aws

Guía para desarrolladores

AMI de aprendizaje profundo

AMI de aprendizaje profundo: Guía para desarrolladores

Table of Contents

¿Qué es la AWS Deep Learning AMI?

Le damos la bienvenida a la Guía del usuario de la AWS Deep Learning AMI.

La AWS Deep Learning AMI (DLAMI) es lo único que necesita para el aprendizaje profundo en la nube. Esta instancia de máquina personalizada está disponible en la mayoría de las regiones de Amazon EC2 para una amplia variedad de tipos de instancias, desde una pequeña instancia que solo tiene CPU hasta las instancias de gran potencia más recientes que disponen de varias GPU. Viene preconfigurada con NVIDIA CUDA y NVIDIA cuDNN, así como con las versiones más recientes de los marcos de trabajo de aprendizaje profundo más populares.

Acerca de esta guía

Esta guía le ayudará a lanzar y utilizar la DLAMI. Abarca varios casos de uso comunes para el aprendizaje profundo, tanto para el entrenamiento como para la inferencia. También se explica cómo elegir la AMI más adecuada y el tipo de instancias que le pueden interesar. La DLAMI incluye varios tutoriales para cada uno de los marcos de trabajo. También cuenta con tutoriales sobre entrenamiento distribuido, depuración, uso de AWS Inferentia y otros conceptos clave. Encontrará instrucciones sobre cómo configurar Jupyter para ejecutar los tutoriales en el navegador.

Requisitos previos

Debe estar familiarizado con las herramientas de línea de comandos y tener conocimientos básicos de Python para ejecutar correctamente la DLAMI. Los propios marcos de trabajo incluyen tutoriales sobre su utilización; sin embargo, esta guía puede mostrarle cómo activar cada uno de ellos y encontrar los tutoriales apropiados para empezar.

Usos de ejemplo de DLAMI

Obtención de conocimientos sobre el aprendizaje profundo: la DLAMI es una gran elección para conocer o enseñar los marcos de trabajo de machine learning y aprendizaje profundo. Elimina los quebraderos de cabeza asociados a la solución de problemas de las instalaciones de los marcos de trabajo para conseguir que funcionen correctamente en el mismo equipo. La DLAMI incluye un cuaderno de Jupyter y facilita la ejecución de los tutoriales proporcionados por los marcos de trabajo a los usuarios que no están familiarizados con el machine learning y aprendizaje profundo.

Desarrollo de aplicaciones: si es un desarrollador de aplicaciones y está interesado en el uso del aprendizaje profundo para conseguir que sus aplicaciones utilicen los avances más recientes en IA, la DLAMI es el banco de pruebas perfecto. Cada marco de trabajo incluye tutoriales sobre cómo empezar a utilizar el aprendizaje profundo, y muchos de ellos tienen colecciones de modelos que permiten probarlo sin necesidad de crear redes neuronales ni de llevar a cabo el entrenamiento de modelos. Algunos ejemplos le muestran cómo crear una aplicación de detección de imágenes en tan solo unos minutos, o cómo crear una aplicación de reconocimiento de voz para su propio chatbot.

Aprendizaje automático y análisis de datos: si es un científico de datos o está interesado en procesar datos con el aprendizaje profundo, comprobará que muchos de los marcos de trabajo son compatibles con R y Spark. Encontrará tutoriales sobre cómo crear desde regresiones sencillas hasta sistemas escalables de procesamiento de datos para sistemas de predicción y personalización.

Investigación: si es un investigador y desea probar un marco de trabajo o un nuevo modelo, o entrenar modelos nuevos, las características de escalado de la DLAMI y las capacidades de AWS pueden aliviar la carga que suponen las instalaciones tediosas y la administración de varios nodos de entrenamiento.

> **Note**
>
> Aunque la elección inicial puede ser actualizar el tipo de instancia a una instancia más grande con más GPU (hasta 8), también es posible escalar de forma horizontal creando un clúster de instancias de DLAMI. Consulte Información relacionada para obtener más información sobre las compilaciones de clústeres.

Características de la DLAMI

Marcos de trabajo preinstalados

Actualmente, hay dos tipos principales de la DLAMI con otras variaciones relacionadas con el sistema operativo y las versiones de software:

- AMI de aprendizaje profundo con Conda: marcos de trabajo instalados por separado utilizando paquetes `conda` y distintos entornos de Python
- AMI base de aprendizaje profundo: sin marcos de trabajo instalados; solo NVIDIA CUDA y otras dependencias

La AMI de aprendizaje profundo con Conda utiliza entornos de `conda` para aislar cada marco de trabajo, de forma que pueda alternar entre ellos cuando desee sin preocuparse por posibles conflictos con sus dependencias.

Esta es la lista completa de plataformas compatibles con la AMI de aprendizaje profundo con Conda:

- Apache MXNet (en fase de incubación)
- PyTorch
- TensorFlow 2

> **Note**
>
> A partir de la versión v28, ya no incluimos los entornos CNTK, Caffe, Caffe2, Theano, Chainer o Keras Conda en la AWS Deep Learning AMI. Las versiones anteriores de AWS Deep Learning AMI que contienen estos entornos seguirán estando disponibles. Sin embargo, solo proporcionaremos actualizaciones a estos entornos si la comunidad de código abierto publica correcciones de seguridad para estos marcos.

Software de GPU preinstalado

Incluso aunque utilice una instancia que solo tiene CPU, la DLAMI tendrá NVIDIA CUDA y NVIDIA cuDNN. El software instalado es el mismo, independientemente del tipo de instancia. Tenga en cuenta que las herramientas específicas de GPU solo funcionan en una instancia que tenga al menos una GPU. Puede encontrar más información sobre este tema en la Selección del tipo de instancia para DLAMI.

Para obtener más información sobre la instalación de CUDA, consute Instalaciones de CUDA y enlaces de marco de trabajo.

Distribución y visualización de modelos

La AMI de aprendizaje profundo con Conda viene preinstalada con dos tipos de servidores de modelos, uno para MXNet y otro para TensorFlow, así como TensorBoard, para las visualizaciones de los modelos.

- Model Server for Apache MXNet (MMS)
- TensorFlow Sirviendo

- TensorBoard

Introducción

Primeros pasos con la DLAMI

Esta guía incluye consejos sobre cómo elegir la DLAMI más adecuada, seleccionar el tipo de instancia apropiada para su caso de uso y presupuesto, y Información relacionada que describen configuraciones personalizadas que pueden resultar interesantes.

Si es la primera vez que utiliza AWS o Amazon EC2, empiece con la AMI de aprendizaje profundo con Conda. Si ya conoce Amazon EC2 y otros servicios de AWS como Amazon EMR, Amazon EFS o Amazon S3 y le interesa integrar dichos servicios en proyectos que necesiten entrenamiento o inferencias distribuidas, consulte Información relacionada para ver si hay alguno que se adapte a su caso de uso.

Le recomendamos que consulte Elección de la DLAMI para que se haga una idea del tipo de instancia que mejor se adapta a su aplicación.

Otra opción es este rápido tutorial: Lanzar un AWS Deep Learning AMI (en 10 minutos).

Paso siguiente

Elección de la DLAMI

Elección de la DLAMI

Ofrecemos una amplia gama de opciones de DLAMI. Para ayudarle a seleccionar la DLAMI correcta para su caso, agrupamos las imágenes por el tipo de hardware o la funcionalidad para la que se desarrollaron. Nuestras agrupaciones principales son:

- Tipo de DLAMI: CUDA versus base versus marco único versus marco múltiple (Conda DLAMI)
- Arquitectura informática: basada en x86 versus basada en ArmAWS Graviton
- Tipo de procesador: GPU versus CPU versus Inferentia versus Habana
- SDK: CUDA versus AWS Neuron versus SynapsesAI
- SO: Amazon Linux versus Ubuntu

En el resto de los temas de esta guía encontrará más detalles.

Temas

Tema siguiente

AMI de aprendizaje profundo con Conda

Instalaciones de CUDA y enlaces de marco de trabajo

Mientras que el aprendizaje profundo es algo bastante novedoso, todos los marcos de trabajo ofrecen versiones "estables". Es posible que estas versiones estables no funcionen con las implementaciones y características más recientes de CUDA o cuDNN. Su caso de uso y las características que necesita pueden ayudarle a elegir un marco. Si no está seguro, utilice la última AMI de aprendizaje profundo con Conda. Tiene binarios `pip` oficiales de todos los marcos con CUDA 10, que utilizan la versión más reciente compatible con cada plataforma. Si desea obtener las versiones más recientes y personalizar su entorno de aprendizaje profundo, utilice la AMI base de aprendizaje profundo.

Eche un vistazo a nuestra guía sobre Comparación de Stable y Release Candidates para obtener más información.

Elija una DLAMI con CUDA

En AMI base de aprendizaje profundo tiene todas las series CUDA 11 disponibles, incluidas las 11.0, 11.1 y 11.2.

En AMI de aprendizaje profundo con Conda tiene todas las series CUDA 11 disponibles, incluidas las 11.0, 11.1 y 11.2.

> **Note**
>
> A partir de la versión v28, ya no incluimos los entornos CNTK, Caffe, Caffe2, Theano, Chainer o Keras Conda en la AWS Deep Learning AMI. Las versiones anteriores de AWS

Deep Learning AMI que contienen estos entornos seguirán estando disponibles. Sin embargo, solo proporcionaremos actualizaciones a estos entornos si la comunidad de código abierto publica correcciones de seguridad para estos marcos.

Para obtener números de versión específicos, consulte las Notas de la versión de DLAMI.

Elija este tipo de DLAMI u obtenga más información sobre las distintas DLAMI mediante la opción Tema siguiente.

Elija una de las versiones de CUDA y consulte en el Apéndice la lista completa de las DLAMI que tengan esa versión, u obtenga más información sobre las distintas DLAMI en el tema indicado en la sección Tema siguiente.

Tema siguiente

AMI base de aprendizaje profundo

Temas relacionados

- Para obtener instrucciones sobre cómo cambiar de versión de CUDA, consulte el tutorial Uso de la AMI base de aprendizaje profundo.

AMI base de aprendizaje profundo

La AMI base de aprendizaje profundo es como un lienzo vacío para el aprendizaje profundo. Incluye todo lo que se necesita hasta el momento de la instalación de un marco de trabajo determinado, e incluye las versiones de CUDA que haya elegido.

Por qué elegir la DLAMI base

Este grupo de AMI es útil para los colaboradores de proyectos que desean adaptar un proyecto de aprendizaje profundo y compilar la versión más reciente. Está pensado para quienes desean actualizar su propio entorno con la confianza de que tienen instalado y en funcionamiento el software más reciente de NVIDIA, y desean centrarse en seleccionar los marcos de trabajo y las versiones que quieren instalar.

Elija este tipo de DLAMI u obtenga más información sobre las distintas DLAMI mediante la opción Tema siguiente.

Tema siguiente

DLAMI con Conda

Temas relacionados

- Uso de la AMI base de aprendizaje profundo

AMI de aprendizaje profundo con Conda

La DLAMI con Conda utiliza entornos virtuales de `conda`. Estos entornos están configurados para mantener separadas las instalaciones de los distintos marcos de trabajo y agilizar el paso de un marco a otro. Esto resulta ideal para aprender y experimentar con todos los marcos de trabajo que ofrece la DLAMI. La mayoría de los usuarios descubrirán que la nueva AMI de aprendizaje profundo con Conda es perfecta para ellos.

Estas AMI son las DLAMI principales. Se actualizan a menudo con las versiones más recientes de los marcos de trabajo, y disponen del software y los controladores de GPU más recientes. Se las denominará en general con la AWS Deep Learning AMI en la mayoría de los documentos.

- La DLAMI de Ubuntu 18.04 tiene los siguientes marcos: Apache MXNet (en fase de inicubación), PyTorch y TensorFlow 2.
- La DLAMI de Amazon Linux 2 tiene los siguientes marcos: Apache MXNet (en fase de inicubación), PyTorch y TensorFlow 2.

> **Note**
>
> A partir de la versión v28, ya no incluimos los entornos CNTK, Caffe, Caffe2, Theano, Chainer y Keras Conda en la AWS Deep Learning AMI. Las versiones anteriores de AWS Deep Learning AMI que contienen estos entornos seguirán estando disponibles. Sin embargo, solo proporcionaremos actualizaciones a estos entornos si la comunidad de código abierto publica correcciones de seguridad para estos marcos.

Comparación de Stable y Release Candidates

Las AMI de Conda utilizan archivos binarios optimizados para las versiones formales más recientes de cada marco de trabajo. No se esperan "release candidates" ni funciones experimentales. Las

optimizaciones dependen de la compatibilidad del marco de trabajo para tecnologías de aceleración como MKL DNN de Intel, que acelerarán el entrenamiento y la inferencia en los tipos de instancias de CPU C5 y C4. Los archivos binarios también se compilan para permitir conjuntos de instrucciones de Intel avanzados, incluidos, entre otros, AVX, AVX-2, SSE4.1 y SSE4.2. Estas instrucciones aceleran las operaciones con vectores y puntos flotantes en las arquitecturas de CPU de Intel. Además, para los tipos de instancias de GPU, se actualizan la CUDA y cuDNN con la versión que sea compatible con la última versión oficial.

La AMI de aprendizaje profundo con Conda instala automáticamente la versión más optimizada del marco de trabajo para su instancia de Amazon EC2 tras la primera activación del marco de trabajo. Para obtener más información, consulte Uso de la AMI de aprendizaje profundo con Conda.

Si desea instalar desde el código fuente mediante opciones de compilación personalizadas u optimizadas, las AMI base de aprendizaje profundo podrían ser una opción más adecuada para usted.

Retirada de Python 2

La comunidad de código abierto de Python finalizó oficialmente la compatibilidad con Python 2 el 1 de enero de 2020. Las comunidades de TensorFlow y PyTorch han anunciado que las versiones TensorFlow 2.1 y PyTorch 1.4 son las últimas compatibles con Python 2. Las versiones anteriores de la DLAMI (v26, v25, etc.) que contienen Python 2 Conda siguen estando disponibles. Sin embargo, proporcionamos actualizaciones para los entornos Conda de Python 2 en las versiones de DLAMI publicadas anteriormente solo si la comunidad de código abierto ha publicado correcciones de seguridad para esas versiones. Las versiones de DLAMI con las últimas versiones de los marcos TensorFlow y PyTorch no contienen los entornos Conda de Python 2.

Compatibilidad con CUDA

Los números de versión específicos de CUDA se encuentran en las notas de la versión de GPU DLAMI.

Tema siguiente

Opciones de arquitectura de CPU para DLAMI

Temas relacionados

- Para ver un tutorial sobre el uso de una AMI de aprendizaje profundo con Conda, consulte el tutorial de Uso de la AMI de aprendizaje profundo con Conda.

Opciones de arquitectura de CPU para DLAMI

Las AWS Deep Learning AMI se ofrecen con arquitecturas de CPU AWS Graviton2 basadas en x86 o Arm.

Elija una de las DLAMI de GPU de Graviton para trabajar con una arquitectura de CPU basada en Arm. Todas las demás DLAMI de GPU están basadas actualmente en x86.

- AMI de aprendizaje profundo de AWS - Graviton GPU CUDA 11.4 (Ubuntu 20.04)
- AMI de aprendizaje profundo de AWS - Graviton GPU TensorFlow 2.6 (Ubuntu 20.04)
- AMI de aprendizaje profundo de AWS - Graviton GPU PyTorch 1.10 (Ubuntu 20.04)

Para obtener más información acerca de la DLAMI de la GPU de Graviton, consulte El DLAMI de Graviton. Para obtener más información sobre los tipos de instancias disponibles, consulte Selección del tipo de instancia para DLAMI.

Tema siguiente

Opciones de sistema operativo para la DLAMI

Opciones de sistema operativo para la DLAMI

Las DLAMI se ofrecen en los siguientes sistemas operativos.

- Amazon Linux 2
- Ubuntu 20.04
- Ubuntu 18.04

Las versiones anteriores de los sistemas operativos están disponibles en las DLAMI obsoletos. Para obtener más información sobre la obsolescencia de DLAMI, consulte Obsolescencias para DLAMI

Antes de elegir una DLAMI, evalúe qué tipo de instancia necesita e identifique su región de AWS.

Tema siguiente

Selección del tipo de instancia para DLAMI

Selección del tipo de instancia para DLAMI

Consulte el catálogo de AWS Deep Learning AMI para ver las familias de instancias de Amazon EC2 recomendadas que son compatibles con DLAMI específicas.

De manera más general, tenga en cuenta lo siguiente al seleccionar un tipo de instancia para una DLAMI.

- Si acaba de llegar al mundo del aprendizaje profundo, una instancia con una sola GPU podría ser suficiente para sus necesidades.
- Si le preocupa su presupuesto, puede usar instancias que solo funcionen con CPU.
- Si busca optimizar el alto rendimiento y la rentabilidad para la inferencia de modelos de aprendizaje profundo, puede utilizar instancias con chips de AWS Inferentia.
- Si busca optimizar el alto rendimiento y la rentabilidad para el entrenamiento de modelos de aprendizaje profundo, puede utilizar instancias con aceleradores de Habana.
- Si busca una instancia de GPU de alto rendimiento con una arquitectura de CPU basada en Arm, puede usar el tipo de instancia de G5g.
- Si está interesado en ejecutar un modelo previamente entrenado para inferencias y predicciones, puede asociar una Amazon Elastic Inference a su instancia de Amazon EC2. Amazon Elastic Inference le da acceso a un acelerador con una fracción de GPU.
- Para los servicios de inferencia de gran volumen, una única instancia de CPU con mucha memoria, o un clúster de dichas instancias, podría ser una mejor solución.
- Si está utilizando un modelo de gran tamaño con muchos datos o un tamaño de lote elevado, necesitará una instancia más grande con más memoria. También puede distribuir su modelo a un clúster de GPU. El uso de una instancia con menos memoria puede ser una solución más adecuada para usted si disminuye el tamaño del lote. Sin embargo, puede afectar a la precisión y a la velocidad de entrenamiento.
- Si desea ejecutar aplicaciones de machine learning con la Biblioteca de comunicación colectiva de NVIDIA (NCCL) que requieran un alto nivel de comunicaciones entre nodos a escala, puede utilizar Elastic Fabric Adapter (EFA).

Para obtener más información sobre las instancias, consulte Tipos de instancias EC2.

Los siguientes temas proporcionan información acerca de las consideraciones del tipo de instancia.

> **Important**
>
> Las AMI de aprendizaje profundo incluyen controladores, software o conjuntos de herramientas desarrollados o facilitados por NVIDIA Corporation o que son de su propiedad. Debe aceptar que va a utilizar esos controladores, el software o esos conjuntos de herramientas de NVIDIA solo en instancias de Amazon EC2 que incluyan hardware de NVIDIA.

Temas

Precios de la DLAMI

Los marcos de trabajo de aprendizaje profundo incluidos en la DLAMI son gratuitos, y cada uno tiene sus propias licencias de código abierto. Aunque el software incluido en la DLAMI es gratuito, tendrá que pagar por el hardware de la instancia de Amazon EC2 subyacente.

Algunos tipos de instancias de Amazon EC2 se ofrecen de forma gratuita. Es posible ejecutar la DLAMI en una de estas instancias gratuitas. Esto significa que usar DLAMI es totalmente gratis cuando solo se usa la capacidad de dicha instancia. Si decide que desea una instancia más potente, con más núcleos de CPU, más espacio en disco, más RAM y una o varias GPU, lo más probable es que esa instancia no se incluya en la capa gratuita.

Para obtener más información acerca de los precios y la selección de las instancias, consulte Precios de Amazon EC2.

Disponibilidad en las regiones de DLAMI

Cada región admite tipos de instancias distintos y, a menudo, un tipo de instancia tiene un costo ligeramente distinto en diferentes regiones. Las DLAMI no están disponibles en todas las regiones,

pero es posible copiarlas de una región a otra. Para obtener más información, consulte Copiar una DLAMI. Fíjese en la lista de selección de regiones y asegúrese de que elige una región que esté cerca de usted o de sus clientes. Si tiene previsto utilizar más de una DLAMI y posiblemente crear un clúster, asegúrese de utilizar la misma región para todos los nodos del clúster.

Para obtener más información sobre las regiones, visite Regiones de EC2 Regions.

Tema siguiente

Instancias de GPU recomendadas

Instancias de GPU recomendadas

Se recomienda una instancia de GPU para la mayoría de los fines de aprendizaje profundo. El entrenamiento de modelos nuevos es más rápido en una instancia de GPU que en una instancia de CPU. Puede escalar de forma sublineal cuando tenga varias instancias de GPU o si utiliza el entrenamiento distribuido en muchas instancias con GPU. Para configurar el entrenamiento distribuido, consulte Entrenamiento distribuido.

Los tipos de instancia que se muestran a continuación admiten DLAMI. Para obtener información sobre las opciones de tipos de instancias de GPU y sus usos, consulte Tipos de instancias de EC2 y seleccione Computación acelerada.

> **Note**
>
> El tamaño del modelo debe ser un factor a tener en cuenta para la selección de una instancia. Si su modelo supera la RAM disponible de una instancia, seleccione otro tipo de instancia con memoria suficiente para la aplicación.

- Las instancias de Amazon EC2 P3 tienen hasta 8 GPU NVIDIA Tesla V100.
- Las instancias de Amazon EC2 P4 tienen hasta 8 GPU NVIDIA Tesla A100.
- Las instancias de Amazon EC2 G3 tienen hasta 4 GPU NVIDIA Tesla M60.
- Las instancias de Amazon EC2 G4 tienen hasta 4 GPU NVIDIA T4.
- Las instancias de Amazon EC2 G5 tienen hasta 8 GPU NVIDIA A10G.
- Las instancias G5g de Amazon EC2 cuentan con procesadores Graviton2 basados en Arm de AWS.

Las instancias de DLAMI proporcionan herramientas para supervisar y optimizar los procesos de la GPU. Para obtener más información sobre la supervisión de los procesos de GPU, consulte Monitorización y optimización de GPU.

Para ver tutoriales específicos sobre cómo trabajar con instancias G5G, consulte El DLAMI de Graviton.

Tema siguiente

Instancias de CPU recomendadas

Instancias de CPU recomendadas

Dispone de muchas opciones asequibles en la categoría de CPU, tanto si cuenta con un presupuesto ajustado, quiere aprender sobre el aprendizaje profundo o desea ejecutar una servicio de predicción. Algunos marcos de trabajo aprovechan los MKL DNN de Intel, que aceleran el entrenamiento y la inferencia en los tipos de instancias de CPU C5 (no disponibles en todas las regiones), C3 y C4. Para obtener información sobre los tipos de instancias de CPU, consulte Tipos de instancias de EC2 y seleccione Computación optimizada.

> **Note**
>
> El tamaño del modelo debe ser un factor a tener en cuenta para la selección de una instancia. Si su modelo supera la RAM disponible de una instancia, seleccione otro tipo de instancia con memoria suficiente para la aplicación.

- Las instancias C5 de Amazon EC2 tienen hasta 72 vCPU Intel. Las instancias C5 son idóneas para el modelado científico, el procesamiento por lotes, los análisis distribuidos, la computación de alto rendimiento (HPC) y la inferencia de machine learning y aprendizaje profundo.
- Las instancias C4 de Amazon EC2 tienen hasta 36 vCPU Intel.

Tema siguiente

Instancias de Inferentia recomendadas

Instancias de Inferentia recomendadas

Las instancias de AWS Inferentia están diseñadas para proporcionar alto rendimiento y rentabilidad para cargas de trabajo de inferencia de modelos de aprendizaje profundo. En concreto, los tipos de

instancias Inf2 utilizan chips de AWS Inferentia y el SDK de AWS Neuron, que está integrado con marcos de machine learning populares, como TensorFlow y PyTorch.

Los clientes pueden usar las instancias de Inf2 para ejecutar aplicaciones de inferencia de machine learning a gran escala, como búsquedas, motores de recomendación, visión artificial, reconocimiento de voz, procesamiento del lenguaje natural, personalización y detección de fraudes, al menor costo en la nube.

> **Note**
>
> El tamaño del modelo debe ser un factor a tener en cuenta para la selección de una instancia. Si su modelo supera la RAM disponible de una instancia, seleccione otro tipo de instancia con memoria suficiente para la aplicación.

- Las instancias Inf2 de Amazon EC2 tienen hasta 16 chips de AWS Inferentia y 100 Gbps de rendimiento de red.

Para obtener más información sobre cómo empezar a utilizar las DLAMI de AWS Inferentia, consulte El chip AWS Inferentia con DLAMI.

Tema siguiente

Instancias de Trainium recomendadas

Instancias de Trainium recomendadas

Las instancias de AWS Trainium están diseñadas para proporcionar alto rendimiento y rentabilidad para cargas de trabajo de inferencia de modelos de aprendizaje profundo. En concreto, los tipos de instancias Trn1 utilizan chips de AWS Trainium y el SDK de AWS Neuron, que está integrado con marcos de machine learning populares, como TensorFlow y PyTorch.

Los clientes pueden usar las instancias de Trn1 para ejecutar aplicaciones de inferencia de machine learning a gran escala, como búsquedas, motores de recomendación, visión artificial, reconocimiento de voz, procesamiento del lenguaje natural, personalización y detección de fraudes, al menor costo en la nube.

> **Note**
>
> El tamaño del modelo debe ser un factor a tener en cuenta para la selección de una instancia. Si su modelo supera la RAM disponible de una instancia, seleccione otro tipo de instancia con memoria suficiente para la aplicación.

- Las instancias Trn1 de Amazon EC2 tienen hasta 16 chips de AWS Trainium y 100 Gbps de rendimiento de red.

Tema siguiente

Instancias de Habana recomendadas

Instancias de Habana recomendadas

Las instancias con aceleradores de Habana están diseñadas para proporcionar alto rendimiento y rentabilidad para las cargas de trabajo de entrenamiento de modelos de aprendizaje profundo. Específicamente, los tipos de instancia DL1 utilizan aceleradores de Habana Gaudi de Habana Labs, una empresa de Intel. Las instancias DL1 son ideales para entrenar modelos de machine learning utilizados en aplicaciones como procesamiento de lenguaje natural, detección y clasificación de objetos, motores de recomendación y percepción de vehículos autónomos.

Las instancias con aceleradores de Habana se configuran con el software Habana SynapseAI y se integran previamente con marcos de machine learning populares, como TensorFlow y PyTorch. Si busca una combinación óptima de rendimiento y precio para entrenar modelos de aprendizaje profundo, considere la posibilidad de instalar instancias con aceleradores de Habana, que ofrecen el menor costo para el entrenamiento.

