web Apache MXNet.

Caffe2

> **Note**
>
> Non includiamo più gli ambienti CNTK, Caffe, Caffe2 e Theano Conda in AWS Deep Learning AMI a partire dalla versione v28. Le versioni precedenti di quelle AWS Deep Learning AMI che contengono questi ambienti continueranno a essere disponibili. Tuttavia, verranno forniti aggiornamenti a questi ambienti solo se sono disponibili correzioni di protezione pubblicate dalla comunità open source per questi framework.

Tutorial di Caffe2

Per attivare il framework, segui queste istruzioni sulla tua AMI Deep Learning con Conda.

È disponibile soltanto l'opzione Python 2 con CUDA 9 e cuDNN 7:

```
$ source activate caffe2_p27
```

Avviare il terminale iPython.

```
(caffe2_p27)$ ipython
```

Eseguire un programma Caffe2 rapido.

```
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)
```

Dovrebbero essere visualizzate le matrici random numpy stampate e quindi quelle caricate in un blob Caffe2. Da notare che queste sono identiche dopo il caricamento.

Altri tutorial

Per ulteriori tutorial ed esempi, fai riferimento ai documenti Python ufficiali del framework, all'API Python per Caffe2 e al sito Web Caffe2.

Chainer

> **Note**
>
> Non includiamo più gli ambienti Chainer Conda all'inizio della AWS Deep Learning AMI versione v28. Le versioni precedenti di quelle AWS Deep Learning AMI che contengono questi ambienti continueranno a essere disponibili. Tuttavia, verranno forniti aggiornamenti a questi ambienti solo se sono disponibili correzioni di protezione pubblicate dalla comunità open source per questi framework.

Chainer è un framework flessibile basato su Python che consente di scrivere facilmente e intuitivamente delle architetture di rete neurali complesse. Chainer facilita l'utilizzo di molteplici istanze GPU per il training. Inoltre, registra automaticamente i risultati, rappresenta graficamente la perdita e la precisione e genera un output per la visualizzazione della rete neurale con un grafico di elaborazione. È incluso nell'AMI Deep Learning con Conda (DLAMI con Conda).

Attivazione di Chainer

1. Connect all'istanza che esegue l'AMI Deep Learning con Conda. Consulta la documentazione di Amazon EC2 the section called “Selezione dell'istanza” o consulta la documentazione di Amazon EC2 su come selezionare o connettersi a un'istanza.
2.
   - Attivare l'ambiente Python 3 Chainer:

     ```
     $ source activate chainer_p36
     ```

   - Attivare l'ambiente Python 2 Chainer:

     ```
     $ source activate chainer_p27
     ```

3. Avviare il terminale iPython:

   ```
   (chainer_p36)$ ipython
   ```

4. Testare l'importazione di Chainer per verificare che funziona correttamente:

```
import chainer
```

È possibile che siano visualizzati alcuni messaggi di avviso, ma nessun messaggio di errore.

Ulteriori informazioni

- Prova i tutorial per Chainer.
- La cartella degli esempi `Chainer` nell'origine scaricata precedentemente contiene ulteriori esempi. Provali per verificarne le prestazioni.
- Per ulteriori informazioni su Chainer, consulta il sito Web della documentazione di Chainer.

CNTK

> **Note**
>
> Non includiamo più gli ambienti CNTK, Caffe, Caffe2 e Theano Conda in AWS Deep Learning AMI a partire dalla versione v28. Le versioni precedenti di quelle AWS Deep Learning AMI che contengono questi ambienti continueranno a essere disponibili. Tuttavia, verranno forniti aggiornamenti a questi ambienti solo se sono disponibili correzioni di protezione pubblicate dalla comunità open source per questi framework.

Attivazione di CNTK

Questo tutorial mostra come attivare CNTK su un'istanza che esegue l'AMI Deep Learning con Conda (DLAMI su Conda) ed eseguire un programma CNTK.

Quando viene rilasciato un pacchetto Conda stabile di un framework, viene testato e preinstallato sul DLAMI. Se desideri eseguire la build notturna più recente non testata, puoi eseguire l'Installazione di una build notturna (sperimentale) di CNTK manualmente.

Per eseguire CNTK su DLAMI con Conda

1. Per attivare CNTK, apri un'istanza Amazon Elastic Compute Cloud (Amazon EC2) di DLAMI con Conda.
   - Per Python 3 con CUDA 9 e cuDNN 7:

```
$ source activate cntk_p36
```

- Per Python 2 con CUDA 9 e cuDNN 7:

```
$ source activate cntk_p27
```

2. Avviare il terminale iPython.

```
(cntk_p36)$ ipython
```

3. • Se si dispone di un'istanza CPU, eseguire questo rapido programma CNTK.

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

Dovrebbe essere visualizzata la versione CNTK, quindi l'output di una matrice 2x3 di 3.

- Se si dispone di un'istanza GPU, è possibile testarla con il seguente codice di esempio. Se CNTK può accedere alla GPU, il risultato dovrebbe essere `True`.

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

Installazione di una build notturna (sperimentale) di CNTK

Puoi installare la build CNTK più recente in uno o entrambi gli ambienti CNTK Conda sulla tua AMI Deep Learning con Conda.

Per installare CNTK da una build notturna

1. • Per CNTK e Keras 2 su Python 3 con CUDA 9.0 e MKL-DNN, eseguire questo comando:

```
$ source activate cntk_p36
```

- Per CNTK e Keras 2 su Python 2 con CUDA 9.0 e MKL-DNN, eseguire questo comando:

```
$ source activate cntk_p27
```

2. Per gli altri passaggi, si presuppone che venga utilizzato l'ambiente `cntk_p36`. Rimuovere la versione attualmente installata di CNTK.

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

3. Per installare la build notturna di CNTK, è necessario prima trovare la versione che si desidera installare dal sito Web di CNTK.

4.
   - (Opzione per istanze GPU) - Per installare la build notturna, utilizzare quanto segue, sostituendo nella build desiderata:

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

     Sostituire l'URL nel comando precedente con la versione GPU dell'ambiente Python corrente.

   - (Opzione per istanze CPU) - Per installare la build notturna, utilizzare quanto segue, sostituendo nella build desiderata:

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

     Sostituire l'URL nel comando precedente con la versione CPU dell'ambiente Python corrente.

5. Per verificare l'installazione della build notturna più recente, avviare il terminale IPython e controllare la versione di CNTK.

```
(cntk_p36)$ ipython
```

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

L'output dovrebbe essere simile a `2.6-rc0.dev20181015`

Altri tutorial

Per ulteriori tutorial ed esempi, consulta i documenti Python ufficiali del framework, l'API Python per CNTK o il sito Web CNTK.

Keras

Tutorial di Keras

1. Per attivare il framework, utilizzare questi comandi sull'interfaccia a riga di comando per l'the section called “Conda DLAMI”.
   - Per Keras 2 con un back-end MXNet su Python 3 con CUDA 9 e cuDNN 7:

     ```
     $ source activate mxnet_p36
     ```
   - Per Keras 2 con un back-end MXNet su Python 2 con CUDA 9 e cuDNN 7:

     ```
     $ source activate mxnet_p27
     ```
   - Per Keras 2 con un TensorFlow backend su Python 3 con CUDA 9 con cuDNN 7:

     ```
     $ source activate tensorflow_p36
     ```
   - Per Keras 2 con un TensorFlow backend su Python 2 con CUDA 9 con cuDNN 7:

     ```
     $ source activate tensorflow_p27
     ```
2. Per testare l'importazione di Keras allo scopo di verificare quale back-end è attivato, utilizzare questi comandi:

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

   Sullo schermo viene visualizzato il messaggio seguente:

   ```
   Using MXNet backend
   ```

   Se Keras utilizza, viene visualizzato quanto segue: TensorFlow

   ```
   Using TensorFlow backend
   ```

> **Note**
>
> Se viene visualizzato un messaggio di errore o se il back-end non corretto è ancora in uso, è possibile aggiornare la configurazione di Keras manualmente. Modificare il file `~/.keras/keras.json` e impostare il parametro di back-end su `mxnet` o `tensorflow`.

Altri tutorial

- Per un tutorial sul training con molteplici GPU utilizzando Keras con un back-end MXNet, consulta Tutorial per il training di molteplici GPU con Keras-MXNet.
- Puoi trovare esempi di Keras con un backend MXNet nella directory Deep Learning AMI with Conda. `~/examples/keras-mxnet`
- Puoi trovare esempi di Keras con un TensorFlow backend nella directory Deep Learning AMI with `~/examples/keras` Conda.
- Per ulteriori tutorial ed esempi, consulta il sito Web Keras.

PyTorch

Attivazione PyTorch

Quando viene rilasciato un pacchetto Conda stabile di un framework, viene testato e preinstallato sul DLAMI. Se desideri eseguire la build notturna più recente non testata, puoi eseguire l'Installa PyTorch Nightly Build (sperimentale) manualmente.

Per attivare il framework attualmente installato, segui queste istruzioni sulla tua AMI Deep Learning con Conda.

Per PyTorch su Python 3 con CUDA 10 e MKL-DNN, esegui questo comando:

```
$ source activate pytorch_p36
```

Per PyTorch su Python 2 con CUDA 10 e MKL-DNN, esegui questo comando:

```
$ source activate pytorch_p27
```

Avviare il terminale iPython.

```
(pytorch_p36)$ ipython
```

Esegui un programma rapido. PyTorch

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

Dovrebbe essere visualizzata la matrice random iniziale stampata, quindi le dimensioni della stessa e infine l'aggiunta di un'altra matrice random.

Installa PyTorch Nightly Build (sperimentale)

Come eseguire l'installazione PyTorch da una build notturna

Puoi installare la PyTorch build più recente in uno o entrambi gli ambienti PyTorch Conda sulla tua AMI Deep Learning con Conda.

1. • (Opzione per Python 3) - Attiva l'ambiente Python 3: PyTorch

   ```
   $ source activate pytorch_p36
   ```

   • (Opzione per Python 2) - Attiva l'ambiente Python 2: PyTorch

   ```
   $ source activate pytorch_p27
   ```

2. Per gli altri passaggi, si presuppone che venga utilizzato l'ambiente `pytorch_p36`. Rimuovi il file attualmente installato: PyTorch

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

3. • (Opzione per istanze GPU) - Installa l'ultima build notturna di PyTorch CUDA 10.0:

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

- (Opzione per istanze CPU) - Installa l'ultima build notturna per le istanze senza GPU: PyTorch

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

4. Per verificare di aver installato correttamente l'ultima nightly build, avvia il terminale IPython e controlla la versione di. PyTorch

```
(pytorch_p36)$ ipython
```

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

L'output dovrebbe essere simile a `1.0.0.dev20180922`

5. Per verificare che la PyTorch nightly build funzioni bene con l'esempio MNIST, puoi eseguire uno script di test dal repository degli esempi: 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
```

Altri tutorial

Puoi trovare altri tutorial nella cartella Deep Learning AMI with Conda tutorials nella home directory del DLAMI. Per ulteriori tutorial ed esempi, fate riferimento ai documenti ufficiali, alla documentazione e al sito Web del framework. PyTorch PyTorch

- PyTorch da ONNX a MXNet Tutorial
- PyTorch da ONNX a CNTK Tutorial

TensorFlow

Attivazione TensorFlow

Questo tutorial mostra come attivare TensorFlow su un'istanza che esegue l'AMI Deep Learning con Conda (DLAMI su Conda) ed eseguire un programma. TensorFlow

Quando viene rilasciato un pacchetto Conda stabile di un framework, viene testato e preinstallato sul DLAMI. Se desideri eseguire la build notturna più recente non testata, puoi eseguire l'Installa TensorFlow Nightly Build (sperimentale) manualmente.

Per funzionare TensorFlow su DLAMI con Conda

1. Per l'attivazione TensorFlow, apri un'istanza Amazon Elastic Compute Cloud (Amazon EC2) di DLAMI con Conda.
   - Per TensorFlow e Keras 2 su Python 3 con CUDA 9.0 e MKL-DNN, esegui questo comando:

     ```
     $ source activate tensorflow_p36
     ```
   - Per TensorFlow e Keras 2 su Python 2 con CUDA 9.0 e MKL-DNN, esegui questo comando:

     ```
     $ source activate tensorflow_p27
     ```
2. Avviare il terminale iPython:

   ```
   (tensorflow_p36)$ ipython
   ```
3. Esegui un TensorFlow programma per verificare che funzioni correttamente:

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

   Viene visualizzato `Hello, TensorFlow!`.

Installa TensorFlow Nightly Build (sperimentale)

Puoi installare la TensorFlow build più recente in uno o entrambi gli ambienti TensorFlow Conda sulla tua AMI Deep Learning con Conda.

Da installare TensorFlow da una build notturna

1. • Per l' TensorFlow ambiente Python 3, esegui il seguente comando:

```
$ source activate tensorflow_p36
```

- Per l' TensorFlow ambiente Python 2, esegui il seguente comando:

```
$ source activate tensorflow_p27
```

2. Rimuovi il file attualmente installato TensorFlow.

> **Note**
>
> Per gli altri passaggi, si presuppone che venga utilizzato l'ambiente `tensorflow_p36`.

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

3. Installa l'ultima build notturna di. TensorFlow

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

4. Per verificare di aver installato correttamente l'ultima nightly build, avvia il terminale IPython e controlla la versione di. TensorFlow

```
(tensorflow_p36)$ ipython
```

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

L'output dovrebbe essere simile a `1.12.0-dev20181012`

Altri tutorial

TensorFlow con Horovod

TensorBoard

TensorFlow Servire

Per i tutorial, consultate la cartella chiamata `Deep Learning AMI with Conda tutorials` nella home directory del DLAMI.

Per altri tutorial ed esempi, consulta la TensorFlow documentazione per l'API TensorFlow Python o visita il sito Web. TensorFlow

TensorFlow 2

Questo tutorial mostra come attivare TensorFlow 2 su un'istanza che esegue l'AMI Deep Learning con Conda (DLAMI su Conda) ed eseguire un programma TensorFlow 2.

Quando viene rilasciato un pacchetto Conda stabile di un framework, viene testato e preinstallato sul DLAMI. Se desideri eseguire la build notturna più recente non testata, puoi eseguire l'Installa Nightly Build di TensorFlow 2 (sperimentale) manualmente.

Attivazione 2 TensorFlow

Per funzionare TensorFlow su DLAMI con Conda

1. Per attivare TensorFlow 2, apri un'istanza Amazon Elastic Compute Cloud (Amazon EC2) di DLAMI con Conda.
2. Per TensorFlow 2 e Keras 2 su Python 3 con CUDA 10.1 e MKL-DNN, esegui questo comando:

   ```
   $ source activate tensorflow2_p36
   ```
3. Avviare il terminale iPython:

   ```
   (tensorflow2_p36)$ ipython
   ```
4. Esegui un programma TensorFlow 2 per verificare che funzioni correttamente:

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

   Viene visualizzato `Hello, TensorFlow!`.

Installa Nightly Build di TensorFlow 2 (sperimentale)

Puoi installare le ultime TensorFlow 2 build in uno o entrambi i TensorFlow 2 ambienti Conda sulla tua AMI Deep Learning con Conda.

Da installare TensorFlow da una build notturna

1. Per l'ambiente Python 3 TensorFlow 2, esegui il seguente comando:

```
$ source activate tensorflow2_p36
```

2. Rimuovi il file attualmente installato TensorFlow.

> **Note**
>
> Per gli altri passaggi, si presuppone che venga utilizzato l'ambiente `tensorflow2_p36`.

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

3. Installa l'ultima build notturna di. TensorFlow

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

4. Per verificare di aver installato correttamente l'ultima nightly build, avvia il terminale IPython e controlla la versione di. TensorFlow

```
(tensorflow2_p36)$ ipython
```

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

L'output dovrebbe essere simile a `2.1.0-dev20191122`

Altri tutorial

Per i tutorial, consultate la cartella chiamata `Deep Learning AMI with Conda tutorials` nella home directory del DLAMI.

Per altri tutorial ed esempi, consulta la TensorFlow documentazione per l'API TensorFlow Python o visita il sito Web. TensorFlow

TensorFlow con Horovod

Questo tutorial mostra come eseguire l'attivazione TensorFlow con Horovod su un (AWS Deep Learning AMI DLAMI) con Conda. Horovod è preinstallato negli ambienti Conda per. TensorFlow L'ambiente Python3 è consigliato.

> **Note**
>
> Solo i tipi di istanza P3.*, P2.* e G3.* sono supportati.

Per attivare TensorFlow e testare Horovod sul DLAMI con Conda

1. Apri un'istanza Amazon Elastic Compute Cloud (Amazon EC2) di DLAMI con Conda. Per informazioni su come iniziare a usare un DLAMI, consulta. the section called “Come iniziare con il DLAMI”
2. (Consigliato) Per TensorFlow 1.15 con Horovod su Python 3 con CUDA 11, esegui il seguente comando:

   ```
   $ source activate tensorflow_p37
   ```

3. Avviare il terminale iPython:

   ```
   (tensorflow_p37)$ ipython
   ```

4. Prova l'importazione TensorFlow con Horovod per verificare che funzioni correttamente:

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

   Sullo schermo può essere visualizzato quanto segue (è possibile ignorare gli eventuali messaggi di avviso).

   ```
   --------------------------------------------------------------------------
   [[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.
--------------------------------------------------------------------------
```

Ulteriori informazioni

- TensorFlow con Horovod
- Per i tutorial, consultate la `examples/horovod` cartella nella home directory del DLAMI.
- Per ulteriori tutorial ed esempi, consultate il progetto Horovod. GitHub

TensorFlow 2 con Horovod

Questo tutorial mostra come attivare TensorFlow 2 con Horovod su un (AWS Deep Learning AMI DLAMI) con Conda. Horovod è preinstallato negli ambienti Conda per 2. TensorFlow L'ambiente Python3 è consigliato.

> **Note**
>
> Solo i tipi di istanza P3.*, P2.* e G3.* sono supportati.

Attivare TensorFlow 2 e testare Horovod sul DLAMI con Conda

1. Apri un'istanza Amazon Elastic Compute Cloud (Amazon EC2) di DLAMI con Conda. Per informazioni su come iniziare a usare un DLAMI, consulta. the section called “Come iniziare con il DLAMI”
   - (Consigliato) Per TensorFlow 2 con Horovod su Python 3 con CUDA 10, esegui questo comando:

     ```
     $ source activate tensorflow2_p36
     ```

   - Per TensorFlow 2 con Horovod su Python 2 con CUDA 10, esegui questo comando:

```
$ source activate tensorflow2_p27
```

2. Avviare il terminale iPython:

```
(tensorflow2_p36)$ ipython
```

3. Prova a importare TensorFlow 2 con Horovod per verificare che funzioni correttamente:

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

Se non ricevi alcun output, Horovod funziona correttamente. Sullo schermo può essere visualizzato quanto segue (è possibile ignorare gli eventuali messaggi di avviso).

```
--------------------------------------------------------------------------
[[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.
--------------------------------------------------------------------------
```

Ulteriori informazioni

- Per i tutorial, consultate la `examples/horovod` cartella nella home directory del DLAMI.
- Per ulteriori tutorial ed esempi, consultate il progetto Horovod. GitHub

Theano

Tutorial di Theano

> **Note**
>
> Non includiamo più gli ambienti CNTK, Caffe, Caffe2 e Theano Conda in AWS Deep Learning AMI a partire dalla versione v28. Le versioni precedenti di AWS Deep Learning AMI che contengono questi ambienti continueranno a essere disponibili. Tuttavia, verranno forniti aggiornamenti a questi ambienti solo se sono disponibili correzioni di protezione pubblicate dalla comunità open source per questi framework.

Per attivare il framework, segui queste istruzioni sulla tua AMI Deep Learning con Conda.

Per Theano + Keras in Python 3 con CUDA 9 e cuDNN 7:

```
$ source activate theano_p36
```

Per Theano + Keras in Python 2 con CUDA 9 e cuDNN 7:

```
$ source activate theano_p27
```

Avviare il terminale iPython.

```
(theano_p36)$ ipython
```

Eseguire un programma Theano rapido.

```
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)
```

Dovrebbe essere visualizzato il calcolo di un gradiente simbolico con Theano.

Altri tutorial

Per ulteriori tutorial ed esempi, fare riferimento ai documenti ufficiali del framework, all'API Theano Python e al sito Web Theano.

Debug e visualizzazione

Scopri le opzioni di debug e visualizzazione per DLAMI. Fai clic su quella desiderata per informazioni su come utilizzarla.

Argomenti

- MXBoard
- TensorBoard

MXBoard

MXBoard consente di ispezionare e interpretare visivamente le esecuzioni e i grafici di MXNet utilizzando il software. TensorBoard Un server Web che gestisce una pagina Web viene eseguito per visualizzare le visualizzazioni MXBoard e interagire con le stesse.

MXNet e MXBoard sono preinstallati con l'AMI Deep Learning con Conda (DLAMI con Conda). TensorBoard In questo tutorial, si utilizza una funzione MXBoard per generare log compatibili con. TensorBoard

Argomenti

- Utilizzo di MXNet con MXBoard
- Ulteriori informazioni

Utilizzo di MXNet con MXBoard

Genera dati di registro MxBoard compatibili con TensorBoard

1. Connettiti alla tua istanza Amazon Elastic Compute Cloud (Amazon EC2) di DLAMI con Conda.
2. Attivare l'ambiente Python 3 MXNet.

```
$ source activate mxnet_p36
```

3. Preparare uno script Python per la scrittura di dati generati dall'operatore normale in un file di eventi. I dati vengono generati dieci volte con una deviazione standard decrescente, quindi scritti ogni volta sul file di eventi. La distribuzione dei dati si accentrerà progressivamente intorno al valore medio. Da notare che il file di eventi sarà specificato nella cartella dei log. Passi questo percorso della cartella al file binario. TensorBoard

```
$ vi mxboard_normal.py
```

4. Incollare quanto segue nel file e salvarlo:

```
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. Eseguire lo script. Questa operazione genererà i log in una cartella `logs` che è possibile utilizzare per le visualizzazioni.

```
$ python mxboard_normal.py
```

6. Ora devi passare all' TensorFlow ambiente da usare TensorBoard e a MXBoard per visualizzare i log. Questa è una dipendenza richiesta per MxBoard e. TensorBoard

```
$ source activate tensorflow_p36
```

7. Impostare `tensorboard` come posizione dei log:

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

   TensorBoard avvia il server web di visualizzazione sulla porta 8888.

8. Per un facile accesso dal browser locale, è possibile impostare la porta 80 o un'altra porta per il server Web. Qualunque sia la porta che utilizzi, dovrai aprire questa porta nel gruppo di sicurezza EC2 per il tuo DLAMI. È anche possibile utilizzare l'inoltro alla porta. Per istruzioni

su come modificare le impostazioni del gruppo di sicurezza e sull'inoltro alla porta, consulta Configurazione di un server di notebook Jupyter. Le impostazioni predefinite sono descritte nella fase successiva.

> **Note**
>
> Se è necessario eseguire il server Jupyter e il server MXBoard, utilizzare una porta differente per ognuno.

9. Aprire la porta 8888 (o la porta assegnata al server Web di visualizzazione) sull'istanza EC2.

   a. Apri la tua istanza EC2 nella console Amazon EC2 all'indirizzo https://console.aws.amazon.com/ec2/.

   b. Nella console Amazon EC2, scegli Rete e sicurezza, quindi scegli Gruppi di sicurezza.

   c. Per Security Group (Gruppo di sicurezza), scegliere l'ultimo gruppo creato (vedere il timestamp nella descrizione).

   d. Scegliere la scheda Inbound (In entrata), quindi scegliere Edit (Modifica).

   e. Selezionare Add Rule (Aggiungi regola).

   f. Nella nuova riga, digita quanto segue:

      Type (Tipo): **`TCP Rule`** Custom (Personalizzato)

      Protocol (Protocollo): **`TCP`**

      Port Range (Intervallo porte): **`8888`** (o la porta assegnata al server di visualizzazione)

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

10. Se si desidera visualizzare i dati dal browser locale, digitare il comando seguente per inoltrare i dati di cui viene eseguito il rendering sull'istanza EC2 al computer locale.

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

11. Apri la pagina web per le visualizzazioni di MxBoard utilizzando l'IP pubblico o l'indirizzo DNS dell'istanza EC2 che esegue DLAMI con Conda e la porta che hai aperto per MxBoard:

    `http://127.0.0.1:8888`

Ulteriori informazioni

Per ulteriori informazioni su MXBoard, consulta il sito Web di MXBoard.

TensorBoard

TensorBoardconsente di ispezionare e interpretare visivamente le TensorFlow esecuzioni e i grafici. Gestisce un server Web che serve una pagina Web per la visualizzazione e l'interazione con le visualizzazioni. TensorBoard

TensorFlow e TensorBoard sono preinstallati con l'AMI Deep Learning con Conda (DLAMI con Conda). Il DLAMI con Conda include anche uno script di esempio che utilizza TensorFlow per addestrare un modello MNIST con funzionalità di registrazione aggiuntive abilitate. MNIST è un database di numeri manoscritti in genere utilizzato per eseguire il training dei modelli di riconoscimento delle immagini. In questo tutorial, si utilizza lo script per addestrare un modello MNIST TensorBoard e i log per creare visualizzazioni.

Argomenti

Addestra un modello MNIST e visualizza l'allenamento con TensorBoard

Visualizza l'addestramento del modello MNIST con TensorBoard

1. Connettiti alla tua istanza Amazon Elastic Compute Cloud (Amazon EC2) di DLAMI con Conda.
2. Attiva l' TensorFlow ambiente Python 2.7 e vai alla directory che contiene la cartella con gli script di TensorBoard esempio:

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

3. Eseguire lo script per il training di un modello MNIST con la registrazione estesa attivata:

   ```
   $ python mnist_with_summaries.py
   ```

   Lo script scrive i log in `/tmp/tensorflow/mnist`.
4. Impostare `tensorboard` come posizione dei log:

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

TensorBoard avvia il server web di visualizzazione sulla porta 6006.

5. Per un facile accesso dal browser locale, è possibile impostare la porta 80 o un'altra porta per il server Web. Qualunque sia la porta che utilizzi, dovrai aprire questa porta nel gruppo di sicurezza EC2 per il tuo DLAMI. È anche possibile utilizzare l'inoltro alla porta. Per istruzioni su come modificare le impostazioni del gruppo di sicurezza e sull'inoltro alla porta, consulta Configurazione di un server di notebook Jupyter. Le impostazioni predefinite sono descritte nella fase successiva.