> **Note**
>
> El tamaño del modelo debe ser un factor a tener en cuenta para la selección de una instancia. Si su modelo supera la RAM disponible de una instancia, seleccione otro tipo de instancia con memoria suficiente para la aplicación.

- Las instancias de Amazon EC2 DL1 tienen hasta ocho aceleradores de Habana Gaudi, 256 GB de memoria de acelerador, 4 TB de almacenamiento NVMe local y 400 Gbps de rendimiento de red.

Si necesita más información sobre cómo empezar a trabajar con las DLAMI, consulte La DLAMI Habana.

Política de compatibilidad de marcos

Los AWS Deep Learning AMI (DLAMI) simplifican la configuración de imágenes para las cargas de trabajo de aprendizaje profundo y están optimizados con los marcos, el hardware, los controladores, las bibliotecas y los sistemas operativos más recientes. En esta página se detalla la política de compatibilidad de marcos de los DLAMI. Para obtener una lista de los DLAMI disponibles, consulte las notas de la versión de DLAMI.

Marcos admitidos

Consulte la siguiente tabla de políticas de compatibilidad de marcos de AWS Deep Learning AMI para comprobar qué marcos y versiones son compatibles en la actualidad.

Consulte la sección End of patch para comprobar durante cuánto tiempo AWS es compatible con las versiones actuales que cuentan con el soporte activo del equipo de mantenimiento del marco de origen. Los marcos y las versiones están disponibles en las DLAMI de un solo marco o en las DLAMI de varios marcos.

> **Note**
>
> En la versión del marco, x.y.z, x se refiere a la versión principal, y se refiere a la versión secundaria y z se refiere a la versión del parche. Por ejemplo, para la versión TensorFlow 2.6.5, la versión principal es la 2, la versión secundaria es la 6 y la versión del parche es la 5.

Consulte las notas de la versión para obtener más información sobre imágenes específicas:

- Notas de la versión de DLAMI de marco único
- Notas de la versión de DLAMI de varios marcos

Preguntas frecuentes

- ¿Qué versiones de marcos incluyen parches de seguridad?
- ¿Qué imágenes publica AWS cuando se lanzan nuevas versiones del marco?
- ¿Qué imágenes tienen nuevas o características SageMaker? AWS

¿Qué versiones de marcos incluyen parches de seguridad?

Si la versión del marco está etiquetada como compatible en la tabla de políticas de compatibilidad del marco de AWS Deep Learning AMI, recibe parches de seguridad.

¿Qué imágenes publica AWS cuando se lanzan nuevas versiones del marco?

Publicamos los nuevos DLAMIs poco después del lanzamiento de las nuevas versiones TensorFlow y PyTorch de su publicación. Esto incluye las versiones principales, las versiones principales y secundarias y major-minor-patch las versiones de los marcos. También actualizamos las imágenes cuando hay nuevas versiones de controladores y bibliotecas disponibles. Para obtener más información sobre el mantenimiento de imágenes, consulte ¿Cuándo finaliza el soporte activo para mi versión de marco?.

¿Qué imágenes tienen nuevas o características SageMaker? AWS

Las nuevas funciones suelen publicarse en la última versión de DLAMis para PyTorch y. TensorFlow Consulte las notas de la versión para ver una imagen específica para obtener más información sobre

las novedades SageMaker o AWS las funciones. Para obtener una lista de los DLAMI disponibles, consulte las notas de la versión de DLAMI. Para obtener más información sobre el mantenimiento de imágenes, consulte ¿Cuándo finaliza el soporte activo para mi versión de marco?.

¿Cómo se define la versión actual en la tabla de marcos compatibles?

La versión actual de la tabla AWS Deep Learning AMI Framework Support Policy AWS hace referencia a la versión más reciente del marco disponible en GitHub. Cada versión más reciente incluye actualizaciones de los controladores, las bibliotecas y los paquetes relevantes de la DLAMI. Para obtener más información sobre el mantenimiento de imágenes, consulte ¿Cuándo finaliza el soporte activo para mi versión de marco?.

¿Qué sucede si estoy ejecutando una versión que no está en la tabla de marcos compatibles?

Si ejecuta una versión que no figura en la tabla de políticas de compatibilidad de marcos de AWS Deep Learning AMI, es posible que no tenga los controladores, las bibliotecas y los paquetes relevantes más actualizados. Para obtener una up-to-date versión posterior, le recomendamos que actualice a uno de los marcos compatibles disponibles con la última DLAMI de su elección. Para obtener una lista de las DLAMI disponibles, consulte las notas de la versión de DLAMI.

¿Las DLAMI son compatibles con las versiones anteriores de? TensorFlow

¿No?. Admitimos la última versión de parche de la última versión principal de cada framework publicada 365 días después de su GitHub lanzamiento inicial, tal y como se indica en la tabla de políticas de soporte de AWS Deep Learning AMI Framework. Para obtener más información, consulte ¿Qué sucede si estoy ejecutando una versión que no está en la tabla de marcos compatibles?

¿Cómo puedo encontrar la última imagen parcheada de una versión de marco compatible?

Para utilizar una DLAMI con la última versión del marco, recupere el ID de DLAMI y utilícelo para lanzar la DLAMI mediante la consola de EC2. Para ver ejemplos de comandos de AWS CLI para recuperar el ID de AWS Deep Learning AMI, consulte la sección Deep Learning Frameworks del catálogo de AWS Deep Learning AMI. AWS Las consultas de ID de la AMI de CLI también se incluyen en las notas de la versión de DLAMI de marco único. La versión del marco que elija debe estar etiquetada como compatible en la tabla de políticas de compatibilidad del marco de AWS Deep Learning AMI.

¿Con qué frecuencia se publican nuevas imágenes?

Proporcionar versiones de parches actualizadas es nuestra máxima prioridad. Creamos imágenes parcheadas de forma rutinaria lo antes posible. Supervisamos las nuevas versiones del framework parcheadas (p. ej. TensorFlow 2.9 a TensorFlow 2.9.1) y nuevas versiones secundarias (p. ej. TensorFlow 2.9 a TensorFlow 2.10) y póngalos disponibles lo antes posible. Cuando TensorFlow se publica una versión existente de con una nueva versión de CUDA, publicamos una nueva DLAMI para esa versión de con soporte para la nueva versión TensorFlow de CUDA.

¿Se instalará el parcheo de mi instancia mientras se ejecute mi carga de trabajo?

No. Las actualizaciones de parches para DLAMI no son actualizaciones “in situ”.

Debe activar una nueva instancia de EC2, migrar las cargas de trabajo y los scripts y, después, desactivar la instancia anterior.

¿Qué ocurre cuando hay disponible una nueva versión del marco parcheada o actualizada?

Consulte con regularidad la página de notas de la versión de su imagen. Le recomendamos que actualice a marcos nuevos, parcheados o actualizados cuando estén disponibles. Para obtener una lista de las DLAMI disponibles, consulte las notas de la versión de DLAMI.

¿Se actualizan las dependencias sin cambiar la versión del marco?

Actualizamos las dependencias sin cambiar la versión del marco. Sin embargo, si una actualización de una dependencia provoca una incompatibilidad, creamos una imagen con una versión diferente. Asegúrese de consultar las notas de la versión de DLAMI para obtener información actualizada sobre las dependencias.

¿Cuándo finaliza el soporte activo para mi versión de marco?

Las imágenes de DLAMI son inmutables. Una vez creadas, no cambian. Hay cuatro razones principales por las que finaliza el soporte activo para una versión de marco:

- Actualizaciones (parche) de la versión del marco
- Parches de seguridad de AWS

- Fecha de finalización del parche (fecha de caducidad)
- Dependencia end-of-support

> **Note**
>
> Debido a la frecuencia con que se actualizan los parches de versión y los parches de seguridad, le recomendamos que consulte con frecuencia la página de notas de la versión de su DLAMI y que la actualice cuando se realicen cambios.

Actualizaciones (parche) de la versión del marco

Si tiene una carga de trabajo de DLAMI basada TensorFlow en la versión 2.7.0 TensorFlow y publica la versión 2.7.1 en adelante, AWS publique una nueva DLAMI con GitHub la versión 2.7.1. TensorFlow Las imágenes anteriores de la versión 2.7.0 ya no se mantienen de forma activa una vez que se publica la nueva imagen de la versión 2.7.1. TensorFlow La DLAMI TensorFlow con 2.7.0 no recibe más parches. A continuación, se actualiza la página de notas de la versión 2.7 TensorFlow de DLAMI con la información más reciente. No hay una página de notas de lanzamiento individual para cada parche menor.

Las nuevas DLAMI creadas debido a las actualizaciones de los parches se designan con un nuevo ID de AMI.

Parches de seguridad de AWS

Si tiene una carga de trabajo basada en una imagen con la versión TensorFlow 2.7.0 y AWS crea un parche de seguridad, se lanza una nueva versión de la DLAMI para la versión 2.7.0. TensorFlow La versión anterior de las imágenes, con la versión TensorFlow 2.7.0, ya no se mantiene de forma activa. Para obtener más información, consulte ¿Se instalará el parcheo de mi instancia mientras se ejecute mi carga de trabajo? Si desea conocer los pasos para encontrar la DLAMI más reciente, consulte ¿Cómo puedo encontrar la última imagen parcheada de una versión de marco compatible?

Las nuevas DLAMI creadas debido a las actualizaciones de los parches se designan con un nuevo ID de AMI.

Fecha de finalización del parche (fecha de caducidad)

Los DLAMIs llegaron a su fecha de finalización del parche 365 días después de la GitHub fecha de lanzamiento.

Para las DLAMI con varios marcos, cuando se actualiza una de las versiones del marco, se requiere una nueva DLAMI con la versión actualizada. La DLAMI con la versión anterior del marco ya no se mantiene activamente.

> **Important**
>
> Hacemos una excepción cuando hay una actualización importante del marco. Por ejemplo, si la versión TensorFlow 1.15 se actualiza a la versión TensorFlow 2.0, seguiremos ofreciendo soporte para la versión más reciente de la TensorFlow 1.15 durante un periodo de dos años a partir de la fecha de GitHub lanzamiento o seis meses después de que el equipo de mantenimiento del framework de origen deje de ofrecer soporte, la fecha que sea anterior.

Dependencia end-of-support

Si está ejecutando una carga de trabajo en una imagen DLAMI TensorFlow 2.7.0 con Python 3.6 y esa versión de Python está end-of-support marcada para, todas las imágenes DLAMI basadas en Python 3.6 dejarán de mantenerse activamente. Del mismo modo, si se marca una versión del sistema operativo como Ubuntu 16.04 end-of-support, todas las imágenes DLAMI que dependan de Ubuntu 16.04 dejarán de mantenerse activamente.

¿Se parchearán las imágenes con versiones de marco que ya no se mantienen activamente?

No. Las imágenes que ya no se mantengan activamente no tendrán nuevas versiones.

¿Cómo utilizo una versión anterior de marco?

Para utilizar una DLAMI con una versión anterior del marco, recupere el ID de DLAMI y utilícelo para lanzar la DLAMI mediante la consola de EC2. Para ver los comandos de la AWS CLI y recuperar el ID de la AMI, consulte la sección Deep Learning Frameworks del catálogo de AMI de aprendizaje profundo de AWS. AWS Las consultas de ID de la AMI de CLI también se incluyen en las notas de la versión de DLAMI de marco único.

¿Cómo puedo mantener la compatibilidad up-to-date con los cambios en los marcos y sus versiones?

Siga up-to-date con los marcos y versiones de DLAMI utilizando la tabla de políticas de soporte de marcos y las AWS Deep Learning AMI notas de la versión de DLAMI.

¿Necesito una licencia comercial para usar el repositorio de Anaconda?

Anaconda adoptó un modelo de licencia comercial para ciertos usuarios. Las DLAMI que se mantienen activamente se han migrado desde el canal Anaconda a la versión de código abierto de Conda (conda-forge), que está disponible para el público.

Lanzamiento y configuración de una DLAMI

Si ha llegado hasta aquí, ya debería tener una buena idea de qué AMI desea lanzar. Si no es así, busque una DLAMI y su hardware, los marcos y la recuperación de ID relacionados en el catálogo de AWS Deep Learning AMI o consulte las notas de la versión actual e histórica de DLAMI en Notas de la versión de DLAMI.

También debe saber el tipo de instancia y la región que va a elegir. En caso contrario, consulte Selección del tipo de instancia para DLAMI.

> **Note**
>
> Usaremos p3.16xlarge como el tipo de instancia predeterminado en los ejemplos. Sustitúyalo por el tipo de instancia que tenga pensado.

> **Important**
>
> Si tiene previsto utilizar Elastic Inference, debe completar la configuración de Elastic Inference antes de lanzar su DLAMI.

Temas

Paso 1: lanzamiento de una DLAMI

> **Note**
>
> En este tutorial, puede que hagamos referencias específicas a la AMI de aprendizaje profundo (Ubuntu 18.04). Incluso aunque seleccione otra DLAMI, debería ser capaz de seguir esta guía.

1. Busque el ID de su DLAMI
2. Lance una instancia de Amazon EC2 desde la DLAMI

   Utilizará la consola de Amazon EC2. Siga las instrucciones detalladas en Lanzamiento desde la consola de Amazon EC2

   > **Tip**
   >
   > Opción de la CLI: si decide implementar una DLAMI mediante la AWS CLI, necesitará el ID de la AMI, la región y el tipo de instancia, y la información sobre el token de seguridad. Asegúrese de que tiene preparado el ID de la AMI y el de la instancia. Si aún no ha configurado la AWS CLI, hágalo primero con la guía de instalación de la interfaz de línea de comandos de AWS.

3. Una vez que haya finalizado los pasos de una de estas opciones, espere a que la instancia esté lista. Este proceso suele tardar unos minutos. Puede comprobar el estado de la instancia en la Consola de EC2.

Recupere el ID de DLAMI

Cada AMI tiene un identificador único (ID). Puede consultar el ID de la DLAMI de su elección con la interfaz de línea de comandos de AWS (AWS CLI). Si aún no tiene la AWS CLI instalada, consulte Introducción a la AWS CLI.

> **Note**
>
> Recordatorio: en el catálogo de AWS Deep Learning AMI puede encontrar todas las DLAMI y sus procesadores/aceleradores, sistemas operativos, arquitectura informática, familias de

instancias de Amazon EC2 recomendadas, estado de soporte y consultas de recuperación de ID. Consulte también las notas de la versión de DLAMI en Notas de la versión de DLAMI para obtener información adicional (controladores, versiones de Python, tipo de Amazon EBS).

1. Asegúrese de que sus credenciales de AWS estén configuradas.

```
aws configure
```

2. Utilice el comando siguiente para recuperar el ID de su DLAMI o busque la consulta iincluida en el catálogo de AWS Deep Learning AMI.

```
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**
>
> Puede especificar una versión de lanzamiento para un marco determinado u obtener la última versión sustituyendo el número de versión por un signo de interrogación.

3. El resultado debería tener un aspecto similar al siguiente:

```
ami-094c089c38ed069f2
```

Copie este ID de DLAMI y pulse q para salir del indicador.

Paso siguiente

Lanzamiento desde la consola de Amazon EC2

Lanzamiento desde la consola de Amazon EC2

> **Note**
>
> Para iniciar una instancia con Elastic Fabric Adapter (EFA), siga estos pasos.

1. Abra la consola de EC2.
2. Tome nota de la región actual en la parte superior del panel de navegación. Si no es la Región de AWS que desea, cambie esta opción antes de continuar. Para obtener más información, consulte Regiones de EC2.
3. Elija Launch Instance (Lanzar instancia).
4. Introduzca un nombre para la instancia y seleccione la DLAMI que le resulte más adecuada.
   a. Busque una DLAMI existente en Mis AMI o seleccione Inicio rápido.
   b. Busque por ID de DLAMI. Examine las opciones y seleccione la que desee.
5. Elija un tipo de instancia. Puede encontrar las familias de instancias recomendadas para su DLAMI en el catálogo de AWS Deep Learning AMI. Para obtener recomendaciones generales sobre los tipos de instancias de DLAMI, consulte Instance Selection.

   > **Note**
   >
   > Si desea utilizar Elastic Inference (EI), haga clic en Configurar los detalles de la instancia, seleccione Add an Amazon EI accelerator y, a continuación, seleccione el tamaño del acelerador de Amazon EI.

6. Elija Launch Instance (Lanzar instancia).

> **Tip**
>
> Consulte Get Started with Deep Learning Using the AWS Deep Learning AMI para obtener una guía con capturas de pantalla.

Paso siguiente

Paso 2: conexión a la DLAMI

Paso 2: conexión a la DLAMI

Conéctese a la DLAMI que ha lanzado desde un cliente (Windows, MacOS o Linux). Para obtener más información, consulte Conexión a la instancia de Linux en la Guía del usuario de Amazon EC2 para instancias de Linux.

Tenga a mano una copia del comando de inicio de sesión de SSH si desea realizar la configuración de Jupyter tras iniciar sesión. Utilizará una variante del comando para conectarse a la página web de Jupyter.

Paso siguiente

Paso 3: comprobación de la DLAMI

Paso 3: comprobación de la DLAMI

En función de la versión de la DLAMI, dispone de diferentes opciones de prueba:

- AMI de aprendizaje profundo con Conda : vaya a Uso de la AMI de aprendizaje profundo con Conda.
- AMI base de aprendizaje profundo : consulte la documentación de instalación del marco que desee.

También puede crear un bloc de notas de Jupyter, probar los tutoriales o comenzar a escribir código en Python. Para obtener más información, consulte Configuración de un servidor de cuadernos de Jupyter.

Paso 4: administre su instancia de DLAMI

Mantenga siempre actualizado su sistema operativo y otro software instalado y aplique los parches y actualizaciones en cuanto estén disponibles.

Si utiliza Amazon Linux o Ubuntu, cuando inicie sesión en su DLAMI, se le notificará cuando haya actualizaciones disponibles y verá las instrucciones de actualización. Para obtener más información sobre el mantenimiento de Amazon Linux, consulte Actualizar software en la instancia de Amazon Linux. Para las instancias de Ubuntu, consulte la documentación oficial de Ubuntu.

En Windows, compruebe Windows Update con regularidad para ver si hay actualizaciones de seguridad y de software. Si lo prefiere, aplique las actualizaciones automáticamente.

> **Important**
>
> Para obtener información sobre las vulnerabilidades de Meltdown y Spectre y cómo aplicar parches a su sistema operativo para abordarlas, consulte el boletín de seguridad de AWS-2018-013.

Eliminación

Cuando no necesite la DLAMI, puede detenerla o terminarla para evitar gastos. Si se detiene una instancia, se conservará para que pueda reanudarla más adelante. Las configuraciones, los archivos y demás información no volátil se almacenan en un volumen en Amazon S3. Se le cobrará una pequeña cuota de S3 por conservar el volumen mientras la instancia esté detenida, pero no se le cobrará por los recursos informáticos mientras se encuentre en ese estado. Cuando inicie la instancia de nuevo, se montará ese volumen y sus datos estarán disponibles. Si termina una instancia, se borrará y no podrá volver a iniciarla. En realidad, los datos todavía se encuentran en S3, por lo que, para evitar nuevos cargos, debe eliminar también el volumen. Para obtener más instrucciones, consulte Terminar la instancia en la Guía del usuario de Amazon EC2 para instancias de Linux.

Configuración de un servidor de cuadernos de Jupyter

Un servidor de cuadernos de Jupyter permite crear y ejecutar cuadernos de Jupyter desde la instancia de DLAMI. Con los cuadernos de Jupyter, puede realizar experimentos de machine learning (ML) para entrenamiento e inferencia mientras utiliza la infraestructura de AWS y accede a los paquetes integrados en la DLAMI. Para obtener más información sobre los cuadernos de Jupyter, consulte la documentación del cuaderno de Jupyter.

Para configurar un cuaderno de Jupyter:

- Configure el servidor de cuadernos de Jupyter en la instancia de la DLAMI de Amazon EC2.
- Configure el cliente para poder conectarse al servidor de cuadernos de Jupyter. Dispone de instrucciones de configuración para clientes Windows, macOS y Linux.
- Pruebe la configuración iniciando sesión en el servidor de cuadernos de Jupyter.

Para completar los pasos necesarios para configurar un Jupyter, siga las instrucciones de los siguientes temas. Una vez que haya configurado un servidor de cuadernos de Jupyter, consulte

Ejecución de los tutoriales del cuaderno de Jupyter para obtener información sobre cómo ejecutar los cuadernos de ejemplo que se incluyen en la DLAMI.

Temas

Protección del servidor de Jupyter

Aquí configuraremos Jupyter con SSL y una contraseña personalizada.

Conéctese a la instancia de Amazon EC2 y, a continuación, complete el siguiente procedimiento.

Configuración del servidor de Jupyter

1. Jupyter proporciona una utilidad de contraseñas. Ejecute el siguiente comando y escriba la contraseña que desee en el símbolo del sistema.

   ```
   $ jupyter notebook password
   ```

   El resultado tendrá un aspecto similar a este:

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

2. Cree un certificado SSL autofirmado. Siga las instrucciones para especificar la configuración regional más adecuada para sus necesidades. Debe introducir `.` si desea dejar un mensaje en blanco. Sus respuestas no afectarán a la funcionalidad del certificado.

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

> **Note**
>
> Tal vez le convenga crear un certificado normal que esté firmado por un tercero y que no haga que el navegador muestre una advertencia de seguridad. Este proceso es mucho más complejo. Consulte la documentación de Jupyter para obtener más información.

Paso siguiente

Inicio del servidor de cuadernos de Jupyter

Inicio del servidor de cuadernos de Jupyter

Ahora puede iniciar el servidor de Jupyter iniciando sesión en la instancia y ejecutando el siguiente comando que utiliza el certificado SSL que ha creado en el paso anterior.

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

Con el servidor iniciado, puede conectarse a él desde el equipo cliente través de un túnel SSH. Cuando se ejecute el servidor, verá un resultado de Jupyter que confirma que el servidor está en ejecución. En este punto, no tenga en cuenta el aviso de que puede obtener acceso al servidor a través de una dirección URL de localhost, ya que eso no funcionará hasta que cree el túnel.

> **Note**
>
> Jupyter se encargará de cambiar de entorno por usted cuando cambie de plataforma con la interfaz web de Jupyter. Puede encontrar más información al respecto en Cambio de entorno con Jupyter.

Paso siguiente

Configuración del cliente para conectarse con el servidor de Jupyter

Configuración del cliente para conectarse con el servidor de Jupyter

Después de configurar el cliente para que se conecte con el servidor de cuadernos de Jupyter, puede crear cuadernos en el servidor y obtener acceso a ellos desde su espacio de trabajo, así como ejecutar código de aprendizaje profundo en el servidor.

Para obtener información sobre la configuración, elija uno de los siguientes enlaces.

Temas

- Configuración de un cliente Windows
- Configurar un cliente Linux o macOS

Configuración de un cliente Windows

Prepare

Asegúrese de tener la siguiente información, ya que la necesitará para configurar el túnel de SSH:

- El nombre de DNS público de la instancia de Amazon EC2. Puede encontrar el nombre DNS público en la consola de EC2.
- El par de claves del archivo de clave privada. Para obtener más información sobre el acceso al par de claves, consulte Pares de claves de Amazon EC2 en la Guía del usuario de Amazon EC2 para instancias de Linux.

Uso de cuadernos de Jupyter desde un cliente Windows

Consulte estas guías sobre cómo conectarse a la instancia de Amazon EC2 desde un cliente de Windows.

1. Solución de problemas con la conexión a la instancia
2. Conexión a la instancia Linux desde Windows utilizando PuTTY

Para crear un túnel con un servidor de Jupyter en ejecución, el enfoque recomendado es instalar Git Bash en su cliente Windows y seguir después las instrucciones del cliente Linux/macOS. Sin embargo, puede utilizar el enfoque que desee para abrir un túnel SSH con mapeo de puertos. Consulte la documentación de Jupyter para obtener más información.

Paso siguiente

Configurar un cliente Linux o macOS

Configurar un cliente Linux o macOS

1. Abra un terminal .

2. Ejecute el siguiente comando para reenviar todas las solicitudes del puerto local 8888 al puerto 8888 de la instancia remota de Amazzon EC2. Actualice el comando sustituyendo la ubicación de la clave para obtener acceso a la instancia de Amazon EC2 y al nombre DNS público de esa instancia. Tenga en cuenta que para una AMI de Amazon Linux el nombre de usuario es `ec2-user` en lugar de `ubuntu`.

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

   Este comando abre un túnel entre el cliente y la instancia remota de EC2 que está ejecutando el servidor de cuadernos de Jupyter.

Paso siguiente

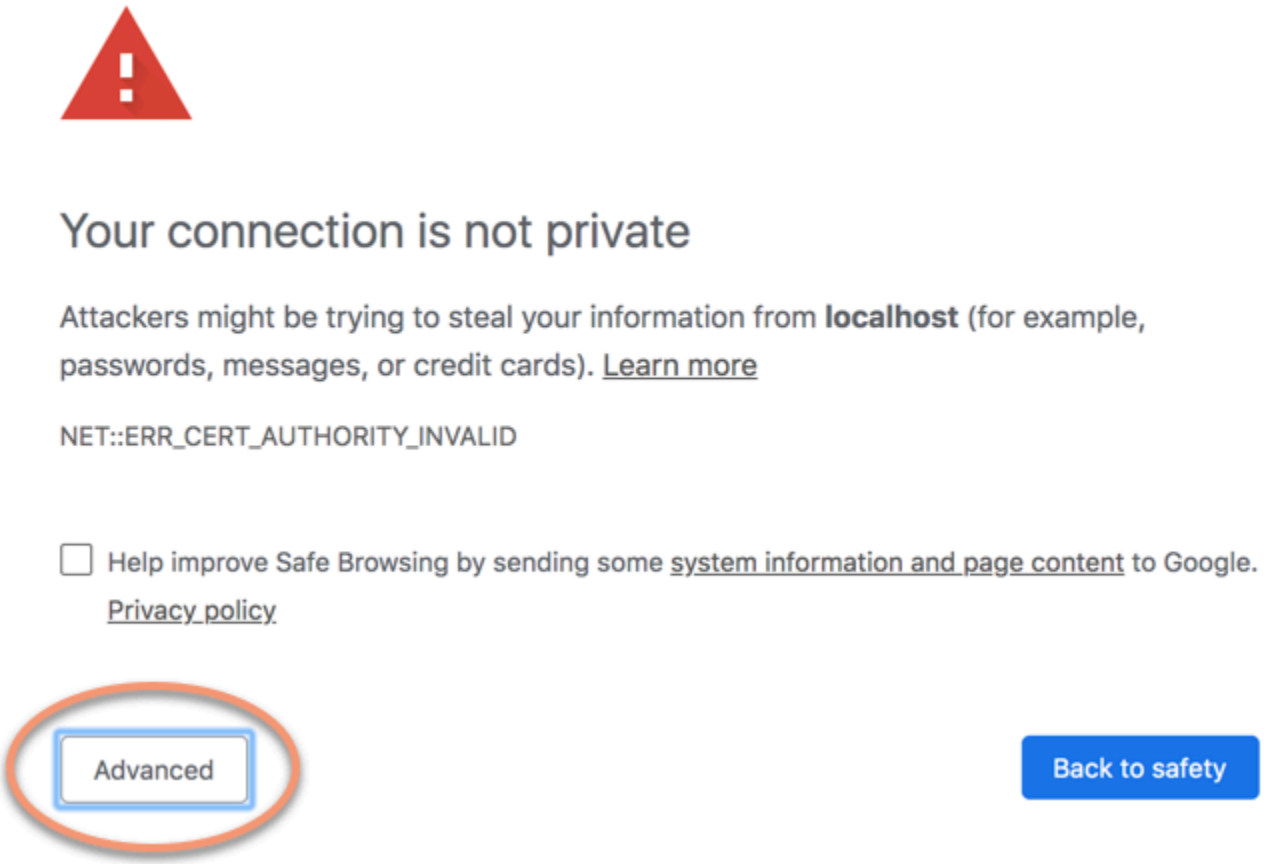
Prueba de la conexión iniciando sesión en el servidor de cuadernos de Jupyter

Prueba de la conexión iniciando sesión en el servidor de cuadernos de Jupyter

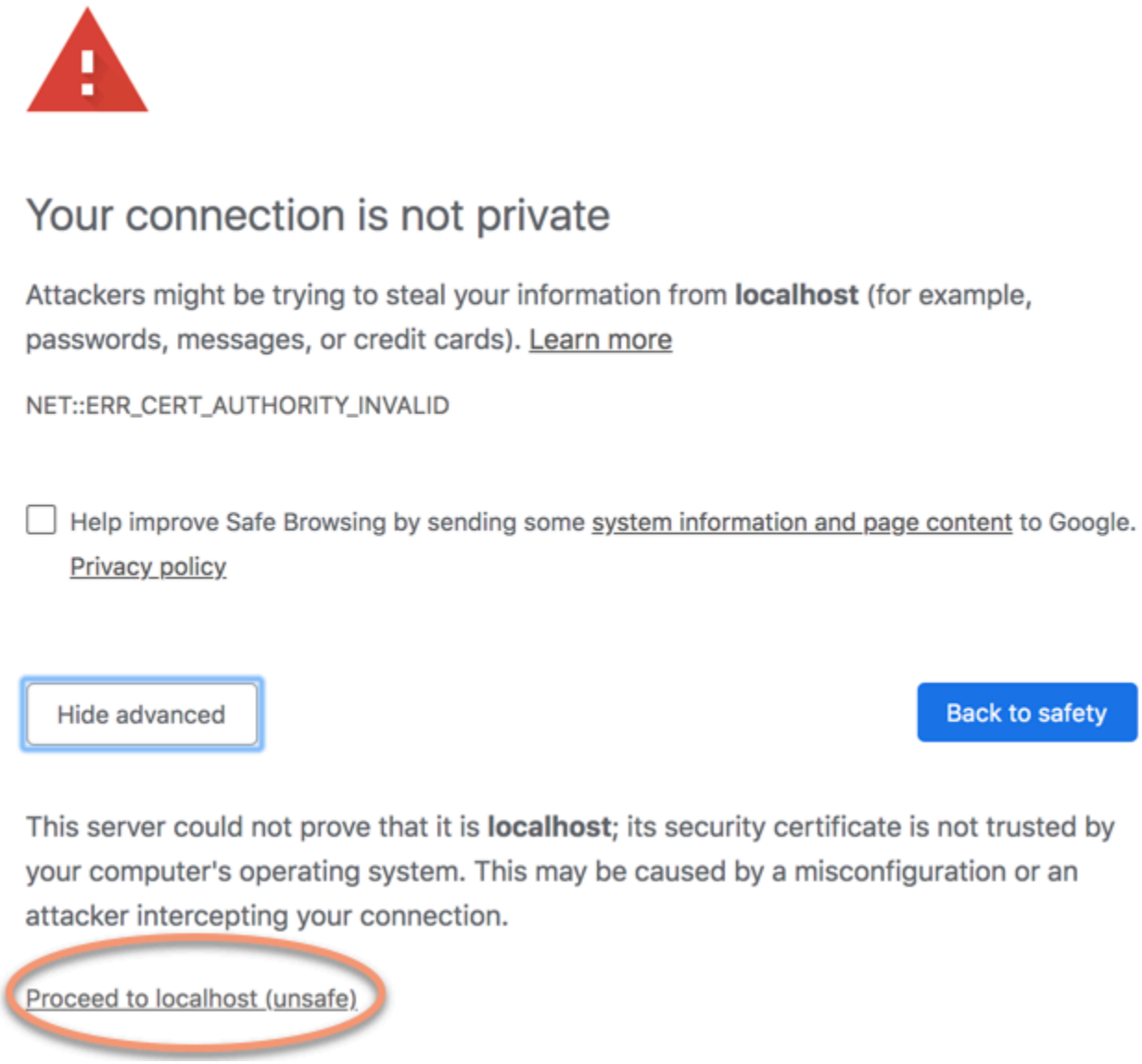
Ahora ya puede iniciar sesión en el servidor de cuadernos de Jupyter.

El siguiente paso consiste en probar la conexión con el servidor a través del navegador.

1. Escriba la siguiente dirección URL en la barra de direcciones del navegador o haga clic en este enlace: https://localhost:8888
2. Con un certificado SSL autofirmado, el navegador emitirá un aviso y le recomendará que abandone el sitio web.



Como lo ha configurado usted, puede continuar sin problema. Dependiendo del navegador, aparece un botón "avanzadas", "mostrar detalles" o similar.



Haga clic en este botón y, a continuación, haga clic en el enlace "ir a localhost". Si la conexión se realiza correctamente, aparecerá la página web del servidor de cuadernos de Jupyter. En este momento, se le pedirá la contraseña que configuró anteriormente.

Ahora ya dispone de acceso al servidor de cuadernos de Jupyter que se ejecuta en la DLAMI. Puede crear cuadernos nuevos o ejecutar los Tutoriales proporcionados.

Uso de una DLAMI

Temas

En las secciones siguientes se describe cómo se puede utilizar la AMI de aprendizaje profundo (DLAMI) para cambiar de entorno y ejecutar código de muestra desde cada uno de los marcos de trabajo y ejecutar Jupyter para poder probar distintos tutoriales de bloc de notas.

Uso de la AMI de aprendizaje profundo con Conda

Temas

Introducción a la AMI de aprendizaje profundo con Conda

Conda es un sistema de código abierto para la administración de paquetes y del entorno que se ejecuta en Windows, macOS y Linux. Conda instala, ejecuta y actualiza rápidamente paquetes y sus dependencias. Conda le permite crear, guardar y cargar entornos en el equipo local, así como alternar entre ellos, con suma facilidad.

La AMI de aprendizaje profundo con Conda se ha configurado de forma que se pueda alternar fácilmente entre los entornos de aprendizaje profundo. Las siguientes instrucciones le orientan acerca de algunos comandos básicos con `conda`. También le ayudan a verificar que la importación básica del marco de trabajo funciona correctamente, y que puede ejecutar un par de operaciones

sencillas con este. Luego puede pasar a tutoriales más exhaustivos incluidos con la DLAMI o a los ejemplos de marcos de trabajo que puede encontrar en el sitio del proyecto de cada uno de los marcos de trabajo.

Inicio de sesión en su DLAMI

Después de iniciar sesión en el servidor, verá un “mensaje del día” (MOTD) del servidor que describe varios comandos de Conda que puede utilizar para alternar entre los distintos marcos de trabajo de aprendizaje profundo. A continuación se muestra un MOTD de ejemplo. Su MOTD (mensaje del día) específico puede variar a medida que se publican nuevas versiones de la DLAMI.

> **Note**
>
> Al principio de la versión 28, ya no incluimos los entornos CNTK, Caffe, Caffe2, Theano, Chainer y Keras Conda. AWS Deep Learning AMI Las versiones anteriores AWS Deep Learning AMI que contienen estos entornos seguirán estando disponibles. Sin embargo, solo proporcionaremos actualizaciones a estos entornos si la comunidad de código abierto publica correcciones de seguridad para estos marcos.

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

Cada uno de los comandos de Conda tiene el siguiente formato:

`source activate framework_python-version`

Por ejemplo, es posible que vea `for MXNet(+Keras1) with Python3 (CUDA 10.1) _____ source activate mxnet_p36`, lo que significa que el entorno tiene MXNet, Keras 1, Python 3 y CUDA 10.1. Para activar este entorno, el comando que utilizaría es:

```
$ source activate mxnet_p36
```

Inicie el TensorFlow entorno

> **Note**
>
> Cuando lance su primer entorno Conda, tenga paciencia mientras se carga. La AMI de aprendizaje profundo con Conda instala automáticamente la versión más optimizada del marco de trabajo para su instancia EC2 tras la primera activación del marco de trabajo. No debe esperar retrasos posteriores.

1. Active el entorno TensorFlow virtual para Python 3.

   ```
   $ source activate tensorflow_p37
   ```

2. Inicie el terminal de iPython.

```
(tensorflow_37)$ ipython
```

3. Ejecute un TensorFlow programa rápido.

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

Debería aparecer "Hello, Tensorflow!"

Tema siguiente

Ejecución de los tutoriales del cuaderno de Jupyter

Cambie al entorno PyTorch Python 3

Si sigue en la consola de iPython, utilice `quit()` y prepárese para cambiar de entorno.

- Active el entorno PyTorch virtual para Python 3.

```
$ source activate pytorch_p36
```

Pruebe algún PyTorch código

Para probar la instalación, utilice Python para escribir PyTorch código que cree e imprima una matriz.

1. Inicie el terminal de iPython.

```
(pytorch_p36)$ ipython
```

2. Importar PyTorch.

```
import torch
```

   Es posible que vea un mensaje de advertencia sobre un paquete de terceros. Puede omitirlo.

3. Cree una matriz de 5x3 con los elementos inicializados de forma aleatoria. Imprima la matriz.

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

Verifique el resultado.

```
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]])
```

Cambie al entorno MXNet Python 3

Si sigue en la consola de iPython, utilice `quit()` y prepárese para cambiar de entorno.

- Active el entorno virtual de MXNet para Python 3.

```
$ source activate mxnet_p36
```

Pruebe código de MXNet

Para probar la instalación, utilice Python para escribir código de MXNet que cree e imprima una matriz mediante la API de `NDArray`. Para obtener más información, consulte NDArray API.

1. Inicie el terminal de iPython.

```
(mxnet_p36)$ ipython
```

2. Importe MXNet.

```
import mxnet as mx
```

Es posible que vea un mensaje de advertencia sobre un paquete de terceros. Puede omitirlo.

3. Cree una matriz de 5x5, una instancia de la clase `NDArray`, con los elementos inicializados a 0. Imprima la matriz.

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

Verifique el resultado.

```
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)
```

Puede encontrar más ejemplos de MXNet en la sección de tutoriales de MXNet.

Eliminación de entornos

Nota: Si se queda sin espacio en la DLAMI, puede desinstalar los paquetes de Conda que no utilice:

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

Uso de la AMI base de aprendizaje profundo

Uso de la AMI base de aprendizaje profundo

La AMI base incluye una plataforma básica de controladores de GPU y bibliotecas de aceleración que le permiten implementar su propio entorno de aprendizaje profundo personalizado. De forma predeterminada, la AMI está configurada con el entorno NVIDIA CUDA 11.0. También puede cambiar de una versión a otra de CUDA. Consulte las siguientes instrucciones para saber cómo hacerlo.

Configuración de las versiones de CUDA

Puede verificar la versión de CUDA ejecutando el programa `nvcc` de NVIDIA.

```
nvcc --version
```

Puede seleccionar y verificar una versión determinada de CUDA con los siguientes comandos bash.

```
sudo rm /usr/local/cuda
```

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

Para obtener más información, consulte las notas de la versión de DLAMI base.

Ejecución de los tutoriales del cuaderno de Jupyter

Con el código fuente de cada uno de los proyectos de aprendizaje profundo se incluyen tutoriales y ejemplos que, en la mayoría de los casos, se ejecutarán en cualquier DLAMI. Si eligió la AMI de aprendizaje profundo con Conda, obtendrá la ventaja añadida de unos tutoriales seleccionados específicamente, preconfigurados y listos para probarlos.

Important

Para ejecutar los tutoriales del cuaderno de Jupyter instalados en la DLAMI, consulte Configuración de un servidor de cuadernos de Jupyter.

Cuando el servidor de Jupyter esté en funcionamiento, puede ejecutar los tutoriales a través de un navegador web. Si está ejecutando la AMI de aprendizaje profundo con Conda o si ha configurado entornos de Python, puede cambiar los kernel de Python desde la interfaz del cuaderno de Jupyter. Seleccione el kernel adecuado antes de ejecutar un tutorial específico para un marco de trabajo. Se proporcionan ejemplos adicionales para los usuarios de la AMI de aprendizaje profundo con Conda.

Note

Muchos tutoriales requieren módulos de Python adicionales que puede que no estén configurados en su DLAMI. Si obtiene un error como `"xyz module not found"`, inicie sesión en la DLAMI, active el entorno tal como se ha descrito anteriormente y, a continuación, instale los módulos necesarios.

Tip

Los tutoriales y los ejemplos de aprendizaje profundo suelen utilizar una o varias GPU. Si su tipo de instancia no cuenta con una GPU, puede que necesite cambiar el código de algunos de los ejemplos para que se ejecuten.

Navegación por los tutoriales instalados

Una vez que haya iniciado sesión en el servidor de Jupyter y tenga acceso al directorio de los tutoriales (solo en la AMI de aprendizaje profundo con Conda), verá carpetas de tutoriales para cada nombre de marco de trabajo. Si no ve ningún marco de trabajo en la lista, significa que no hay tutoriales para ese marco de trabajo en la DLAMI actual. Haga clic en el nombre del marco de trabajo para ver los tutoriales de la lista y, a continuación, haga clic en un tutorial para lanzarlo.

La primera vez que ejecute un bloc de notas en la AMI de aprendizaje profundo con Conda, deberá indicar el entorno que desea utilizar. Se le pedirá que lo seleccione en una lista. El nombre de cada entorno sigue este patrón:

`Environment (conda_framework_python-version)`

Por ejemplo, es posible que vea `Environment (conda_mxnet_p36)`, lo que significa que el entorno tiene MXNet y Python 3. Otra variante sería `Environment (conda_mxnet_p27)`, que significa que el entorno tiene MXNet y Python 2.

> **Tip**
>
> Si desea saber qué versión de CUDA está activa, puede verla en el “mensaje del día” (MOTD) que aparece la primera vez que se inicia sesión en la DLAMI.

Cambio de entorno con Jupyter

Si decide probar un tutorial para otro marco de trabajo, asegúrese de comprobar cuál es el kernel que se está ejecutando actualmente. Esta información se puede ver en la esquina superior derecha de la interfaz de Jupyter, justo debajo del botón de cerrar sesión. Puede cambiar el kernel en cualquier bloc de notas abierto haciendo clic en la opción Kernel del menú de Jupyter, seguido de Change Kernel y, a continuación, haciendo clic en el entorno correspondiente al bloc de notas que esté ejecutando.

En este punto, tendrá que volver a ejecutar todas las celdas, debido a que un cambio en el kernel borrará el estado de cualquier elemento que se haya ejecutado anteriormente.

> **Tip**
>
> Cambiar de marco de trabajo puede ser divertido y educativo, pero puede hacer que se agote la memoria. Si comienzan a aparecer errores, examine la ventana de terminal en

la que se está ejecutando el servidor de Jupyter. Aquí hay mensajes útiles y registros de errores, y es posible que veas un out-of-memory error. Para solucionar este problema, vaya a la página de inicio del servidor de Jupyter, haga clic en la pestaña Running y, a continuación, haga clic en Shutdown para cada uno de los tutoriales que probablemente sigan ejecutándose en segundo plano y estén consumiendo toda la memoria.

Tema siguiente

Para obtener más ejemplos y código de muestra para cada marco de trabajo, haga clic en Next o continúe en Apache MXNet (incubating).

Tutoriales

Los siguientes tutoriales muestran cómo utilizar el software de la AMI de aprendizaje profundo con Conda.

Temas

- Tutoriales de 10 minutos
- Activación de los marcos de trabajo
- Depuración y visualización
- Entrenamiento distribuido
- Elastic Fabric Adapter
- Monitorización y optimización de GPU
- El chip AWS Inferentia con DLAMI
- El DLAMI de Graviton
- La DLAMI Habana
- Inferencia
- Uso de marcos de trabajo con ONNX
- Distribución de modelos

Tutoriales de 10 minutos

- Lanza un AWS Deep Learning AMI (en 10 minutos)
- Entrene un modelo de aprendizaje profundo con DLC en Amazon EC2 (en 10 minutos)

Activación de los marcos de trabajo

A continuación, se muestran los marcos de trabajo de aprendizaje profundo instalados en la AMI de aprendizaje profundo con Conda. Haga clic en un marco de trabajo para obtener información acerca de cómo activarlo.

Temas

Apache MXNet (incubating)

Activación de Apache MXNet (incubación)

En este tutorial se muestra cómo activar MXNet en una instancia que ejecuta la AMI de aprendizaje profundo con Conda (DLAMI en Conda) y ejecutar un programa de MXNet.

Cuando se lanza un paquete Conda estable de un marco de trabajo, se prueba y se preinstala en la DLAMI. Si desea ejecutar la última compilación nocturna sin probar, puede Instalación de una compilación nocturna de MXNet (experimental) manualmente.

Para ejecutar MXNet en la DLAMI con Conda

1. Para activar el marco de trabajo, abra una instancia de Amazon Elastic Compute Cloud (Amazon EC2) de la DLAMI con Conda.
   - Para MXNet y Keras 2 en Python 3 con CUDA 9.0 y MKL-DNN, ejecute este comando:

```
$ source activate mxnet_p36
```

- Para MXNet y Keras 2 en Python 2 con CUDA 9.0 y MKL-DNN, ejecute este comando:

```
$ source activate mxnet_p27
```

2. Inicie el terminal de iPython.

```
(mxnet_p36)$ ipython
```

3. Ejecute un programa rápido en MXNet. Cree una matriz de 5x5, una instancia de la clase `NDArray`, con los elementos inicializados a 0. Imprima la matriz.

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

4. Verifique el resultado.

```
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)
```

Instalación de una compilación nocturna de MXNet (experimental)

Puede instalar la última compilación de MXNet en uno o en ambos entornos MXNet Conda en su AMI de aprendizaje profundo con Conda.

Para instalar MXNet desde una compilación nocturna

1. • Para el entorno de MXNet para Python 3, ejecute este comando:

```
$ source activate mxnet_p36
```

- Para el entorno de MXNet para Python 2, ejecute este comando:

```
$ source activate mxnet_p27
```

2. Elimine el entorno de MXNet instalado actualmente.

> **Note**
>
> En los pasos restantes se da por hecho que está utilizando el entorno `mxnet_p36`.

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

3. Instale la compilación nocturna más reciente de MXNet.

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

4. Para verificar que ha instalado correctamente la compilación nocturna más reciente, inicie el terminal de IPython y compruebe la versión de MXNet.

```
(mxnet_p36)$ ipython
```

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

El resultado debería mostrar la última versión estable de MXNet.

Más tutoriales

Puede encontrar más tutoriales en la carpeta de tutoriales de la AMI de aprendizaje profundo con Conda que se encuentra en el directorio principal de la DLAMI.

1. Utilice Apache MXNet (en incubación) para la inferencia con un modelo 50 ResNet
2. Uso de Apache MXNet (Incubating) para inferencia con un modelo ONNX
3. Model Server for Apache MXNet (MMS)

Si desea ver más tutoriales y ejemplos, consulte la documentación oficial de Python del marco de trabajo, la API de Python para MXNet o el sitio web de Apache MXNet.

Caffe2

> **Note**
>
> Ya no incluimos los entornos CNTK, Caffe, Caffe 2 y Theano Conda en la AWS Deep Learning AMI a partir de la versión v28. Las versiones anteriores AWS Deep Learning AMI que contienen estos entornos seguirán estando disponibles. Sin embargo, solo proporcionaremos actualizaciones a estos entornos si la comunidad de código abierto publica correcciones de seguridad para estos marcos.

Tutorial de Caffe2

Para activar el marco de trabajo, siga estas instrucciones en la AMI de aprendizaje profundo con Conda.

Solo existe la opción de Python 2 con CUDA 9 con cuDNN 7:

```
$ source activate caffe2_p27
```

Inicie el terminal de iPython.

```
(caffe2_p27)$ ipython
```

Ejecute un programa Caffe2 rápido.

```
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)
```

Debería ver las matrices aleatorias iniciales de numpy impresas y a continuación, cargadas en un blob de Caffe2. Observe que después de la carga son las mismas.

Más tutoriales

Si desea ver más tutoriales y ejemplos, consulte la documentación oficial de Python del marco de trabajo, la API de Python para Caffe2 y el sitio web de Caffe2.

Chainer

> **Note**
>
> Desde la versión v28, ya no incluimos los entornos Chainer Conda en AWS Deep Learning AMI . Las versiones anteriores AWS Deep Learning AMI que contienen estos entornos seguirán estando disponibles. Sin embargo, solo proporcionaremos actualizaciones a estos entornos si la comunidad de código abierto publica correcciones de seguridad para estos marcos.

Chainer es un marco de trabajo flexible basado en Python para escribir de forma sencilla e intuitiva arquitecturas de red neuronal complejas. Chainer facilita la utilización de instancias de varias GPU para entrenamiento. Chainer además registra automáticamente los resultados, representa gráficamente pérdidas y precisión y produce una salida para visualizar la red neuronal con un grafo computacional. Se incluye en la AMI de aprendizaje profundo con Conda (DLAMI con Conda).

Activación de Chainer

1. Conéctese a la instancia que ejecuta la AMI de aprendizaje profundo con Conda. Consulte la the section called “Selección de una instancia” o la documentación de Amazon EC2 acerca de cómo seleccionar una instancia o conectarse a ella.
2.
   - Active el entorno de Chainer de Python 3:

     ```
     $ source activate chainer_p36
     ```

   - Active el entorno de Chainer de Python 2:

     ```
     $ source activate chainer_p27
     ```

3. Inicie el terminal de iPython:

   ```
   (chainer_p36)$ ipython
   ```

4. Pruebe la importación de Chainer para comprobar que está funcionando correctamente:

```
import chainer
```

Es posible que vea algunos mensajes de advertencia, pero ningún error.

Más información

- Pruebe los tutoriales para Chainer.
- La carpeta de ejemplos de `Chainer` dentro del origen que ha descargado anteriormente contiene más ejemplos. Pruébelos para comprobar su desempeño.
- Para obtener más información sobre Chainer, consulte el sitio web de documentación de Chainer.

CNTK

> **Note**
>
> Ya no incluimos los entornos CNTK, Caffe, Caffe 2 y Theano Conda en la AWS Deep Learning AMI a partir de la versión v28. Las versiones anteriores AWS Deep Learning AMI que contienen estos entornos seguirán estando disponibles. Sin embargo, solo proporcionaremos actualizaciones a estos entornos si la comunidad de código abierto publica correcciones de seguridad para estos marcos.

Activación de CNTK

En este tutorial se muestra cómo activar CNTK en una instancia que ejecuta la AMI de aprendizaje profundo con Conda (DLAMI en Conda) y ejecutar un programa de CNTK.

Cuando se lanza un paquete Conda estable de un marco de trabajo, se prueba y se preinstala en la DLAMI. Si desea ejecutar la última compilación nocturna sin probar, puede Instalar una compilación nocturna de CNTK (experimental) manualmente.

Para ejecutar CNTK en la DLAMI con Conda

1. Para activar CNTK, abra una instancia de Amazon Elastic Compute Cloud (Amazon EC2) de la DLAMI con Conda.
   - Para Python 3 con CUDA 9 con cuDNN 7:

```
$ source activate cntk_p36
```

- Para Python 2 con CUDA 9 con cuDNN 7:

```
$ source activate cntk_p27
```

2. Inicie el terminal de iPython.

```
(cntk_p36)$ ipython
```

3. • Si tiene una instancia de CPU, ejecute este programa rápido de CNTK.

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

Debería ver la versión de CNTK y, a continuación, la salida de una matriz de treses de 2x3.

- Si tiene una instancia de GPU, puede probarla con el siguiente ejemplo de código. Si CNTK puede acceder a la GPU, el resultado debería ser `True`.

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

Instalar una compilación nocturna de CNTK (experimental)

Puede instalar la última compilación de CNTK en uno o en ambos entornos CNTK Conda en su AMI de aprendizaje profundo con Conda.

Para instalar CNTK desde una compilación nocturna

1. • Para CNTK y Keras 2 en Python 3 con CUDA 9.0 y MKL-DNN, ejecute este comando:

```
$ source activate cntk_p36
```

- Para CNTK y Keras 2 en Python 2 con CUDA 9.0 y MKL-DNN, ejecute este comando:

```
$ source activate cntk_p27
```

2. En los pasos restantes se da por hecho que está utilizando el entorno `cntk_p36`. Elimine el entorno de CNTK instalado actualmente:

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

3. Para instalar la compilación nocturna de CNTK, primero debe encontrar la versión que desea instalar en el sitio web de compilaciones nocturnas de CNTK.

4.
   - (Opción para instancias de GPU): para instalar la compilación, usaría lo siguiente, sustituyendo la compilación deseada:

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

     Sustituya la URL del comando anterior por la versión de la GPU de su entorno de Python actual.

   - (Opción para instancias de CPU): para instalar la compilación, usaría lo siguiente, sustituyendo la compilación deseada:

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

     Sustituya la URL del comando anterior por la versión de la CPU de su entorno de Python actual.

5. Para verificar que ha instalado correctamente la última compilación nocturna, inicie el terminal de IPython y compruebe la versión de CNTK.