   > **Note**
   >
   > Se devi eseguire sia un server Jupyter che un TensorBoard server, usa una porta diversa per ciascuno.

6. Aprire la porta 6006 (o la porta assegnata al server Web di visualizzazione) sull'istanza EC2.

   a. Apri la tua istanza EC2 nella console Amazon EC2 all'indirizzo https://console.aws.amazon.com/ec2/.
   b. Nella console Amazon EC2, scegli Rete e sicurezza, quindi scegli Gruppi di sicurezza.
   c. Per Security Group (Gruppo di sicurezza), scegliere l'ultimo gruppo creato (vedere il timestamp nella descrizione).
   d. Scegliere la scheda Inbound (In entrata), quindi scegliere Edit (Modifica).
   e. Selezionare Add Rule (Aggiungi regola).
   f. Nella nuova riga, digitare quanto segue:

      Type (Tipo): **`TCP Rule`** Custom (Personalizzato)

      Protocol (Protocollo): **`TCP`**

      Port Range (Intervallo porte): **`6006`** (o la porta assegnata al server di visualizzazione)

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

7. Apri la pagina Web per TensorBoard le visualizzazioni utilizzando l'indirizzo IP o DNS pubblico dell'istanza EC2 che esegue il DLAMI con Conda e la porta che hai aperto per: TensorBoard

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

Ulteriori informazioni

Per saperne di più, consulta il sito Web. TensorBoard TensorBoard

Training distribuito

Scopri le opzioni offerte da DLAMI per l'addestramento con più GPU. Per migliorare le prestazioni, consulta Elastic Fabric Adapter Fai clic su una delle opzioni per informazioni su come utilizzarla.

Argomenti

- Chainer
- Keras con MXNet
- TensorFlow con Horovod

Chainer

> **Note**
>
> Non includiamo più gli ambienti Chainer Conda nella versione AWS Deep Learning AMI iniziale della versione v28. Le versioni precedenti di quelle AWS Deep Learning AMI che contengono questi ambienti continueranno a essere disponibili. Tuttavia, verranno forniti aggiornamenti a questi ambienti solo se sono disponibili correzioni di protezione pubblicate dalla comunità open source per questi framework.

Chainer è un framework flessibile basato su Python che consente di scrivere facilmente e intuitivamente delle architetture di rete neurali complesse. Chainer facilita l'utilizzo di molteplici istanze GPU per il training. Inoltre, registra automaticamente i risultati, rappresenta graficamente la perdita e la precisione e genera un output per la visualizzazione della rete neurale con un grafico di elaborazione. È incluso nell'AMI Deep Learning con Conda (DLAMI con Conda).

Gli argomenti seguenti descrivono come eseguire il training su più GPU, una singola GPU e una CPU, creare visualizzazioni e testare l'installazione Chainer.

Argomenti

- Training di un modello con Chainer
- Utilizzo di Chainer per eseguire il training su più GPU

Training di un modello con Chainer

Questo tutorial illustra come utilizzare esempi di script Chainer per eseguire il training di un modello con il set di dati MNIST. MNIST è un database di numeri manoscritti in genere utilizzato per eseguire il training dei modelli di riconoscimento delle immagini. Il tutorial descrive inoltre la differenza di velocità tra il training su una CPU e il training su una o più GPU.

Utilizzo di Chainer per eseguire il training su più GPU

Per eseguire il training su più GPU

1. Connect all'istanza che esegue l'AMI Deep Learning con Conda. Consulta la documentazione di Amazon EC2 the section called “Selezione dell'istanza” o consulta la documentazione di Amazon EC2 su come selezionare o connettersi a un'istanza. Per eseguire questo tutorial, è possibile utilizzare un'istanza con almeno due GPU.
2. Attivare l'ambiente Python 3 Chainer:

   ```
   $ source activate chainer_p36
   ```

3. Per ottenere i tutorial più recenti, clonare il repository Chainer e accedere alla cartella degli esempi:

   ```
   (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. Eseguire l'esempio nello script `train_mnist_data_parallel.py`. Per impostazione predefinita, lo script utilizza le GPU in esecuzione sulla tua istanza di Deep Learning AMI con Conda. Lo script può essere eseguito su un massimo di due GPU e ignorerà quindi ogni altra GPU. Inoltre, rileva automaticamente una o entrambe le GPU. Se si esegue un'istanza senza

GPU, passare alla sezione Utilizzo di Chainer per eseguire il training con CPU più avanti in questo tutorial.

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

> **Note**
>
> Questo esempio restituirà il seguente errore a causa dell'inclusione di una caratteristica beta non inclusa nella DLAMI.
> `chainerx ModuleNotFoundError: No module named 'chainerx'`

Mentre lo script Chainer esegue il training di un modello utilizzando il database MNIST, è possibile vedere i risultati per ogni epoca.

Successivamente, durante l'esecuzione dello script, viene visualizzato un esempio di output. L'esempio di output seguente è stato eseguito su un'istanza p3.8xlarge. L'output dello script mostra "GPU: 0, 1", a indicare che si stanno utilizzando le prime due delle quattro GPU disponibili. Gli script in genere utilizzano un indice di GPU a partire da zero, anziché da un totale.

```
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. Durante l'esecuzione del training, è utile esaminare l'utilizzo della GPU. È possibile verificare quali GPU sono attive e visualizzarne il relativo carico. NVIDIA fornisce uno strumento a tale scopo, che può essere eseguito con il comando `nvidia-smi`. Tuttavia, fornisce soltanto una snapshot dell'utilizzo, di conseguenza è più utile associarlo al comando Linux `watch`. Il comando seguente utilizzerà `watch` con `nvidia-smi` per aggiornare l'utilizzo corrente delle GPU ogni decimo di secondo. Aprire un'altra sessione del terminale sulla DLAMI ed eseguire il comando seguente:

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

   Si ottiene un output simile a quanto segue. Utilizzare `ctrl-c` per chiudere lo strumento oppure mantenerlo in esecuzione mentre si provano altri esempi nella prima sessione del terminale.

```
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 questo esempio, le GPU 0 e 1 sono attive mentre le GPU 2 e 3 non lo sono. È anche possibile visualizzare l'utilizzo della memoria per GPU.

6. Durante il completamento del training, annotare il tempo trascorso nella prima sessione del terminale. Nell'esempio, è pari a 38,1389 secondi.

Utilizzo di Chainer per eseguire il training su una singola GPU

Questo esempio illustra il modo in cui eseguire il training su una singola GPU. Ciò potrebbe essere necessario se disponi di una sola GPU o anche solo per determinare come il training su più GPU può evolvere con Chainer.

Per utilizzare Chainer per il training su una singola GPU

- Per questo esempio, si utilizza un altro script, `train_mnist.py`, e si indica allo stesso di utilizzare soltanto la GPU 0 con l'argomento `--gpu=0`. Per determinare come un'altra GPU viene attivata nella console `nvidia-smi`, è possibile indicare allo script di utilizzare la GPU 1 mediante `--gpu=1`.

```
(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 questo esempio, l'esecuzione su una singola GPU ha richiesto quasi il doppio del tempo. Il training di modelli o di set di dati di maggiori dimensioni genererà risultati differenti da questo esempio. Di conseguenza, è necessario sperimentare per valutare ulteriormente le prestazioni della GPU.

Utilizzo di Chainer per eseguire il training con CPU

Ora proverai il training in una modalità solo CPU. Esegui lo stesso script, `python train_mnist.py`, senza argomenti:

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

Nell'output, `GPU: -1` indica che non viene utilizzata alcuna GPU:

```
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 questo esempio, il training di MNIST è durato 323 secondi, un tempo 11 volte superiore al training con due GPU. Se hai mai dubitato della potenza delle GPU, questo esempio dimostra a che punto sono efficienti.

Rappresentazione grafica dei risultati

Chainer registra automaticamente i risultati, rappresenta graficamente perdita e precisione e produce l'output per il tracciamento del grafico di elaborazione.

Per generare il grafico di elaborazione

1. Al termine dell'esecuzione di un training, è possibile accedere alla directory `result` e visualizzare la precisione e la perdita dell'esecuzione mediante due immagini generate automaticamente. Accedervi adesso ed elencare il contenuto:

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

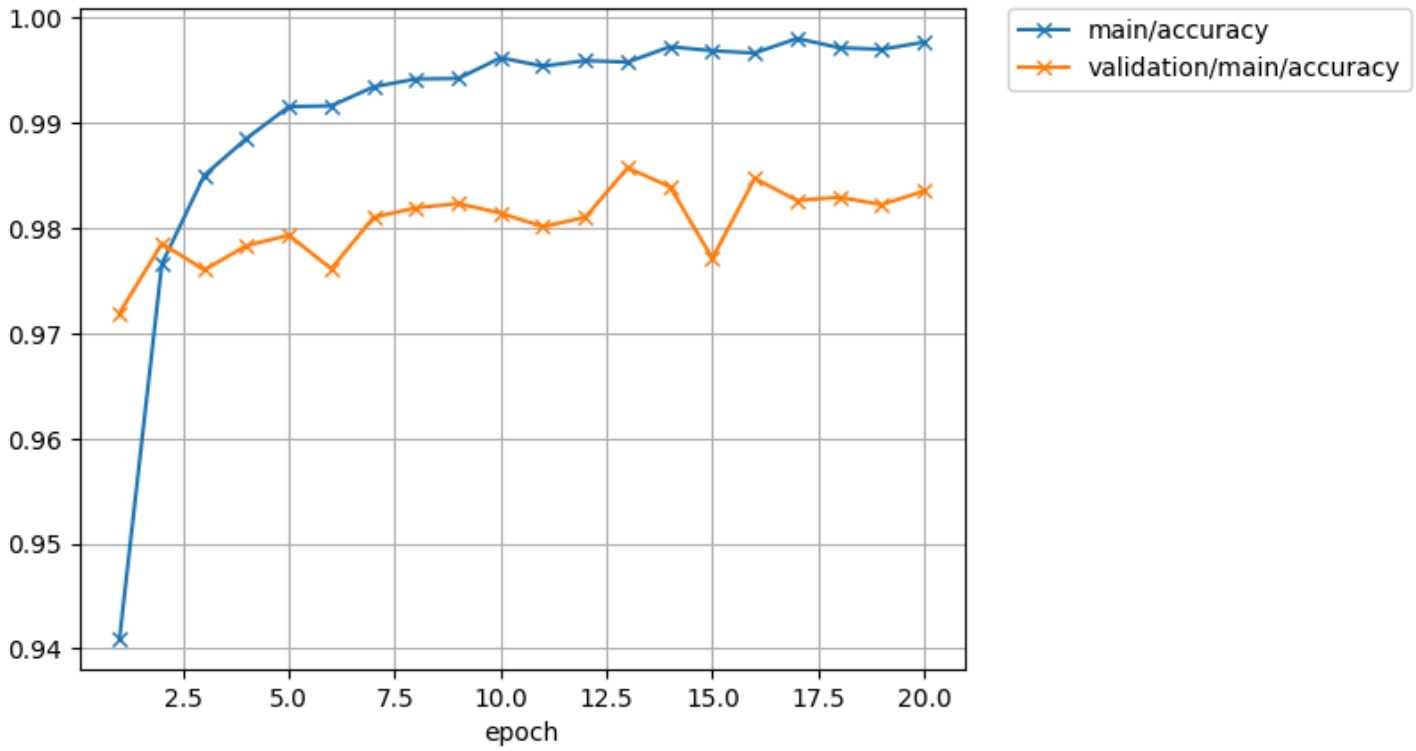
   La directory `result` contiene due file in formato png: `accuracy.png` e `loss.png`.

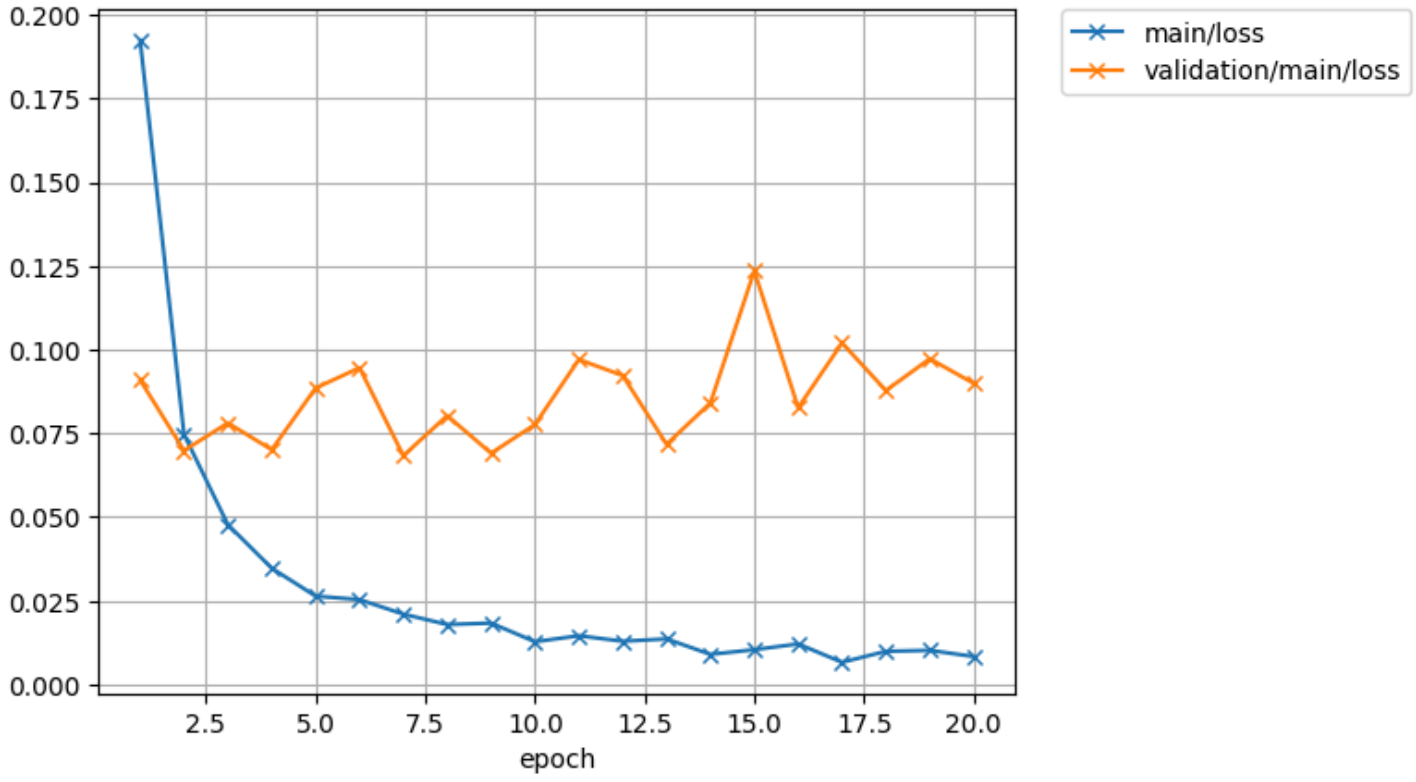
2. Per visualizzare i grafici, utilizzare il comando `scp` per copiarli sul computer locale.

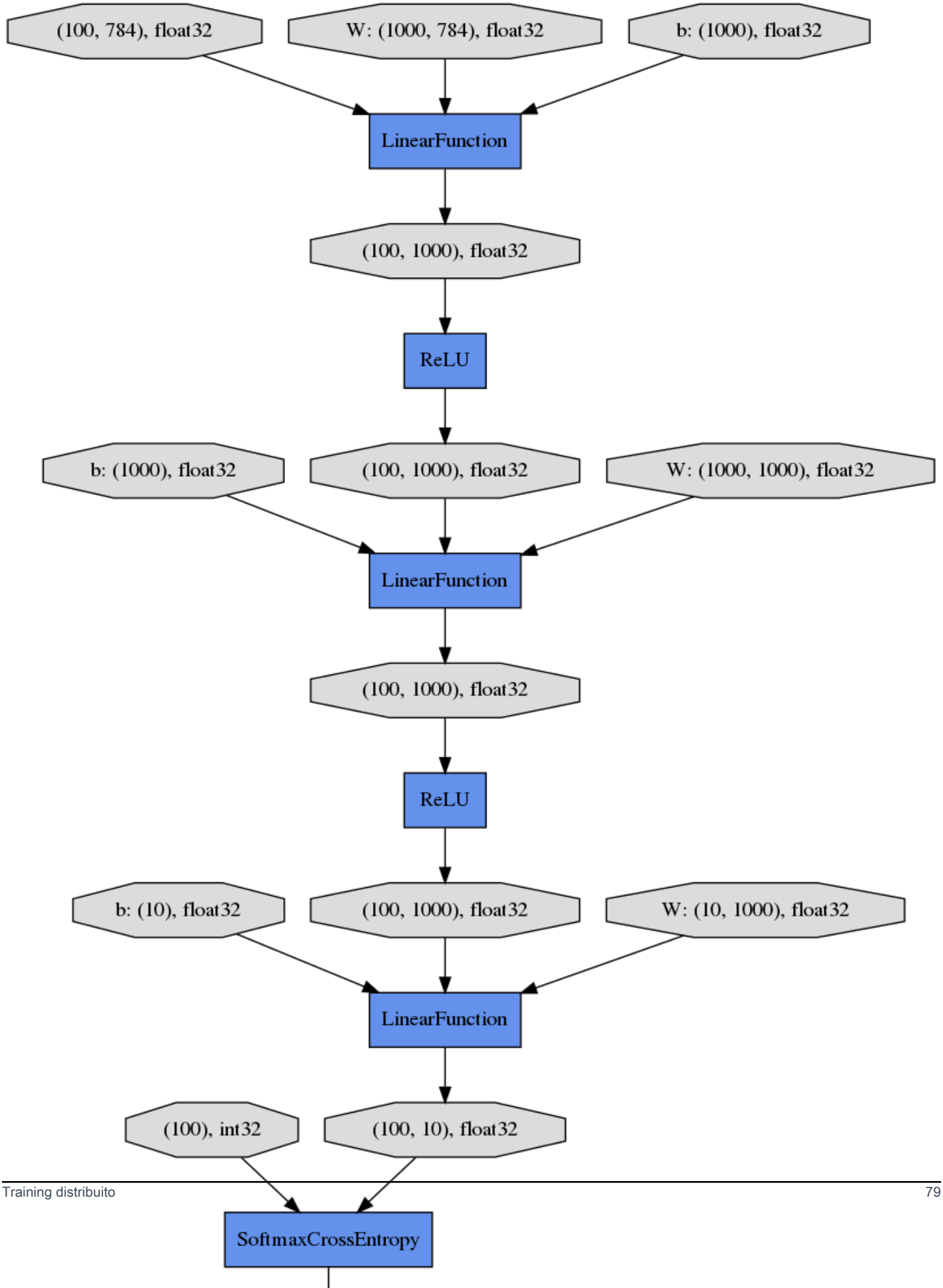
   In un terminale macOS, l'esecuzione del comando `scp` scarica tutti e tre i file nella cartella `Downloads`. Sostituire i segnaposto per la posizione del file delle chiavi e dell'indirizzo del server con le informazioni. Per altri sistemi operativi, utilizzare il formato del comando `scp` appropriato. Nota, per un'AMI Amazon Linux, il nome utente è 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
   ```

Le immagini seguenti sono rispettivamente esempi di grafici di precisione, perdita ed elaborazione.


main/accuracy
validation/main/accuracy
1.00
0.99
0.98
0.97
0.96
0.95
0.94
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
epoch


main/loss
validation/main/loss
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


(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

Test di Chainer

Per testare Chainer e verificare il supporto GPU con uno script di prova preinstallato, esegui il comando seguente:

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

Questo comando scarica il codice sorgente di Chainer ed esegue l'esempio MNIST con più GPU di Chainer.

Ulteriori informazioni

Per ulteriori informazioni su Chainer, consulta il sito Web della documentazione di Chainer. La cartella degli esempi `Chainer` contiene ulteriori esempi. Provali per verificarne le prestazioni.

Keras con MXNet

Questo tutorial mostra come attivare e utilizzare Keras 2 con il backend MXNet su un'AMI Deep Learning con Conda.

Attiva Keras con il backend MXNet e testalo su DLAMI con Conda

1. Per attivare Keras con il backend MXNet, apri un'istanza Amazon Elastic Compute Cloud (Amazon EC2) di DLAMI con Conda.
   - Per Python 3, eseguire questo comando:

     ```
     $ source activate mxnet_p36
     ```
   - Per Python 2, eseguire questo comando:

     ```
     $ source activate mxnet_p27
     ```
2. Avviare il terminale iPython:

   ```
   (mxnet_p36)$ ipython
   ```
3. Testare l'importazione di Keras con MXNet per verificare che funziona correttamente:

```
import keras as k
```

Sullo schermo dovrebbe essere visualizzato quanto segue (possibilmente dopo alcuni messaggi di avviso).

```
Using MXNet backend
```

> **Note**
>
> Se ricevi un errore o se il TensorFlow backend è ancora in uso, devi aggiornare manualmente la configurazione di Keras. Modificare il file `~/.keras/keras.json` e impostare il parametro di back-end su `mxnet`.

Tutorial per il training di molteplici GPU con Keras-MXNet

Training di una rete neurale convoluzionale (CNN)

1. Apri un terminale e inserisci SSH nel tuo DLAMI.
2. Accedi alla cartella `~/examples/keras-mxnet/`.
3. Eseguire `nvidia-smi` nella finestra del terminale per determinare il numero di GPU disponibili sulla DLAMI. Nella fase successiva, si eseguirà lo script così com'è se si dispone di quattro GPU.
4. (Facoltativo) Eseguire il comando seguente per aprire lo script per la modifica.

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

5. (Facoltativo) Lo script include la seguente riga che definisce il numero di GPU. Aggiornare se necessario.

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

6. Eseguire il training.

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

> **Note**
>
> L'esecuzione Keras-MXNet è fino a due volte più rapida con il set `channels_first` image_data_format. Per passare a `channels_first`, modificare il file di configurazione di Keras (`~/.keras/keras.json`) e impostare quanto segue: `"image_data_format": "channels_first"`.
> Per ulteriori tecniche di ottimizzazione delle prestazioni, consulta Keras-MXNet performance tuning guide.

Ulteriori informazioni

- Puoi trovare esempi di Keras con un backend MXNet nella directory Deep Learning AMI with Conda. `~/examples/keras-mxnet`
- Per ulteriori tutorial ed esempi, consultate il progetto GitHub Keras-MXNet.

TensorFlow con Horovod

Questo tutorial mostra come usare TensorFlow Horovod su un'AMI Deep Learning con Conda. Horovod è preinstallato negli ambienti Conda per. TensorFlow L'ambiente Python 3 è consigliato. Le istruzioni qui di seguito presuppongono che si disponga di un'istanza DLAMI funzionante con una o più GPU. Per ulteriori informazioni, consulta Come iniziare con il DLAMI.

> **Note**
>
> Solo i tipi di istanza P3.*, P2.* e G3.* sono supportati.

> **Note**
>
> Ci sono due posizioni in cui mpirun (tramite OpenMPI) è disponibile. È disponibile in `/usr/bin` e in `/home/ubuntu/anaconda3/envs/<env>/bin`. env è un ambiente corrispondente al framework, come Tensorflow e Apache MXNet. Le versioni OpenMPI più recenti sono disponibili negli ambienti conda. Consigliamo di utilizzare il percorso assoluto del binario mpirun o il flag --prefix per eseguire i carichi di lavoro mpi. Ad esempio, con l'ambiente Tensorflow python36, utilizza:

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

or

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

Attiva e prova con Horovod TensorFlow

1. Verifica che l'istanza abbia GPU attive. NVIDIA fornisce uno strumento apposito:

   ```
   $ nvidia-smi
   ```

2. Attiva l'ambiente Python 3: TensorFlow

   ```
   $ source activate tensorflow_p36
   ```

3. Avviare il terminale iPython:

   ```
   (tensorflow_p36)$ ipython
   ```

4. Prova l'importazione TensorFlow con Horovod per verificare che funzioni correttamente:

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

   Sullo schermo può essere visualizzato quanto segue (possibilmente dopo alcuni messaggi di avviso).

   ```
   --------------------------------------------------------------------------
   [[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.
   ```

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

Configurazione del file Hosts Horovod

È possibile utilizzare Horovod per il training a singolo nodo e più GPU o a più nodi e più GPU. Se si prevede di utilizzare più nodi per la formazione distribuita, è necessario aggiungere ogni indirizzo IP privato DLAMI a un file hosts. Il DLAMI a cui sei attualmente connesso viene definito leader. Le altre istanze DLAMI che fanno parte del cluster vengono chiamate membri.

Prima di iniziare questa sezione, avviate uno o più DLAMI e aspettate che siano nello stato Ready. Gli script di esempio prevedono un file hosts, quindi anche se prevedi di utilizzare un solo DLAMI, crea un file hosts con una sola voce. Se modifichi il file hosts dopo l'inizio del training, è necessario riavviare il training per rendere effettivi gli hosts aggiunti o rimossi.

Per configurare Horovod per il training

1. Modifica le directory in cui si trovano gli script di training.

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

2. Utilizza vim per modificare un file nella directory home del leader.

   ```
   vim hosts
   ```

3. Seleziona uno dei membri nella console Amazon Elastic Compute Cloud e viene visualizzato il riquadro descrittivo della console. Trova il campo Private IPs (IP privati) e copia e incolla l'IP in un file di testo. Copia l'IP privato di ogni membro su una nuova riga. Quindi, accanto a ciascun IP, aggiungi uno spazio e il testo `slots=8` come indicato in basso. Questo rappresenta la quantità di GPU che ha ogni istanza. Le istanze p3.16xlarge dispongono di 8 GPU, perciò se scegli un altro tipo di istanza devi fornire il numero effettivo di GPU per ogni istanza. Per il leader puoi utilizzare `localhost`. Con un cluster di 4 nodi, dovrebbe essere simile a quanto segue:

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

   Salva il file e torna al terminale del leader.

4. Aggiungi la chiave SSH utilizzata dalle istanze membri all'agente ssh-agent.

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

5. Ora il leader sa come raggiungere ogni membro. Tutto questo si verifica sulle interfacce di rete privata. Successivamente, utilizza una breve funzione bash per inviare i comandi a ciascun membro.