```
(cntk_p36)$ ipython
```

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

El resultado debería ser similar a `2.6-rc0.dev20181015`

Más tutoriales

Si desea ver más tutoriales y ejemplos, consulte la documentación oficial de Python del marco de trabajo, la API de Python para CNTK o el sitio web de CNTK.

Keras

Tutorial de Keras

1. Para activar el marco de trabajo, utilice estos comandos en su interfaz de línea de comandos de the section called “DLAMI con Conda”.
   - Para Keras 2 con un backend MXNet en Python 3 con CUDA 9 con cuDNN 7:

     ```
     $ source activate mxnet_p36
     ```

   - Para Keras 2 con un backend MXNet en Python 2 con CUDA 9 con cuDNN 7:

     ```
     $ source activate mxnet_p27
     ```

   - Para Keras 2 con un TensorFlow backend en Python 3 con CUDA 9 con cuDNN 7:

     ```
     $ source activate tensorflow_p36
     ```

   - Para Keras 2 con un TensorFlow backend en Python 2 con CUDA 9 con cuDNN 7:

     ```
     $ source activate tensorflow_p27
     ```

2. Para probar la importación de Keras para verificar el backend que está activado, utilice estos comandos:

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

   En la pantalla debería aparecer lo siguiente:

   ```
   Using MXNet backend
   ```

   Si Keras está utilizando, se muestra lo siguiente TensorFlow:

   ```
   Using TensorFlow backend
   ```

> **Note**
>
> Si recibe un error o si se sigue utilizando un backend incorrecto, puede actualizar su configuración de Keras manualmente. Edite el archivo `~/.keras/keras.json` y cambie la configuración del backend a `mxnet` o `tensorflow`.

Más tutoriales

- Para ver un tutorial en modo de varias GPU con Keras con un backend MXNet, consulte Tutorial de entrenamiento en varias GPU de Keras-MXNet.
- Puede encontrar ejemplos para Keras con un backend MXNet en el directorio `~/examples/keras-mxnet` de la AMI de aprendizaje profundo con Conda.
- Puede encontrar ejemplos de Keras con un TensorFlow backend en el directorio AMI de aprendizaje profundo con `~/examples/keras` Conda.
- Para obtener más tutoriales y ejemplos, consulte el sitio web de Keras.

PyTorch

¿Activando PyTorch

Cuando se lanza un paquete Conda estable de un marco de trabajo, se prueba y se preinstala en la DLAMI. Si desea ejecutar la última compilación nocturna sin probar, puede PyTorchInstall's Nightly Build (experimental) manualmente.

Para activar el marco de trabajo instalado actualmente, siga estas instrucciones en la AMI de aprendizaje profundo con Conda.

Para PyTorch Python 3 con CUDA 10 y MKL-DNN, ejecute este comando:

```
$ source activate pytorch_p36
```

Para PyTorch Python 2 con CUDA 10 y MKL-DNN, ejecute este comando:

```
$ source activate pytorch_p27
```

Inicie el terminal de iPython.

```
(pytorch_p36)$ ipython
```

Ejecute un programa rápido. PyTorch

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

Debería ver la matriz aleatoria inicial, a continuación su tamaño y, por último, la adición de otra matriz aleatoria.

PyTorchInstall's Nightly Build (experimental)

¿Cómo realizar una instalación a PyTorch partir de una compilación nocturna

Puede instalar la última versión PyTorch en uno o ambos entornos de PyTorch Conda de su AMI de aprendizaje profundo con Conda.

1. • (Opción para Python 3): active el PyTorch entorno Python 3:

   ```
   $ source activate pytorch_p36
   ```

   • (Opción para Python 2): active el PyTorch entorno Python 2:

   ```
   $ source activate pytorch_p27
   ```

2. En los pasos restantes se da por hecho que está utilizando el entorno `pytorch_p36`. Elimine lo que está instalado actualmente PyTorch:

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

3. • (Opción para instancias de GPU): instale la última versión nocturna PyTorch con CUDA 10.0:

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

- (Opción para instancias de CPU): instala la última versión nocturna PyTorch para las instancias sin GPU:

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

4. Para comprobar que has instalado correctamente la última versión nocturna, inicia la terminal IPython y comprueba la versión de. PyTorch

```
(pytorch_p36)$ ipython
```

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

El resultado debería ser similar a `1.0.0.dev20180922`

5. Para comprobar que la compilación PyTorch nocturna funciona bien con el ejemplo del MNIST, puedes ejecutar un script de prueba desde PyTorch el repositorio de ejemplos:

```
(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
```

Más tutoriales

Puede encontrar más tutoriales en la carpeta de tutoriales de la AMI de aprendizaje profundo con Conda del directorio principal de la DLAMI. Para obtener más tutoriales y ejemplos, consulta los documentos oficiales, la PyTorch documentación y el sitio web del PyTorchframework.

- PyTorch Tutorial de ONNX a MXNet
- PyTorch Tutorial de ONNX a CNTK

TensorFlow

¿Activando TensorFlow

En este tutorial, se muestra cómo TensorFlow activar una instancia que ejecute la AMI de aprendizaje profundo con Conda (DLAMI en Conda) y ejecutar un programa. TensorFlow

Cuando se lanza un paquete Conda estable de un marco de trabajo, se prueba y se preinstala en la DLAMI. Si desea ejecutar la última compilación nocturna sin probar, puede TensorFlowInstall's Nightly Build (experimental) manualmente.

Para ejecutar TensorFlow el DLAMI con Conda

1. Para activarlo TensorFlow, abra una instancia de Amazon Elastic Compute Cloud (Amazon EC2) de la DLAMI con Conda.
   - Para TensorFlow Keras 2 en Python 3 con CUDA 9.0 y MKL-DNN, ejecute este comando:

     ```
     $ source activate tensorflow_p36
     ```
   - Para TensorFlow Keras 2 en Python 2 con CUDA 9.0 y MKL-DNN, ejecute este comando:

     ```
     $ source activate tensorflow_p27
     ```
2. Inicie el terminal de iPython:

   ```
   (tensorflow_p36)$ ipython
   ```
3. Ejecute un TensorFlow programa para comprobar que funciona correctamente:

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

   `Hello, TensorFlow!` debe aparecer en la pantalla.

TensorFlowInstall's Nightly Build (experimental)

Puede instalar la última versión TensorFlow en uno o ambos entornos de TensorFlow Conda de su AMI de aprendizaje profundo con Conda.

Para instalar a TensorFlow partir de una compilación nocturna

1. • Para el TensorFlow entorno Python 3, ejecute el siguiente comando:

```
$ source activate tensorflow_p36
```

- Para el TensorFlow entorno Python 2, ejecute el siguiente comando:

```
$ source activate tensorflow_p27
```

2. Elimine el que está instalado actualmente TensorFlow.

> **Note**
>
> En los pasos restantes se da por hecho que está utilizando el entorno `tensorflow_p36`.

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

3. Instale la última versión nocturna de TensorFlow.

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

4. Para comprobar que has instalado correctamente la última versión nocturna, inicia la terminal IPython y comprueba la versión de. TensorFlow

```
(tensorflow_p36)$ ipython
```

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

El resultado debería ser similar a `1.12.0-dev20181012`

Más tutoriales

TensorFlow con Horovod

TensorBoard

TensorFlow Sirviendo

Para ver tutoriales, consulte la carpeta denominada `Deep Learning AMI with Conda tutorials` en el directorio de inicio de la DLAMI.

Para obtener más tutoriales y ejemplos, consulta la TensorFlow documentación de la API de TensorFlow Python o visita el TensorFlowsitio web.

TensorFlow 2

En este tutorial, se muestra cómo activar TensorFlow 2 en una instancia que ejecuta la AMI de aprendizaje profundo con Conda (DLAMI en Conda) y cómo ejecutar un programa 2. TensorFlow

Cuando se lanza un paquete Conda estable de un marco de trabajo, se prueba y se preinstala en la DLAMI. Si desea ejecutar la última compilación nocturna sin probar, puede Instale TensorFlow 2's Nightly Build (experimental) manualmente.

Activando 2 TensorFlow

Para ejecutar TensorFlow el DLAMI con Conda

1. Para activar TensorFlow 2, abra una instancia de Amazon Elastic Compute Cloud (Amazon EC2) de la DLAMI con Conda.
2. Para TensorFlow 2 y Keras 2 en Python 3 con CUDA 10.1 y MKL-DNN, ejecute este comando:

   ```
   $ source activate tensorflow2_p36
   ```

3. Inicie el terminal de iPython:

   ```
   (tensorflow2_p36)$ ipython
   ```

4. Ejecute un programa TensorFlow 2 para comprobar que funciona correctamente:

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

   `Hello, TensorFlow!` debe aparecer en la pantalla.

Instale TensorFlow 2's Nightly Build (experimental)

Puede instalar las TensorFlow 2 versiones más recientes en uno o ambos entornos de TensorFlow Conda de su AMI de aprendizaje profundo con Conda.

Para instalar TensorFlow desde una compilación nocturna

1. Para el entorno Python 3 TensorFlow 2, ejecute el siguiente comando:

```
$ source activate tensorflow2_p36
```

2. Elimine el que está instalado actualmente TensorFlow.

> **Note**
>
> En los pasos restantes se da por hecho que está utilizando el entorno `tensorflow2_p36`.

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

3. Instale la última versión nocturna de TensorFlow.

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

4. Para comprobar que has instalado correctamente la última versión nocturna, inicia la terminal IPython y comprueba la versión de. TensorFlow

```
(tensorflow2_p36)$ ipython
```

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

El resultado debería ser similar a `2.1.0-dev20191122`

Más tutoriales

Para ver tutoriales, consulte la carpeta denominada `Deep Learning AMI with Conda tutorials` en el directorio de inicio de la DLAMI.

Para obtener más tutoriales y ejemplos, consulta la TensorFlow documentación de la API de TensorFlow Python o visita el TensorFlowsitio web.

TensorFlow con Horovod

Este tutorial muestra cómo activar TensorFlow con Horovod en un (AWS Deep Learning AMI DLAMI) con Conda. Horovod viene preinstalado en los entornos de Conda para. TensorFlow Se recomienda el entorno Python3.

> **Note**
>
> Solo se admiten los tipos de instancias P3.*, P2.* y G3.*

Para activar TensorFlow y probar Horovod en la DLAMI con Conda

1. Abra una instancia de Amazon Elastic Compute Cloud (Amazon EC2) de la DLAMI con Conda. Si desea obtener ayuda para empezar con DLAMI consulte the section called “Primeros pasos con la DLAMI”.
2. (Recomendado) Para la TensorFlow versión 1.15 con Horovod en Python 3 con CUDA 11, ejecute el siguiente comando:

   ```
   $ source activate tensorflow_p37
   ```

3. Inicie el terminal de iPython:

   ```
   (tensorflow_p37)$ ipython
   ```

4. Prueba la importación TensorFlow con Horovod para comprobar que funciona correctamente:

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

   En la pantalla podría aparecer lo siguiente (puede pasar por alto los mensajes de advertencia).

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

Más información

- TensorFlow con Horovod
- Para ver tutoriales, consulte la carpeta `examples/horovod` en el directorio de inicio de DLAMI.
- Para ver más tutoriales y ejemplos, consulta el proyecto GitHub Horovod.

TensorFlow 2 con Horovod

Este tutorial muestra cómo activar TensorFlow 2 con Horovod en un (AWS Deep Learning AMI DLAMI) con Conda. Horovod viene preinstalado en los entornos de Conda para 2. TensorFlow Se recomienda el entorno Python3.

> **Note**
>
> Solo se admiten los tipos de instancias P3.*, P2.* y G3.*

Para activar TensorFlow 2 y probar Horovod en la DLAMI con Conda

1. Abra una instancia de Amazon Elastic Compute Cloud (Amazon EC2) de la DLAMI con Conda. Si desea obtener ayuda para empezar con DLAMI consulte the section called “Primeros pasos con la DLAMI”.
   - (Recomendado) Para TensorFlow 2 con Horovod en Python 3 con CUDA 10, ejecute este comando:

     ```
     $ source activate tensorflow2_p36
     ```

- Para TensorFlow 2 con Horovod en Python 2 con CUDA 10, ejecute este comando:

```
$ source activate tensorflow2_p27
```

2. Inicie el terminal de iPython:

```
(tensorflow2_p36)$ ipython
```

3. Prueba a importar TensorFlow 2 con Horovod para comprobar que funciona correctamente:

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

Si no recibe ninguna salida, Horovod está funcionando correctamente. En la pantalla podría aparecer lo siguiente (puede pasar por alto los mensajes de advertencia).

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

Más información

- Para ver tutoriales, consulte la carpeta `examples/horovod` en el directorio de inicio de DLAMI.
- Para ver más tutoriales y ejemplos, consulta el proyecto GitHub Horovod.

Theano

Tutorial de Theano

> **Note**
>
> Ya no incluimos los entornos CNTK, Caffe, Caffe 2 y Theano Conda en la AWS Deep Learning AMI a partir de la versión v28. Las versiones anteriores AWS Deep Learning AMI que contienen estos entornos seguirán estando disponibles. Sin embargo, solo proporcionaremos actualizaciones a estos entornos si la comunidad de código abierto publica correcciones de seguridad para estos marcos.

Para activar el marco de trabajo, siga estas instrucciones en la AMI de aprendizaje profundo con Conda.

Para Theano + Keras en Python 3 con CUDA 9 con cuDNN 7:

```
$ source activate theano_p36
```

Para Theano + Keras en Python 2 con CUDA 9 con cuDNN 7:

```
$ source activate theano_p27
```

Inicie el terminal de iPython.

```
(theano_p36)$ ipython
```

Ejecute un programa rápido en Theano.

```
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)
```

Debería ver cómo Theano calcula un gradiente simbólico.

Más tutoriales

Si desea ver más tutoriales y ejemplos, consulte la documentación oficial del marco de trabajo, la API de Python para Theano y el sitio web de Theano.

Depuración y visualización

Obtenga más información acerca de las opciones de visualización y depuración para la DLAMI. Haga clic en una de las opciones para obtener información acerca de cómo utilizarla.

Temas

- MXBoard
- TensorBoard

MXBoard

MXBoard le permite inspeccionar e interpretar visualmente sus corridas y gráficos de MXNet mediante TensorBoard el software. Ejecuta un servidor web que sirve una página web para visualizar e interaccionar con las visualizaciones de MXBoard.

MXNet y MXBoard vienen preinstalados con la AMI de aprendizaje profundo con Conda (DLAMI con Conda). TensorBoard En este tutorial, utilizará una función de MXBoard para generar registros compatibles con. TensorBoard

Temas

- Uso de MXNet con MXBoard
- Más información

Uso de MXNet con MXBoard

Genere datos de registro de MXBoard compatibles con TensorBoard

1. Conéctese a su instancia de Amazon Elastic Compute Cloud (Amazon EC2) de la DLAMI con Conda.

2. Active el entorno MXNet de Python 3.

```
$ source activate mxnet_p36
```

3. Prepare un script de Python para escribir los datos generados por el operador normal a un archivo de eventos. Los datos se genera diez veces con desviación estándar decreciente y, a continuación, se escriben en el archivo de eventos cada vez. Verá cómo la distribución de datos gradualmente se centra más en torno a un valor medio. Tenga en cuenta que especificará el archivo de eventos en la carpeta de registros. Pasas esta ruta de carpeta al TensorBoard binario.

```
$ vi mxboard_normal.py
```

4. Pegue lo siguiente en el archivo y guárdelo:

```
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. Ejecute el script. Esto generará registros en una carpeta `logs` que puede utilizar para visualizaciones.

```
$ python mxboard_normal.py
```

6. Ahora debe cambiar al TensorFlow entorno para utilizar TensorBoard MXBoard para visualizar los registros. Esta es una dependencia obligatoria para MXBoard y. TensorBoard

```
$ source activate tensorflow_p36
```

7. Transfiera la ubicación de los registros a `tensorboard`:

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

TensorBoard inicia el servidor web de visualización en el puerto 8888.

8. Para facilitar el acceso desde su navegador local, puede cambiar el puerto del servidor web al puerto 80 o a otro puerto. Con independencia del puerto que utilice, tendrá que abrirlo en el grupo de seguridad de EC2 para su DLAMI. También puede usar el redireccionamiento de puertos. Para obtener instrucciones acerca del cambio de la configuración de su grupo de seguridad y redireccionamiento de puertos, consulte Configuración de un servidor de cuadernos de Jupyter. La configuración predeterminada se describe en el siguiente paso.

   > **Note**
   >
   > Si tiene que ejecutar el servidor Jupyter y un servidor MXBoard, utilice un puerto diferente para cada uno de ellos.

9. Abra el puerto 8888 (o el puerto asignado al servidor web de visualización) en la instancia EC2.

   a. Abra su instancia de EC2 en la consola de Amazon EC2 en https://console.aws.amazon.com/ec2/.

   b. En la consola Amazon EC2, elija Red y seguridad y, a continuación, elija Grupos de seguridad.

   c. Para Security Group (Grupo de seguridad), elija el que se creó más recientemente (consulte la marca de tiempo en la descripción).

   d. Elija la pestaña Inbound (Entrada) y elija Edit (Editar).

   e. Seleccione Add Rule (Agregar regla).

   f. En la nueva fila, escriba lo siguiente:

      Tipo: **`TCP Rule`** personalizada

      Protocolo: **`TCP`**

      Intervalo de puertos: **`8888`** (o el puerto que haya asignado al servidor de visualización)

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

10. Si desea visualizar los datos del navegador local, escriba el siguiente comando para reenviar los datos que está representando en la instancia EC2 al equipo local.

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

11. Abra la página web para las visualizaciones de MXBoard mediante la dirección DNS o IP pública de la instancia EC2 que ejecuta la DLAMI con Conda y el puerto que abrió para MXBoard:

    `http://127.0.0.1:8888`

Más información

Para obtener más información sobre MXBoard, consulte el sitio web de MXBoard.

TensorBoard

TensorBoardle permite inspeccionar e interpretar visualmente sus TensorFlow corridas y gráficos. Ejecuta un servidor web que sirve de página web para ver TensorBoard las visualizaciones e interactuar con ellas.

TensorFlow y TensorBoard vienen preinstalados con la AMI de aprendizaje profundo con Conda (DLAMI con Conda). El DLAMI con Conda también incluye un script de ejemplo que se TensorFlow utiliza para entrenar un modelo MNIST con funciones de registro adicionales habilitadas. MNIST es una base de datos de números escritos a mano que se utiliza generalmente para entrenar modelos de reconocimiento de imágenes. En este tutorial, utilizará el script para entrenar un modelo MNIST TensorBoard y los registros para crear visualizaciones.

Temas

- Entrene un modelo MNIST y visualice el entrenamiento con TensorBoard
- Más información

Entrene un modelo MNIST y visualice el entrenamiento con TensorBoard

Visualice el entrenamiento del modelo MNIST con TensorBoard

1. Conéctese a su instancia de Amazon Elastic Compute Cloud (Amazon EC2) de la DLAMI con Conda.
2. Active el TensorFlow entorno Python 2.7 y navegue hasta el directorio que contiene la carpeta con los scripts de TensorBoard ejemplo:

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

3. Ejecute el script que entrena un modelo de MNIST con el registro extendido habilitado:

```
$ python mnist_with_summaries.py
```

   El script escribe los registros en `/tmp/tensorflow/mnist`.

4. Transfiera la ubicación de los registros a `tensorboard`:

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

   TensorBoard inicia el servidor web de visualización en el puerto 6006.

5. Para facilitar el acceso desde su navegador local, puede cambiar el puerto del servidor web al puerto 80 o a otro puerto. Con independencia del puerto que utilice, tendrá que abrirlo en el grupo de seguridad de EC2 para su DLAMI. También puede usar el redireccionamiento de puertos. Para obtener instrucciones acerca del cambio de la configuración de su grupo de seguridad y redireccionamiento de puertos, consulte Configuración de un servidor de cuadernos de Jupyter. La configuración predeterminada se describe en el siguiente paso.

   > **Note**
   >
   > Si necesita ejecutar tanto el servidor Jupyter como un TensorBoard servidor, utilice un puerto diferente para cada uno.

6. Abra el puerto 6006 (o el puerto asignado al servidor web de visualización) en la instancia EC2.

   a. Abra su instancia de EC2 en la consola de Amazon EC2 en https://console.aws.amazon.com/ec2/.
   b. En la consola de Amazon EC2;, elija Red y seguridad y, a continuación, elija Grupos de seguridad.
   c. Para Security Group (Grupo de seguridad), elija el que se creó más recientemente (consulte la marca de tiempo en la descripción).
   d. Elija la pestaña Inbound (Entrada) y elija Edit (Editar).
   e. Seleccione Add Rule (Agregar regla).
   f. En la nueva fila, escriba lo siguiente:

      Tipo: **`TCP Rule`** personalizada

      Protocolo: **`TCP`**

Intervalo de puertos: **6006** (o el puerto que haya asignado al servidor de visualización)

Origen: **Custom IP (specify address/range)**

7. Abra la página web de TensorBoard las visualizaciones mediante la dirección IP o DNS pública de la instancia EC2 que ejecuta la DLAMI con Conda y el puerto que ha abierto para: TensorBoard

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

Más información

Para obtener más información, consulte el sitio web TensorBoard. TensorBoard

Entrenamiento distribuido

Obtenga más información acerca de las opciones que tiene la DLAMI para el entrenamiento con varias GPU. Para lograr el mejor rendimiento, consulte Elastic Fabric Adapter Haga clic en una de las opciones para obtener información acerca de cómo utilizarlo.

Temas

- Chainer
- Keras con MXNet
- TensorFlow con Horovod

Chainer

> **Note**
>
> Desde la versión v28, ya no incluimos los entornos Chainer Conda en AWS Deep Learning AMI . Las versiones anteriores AWS Deep Learning AMI que contienen estos entornos seguirán estando disponibles. Sin embargo, solo proporcionaremos actualizaciones a estos entornos si la comunidad de código abierto publica correcciones de seguridad para estos marcos.

Chainer es un marco de trabajo flexible basado en Python para escribir de forma sencilla e intuitiva arquitecturas de red neuronal complejas. Chainer facilita la utilización de instancias de varias

GPU para entrenamiento. Chainer además registra automáticamente los resultados, representa gráficamente pérdidas y precisión y produce una salida para visualizar la red neuronal con un grafo computacional. Se incluye en la AMI de aprendizaje profundo con Conda (DLAMI con Conda).

Los temas siguientes le muestran cómo entrenar varias GPU, una única GPU y una CPU, crear visualizaciones y probar su instalación de Chainer.

Temas

- Entrenamiento de un modelo con Chainer
- Utilizar Chainer para entrenar en varias GPU
- Utilizar Chainer para entrenar en una única GPU
- Utilice Chainer para entrenar con CPU
- Representación gráfica de resultados
- Prueba de Chainer
- Más información

Entrenamiento de un modelo con Chainer

En este tutorial se muestra cómo utilizar scripts de Chainer de ejemplo para entrenar un modelo con el conjunto de datos MNIST. MNIST es una base de datos de números escritos a mano que se utiliza generalmente para entrenar modelos de reconocimiento de imágenes. El tutorial también muestra la diferencia en la velocidad de entrenamiento entre el entrenamiento en una CPU y en una o varias GPU.

Utilizar Chainer para entrenar en varias GPU

Para entrenar en varias GPU

1. Conéctese a la instancia que ejecuta la AMI de aprendizaje profundo con Conda. Consulte la the section called “Selección de una instancia” o la documentación de Amazon EC2 acerca de cómo seleccionar una instancia o conectarse a ella. Para ejecutar este tutorial, deberá utilizar una instancia con al menos dos GPU.
2. Active el entorno de Chainer de Python 3:

```
$ source activate chainer_p36
```

3. Para obtener los tutoriales más recientes, clone el repositorio de Chainer y acceda a la carpeta de ejemplos:

```
(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. Ejecute el ejemplo en el script `train_mnist_data_parallel.py`. De forma predeterminada, el script utiliza las GPU que se ejecutan en su instancia de la AMI de aprendizaje profundo con Conda. El script se puede ejecutar en un máximo de dos GPU. Se hará caso omiso de cualquier GPU después de las dos primeras. Detecta una o ambas de forma automática. Si ejecuta una instancia sin GPU, vaya a Utilice Chainer para entrenar con CPU, más adelante en este tutorial.

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

> **Note**
>
> Este ejemplo devolverá el siguiente error debido a la inclusión de una característica beta no incluida en la DLAMI.
> `chainerx ModuleNotFoundError: No module named 'chainerx'`

Aunque el script de Chainer entrena un modelo utilizando la base de datos MNIST, verá los resultados para cada fecha de inicio.

A continuación, verá un resultado de ejemplo a medida que se ejecuta el script. El siguiente resultado de ejemplo se ejecutó en una instancia p3.8xlarge. El resultado del script muestra "GPU: 0, 1", que indica que está utilizando las dos primeras de las cuatro GPU disponibles. Los scripts suelen utilizar un índice de GPU que empieza en cero, en lugar de un recuento total.

```
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. Aunque su entrenamiento se esté ejecutando, es útil echar un vistazo a su utilización de GPU. Puede comprobar qué GPU están activas y ver su carga. NVIDIA proporciona una herramienta para este fin, que se puede ejecutar con el comando `nvidia-smi`. Sin embargo, solo le mostrará una instantánea de la utilización, por lo que resulta más informativo combinarla con el comando de Linux `watch`. El siguiente comando utilizará `watch` con `nvidia-smi` para actualizar la utilización de GPU actual cada décima de segundo. Abra otra sesión de terminal en su DLAMI y ejecute el siguiente comando:

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

Verá una salida similar al siguiente resultado. Utilice `ctrl-c` para cerrar la herramienta o manténgala en ejecución mientras prueba otros ejemplos en su primera sesión de terminal.

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

En este ejemplo, GPU 0 y GPU 1 están activas y GPU 2 y 3 no lo están. También puede ver la utilización de memoria por GPU.

6. Cuando se complete el entrenamiento, anote el tiempo transcurrido en su primera sesión de terminal. En el ejemplo, el tiempo transcurrido es 38,1389 segundos.

Utilizar Chainer para entrenar en una única GPU

Este ejemplo muestra cómo entrenar en una única GPU. Podría hacerlo si tiene solo una única GPU disponible o simplemente para ver cómo podría escalarse el entrenamiento en varias GPU con Chainer.

Para utilizar Chainer para entrenar en una única GPU

- En este ejemplo, utiliza otro script, `train_mnist.py`, y le indica que utilice solo GPU 0 con el argumento `--gpu=0`. Para ver como distintas GPU se activan en la consola de `nvidia-smi`, puede indicarle al script que utilice la GPU número 1 utilizando `--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
```

En este ejemplo, la ejecución en una única GPU requirió casi el doble de tiempo. El entrenamiento de modelos más grandes o de conjuntos de datos de mayor tamaño dará un resultado distinto de este ejemplo, por tanto experimente para evaluar mejor el desempeño de GPU.

Utilice Chainer para entrenar con CPU

Ahora, pruebe a entrenar en un modo de solo CPU. Ejecute el mismo script, `python train_mnist.py`, sin argumentos:

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

En el resultado, GPU: -1 indica que no se ha utilizado ninguna 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
```

En este ejemplo, MNIST se entrenó en 323 segundos, lo que supone un entrenamiento 11 veces más prolongado que con dos GPU. Si ha dudado alguna vez de la potencia de las GPU, este ejemplo muestra hasta qué punto son más eficientes.

Representación gráfica de resultados

Chainer además registra automáticamente los resultados, representa gráficamente pérdidas y precisión y genera salida para trazar el grafo computacional.

Para generar el grafo computacional

1. Una vez que una ejecución de entrenamiento finaliza, puede acceder al directorio `result` y ver la precisión y pérdida de la ejecución en forma de dos imágenes generadas automáticamente. Acceda allí ahora y enumere el contenido:

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

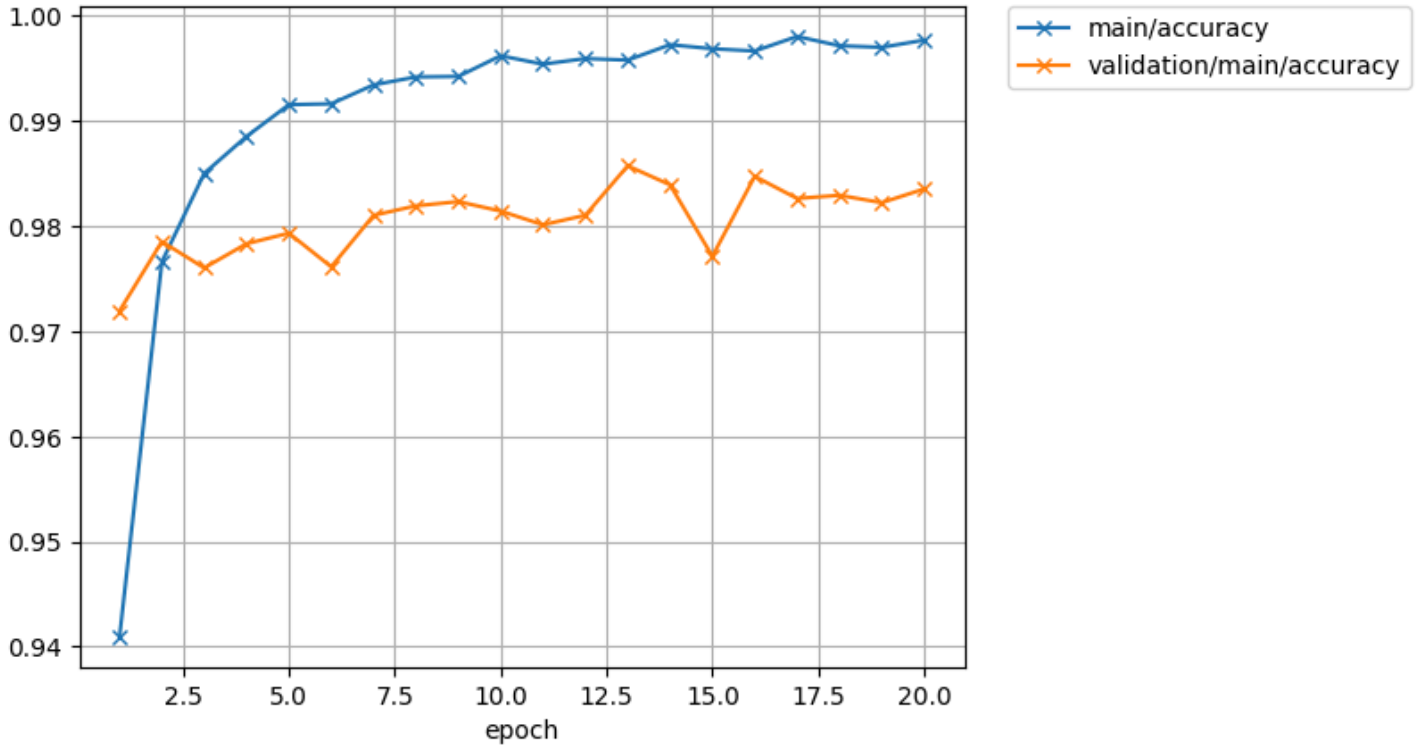
   El directorio `result` contiene dos archivos en formato .png: `accuracy.png` y `loss.png`.

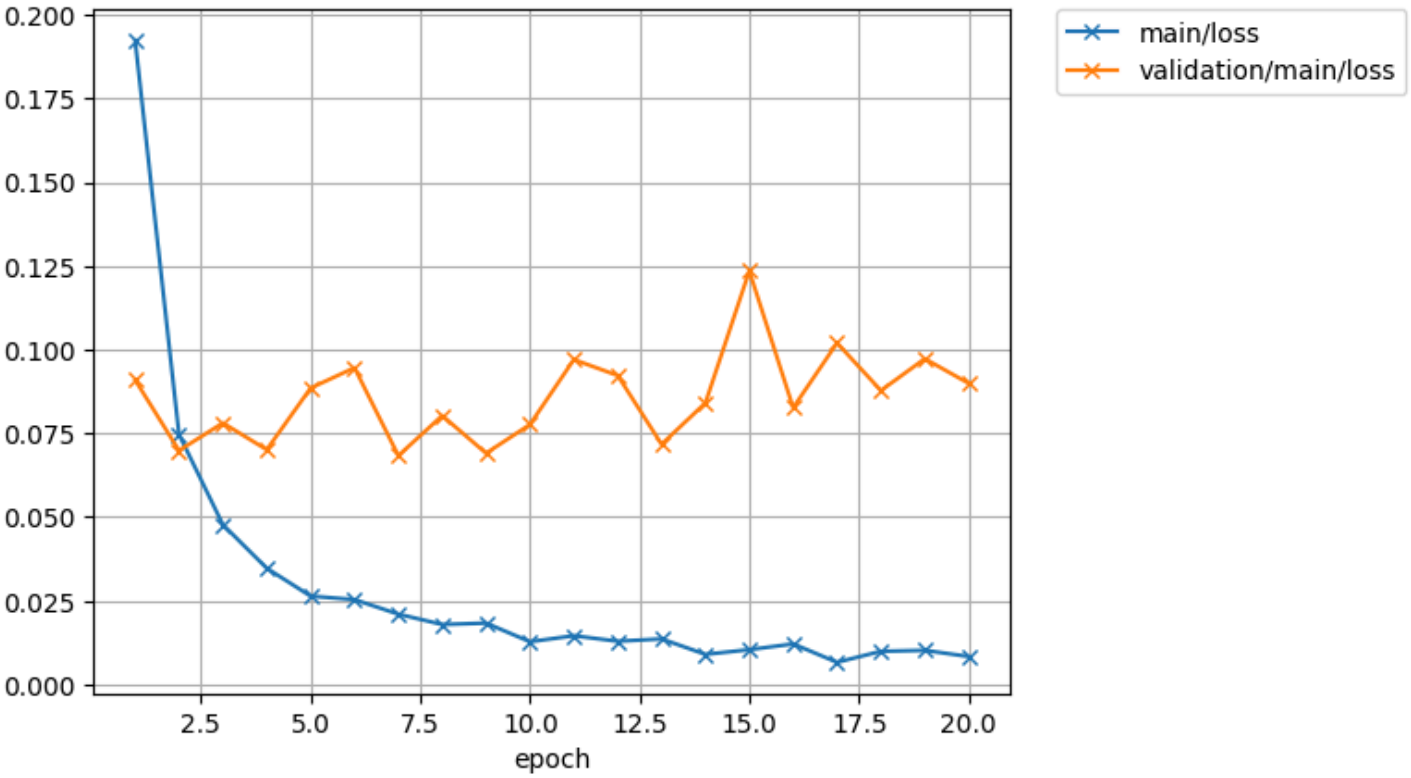
2. Para ver los gráficos, use el comando `scp` para copiarlos en su equipo local.

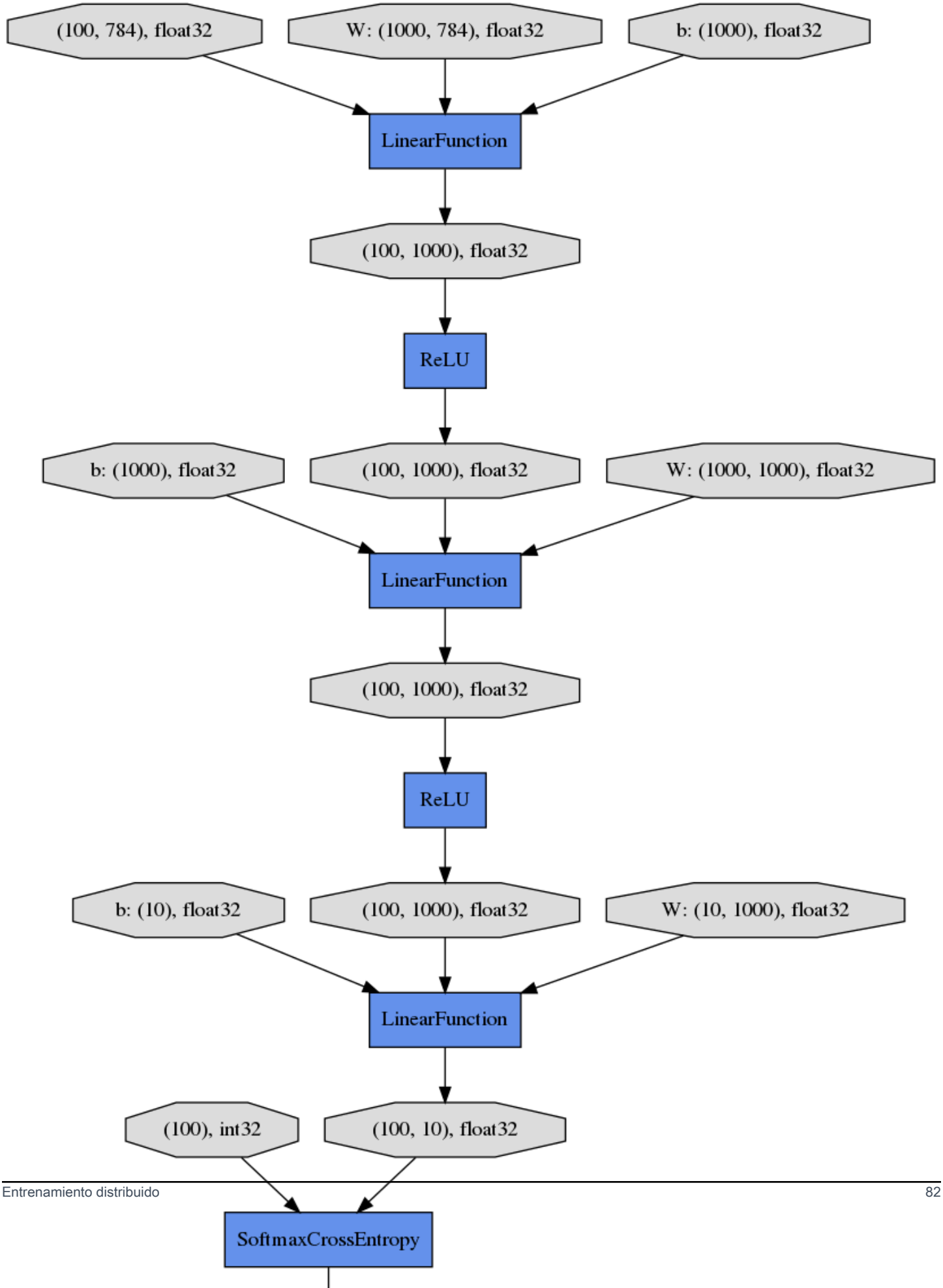
   En un terminal macOS, la ejecución del siguiente comando `scp` descarga los tres archivos en la carpeta `Downloads`. Sustituya los marcadores de posición para la ubicación del archivo de claves y de la dirección del servidor con su información. Para otros sistemas operativos, utilice el formato de comando `scp` adecuado. Tenga en cuenta que, para una AMI de Amazon Linux, el nombre de usuario es 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
```

Las siguientes imágenes son ejemplos de grafos de precisión, pérdidas y computacionales, respectivamente.




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


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

Prueba de Chainer

Para probar Chainer y verificar la compatibilidad de GPU con un script de prueba preinstalado, ejecute el siguiente comando:

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

Esto descarga el código fuente de Chainer y ejecuta el ejemplo MNIST con varias GPU de Chainer.

Más información

Para obtener más información sobre Chainer, consulte el sitio web de documentación de Chainer. La carpeta de ejemplos de `Chainer` contiene más ejemplos. Pruébelos para comprobar su desempeño.

Keras con MXNet

En este tutorial se muestra cómo activar y utilizar Keras 2 con el backend MXNet en una AMI de aprendizaje profundo con Conda.

Activar Keras con el backend MXNet y probarlo en la DLAMI con Conda

1. Para activar Keras con el backend de MXNet, abra una instancia de Amazon Elastic Compute Cloud (Amazon EC2) de la DLAMI con Conda.
   - Para Python 3, ejecute este comando:

     ```
     $ source activate mxnet_p36
     ```
   - Para Python 2, ejecute este comando:

     ```
     $ source activate mxnet_p27
     ```
2. Inicie el terminal de iPython:

   ```
   (mxnet_p36)$ ipython
   ```
3. Pruebe a importar Keras con MXNet para verificar que funciona correctamente:

   ```
   import keras as k
   ```

En la pantalla debería aparecer lo siguiente (posiblemente tras algunos mensajes de advertencia).

```
Using MXNet backend
```

> **Note**
>
> Si recibes un error o si el TensorFlow backend sigue utilizándose, tendrás que actualizar la configuración de Keras manualmente. Edite el archivo `~/.keras/keras.json` y cambie la configuración del backend a `mxnet`.

Tutorial de entrenamiento en varias GPU de Keras-MXNet

Entrenar una red neuronal convolucional (CNN)

1. Abra un terminal y SSH en su DLAMI.
2. Vaya a la carpeta `~/examples/keras-mxnet/`.
3. Ejecute `nvidia-smi` en su ventana de terminal para determinar el número de GPU disponibles en su DLAMI. En el paso siguiente, ejecutará el script tal y como está si tiene cuatro GPU.
4. De manera opcional, ejecute el siguiente comando para abrir el script para editarlo.

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

5. De manera opcional, el script tiene la siguiente línea que define el número de GPU. Actualícelo si es necesario.

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

6. Ahora, ejecute el entrenamiento.

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

> **Note**
>
> Keras-MXNet se ejecuta hasta dos veces más rápido con el conjunto `channels_first` image_data_format. Para cambiar a `channels_first`, edite el archivo de configuración de Keras (`~/.keras/keras.json`) y defina lo siguiente: `"image_data_format": "channels_first"`.
> Para obtener más técnicas de ajuste del rendimiento, consulte la Guía para ajustar el rendimiento de Keras-MXNet.

Más información

- Puede encontrar ejemplos para Keras con un backend MXNet en el directorio `~/examples/keras-mxnet` de la AMI de aprendizaje profundo con Conda.
- Para ver más tutoriales y ejemplos, consulte el proyecto Keras-MXNet GitHub .

TensorFlow con Horovod

Este tutorial muestra cómo usarlo TensorFlow con Horovod en una AMI de aprendizaje profundo con Conda. Horovod viene preinstalado en los entornos de Conda para. TensorFlow Se recomienda el entorno Python 3. Las instrucciones siguientes presuponen que tiene una instancia de DLAMI en funcionamiento con una o varias GPU. Para obtener más información, consulte Primeros pasos con la DLAMI.

> **Note**
>
> Solo se admiten los tipos de instancias P3.*, P2.* y G3.*

> **Note**
>
> Hay dos ubicaciones donde está disponible mpirun (a través de OpenMPI). Está disponible en `/usr/bin` y `/home/ubuntu/anaconda3/envs/<env>/bin`. `env` es un entorno correspondiente al marco, como Tensorflow y Apache MXNet. Las versiones más recientes de OpenMPI están disponibles en los entornos conda. Recomendamos utilizar la ruta absoluta del binario mpirun o el indicador --prefix para ejecutar cargas de trabajo de mpi. Por ejemplo, con el entorno python36 de Tensorflow, utilice cualquiera de estas dos opciones:

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

or

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

Activa y prueba con Horovod TensorFlow

1. Verifique que la instancia tiene GPU activas. NVIDIA proporciona una herramienta para hacerlo:

   ```
   $ nvidia-smi
   ```

2. Active el TensorFlow entorno Python 3:

   ```
   $ source activate tensorflow_p36
   ```

3. Inicie el terminal de iPython:

   ```
   (tensorflow_p36)$ ipython
   ```

4. Pruebe la importación TensorFlow con Horovod para comprobar que funciona correctamente:

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

   En la pantalla podría aparecer lo siguiente (posiblemente tras algunos mensajes de advertencia).

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

Configurar el archivo de hosts de Horovod

Puede utilizar Horovod para realizar un entrenamiento de un solo nodo con varias GPU o un entrenamiento de varios nodos con varias GPU. Si tiene previsto utilizar varios nodos para el entrenamiento distribuido, debe añadir la dirección IP privada de cada DLAMI a un archivo de hosts. La DLAMI en la que se ha iniciado sesión se conoce como el líder. Las demás instancias de la DLAMI que forman parte del clúster se denominan miembros.

Antes de comenzar con esta sección, lance una o varias y espere a que estén en el estado Lista. Los scripts de ejemplo esperan un archivo de hosts, por lo que, incluso si pretende utilizar una única DLAMI, debe crear un archivo de hosts con una sola entrada. Si edita el archivo de hosts después de que comience el entrenamiento, debe reiniciar el entrenamiento para que los hosts que ha añadido o eliminado surtan efecto.

Para configurar Horovod para el entrenamiento

1. Cambie al directorio en el que residen los scripts de entrenamiento.

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

2. Utilice vim para editar un archivo en el directorio principal del líder.

   ```
   vim hosts
   ```

3. Seleccione uno de los miembros en la consola de Amazon Elastic Compute Cloud y aparecerá el panel de descripción de la consola. Busque el campo Private IPs (Direcciones IP privadas) y copie la dirección IP y péguela en un archivo de texto. Copie la dirección IP privada de cada miembro en una línea nueva. A continuación, junto a cada dirección IP, añada un espacio y, después, el texto `slots=8`, tal y como se muestra a continuación. Esto representa el número de GPU de cada instancia. Las instancias p3.16xlarge tienen 8 GPU, por lo que si elige otro tipo de instancia, debería introducir el número real de GPU de cada instancia. Para el líder puede utilizar `localhost`. Con un clúster de 4 nodos, debe tener un aspecto similar al siguiente:

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

   Guarde el archivo y vuelva al terminal del líder.

4. Agregue la clave SSH utilizada por las instancias miembros al ssh-agent.

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

5. Ahora, el líder sabe cómo llegar a cada miembro. Todo esto va a suceder en las interfaces de red privadas. A continuación, utilice una corta función de bash para ayudar a enviar comandos a cada miembro.

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

6. Dígales a los demás miembros que no hagan «StrickHostKeyChecking», ya que esto puede provocar que el entrenamiento deje de responder.

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

Entrenamiento con datos sintéticos

La DLAMI se suministra con un script de ejemplo para entrenar un modelo con datos sintético. Sirve para probar si el líder puede comunicarse con los miembros del clúster. Se necesita un archivo de hosts. Para obtener instrucciones, consulte Configurar el archivo de hosts de Horovod.

Para probar el entrenamiento de Horovod mediante con datos de ejemplo.

1. De forma predeterminada, `~/examples/horovod/tensorflow/train_synthetic.sh` funciona con 8 GPU, pero puede indicar el número de GPU que desee ejecutar. En el ejemplo siguiente, se ejecuta el script, y se pasa 4 como parámetro para 4 GPU.

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

   Después de algunos mensajes de advertencia, verá el siguiente resultado que confirma que Horovod utiliza 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
```

A continuación, después de otras advertencias, verá el inicio de una tabla y algunos puntos de datos. Si no desea ver 1000 lotes, interrumpa el entrenamiento.

```
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 utiliza todas las GPU locales antes de intentar utilizar las GPU de los miembros del clúster. Por lo tanto, para asegurarse de que funciona el entrenamiento distribuido en el clúster, pruebe con todas las GPU que tenga previsto utilizar. Si, por ejemplo, tiene 4 miembros con el tipo de instancia p3.16xlarge, tiene 32 GPU en todo el clúster. En este caso, podría probar a usar todas las 32 GPU juntas.

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

   El resultado es similar al de la prueba anterior. El tamaño de Horovod es de 32 y aproximadamente cuatro veces la velocidad. Una vez finalizadas estas pruebas, ya ha probado el nodo principal y su capacidad para comunicarse con los miembros. Si tiene algún problema, consulte la sección Solución de problemas.

Prepare el ImageNet conjunto de datos

En esta sección, descarga el ImageNet conjunto de datos y, a continuación, genera un conjunto de datos con formato TFRecord a partir del conjunto de datos sin procesar. En la DLAMI se proporciona un conjunto de scripts de preprocesamiento para el conjunto de datos que puede usar para ImageNet otro conjunto de datos ImageNet o como plantilla para otro conjunto de datos. También se proporcionan los principales guiones de entrenamiento para ImageNet los que están configurados. En la sección siguiente, se supone que ha lanzado una DLAMI con una instancia EC2 con 8 GPU. Recomendamos el tipo de instancia p3.16xlarge.

En el directorio `~/examples/horovod/tensorflow/utils` de la DLAMI, se encuentran los siguientes scripts:

- `utils/preprocess_imagenet.py`- Úselo para convertir el ImageNet conjunto de datos sin procesar al `TFRecord` formato.
- `utils/tensorflow_image_resizer.py`- Úselo para cambiar el tamaño del `TFRecord` conjunto de datos según lo recomendado para el ImageNet entrenamiento.

Prepare el conjunto de datos ImageNet

1. Visite image-net.org, cree una cuenta, adquiera una clave de acceso y descargue el conjunto de datos. image-net.org aloja el conjunto de datos sin procesar. Para descargarlo, debe tener una ImageNet cuenta y una clave de acceso. La cuenta es gratuita y, para obtener la clave de acceso gratuita, debe aceptar la ImageNet licencia.
2. Utilice el script de preprocesamiento de imágenes para generar un conjunto de datos en formato TFRecord a partir del conjunto de datos sin procesar. ImageNet Desde el directorio `~/examples/horovod/tensorflow/utils`:

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

3. Utilice el script de cambio de tamaño de imagen. Si cambias el tamaño de las imágenes, el entrenamiento se ejecuta más rápido y se alinea mejor con el ResNet paper de referencia. Desde el directorio `~/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
```

Entrene un ImageNet modelo ResNet -50 en una sola DLAMI

> **Note**
>
> - El script de este tutorial espera que los datos de entrenamiento preprocesados se encuentren en la carpeta `~/data/tf-imagenet/`. Para obtener instrucciones, consulte Prepare el ImageNet conjunto de datos.
> - Se necesita un archivo de hosts. Para obtener instrucciones, consulte Configurar el archivo de hosts de Horovod.

Usa Horovod para entrenar a una ResNet CNN de 50 grados con el conjunto de datos ImageNet

1. Vaya a la carpeta `~/examples/horovod/tensorflow`.

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

2. Compruebe la configuración y defina el número de GPU que va a utilizar en el entrenamiento. En primer lugar, revise el `hosts` que se encuentra en la misma carpeta que los scripts. Este archivo debe actualizarse si utiliza una instancia con menos de 8 GPU. De forma predeterminada indica `localhost slots=8`. Actualice el número 8 al número de GPU que desea utilizar.
3. Se proporciona un script de shell que toma como único parámetro el número de GPU que se deben utilizar. Ejecute este script para iniciar el entrenamiento. El ejemplo siguiente utiliza 4 para cuatro GPU.

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

4. Tarda varias horas en terminar. Utiliza `mpirun` para distribuir el entrenamiento en las GPU.

Entrene un ImageNet modelo ResNet -50 en un grupo de DLAMI

> **Note**
>
> - El script de este tutorial espera que los datos de entrenamiento preprocesados se encuentren en la carpeta `~/data/tf-imagenet/`. Para obtener instrucciones, consulte Prepare el ImageNet conjunto de datos.

- Se necesita un archivo de hosts. Para obtener instrucciones, consulte Configurar el archivo de hosts de Horovod.

En este ejemplo, se explica cómo entrenar un modelo ResNet -50 en un conjunto de datos preparado en varios nodos de un clúster de DLAMI.

- Para mejorar el rendimiento, recomendamos que tenga el conjunto de datos localmente en cada miembro del clúster.

  Utilice esta función `copyclust` para copiar datos a otros miembros.

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

O bien, si tiene los archivos en un bucket de S3, utilice la función `runclust` para descargar directamente los archivos en cada miembro.

```
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/\""
```

El uso de herramientas que le permiten administrar varios nodos a la vez proporciona un ahorro de tiempo considerable. Puede esperar a que finalice cada paso y administrar cada instancia por separado o utilizar herramientas como tmux o screen que le permiten desconectar y reanudar sesiones.

Cuando finalice la copia, estará listo para comenzar el entrenamiento. Ejecute el script, pasando 32 como parámetro para las 32 GPU que estamos utilizando para esta ejecución. Utilice tmux o una herramienta similar si le preocupa que se produzca una desconexión y termine la sesión, lo que finalizaría la sesión de entrenamiento.