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

6. Chiedi agli altri membri di non fare «StrickHostKeyChecking» poiché ciò potrebbe impedire all'addestramento di rispondere.

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

Training con dati sintetici

Il DLAMI viene fornito con uno script di esempio per addestrare un modello con dati sintetici. Questo verifica se il leader è in grado di comunicare con i membri del cluster. È richiesto un file hosts. Per le istruzioni, fai riferimento a Configurazione del file Hosts Horovod.

Per testare il training Horovod con dati di esempio

1. L'impostazione predefinita di `~/examples/horovod/tensorflow/train_synthetic.sh` è di 8 GPU, ma puoi indicare il numero di GPU da eseguire. L'esempio seguente esegue lo script, passando 4 come parametro per 4 GPU.

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

   Dopo alcuni messaggi di avviso, verrà visualizzato l'output seguente che verifica se Horovod sta utilizzando 4 GPU.

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

   Quindi, dopo altri avvisi, verrà visualizzato l'inizio di una tabella e alcuni punti dati. Se vuoi controllare 1.000 batch, esci dal training.

```
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 usa tutte le GPU locali prima di tentare di utilizzare le GPU dei membri del cluster. Quindi, per accertarti che il training distribuito nel cluster funzioni correttamente, prova il numero totale di GPU che vuoi utilizzare. Se, ad esempio, hai 4 membri che sono tipi di istanza p3.16xlarge, avrai 32 GPU in tutto il cluster. In questo caso vuoi provare con tutte le 32 GPU.

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

   L'output è simile a quello del test precedente. La dimensione di Horovod è 32 e la velocità di circa 4 volte. Una volta completata questa prova, hai testato il leader e la capacità di comunicare con i membri. In caso di problemi, consulta la sezione Risoluzione dei problemi.

Preparare il set di dati ImageNet

In questa sezione, scarichi il ImageNet set di dati, quindi generi un set di dati in formato TFRecord dal set di dati non elaborato. Nel DLAMI viene fornito un set di script di preelaborazione per il set di dati che è possibile utilizzare per uno o più ImageNet set di dati ImageNet o come modello per un altro set di dati. Vengono inoltre forniti i principali script di formazione configurati per. ImageNet La sezione seguente presuppone che tu abbia lanciato un DLAMI con un'istanza EC2 con 8 GPU. Consigliamo di utilizzare il tipo di istanza p3.16xlarge.

Nella `~/examples/horovod/tensorflow/utils` directory del tuo DLAMI trovi i seguenti script:

- `utils/preprocess_imagenet.py`- Utilizzatelo per convertire il ImageNet set di dati non elaborato nel formato. `TFRecord`
- `utils/tensorflow_image_resizer.py`- Utilizzatelo per ridimensionare il `TFRecord` set di dati come consigliato per l'addestramento. ImageNet

Preparare il set di dati ImageNet

1. Visitare image-net.org, creare un account, acquisire una chiave di accesso e scaricare il set di dati. image-net.org ospita il set di dati non elaborato. Per scaricarlo, è necessario disporre di un ImageNet account e di una chiave di accesso. L'account è gratuito e per ottenere la chiave di accesso gratuita è necessario accettare la ImageNet licenza.
2. Utilizza lo script di preelaborazione delle immagini per generare un set di dati in formato TFRecord dal set di dati grezzo. ImageNet A partire dalla directory `~/examples/horovod/tensorflow/utils`:

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

3. Utilizzare lo script di ridimensionamento delle immagini. Se si ridimensionano le immagini, la formazione viene eseguita più rapidamente e si allinea meglio al ResNet documento di riferimento. A partire dalla directory `~/examples/horovod/utils/preprocess`:

```
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
```

Addestramento ResNet di un ImageNet modello -50 su un singolo DLAMI

> **Note**
>
> - Lo script in questo tutorial presuppone che i dati di training preelaborati si trovino nella cartella `~/data/tf-imagenet/`. Per le istruzioni, fai riferimento a Preparare il set di dati ImageNet .

- È richiesto un file hosts. Per le istruzioni, fai riferimento a Configurazione del file Hosts Horovod.

Usa Horovod per addestrare un ResNet 50 CNN sul set di dati ImageNet

1. Accedi alla cartella `~/examples/horovod/tensorflow`.

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

2. Verificare la configurazione e impostare il numero di GPU da utilizzare nel training. In primo luogo, esamina il file `hosts` che si trova nella stessa cartella degli script. Questo file deve essere aggiornato se si utilizza un'istanza con meno di 8 GPU. Il valore predefinito è `localhost slots=8`. Sostituire il valore 8 con il numero di GPU che si desidera utilizzare.
3. Viene fornito uno script della shell che utilizza il numero di GPU che prevedi di utilizzare come unico parametro. Esegui questo script per iniziare il training. L'esempio seguente utilizza 4 per quattro GPU.

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

4. L'esecuzione dura alcune ore. Questo script utilizza `mpirun` per distribuire il training alle GPU.

Addestra un ImageNet modello ResNet -50 su un cluster di DLAMI

Note

- Lo script in questo tutorial presuppone che i dati di training preelaborati si trovino nella cartella `~/data/tf-imagenet/`. Per le istruzioni, fai riferimento a Preparare il set di dati ImageNet .
- È richiesto un file hosts. Per le istruzioni, fai riferimento a Configurazione del file Hosts Horovod.

Questo esempio illustra l'addestramento di un modello ResNet -50 su un set di dati preparato su più nodi in un cluster di DLAMI.

- Per prestazioni più veloci, ti consigliamo di avere il set di dati in locale su ogni membro del cluster.

  Utilizzare questa funzione `copyclust` per copiare i dati per altri membri.

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

In alternativa, se hai i file in un bucket S3, utilizza la funzione `runclust` per scaricare i file direttamente su ogni membro.

```
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/\""
```

L'utilizzo di strumenti che consentono di gestire più nodi contemporaneamente consente di risparmiare tempo. Puoi aspettare ogni fase e gestire ogni istanza separatamente oppure utilizzare strumenti come tmux o screen che consentono di scollegare e riprendere le sessioni.

Una volta completata la copia, sei pronto per iniziare il training. Esegui lo script, passando 32 come parametro per le 32 GPU utilizzate per questa sessione. Utilizza tmux o uno strumento simile se sei preoccupato di scollegare e terminare la sessione, che terminerebbe l'esecuzione del training.

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

L'output seguente è quello che vedi quando esegui il training su ImageNet 32 GPU. Trentadue GPU impiegano 90-110 minuti.

```
    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
```

Una volta completata l'esecuzione del training, lo script prosegue con una valutazione, che viene eseguita sul leader perché viene eseguita rapidamente senza dover distribuire il processo agli altri membri. Di seguito è riportato l'output dell'esecuzione di valutazione.

```
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
```

Di seguito è riportato un esempio di output quando questo script viene eseguito con 256 GPU e il runtime era tra 14 e 15 minuti.

```
  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
```

Risoluzione dei problemi

Il comando seguente può aiutare a superare gli errori che si verificano quando si eseguono delle prove con Horovod.

- Se il training si interrompe per qualsiasi motivo, mpirun potrebbe non eliminare correttamente tutti i processi Python su ciascuna macchina. In tal caso, prima di iniziare il lavoro successivo, interrompi i processi Python su tutte le macchine nel modo seguente:

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

- Se il processo termina all'improvviso senza errori, prova a eliminare la cartella di log.

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

- Se si verificano altri errori imprevisti, controlla lo spazio su disco. Se è esaurito, prova a rimuovere la cartella di log che è piena di checkpoint e dati. Puoi anche aumentare le dimensioni dei volumi per ciascun membro.

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

- In ultima istanza puoi anche provare a riavviare.

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

Potresti ricevere il seguente codice di errore se provi a utilizzarlo TensorFlow con Horovod su un tipo di istanza non supportato:

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

Ulteriori informazioni

Per utilità ed esempi, consultate la `~/examples/horovod` cartella nella home directory del DLAMI.

Per ulteriori tutorial ed esempi, consultate il progetto Horovod. GitHub

Elastic Fabric Adapter

Un Elastic Fabric Adapter (EFA) è un dispositivo di rete che è possibile collegare all'istanza DLAMI per accelerare le applicazioni HPC (High Performance Computing). EFA consente di ottenere le prestazioni applicative di un cluster HPC locale, con la scalabilità, la flessibilità e l'elasticità fornite dal cloud. AWS

I seguenti argomenti mostrano come iniziare a utilizzare EFA con DLAMI.

> **Note**
>
> Scegli il tuo DLAMI da questo elenco DLAMI di GPU di base

Argomenti

- Avvio di un'istanza con EFA AWS Deep Learning AMI
- Utilizzo di EFA su DLAMI

Avvio di un'istanza con EFA AWS Deep Learning AMI

La versione più recente di Base DLAMI è pronta per l'uso con EFA e viene fornita con i driver necessari, i moduli del kernel, libfabric, openmpi e il plug-in NCCL OFI per le istanze GPU.

È possibile trovare le versioni CUDA supportate di un DLAMI di base nelle note di rilascio.

Nota:

- Quando si esegue un'applicazione NCCL utilizzando `mpirun` EFA, è necessario specificare il percorso completo dell'installazione supportata da EFA come:

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

- Per consentire all'applicazione di utilizzare EFA, aggiungere `FI_PROVIDER="efa"` al comando `mpirun` come mostrato in Utilizzo di EFA su DLAMI.

Argomenti

- Preparare un gruppo di sicurezza abilitato all'EFA
- Avvio dell'istanza
- Verifica l'allegato EFA

Preparare un gruppo di sicurezza abilitato all'EFA

L'EFA richiede un gruppo di sicurezza che consenta tutto il traffico in entrata e in uscita da e verso il gruppo di sicurezza stesso. Per ulteriori informazioni, consulta la documentazione EFA.

1. Apri la console Amazon EC2 all'indirizzo https://console.aws.amazon.com/ec2/.
2. Nel riquadro di navigazione, scegliere Security Groups (Gruppi di sicurezza) e quindi Create Security Group (Crea gruppo di sicurezza).
3. Nella finestra Create Security Group (Crea gruppo di sicurezza) effettuare le operazioni seguenti:
   - In Nome gruppo di sicurezza, immettere un nome descrittivo per il gruppo di sicurezza, ad esempio `EFA-enabled security group`.
   - (Facoltativo) In Description (Descrizione), inserire una breve descrizione del gruppo di sicurezza.
   - In VPC, selezionare il VPC in cui avviare le istanze abilitate per EFA.
   - Scegliere Create (Crea).
4. Selezionare il gruppo di sicurezza creato e, nella scheda Description (Descrizione), copiare il valore Group ID (ID gruppo).
5. Nelle schede In entrata e In uscita, effettuate le seguenti operazioni:
   - Seleziona Edit (Modifica).
   - In Type (Tipo), selezionare All traffic (Tutto il traffico).
   - In Source (Origine), scegliere Custom (Personalizzata).
   - Incollare nel campo l'ID del gruppo di sicurezza copiato in precedenza.
   - Selezionare Salva.
6. Abilitare il traffico in entrata facente riferimento a Autorizzazione del traffico in entrata per le istanze Linux. Se salti questo passaggio, non sarai in grado di comunicare con l'istanza DLAMI.

Avvio dell'istanza

EFA on the AWS Deep Learning AMI è attualmente supportato con i seguenti tipi di istanze e sistemi operativi:

- P3DN.24xlarge: Amazon Linux 2, Ubuntu 20.04
- P4D.24xlarge: Amazon Linux 2, Ubuntu 20.04
- P5.48xLarge: Amazon Linux 2, Ubuntu 20.04

La sezione seguente mostra come avviare un'istanza DLAMI abilitata per EFA. Per ulteriori informazioni sul lancio di un'istanza abilitata per EFA, consulta Launch Enabled Instances into a Cluster Placement Group.

1. Aprire la console Amazon EC2 all'indirizzo https://console.aws.amazon.com/ec2/.
2. Scegliere Launch Instance (Avvia istanza).
3. Nella pagina Scegli un AMI, seleziona un DLAMI supportato che si trova nella pagina delle note di rilascio di DLAMI
4. Nella pagina Scegli un tipo di istanza, seleziona uno dei seguenti tipi di istanza supportati, quindi scegli Avanti: Configura i dettagli dell'istanza. Fai riferimento a questo link per l'elenco delle istanze supportate: Guida introduttiva a EFA e MPI
5. Nella pagina Configure Instance Details (Configura i dettagli dell'istanza), procedere come segue:
   - In Number of instances (Numero di istanze), immettere il numero di istanze abilitate per EFA che si desidera avviare.
   - In Network (Rete) e Subnet (Sottorete), selezionare il VPC e la sottorete in cui avviare le istanze.
   - [Facoltativo] Per il gruppo di collocamento, selezionate Aggiungi istanza al gruppo di collocamento. Per ottenere prestazioni ottimali, avviare le istanze all'interno di un gruppo di collocazione.
   - [Facoltativo] Per il nome del gruppo di collocamento, selezionate Aggiungi a un nuovo gruppo di collocamento, inserite un nome descrittivo per il gruppo di collocamento, quindi per la strategia del gruppo di collocamento, selezionate cluster.
   - Assicurati di abilitare «Elastic Fabric Adapter» in questa pagina. Se questa opzione è disabilitata, modificare la subnet in una che supporta il tipo di istanza selezionato.
   - Nella sezione Network Interfaces (Interfacce di rete), per il dispositivo eth0 scegliere New network interface (Nuova interfaccia di rete). Facoltativamente, è possibile specificare un indirizzo IPv4 primario e uno o più indirizzi IPv4 secondari. Se l'istanza viene lanciata in una sottorete alla quale è associato un blocco CIDR IPv6, è possibile specificare un indirizzo IPv6 primario e uno o più indirizzi IPv6 secondari.
   - Scegliere Next: Add Storage (Successivo: aggiungi storage).
6. Nella pagina Add archiviazione (Aggiungi archiviazione), specificare i volumi da collegare all'istanza, oltre a quelli specificati dall'AMI (ad esempio il volume dispositivo root), quindi selezionare Next: Add Tags (Successivo: aggiungi tag).

7. Nella pagina Add Tags (Aggiungi tag) specificare i tag per l'istanza, ad esempio un nome intuitivo, quindi selezionare Next: Configure Security Group (Successivo: configurazione del gruppo di sicurezza).
8. Nella pagina Configura gruppo di sicurezza, per Assegna un gruppo di sicurezza, seleziona Seleziona un gruppo di sicurezza esistente, quindi seleziona il gruppo di sicurezza creato in precedenza.
9. Scegliere Review and Launch (Analizza e avvia).
10. Nella pagina Review Instance Launch (Verifica avvio istanza) controllare le impostazioni e selezionare Launch (Avvia) per scegliere una coppia di chiavi e avviare l'istanza.

Verifica l'allegato EFA

Dalla console

Dopo aver avviato l'istanza, controlla i dettagli dell'istanza nella AWS console. A tale scopo, seleziona l'istanza nella console EC2 ed esamina la scheda Descrizione nel riquadro inferiore della pagina. Trova il parametro 'Interfacce di rete: eth0' e fai clic su eth0 che apre un pop-up. Assicurati che «Elastic Fabric Adapter» sia abilitato.

Se EFA non è abilitato, puoi risolvere il problema in uno dei seguenti modi:

- Terminando l'istanza EC2 e avviandone una nuova con le stesse fasi. Assicurati che l'EFA sia collegato.
- Collega EFA a un'istanza esistente.

  1. Nella console EC2, passa a Network Interfaces (Interfacce di rete).
  2. Fai clic su Create a Network Interface (Crea un'interfaccia di rete).
  3. Seleziona la stessa subnet in cui si trova l'istanza.
  4. Assicurati di abilitare «Elastic Fabric Adapter» e fai clic su Crea.
  5. Torna alla scheda EC2 Instances (Istanze EC2) e seleziona l'istanza.
  6. Vai a Azioni: Stato dell'istanza e interrompi l'istanza prima di collegare EFA.
  7. Da Actions (Operazioni), seleziona Networking: Attach Network Interface (Rete: Collega interfaccia di rete).
  8. Seleziona l'interfaccia appena creata e clicca su attach (collega).
  9. Riavviare l'istanza.

Dall'istanza

Il seguente script di test è già presente sul DLAMI. Eseguilo per assicurarti che i moduli del kernel siano caricati correttamente.

```
$ fi_info -p efa
```

L'aspetto dell'output sarà simile al seguente.

```
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
```

Verifica della configurazione del gruppo di sicurezza

Il seguente script di test è già presente sul DLAMI. Eseguilo per assicurarti che il gruppo di sicurezza creato sia configurato correttamente.

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

L'aspetto dell'output sarà simile al seguente.

```
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
```

Se smette di rispondere o non viene completato, assicuratevi che il gruppo di sicurezza disponga delle regole in entrata/uscita corrette.

Utilizzo di EFA su DLAMI

La sezione seguente descrive come utilizzare EFA per eseguire applicazioni multinodo su. AWS Deep Learning AMI

Esecuzione di applicazioni multinodo con EFA

Per eseguire un'applicazione su un cluster di nodi è richiesta la seguente configurazione

Argomenti

- Abilitazione di SSH senza password
- Creazione di file hosts
- Test NCCL

Abilitazione di SSH senza password

Seleziona un nodo nel cluster come il nodo principale. I nodi rimanenti sono indicati come nodi membro.

1. Nel nodo principale, genera la coppia di chiavi RSA.

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

2. Modifica le autorizzazioni della chiave privata sul nodo principale.

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

3. Copia la chiave `~/.ssh/id_rsa.pub` pubblica e aggiungila a uno `~/.ssh/authorized_keys` dei nodi membri del cluster.

4. Puoi ora accedere direttamente ai nodi membro dal nodo principale utilizzando l'IP privato.

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

5. Disabilita strictHostKey Checking e abilita l'inoltro degli agenti sul nodo leader aggiungendo quanto segue al file ~/.ssh/config sul nodo leader:

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

6. Nelle istanze Amazon Linux 2, esegui il seguente comando sul nodo leader per fornire le autorizzazioni corrette al file di configurazione:

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

Creazione di file hosts

Nel nodo principale, creare un file hosts per identificare i nodi nel cluster. Il file hosts deve contenere una voce per ogni nodo del cluster. Crea un file ~/hosts e aggiungi ogni nodo utilizzando l'IP privato come riportato di seguito:

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

Test NCCL

> **Note**
>
> Questi test sono stati eseguiti utilizzando la versione EFA 1.30.0 e il plugin OFI NCCL 1.7.4.

Di seguito sono elencati un sottoinsieme di test NCCL forniti da Nvidia per testare funzionalità e prestazioni su più nodi di elaborazione

Istanze supportate: P3dn, P4, P5

Test di funzionalità

Test multinodo NCCL Message Transfer

Il nccl_message_transfer è un semplice test per garantire che il plugin NCCL OFI funzioni come previsto. Il test convalida la funzionalità delle API di avvio della connessione e di trasferimento dati di NCCL. Assicurati di utilizzare il percorso completo di mpirun come mostrato nell'esempio durante l'esecuzione di applicazioni NCCL con EFA. Modifica i parametri np e N in base al numero di istanze e GPU nel cluster. Per ulteriori informazioni, consultate la documentazione OFI NCCL.AWS

Il seguente test nccl_message_transfer riguarda una versione generica di CUDA xx.x. Puoi eseguire i comandi per qualsiasi versione CUDA disponibile nella tua istanza Amazon EC2 sostituendo la versione CUDA nello script.

```
$/opt/amazon/openmpi/bin/mpirun -n 2 -N 1 --hostfile hosts \
-x LD_LIBRARY_PATH=/usr/local/cuda-xx.x/efa/lib:/usr/local/cuda-xx.x/lib:/usr/
local/cuda-xx.x/lib64:/usr/local/cuda-xx.x:$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
```

L'aspetto dell'output sarà simile al seguente. Puoi controllare l'output per vedere che EFA viene utilizzato come provider OFI.

```
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
```

Test delle prestazioni

Test delle prestazioni NCCL multinodo su P4D.24XLarge

Per verificare le prestazioni NCCL con EFA, esegui il test NCCL Performance standard disponibile sul Repo ufficiale di NCCL-Tests. Il DLAMI viene fornito con questo test già creato per CUDA XX.X. Allo stesso modo è possibile eseguire il proprio script con EFA.

Quando costruisci il tuo script, fai riferimento alla seguente guida:

- Utilizzate il percorso completo di mpirun come mostrato nell'esempio durante l'esecuzione di applicazioni NCCL con EFA.
- Modifica i parametri np e N in base al numero di istanze e GPU nel cluster.
- Aggiungi il flag NCCL_DEBUG=INFO e assicurati che i log indichino l'utilizzo di EFA come «Il provider selezionato è EFA».
- Imposta la posizione del registro di formazione da analizzare per la convalida

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

Utilizza il comando `watch nvidia-smi` su uno qualsiasi dei nodi membri per monitorare l'utilizzo di GPU. I `watch nvidia-smi` comandi seguenti si riferiscono a una versione generica di CUDA xx.x e dipendono dal sistema operativo dell'istanza. Puoi eseguire i comandi per qualsiasi versione CUDA disponibile nella tua istanza Amazon EC2 sostituendo la versione CUDA nello script.

- 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}
```

L'aspetto dell'output deve essere simile al seguente:

```
# 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
```

Test di convalida

Per verificare che i test EFA abbiano restituito un risultato valido, utilizza i seguenti test per confermare:

- Ottieni il tipo di istanza utilizzando EC2 Instance Metadata:

```
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)
```

- Eseguire Test delle prestazioni
- Imposta i seguenti parametri

```
CUDA_VERSION
CUDA_RUNTIME_VERSION
NCCL_VERSION
```

- Convalida i risultati come mostrato:

```
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
```

- Per accedere ai dati del benchmark, possiamo analizzare l'ultima riga della tabella in uscita dal test multinodo all_reduce:

```
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}"
```

Monitoraggio e ottimizzazione GPU

La sezione seguente descrive le opzioni di ottimizzazione e monitoraggio della GPU. Questa sezione è organizzata come un flusso di lavoro tipico in cui il monitoraggio supervisiona la pre-elaborazione e il training.

- Monitoraggio
  - Monitora le GPU con CloudWatch
- Ottimizzazione
  - Pre-elaborazione
  - Addestramento

Monitoraggio

Il tuo DLAMI è preinstallato con diversi strumenti di monitoraggio della GPU. Questa guida fa anche riferimento a strumenti disponibili per scaricare e installare.

- Monitora le GPU con CloudWatch- un'utilità preinstallata che riporta le statistiche sull'utilizzo della GPU ad Amazon. CloudWatch
- nvidia-smi CLI - un'utilità per il monitoraggio di calcolo e utilizzo di memoria della GPU. È preinstallato sul tuo AWS Deep Learning AMI (DLAMI).
- NVML libreria C: un'API basata sul C per accedere direttamente alle funzioni di monitoraggio e gestione della GPU. Viene utilizzata dall'interfaccia a riga di comando nvidia-smi dietro le quinte ed è preinstallata sulla DLAMI. Dispone anche di associazioni Python e Perl per facilitare lo sviluppo in tali lingue. L'utilità gpumon.py preinstallata sul DLAMI utilizza il pacchetto pynvml di. nvidia-ml-py
- NVIDIA DCGM: uno strumento di gestione cluster. Per informazioni su come installare e configurare questo strumento, visita la pagina per gli sviluppatori.

> **Tip**
>
> Dai un'occhiata al blog degli sviluppatori di NVIDIA per le ultime informazioni sull'utilizzo degli strumenti CUDA per installare il tuo DLAMI:
>
> - Monitoraggio dell' TensorCore utilizzo tramite Nsight IDE e nvprof.

Monitora le GPU con CloudWatch

Quando si utilizza la DLAMI con una GPU, è possibile che si stiano cercando modi per tenere traccia del suo utilizzo durante il training o l'inferenza. Questo può essere utile per ottimizzare la data pipeline e regolare la rete di deep learning.

Esistono due modi per configurare le metriche della GPU con: CloudWatch

- Configura le metriche con l' AWS CloudWatch agente (consigliato)
- Configura le metriche con lo script preinstallato `gpumon.py`

Configura le metriche con l' AWS CloudWatch agente (consigliato)

Integra il tuo DLAMI con l' CloudWatch agente unificato per configurare i parametri della GPU e monitorare l'utilizzo dei coprocessi GPU nelle istanze accelerate di Amazon EC2.

Esistono quattro modi per configurare le metriche della GPU con DLAMI:

- Configura metriche minime per la GPU

- Configura le metriche parziali della GPU
- Configura tutte le metriche GPU disponibili
- Configura metriche GPU personalizzate

Per informazioni sugli aggiornamenti e le patch di sicurezza, consulta Applicazione di patch di sicurezza per l'agente AWS CloudWatch

Prerequisiti

Per iniziare, devi configurare le autorizzazioni IAM dell'istanza Amazon EC2 che consentano all'istanza di inviare parametri a. CloudWatch Per i passaggi dettagliati, consulta Creare ruoli e utenti IAM da utilizzare con l'agente. CloudWatch

Configura metriche minime per la GPU

Configura metriche minime per la GPU utilizzando il servizio. `dlami-cloudwatch-agent@minimal` `systemd` Questo servizio configura le seguenti metriche:

- `utilization_gpu`
- `utilization_memory`

Puoi trovare il `systemd` servizio per le metriche minime preconfigurate della GPU nella seguente posizione:

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

Abilita e avvia il `systemd` servizio con i seguenti comandi:

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

Configura le metriche parziali della GPU

Configura le metriche parziali della GPU utilizzando il servizio. `dlami-cloudwatch-agent@partial` `systemd` Questo servizio configura le seguenti metriche:

- `utilization_gpu`
- `utilization_memory`

- memory_total
- memory_used
- memory_free

Puoi trovare il systemd servizio per le metriche parziali preconfigurate della GPU nella seguente posizione:

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

Abilita e avvia il systemd servizio con i seguenti comandi:

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

Configura tutte le metriche GPU disponibili

Configura tutte le metriche GPU disponibili utilizzando il servizio. dlami-cloudwatch-agent@all systemd Questo servizio configura le seguenti metriche:

- 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`

Puoi trovare il `systemd` servizio per tutte le metriche GPU preconfigurate disponibili nella seguente posizione:

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

Abilita e avvia il `systemd` servizio con i seguenti comandi:

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

Configura metriche GPU personalizzate

Se le metriche preconfigurate non soddisfano i tuoi requisiti, puoi creare un file di configurazione dell'agente personalizzato CloudWatch .

Crea un file di configurazione personalizzato

Per creare un file di configurazione personalizzato, consulta i passaggi dettagliati in Creare o modificare manualmente il file di configurazione dell' CloudWatch agente.

Per questo esempio, supponiamo che la definizione dello schema si trovi in`/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json`.

Configura le metriche con il tuo file personalizzato

Esegui il comando seguente per configurare l' CloudWatch agente in base al tuo file personalizzato:

```
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
```

Applicazione di patch di sicurezza per l'agente AWS CloudWatch

I DLAMI appena rilasciati sono configurati con le ultime patch di sicurezza disponibili per gli AWS CloudWatch agenti. Consultate le seguenti sezioni per aggiornare il vostro attuale DLAMI con le patch di sicurezza più recenti a seconda del sistema operativo scelto.

Amazon Linux 2

yumUsalo per ottenere le patch di sicurezza degli AWS CloudWatch agenti più recenti per un DLAMI Amazon Linux 2.

```
sudo yum update
```

Ubuntu

Per ottenere le patch AWS CloudWatch di sicurezza più recenti per un DLAMI con Ubuntu, è necessario reinstallare AWS CloudWatch l'agente utilizzando un link per il download di Amazon S3.

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

Per ulteriori informazioni sull'installazione dell' AWS CloudWatch agente utilizzando i link di download di Amazon S3, consulta Installazione ed esecuzione dell' CloudWatch agente sui server.

Configura le metriche con lo script preinstallato **`gpumon.py`**

Un'utilità denominata gpumon.py è preinstallata sulla DLAMI. Si integra CloudWatch e supporta il monitoraggio dell'utilizzo per GPU: memoria GPU, temperatura della GPU e potenza della GPU. Lo script invia periodicamente i dati monitorati a. CloudWatch È possibile configurare il livello di granularità dei dati a cui vengono inviati CloudWatch modificando alcune impostazioni nello script. Prima di avviare lo script, tuttavia, è necessario configurarlo per CloudWatch ricevere le metriche.

Come configurare ed eseguire il monitoraggio della GPU con CloudWatch

1. Crea un utente IAM o modificane uno esistente per disporre di una policy su cui pubblicare la metrica. CloudWatch Se crei un nuovo utente, prendi nota delle credenziali poiché saranno necessarie nella fase successiva.

   La policy IAM da cercare è «cloudwatch:». PutMetricData La policy che viene aggiunta è la seguente:

   ```
   {
      "Version": "2012-10-17",
      "Statement": [
           {
               "Action": [
   ```

```
                "cloudwatch:PutMetricData"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

> **Tip**
>
> Per ulteriori informazioni sulla creazione di un utente IAM e sull'aggiunta di policy per CloudWatch, consulta la CloudWatch documentazione.

2. Sul tuo DLAMI, esegui AWS configure e specifica le credenziali utente IAM.

```
$ aws configure
```

3. Potrebbe essere necessario apportare alcune modifiche all'utilità gpumon prima di eseguirla. È possibile trovare l'utilità gpumon e README nella posizione definita nel seguente blocco di codice. Per ulteriori informazioni sullo `gpumon.py` script, consulta la posizione dello script in Amazon S3.

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

   Opzioni:

   - Cambia la regione in gpumon.py se l'istanza NON è in us-east-1.
   - Modifica altri parametri, ad esempio CloudWatch `namespace` il periodo di riferimento `constore_reso`.

4. Attualmente lo script supporta solo Python 3. Attiva l'ambiente Python 3 del tuo framework preferito o attiva l'ambiente Python 3 generale DLAMI.

```
$ source activate python3
```

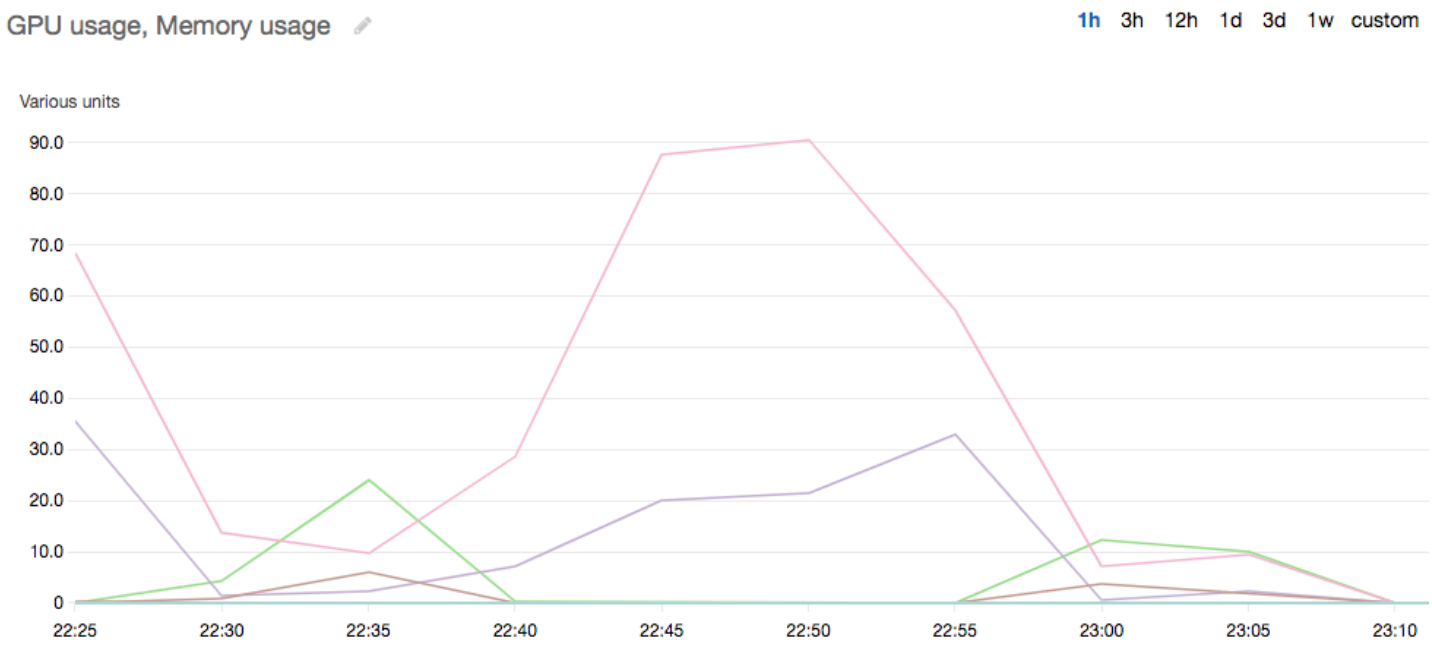
5. Esegui l'utilità gpumon in background.

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

6. Apri il browser su https://console.aws.amazon.com/cloudwatch/, quindi seleziona metrica. Avrà un namespace ". DeepLearningTrain

> **Tip**
>
> Puoi cambiare lo spazio dei nomi modificando gpumon.py. Puoi anche modificare l'intervallo di reporting regolando `store_reso`.

Di seguito è riportato un esempio di CloudWatch grafico che riporta un'esecuzione di gpumon.py che monitora un processo di formazione sull'istanza p2.8xlarge.



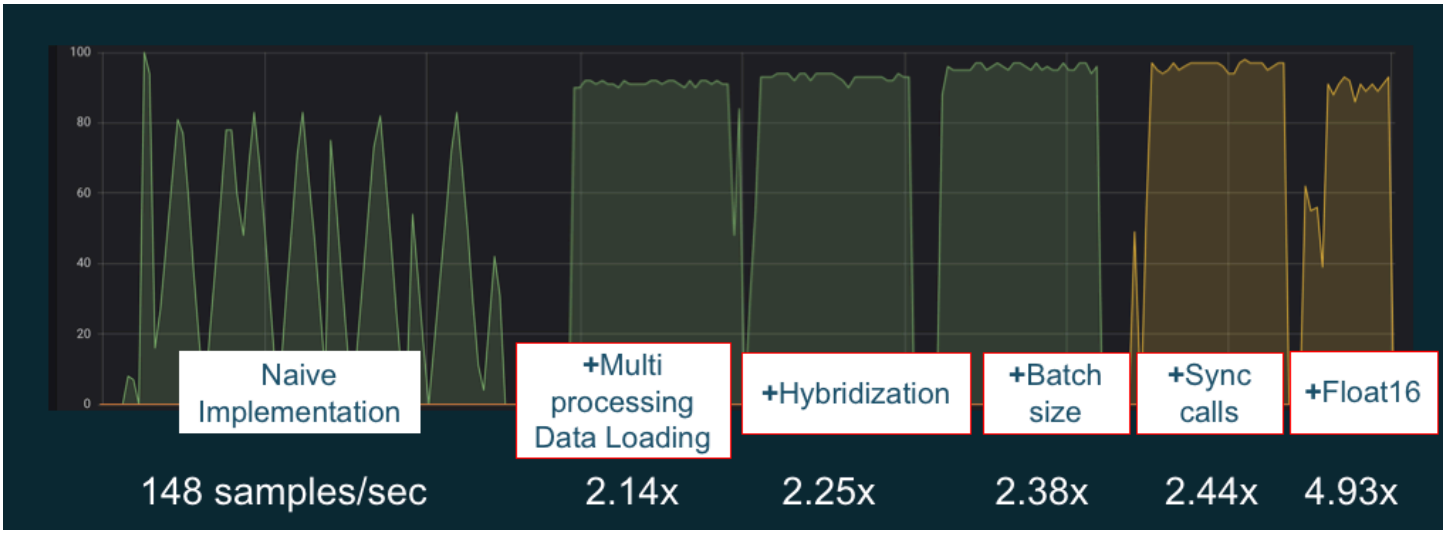
Questi altri argomenti sul monitoraggio e l'ottimizzazione GPU potrebbero essere interessanti:

- Monitoraggio
  - Monitora le GPU con CloudWatch
- Ottimizzazione
  - Pre-elaborazione
  - Addestramento

Ottimizzazione

Per ottenere il massimo dalle GPU, puoi ottimizzare la data pipeline e regolare la rete di deep learning. Come descritto nel grafico seguente, un'implementazione nativa o di base di una rete neurale potrebbe utilizzare la GPU in maniera non omogenea e non a pieno potenziale. Quando ottimizzi la pre-elaborazione e il caricamento dei dati, puoi ridurre il collo di bottiglia dalla CPU alla GPU. Puoi regolare la rete neurale stessa, utilizzando l'ibridazione (quando supportata dal framework), modificando le dimensioni batch e sincronizzando le chiamate. Puoi anche utilizzare training a precisione multipla (float16 o int8) nella maggior parte dei framework, che può avere un effetto significativo sul miglioramento del throughput.

Il grafico seguente mostra i miglioramenti delle prestazioni cumulativi quando si applicano ottimizzazioni differenti. I risultati dipenderanno dai dati in corso di elaborazione e dalla rete che si sta ottimizzando.



Esempio di ottimizzazioni delle prestazioni GPU. Origine grafico: Performance Tricks with MXNet Gluon

Le seguenti guide introducono le opzioni che funzionano con il tuo DLAMI e ti aiutano a migliorare le prestazioni della GPU.

Argomenti

- Pre-elaborazione
- Addestramento

Pre-elaborazione

La pre-elaborazione dei dati tramite trasformazioni o ottimizzazioni può essere spesso un processo basato sulla CPU e questo può essere il collo di bottiglia nella pipeline complessiva. I framework dispongono di operatori integrati per l'elaborazione di immagini, ma DALI (Data augmentation Library) mostra prestazioni migliorate rispetto a opzioni integrate dei framework.

- NVIDIA Data augmentation Library (DALI): DALI esegue l'offload dell'ottimizzazione dei dati nella GPU. Non è preinstallato su DLAMI, ma puoi accedervi installandolo o caricando un contenitore di framework supportato sul tuo DLAMI o su un'altra istanza Amazon Elastic Compute Cloud. Per informazioni dettagliate, consulta la pagina di progetto DALI sul sito Web NVIDIA. Per un caso d'uso di esempio e per scaricare esempi di codice, consulta l'esempio Preprocessing Training Performance. SageMaker
- nvJPEG: una libreria di decoder JPEG con accelerazione GPU per programmatori C. Supporta la decodifica di immagini singole o batch, nonché operazioni di trasformazione successive che sono comuni in deep learning. nvJPEG è integrato con DALI, oppure è possibile scaricarlo dalla pagina nvjpeg del sito Web NVIDIA e utilizzarlo separatamente.

Questi altri argomenti sul monitoraggio e l'ottimizzazione GPU potrebbero essere interessanti:

- Monitoraggio
  - Monitora le GPU con CloudWatch
- Ottimizzazione
  - Pre-elaborazione
  - Addestramento

Addestramento

Grazie al training a precisione mista puoi distribuire reti più grandi con la stessa quantità di memoria o ridurre l'utilizzo della memoria rispetto alla rete a precisione singola o doppia, registrando al contempo un incremento delle prestazioni di calcolo. Hai anche il vantaggio di trasferimenti dati più piccoli e rapidi, un fattore importante nel training distribuito a più nodi. Per sfruttare il training a precisione mista occorre regolare casting dei dati e perdita di scaling. Le guide seguenti descrivono come eseguire questa operazione per i framework che supportano la precisione mista.

- NVIDIA Deep Learning SDK: documenti sul sito Web di NVIDIA che descrivono l'implementazione a precisione mista per MXNet e. PyTorch TensorFlow

> **Tip**
>
> Assicurati di controllare il sito Web per il framework scelto e cerca "mixed precision" o "fp16" per le tecniche di ottimizzazione più recenti. Di seguito sono elencate alcune guide a precisione mista che possono essere utili:
>
> - Formazione a precisione mista con (video) - sul sito del blog NVIDIA TensorFlow .
> - Mixed-precision training using float16 with MXNet – Un articolo Domande frequenti sul sito Web MXNet.
> - NVIDIA Apex: uno strumento per un facile allenamento a precisione mista con PyTorch: un articolo di blog sul sito Web di NVIDIA.

Questi altri argomenti sul monitoraggio e l'ottimizzazione GPU potrebbero essere interessanti:

- Monitoraggio
  - Monitora le GPU con CloudWatch
- Ottimizzazione
  - Pre-elaborazione
  - Addestramento

Il chip AWS Inferentia con DLAMI

AWS Inferentia è un chip di machine learning personalizzato progettato da AWS cui è possibile utilizzare per previsioni di inferenza ad alte prestazioni. Per utilizzare il chip, configura un'istanza Amazon Elastic Compute Cloud e utilizza il kit di sviluppo software (SDK) AWS Neuron per richiamare il chip Inferentia. Per offrire ai clienti la migliore esperienza con Inferentia, Neuron è stato integrato in (AWS Deep Learning AMI DLAMI).

I seguenti argomenti mostrano come iniziare a usare Inferentia con DLAMI.

Indice

Avvio di un'istanza DLAMI con Neuron AWS

L'ultimo DLAMI è pronto per l'uso con AWS Inferentia e viene fornito con il AWS pacchetto API Neuron. Per avviare un'istanza DLAMI, vedere Avvio e configurazione di un DLAMI. Dopo aver installato un DLAMI, segui questi passaggi per assicurarti che il tuo chip AWS Inferentia e le risorse AWS Neuron siano attivi.

Indice

Verifica la tua istanza

Prima di usare l'istanza, verifica che sia correttamente configurata e configurata con Neuron.

Identificazione dei dispositivi AWS Inferentia

Per identificare il numero di dispositivi Inferentia sulla tua istanza, usa il seguente comando:

```
neuron-ls
```

Se all'istanza sono collegati dispositivi Inferentia, l'output sarà simile al seguente:

```
+--------+--------+--------+-----------+--------------+
| 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 |
+--------+--------+--------+-----------+--------------+
```

L'output fornito è tratto da un'istanza INF1.6xLarge e include le seguenti colonne:

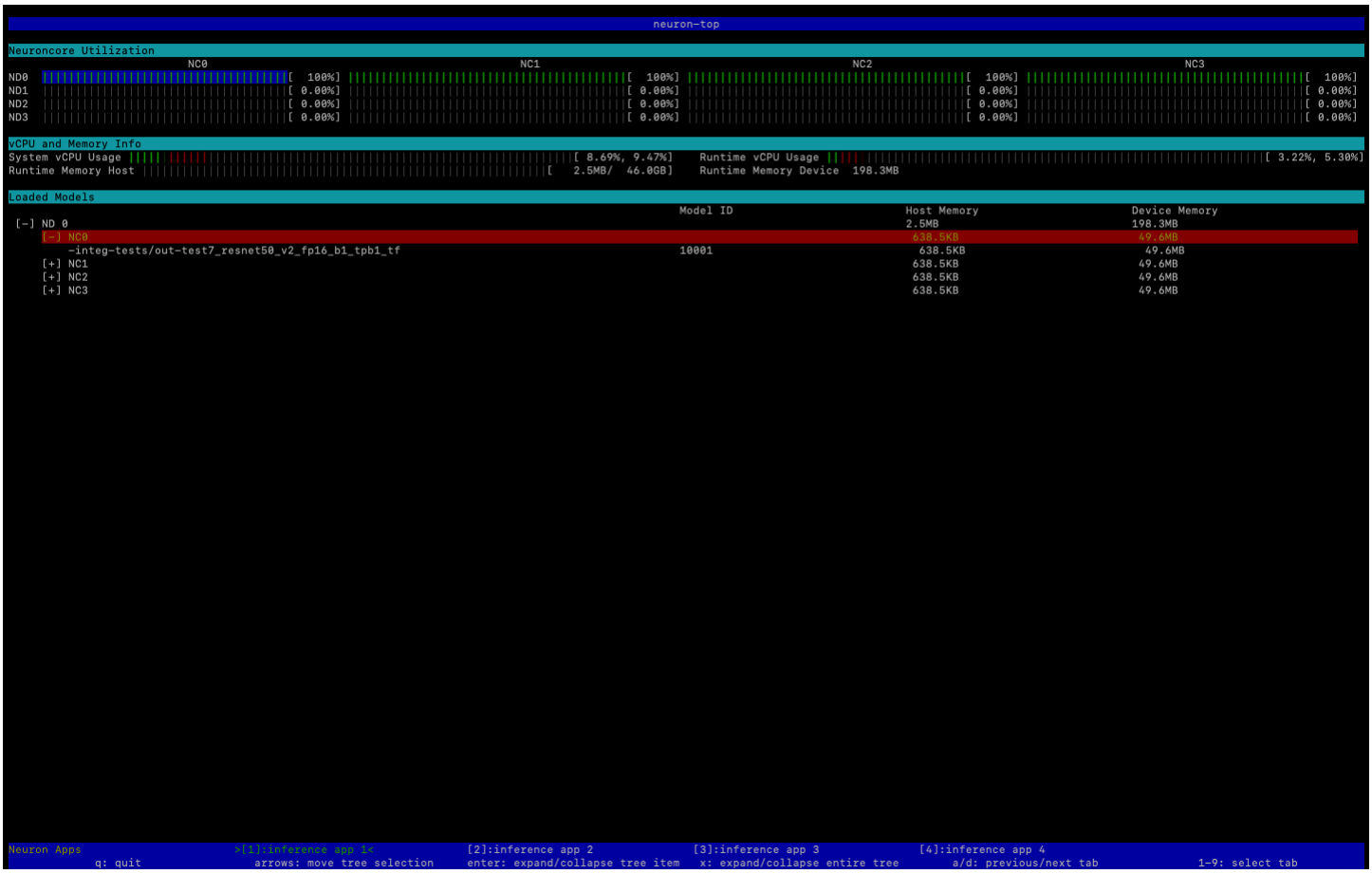
- NEURON DEVICE: L'ID logico assegnato a. NeuronDevice Questo ID viene utilizzato quando si configurano più runtime per utilizzarne diversi. NeuronDevices
- NEURON CORES: Il numero di NeuronCores core presenti in. NeuronDevice
- NEURON MEMORY: La quantità di memoria DRAM contenuta in. NeuronDevice
- DISPOSITIVI COLLEGATI: Altri NeuronDevices collegati a. NeuronDevice
- PCI BDF: L'ID PCI Bus Device Function (BDF) di. NeuronDevice

Visualizza l'utilizzo delle risorse

Visualizza informazioni utili sull' NeuronCore utilizzo della vCPU, sull'utilizzo della memoria, sui modelli caricati e sulle applicazioni Neuron con il comando. `neuron-top` L'avvio `neuron-top` senza argomenti mostrerà i dati per tutte le applicazioni di machine learning che utilizzano. NeuronCores

```
neuron-top
```

Quando un'applicazione ne utilizza quattro NeuronCores, l'output dovrebbe essere simile all'immagine seguente:



Per ulteriori informazioni sulle risorse per monitorare e ottimizzare le applicazioni di inferenza basate su Neuron, consulta Neuron Tools.

Utilizzo di Neuron Monitor (neuron-monitor)

Neuron Monitor raccoglie le metriche dai runtime Neuron in esecuzione sul sistema e trasmette i dati raccolti a stdout in formato JSON. Queste metriche sono organizzate in gruppi di metriche che puoi configurare fornendo un file di configurazione. Per ulteriori informazioni su Neuron Monitor, consulta la Guida per l'utente di Neuron Monitor.

Aggiornamento del software Neuron

Per informazioni su come aggiornare il software Neuron SDK all'interno di DLAMI, consultate la Neuron Setup Guide. AWS

Fase succcessiva

Usare il DLAMI con Neuron AWS

Usare il DLAMI con Neuron AWS

Un tipico flusso di lavoro con AWS Neuron SDK consiste nel compilare un modello di machine learning precedentemente addestrato su un server di compilazione. Successivamente, distribuisci gli artefatti alle istanze Inf1 per l'esecuzione. AWS Deep Learning AMI (DLAMI) è preinstallato con tutto il necessario per compilare ed eseguire l'inferenza in un'istanza Inf1 che utilizza Inferentia.

Le seguenti sezioni descrivono come usare DLAMI con Inferentia.

Indice

Utilizzo di TensorFlow -Neuron e del Neuron Compiler AWS

Questo tutorial mostra come utilizzare il compilatore AWS Neuron per compilare il modello Keras ResNet -50 ed esportarlo come modello salvato in formato. SavedModel Questo formato è un tipico

formato intercambiabile del modello. TensorFlow Il tutorial illustra anche come eseguire l'inferenza su un'istanza di Inf1 con input di esempio.

Per ulteriori informazioni su Neuron SDK, consulta la documentazione di Neuron SDK.AWS

Indice

Prerequisiti

Prima di utilizzare questo tutorial, è necessario aver completato la procedura di configurazione in Avvio di un'istanza DLAMI con Neuron AWS. È inoltre necessario avere dimestichezza con il deep learning e l'uso del DLAMI.

Attivare l'ambiente Conda

Attiva l'ambiente TensorFlow -Neuron conda usando il seguente comando:

```
source activate aws_neuron_tensorflow_p36
```

Per uscire dall'ambiente Conda corrente, eseguire il comando seguente:

```
source deactivate
```

Compilazione Resnet50

Creare uno script Python chiamato **`tensorflow_compile_resnet50.py`** che abbia il seguente contenuto. Questo script Python compila il modello Keras ResNet 50 e lo esporta come modello salvato.

```
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')
```

Compilare il modello utilizzando il seguente comando:

```
python tensorflow_compile_resnet50.py
```

Il processo di compilazione richiederà alcuni minuti. Al termine, l'output dovrebbe essere simile al seguente:

```
...
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
...
```

Dopo la compilazione, il modello salvato viene compresso a **`ws_resnet50/resnet50_neuron.zip`**. Decomprimere il modello e scaricare l'immagine di esempio per l'inferenza utilizzando i seguenti comandi:

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

ResNet50 Inferenza

Creare uno script Python chiamato **`tensorflow_infer_resnet50.py`** che abbia il seguente contenuto. Questo script esegue l'inferenza sul modello scaricato utilizzando un modello di inferenza precedentemente compilato.

```
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])
```

Eseguire l'inferenza sul modello utilizzando il seguente comando:

```
python tensorflow_infer_resnet50.py
```

L'aspetto dell'output deve essere simile al seguente:

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

Fase succcessiva

Utilizzo di AWS Neuron Serving TensorFlow

Utilizzo di AWS Neuron Serving TensorFlow

Questo tutorial mostra come costruire un grafico e aggiungere una fase di compilazione di AWS Neuron prima di esportare il modello salvato da utilizzare con Serving. TensorFlow TensorFlow Serving è un sistema di servizio che consente di aumentare l'inferenza su una rete. Neuron TensorFlow Serving utilizza la stessa API del normale Serving. TensorFlow L'unica differenza è che un modello salvato deve essere compilato per AWS Inferentia e il punto di ingresso è un nome binario diverso. `tensorflow_model_server_neuron` Il file binario si trova in `/usr/local/bin/tensorflow_model_server_neuron` ed è preinstallato nel DLAMI.

Per ulteriori informazioni su Neuron SDK, consulta la documentazione di Neuron SDK.AWS

Indice

- Generare richieste di inferenza al server del modello

Prerequisiti

Prima di utilizzare questo tutorial, è necessario aver completato la procedura di configurazione in Avvio di un'istanza DLAMI con Neuron AWS. È inoltre necessario avere dimestichezza con il deep learning e l'uso del DLAMI.

Attivare l'ambiente Conda

Attiva l'ambiente TensorFlow -Neuron conda usando il seguente comando:

```
source activate aws_neuron_tensorflow_p36
```

Se è necessario uscire dall'ambiente Conda corrente, eseguire:

```
source deactivate
```

Compilare ed esportare il modello salvato

Crea uno script Python chiamato `tensorflow-model-server-compile.py` con il seguente contenuto. Questo script costruisce un grafico e lo compila usando Neuron. Esporta quindi il grafico compilato come modello salvato.

```
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)
```

Compilare il modello utilizzando il seguente comando:

```
python tensorflow-model-server-compile.py
```

L'aspetto dell'output deve essere simile al seguente:

```
...
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
```

Servire il modello salvato

Una volta compilato il modello, è possibile utilizzare il seguente comando per servire il modello salvato con il binario tensorflow_model_server_neuron:

```
tensorflow_model_server_neuron --model_name=resnet50_inf1 \
    --model_base_path=$HOME/resnet50_inf1/ --port=8500 &
```

L'aspetto dell'output sarà simile al seguente. Il modello compilato viene inserito nella DRAM del dispositivo Inferentia dal server per prepararsi all'inferenza.