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

El siguiente resultado es lo que se ve al ejecutar el entrenamiento ImageNet con 32 GPU. Treinta y dos GPU tardan 90–110 minutos.

```
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
```

Cuando finaliza la sesión de entrenamiento, el script continúa con una sesión de evaluación. Se ejecuta en el líder, ya es lo suficientemente rápida como para no tener que distribuir el trabajo entre los demás miembros. A continuación, se muestra el resultado de la sesión de evaluación.

```
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
```

A continuación, se muestra un ejemplo del resultado obtenido al ejecutar este script con 256 GPU, donde el tiempo de ejecución fue de entre 14 y 15 minutos.

```
  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
```

Solución de problemas

El comando siguiente puede ayudar a superar los errores que surgen cuando se experimenta con Horovod.

- Si el entrenamiento se bloquea por algún motivo, es posible que mpirun no sea capaz de limpiar todos los procesos de python de cada máquina. En ese caso, antes de iniciar el siguiente trabajo, detenga los procesos de python en todas las máquinas como se indica a continuación:

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

- Si el proceso termina bruscamente sin errores, intente eliminar carpeta de registro.

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

- Si se presentan otros problemas inexplicables, compruebe el espacio en disco. Si se ha agotado, pruebe a borrar la carpeta de registros, ya que está llena de datos y puntos de comprobación. También puede aumentar el tamaño de los volúmenes de cada miembro.

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

- En última instancia también puede probar a reiniciar.

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

Es posible que recibas el siguiente código de error si intentas usarlo TensorFlow con Horovod en un tipo de instancia no compatible:

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

Más información

Para obtener las utilidades y los ejemplos, consulte la carpeta `~/examples/horovod` en el directorio principal de la DLAMI.

Para ver más tutoriales y ejemplos, consulta el proyecto Horovod. GitHub

Elastic Fabric Adapter

Un Elastic Fabric Adapter (EFA) es un dispositivo de red que puede adjuntar a su instancia de Amazon EC2 para acelerar las aplicaciones de computación de alto rendimiento (HPC). EFA le permite lograr el rendimiento de las aplicaciones de un clúster de HPC local, con la escalabilidad, flexibilidad y elasticidad que ofrece la nube. AWS

En los siguientes temas se muestra cómo comenzar a usar EFA con la DLAMI.

> **Note**
>
> Elige tu DLAMI de esta lista de DLAMI de GPU base

Temas

- Lanzamiento de una instancia con EFA AWS Deep Learning AMI
- Uso del EFA en la DLAMI

Lanzamiento de una instancia con EFA AWS Deep Learning AMI

La DLAMI base más reciente está lista para utilizarse con EFA y viene con los controladores, módulos de kernel, libfabric, openmpi y el complemento de NCCL OFI necesarios para instancias de GPU.

Puede encontrar las versiones CUDA compatibles de una DLAMI base en las notas de la versión.

Nota:

- Al ejecutar una aplicación de NCCL mediante `mpirun` en EFA, tendrá que especificar la ruta completa a la instalación compatible con EFA como:

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

- Para permitir que la aplicación utilice EFA, añada `FI_PROVIDER="efa"` al comando `mpirun`, tal y como se muestra en Uso del EFA en la DLAMI.

Temas

- Preparación de un grupo de seguridad habilitado para EFA
- Lanzar la instancia
- Verificación de una asociación de EFA

Preparación de un grupo de seguridad habilitado para EFA

La EFA requiere un grupo de seguridad que permita que todo el tráfico entrante y saliente entre y hacia el propio grupo de seguridad. Para obtener más información, consulte la documentación de la EFA.

1. Abra la consola de Amazon EC2 en https://console.aws.amazon.com/ec2/.
2. En el panel de navegación, elija Security Groups (Grupos de seguridad) y, a continuación, elija Create Security Group (Crear grupo de seguridad).
3. En la ventana Create Security Group, haga lo siguiente:
   - En Security group name (Nombre del grupo de seguridad), ingrese un nombre descriptivo para el grupo de seguridad, como, por ejemplo, `EFA-enabled security group`.
   - (Opcional) En Description (Descripción), ingrese una breve descripción del grupo de seguridad.
   - En VPC, seleccione la VPC en la que desea lanzar sus instancias habilitadas para EFA.
   - Seleccione Create (Crear).
4. Seleccione el grupo de seguridad que ha creado y, en la pestaña Description (Descripción), copie el Group ID (ID de grupo).
5. En las pestañas Entrante y Saliente, haga lo siguiente:
   - Elija Edit (Editar).
   - En Type (Tipo), seleccione All traffic (Todo el tráfico).
   - En Source (Origen), seleccione Custom (Personalizado).
   - Pegue el ID del grupo de seguridad que copió en el campo.
   - Seleccione Guardar.

6. Habilite el tráfico de entrada que hace referencia a Autorización del tráfico de entrada para sus instancias de Linux. Si omite este paso, no podrá comunicarse con su instancia de DLAMI.

Lanzar la instancia

Actualmente, EFA on the AWS Deep Learning AMI es compatible con los siguientes tipos de instancias y sistemas operativos:

- P3DN.24xlarge: Amazon Linux 2, Ubuntu 20.04
- P4D.24xlarge: Amazon Linux 2, Ubuntu 20.04
- p5.48xlarge: Amazon Linux 2, Ubuntu 20.04

En la siguiente sección se muestra cómo lanzar una instancia de DLAMI habilitada para EFA. Para obtener más información, consulte el paso Lance instancias habilitadas para EFA en un grupo con ubicación en clúster.

1. Abra la consola de Amazon EC2 en https://console.aws.amazon.com/ec2/.
2. Elija Launch Instance (Lanzar instancia).
3. En la página Elija una AMI, seleccione una DLAMI compatible que se encuentra en la página de notas de la versión de DLAMI
4. En la página Elegir un tipo de instancia , seleccione uno de los tipos de instancias admitidos y, a continuación, elija Next: Configure Instance Details. Consulte este enlace para ver la lista de instancias compatibles: Introducción a EFA y MPI
5. En la página Configure Instance Details (Siguiente: Configurar detalles de instancia), haga lo siguiente:
   - En Number of instances (Número de instancias), introduzca el número de instancias habilitadas para EFA que desea lanzar.
   - En Network (Red) y Subnet (Subred), seleccione la VPC y la subred en la que lanzar las instancias.
   - [Opcional] En el grupo de ubicación, seleccione Añadir instancia al grupo de ubicación. Para lograr el mejor rendimiento, lance las instancias dentro de un grupo de ubicación.
   - [Opcional] En el nombre del grupo de ubicación, seleccione Añadir a un nuevo grupo de ubicación, introduzca un nombre descriptivo para el grupo de ubicación y, a continuación, en Estrategia de grupo de ubicación, seleccione clúster.

- Asegúrese de activar el “Elastic Fabric Adapter” en esta página. Si esta opción está deshabilitada, cambie la subred por una que admita el tipo de instancia seleccionado.
- En la sección Network Interfaces (Interfaces de red), para el dispositivo eth0, elija New network interface (Nueva interfaz de red). Como opción, puede especificar una dirección IPv4 principal y una o varias direcciones IPv4 secundarias. Si está lanzando la instancia en una subred que tenga un bloque de CIDR de IPv6 asociado, como opción puede especificar una dirección IPv6 principal y una o varias direcciones IPv6 secundarias.
- Elija Next: Add Storage (Siguiente: Añadir almacenamiento).

6. En la página Add Storage (Añadir almacenamiento), especifique los volúmenes que desea adjuntar a la instancia además de los volúmenes especificados por la AMI (como el volumen de dispositivo raíz) y, a continuación, elija Next: Add Tags (Siguiente: Añadir etiquetas).
7. En la página Add Tags (Añadir etiquetas), especifique etiquetas para las instancias, por ejemplo, un nombre fácil de recordar, y, a continuación, elija Next: Configure Security Group (Siguiente: Configurar grupo de seguridad).
8. En la página Configurar un grupo de seguridad, en Asignar un grupo de seguridad, seleccione Seleccionar un grupo de seguridad existente y, a continuación, seleccione el grupo de seguridad que creó anteriormente.
9. Elija Review and Launch (Revisar y lanzar).
10. En la página Review Instance Launch (Revisar lanzamiento de instancia), revise la configuración y, a continuación, elija Launch (Lanzar) para elegir un par de claves y lanzar las instancias.

Verificación de una asociación de EFA

En la consola

Tras lanzar la instancia, compruebe los detalles de la instancia en la AWS consola. Para ello, seleccione la instancia en la consola de EC2 y revise la pestaña Description (Descripción) en el panel inferior de la página. Busque el parámetro “Network Interfaces: eth0” y haga clic en eth0 para que aparezca una ventana emergente. Asegúrese de que la opción “Elastic Fabric Adapter” esté habilitada.

Si EFA no está habilitado, puede solucionarlo mediante las siguientes dos opciones:

- Finalizar la instancia EC2 y lanzar una nueva con los mismos pasos. Asegúrese de que EFA esté asociado.
- Asocie EFA a una instancia existente.

1. En la consola EC2, diríjase a Network Interfaces (Interfaces de red).
2. Haga clic en Create a Network Interface (Crear una interfaz de red).
3. Seleccione la misma subred en la que se encuentra la instancia.
4. Asegúrese de habilitar “Elastic Fabric Adapter” y haga clic en Crear.
5. Vuelva a la pestaña EC2 Instances (Instancias EC2) y seleccione su instancia.
6. Vaya a Actions: Instance State y detenga la instancia antes de asociar EFA.
7. En Actions (Acciones), seleccione Networking: Attach Network Interface (Redes: Asociar interfaz de red).
8. Seleccione la interfaz que acaba de crear y haga clic en Attach (Asociar).
9. Reinicie la instancia.

En la instancia

El siguiente script de prueba ya está presente en la DLAMI. Ejecútelo para asegurarse de que los módulos de kernel estén cargados correctamente.

```
$ fi_info -p efa
```

El resultado debería tener un aspecto similar al siguiente.

```
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
```

Verificación de la configuración del grupo de seguridad

El siguiente script de prueba ya está presente en la DLAMI. Ejecútelo para asegurarse de que el grupo de seguridad que creó esté configurado correctamente.

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

El resultado debería tener un aspecto similar al siguiente.

```
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
```

Si deja de responder o no se completa, asegúrese de que el grupo de seguridad tenga las reglas de entrada/salida correctas.

Uso del EFA en la DLAMI

En la siguiente sección se describe cómo utilizar EFA para ejecutar aplicaciones de varios nodos en AWS Deep Learning AMI.

Ejecución de aplicaciones de varios nodos con EFA

Para ejecutar una aplicación en un clúster de nodos, se requiere la siguiente configuración

Temas

- Habilitar SSH sin contraseña
- Creación de archivo de hosts
- Pruebas NCCL

Habilitar SSH sin contraseña

Seleccione un nodo del clúster como nodo principal. Los nodos restantes se denominan nodos miembro.

1. En el nodo principal, genere el par de claves RSA.

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

2. Cambie los permisos de la clave privada en el nodo principal.

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

3. Copie la clave `~/.ssh/id_rsa.pub` pública y añádala a `~/.ssh/authorized_keys` los nodos miembros del clúster.
4. Ahora debe poder iniciar sesión directamente en los nodos miembro desde el nodo principal mediante la ip privada.

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

5. Desactive la strictHostKey comprobación y habilite el reenvío de agentes en el nodo líder añadiendo lo siguiente al archivo ~/.ssh/config del nodo líder:

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

6. En las instancias de Amazon Linux 2, ejecute el siguiente comando en el nodo principal para proporcionar los permisos correctos al archivo de configuración:

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

Creación de archivo de hosts

En el nodo principal, cree un archivo de hosts para identificar los nodos del clúster. El archivo de hosts debe tener una entrada para cada nodo del clúster. Cree un archivo ~/hosts y añada cada nodo mediante la ip privada de la siguiente manera:

```
localhost slots=8
```

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

Pruebas NCCL

> **Note**
>
> Estas pruebas se realizaron con la versión 1.30.0 de la EFA y el complemento OFI NCCL 1.7.4.

A continuación, se muestra un subconjunto de pruebas NCCL realizadas por Nvidia para comprobar tanto la funcionalidad como el rendimiento en varios nodos de cómputo

Instancias compatibles: P3dn, P4, P5

Pruebas de funcionalidad

Prueba de nodos múltiples de transferencia de mensajes NCCL

La opción nccl_message_transfer es una prueba sencilla para garantizar que el complemento de NCCL OFI funciona según lo esperado. La prueba valida la funcionalidad de las API de transferencia de datos y de establecimiento de conexiones de NCCL. Asegúrese de utilizar la ruta completa a mpirun, tal y como se muestra en el ejemplo, mientras ejecuta aplicaciones de NCCL con EFA. Cambie los parámetros np y N en función del número de instancias y GPU del clúster. Para obtener más información, consulte la documentación de OFI NCCL de AWS.

La siguiente prueba nccl_message_transfer es para una versión genérica xx.x de CUDA. Puede ejecutar los comandos para cualquier versión de CUDA disponible en su instancia de Amazon EC2 sustituyendo la versión de CUDA en el 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
```

El resultado debería tener el siguiente aspecto. Puede comprobar la salida para ver si EFA se está utilizando como proveedor de 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
```

Pruebas de rendimiento

Prueba de rendimiento NCCL de varios nodos en P4d.24xlarge

Para comprobar el rendimiento de NCCL con EFA, ejecute la prueba de rendimiento de NCCL estándar que está disponible en el repositorio oficial de pruebas de NCCL. El DLAMI viene con esta prueba ya creada para CUDA XX.X. También puede ejecutar su propio script con EFA.

Cuando cree su propio script, siga estas directrices:

- Utilice la ruta completa a mpirun, tal y como se muestra en el ejemplo, mientras ejecuta aplicaciones NCCL con EFA.
- Cambie los parámetros np y N en función del número de instancias y GPU del clúster.
- Añada el indicador NCCL_DEBUG=INFO y asegúrese de que los registros indiquen el uso del EFA como “El proveedor seleccionado es EFA”.
- Defina la ubicación del registro de entrenamiento para analizarla y validarla

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

Utilice el comando `watch nvidia-smi` en cualquiera de los nodos miembro para monitorizar el uso de la GPU. Los siguientes comandos `watch nvidia-smi` son para la versión xx.x de CUDA y dependen del sistema operativo de la instancia. Puede ejecutar los comandos para cualquier versión de CUDA disponible en su instancia de Amazon EC2 sustituyendo la versión de CUDA en el 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}
```

El resultado debería tener el siguiente aspecto:

```
# 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
```

Pruebas de validación

Para validar que las pruebas de EFA arrojaron un resultado válido, utilice las siguientes pruebas para confirmarlo:

- Obtenga el tipo de instancia mediante los metadatos de la instancia de EC2:

```
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)
```

- Ejecute la Pruebas de rendimiento

- Establezca los siguientes parámetros

```
CUDA_VERSION
CUDA_RUNTIME_VERSION
NCCL_VERSION
```

- Valide los resultados como se muestra:

```
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
```

- Para acceder a los datos de referencia, podemos analizar la última fila del resultado de la tabla de la prueba all_reduce de varios nodos:

```
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}"
```

Monitorización y optimización de GPU

La siguiente sección le guiará en el proceso de configurar las opciones de optimización y monitorización de GPU. Esta sección está organizada como un flujo de trabajo típico en la que se monitoriza el procesamiento previo y el entrenamiento.

- Monitorización
  - Supervise las GPU con CloudWatch
- Optimización
  - Procesamiento previo
  - Formación

Monitorización

La DLAMI viene preinstalada con varias herramientas de supervisión de GPU. Esta guía también menciona las herramientas que están disponibles para su descarga e instalación.

- Supervise las GPU con CloudWatch- una utilidad preinstalada que informa a Amazon CloudWatch de las estadísticas de uso de la GPU.
- CLI de nvidia-smi: una utilidad para monitorizar el uso general de memoria y computación de GPU. Está preinstalado en su AWS Deep Learning AMI (DLAMI).
- Biblioteca de C de NVML: una API basada en C para obtener acceso directo a las funciones de administración y monitorización de GPU. Esta API la utiliza internamente la CLI de nvidia-smi y viene preinstalada en la DLAMI. También tiene enlaces a Python y Perl para facilitar el desarrollo en dichos lenguajes. La utilidad gpumon.py preinstalada en su DLAMI utiliza el paquete pynvml de. nvidia-ml-py
- NVIDIA DCGM: una herramienta de administración de clústeres. Visite la página del desarrollador para obtener información sobre cómo instalar y configurar esta herramienta.

> **Tip**
>
> Consulte el blog de desarrolladores de NVIDIA para obtener la información más reciente sobre el uso de las herramientas de CUDA instaladas en la DLAMI:
>
> - Supervisión de la TensorCore utilización mediante Nsight IDE y nvprof.

Supervise las GPU con CloudWatch

Cuando utilice la DLAMI con una GPU, es probable que sienta la necesidad de realizar un seguimiento de su uso durante el entrenamiento o la inferencia. Esto puede resultar útil para optimizar la canalización de datos y ajustar la red de aprendizaje profundo.

Hay dos formas de configurar las métricas de la GPU con CloudWatch:

- Configure las métricas con el AWS CloudWatch agente (recomendado)
- Configure las métricas con el script preinstalado `gpumon.py`

Configure las métricas con el AWS CloudWatch agente (recomendado)

Integre su DLAMI con el agente CloudWatch unificado para configurar las métricas de la GPU y supervisar la utilización de los coprocesos de la GPU en las instancias aceleradas de Amazon EC2.

Hay cuatro formas de configurar las métricas de la GPU con su DLAMI:

- Configuración de métricas de GPU mínimas
- Configuración de métricas de GPU parciales
- Configure todas las métricas de GPU disponibles
- Configuración de métricas de GPU personalizadas

Para obtener más información sobre actualizaciones y parches de seguridad, consulte Parches de seguridad para el agente AWS CloudWatch .

Requisitos previos

Para empezar, debe configurar los permisos de IAM de la instancia Amazon EC2 que permitan a la instancia enviar métricas a ella. CloudWatch Para ver los pasos detallados, consulte Crear roles y usuarios de IAM para usarlos con el agente. CloudWatch

Configuración de métricas de GPU mínimas

Configure las métricas mínimas de GPU mediante el servicio `systemd` de `dlami-cloudwatch-agent@minimal`. En este servicio se configuran las siguientes métricas:

- `utilization_gpu`
- `utilization_memory`

Puede encontrar el servicio `systemd` de métricas mínimas de GPU preconfiguradas en la siguiente ubicación:

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

Active e inicie el servicio `systemd` con los siguientes comandos:

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

Configuración de métricas de GPU parciales

Configure las métricas parciales de GPU mediante el servicio `systemd` de `dlami-cloudwatch-agent@partial`. En este servicio se configuran las siguientes métricas:

- `utilization_gpu`
- `utilization_memory`
- `memory_total`
- `memory_used`
- `memory_free`

Puede encontrar el servicio `systemd` de métricas parciales de GPU preconfiguradas en la siguiente ubicación:

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

Active e inicie el servicio `systemd` con los siguientes comandos:

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

Configure todas las métricas de GPU disponibles

Configure todas las métricas disponibles de GPU mediante el servicio `systemd` de `dlami-cloudwatch-agent@all`. En este servicio se configuran las siguientes métricas:

- `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

Puede encontrar el servicio systemd de todas las métricas disponibles de GPU preconfiguradas en la siguiente ubicación:

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

Active e inicie el servicio systemd con los siguientes comandos:

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

Configuración de métricas de GPU personalizadas

Si las métricas preconfiguradas no cumplen sus requisitos, puede crear un archivo de configuración de CloudWatch agente personalizado.

Para crear un archivo de configuración personalizado

Para crear un archivo de configuración personalizado, consulte los pasos detallados en Crear o editar manualmente el archivo de configuración del CloudWatch agente.

Para este ejemplo, suponga que la definición del esquema se encuentra en `/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json`.

Configure las métricas con su archivo personalizado

Ejecute el siguiente comando para configurar el CloudWatch agente según su archivo personalizado:

```
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
```

Parches de seguridad para el agente AWS CloudWatch

Los DLAMI recién lanzados están configurados con los últimos parches de seguridad disponibles para los AWS CloudWatch agentes. Consulte las siguientes secciones para actualizar su DLAMI actual con los últimos parches de seguridad en función del sistema operativo que elija.

Amazon Linux 2

Úselo `yum` para obtener los parches de seguridad de AWS CloudWatch agentes más recientes para una DLAMI de Amazon Linux 2.

```
sudo yum update
```

Ubuntu

Para obtener los últimos parches AWS CloudWatch de seguridad para una DLAMI con Ubuntu, es necesario volver a instalar AWS CloudWatch el agente mediante un enlace de descarga de Amazon S3.

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

Para obtener más información sobre la instalación del AWS CloudWatch agente mediante los enlaces de descarga de Amazon S3, consulte Instalación y ejecución del CloudWatch agente en sus servidores.

Configure las métricas con el script preinstalado **`gpumon.py`**

Una utilidad denominada gpumon.py viene preinstalada en la DLAMI. Se integra CloudWatch y admite la supervisión del uso por GPU: memoria de la GPU, temperatura de la GPU y potencia de la

GPU. El script envía periódicamente los datos monitorizados a CloudWatch. Puede configurar el nivel de granularidad de los datos a los que se envían CloudWatch cambiando algunos ajustes del script. Sin embargo, antes de iniciar el script, tendrá que configurarlo CloudWatch para recibir las métricas.

Cómo configurar y ejecutar la supervisión de la GPU con CloudWatch

1. Cree un usuario de IAM o modifique uno existente para tener una política en la que publicar la métrica. CloudWatch Si crea un usuario nuevo, anote las credenciales, ya que las necesitará en el siguiente paso.

   La política de IAM que hay que buscar es «cloudwatch:». PutMetricData La política que se añade es la siguiente:

   ```
   {
      "Version": "2012-10-17",
      "Statement": [
           {
               "Action": [
                   "cloudwatch:PutMetricData"
                ],
                "Effect": "Allow",
                "Resource": "*"
           }
      ]
   }
   ```

   > **Tip**
   >
   > Para obtener más información sobre cómo crear un usuario de IAM y añadir políticas CloudWatch, consulte la documentación. CloudWatch

2. En la DLAMI, ejecute el comando AWS configure y especifique las credenciales de usuario de IAM.

   ```
   $ aws configure
   ```

3. Es posible que tenga que realizar algunas modificaciones en la utilidad gpumon antes de ejecutarla. Puede encontrar la utilidad gpumon y el archivo README en la ubicación definida en el siguiente bloque de código. Para obtener más información sobre el script `gpumon.py`, consulte la ubicación del script en Amazon S3.

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

Opciones:

- Cambie la región en gpumon.py si la instancia NO está en us-east-1.
- Cambie otros parámetros, como el período del informe CloudWatch `namespace` o el período sobre el que se informa, con`store_reso`.

4. Actualmente, el script solo es compatible con Python 3. Active el entorno de Python 3 de su marco de trabajo preferido o active el entorno general de Python 3 de la DLAMI.