```
...
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 ...
```

Generare richieste di inferenza al server del modello

Creare uno script Python chiamato `tensorflow-model-server-infer.py` con il seguente contenuto. Questo script esegue inferenza tramite gRPC, che è framework di servizio.

```
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))
```

Eseguire l'inferenza sul modello utilizzando gRPC con il seguente comando:

```
python tensorflow-model-server-infer.py
```

L'aspetto dell'output deve essere simile al seguente:

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

Utilizzo di MXNet-Neuron e del Neuron Compiler AWS

L'API di compilazione MXNet-Neuron fornisce un metodo per compilare un grafico modello che è possibile eseguire su un dispositivo Inferentia. AWS

In questo esempio, si utilizza l'API per compilare un ResNet modello -50 e utilizzarlo per eseguire l'inferenza.

Per ulteriori informazioni su Neuron SDK, consulta la documentazione di Neuron SDK.AWS

Indice

Prerequisiti

Prima di utilizzare questo tutorial, è necessario aver completato la procedura di configurazione in Avvio di un'istanza DLAMI con Neuron AWS. È inoltre necessario avere dimestichezza con il deep learning e l'uso del DLAMI.

Attivare l'ambiente Conda

Attivare l'ambiente Conda MXNet-Neuron utilizzando il seguente comando:

```
source activate aws_neuron_mxnet_p36
```

Per uscire dall'ambiente conda corrente, eseguire:

```
source deactivate
```

Compilazione Resnet50

Creare uno script Python chiamato **`mxnet_compile_resnet50.py`** con il seguente contenuto. Questo script utilizza l'API Python di compilazione MXNet-Neuron per compilare un modello -50. ResNet

```
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)
```

Compilare il modello utilizzando il seguente comando:

```
python mxnet_compile_resnet50.py
```

La compilazione richiederà alcuni minuti. Al termine della compilazione, i seguenti file si troveranno nella directory corrente:

```
resnet-50-0000.params
resnet-50-symbol.json
compiled_resnet50-0000.params
compiled_resnet50-symbol.json
```

ResNet50 Inferenza

Creare uno script Python chiamato **`mxnet_infer_resnet50.py`** con il seguente contenuto. Questo script scarica un'immagine di esempio e la usa per eseguire l'inferenza con il modello compilato.

```
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]))
```

Eseguire l'inferenza con il modello compilato utilizzando il seguente comando:

```
python mxnet_infer_resnet50.py
```

L'aspetto dell'output deve essere simile al seguente:

```
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
```

Fase succcessiva

Utilizzo di MXNet-Neuron Model Serving

Utilizzo di MXNet-Neuron Model Serving

Questo tutorial illustra come utilizzare un modello MXNet pre-addestrato per eseguire la classificazione delle immagini in tempo reale con Multi Model Server (MMS). MMS è uno easy-to-use strumento flessibile per fornire modelli di deep learning addestrati utilizzando qualsiasi framework di machine learning o deep learning. Questo tutorial include una fase di compilazione con AWS Neuron e un'implementazione di MMS con MXNet.

Per ulteriori informazioni su Neuron SDK, consulta la documentazione di Neuron SDK.AWS

Indice

Prerequisiti

Prima di utilizzare questo tutorial, è necessario aver completato la procedura di configurazione in Avvio di un'istanza DLAMI con Neuron AWS. È inoltre necessario avere dimestichezza con il deep learning e l'uso del DLAMI.

Attivare l'ambiente Conda

Attivare l'ambiente Conda MXNet-Neuron utilizzando il seguente comando:

```
source activate aws_neuron_mxnet_p36
```

Per uscire dall'ambiente conda corrente, eseguire:

```
source deactivate
```

Scarica il codice di esempio

Per eseguire questo esempio, scaricare il codice di esempio utilizzando i seguenti comandi:

```
git clone https://github.com/awslabs/multi-model-server
cd multi-model-server/examples/mxnet_vision
```

Compila il modello

Creare uno script Python chiamato `multi-model-server-compile.py` con il seguente contenuto. Questo script compila il modello ResNet 50 nella destinazione del dispositivo Inferentia.

```
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)
```

Per compilare il modello, utilizzare il seguente comando:

```
python multi-model-server-compile.py
```

L'aspetto dell'output deve essere simile al seguente:

```
...
[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!
```

Creare un file denominato `signature.json` con il seguente contenuto per configurare il nome e la forma di input:

```
{
  "inputs": [
    {
      "data_name": "data",
      "data_shape": [
        1,
        3,
        224,
        224
      ]
    }
  ]
```

```
}
```

Scaricare il file `synset.txt` utilizzando il comando seguente: Questo file è un elenco di nomi per ImageNet le classi di previsione.

```
curl -O https://s3.amazonaws.com/model-server/model_archive_1.0/examples/
squeezenet_v1.1/synset.txt
```

Creare una classe di servizio personalizzata seguendo il modello nella cartella `model_server_template`. Copiare il modello nella directory di lavoro corrente utilizzando il seguente comando:

```
cp -r ../model_service_template/* .
```

Modificare il modulo `mxnet_model_service.py` per sostituire il contesto `mx.cpu()` con il contesto `mx.neuron()` come segue. È inoltre necessario commentare la copia dei dati non necessaria `model_input` perché MXNet-Neuron non supporta le API NDArray e Gluon.

```
...
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]
```

Comprimere il modello con model-archiver utilizzando i seguenti comandi:

```
cd ~/multi-model-server/examples
model-archiver --force --model-name resnet-50_compiled --model-path mxnet_vision --
handler mxnet_vision_service:handle
```

Eseguire l’inferenza

Avviare il server del modello Multi Model Server e caricare il modello che utilizza l'API RESTful con i comandi seguenti. Assicurarsi che neuron-rtd sia in esecuzione con le impostazioni predefinite.

```
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
```

Eseguire l'inferenza utilizzando un'immagine di esempio con i seguenti comandi:

```
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
```

L'aspetto dell'output deve essere simile al seguente:

```
[
  {
    "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"
  }
]
```

Per eseguire il cleanup dopo il test, utilizzare il comando delete tramite l'API RESTful e arrestare il server del modello utilizzando i seguenti comandi:

```
curl -X DELETE http://127.0.0.1:8081/models/resnet-50_compiled

multi-model-server --stop
```

Verrà visualizzato l'output seguente:

```
{
  "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
```

Utilizzo di PyTorch -Neuron e del Neuron Compiler AWS

L'API di compilazione PyTorch -Neuron fornisce un metodo per compilare un grafico modello che è possibile eseguire su un dispositivo Inferentia. AWS

Un modello addestrato deve essere compilato in un target Inferentia prima di poter essere distribuito nelle istanze di Inf1. Il seguente tutorial compila il modello torchvision ResNet 50 e lo esporta come modulo salvato. TorchScript Questo modello viene quindi utilizzato per eseguire l'inferenza.

Per comodità, questa esercitazione utilizza un'istanza di Inf1 sia per la compilazione sia per l'inferenza. In pratica, è possibile compilare il modello utilizzando un altro tipo di istanza, ad esempio la famiglia di istanze c5. È quindi necessario distribuire il modello compilato al server di inferenza Inf1. Per ulteriori informazioni, consulta la documentazione di AWS Neuron SDK PyTorch .

Indice

Prerequisiti

Prima di utilizzare questo tutorial, è necessario aver completato la procedura di configurazione in Avvio di un'istanza DLAMI con Neuron AWS. È inoltre necessario avere dimestichezza con il deep learning e l'uso del DLAMI.

Attivare l'ambiente Conda

Attiva l'ambiente PyTorch -Neuron conda usando il seguente comando:

```
source activate aws_neuron_pytorch_p36
```

Per uscire dall'ambiente conda corrente, eseguire:

```
source deactivate
```

Compilazione Resnet50

Creare uno script Python chiamato **`pytorch_trace_resnet50.py`** con il seguente contenuto. Questo script utilizza l'API Python di compilazione PyTorch -Neuron per compilare un modello -50. ResNet

> **Note**
>
> Esiste una dipendenza tra le versioni di torchvision e il pacchetto torch di cui dovresti essere a conoscenza durante la compilazione dei modelli torchvision. Queste regole di dipendenza possono essere gestite tramite pip. Torchvision==0.6.1 corrisponde alla versione torch==1.5.1, mentre torchvision==0.8.2 corrisponde alla versione torch==1.7.1.

```
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")
```

Eseguire lo script di compilazione.

```
python pytorch_trace_resnet50.py
```

La compilazione richiederà alcuni minuti. Al termine della compilazione, il modello compilato viene salvato come `resnet50_neuron.pt` nella directory locale.

ResNet50 Inferenza

Creare uno script Python chiamato **`pytorch_infer_resnet50.py`** con il seguente contenuto. Questo script scarica un'immagine di esempio e la usa per eseguire l'inferenza con il modello compilato.

```
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) )
```

Eseguire l'inferenza con il modello compilato utilizzando il seguente comando:

```
python pytorch_infer_resnet50.py
```

L'aspetto dell'output deve essere simile al seguente:

```
Top 5 labels:
 ['tiger', 'lynx', 'tiger_cat', 'Egyptian_cat', 'tabby']
```

Il Graviton DLAMI

AWS Le DLAMI GPU Graviton sono progettate per fornire alte prestazioni ed efficienza in termini di costi per carichi di lavoro di deep learning. In particolare, il tipo di istanza g5G presenta il processore AWS Graviton2 basato su ARM, che è stato costruito da zero AWS e ottimizzato per il modo in cui i clienti eseguono i loro carichi di lavoro nel cloud. AWS Le DLAMI della GPU Graviton sono

preconfigurate con Docker, NVIDIA Docker, NVIDIA Driver, CUDA, cuDNN, NCCL e TensorRT, oltre ai più diffusi framework di machine learning come e. TensorFlow PyTorch

Con il tipo di istanza g5G, puoi sfruttare i vantaggi in termini di prezzo e prestazioni di Graviton2 per implementare modelli di deep learning accelerati da GPU a un costo notevolmente inferiore rispetto alle istanze basate su x86 con accelerazione GPU.

Seleziona un Graviton DLAMI

Avvia un'istanza g5G con il Graviton DLAMI di tua scelta.

Per step-by-step istruzioni sull'avvio di un DLAMI, vedere Avvio e configurazione di un DLAMI.

Per un elenco dei DLAMI Graviton più recenti, consulta le Note di rilascio per DLAMI.

Inizia

I seguenti argomenti mostrano come iniziare a usare Graviton DLAMI.

Indice

Utilizzo della GPU Graviton DLAMI

AWS Deep Learning AMI È pronto per l'uso con le GPU Graviton basate su processori Arm. La GPU Graviton DLAMI è dotata di una piattaforma fondamentale di driver GPU e librerie di accelerazione per implementare il tuo ambiente di deep learning personalizzato. Docker e NVIDIA Docker sono preconfigurati sulla GPU Graviton DLAMI per consentirti di distribuire applicazioni containerizzate. Consulta le note di rilascio per ulteriori dettagli sulla GPU Graviton DLAMI.

Indice

Controlla lo stato della GPU

Usa l'interfaccia di gestione del sistema NVIDIA per controllare lo stato della tua GPU Graviton.

```
nvidia-smi
```

L'output del `nvidia-smi` comando dovrebbe essere simile al seguente:

```
+-----------------------------------------------------------------------------+
| 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                                                 |
+-----------------------------------------------------------------------------+
```

Controlla la versione CUDA

Esegui il seguente comando per verificare la tua versione CUDA:

```
/usr/local/cuda/bin/nvcc --version | grep Cuda
```

L'aspetto dell'output sarà simile al seguente:

```
nvcc: NVIDIA (R) Cuda compiler driver
Cuda compilation tools, release 11.4, V11.4.120
```

Verifica Docker

Esegui un contenitore CUDA DockerHubper verificare la funzionalità Docker sulla tua GPU Graviton:

```
sudo docker run --platform=linux/arm64 --rm \
    --gpus all nvidia/cuda:11.4.2-base-ubuntu20.04 nvidia-smi
```

L'aspetto dell'output sarà simile al seguente:

```
+-----------------------------------------------------------------------------+
| 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

Usa il seguente comando per accedere allo strumento da riga di comando TensorRT:

```
trtexec
```

L'aspetto dell'output sarà simile al seguente:

```
&&&& RUNNING TensorRT.trtexec [TensorRT v8200] # trtexec
...
&&&& PASSED TensorRT.trtexec [TensorRT v8200] # trtexec
```

Sono disponibili ruote TensorRT Python per l'installazione su richiesta. Puoi trovare queste ruote nelle seguenti posizioni dei file:

```
/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
```

Per ulteriori dettagli, consulta la documentazione di NVIDIA TensorRT.

Esegui esempi CUDA

La GPU Graviton DLAMI fornisce esempi CUDA precompilati per aiutarti a verificare diverse funzionalità CUDA.

```
ls /usr/local/cuda/compiled_samples
```

Ad esempio, esegui l'esempio con il seguente comando: `vectorAdd`

```
/usr/local/cuda/compiled_samples/vectorAdd
```

L'aspetto dell'output sarà simile al seguente:

```
[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
```

Esegui l'`transposee`esempio:

```
/usr/local/cuda/compiled_samples/transpose
```

L'aspetto dell'output sarà simile al seguente:

```
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
```

Argomento successivo

Utilizzo della GPU Graviton DLAMI TensorFlow

Utilizzo della GPU Graviton DLAMI TensorFlow

AWS Deep Learning AMI È pronto per l'uso con le GPU Graviton basate su processori Arm ed è ottimizzato per. TensorFlow La GPU Graviton TensorFlow DLAMI include un ambiente Python preconfigurato con TensorFlow Serving per casi d'uso di inferenza di deep learning. Consulta le note di rilascio per ulteriori dettagli sulla GPU Graviton DLAMI TensorFlow .

Indice

- Verifica TensorFlow e TensorFlow servi la disponibilità dell'API
- Esegui l'inferenza di esempio con Serving TensorFlow

Verifica la disponibilità del servizio TensorFlow

Esegui il seguente comando per verificare la disponibilità e la versione di Serving: TensorFlow

```
tensorflow_model_server --version
```

L'aspetto dell'output sarà simile al seguente:

```
TensorFlow ModelServer: 0.0.0+dev.sha.3e05381e
TensorFlow Library: 2.8.0
```

Verifica TensorFlow e TensorFlow servi la disponibilità dell'API

Esegui il seguente comando per verificare la disponibilità TensorFlow e la TensorFlow Serving API:

```
python3 -c "import tensorflow, tensorflow_serving"
```

Se il comando ha esito positivo, non viene prodotto alcun output.

Esegui l'inferenza di esempio con Serving TensorFlow

Usa i seguenti comandi per scaricare un modello ResNet 50 pre-addestrato ed eseguire l'inferenza utilizzando Serving: 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" &
```

L'aspetto dell'output sarà simile al seguente:

```
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 ...
```

Usa l'`resnet_client`esempio TensorFlow Serving per eseguire l'inferenza:

```
python3 serving/tensorflow_serving/example/resnet_client.py
```

L'aspetto dell'output sarà simile al seguente:

```
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
```

Stop TensorFlow Serving con il seguente comando:

```
kill $(pidof tensorflow_model_server)
```

Argomento successivo

Utilizzo della GPU Graviton DLAMI PyTorch

Utilizzo della GPU Graviton DLAMI PyTorch

AWS Deep Learning AMI È pronto per l'uso con le GPU Graviton basate su processori Arm ed è ottimizzato per. PyTorch La GPU Graviton PyTorch DLAMI include un ambiente Python

preconfigurato con PyTorche TorchServeper casi d'uso di addestramento e TorchVisioninferenza di deep learning. Consulta le note di rilascio per ulteriori dettagli sulla GPU Graviton DLAMI PyTorch .

Indice

Verifica dell' PyTorch ambiente Python

Connettiti alla tua istanza G5g e attiva l'ambiente Conda di base con il seguente comando:

```
source activate base
```

Il prompt dei comandi dovrebbe indicare che stai lavorando nell'ambiente Conda di base, che contiene e altre PyTorch librerie TorchVision.

```
(base) $
```

Verificate i percorsi utensile predefiniti dell' PyTorch ambiente:

```
(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
```

Verificate che Torch e TorchVersion C siano disponibili, controllate le relative versioni e verificate le funzionalità di base:

```
(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)
```

Esegui Training Sample con PyTorch

Esegui un esempio di lavoro di formazione MNIST:

```
git clone https://github.com/pytorch/examples.git
cd examples/mnist
python main.py
```

L'aspetto dell'output sarà simile al seguente:

```
...
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%)
```

Esegui Inference Sample con PyTorch

Usa i seguenti comandi per scaricare un modello densenet161 pre-addestrato ed eseguire l'inferenza utilizzando: 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
```

L'aspetto dell'output sarà simile al seguente:

```
{
  "tiger_cat": 0.4693363308906555,
  "tabby": 0.4633873701095581,
  "Egyptian_cat": 0.06456123292446136,
  "lynx": 0.0012828150065615773,
  "plastic_bag": 0.00023322898778133094
}
```

Utilizzate i seguenti comandi per annullare la registrazione del modello densenet161 e arrestare il server:

```
curl -X DELETE http://localhost:8081/models/densenet161/1.0
torchserve --stop
```

L'aspetto dell'output sarà simile al seguente:

```
{
  "status": "Model \"densenet161\" unregistered"
}
```

```
TorchServe has stopped.
```

I DLAMI dell'Habana

Le istanze con acceleratori Habana sono progettate per fornire prestazioni elevate ed efficienza in termini di costi per carichi di lavoro di formazione basati su modelli di deep learning. In particolare, i tipi di istanze DL1 utilizzano gli acceleratori Habana Gaudi di Habana Labs, un'azienda Intel. Le istanze con acceleratori Habana sono configurate con il software Habana SynapseAI e preintegrate con i più diffusi framework di machine learning come e. TensorFlow PyTorch

I seguenti argomenti mostrano come iniziare a utilizzare l'hardware Habana Gaudi con DLAMI.

Indice

Lancio di un DLAMI Habana

L'ultimo DLAMI è pronto per l'uso con gli acceleratori Habana Gaudi. Usa i seguenti passaggi per avviare Habana DLAMI e assicurarti che le tue risorse Python e specifiche del framework siano attive. Per ulteriori risorse di configurazione, consultate il repository di configurazione e installazione di Habana Gaudi.

Indice

Seleziona un Habana DLAMI

Avvia un'istanza DL1 con il DLAMI Habana di tua scelta.

Per step-by-step istruzioni sull'avvio di un DLAMI, vedere Avvio e configurazione di un DLAMI.

Per un elenco dei DLAMI Habana più recenti, consulta le note di rilascio per DLAMI.

Attiva l'ambiente Python

Connect all'istanza DL1 e attiva l'ambiente Python consigliato per Habana DLAMI. Per verificare l'ambiente Python consigliato, selezionate il vostro DLAMI nelle Note di rilascio.

Importazione del framework di Machine Learning

Le istanze con acceleratori Habana sono preintegrate con i più diffusi framework di machine learning come e. TensorFlow PyTorch Importa il framework di machine learning che preferisci.

Importa TensorFlow

Per utilizzarlo TensorFlow sul tuo Habana DLAMI, vai alla cartella dell'ambiente Python che hai attivato e importato. TensorFlow

```
/usr/bin/$PYTHON_VERSION
import tensorflow
tensorflow.__version__
```

Per verificare la TensorFlow versione compatibile con il tuo Habana DLAMI, seleziona il tuo DLAMI nelle Note di rilascio.

Importa PyTorch

Per utilizzarlo PyTorch sul tuo Habana DLAMI, vai alla cartella dell'ambiente Python che hai attivato e importa la versione appropriata. PyTorch

```
/usr/bin/$PYTHON_VERSION
import torch
torch.__version__
```

Per verificare la PyTorch versione compatibile con il tuo Habana DLAMI, seleziona il tuo DLAMI nelle Note di rilascio.

Per ulteriori informazioni su come eseguire e addestrare modelli di machine learning all'interno TensorFlow e sull' PyTorch utilizzo di Habana DLAMI, consulta l'archivio Habana Model References. GitHub Per ulteriori risorse su come lavorare con Habana DLAMI, consulta la documentazione di Habana Gaudi.

Inferenza

Questa sezione fornisce tutorial su come eseguire l'inferenza utilizzando i framework e gli strumenti di DLAMI.

Per esercitazioni utilizzando Elastic Inference, consulta Working with Amazon Elastic Inference

Inferenza con i framework

- Usa Apache MXNet (incubazione) per l'inferenza con un modello ONNX
- Usa Apache MXNet (incubazione) per l'inferenza con un modello 5.0 ResNet
- Utilizzo di CNTK per l'inferenza con un modello ONNX

Strumenti di inferenza

- Model Server for Apache MXNet (MMS)
- TensorFlow Servire

Usa Apache MXNet (incubazione) per l'inferenza con un modello ONNX

Come utilizzare un modello ONNX per l'inferenza delle immagini con Apache MXNet (incubazione)

1.
   - (Opzione per Python 3) - Attiva l'ambiente Python 3 Apache MXNet (incubazione):

     ```
     $ source activate mxnet_p36
     ```

   - (Opzione per Python 2) - Attiva l'ambiente Python 2 Apache MXNet (incubazione):

     ```
     $ source activate mxnet_p27
     ```

2. Per gli altri passaggi, si presuppone che venga utilizzato l'ambiente `mxnet_p36`.
3. Scaricare un'immagine di un husky.

   ```
   $ curl -O https://upload.wikimedia.org/wikipedia/commons/b/b5/Siberian_Husky_bi-
   eyed_Flickr.jpg
   ```

4. Scaricare un elenco di classi da utilizzare con questo modello.

   ```
   $ curl -O https://gist.githubusercontent.com/yrevar/6135f1bd8dcf2e0cc683/raw/
   d133d61a09d7e5a3b36b8c111a8dd5c4b5d560ee/imagenet1000_clsid_to_human.pkl
   ```

5. Scarica il modello VGG 16 pre-addestrato in formato ONNX.

   ```
   $ wget -O vgg16.onnx https://github.com/onnx/models/raw/master/vision/
   classification/vgg/model/vgg16-7.onnx
   ```

6. Utilizzare l'editor di testo preferito per creare uno script che ha i seguenti contenuti. Questo script utilizzerà l'immagine dell'husky, otterrà un risultato di previsione dal modello preformato, quindi eseguirà una ricerca nel file di classi e restituirà un risultato di classificazione.

```
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. Eseguire lo script. Il risultato dovrebbe essere simile a quanto segue:

```
248
Eskimo dog, husky
```

Usa Apache MXNet (incubazione) per l'inferenza con un modello 5.0 ResNet

Come utilizzare un modello Apache MXNet (incubazione) pre-addestrato con l'API Symbol per l'inferenza delle immagini con MXNet

1. • (Opzione per Python 3) - Attiva l'ambiente Python 3 Apache MXNet (incubazione):

```
$ source activate mxnet_p36
```

   • (Opzione per Python 2) - Attiva l'ambiente Python 2 Apache MXNet (incubazione):

```
$ source activate mxnet_p27
```

2. Per gli altri passaggi, si presuppone che venga utilizzato l'ambiente `mxnet_p36`.
3. Utilizzare l'editor di testo preferito per creare uno script che ha i seguenti contenuti. Questo script scaricherà i file del modello ResNet -50 (resnet-50-0000.params e resnet-50-symbol.json) e l'elenco delle etichette (synset.txt), scaricherà un'immagine cat per ottenere un risultato di previsione dal modello pre-addestrato, quindi lo cercherà nell'elenco dei risultati nelle etichette, restituendo un risultato di previsione.

```
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. Eseguire lo script. Il risultato dovrebbe essere simile a quanto segue:

```
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
```

Utilizzo di CNTK per l'inferenza con un modello ONNX

> **Note**
>
> Non includiamo più gli ambienti CNTK, Caffe, Caffe2 e Theano Conda in AWS Deep Learning AMI a partire dalla versione v28. AWS Deep Learning AMI Le versioni precedenti di che contengono questi ambienti continueranno a essere disponibili. Tuttavia, verranno forniti aggiornamenti a questi ambienti solo se sono disponibili correzioni di protezione pubblicate dalla comunità open source per questi framework.

> **Note**
>
> Il modello VGG-16 utilizzato in questo tutorial consuma una grande quantità di memoria. Quando si seleziona l' AWS Deep Learning AMI istanza, potrebbe essere necessaria un'istanza con più di 30 GB di RAM.

Come utilizzare un modello ONNX per l'inferenza con CNTK

1.
   - (Opzione per Python 3) – Attivare l'ambiente Python 3 CNTK:

     ```
     $ source activate cntk_p36
     ```

   - (Opzione per Python 2) – Attivare l'ambiente Python 2 CNTK:

     ```
     $ source activate cntk_p27
     ```

2. Per gli altri passaggi, si presuppone che venga utilizzato l'ambiente `cntk_p36`.
3. Creare un nuovo file con l'editor di testo e utilizzare il programma seguente in uno script per aprire il file in formato ONNX in CNTK.

   ```
   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!")
   ```

   Dopo l'esecuzione di questo script, CNTK avrà caricato il modello.
4. È anche possibile provare a eseguire l'inferenza con CNTK. Per prima cosa, scaricare un'immagine di un husky.

   ```
   $ curl -O https://upload.wikimedia.org/wikipedia/commons/b/b5/Siberian_Husky_bi-
   eyed_Flickr.jpg
   ```

5. Scaricare quindi un elenco di classi da utilizzare con questo modello.

   ```
   $ curl -O https://gist.githubusercontent.com/yrevar/6135f1bd8dcf2e0cc683/raw/
   d133d61a09d7e5a3b36b8c111a8dd5c4b5d560ee/imagenet1000_clsid_to_human.pkl
   ```

6. Modificare lo script creato in precedenza affinché abbia il contenuto seguente. Questa nuova versione utilizzerà l'immagine dell'husky, otterrà un risultato di stima, quindi eseguirà una ricerca nel file di classi e restituirà un risultato di stima.

```
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. Eseguire lo script. Il risultato dovrebbe essere simile a quanto segue:

```
248
Eskimo dog, husky
```

Utilizzo di frameworks con ONNX

L'AMI Deep Learning con Conda ora supporta i modelli Open Neural Network Exchange (ONNX) per alcuni framework. Scegli uno degli argomenti elencati di seguito per imparare a usare ONNX sulla tua AMI Deep Learning con Conda.

Se si desidera utilizzare un modello ONNX esistente su un DLAMI, vedere. Usa Apache MXNet (incubazione) per l'inferenza con un modello ONNX

Informazioni su ONNX

ONNX (Open Neural Network Exchange) è un formato aperto utilizzato per rappresentare modelli di apprendimento profondo. ONNX è supportato da Amazon Web Services, Microsoft, Facebook e diversi altri partner. Puoi progettare modelli di apprendimento profondi, eseguirne il training e distribuirli con qualsiasi framework di tua scelta. I modelli ONNX offrono il vantaggio di poter essere spostati facilmente da un framework all'altro.

L'AMI Deep Learning con Conda evidenzia attualmente alcune delle funzionalità di ONNX nella seguente raccolta di tutorial.

- Tutorial sulla conversione di un modello MXNet in ONNX per CNTK
- Tutorial sulla conversione di un modello Chainer in ONNX per CNTK
- Tutorial sulla conversione di un modello Chainer in ONNX per MXNet
- PyTorch da ONNX a CNTK Tutorial
- PyTorch da ONNX a MXNet Tutorial

Puoi inoltre consultare i tutorial e la documentazione di progetto ONNX:

- Progetto ONNX su GitHub
- ONNX Tutorials

Tutorial sulla conversione di un modello MXNet in ONNX per CNTK

> **Note**
>
> Non includiamo più gli ambienti CNTK, Caffe, Caffe2 e Theano Conda in AWS Deep Learning AMI a partire dalla versione v28. Le versioni precedenti di quelle AWS Deep Learning AMI che contengono questi ambienti continueranno a essere disponibili. Tuttavia, verranno forniti aggiornamenti a questi ambienti solo se sono disponibili correzioni di protezione pubblicate dalla comunità open source per questi framework.

Panoramica di ONNX

ONNX (Open Neural Network Exchange) è un formato aperto utilizzato per rappresentare modelli di apprendimento profondo. ONNX è supportato da Amazon Web Services, Microsoft, Facebook

e diversi altri partner. Puoi progettare modelli di apprendimento profondi, eseguirne il training e distribuirli con qualsiasi framework di tua scelta. I modelli ONNX offrono il vantaggio di poter essere spostati facilmente da un framework all'altro.

Questo tutorial mostra come utilizzare l'AMI Deep Learning con Conda con ONNX. Le informazioni in esso contenute ti consentono di eseguire il training di un modello o caricare un modello con training da un framework, esportare tale modello in formato ONNX e quindi importarlo in un altro framework.

Prerequisiti per ONNX

Per utilizzare questo tutorial ONNX, devi avere accesso a un'AMI Deep Learning con Conda versione 12 o successiva. Per ulteriori informazioni su come iniziare a utilizzare un'AMI Deep Learning con Conda, consultaAMI di deep learning di.

> **Important**
>
> Questi esempi utilizzano funzioni che potrebbero richiedere fino a 8 GB di memoria (o più). Assicurati di scegliere un tipo di istanza con memoria sufficiente.

Avvia una sessione terminale con la tua AMI Deep Learning con Conda per iniziare il seguente tutorial.

Converti un modello Apache MXNet (incubazione) in ONNX e caricalo in CNTK

Come esportare un modello da Apache MXNet (incubazione)

Puoi installare la build MXNet più recente in uno o entrambi gli ambienti MXNet Conda sulla tua AMI Deep Learning con Conda.

1.
   - (Opzione per Python 3) – Attivare l'ambiente Python 3 MXNet:

     ```
     $ source activate mxnet_p36
     ```

   - (Opzione per Python 2) – Attivare l'ambiente Python 2 MXNet:

     ```
     $ source activate mxnet_p27
     ```

2. Per gli altri passaggi, si presuppone che sia utilizzato l'ambiente `mxnet_p36`.
3. Scaricare i file del modello.

```
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. Per esportare i file del modello da MXNet in formato ONNX, creare un nuovo file con l'editor di testo e utilizzare il programma seguente in uno script.

```
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)
```

   Se vengono visualizzati alcuni messaggi di avviso, è possibile ignorarli per il momento. Una volta eseguito questo script, il file onnx creato sarà visualizzato nella stessa directory.

5. Ora che si dispone di un file ONNX è possibile provare a eseguire l'inferenza con il seguente esempio:

   - Utilizzo di CNTK per l'inferenza con un modello ONNX

ONNX Tutorials

- Tutorial sulla conversione di un modello MXNet in ONNX per CNTK
- Tutorial sulla conversione di un modello Chainer in ONNX per CNTK
- Tutorial sulla conversione di un modello Chainer in ONNX per MXNet
- PyTorch da ONNX a MXNet Tutorial
- PyTorch da ONNX a CNTK Tutorial

Tutorial sulla conversione di un modello Chainer in ONNX per CNTK

> **Note**
>
> Non includiamo più gli ambienti CNTK, Caffe, Caffe2 e Theano Conda in AWS Deep Learning AMI a partire dalla versione v28. Le versioni precedenti di AWS Deep Learning AMI che contengono questi ambienti continueranno a essere disponibili. Tuttavia, verranno forniti aggiornamenti a questi ambienti solo se sono disponibili correzioni di protezione pubblicate dalla comunità open source per questi framework.

Panoramica di ONNX

ONNX (Open Neural Network Exchange) è un formato aperto utilizzato per rappresentare modelli di apprendimento profondo. ONNX è supportato da Amazon Web Services, Microsoft, Facebook e diversi altri partner. Puoi progettare modelli di apprendimento profondi, eseguirne il training e distribuirli con qualsiasi framework di tua scelta. I modelli ONNX offrono il vantaggio di poter essere spostati facilmente da un framework all'altro.

Questo tutorial mostra come utilizzare l'AMI Deep Learning con Conda con ONNX. Le informazioni in esso contenute ti consentono di eseguire il training di un modello o caricare un modello con training da un framework, esportare tale modello in formato ONNX e quindi importarlo in un altro framework.

Prerequisiti per ONNX

Per utilizzare questo tutorial ONNX, devi avere accesso a un'AMI Deep Learning con Conda versione 12 o successiva. Per ulteriori informazioni su come iniziare a utilizzare un'AMI Deep Learning con Conda, consultaAMI di deep learning di.

> **Important**
>
> Questi esempi utilizzano funzioni che potrebbero richiedere fino a 8 GB di memoria (o più). Assicurati di scegliere un tipo di istanza con memoria sufficiente.

Avvia una sessione terminale con la tua AMI Deep Learning con Conda per iniziare il seguente tutorial.

Converti un modello Chainer in ONNX e caricalo in CNTK

Per prima cosa, attivare l'ambiente Chainer:

```
$ source activate chainer_p36
```

Creare un nuovo file con l'editor di testo e utilizzare il programma seguente in uno script per recuperare un modello dalla serie di modelli Chainer, quindi esportarlo in formato ONNX.

```
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)
```

Una volta eseguito questo script, il file onnx creato sarà visualizzato nella stessa directory.

Ora che si dispone di un file ONNX è possibile provare a eseguire l'inferenza con il seguente esempio:

- Utilizzo di CNTK per l'inferenza con un modello ONNX

ONNX Tutorials

- Tutorial sulla conversione di un modello MXNet in ONNX per CNTK
- Tutorial sulla conversione di un modello Chainer in ONNX per CNTK
- Tutorial sulla conversione di un modello Chainer in ONNX per MXNet
- PyTorch da ONNX a MXNet Tutorial
- PyTorch da ONNX a CNTK Tutorial

Tutorial sulla conversione di un modello Chainer in ONNX per MXNet

Panoramica di ONNX

ONNX (Open Neural Network Exchange) è un formato aperto utilizzato per rappresentare modelli di apprendimento profondo. ONNX è supportato da Amazon Web Services, Microsoft, Facebook e diversi altri partner. Puoi progettare modelli di apprendimento profondi, eseguirne il training e distribuirli con qualsiasi framework di tua scelta. I modelli ONNX offrono il vantaggio di poter essere spostati facilmente da un framework all'altro.

Questo tutorial mostra come utilizzare l'AMI Deep Learning con Conda con ONNX. Le informazioni in esso contenute ti consentono di eseguire il training di un modello o caricare un modello con training da un framework, esportare tale modello in formato ONNX e quindi importarlo in un altro framework.

Prerequisiti per ONNX

Per utilizzare questo tutorial ONNX, devi avere accesso a un'AMI Deep Learning con Conda versione 12 o successiva. Per ulteriori informazioni su come iniziare a utilizzare un'AMI Deep Learning con Conda, consultaAMI di deep learning di.

> **Important**
>
> Questi esempi utilizzano funzioni che potrebbero richiedere fino a 8 GB di memoria (o più). Assicurati di scegliere un tipo di istanza con memoria sufficiente.

Avvia una sessione terminale con la tua AMI Deep Learning con Conda per iniziare il seguente tutorial.

Converti un modello Chainer in ONNX e caricalo in MXNet

Per prima cosa, attivare l'ambiente Chainer:

```
$ source activate chainer_p36
```

Creare un nuovo file con l'editor di testo e utilizzare il programma seguente in uno script per recuperare un modello dalla serie di modelli Chainer, quindi esportarlo in formato ONNX.

```
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)
```

Una volta eseguito questo script, il file onnx creato sarà visualizzato nella stessa directory.

Ora che si dispone di un file ONNX è possibile provare a eseguire l'inferenza con il seguente esempio:

- Usa Apache MXNet (incubazione) per l'inferenza con un modello ONNX

ONNX Tutorials

- Tutorial sulla conversione di un modello MXNet in ONNX per CNTK
- Tutorial sulla conversione di un modello Chainer in ONNX per CNTK
- Tutorial sulla conversione di un modello Chainer in ONNX per MXNet
- PyTorch da ONNX a MXNet Tutorial
- PyTorch da ONNX a CNTK Tutorial

PyTorch da ONNX a CNTK Tutorial

> **Note**
>
> Non includiamo più gli ambienti CNTK, Caffe, Caffe2 e Theano Conda in AWS Deep Learning AMI a partire dalla versione v28. Le versioni precedenti di AWS Deep Learning AMI che contengono questi ambienti continueranno a essere disponibili. Tuttavia, verranno forniti aggiornamenti a questi ambienti solo se sono disponibili correzioni di protezione pubblicate dalla comunità open source per questi framework.

Panoramica di ONNX

ONNX (Open Neural Network Exchange) è un formato aperto utilizzato per rappresentare modelli di apprendimento profondo. ONNX è supportato da Amazon Web Services, Microsoft, Facebook e diversi altri partner. Puoi progettare modelli di apprendimento profondi, eseguirne il training e distribuirli con qualsiasi framework di tua scelta. I modelli ONNX offrono il vantaggio di poter essere spostati facilmente da un framework all'altro.

Questo tutorial mostra come utilizzare l'AMI Deep Learning con Conda con ONNX. Le informazioni in esso contenute ti consentono di eseguire il training di un modello o caricare un modello con training da un framework, esportare tale modello in formato ONNX e quindi importarlo in un altro framework.

Prerequisiti per ONNX

Per utilizzare questo tutorial ONNX, devi avere accesso a un'AMI Deep Learning con Conda versione 12 o successiva. Per ulteriori informazioni su come iniziare a utilizzare un'AMI Deep Learning con Conda, consultaAMI di deep learning di.

> **Important**
>
> Questi esempi utilizzano funzioni che potrebbero richiedere fino a 8 GB di memoria (o più). Assicurati di scegliere un tipo di istanza con memoria sufficiente.

Avvia una sessione terminale con la tua AMI Deep Learning con Conda per iniziare il seguente tutorial.

Converti un PyTorch modello in ONNX, quindi carica il modello in CNTK

Innanzitutto, attiva l'ambiente: PyTorch

```
$ source activate pytorch_p36
```

Crea un nuovo file con il tuo editor di testo e usa il seguente programma in uno script per addestrare un modello fittizio PyTorch, quindi esportalo nel formato ONNX.

```
# 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)
```

Una volta eseguito questo script, il file onnx creato sarà visualizzato nella stessa directory. Passare all'ambiente CNTK Conda per caricare il modello con CNTK.

Attivare quindi l'ambiente CNTK:

```
$ source deactivate
$ source activate cntk_p36
```

Creare un nuovo file con l'editor di testo e utilizzare il programma seguente in uno script per aprire il file in formato ONNX in CNTK.

```
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)
```

Dopo l'esecuzione di questo script, CNTK avrà caricato il modello.

È anche possibile esportare in formato ONNX mediante CNTK aggiungendo quanto segue allo script precedente ed eseguendo quest'ultimo.

```
# Export the model to ONNX via the CNTK export API
z.save("cntk_model.onnx", format=C.ModelFormat.ONNX)
```

ONNX Tutorials

- Tutorial sulla conversione di un modello MXNet in ONNX per CNTK
- Tutorial sulla conversione di un modello Chainer in ONNX per CNTK

- Tutorial sulla conversione di un modello Chainer in ONNX per MXNet
- PyTorch da ONNX a MXNet Tutorial
- PyTorch da ONNX a CNTK Tutorial

PyTorch da ONNX a MXNet Tutorial

Panoramica di ONNX

ONNX (Open Neural Network Exchange) è un formato aperto utilizzato per rappresentare modelli di apprendimento profondo. ONNX è supportato da Amazon Web Services, Microsoft, Facebook e diversi altri partner. Puoi progettare modelli di apprendimento profondi, eseguirne il training e distribuirli con qualsiasi framework di tua scelta. I modelli ONNX offrono il vantaggio di poter essere spostati facilmente da un framework all'altro.

Questo tutorial mostra come utilizzare l'AMI Deep Learning con Conda con ONNX. Le informazioni in esso contenute ti consentono di eseguire il training di un modello o caricare un modello con training da un framework, esportare tale modello in formato ONNX e quindi importarlo in un altro framework.

Prerequisiti per ONNX

Per utilizzare questo tutorial ONNX, devi avere accesso a un'AMI Deep Learning con Conda versione 12 o successiva. Per ulteriori informazioni su come iniziare a utilizzare un'AMI Deep Learning con Conda, consultaAMI di deep learning di.

> **Important**
>
> Questi esempi utilizzano funzioni che potrebbero richiedere fino a 8 GB di memoria (o più). Assicurati di scegliere un tipo di istanza con memoria sufficiente.

Avvia una sessione terminale con la tua AMI Deep Learning con Conda per iniziare il seguente tutorial.

Convertire un PyTorch modello in ONNX, quindi caricare il modello in MXNet

Innanzitutto, attiva l'ambiente: PyTorch

```
$ source activate pytorch_p36
```

Crea un nuovo file con il tuo editor di testo e usa il seguente programma in uno script per addestrare un modello fittizio PyTorch, quindi esportalo nel formato ONNX.

```
# 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!")
```

Una volta eseguito questo script, il file onnx creato sarà visualizzato nella stessa directory. Passare all'ambiente MXNet Conda per caricare il modello con MXNet.

Attivare l'ambiente MXNet:

```
$ source deactivate
$ source activate mxnet_p36
```

Creare un nuovo file con l'editor di testo e utilizzare il programma seguente in uno script per aprire il file in formato ONNX in MXNet.

```
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())
```

Dopo l'esecuzione di questo script, MXNet avrà caricato il modello e stamperà alcune informazioni di base sullo stesso.

ONNX Tutorials

- Tutorial sulla conversione di un modello MXNet in ONNX per CNTK
- Tutorial sulla conversione di un modello Chainer in ONNX per CNTK
- Tutorial sulla conversione di un modello Chainer in ONNX per MXNet
- PyTorch da ONNX a MXNet Tutorial
- PyTorch da ONNX a CNTK Tutorial

Model serving

Di seguito sono riportate le opzioni di model serving installate sull'AMI Deep Learning con Conda. Fai clic su quella desiderata per informazioni su come utilizzarla.

Argomenti

- Model Server for Apache MXNet (MMS)
- TensorFlow Servire
- TorchServe

Model Server for Apache MXNet (MMS)

Model Server for Apache MXNet (MMS) è uno strumento flessibile per servire i modelli di apprendimento profondo che sono stati esportati da Apache MXNet (incubating) o esportati in un formato di modello Open Neural Network Exchange (ONNX). MMS è preinstallato con la DLAMI con Conda. Questo tutorial per MMS illustra come servire un modello di classificazione delle immagini.

Argomenti

Servire un modello di classificazione delle immagini su MMS

Questo tutorial illustra come servire un modello di classificazione delle immagini con MMS. Il modello è disponibile sulla pagina Model Zoo e viene scaricato automaticamente all'avvio di MMS. Quando il server è in esecuzione, ascolta le richieste di stima. Quando carichi un'immagine, in questo caso un'immagine di un gattino, il server restituisce una stima delle 5 migliori classi corrispondenti tra le 1.000 utilizzate per il training del modello. Ulteriori informazioni sui modelli, sul relativo training e su come testarli sono disponibili nella pagina Model Zoo.

Per servire un esempio di modello di classificazione delle immagini su MMS

1. Connettiti a un'istanza Amazon Elastic Compute Cloud (Amazon EC2) dell'AMI Deep Learning con Conda.
2. Attivare un ambiente MXNet:
   - Per MXNet e Keras 2 su Python 3 con CUDA 9.0 e MKL-DNN, eseguire questo comando:

     ```
     $ source activate mxnet_p36
     ```

   - Per MXNet e Keras 2 su Python 2 con CUDA 9.0 e MKL-DNN, eseguire questo comando:

     ```
     $ source activate mxnet_p27
     ```

3. Eseguire MMS con il comando seguente. L'aggiunta di `> /dev/null` silenzierà l'output di log mentre si eseguono altri test.

   ```
   $ mxnet-model-server --start > /dev/null
   ```

MMS è ora in esecuzione sull'host e ascolta le richieste di inferenza.

4. Quindi, utilizzare un comando curl per amministrare gli endpoint di gestione di MMS e indicare quale modello servire.

```
$ 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 deve sapere il numero di lavoratori che si desidera utilizzare. Per questo test è possibile provare 3.

```
$ curl -v -X PUT "http://localhost:8081/models/squeezenet_v1.1?min_worker=3"
```

6. Scaricare un'immagine di un gattino e inviarla all'endpoint di stima MMS:

```
$ 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
```

L'endpoint di stima restituisce una stima in formato JSON simile alle cinque migliori stime seguenti, dove l'immagine ha il 94% di probabilità di contenere un gatto egiziano e il 5,5% di probabilità di contenere una lince o un puma:

```
{
 "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. Testare altre immagini oppure, se il testing è completo, arrestare il server:

```
$ mxnet-model-server --stop
```

Questo tutorial è incentrato sul model serving di base. MMS supporta l'uso dell'inferenza elastica con model serving. Per ulteriori informazioni, consulta Model Serving con Amazon Elastic Inference

Quando sei pronto per saperne di più su altre funzionalità MMS, consulta la documentazione MMS su. GitHub

Altri esempi

MMS offre una serie di esempi che è possibile eseguire sul proprio DLAMI. È possibile visualizzarli sul repository di progetto MMS.

Ulteriori informazioni

Per ulteriori informazioni sugli MMS, tra cui come configurare gli MMS con Docker o per sfruttare le funzionalità MMS più recenti, dai un'occhiata alla pagina del progetto MMS. GitHub

TensorFlow Servire

TensorFlow Serving è un sistema di servizio flessibile e ad alte prestazioni per modelli di apprendimento automatico.

`tensorflow-serving-api`È preinstallato con Deep Learning AMI con Conda! Troverai esempi di script per eseguire il training di un modello MNIST, esportarlo e servirlo in `~/examples/tensorflow-serving/`.

Per eseguire uno di questi esempi, connettiti prima alla tua AMI Deep Learning con Conda e attiva l' TensorFlow ambiente.

```
$ source activate tensorflow_p37
```

Ora accedi alla cartella con gli esempi di script.

```
$ cd ~/examples/tensorflow-serving/
```

Distribuzione di un modello Inception preformato

Di seguito è riportato un esempio che può essere provato per distribuire diversi modelli come Inception. Come regola generale, è necessario disporre di un modello utilizzabile e degli script client già scaricati sul DLAMI.

Distribuzione e test dell'inferenza con un modello Inception

1. Scarica il modello.

   ```
   $ curl -O https://s3-us-west-2.amazonaws.com/tf-test-models/INCEPTION.zip
   ```

2. Estrai il modello.

   ```
   $ unzip INCEPTION.zip
   ```

3. Scaricare un'immagine di un husky.

   ```
   $ curl -O https://upload.wikimedia.org/wikipedia/commons/b/b5/Siberian_Husky_bi-
   eyed_Flickr.jpg
   ```

4. Avvia il server. Per Amazon Linux è necessario modificare la directory utilizzata per `model_base_path` da `/home/ubuntu` a `/home/ec2-user`.

   ```
   $ tensorflow_model_server --model_name=INCEPTION --model_base_path=/home/ubuntu/
   examples/tensorflow-serving/INCEPTION/INCEPTION --port=9000
   ```

5. Con il server in esecuzione in primo piano, è necessario avviare un'altra sessione di terminale per continuare. Apri un nuovo terminale e attiva TensorFlow con. `source activate tensorflow_p37` Quindi utilizza l'editor di testo preferito per creare uno script che ha i seguenti contenuti. Denominalo `inception_client.py`. Lo script prenderà un nome file immagine come parametro e otterrà un risultato di previsione dal modello preformato.

   ```
   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. Ora esegui lo script fornendo la posizione e la porta del server e il nome della foto dell'husky come parametri.

```
$ python3 inception_client.py --server=localhost:9000 --image Siberian_Husky_bi-
eyed_Flickr.jpg
```

Training e distribuzione di un modello MNIST

Per questo tutorial, esporteremo un modello, quindi lo serviremo con l'applicazione `tensorflow_model_server`. Infine, testeremo il server di modelli con un esempio di script client.

Esegui lo script che utilizzerai per il training e l'esportazione del modello MNIST. Come unico argomento dello script, è necessario fornire il percorso di una cartella in cui salvare il modello. Per il momento, possiamo metterla in `mnist_model`. Lo script creerà la cartella.

```
$ python mnist_saved_model.py /tmp/mnist_model
```

L'esecuzione dello script può durare alcuni minuti. Al termine del training e dell'esportazione del modello, viene visualizzato quanto segue:

```
Done training!
Exporting trained model to mnist_model/1
Done exporting!
```

L'operazione successiva consiste nell'eseguire `tensorflow_model_server` per servire il modello esportato.

```
$ tensorflow_model_server --port=9000 --model_name=mnist --model_base_path=/tmp/
mnist_model
```

Uno script client viene fornito per testare il server.

Per eseguire il test, devi aprire una nuova finestra del terminale.

```
$ python mnist_client.py --num_tests=1000 --server=localhost:9000
```

Ulteriori funzionalità ed esempi

Se sei interessato a saperne di più su TensorFlow Serving, consulta il TensorFlow sito web.

Puoi anche usare TensorFlow Serving with Amazon Elastic Inference. Consulta la guida su come utilizzare Elastic Inference with TensorFlow Serving per maggiori informazioni.

TorchServe

TorchServe è uno strumento flessibile per servire modelli di deep learning che sono stati esportati da. PyTorch TorchServe viene preinstallato con l'AMI Deep Learning con Conda a partire dalla v34.

Per ulteriori informazioni sull'utilizzo TorchServe, consulta Model Server for Documentation. PyTorch

Argomenti

Offri un modello di classificazione delle immagini su TorchServe

Questo tutorial mostra come utilizzare un modello di classificazione delle immagini con TorchServe. Utilizza un modello DenseNet -161 fornito da PyTorch. Una volta che il server è in esecuzione, ascolta le richieste di previsione. Quando carichi un'immagine, in questo caso l'immagine di un gattino, il server restituisce una previsione delle 5 migliori classi corrispondenti tra le classi su cui è stato addestrato il modello.

Per fornire un esempio di modello di classificazione delle immagini su TorchServe

1. Connettiti a un'istanza Amazon Elastic Compute Cloud (Amazon EC2) con Deep Learning AMI con Conda v34 o versione successiva.
2. Attiva l'ambiente. `pytorch_latest_p36`

   ```
   source activate pytorch_latest_p36
   ```

3. Clona il TorchServe repository, quindi crea una directory per archiviare i tuoi modelli.

   ```
   git clone https://github.com/pytorch/serve.git
   mkdir model_store
   ```

4. Archivia il modello utilizzando il model archiver. Il `extra-files` parametro utilizza un file del `TorchServe` repository, quindi aggiorna il percorso se necessario. Per ulteriori informazioni sul model archiver, vedere 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. Esegui TorchServe per avviare un endpoint. L'aggiunta > `/dev/null` disattiva l'output del registro.

```
torchserve --start --ncs --model-store model_store --models densenet161.mar > /dev/
null
```

6. Scaricate l'immagine di un gattino e inviatela all'endpoint TorchServe previsto:

```
curl -O https://s3.amazonaws.com/model-server/inputs/kitten.jpg
curl http://127.0.0.1:8080/predictions/densenet161 -T kitten.jpg
```

   L'endpoint di previsione restituisce una previsione in JSON simile alle prime cinque previsioni seguenti, in cui l'immagine ha una probabilità del 47% di contenere un gatto egiziano, seguita da una probabilità del 46% che abbia un gatto soriano.