```
$ source activate python3
```

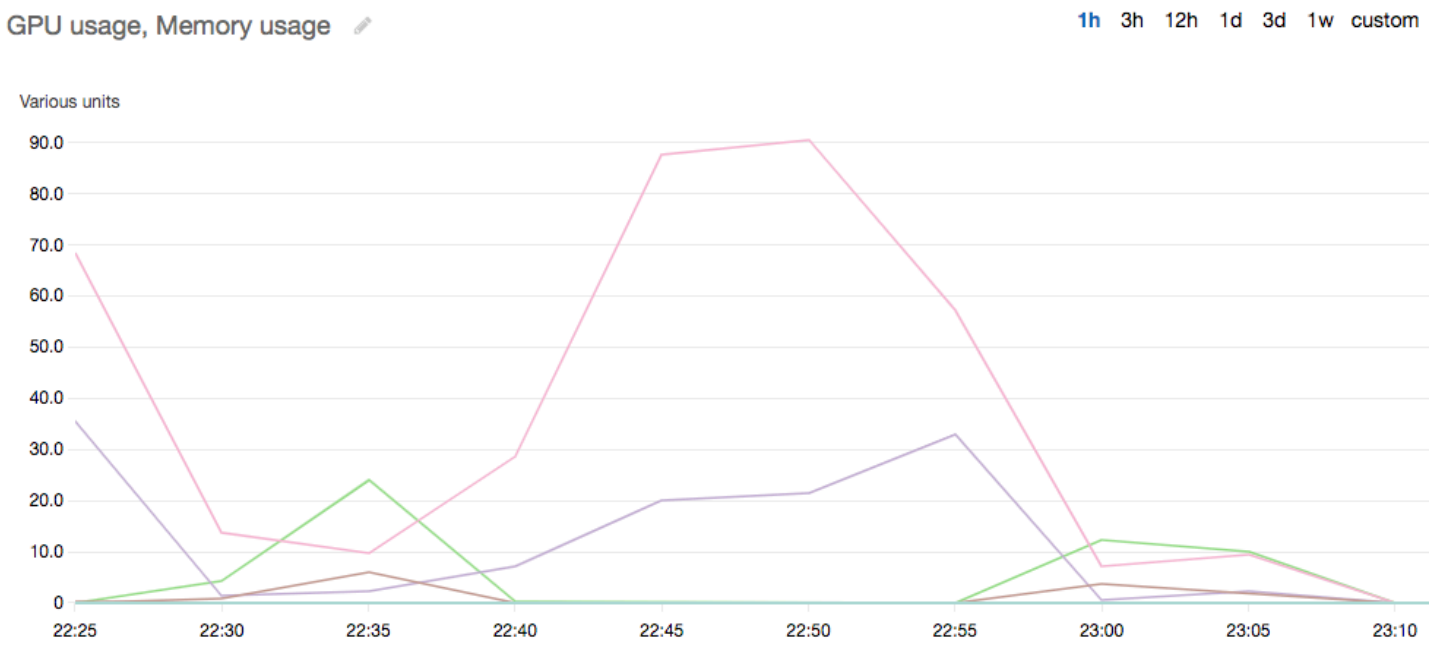
5. Ejecute la utilidad gpumon en segundo plano.

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

6. Abra https://console.aws.amazon.com/cloudwatch/ en su navegador y seleccione la métrica. Tendrá un espacio de nombres ". DeepLearningTrain

> **Tip**
>
> Puede cambiar el espacio de nombres modificando gpumon.py. También puede modificar el intervalo de notificación ajustando `store_reso`.

El siguiente es un ejemplo de CloudWatch gráfico que informa sobre una ejecución de gpumon.py supervisando un trabajo de entrenamiento en una instancia p2.8xlarge.



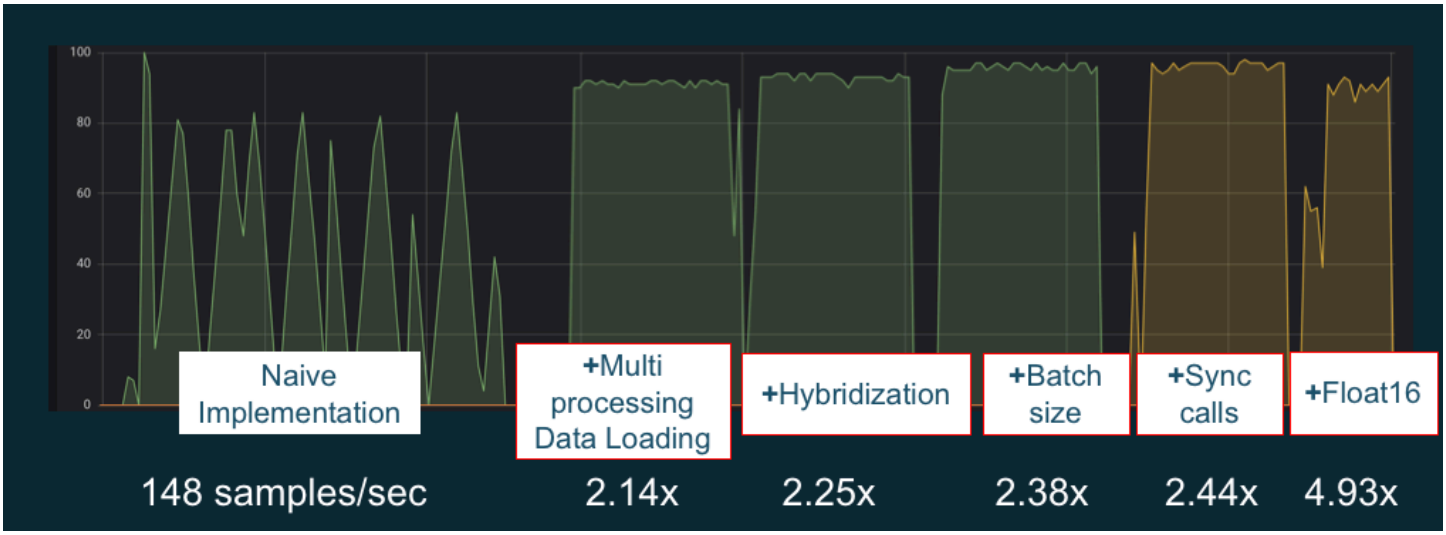
Es posible que le interesen estos otros temas sobre la monitorización y optimización de GPU:

- Monitorización
  - Supervise las GPU con CloudWatch
- Optimización
  - Procesamiento previo
  - Formación

Optimización

Para sacar el máximo partido de las GPU, puede optimizar la canalización de datos y ajustar la red de aprendizaje profundo. Tal como se describe en el siguiente gráfico, una implementación ingenua o básica de una red neuronal podría utilizar la GPU de un modo incoherente y no aprovechar todo su potencial. Al optimizar el procesamiento previo y la carga de datos, es posible reducir el cuello de botella de la CPU a la GPU. Puede ajustar la propia red neuronal mediante la hibridación (si el marco de trabajo la admite), ajustando el tamaño del lote y sincronizando las llamadas. También puede utilizar entrenamiento de precisión múltiple (float16 o int8) en la mayoría de los marcos de trabajo, lo que puede tener un efecto drástico en la mejora del rendimiento.

El gráfico siguiente muestra la mejora acumulativa del rendimiento que se obtiene cuando se aplican distintas optimizaciones. Los resultados dependerán de los datos que esté procesando y de la red que esté optimizando.



Ejemplo de optimizaciones del rendimiento de la GPU. Fuente del gráfico: Performance Tricks with MXNet Gluon

Las siguientes guías introducen opciones que funcionarán con la DLAMI y que le ayudarán a aumentar el rendimiento de la GPU.

Temas

- Procesamiento previo
- Formación

Procesamiento previo

El procesamiento previo de los datos a través de transformaciones o aumentos es con frecuencia un proceso asociado a la CPU y puede ser el cuello de botella en la canalización general. Los marcos de trabajo tienen operadores integrados para el procesamiento de imágenes, pero DALI (Data Augmentation Library) demuestra un rendimiento mejorado sobre las opciones integradas de los marcos de trabajo.

- Biblioteca de aumento de datos de NVIDIA (DALI): DALI descarga el aumento de datos a la GPU. No está preinstalada en la DLAMI, pero puede obtener acceso a ella instalando o cargando un contenedor de marcos de trabajo compatible en la DLAMI o en otra instancia de Amazon Elastic Compute Cloud. Consulte la página del proyecto DALI en el sitio web de NVIDIA para obtener más información. Para ver un ejemplo de caso de uso y descargar ejemplos de código, consulta el ejemplo sobre el rendimiento del entrenamiento SageMaker previo al procesamiento.

- nvJPEG: una biblioteca de decodificadores JPEG acelerada por GPU para programadores de C. Admite la decodificación de imágenes únicas o de lotes de imágenes, así como las operaciones de transformación subsiguientes tan frecuentes en el aprendizaje profundo. nvJPEG viene integrada en DALI, pero puede descargarla en la página de nvjpeg del sitio web de NVIDIA y utilizarla de forma independiente.

Es posible que le interesen estos otros temas sobre la monitorización y optimización de GPU:

- Monitorización
  - Supervise las GPU con CloudWatch
- Optimización
  - Procesamiento previo
  - Formación

Formación

El entrenamiento de precisión mixta permite implementar redes de mayor tamaño con la misma cantidad de memoria, o reducir el uso de esta en comparación con las redes de precisión única o doble, lo que se traduce en un aumento del rendimiento informático. También ofrece el beneficio de transferencias de datos más pequeñas y rápidas, un factor importante en el entrenamiento distribuido con varios nodos. Para utilizar el entrenamiento de precisión mixta es necesario ajustar el envío de datos y el escalado de pérdidas. Las siguientes guías describen cómo realizar esta operación en los marcos de trabajo compatibles con la precisión mixta.

- SDK de aprendizaje profundo de NVIDIA: documentos en el sitio web de NVIDIA que describen la implementación de precisión mixta para MXNet, y. PyTorch TensorFlow

> **Tip**
>
> Asegúrese de consultar el sitio web de su marco de trabajo preferido y busque "mixed precision" o "fp16" para conocer las técnicas de optimización más recientes. A continuación se muestran algunas guías de precisión mixta que pueden resultarle de utilidad:
>
> - Formación de precisión mixta con TensorFlow (vídeo): en el blog de NVIDIA.
> - Mixed-precision training using float16 with MXNet: un artículo con preguntas frecuentes del sitio web de MXNet.

- NVIDIA Apex: una herramienta para un entrenamiento sencillo de precisión mixta con un artículo de blog en el PyTorch sitio web de NVIDIA.

Es posible que le interesen estos otros temas sobre la monitorización y optimización de GPU:

- Monitorización
  - Supervise las GPU con CloudWatch
- Optimización
  - Procesamiento previo
  - Formación

El chip AWS Inferentia con DLAMI

AWS Inferentia es un chip de aprendizaje automático personalizado diseñado por AWS el usuario para realizar predicciones de inferencias de alto rendimiento. Para usar el chip, configure una instancia de Amazon Elastic Compute Cloud y use el kit de desarrollo de software (SDK) AWS Neuron para invocar el chip Inferentia. Para proporcionar a los clientes la mejor experiencia de Inferentia, Neuron está incorporado en la AWS Deep Learning AMI (DLAMI).

En los siguientes temas se muestra cómo comenzar a usar Inferentia con la DLAMI.

Contenido

- Lanzamiento de una instancia DLAMI con Neuron AWS
- Uso del DLAMI con Neuron AWS

Lanzamiento de una instancia DLAMI con Neuron AWS

La última versión de DLAMI está lista para usarse AWS con Inferentia e incluye el paquete de API Neuron. AWS Para iniciar una instancia de la DLAMI, consulte Lanzamiento y configuración de una DLAMI. Una vez que tenga un DLAMI, siga estos pasos para asegurarse de que AWS su chip AWS Inferentia y sus recursos de Neuron estén activos.

Contenido

- Comprobación de la instancia
- Identificación de los dispositivos de inferencia AWS

- Visualización del uso de recursos
- Uso de Neuron Monitor (monitor de neuronas)
- Actualización del software Neuron

Comprobación de la instancia

Antes de usar la instancia, compruebe que esté correctamente instalada y configurada con Neuron.

Identificación de los dispositivos de inferencia AWS

Para identificar el número de dispositivos de Inferencia de la instancia, utilice el siguiente comando:

```
neuron-ls
```

Si su instancia tiene dispositivos de Inferentia asociados a ella, la salida tendrá un aspecto similar al siguiente:

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

El resultado suministrado se toma de una instancia INF1.6xLarge e incluye las siguientes columnas:

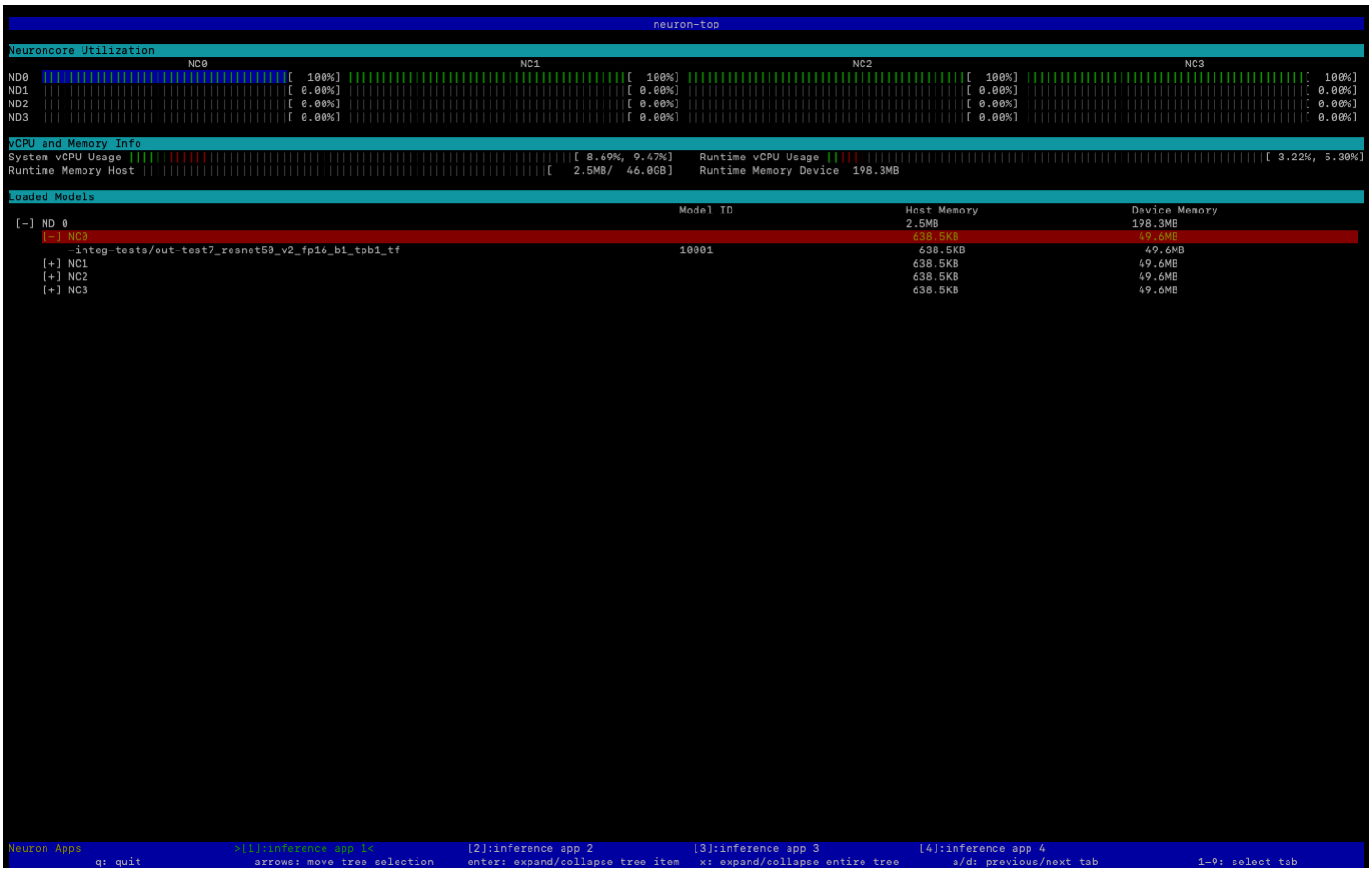
- DISPOSITIVO NEURONAL: el identificador lógico asignado al. NeuronDevice Este ID se usa al configurar varios tiempos de ejecución para usar diferentes. NeuronDevices
- NÚCLEOS NEURONALES: el número de núcleos NeuronCores presentes en. NeuronDevice
- MEMORIA NEURONAL: La cantidad de memoria DRAM en el. NeuronDevice
- DISPOSITIVOS CONECTADOS: Otros NeuronDevices conectados al. NeuronDevice
- PCI BDF: El identificador de la función de dispositivo de bus PCI (BDF) del. NeuronDevice

Visualización del uso de recursos

Vea información útil sobre NeuronCore el uso de la vCPU, el uso de la memoria, los modelos cargados y las aplicaciones de Neuron con el comando. `neuron-top` Si se inicia `neuron-top` sin argumentos, se mostrarán los datos de todas las aplicaciones de aprendizaje automático que se utilicen. NeuronCores

```
neuron-top
```

Cuando una aplicación utiliza cuatro NeuronCores, el resultado debe tener un aspecto similar al de la imagen siguiente:



Para obtener más información sobre los recursos para supervisar y optimizar las aplicaciones de inferencia basadas en Neuron, consulte Neuron Tools.

Uso de Neuron Monitor (monitor de neuronas)

Neuron Monitor recopila las métricas de los tiempos de ejecución de Neuron que se ejecutan en el sistema y transmite los datos recopilados a la salida estándar en formato JSON. Estas métricas se

organizan en grupos de métricas que se configuran proporcionando un archivo de configuración. Para obtener más información sobre Neuron Monitor, consulte la User Guide for Neuron Monitor.

Actualización del software Neuron

Para obtener información sobre cómo actualizar el software Neuron SDK en DLAMI, consulte AWS la Guía de configuración de Neuron.

Paso siguiente

Uso del DLAMI con Neuron AWS

Uso del DLAMI con Neuron AWS

Un flujo de trabajo típico con el SDK de AWS Neuron consiste en compilar un modelo de aprendizaje automático previamente entrenado en un servidor de compilación. Después, distribuya los artefactos a las instancias de Inf1 para su ejecución. AWS Deep Learning AMI (DLAMI) viene preinstalado con todo lo que necesita para compilar y ejecutar inferencias en una instancia de Inf1 que usa Inferentia.

En las siguientes secciones se describe cómo utilizar la DLAMI con Inferentia.

Contenido

TensorFlowUso de AWS -Neuron y el compilador Neuron

Este tutorial muestra cómo usar el compilador AWS Neuron para compilar el modelo Keras ResNet -50 y exportarlo como un modelo guardado en formato. SavedModel Este formato es un formato típico TensorFlow de modelos intercambiables. También aprenderá a ejecutar la inferencia en una instancia Inf1 con entrada de ejemplo.

Para obtener más información sobre el SDK de Neuron, consulte la documentación del SDK de AWS Neuron.

Contenido

- Activación del entorno Conda
- Compilación de Resnet50
- ResNet50 Inferencia

Requisitos previos

Antes de utilizar este tutorial, debería haber completado los pasos de configuración de Lanzamiento de una instancia DLAMI con Neuron AWS. También debe estar familiarizado con el aprendizaje profundo y con el uso de la DLAMI.

Activación del entorno Conda

Active el entorno conda TensorFlow -Neuron mediante el siguiente comando:

```
source activate aws_neuron_tensorflow_p36
```

Para salir del entorno Conda actual, ejecute el siguiente comando:

```
source deactivate
```

Compilación de Resnet50

Cree un script de Python llamada denominado **`tensorflow_compile_resnet50.py`** que tenga el siguiente contenido. Este script de Python compila el modelo Keras ResNet 50 y lo exporta como un modelo guardado.

```
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')
```

Compile el modelo con el siguiente comando:

```
python tensorflow_compile_resnet50.py
```

El proceso de compilación tardará unos minutos. Cuando concluya, la salida debe tener el siguiente aspecto:

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

Después de la compilación, el modelo guardado se comprime en **`ws_resnet50/resnet50_neuron.zip`**. Descomprima el modelo y descargue la imagen de muestra para la inferencia mediante los siguientes comandos:

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

ResNet50 Inferencia

Cree un script de Python llamada denominado **`tensorflow_infer_resnet50.py`** que tenga el siguiente contenido. Este script ejecuta la inferencia en el modelo descargado utilizando un modelo de inferencia compilado previamente.

```
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])
```

Ejecute la inferencia en el modelo mediante el siguiente comando:

```
python tensorflow_infer_resnet50.py
```

El resultado debería tener el siguiente aspecto:

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

Paso siguiente

Uso de AWS Neuron Serving TensorFlow

Uso de AWS Neuron Serving TensorFlow

Este tutorial muestra cómo construir un gráfico y añadir un paso de compilación de AWS Neuron antes de exportar el modelo guardado para usarlo con TensorFlow Serving. TensorFlow Serving es un sistema de servidor que permite ampliar las inferencias en una red. Neuron TensorFlow Serving utiliza la misma API que el Serving normal. TensorFlow La única diferencia es que se debe compilar un modelo guardado para AWS Inferentia y el punto de entrada es un nombre binario diferente. `tensorflow_model_server_neuron` El binario se encuentra en `/usr/local/bin/tensorflow_model_server_neuron` y está preinstalado en la DLAMI.

Para obtener más información sobre el SDK de Neuron, consulte la documentación del SDK de AWS Neuron.

Contenido

Requisitos previos

Antes de utilizar este tutorial, debería haber completado los pasos de configuración de Lanzamiento de una instancia DLAMI con Neuron AWS. También debe estar familiarizado con el aprendizaje profundo y con el uso de la DLAMI.

Activación del entorno Conda

Active el entorno conda TensorFlow -Neuron mediante el siguiente comando:

```
source activate aws_neuron_tensorflow_p36
```

Si necesita salir del entorno Conda actual, ejecute:

```
source deactivate
```

Compilación y exportación del modelo guardado

Cree un script de Python denominado `tensorflow-model-server-compile.py` con el siguiente contenido. Este script construye un gráfico y lo compila con Neuron. A continuación, exporta el gráfico compilado como un modelo guardado.

```
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)
```

Compile el modelo con el siguiente comando:

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

El resultado debería tener el siguiente aspecto:

```
...
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
```

Distribución del modelo guardado

Una vez compilado el modelo, puede usar el siguiente comando para distribuir el modelo guardado con el binario tensorflow_model_server_neuron:

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

El resultado debería tener el siguiente aspecto. El servidor almacena el modelo compilado de manera provisional en la DRAM del dispositivo de Inferentia para preparar la inferencia.

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

Generación de solicitudes de inferencia al servidor de modelos

Cree un script de Python denominado `tensorflow-model-server-infer.py` con el siguiente contenido. Este script ejecuta la inferencia a través de gRPC, que es el marco de trabajo de servicio.

```
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))
```

Ejecute la inferencia en el modelo utilizando gRPC con el siguiente comando:

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

El resultado debería tener el siguiente aspecto:

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

Uso de MXnet-Neuron y el compilador Neuron AWS

La API de compilación MXNet-Neuron proporciona un método para compilar un gráfico modelo que se puede ejecutar en un dispositivo Inferentia. AWS

En este ejemplo, usa la API para compilar un modelo ResNet -50 y usarlo para ejecutar inferencias.

Para obtener más información sobre el SDK de Neuron, consulte la documentación del SDK de AWS Neuron.

Contenido

Requisitos previos

Antes de utilizar este tutorial, debería haber completado los pasos de configuración de Lanzamiento de una instancia DLAMI con Neuron AWS. También debe estar familiarizado con el aprendizaje profundo y con el uso de la DLAMI.

Activación del entorno Conda

Active el entorno Conda de MXNet-Neuron mediante el siguiente comando:

```
source activate aws_neuron_mxnet_p36
```

Para salir del entorno Conda actual, ejecute:

```
source deactivate
```

Compilación de Resnet50

Cree un script de Python denominado **`mxnet_compile_resnet50.py`** con el siguiente contenido. Este script utiliza la API de Python de compilación MXNet-Neuron para ResNet compilar un modelo -50.

```
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)
```

Compile el modelo con el siguiente comando:

```
python mxnet_compile_resnet50.py
```

La compilación tardará unos minutos. Cuando haya finalizado, los siguientes archivos estarán en su directorio actual:

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

ResNetInferencia 5.0

Cree un script de Python denominado **`mxnet_infer_resnet50.py`** con el siguiente contenido. Este script descarga una imagen de muestra y la utiliza para ejecutar la inferencia con el modelo compilado.

```
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]))
```

Ejecute la inferencia con el modelo compilado mediante el siguiente comando:

```
python mxnet_infer_resnet50.py
```

El resultado debería tener el siguiente aspecto:

```
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
```

Paso siguiente

[Uso de la distribución de modelos de MXNet-Neuron]

Uso de la distribución de modelos de MXNet-Neuron

En este tutorial, aprenderá a utilizar un modelo de MXNet previamente entrenado para realizar la clasificación de imágenes en tiempo real con Multi Model Server (MMS). El MMS es una easy-to-use herramienta flexible para utilizar modelos de aprendizaje profundo que se entrenan con cualquier marco de aprendizaje automático o aprendizaje profundo. Este tutorial incluye un paso de compilación con AWS Neuron y una implementación de MMS con MXNet.

Para obtener más información sobre el SDK de Neuron, consulte la documentación del SDK de AWS Neuron.

Contenido

Requisitos previos

Antes de utilizar este tutorial, debería haber completado los pasos de configuración de Lanzamiento de una instancia DLAMI con Neuron AWS. También debe estar familiarizado con el aprendizaje profundo y con el uso de la DLAMI.

Activación del entorno Conda

Active el entorno Conda de MXNet-Neuron mediante el siguiente comando:

```
source activate aws_neuron_mxnet_p36
```

Para salir del entorno Conda actual, ejecute:

```
source deactivate
```

Descarga del código de ejemplo

Para ejecutar este ejemplo, descargue el código de ejemplo mediante los siguientes comandos:

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

Compile el modelo.

Cree un script de Python denominado `multi-model-server-compile.py` con el siguiente contenido. Este script compila el modelo ResNet 50 con el objetivo 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)
```

Para compilar el modelo, utilice el siguiente comando:

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

El resultado debería tener el siguiente aspecto:

```
...
[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!
```

Cree un archivo `signature.json` con el siguiente contenido para configurar el nombre y la forma de entrada:

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

Descargue el archivo `synset.txt` con el siguiente comando. Este archivo es una lista de nombres para ImageNet las clases de predicción.

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

Cree una clase de servicio personalizada siguiendo la plantilla de la carpeta `model_server_template`. Copie la plantilla en su directorio de trabajo actual mediante el siguiente comando:

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

Edite el módulo `mxnet_model_service.py` para reemplazar el contexto `mx.cpu()` por el contexto `mx.neuron()` de la siguiente manera. También necesita comentar la copia de datos innecesaria para `model_input`, porque MXNet-Neuron no es compatible con las API NDArray y 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]
```

Empaquete el modelo con model-archiver utilizando los siguientes comandos:

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

Ejecutar inferencia

Inicie Multi Model Server y cargue el modelo que utiliza la API RESTful mediante los siguientes comandos. Asegúrese de que neuron-rtd se está ejecutando con la configuración predeterminada.

```
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
```

Ejecute la inferencia utilizando una imagen de ejemplo con los siguientes comandos:

```
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
```

El resultado debería tener el siguiente aspecto:

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

Para limpiar después de la prueba, ejecute un comando delete a través de la API RESTful y detenga el servidor modelo mediante los siguientes comandos:

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

multi-model-server --stop
```

Debería ver los siguientes datos de salida:

```
{
  "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
```

Uso de PyTorch -Neuron y el compilador Neuron AWS

La API de compilación PyTorch -Neuron proporciona un método para compilar un gráfico modelo que se puede ejecutar en un dispositivo de inferencia. AWS

Un modelo entrenado debe compilarse en un destino de Inferentia antes de poder implementarlo en instancias Inf1. El siguiente tutorial compila el modelo torchvision ResNet 50 y lo exporta como un módulo guardado. TorchScript A continuación, el modelo se utiliza para ejecutar la inferencia.

Para mayor comodidad, el tutorial utiliza una instancia Inf1 tanto para la compilación como para la inferencia. En la práctica, puede compilar el modelo con otro tipo de instancia, como la familia de instancias c5. A continuación, debe implementar el modelo compilado en el servidor de inferencia Inf1. Para obtener más información, consulte la documentación del SDK de AWS Neuron PyTorch .

Contenido

Requisitos previos

Antes de utilizar este tutorial, debería haber completado los pasos de configuración de Lanzamiento de una instancia DLAMI con Neuron AWS. También debe estar familiarizado con el aprendizaje profundo y con el uso de la DLAMI.

Activación del entorno Conda

Active el entorno conda PyTorch -Neuron mediante el siguiente comando:

```
source activate aws_neuron_pytorch_p36
```

Para salir del entorno Conda actual, ejecute:

```
source deactivate
```

Compilación de Resnet50

Cree un script de Python denominado **`pytorch_trace_resnet50.py`** con el siguiente contenido. Este script usa la API de Python de PyTorch compilación -Neuron para compilar un modelo ResNet -50.

> **Note**
>
> Hay una dependencia entre las versiones de torchvision y el paquete de torch que debe tener en cuenta al compilar los modelos de torchvision. Estas reglas de dependencia se pueden gestionar a través de pip. Torchvision==0.6.1 coincide con la versión torch==1.5.1, mientras que torchvision==0.8.2 coincide con la versión 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")
```

Ejecute el script de compilación.

```
python pytorch_trace_resnet50.py
```

La compilación tardará unos minutos. Cuando haya finalizado, el modelo compilado se guardará como `resnet50_neuron.pt` en el directorio local.

ResNetInferencia 5.0

Cree un script de Python denominado **`pytorch_infer_resnet50.py`** con el siguiente contenido. Este script descarga una imagen de muestra y la utiliza para ejecutar la inferencia con el modelo compilado.