```
{
 "tiger_cat": 0.46933576464653015,
 "tabby": 0.463387668132782,
 "Egyptian_cat": 0.0645613968372345,
 "lynx": 0.0012828196631744504,
 "plastic_bag": 0.00023323058849200606
}
```

7. Al termine del test, ferma il server:

```
torchserve --stop
```

Altri esempi

TorchServe contiene una serie di esempi che è possibile eseguire sulla propria istanza DLAMI. È possibile visualizzarli nella pagina degli esempi del repository TorchServe del progetto.

Maggiori informazioni

Per ulteriore TorchServe documentazione, incluso come configurare Docker e TorchServe le TorchServe funzionalità più recenti, consulta la pagina del TorchServe progetto su GitHub.

Aggiornamento del tuo DLAMI

Qui troverai informazioni sull'aggiornamento del tuo DLAMI e suggerimenti sull'aggiornamento del software sul tuo DLAMI.

Argomenti

Aggiornamento a una nuova versione di DLAMI

Le immagini di sistema di DLAMI vengono aggiornate regolarmente per sfruttare le nuove versioni del framework di deep learning, gli aggiornamenti di CUDA e altri software e l'ottimizzazione delle prestazioni. Se usi un DLAMI da qualche tempo e desideri sfruttare un aggiornamento, dovresti lanciare una nuova istanza. Devi inoltre trasferire manualmente set di dati, checkpoint o altri dati importanti. Puoi invece utilizzare Amazon EBS per conservare i tuoi dati e collegarli a un nuovo DLAMI. In questo modo, puoi eseguire regolarmente l'upgrade riducendo al minimo il tempo necessario per la transizione dei dati.

> **Note**
>
> Quando colleghi e sposti volumi Amazon EBS tra DLAMI, devi avere sia i DLAMI che il nuovo volume nella stessa zona di disponibilità.

1. Usa la console Amazon EC2 per creare un nuovo volume Amazon EBS. Per indicazioni dettagliate, consulta Creazione di un volume Amazon EBS.
2. Collegare un volume Amazon EBS all'unità DLAMI esistente Per istruzioni dettagliate, consulta Allegare un volume Amazon EBS.
3. Trasferire i dati, come set di dati, checkpoint e file di configurazione.
4. Avvia un DLAMI. Per istruzioni dettagliate, consulta Avvio e configurazione di un DLAMI.
5. Scollega il volume Amazon EBS dal tuo vecchio DLAMI. Per indicazioni dettagliate, consulta Scollegare un volume Amazon EBS.
6. Collegare il volume Amazon EBS all'unità DLAMI Seguire le istruzioni dal punto 2 per collegare il volume.

7. Dopo aver verificato che i dati sono disponibili sul nuovo DLAMI, interrompi e interrompi il vecchio DLAMI. Per istruzioni di pulizia dettagliate, consulta Eliminare.

Suggerimenti per gli aggiornamenti software

Di tanto in tanto, potresti voler aggiornare manualmente il software sul tuo DLAMI. In generale, è consigliabile utilizzare `pip` per aggiornare i pacchetti Python. Dovresti anche usarlo`pip` per aggiornare i pacchetti all'interno di un ambiente Conda sull'AMI Deep Learning con Conda. Per istruzioni relative all'aggiornamento e all'installazione, visita il sito Web del framework o del software in questione.

> **Note**
>
> Non possiamo garantire che l'aggiornamento del pacchetto avrà successo. Il tentativo di aggiornare un pacchetto in un ambiente con dipendenze incompatibili può causare un errore. In tal caso, è necessario contattare il manutentore della libreria per vedere se è possibile aggiornare le dipendenze dei pacchetti. In alternativa, puoi provare a modificare l'ambiente in modo tale da consentirne l'aggiornamento. Tuttavia, questa modifica comporterà probabilmente la rimozione o l'aggiornamento dei pacchetti esistenti, il che significa che non possiamo più garantire la stabilità di questo ambiente.

Se sei interessato a eseguire l'ultimo ramo principale di un particolare pacchetto, attiva l'ambiente appropriato, quindi aggiungilo`--pre` alla fine del`pip install --upgrade` comando. Ad esempio:

```
source activate mxnet_p36
pip install --upgrade mxnet --pre
```

VieneAWS Deep Learning AMI fornito con molti ambienti Conda e molti pacchetti preinstallati. A causa del numero di pacchetti preinstallati, è difficile trovare un set di pacchetti la cui compatibilità sia garantita. Potresti visualizzare un avviso «L'ambiente non è coerente, controlla attentamente il piano del pacchetto». DLAMI garantisce che tutti gli ambienti forniti da DLAMI siano corretti, ma non può garantire che i pacchetti installati dall'utente funzionino correttamente.

Sicurezza in AWS Deep Learning AMI

La sicurezza del cloud AWS è la massima priorità. In qualità di AWS cliente, puoi beneficiare di un data center e di un'architettura di rete progettati per soddisfare i requisiti delle organizzazioni più sensibili alla sicurezza.

La sicurezza è una responsabilità condivisa tra AWS te e te. Il modello di responsabilità condivisa descrive questo aspetto come sicurezza del cloud e sicurezza nel cloud:

- Sicurezza del cloud: AWS è responsabile della protezione dell'infrastruttura che gestisce AWS i servizi nel AWS cloud. AWS ti fornisce anche servizi che puoi utilizzare in modo sicuro. I revisori esterni testano e verificano regolarmente l'efficacia della nostra sicurezza nell'ambito dei AWS Programmi di AWS conformità dei Programmi di conformità dei di . Per ulteriori informazioni sui programmi di conformità che si applicano a DLAMI, vedere AWS Servizi nell'ambito del programma di conformitàAWS Servizi nell'ambito del programma .
- Sicurezza nel cloud: la responsabilità dell'utente è determinata dal AWS servizio utilizzato. Sei anche responsabile di altri fattori, tra cui la riservatezza dei dati, i requisiti della tua azienda e le leggi e normative vigenti.

Questa documentazione aiuta a capire come applicare il modello di responsabilità condivisa quando si utilizza DLAMI. I seguenti argomenti mostrano come configurare DLAMI per soddisfare gli obiettivi di sicurezza e conformità. Scopri anche come utilizzare altri AWS servizi che ti aiutano a monitorare e proteggere le tue risorse DLAMI.

Per ulteriori informazioni, consulta Sicurezza in Amazon EC2.

Argomenti

Protezione dei dati in AWS Deep Learning AMI

Il modello di responsabilità AWS condivisa modello di di si applica alla protezione dei dati nell'AMI di AWS Deep Learning. Come descritto in questo modello, AWS è responsabile della protezione dell'infrastruttura globale che gestisce tutti i Cloud AWS. L'utente è responsabile del controllo dei contenuti ospitati su questa infrastruttura. Inoltre, sei responsabile della configurazione della protezione e delle attività di gestione per i Servizi AWS che utilizzi. Per ulteriori informazioni sulla privacy dei dati, vedi le Domande frequenti sulla privacy dei dati. Per informazioni sulla protezione dei dati in Europa, consulta il post del blog relativo al Modello di responsabilità condivisa AWS e GDPR nel Blog sulla sicurezza AWS .

Ai fini della protezione dei dati, consigliamo di proteggere Account AWS le credenziali e configurare i singoli utenti con AWS IAM Identity Center or AWS Identity and Access Management (IAM). In tal modo, a ogni utente verranno assegnate solo le autorizzazioni necessarie per svolgere i suoi compiti. Ti suggeriamo, inoltre, di proteggere i dati nei seguenti modi:

- Utilizza l'autenticazione a più fattori (MFA) con ogni account.
- Usa SSL/TLS per comunicare con le risorse. AWS È richiesto TLS 1.2 ed è consigliato TLS 1.3.
- Configura l'API e la registrazione delle attività degli utenti con. AWS CloudTrail
- Utilizza soluzioni di AWS crittografia, insieme a tutti i controlli di sicurezza predefiniti all'interno Servizi AWS.
- Utilizza i servizi di sicurezza gestiti avanzati, come Amazon Macie, che aiutano a individuare e proteggere i dati sensibili archiviati in Amazon S3.
- Se hai bisogno di moduli crittografici convalidati FIPS 140-2 per l'accesso AWS tramite un'interfaccia a riga di comando o un'API, utilizza un endpoint FIPS. Per ulteriori informazioni sugli endpoint FIPS disponibili, consulta il Federal Information Processing Standard (FIPS) 140-2.

Ti consigliamo vivamente di non inserire mai informazioni riservate o sensibili, ad esempio gli indirizzi e-mail dei clienti, nei tag o nei campi di testo in formato libero, ad esempio nel campo Nome. Ciò include quando si lavora con DLAMI o altro Servizi AWS utilizzando la console, l'API o AWS gli AWS CLI SDK. I dati inseriti nei tag o nei campi di testo in formato libero utilizzati per i nomi possono essere utilizzati per i la fatturazione o i log di diagnostica. Quando fornisci un URL a un server esterno, ti suggeriamo vivamente di non includere informazioni sulle credenziali nell'URL per convalidare la tua richiesta al server.

Identity and Access Management in AWS Deep Learning AMI

AWS Identity and Access Management (IAM) è un software Servizio AWS che aiuta un amministratore a controllare in modo sicuro l'accesso alle risorse. AWS Gli amministratori IAM controllano chi può essere autenticato (effettuato l'accesso) e autorizzato (dispone delle autorizzazioni) a utilizzare le risorse DLAMI. IAM è uno strumento Servizio AWS che puoi utilizzare senza costi aggiuntivi.

Per ulteriori informazioni su Identity and Access Management, consulta Identity and Access Management per Amazon EC2.

Argomenti

Autenticazione con identità

L'autenticazione è il modo in cui accedi AWS utilizzando le tue credenziali di identità. Devi essere autenticato (aver effettuato l' Utente root dell'account AWS accesso AWS) come utente IAM o assumendo un ruolo IAM.

Puoi accedere AWS come identità federata utilizzando le credenziali fornite tramite una fonte di identità. AWS IAM Identity Center Gli utenti (IAM Identity Center), l'autenticazione Single Sign-On della tua azienda e le tue credenziali di Google o Facebook sono esempi di identità federate. Se accedi come identità federata, l'amministratore ha configurato in precedenza la federazione delle identità utilizzando i ruoli IAM. Quando accedi AWS utilizzando la federazione, assumi indirettamente un ruolo.

A seconda del tipo di utente, puoi accedere al AWS Management Console o al portale di AWS accesso. Per ulteriori informazioni sull'accesso a AWS, vedi Come accedere al tuo Account AWS nella Guida per l'Accedi ad AWS utente.

Se accedi a AWS livello di codice, AWS fornisce un kit di sviluppo software (SDK) e un'interfaccia a riga di comando (CLI) per firmare crittograficamente le tue richieste utilizzando le tue credenziali. Se non utilizzi AWS strumenti, devi firmare tu stesso le richieste. Per ulteriori informazioni sull'utilizzo del metodo consigliato per firmare autonomamente le richieste, consulta Signing AWS API request nella IAM User Guide.

A prescindere dal metodo di autenticazione utilizzato, potrebbe essere necessario specificare ulteriori informazioni sulla sicurezza. Ad esempio, ti AWS consiglia di utilizzare l'autenticazione a più fattori (MFA) per aumentare la sicurezza del tuo account. Per ulteriori informazioni, consulta Autenticazione a più fattori nella Guida per l'utente di AWS IAM Identity Center e Utilizzo dell'autenticazione a più fattori (MFA) in AWS nella Guida per l'utente di IAM.

Account AWS utente root

Quando si crea un account Account AWS, si inizia con un'identità di accesso che ha accesso completo a tutte Servizi AWS le risorse dell'account. Questa identità è denominata utente Account AWS root ed è accessibile effettuando l'accesso con l'indirizzo e-mail e la password utilizzati per creare l'account. Si consiglia vivamente di non utilizzare l'utente root per le attività quotidiane. Conserva le credenziali dell'utente root e utilizzarle per eseguire le operazioni che solo l'utente root può eseguire. Per un elenco completo delle attività che richiedono l'accesso come utente root, consulta la sezione Attività che richiedono le credenziali dell'utente root nella Guida per l'utente di IAM.

Utenti e gruppi IAM

Un utente IAM è un'identità interna Account AWS che dispone di autorizzazioni specifiche per una singola persona o applicazione. Ove possibile, consigliamo di fare affidamento a credenziali temporanee invece di creare utenti IAM con credenziali a lungo termine come le password e le chiavi di accesso. Tuttavia, per casi d'uso specifici che richiedono credenziali a lungo termine con utenti IAM, si consiglia di ruotare le chiavi di accesso. Per ulteriori informazioni, consulta la pagina Rotazione periodica delle chiavi di accesso per casi d'uso che richiedono credenziali a lungo termine nella Guida per l'utente di IAM.

Un gruppo IAM è un'identità che specifica un insieme di utenti IAM. Non è possibile eseguire l'accesso come gruppo. È possibile utilizzare gruppi per specificare le autorizzazioni per più utenti alla volta. I gruppi semplificano la gestione delle autorizzazioni per set di utenti di grandi dimensioni. Ad esempio, è possibile avere un gruppo denominato Amministratori IAM e concedere a tale gruppo le autorizzazioni per amministrare le risorse IAM.

Gli utenti sono diversi dai ruoli. Un utente è associato in modo univoco a una persona o un'applicazione, mentre un ruolo è destinato a essere assunto da chiunque ne abbia bisogno. Gli utenti dispongono di credenziali a lungo termine permanenti, mentre i ruoli forniscono credenziali temporanee. Per ulteriori informazioni, consulta Quando creare un utente IAM (invece di un ruolo) nella Guida per l'utente di IAM.

Ruoli IAM

Un ruolo IAM è un'identità interna all'utente Account AWS che dispone di autorizzazioni specifiche. È simile a un utente IAM, ma non è associato a una persona specifica. Puoi assumere temporaneamente un ruolo IAM in AWS Management Console cambiando ruolo. Puoi assumere un ruolo chiamando un'operazione AWS CLI o AWS API o utilizzando un URL personalizzato. Per ulteriori informazioni sui metodi per l'utilizzo dei ruoli, consulta Utilizzo di ruoli IAM nella Guida per l'utente di IAM.

I ruoli IAM con credenziali temporanee sono utili nelle seguenti situazioni:

- Accesso utente federato: per assegnare le autorizzazioni a una identità federata, è possibile creare un ruolo e definire le autorizzazioni per il ruolo. Quando un'identità federata viene autenticata, l'identità viene associata al ruolo e ottiene le autorizzazioni da esso definite. Per ulteriori informazioni sulla federazione dei ruoli, consulta Creazione di un ruolo per un provider di identità di terza parte nella Guida per l'utente di IAM. Se utilizzi IAM Identity Center, configura un set di autorizzazioni. IAM Identity Center mette in correlazione il set di autorizzazioni con un ruolo in IAM per controllare a cosa possono accedere le identità dopo l'autenticazione. Per informazioni sui set di autorizzazioni, consulta Set di autorizzazioni nella Guida per l'utente di AWS IAM Identity Center .
- Autorizzazioni utente IAM temporanee: un utente IAM o un ruolo può assumere un ruolo IAM per ottenere temporaneamente autorizzazioni diverse per un'attività specifica.
- Accesso multi-account: è possibile utilizzare un ruolo IAM per permettere a un utente (un principale affidabile) con un account diverso di accedere alle risorse nell'account. I ruoli sono lo strumento principale per concedere l'accesso multi-account. Tuttavia, con alcuni Servizi AWS, è possibile allegare una policy direttamente a una risorsa (anziché utilizzare un ruolo come proxy). Per informazioni sulle differenze tra ruoli e policy basate su risorse per l'accesso multi-account, consulta Differenza tra i ruoli IAM e le policy basate su risorse nella Guida per l'utente di IAM.
- Accesso a più servizi: alcuni Servizi AWS utilizzano le funzionalità di altri Servizi AWS. Ad esempio, quando effettui una chiamata in un servizio, è comune che tale servizio esegua applicazioni in Amazon EC2 o archivi oggetti in Amazon S3. Un servizio può eseguire questa operazione utilizzando le autorizzazioni dell'entità chiamante, utilizzando un ruolo di servizio o utilizzando un ruolo collegato al servizio.
  - Sessioni di accesso diretto (FAS): quando utilizzi un utente o un ruolo IAM per eseguire azioni AWS, sei considerato un preside. Quando si utilizzano alcuni servizi, è possibile eseguire un'operazione che attiva un'altra azione in un servizio diverso. FAS utilizza le autorizzazioni del principale che chiama an Servizio AWS, combinate con la richiesta Servizio AWS per

effettuare richieste ai servizi downstream. Le richieste FAS vengono effettuate solo quando un servizio riceve una richiesta che richiede interazioni con altri Servizi AWS o risorse per essere completata. In questo caso è necessario disporre delle autorizzazioni per eseguire entrambe le azioni. Per i dettagli delle policy relative alle richieste FAS, consulta la pagina Forward access sessions.

- Ruolo di servizio: un ruolo di servizio è un ruolo IAM che un servizio assume per eseguire azioni per tuo conto. Un amministratore IAM può creare, modificare ed eliminare un ruolo di servizio dall'interno di IAM. Per ulteriori informazioni, consulta la sezione Creazione di un ruolo per delegare le autorizzazioni a un Servizio AWS nella Guida per l'utente di IAM.
- Ruolo collegato al servizio: un ruolo collegato al servizio è un tipo di ruolo di servizio collegato a un. Servizio AWS Il servizio può assumere il ruolo per eseguire un'azione per tuo conto. I ruoli collegati al servizio vengono visualizzati nel tuo account Account AWS e sono di proprietà del servizio. Un amministratore IAM può visualizzare le autorizzazioni per i ruoli collegati ai servizi, ma non modificarle.

- Applicazioni in esecuzione su Amazon EC2: puoi utilizzare un ruolo IAM per gestire le credenziali temporanee per le applicazioni in esecuzione su un'istanza EC2 e che AWS CLI effettuano richieste API. AWS Ciò è preferibile all'archiviazione delle chiavi di accesso nell'istanza EC2. Per assegnare un AWS ruolo a un'istanza EC2 e renderlo disponibile per tutte le sue applicazioni, crei un profilo di istanza collegato all'istanza. Un profilo dell'istanza contiene il ruolo e consente ai programmi in esecuzione sull'istanza EC2 di ottenere le credenziali temporanee. Per ulteriori informazioni, consulta Utilizzo di un ruolo IAM per concedere autorizzazioni ad applicazioni in esecuzione su istanze di Amazon EC2 nella Guida per l'utente di IAM.

Per informazioni sull'utilizzo dei ruoli IAM, consulta Quando creare un ruolo IAM (invece di un utente) nella Guida per l'utente di IAM.

Gestione dell'accesso tramite policy

Puoi controllare l'accesso AWS creando policy e collegandole a AWS identità o risorse. Una policy è un oggetto AWS che, se associato a un'identità o a una risorsa, ne definisce le autorizzazioni. AWS valuta queste politiche quando un principale (utente, utente root o sessione di ruolo) effettua una richiesta. Le autorizzazioni nelle policy determinano l'approvazione o il rifiuto della richiesta. La maggior parte delle politiche viene archiviata AWS come documenti JSON. Per ulteriori informazioni sulla struttura e sui contenuti dei documenti delle policy JSON, consulta Panoramica delle policy JSON nella Guida per l'utente di IAM.

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale principale può eseguire azioni su quali risorse e in quali condizioni.

Per impostazione predefinita, utenti e ruoli non dispongono di autorizzazioni. Per concedere agli utenti l'autorizzazione a eseguire azioni sulle risorse di cui hanno bisogno, un amministratore IAM può creare policy IAM. Successivamente l'amministratore può aggiungere le policy IAM ai ruoli e gli utenti possono assumere i ruoli.

Le policy IAM definiscono le autorizzazioni relative a un'azione, a prescindere dal metodo utilizzato per eseguirla. Ad esempio, supponiamo di disporre di una policy che consente l'azione `iam:GetRole`. Un utente con tale policy può ottenere informazioni sul ruolo dall' AWS Management Console AWS CLI, dall'o dall' AWS API.

Policy basate sulle identità

Le policy basate su identità sono documenti di policy di autorizzazione JSON che è possibile allegare a un'identità (utente, gruppo di utenti o ruoli IAM). Tali policy definiscono le azioni che utenti e ruoli possono eseguire, su quali risorse e in quali condizioni. Per informazioni su come creare una policy basata su identità, consulta Creazione di policy IAM nella Guida per l'utente di IAM.

Le policy basate su identità possono essere ulteriormente classificate come policy inline o policy gestite. Le policy inline sono integrate direttamente in un singolo utente, gruppo o ruolo. Le politiche gestite sono politiche autonome che puoi allegare a più utenti, gruppi e ruoli nel tuo Account AWS. Le politiche gestite includono politiche AWS gestite e politiche gestite dai clienti. Per informazioni su come scegliere tra una policy gestita o una policy inline, consulta Scelta fra policy gestite e policy inline nella Guida per l'utente di IAM.

Policy basate su risorse

Le policy basate su risorse sono documenti di policy JSON che è possibile collegare a una risorsa. Gli esempi più comuni di policy basate su risorse sono le policy di attendibilità dei ruoli IAM e le policy dei bucket Amazon S3. Nei servizi che supportano policy basate sulle risorse, gli amministratori dei servizi possono utilizzarle per controllare l'accesso a una risorsa specifica. Quando è collegata a una risorsa, una policy definisce le azioni che un principale può eseguire su tale risorsa e a quali condizioni. È necessario specificare un principale in una policy basata sulle risorse. I principali possono includere account, utenti, ruoli, utenti federati o. Servizi AWS

Le policy basate sulle risorse sono policy inline che si trovano in tale servizio. Non puoi utilizzare le policy AWS gestite di IAM in una policy basata sulle risorse.

Liste di controllo degli accessi (ACL)

Le liste di controllo degli accessi (ACL) controllano quali principali (membri, utenti o ruoli dell'account) hanno le autorizzazioni per accedere a una risorsa. Le ACL sono simili alle policy basate su risorse, sebbene non utilizzino il formato del documento di policy JSON.

Amazon S3 e Amazon VPC sono esempi di servizi che supportano gli ACL. AWS WAF Per maggiori informazioni sulle ACL, consulta Panoramica delle liste di controllo degli accessi (ACL) nella Guida per gli sviluppatori di Amazon Simple Storage Service.

Altri tipi di policy

AWS supporta tipi di policy aggiuntivi e meno comuni. Questi tipi di policy possono impostare il numero massimo di autorizzazioni concesse dai tipi di policy più comuni.

- Limiti delle autorizzazioni: un limite delle autorizzazioni è una funzione avanzata nella quale si imposta il numero massimo di autorizzazioni che una policy basata su identità può concedere a un'entità IAM (utente o ruolo IAM). È possibile impostare un limite delle autorizzazioni per un'entità. Le autorizzazioni risultanti sono l'intersezione delle policy basate su identità dell'entità e i relativi limiti delle autorizzazioni. Le policy basate su risorse che specificano l'utente o il ruolo nel campo `Principal` sono condizionate dal limite delle autorizzazioni. Un rifiuto esplicito in una qualsiasi di queste policy sostituisce l'autorizzazione. Per ulteriori informazioni sui limiti delle autorizzazioni, consulta Limiti delle autorizzazioni per le entità IAM nella Guida per l'utente di IAM.
- Politiche di controllo dei servizi (SCP): le SCP sono politiche JSON che specificano le autorizzazioni massime per un'organizzazione o un'unità organizzativa (OU) in. AWS Organizations AWS Organizations è un servizio per il raggruppamento e la gestione centralizzata di più Account AWS di proprietà dell'azienda. Se abiliti tutte le funzionalità in un'organizzazione, puoi applicare le policy di controllo dei servizi (SCP) a uno o tutti i tuoi account. L'SCP limita le autorizzazioni per le entità presenti negli account dei membri, inclusa ciascuna. Utente root dell'account AWS Per ulteriori informazioni su organizzazioni e policy SCP, consulta la pagina sulle Policy di controllo dei servizi nella Guida per l'utente di AWS Organizations .
- Policy di sessione: le policy di sessione sono policy avanzate che vengono trasmesse come parametro quando si crea in modo programmatico una sessione temporanea per un ruolo o un utente federato. Le autorizzazioni della sessione risultante sono l'intersezione delle policy basate su identità del ruolo o dell'utente e le policy di sessione. Le autorizzazioni possono anche provenire da una policy basata su risorse. Un rifiuto esplicito in una qualsiasi di queste policy sostituisce l'autorizzazione. Per ulteriori informazioni, consulta Policy di sessione nella Guida per l'utente di IAM.

Più tipi di policy

Quando più tipi di policy si applicano a una richiesta, le autorizzazioni risultanti sono più complicate da comprendere. Per scoprire come si AWS determina se consentire una richiesta quando sono coinvolti più tipi di policy, consulta Logica di valutazione delle policy nella IAM User Guide.

IAM con Amazon EMR

Puoi usarlo AWS Identity and Access Management con Amazon EMR per definire utenti, AWS risorse, gruppi, ruoli e policy. Puoi anche controllare a quali AWS servizi possono accedere questi utenti e ruoli.

Per ulteriori informazioni sull'utilizzo di IAM con Amazon EMR, consulta AWS Identity and Access Management for Amazon EMR.

Registrazione e monitoraggio AWS Deep Learning AMI

L' AWS Deep Learning AMI istanza include diversi strumenti di monitoraggio della GPU, tra cui un'utilità che riporta le statistiche sull'utilizzo della GPU ad Amazon. CloudWatch Per ulteriori informazioni, consulta la sezione Monitoraggio e ottimizzazione GPU e Monitoraggio di Amazon EC2.

Monitoraggio dell'utilizzo

Le seguenti distribuzioni di sistemi AWS Deep Learning AMI operativi includono codice che consente di AWS raccogliere informazioni sul tipo di istanza, l'ID dell'istanza, il tipo DLAMI e il sistema operativo. Nessuna informazione sui comandi utilizzati all'interno del DLAMI viene raccolta o conservata. Non vengono raccolte o conservate altre informazioni sul DLAMI.

- Ubuntu 16.04
- Ubuntu 18.04
- Ubuntu 20.04
- Amazon Linux 2

Per disattivare il tracciamento dell'utilizzo per il tuo DLAMI, aggiungi un tag all'istanza Amazon EC2 durante l'avvio. Il tag deve utilizzare la chiave `OPT_OUT_TRACKING` con il valore associato impostato su. `true` Per ulteriori informazioni, consulta Etichettare le risorse Amazon EC2.

Convalida della conformità per AWS Deep Learning AMI

I revisori esterni valutano la sicurezza e la conformità nell' AWS Deep Learning AMI ambito di più programmi di AWS conformità. Per informazioni sui programmi di conformità supportati, consulta Convalida della conformità per Amazon EC2.

Per un elenco dei AWS servizi che rientrano nell'ambito di specifici programmi di conformità, vedere AWS Servizi nell'ambito del programma di conformitàAWS . Per informazioni generali, vedere Programmi di AWS conformità Programmi di di .

È possibile scaricare report di audit di terze parti utilizzando AWS Artifact. Per ulteriori informazioni, consulta Scaricamento dei report in AWS Artifact Scaricamento dei . AWS

La responsabilità di conformità dell'utente nell'utilizzo di DLAMI è determinata dalla sensibilità dei dati, dagli obiettivi di conformità dell'azienda e dalle leggi e dai regolamenti applicabili. AWS fornisce le seguenti risorse per contribuire alla conformità:

- Security and Compliance Quick Start Guides (Guide Quick Start Sicurezza e compliance): queste guide alla distribuzione illustrano considerazioni relative all'architettura e forniscono procedure per la distribuzione di ambienti di base incentrati sulla sicurezza e sulla conformità su AWS.
- AWS Risorse per la per la conformità: questa raccolta di cartelle di lavoro e guide potrebbe riguardare il settore e la località in cui operi.
- Valutazione delle risorse con le regole nella Guida per gli AWS Config sviluppatori: il AWS Config servizio valuta la conformità delle configurazioni delle risorse alle pratiche interne, alle linee guida del settore e alle normative.
- AWS Security Hub— Questo AWS servizio offre una visione completa dello stato di sicurezza dell'utente, AWS che consente di verificare la conformità agli standard e alle best practice del settore della sicurezza.

Resilienza in AWS Deep Learning AMI

L'infrastruttura AWS globale è costruita attorno a AWS regioni e zone di disponibilità. AWS Le regioni forniscono più zone di disponibilità fisicamente separate e isolate, collegate con reti a bassa latenza, ad alto throughput e altamente ridondanti. Con le zone di disponibilità, puoi progettare e gestire applicazioni e database che eseguono automaticamente il failover tra zone di disponibilità senza interruzioni. Le zone di disponibilità sono più disponibili, tolleranti ai guasti e scalabili rispetto alle infrastrutture a data center singolo o multiplo tradizionali.

Per ulteriori informazioni su AWS regioni e zone di disponibilità, consulta Global Infrastructure.AWS

Per informazioni sulle funzionalità per supportare le esigenze di resilienza di dati e backup, consulta la sezione Resilienza in Amazon EC2.

Sicurezza dell'infrastruttura in AWS Deep Learning AMI

La sicurezza dell'infrastruttura di AWS Deep Learning AMI è supportata da Amazon EC2. Per ulteriori informazioni, consulta la sezione Sicurezza dell'infrastruttura in Amazon EC2.

Modifiche importanti a DLAMI

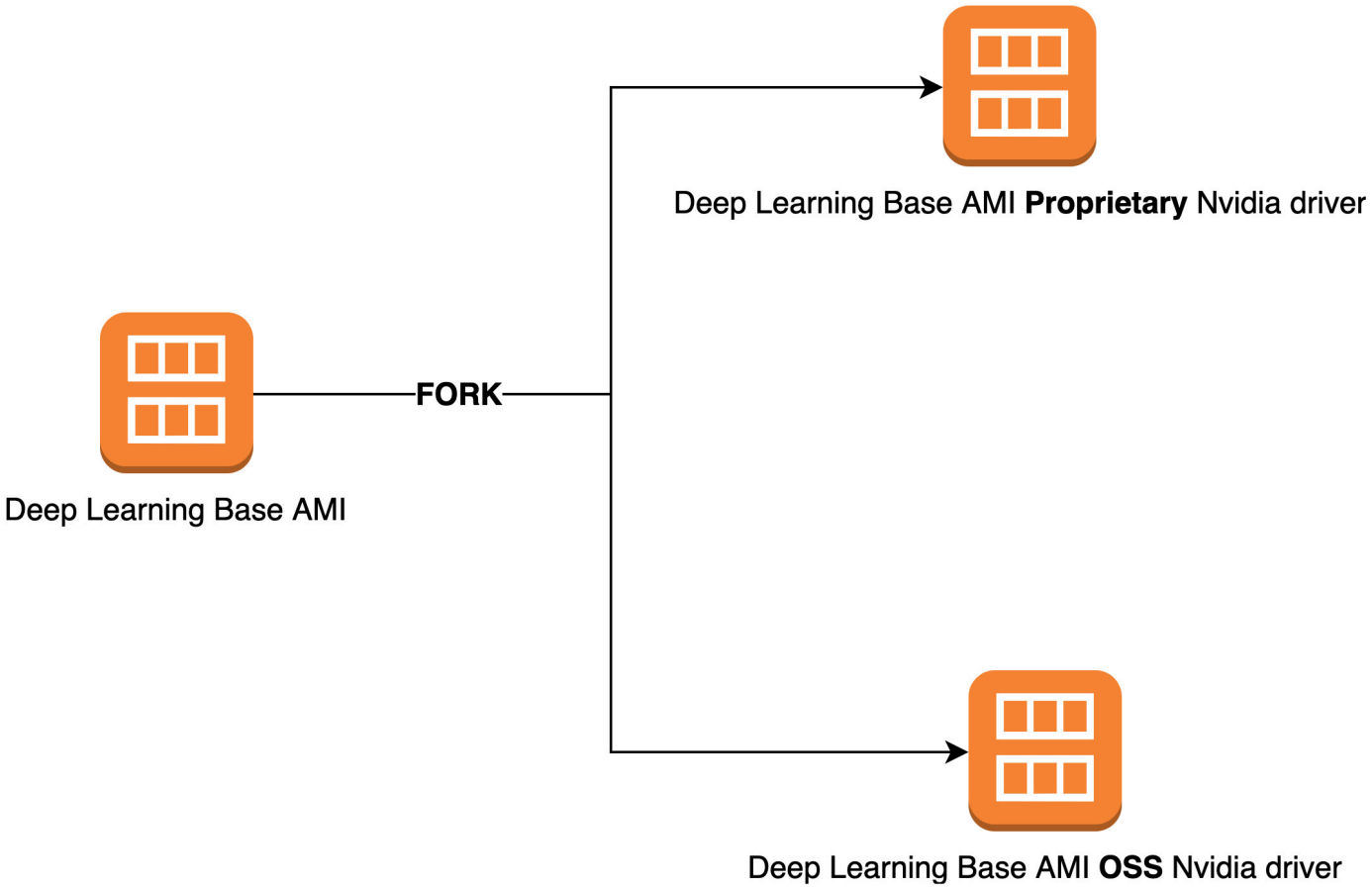
Domande frequenti

Cosa sta cambiando?

Il 15/11/2023 AWS Deep Learning AMI (DLAMIS) sarà diviso in due gruppi separati:

- DLAMI che utilizzano driver proprietari Nvidia (per supportare P3, P3dn, G3).
- DLAMI che utilizzano il driver Nvidia OSS (per supportare G4dn, G5, P4, P5).

Di conseguenza, verranno creati nuovi DLAMI per ciascuna delle due categorie con nuovi nomi e nuovi ID AMI. Questi DLAMI non saranno intercambiabili, ovvero i DLAMI di un gruppo non supporteranno le istanze supportate dall'altro gruppo, ad esempio i DLAMI che supportano p5 non supporteranno g3 e viceversa.



Perché è necessaria questa modifica?

Attualmente le DLAMI per GPU NVIDIA includono un driver kernel proprietario di NVIDIA. Tuttavia, recentemente la comunità del kernel Linux upstream ha accettato una modifica che isola i driver del kernel proprietari, come il driver per GPU NVIDIA, dalla comunicazione con altri driver del kernel. Questa modifica disabilita GPUDirect RDMA sulle istanze della serie P4/P5, ovvero il meccanismo che consente alle GPU di utilizzare in modo efficiente EFA per la formazione distribuita. Di conseguenza, DLamis utilizzerà il driver OpenRM (driver open source NVIDIA), collegato ai driver open source EFA per supportare G4dn, G5, P4 e P5. Tuttavia, questo driver OpenRM non supporterà le istanze più vecchie (P3, G3 ecc.) Pertanto, per continuare a fornire DLAMI attuali, performanti e sicuri che supportino entrambi i tipi di istanze, divideremo le DLAMI in due gruppi: uno con il driver OpenRM (che supporta G4dn, G5, P4 e P5) e uno con il driver proprietario precedente (che supporta le istanze precedenti P3, P3dn, G3).

Quali DLAMI sono interessati da questa modifica?

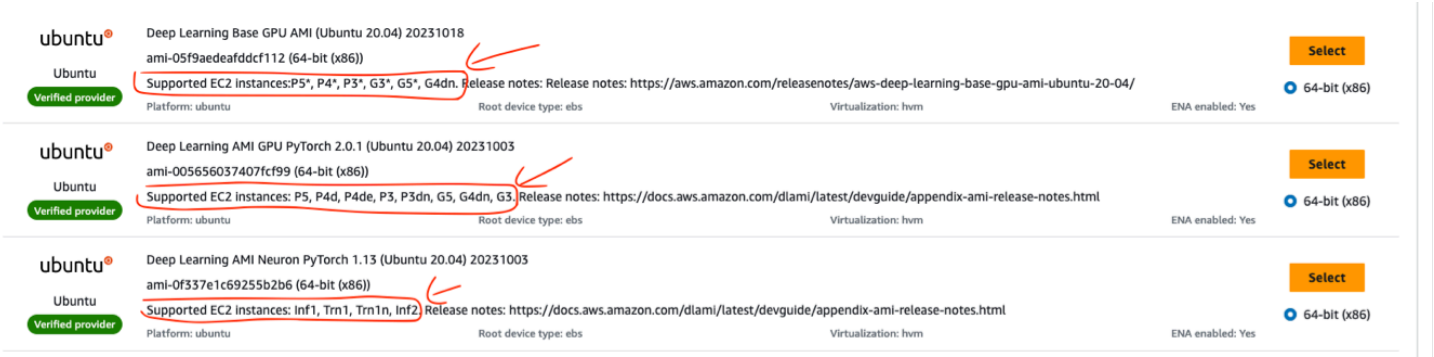
Tutte le DLAMI sono interessate da questa modifica.

Cosa significa questo per te?

I nuovi DLAMI continueranno a fornire funzionalità, prestazioni e sicurezza degli attuali DLAMI purché vengano eseguiti su un tipo di istanza compatibile. Se si utilizza DLAMIS, è necessario assicurarsi che venga avviato un DLAMI su una delle istanze compatibili menzionate nelle note di rilascio di ciascun DLAMI (vedere qui). Ad esempio: dovrai adattare questa modifica a:

- Richiama DLamis con le query CLI corrette (vedi sotto)
- Avvia DLamis dalla console e dalla CLI su un tipo di istanza compatibile

Se stai avviando DLAMI dalla console EC2 Quickstart: ogni descrizione DLAMI elenca i tipi di istanze supportate nella console EC2. È necessario avviare i DLAMIS su istanze compatibili.



Se state avviando DLamis utilizzando la CLI, dovrete modificare le vostre interrogazioni. Per esempio:

Attualmente la seguente query CLI viene utilizzata per le DLAMI di base che supportano tutte le istanze [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
```

Le nuove interrogazioni CLI saranno:

Per DLAMI di base che supportano P3, P3dn e G3:

```
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
```

Per DLAMI di base che supportano G4dn, G5, P4 e P5:

```
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
```

Consulta le note di rilascio aggiornate per le nuove AMI qui. Per come avviare le AMI sulle istanze EC2, consulta le istruzioni qui.

Quando dovresti iniziare a usare i nuovi DLAMI?

È necessario iniziare a utilizzare i nuovi DLAMI il prima possibile per i framework, le dipendenze, le patch e le funzionalità più recenti. Facoltativamente, se utilizzi le DLAMI di Amazon Linux 2 rilasciate prima dell'8/11/2023, puoi scegliere di continuare a applicare le patch in tempo reale alle loro DLAMI (vedi le istruzioni qui) fino al 30/11/2023.

Ci saranno perdite di funzionalità con i nuovi DLAMI?

No, non vi è alcuna perdita di funzionalità con i nuovi DLamis. I nuovi DLAMI dopo la divisione continueranno a fornire tutte le funzionalità, le prestazioni e la sicurezza dei vecchi DLAMI prima della divisione, purché vengano eseguiti su un'istanza compatibile. Stiamo dividendo i DLAMI in due gruppi in modo da continuare a offrire DLAMI aggiornati, performanti e sicuri da utilizzare su un'ampia gamma di istanze.

Che dire dei DLC?

I DLC non includono il driver NVIDIA, quindi non sono interessati da questa modifica. Tuttavia, dovresti assicurarti che i DLC vengano eseguiti su AMI compatibili con le istanze sottostanti.

Informazioni correlate

Argomenti

- Forum
- Post di blog correlati
- Domande frequenti

Forum

- Forum:AWS AMI di deep learning

Post di blog correlati

- Elenco aggiornato degli articoli relativi alle AMI di deep learning
- Lanciare unAWS Deep Learning AMI (in 10 minuti)
- Formazione più rapida con la TensorFlow versione 1.6 ottimizzata su istanze Amazon EC2 C5 e P3
- Nuove AMI diAWS deep learning per i professionisti dell'apprendimento automatico
- Nuovi corsi di formazione disponibili: Introduzione all'Machine Learning e al deep learning suAWS
- Viaggio nel deep learning conAWS

Domande frequenti

- D: Come posso tenere traccia degli annunci di prodotto relativi a DLAMI?

  Puoi farlo in due modi:

  - Aggiungi ai preferiti questa categoria di blog, «AWSAMI di deep learning», che si trova qui: Elenco aggiornato degli articoli relativi alle AMI di deep learning.
  - «Guarda» il forum:AWS AMI di deep learning
- D: I driver NVIDIA e CUDA sono installati?

  Sì. Alcuni DLAMI hanno versioni diverse. AMI di deep learning diHa le versioni più recenti di qualsiasi DLAMI. Per informazioni più dettagliate, consulta Installazioni CUDA e binding di

framework. Puoi anche fare riferimento alle note di rilascio specifiche dell'AMI per confermare cosa è installato.

- D: CuDNN è installato?

  Sì.

- D: Come posso verificare che le GPU siano state rilevate e il loro stato attuale?

  Esegui `nvidia-smi`. Questo comando visualizza una o più GPU, a seconda del tipo di istanza, nonché il consumo di memoria corrente delle stesse.

- D: Gli ambienti virtuali sono configurati per me?

  Sì, ma solo sulla AMI di deep learning di.

- D: Quale versione di Python è installata?

  Ogni DLAMI ha sia Python 2 che 3. La AMI di deep learning di include ambienti per entrambe le versioni di ogni framework.

- D. Keras è installato?

  Dipende dall'AMI. La AMI di deep learning di include Keras come front-end per ogni framework. La versione di Keras dipende dal supporto del framework.

- D. È gratuito?

  Tutti i DLAMI sono gratuiti. Tuttavia, a seconda del tipo di istanza scelto, l'istanza potrebbe non essere gratuita. Per ulteriori informazioni, consulta Prezzi per il DLAMI.

- D: Ricevo errori CUDA o messaggi relativi alla GPU dal mio framework. Qual è il problema?

  Verifica il tipo di istanza utilizzato. Molti esempi e tutorial funzionano soltanto se è presente una GPU. Se l'esecuzione non`nvidia-smi` mostra alcuna GPU, è necessario avviare un altro DLAMI utilizzando un'istanza con una o più GPU. Per ulteriori informazioni, consulta Selezione del tipo di istanza per DLAMI.

- D: Posso usare Docker?

  Docker è preinstallato dalla versione 14 dell'AMI Deep Learning con Conda. Nota che vorrai usare nvidia-docker su istanze GPU per utilizzare la GPU.

- D: In quali aree geografiche sono disponibili i DLAMI Linux?

Regione	Codice
Stati Uniti orientali (Ohio)	us-east-2
US East (N. Virginia)	us-east-1
GovCloud	us-gov-west-1
Stati Uniti occidentali (California settentrionale)	us-west-1
US West (Oregon)	us-west-2
Beijing (Cina)	cn-north-1
Ningxia (Cina)	cn-northwest-1
Asia Pacifico (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
UE (Francoforte)	eu-central-1
UE (Irlanda)	eu-west-1
UE (Londra)	eu-west-2
UE (Parigi)	eu-west-3
SA (San Paolo)	sa-east-1

- D: In quali aree geografiche sono disponibili i DLAMI di Windows?

Regione	Codice
Stati Uniti orientali (Ohio)	us-east-2
US East (N. Virginia)	us-east-1
GovCloud	us-gov-west-1
Stati Uniti occidentali (California settentrionale)	us-west-1
US West (Oregon)	us-west-2
Beijing (Cina)	cn-north-1
Asia Pacifico (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
UE (Francoforte)	eu-central-1
UE (Irlanda)	eu-west-1
UE (Londra)	eu-west-2
UE (Parigi)	eu-west-3
SA (San Paolo)	sa-east-1

Note di rilascio per DLAMI

> **Note**
>
> AWS Deep Learning AMI I hanno una cadenza di rilascio notturna per le patch di sicurezza. Queste patch di sicurezza incrementali non sono incluse nelle note di rilascio ufficiali.

Consultate la pagina DLAMI Support Policy per eventuali note di rilascio del framework non supportate.

Base DLAMI

GPU

- AWS AMI di base di apprendimento approfondito (Amazon Linux 2)
- AWS AMI base di apprendimento approfondito (Ubuntu 20.04)

AWS Neurone

- AWS Base di apprendimento profondo AMI Neuron (Amazon Linux 2)
- AWS Base di apprendimento profondo AMI Neuron (Ubuntu 20.04)

Qualcomm

- AWS Base di deep learning AMI Qualcomm (Amazon Linux 2)

DLAMI a framework singolo

PyTorch-AMI specifico

- GPU
  - AWS GPU AMI PyTorch 2.1 con apprendimento approfondito (Ubuntu 20.04)
  - AWS GPU AMI PyTorch 1.13 per apprendimento approfondito (Amazon Linux 2)

  - AWS GPU AMI di deep learning PyTorch 1.13 (Ubuntu 20.04)
- AWS Neurone
  - AWS AMI Neuron PyTorch 1.13 per apprendimento approfondito (Amazon Linux 2)
  - AWS AMI Neuron PyTorch 1.13 per apprendimento profondo (Ubuntu 20.04)

TensorFlow-AMI specifico

- GPU
  - AWS GPU AMI TensorFlow 2.15 con apprendimento approfondito (Amazon Linux 2)
  - AWS GPU AMI di deep learning TensorFlow 2.15 (Ubuntu 20.04)
  - AWS GPU AMI TensorFlow 2.13 con apprendimento approfondito (Amazon Linux 2)
  - AWS GPU AMI di deep learning TensorFlow 2.13 (Ubuntu 20.04)
- AWS Neurone
  - AWS AMI Neuron TensorFlow 2.10 per apprendimento approfondito (Amazon Linux 2)
  - AWS AMI Neuron TensorFlow 2.10 per apprendimento profondo (Ubuntu 20.04)

DLAMI multi-framework

GPU

> **Note**
>
> Se utilizzi solo un framework di machine learning, ti consigliamo un DLAMI a framework singolo

- AWS AMI di apprendimento approfondito (Amazon Linux 2)

AWS Neurone

- AWS AMI Neuron con apprendimento profondo (Ubuntu 22.04)

Deep Learning

Nella tabella seguente sono elencate le informazioni sulle feature deprecate in AWS Deep Learning AMI.

Caratteristiche obsolete	Data di deprecazione	Avviso di deprecazione
Ubuntu 16.04	10/07/2021	Ubuntu Linux 16.04 LTS ha raggiunto la fine della sua finestra LTS quinquennale il 30 aprile 2021 e non è più supportato dal suo fornitore . Non ci sono più aggiornamenti all'AMI Deep Learning Base (Ubuntu 16.04) nelle nuove versioni a partire da ottobre 2021. Le versioni precedenti continueranno a essere disponibili.
Amazon Linux	10/07/2021	Amazon Linux è end-of-life aggiornato a dicembre 2020. Non ci sono più aggiornamenti all'AMI Deep Learning (Amazon Linux) nelle nuove versioni a partire da ottobre 2021. Deep Learning (Amazon Linux) (Amazon Linux AMI Deep Learning (Amazon Linux) Deep Learning (Amazon Linux)
Chainer	01/07/2020	Chainer ha annunciato la fine delle versioni principali a dicembre 2019. Di conseguenza, non includere

Caratteristiche obsolete	Data di deprecazione	Avviso di deprecazione
		mo più gli ambienti Chainer Conda nel DLAMI a partire da luglio 2020. Le versioni precedenti del DLAMI che contengono questi ambienti continueranno a essere disponibili. Tuttavia, verranno forniti aggiornamenti a questi ambienti solo se sono disponibili correzioni di sicurezza pubblicate dalla comunità open source per questi framework.
Python 3.6	15/06/2020	A causa delle richieste dei clienti, stiamo passando a Python 3.7 per le nuove versioni TF/MX/PT.
Python 2	01/01/2020	La comunità open source di Python ha ufficialmente interrotto il supporto per Python 2. Le comunità TensorFlow PyTorch, e MXNet hanno anche annunciato che le versioni TensorFlow 1.15, TensorFlow 2.1, PyTorch 1.4 e MXNet 1.6.0 saranno le ultime a supportare Python 2.

Cronologia dei documenti della guida per sviluppatori di AWS Deep Learning AMI

Modifica	Descrizione	Data
DLAMI Graviton	AWS Deep Learning AMIOra supporta immagini su GPU Graviton basate su processori Arm.	29 novembre 2021
Habana DLAMI	AWS Deep Learning AMIOra supporta l'hardware Habana Gaudi e l'SDK Habana SynapseAI.	25 ottobre 2021
TensorFlow 2	L'AMI Deep Learning con Conda ora ne include TensorFlow 2 con CUDA 10.	3 dicembre 2019
AWSInferenza	L'AMI Deep Learning ora supporta l'hardwareAWS Inferentia e l'SDKAWS Neuron.	3 dicembre 2019
Utilizzo di TensorFlow Serving with a Inception Model	Un esempio di utilizzo dell'inferenza con un modello Inception è stato aggiunto per TensorFlow Serving, sia con che senza Elastic Inference.	28 Novembre 2018
Allenamento con 256 GPU con TensorFlow e Horovod	Il tutorial TensorFlow con Horovod è stato aggiornato per aggiungere un esempio di addestramento a più nodi.	28 Novembre 2018
Elastic Inference	I prerequisiti di inferenza elastica e le informazioni	28 Novembre 2018

	correlate sono stati aggiunti alla guida all'installazione.	
MMS v1.0 rilasciato sul DLAMI.	Il tutorial MMS è stato aggiornato per utilizzare il nuovo formato di archivio modello (.mar) e mostra le nuove caratteristiche di avvio e arresto.	15 Novembre 2018
Installazione TensorFlow da una build notturna	È stato aggiunto un tutorial che spiega come TensorFlow disinstallare e quindi installar e una build notturna TensorFlo w sulla tua AMI Deep Learning con Conda.	16 ottobre 2018
Installazione di CNTK da una build notturna	È stato aggiunto un tutorial che spiega come disinstallare CNTK, quindi installare una build notturna di CNTK sulla tua AMI Deep Learning con Conda.	16 ottobre 2018
Installazione di Apache MXNet (incubazione) da una build notturna	È stato aggiunto un tutorial che spiega come disinstallare MXNet, quindi installare una build notturna di MXNet sulla tua AMI di Deep Learning con Conda.	16 ottobre 2018
Installazione PyTorch da una build notturna	È stato aggiunto un tutorial che spiega come PyTorch disinstallare e quindi installar e una build notturna PyTorch sulla tua AMI Deep Learning con Conda.	25 settembre 2018

Docker è ora preinstallato sul tuo DLAMI	Dalla versione 14 dell'AMI Deep Learning con Conda, Docker e NVIDIA la versione di Docker per GPU è preinstal lata.	25 settembre 2018
TensorBoard Tutorial	Esempio spostato in ~ / examples/tensorboard. Percorsi del tutorial aggiornati.	23 luglio 2018
Tutorial MXBoard	È stato aggiunto un tutorial su come utilizzare MXBoard per la visualizzazione di modelli MXNet.	23 luglio 2018
Tutorial di formazione distribui ti	È stato aggiunto un tutorial su come utilizzare Keras-MXN et per il training con più GPU. Il tutorial per Chainer è stato aggiornato a v4.2.0.	23 luglio 2018
Tutorial Conda	L'esempio di MOTD è stato aggiornato per riflettere una versione più recente.	23 luglio 2018
Tutorial Chainer	Il tutorial è stato aggiornato per utilizzare gli esempi più recenti dal codice sorgente di Chainer.	23 luglio 2018

Aggiornamenti precedenti:

La tabella seguente elenca importanti modifiche introdotte in ogni versione della AWS Deep Learning AMI prima di luglio 2018.

Modifica	Descrizione	Data
TensorFlow con Horovod	È stato aggiunto un tutorial per allenarsi ImageNet con TensorFlow e Horovod.	6 giugno 2018
Guida per l'upgrade	Aggiunta della guida per l'upgrade.	15 maggio 2018
Nuovo regioni e nuovo tutorial di 10 minuti	Aggiunta di nuove regioni: Stati Uniti occidentali (California settentrionale), Sud America, Canada (Centrale), UE (Londra) e UE (Parigi). Inoltre, prima versione di un tutorial di 10 minuti intitolato "Getting Started with Deep Learning AMI".	26 Aprile 2018
Tutorial di Chainer	Aggiunto un tutorial per l'utilizzo di Chainer con più GPU, con una singola GPU e con CPU. Upgrade dell'integrazione CUDA da CUDA 8 a CUDA 9 per vari framework.	28 febbraio 2018
AMI Linux v3.0, oltre all'introduzione di MXNet Model Server, TensorFlow Serving e TensorBoard	Sono stati aggiunti tutorial per le AMI Conda con nuove funzionalità di modellazione e visualizzazione utilizzando MXNet Model Server v0.1.5, TensorFlow Serving v1.4.0 e TensorBoard v0.4.0. Funzionalità CUDA per AMI e framework descritte nelle panoramiche di Conda e CUDA. Note di rilascio più	25 gennaio 2018

Modifica	Descrizione	Data
	recenti spostate in https://aws.amazon.com/releasenotes/	
AMI Linux v2.0	Aggiornamento delle AMI Base, Source e Conda con NCCL 2.1. Le AMI Source e Conda sono state aggiornate con MXNet v1.0, PyTorch 0.3.0 e Keras 2.0.9.	11 dicembre 2017
Aggiunta di due opzioni di AMI Windows	Rilascio delle AMI Windows 2012 R2 e 2016: aggiunta alla guida di selezione delle AMI e alle note di rilascio.	30 novembre 2017
Prima versione della documentazione	Descrizione dettagliata della modifica con link all'argomento/alla sezione oggetto della modifica.	15 novembre 2017

Glossario AWS

Per la terminologia AWS più recente, consultare il glossario AWS nella documentazione di riferimento per Glossario AWS.

Le traduzioni sono generate tramite traduzione automatica. In caso di conflitto tra il contenuto di una traduzione e la versione originale in Inglese, quest'ultima prevarrà.