```
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) )
```

Ejecute la inferencia con el modelo compilado mediante el siguiente comando:

```
python pytorch_infer_resnet50.py
```

El resultado debería tener el siguiente aspecto:

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

El DLAMI de Graviton

AWS Las GPU DLAMI de Graviton están diseñadas para ofrecer un alto rendimiento y rentabilidad para las cargas de trabajo de aprendizaje profundo. En concreto, el tipo de instancia G5g incluye el procesador AWS Graviton2 basado en ARM, creado desde cero AWS y optimizado para la forma en que los clientes ejecutan sus cargas de trabajo en la nube. AWS Las DLAMI de GPU Graviton están preconfiguradas con Docker, NVIDIA Docker, NVIDIA Driver, CUDA, cuDNN, NCCL y TensorRT, así como con marcos de aprendizaje automático populares como y. TensorFlow PyTorch

Con el tipo de instancia G5g, puede aprovechar las ventajas de precio y rendimiento de Graviton2 para implementar modelos de aprendizaje profundo acelerados por GPU a un costo significativamente menor en comparación con las instancias basadas en x86 con aceleración por GPU.

Seleccione un DLAMI de Graviton

Lance una instancia de G5G con la DLAMI de Graviton que prefiera.

Para step-by-step obtener instrucciones sobre el lanzamiento de una DLAMI, consulte Lanzamiento y configuración de una DLAMI.

Para obtener una lista de los DLAMI de Graviton más recientes, consulte las notas de la versión de DLAMI.

Introducción

En los siguientes temas se muestra cómo comenzar a usar el DLAMI de Graviton.

Contenido

Uso de la DLAMI con GPU Graviton

AWS Deep Learning AMI Está lista para usarse con las GPU Graviton basadas en el procesador Arm. La DLAMI base con GPU Graviton incluye una plataforma básica de controladores de GPU y bibliotecas de aceleración que le permiten implementar su propio entorno de aprendizaje profundo personalizado. Docker y NVIDIA Docker vienen preconfigurados en la DLAMI de la GPU Graviton para que pueda implementar aplicaciones en contenedores. Consulte las notas de la versión para obtener más información sobre la DLAMI con GPU Graviton.

Contenido

Verifique el estado de la GPU

Use la interfaz de administración del sistema NVIDIA para comprobar el estado de su GPU Graviton.

```
nvidia-smi
```

El resultado del comando `nvidia-smi` será similar a lo siguiente.

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

Compruebe la versión de CUDA

Para verificar la versión de CUDA, ejecute el siguiente comando:

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

El resultado debería tener un aspecto similar al siguiente:

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

Verifique Docker

Ejecute un contenedor CUDA desde DockerHubpara verificar la funcionalidad de Docker en su GPU Graviton:

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

El resultado debería tener un aspecto similar al siguiente:

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

Use el siguiente comando para acceder a la herramienta de línea de comandos de TensorRT:

```
trtexec
```

El resultado debería tener un aspecto similar al siguiente:

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

Hay ruedas Python de TensorRT disponibles para su instalación bajo demanda. Las encontrará en las siguientes ubicaciones:

```
/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
```

Para obtener más información, consulte la documentación de NVIDIA TensorRT.

Ejecute ejemplos de CUDA

La DLAMI con GPU Graviton proporciona ejemplos de CUDA precompilados para ayudarle a comprobar las diferentes funcionalidades de CUDA.

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

Por ejemplo, ejecute el ejemplo de `vectorAdd` con el siguiente comando:

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

El resultado debería tener un aspecto similar al siguiente:

```
[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
```

Ejecute el ejemplo de `transpose`:

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

El resultado debería tener un aspecto similar al siguiente:

```
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
```

Tema siguiente

Uso de la GPU TensorFlow DLAMI de Graviton

Uso de la GPU TensorFlow DLAMI de Graviton

AWS Deep Learning AMI Está lista para usarse con las GPU Graviton basadas en el procesador Arm y viene optimizada para ello. TensorFlow La TensorFlow DLAMI de la GPU Graviton incluye un entorno Python TensorFlow preconfigurado con Serving para casos de uso de inferencias de aprendizaje profundo. Consulta las notas de la versión para obtener más información sobre la GPU TensorFlow DLAMI Graviton.

Contenido

Verifique la disponibilidad del servicio TensorFlow

Ejecute el siguiente comando para verificar la disponibilidad y la versión de Serving: TensorFlow

```
tensorflow_model_server --version
```

El resultado debería tener un aspecto similar al siguiente:

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

Verifique TensorFlow y TensorFlow sirva la disponibilidad de la API

Ejecute el siguiente comando para verificar la disponibilidad TensorFlow y la TensorFlow API de servicio:

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

Si el comando se ejecuta correctamente, no hay respuesta.

Ejecute un ejemplo de inferencia con Serving TensorFlow

Usa los siguientes comandos para descargar un modelo ResNet 50 previamente entrenado y ejecutar la inferencia mediante 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" &
```

El resultado debería tener un aspecto similar al siguiente:

```
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 ...
```

Utilice el `resnet_client` ejemplo de TensorFlow Serving para ejecutar la inferencia:

```
python3 serving/tensorflow_serving/example/resnet_client.py
```

El resultado debería tener un aspecto similar al siguiente:

```
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
```

Deje de TensorFlow servir con el siguiente comando:

```
kill $(pidof tensorflow_model_server)
```

Tema siguiente

Uso de la GPU PyTorch DLAMI de Graviton

Uso de la GPU PyTorch DLAMI de Graviton

AWS Deep Learning AMI Está lista para usarse con las GPU Graviton basadas en el procesador Arm y viene optimizada para ello. PyTorch La PyTorch DLAMI de la GPU Graviton incluye un entorno Python PyTorchpreconfigurado TorchVisioncon y para casos de uso de TorchServeinferencia y entrenamiento de aprendizaje profundo. Consulta las notas de la versión para obtener más información sobre la GPU PyTorch DLAMI Graviton.

Contenido

- Verificar el entorno de PyTorch Python
- Ejecute un ejemplo de entrenamiento con PyTorch
- Ejecute un ejemplo de inferencia con PyTorch

Verificar el entorno de PyTorch Python

Conéctese a su instancia de G5g y active el entorno base de Conda con el siguiente comando:

```
source activate base
```

La línea de comandos debe indicar que está trabajando en el entorno base de Conda, que contiene PyTorch TorchVision, y otras bibliotecas.

```
(base) $
```

Compruebe las rutas de herramientas predeterminadas del PyTorch entorno:

```
(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
```

Compruebe que Torch y TorchVersion X estén disponibles, compruebe sus versiones y compruebe su funcionalidad básica:

```
(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)
```

Ejecute un ejemplo de entrenamiento con PyTorch

Ejecute un ejemplo de trabajo de entrenamiento sobre MNIST:

```
git clone https://github.com/pytorch/examples.git
cd examples/mnist
python main.py
```

El resultado debería tener un aspecto similar al siguiente:

```
...
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%)
```

Ejecute un ejemplo de inferencia con PyTorch

Utilice los siguientes comandos para descargar un modelo densenet161 previamente entrenado y ejecutar la inferencia mediante: 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
```

El resultado debería tener un aspecto similar al siguiente:

```
{
  "tiger_cat": 0.4693363308906555,
  "tabby": 0.4633873701095581,
  "Egyptian_cat": 0.06456123292446136,
  "lynx": 0.0012828150065615773,
  "plastic_bag": 0.00023322898778133094
}
```

Utilice los siguientes comandos para anular el registro del modelo densenet161 y detener el servidor:

```
curl -X DELETE http://localhost:8081/models/densenet161/1.0
torchserve --stop
```

El resultado debería tener un aspecto similar al siguiente:

```
{
  "status": "Model \"densenet161\" unregistered"
}
TorchServe has stopped.
```

La DLAMI Habana

Las instancias con aceleradores de Habana están diseñadas para proporcionar alto rendimiento y rentabilidad para el entrenamiento de modelos de aprendizaje profundo. En concreto, los tipos de instancias DL1 utilizan aceleradores de Habana Gaudi de Habana Labs, una empresa de Intel. Las instancias con aceleradores Habana se configuran con el software Habana SynapseAI y se integran previamente con los marcos de aprendizaje automático más populares, como y. TensorFlow PyTorch

En los siguientes temas se muestra cómo comenzar a usar el hardware de Habana Gaudi con la DLAMI.

Contenido

Lanzamiento de una DLAMI con Habana

El último DLAMI está preparado para su uso con los aceleradores de Habana Gaudi. Realice los pasos siguientes para lanzar su DLAMI con Habana y asegurarse de que sus recursos de Python y específicos del marco estén activos. Para obtener recursos de configuración adicionales, consulte el repositorio Habana Gaudi Setup and Installation.

Contenido

Selección de una DLAMI de Habana

Lance una instancia de DL1 con la DLAMI de Habana que prefiera.

Para step-by-step obtener instrucciones sobre el lanzamiento de una DLAMI, consulte Lanzamiento y configuración de una DLAMI.

Para obtener una lista de las DLAMI de Habana más recientes, consulte las Release Notes for DLAMI.

Active el entorno de Python.

Conéctese a su instancia de DL1 y active el entorno de Python recomendado para su DLAMI de Habana. Para comprobar el entorno de Python recomendado, seleccione su DLAMI en las notas de la versión.

Importación del marco de machine learning

Las instancias con aceleradores Habana vienen preintegradas con los marcos de aprendizaje automático más populares, como y. TensorFlow PyTorch Importe el marco de machine learning que prefiera.

Importar TensorFlow

Para usarlo TensorFlow en tu DLAMI de Habana, navega hasta la carpeta del entorno Python que activaste e importaste. TensorFlow

```
/usr/bin/$PYTHON_VERSION
import tensorflow
tensorflow.__version__
```

Para comprobar la TensorFlow versión compatible con tu DLAMI de Habana, selecciona tu DLAMI en las notas de la versión.

Importar PyTorch

Para usarlo PyTorch en tu DLAMI de Habana, navega hasta la carpeta del entorno Python que activaste e importa la versión correspondiente. PyTorch

```
/usr/bin/$PYTHON_VERSION
import torch
torch.__version__
```

Para comprobar la PyTorch versión compatible con tu DLAMI de Habana, selecciona tu DLAMI en las notas de la versión.

Para obtener más información sobre cómo ejecutar y entrenar modelos de aprendizaje automático en TensorFlow y PyTorch usar tu DLAMI de Habana, consulta el repositorio de referencias de modelos de Habana. GitHub Para obtener recursos adicionales sobre cómo trabajar con su DLAMI de Habana, consulte la documentación de Habana Gaudi.

Inferencia

En esta sección, se proporcionan tutoriales sobre cómo ejecutar la inferencia utilizando los marcos de trabajo y las herramientas de la DLAMI.

Si desea ver tutoriales sobre el uso de Elastic Inference, consulte Trabajar con Amazon Elastic Inference

Inferencia con marcos de inferencia

- Uso de Apache MXNet (Incubating) para inferencia con un modelo ONNX

- Utilice Apache MXNet (en incubación) para la inferencia con un modelo 50 ResNet
- Cómo utilizar CNTK para las inferencias con un modelo ONNX

Herramientas de inferencia

- Model Server for Apache MXNet (MMS)
- TensorFlow Sirviendo

Uso de Apache MXNet (Incubating) para inferencia con un modelo ONNX

Cómo utilizar un modelo ONNX para la inferencia de imágenes con Apache MXNet (en fase de incubación)

1.
   - (Opción para Python 3) - Active el entorno Apache MXNet de Python 3 (en fase de incubación):

     ```
     $ source activate mxnet_p36
     ```

   - (Opción para Python 2) - Active el entorno Apache MXNet de Python 2 (en fase de incubación):

     ```
     $ source activate mxnet_p27
     ```

2. En los pasos restantes se da por hecho que está utilizando el entorno `mxnet_p36`.
3. Descargue una imagen de un perro esquimal.

   ```
   $ curl -O https://upload.wikimedia.org/wikipedia/commons/b/b5/Siberian_Husky_bi-
   eyed_Flickr.jpg
   ```

4. Descargue una lista de clases que funcionen con este modelo.

   ```
   $ curl -O https://gist.githubusercontent.com/yrevar/6135f1bd8dcf2e0cc683/raw/
   d133d61a09d7e5a3b36b8c111a8dd5c4b5d560ee/imagenet1000_clsid_to_human.pkl
   ```

5. Descargue el modelo VGG 16 entrenado previamente en formato ONNX.

   ```
   $ wget -O vgg16.onnx https://github.com/onnx/models/raw/master/vision/
   classification/vgg/model/vgg16-7.onnx
   ```

6. Utilice su editor de texto preferido para crear un script que tenga el siguiente contenido. Este script utilizará la imagen del perro esquimal, obtendrá un resultado de predicción del modelo entrenado previamente y, a continuación, lo buscará en el archivo de clases, que devuelve un resultado de clasificación de imágenes.

```
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. A continuación, ejecute el script y debería ver un resultado tal y como se indica a continuación:

```
248
Eskimo dog, husky
```

Utilice Apache MXNet (en incubación) para la inferencia con un modelo 50 ResNet

Cómo utilizar un modelo de Apache MXNet (en fase de incubación) entrenado previamente con la API de símbolo para la inferencia de imágenes con MXNet

1. • (Opción para Python 3) - Active el entorno Apache MXNet de Python 3 (en fase de incubación):

   ```
   $ source activate mxnet_p36
   ```

   • (Opción para Python 2) - Active el entorno Apache MXNet de Python 2 (en fase de incubación):

   ```
   $ source activate mxnet_p27
   ```

2. En los pasos restantes se da por hecho que está utilizando el entorno `mxnet_p36`.
3. Utilice su editor de texto preferido para crear un script que tenga el siguiente contenido. Este script descargará los archivos del modelo ResNet -50 (resnet-50-0000.params y resnet-50-symbol.json) y la lista de etiquetas (synset.txt), descargará una imagen de gato para obtener un resultado de predicción a partir del modelo previamente entrenado y, a continuación, lo buscará en el resultado en la lista de etiquetas y devolverá un resultado de predicción.

```
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. A continuación, ejecute el script y debería ver un resultado tal y como se indica a continuación:

```
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
```

Cómo utilizar CNTK para las inferencias con un modelo ONNX

> **Note**
>
> Ya no incluimos los entornos CNTK, Caffe, Caffe 2 y Theano Conda en la AWS Deep Learning AMI a partir de la versión v28. AWS Deep Learning AMI Las versiones anteriores que contienen estos entornos seguirán estando disponibles. Sin embargo, solo

proporcionaremos actualizaciones a estos entornos si la comunidad de código abierto publica correcciones de seguridad para estos marcos.

> **Note**
>
> El modelo VGG-16 utilizado en este tutorial consume una gran cantidad de memoria. Al seleccionar la AWS Deep Learning AMI instancia, es posible que necesite una instancia con más de 30 GB de RAM.

Cómo utilizar un modelo ONNX para las inferencias con CNTK

1.
   - (Opción para Python 3) - Active el entorno CNTK de Python 3:

     ```
     $ source activate cntk_p36
     ```

   - (Opción para Python 2) - Active el entorno CNTK de Python 2:

     ```
     $ source activate cntk_p27
     ```

2. En los pasos restantes se da por hecho que está utilizando el entorno `cntk_p36`.
3. Cree un nuevo archivo con el editor de texto y utilice el siguiente programa en un script para abrir el archivo en formato ONNX en 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!")
   ```

   Después de ejecutar este script, CNTK habrá cargado el modelo.
4. También puede probar a ejecutar inferencias con CNTK. En primer lugar, descargue una imagen de un perro esquimal.

   ```
   $ curl -O https://upload.wikimedia.org/wikipedia/commons/b/b5/Siberian_Husky_bi-eyed_Flickr.jpg
   ```

5. A continuación, descargar una lista de clases que funcionarán con este modelo.

```
$ curl -O https://gist.githubusercontent.com/yrevar/6135f1bd8dcf2e0cc683/raw/
d133d61a09d7e5a3b36b8c111a8dd5c4b5d560ee/imagenet1000_clsid_to_human.pkl
```

6. Edite el script creado anteriormente para que tenga el siguiente contenido. Esta nueva versión utilizará la imagen del perro esquimal, obtendrá un resultado de predicción y, a continuación, lo buscará en el archivo de clases para obtener un resultado de predicción.

```
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. A continuación, ejecute el script y debería ver un resultado tal y como se indica a continuación:

```
248
Eskimo dog, husky
```

Uso de marcos de trabajo con ONNX

La AMI de aprendizaje profundo con Conda ahora es compatible con modelos de intercambio de red neuronal abierta (ONNX) para algunos marcos de trabajo. Elija uno de los temas que se indican a continuación para obtener información sobre cómo utilizar ONNX en su AMI de aprendizaje profundo con Conda.

Si desea utilizar un modelo ONNX existente en una DLAMI, consulte Uso de Apache MXNet (Incubating) para inferencia con un modelo ONNX.

Acerca de ONNX

El intercambio de red neuronal abierta (ONNX) es un formato abierto que se usa para representar modelos de aprendizaje profundo. ONNX es compatible con Amazon Web Services, Microsoft, Facebook y muchos otros socios. Puede diseñar, entrenar e implementar modelos de aprendizaje profundo con cualquier marco de trabajo que elija. El beneficio de los modelos ONNX es que se pueden mover entre marcos de trabajo con facilidad.

La AMI de aprendizaje profundo con Conda; actualmente destaca algunas de las características de ONNX en la siguiente colección de tutoriales.

- Tutorial de MXNet a ONNX a CNTK
- Tutorial de Chainer a ONNX a CNTK
- Tutorial de Chainer a ONNX a MXNet
- PyTorch Tutorial de ONNX a CNTK
- PyTorch Tutorial de ONNX a MXNet

También puede consultar la documentación y los tutoriales del proyecto ONNX:

- Proyecto ONNX en GitHub
- Tutoriales de ONNX

Tutorial de MXNet a ONNX a CNTK

> **Note**
>
> Ya no incluimos los entornos CNTK, Caffe, Caffe 2 y Theano Conda en la AWS Deep Learning AMI a partir de la versión v28. Las versiones anteriores AWS Deep Learning AMI que contienen estos entornos seguirán estando disponibles. Sin embargo, solo proporcionaremos actualizaciones a estos entornos si la comunidad de código abierto publica correcciones de seguridad para estos marcos.

Información general de ONNX

El intercambio de red neuronal abierta (ONNX) es un formato abierto que se usa para representar modelos de aprendizaje profundo. ONNX es compatible con Amazon Web Services, Microsoft, Facebook y muchos otros socios. Puede diseñar, entrenar e implementar modelos de aprendizaje profundo con cualquier marco de trabajo que elija. El beneficio de los modelos ONNX es que se pueden mover entre marcos de trabajo con facilidad.

En este tutorial se muestra cómo utilizar la AMI de aprendizaje profundo con Conda con ONNX. Si sigue estos pasos, puede entrenar un modelo o cargar un modelo preentrenado desde un marco de trabajo, exportar este modelo a ONNX y, a continuación, importar el modelo en otro marco de trabajo.

Requisitos previos de ONNX

Para utilizar este tutorial de ONNX, debe disponer de acceso a una AMI de aprendizaje profundo con Conda, versión 12 o posterior. Para obtener más información sobre cómo empezar a usar una AMI de aprendizaje profundo con Conda, consulte AMI de aprendizaje profundo con Conda.

> **Important**
>
> Estos ejemplos utilizan funciones que pueden requerir hasta 8 GB de memoria (o más). Asegúrese de elegir un tipo de instancia con memoria suficiente.

Lance una sesión de terminal con su AMI de aprendizaje profundo para comenzar el siguiente tutorial.

Conversión de un modelo Apache MXNet (en fase de incubación) a ONNX y carga posterior del modelo en CNTK

Cómo exportar un modelo de Apache MXNet (en fase de incubación)

Puede instalar la última compilación de MXNet en uno o en ambos entornos de MXNet Conda en su AMI de aprendizaje profundo con Conda.

1.
   - (Opción para Python 3) - Active el entorno MXNet de Python 3:

     ```
     $ source activate mxnet_p36
     ```

   - (Opción para Python 2) - Active el entorno MXNet de Python 2:

```
$ source activate mxnet_p27
```

2. En los pasos restantes se supone que está utilizando el entorno `mxnet_p36`.
3. Descargue los archivos del modelo.

```
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. Para exportar los archivos del modelo del formato MXNet a ONNX, cree un nuevo archivo con el editor de texto y utilice el siguiente programa en un 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)
```

   Si bien pueden aparecer algunos mensajes de advertencia, puede pasarlos por alto de forma segura por ahora. Después de ejecutar este script, verá el archivo .onnx que acaba de crear en el mismo directorio.
5. Ahora que tiene un archivo ONNX, puede probar a ejecutar la inferencia con él mediante el siguiente ejemplo:

   - Cómo utilizar CNTK para las inferencias con un modelo ONNX

Tutoriales de ONNX

- Tutorial de MXNet a ONNX a CNTK
- Tutorial de Chainer a ONNX a CNTK
- Tutorial de Chainer a ONNX a MXNet

- PyTorch Tutorial de ONNX a MXNet
- PyTorch Tutorial de ONNX a CNTK

Tutorial de Chainer a ONNX a CNTK

> **Note**
>
> Ya no incluimos los entornos CNTK, Caffe, Caffe 2 y Theano Conda en la AWS Deep Learning AMI a partir de la versión v28. Las versiones anteriores AWS Deep Learning AMI que contienen estos entornos seguirán estando disponibles. Sin embargo, solo proporcionaremos actualizaciones a estos entornos si la comunidad de código abierto publica correcciones de seguridad para estos marcos.

Información general de ONNX

El intercambio de red neuronal abierta (ONNX) es un formato abierto que se usa para representar modelos de aprendizaje profundo. ONNX es compatible con Amazon Web Services, Microsoft, Facebook y muchos otros socios. Puede diseñar, entrenar e implementar modelos de aprendizaje profundo con cualquier marco de trabajo que elija. El beneficio de los modelos ONNX es que se pueden mover entre marcos de trabajo con facilidad.

En este tutorial se muestra cómo utilizar la AMI de aprendizaje profundo con Conda con ONNX. Si sigue estos pasos, puede entrenar un modelo o cargar un modelo preentrenado desde un marco de trabajo, exportar este modelo a ONNX y, a continuación, importar el modelo en otro marco de trabajo.

Requisitos previos de ONNX

Para utilizar este tutorial de ONNX, debe disponer de acceso a una AMI de aprendizaje profundo con Conda, versión 12 o posterior. Para obtener más información sobre cómo empezar a usar una AMI de aprendizaje profundo con Conda, consulte AMI de aprendizaje profundo con Conda.

> **Important**
>
> Estos ejemplos utilizan funciones que pueden requerir hasta 8 GB de memoria (o más). Asegúrese de elegir un tipo de instancia con memoria suficiente.

Lance una sesión de terminal con su AMI de aprendizaje profundo para comenzar el siguiente tutorial.

Convertir un modelo Chainer a ONNX y luego cargar el modelo en CNTK

En primer lugar, active el entorno Chainer:

```
$ source activate chainer_p36
```

Cree un nuevo archivo con el editor de texto y utilice el siguiente programa en un script para recuperar un modelo de Model Zoo de Chainer, luego expórtelo al 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)
```

Después de ejecutar este script, verá el archivo .onnx que acaba de crear en el mismo directorio.

Ahora que tiene un archivo ONNX, puede probar a ejecutar la inferencia con él mediante el siguiente ejemplo:

- Cómo utilizar CNTK para las inferencias con un modelo ONNX

Tutoriales de ONNX

- Tutorial de MXNet a ONNX a CNTK
- Tutorial de Chainer a ONNX a CNTK
- Tutorial de Chainer a ONNX a MXNet
- PyTorch Tutorial de ONNX a MXNet
- PyTorch Tutorial de ONNX a CNTK

Tutorial de Chainer a ONNX a MXNet

Información general de ONNX

El intercambio de red neuronal abierta (ONNX) es un formato abierto que se usa para representar modelos de aprendizaje profundo. ONNX es compatible con Amazon Web Services, Microsoft, Facebook y muchos otros socios. Puede diseñar, entrenar e implementar modelos de aprendizaje profundo con cualquier marco de trabajo que elija. El beneficio de los modelos ONNX es que se pueden mover entre marcos de trabajo con facilidad.

En este tutorial se muestra cómo utilizar la AMI de aprendizaje profundo con Conda con ONNX. Si sigue estos pasos, puede entrenar un modelo o cargar un modelo preentrenado desde un marco de trabajo, exportar este modelo a ONNX y, a continuación, importar el modelo en otro marco de trabajo.

Requisitos previos de ONNX

Para utilizar este tutorial de ONNX, debe disponer de acceso a una AMI de aprendizaje profundo con Conda, versión 12 o posterior. Para obtener más información sobre cómo empezar a usar una AMI de aprendizaje profundo con Conda, consulte AMI de aprendizaje profundo con Conda.

> **Important**
>
> Estos ejemplos utilizan funciones que pueden requerir hasta 8 GB de memoria (o más). Asegúrese de elegir un tipo de instancia con memoria suficiente.

Lance una sesión de terminal con su AMI de aprendizaje profundo para comenzar el siguiente tutorial.

Convertir un modelo Chainer a ONNX y luego cargar el modelo en MXNet

En primer lugar, active el entorno Chainer:

```
$ source activate chainer_p36
```

Cree un nuevo archivo con el editor de texto y utilice el siguiente programa en un script para recuperar un modelo de Model Zoo de Chainer, luego expórtelo al 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)
```

Después de ejecutar este script, verá el archivo .onnx que acaba de crear en el mismo directorio.

Ahora que tiene un archivo ONNX, puede probar a ejecutar la inferencia con él mediante el siguiente ejemplo:

- Uso de Apache MXNet (Incubating) para inferencia con un modelo ONNX

Tutoriales de ONNX

- Tutorial de MXNet a ONNX a CNTK
- Tutorial de Chainer a ONNX a CNTK
- Tutorial de Chainer a ONNX a MXNet
- PyTorch Tutorial de ONNX a MXNet
- PyTorch Tutorial de ONNX a CNTK

PyTorch Tutorial de ONNX a CNTK

> **Note**
>
> Ya no incluimos los entornos CNTK, Caffe, Caffe 2 y Theano Conda en la AWS Deep Learning AMI a partir de la versión v28. Las versiones anteriores AWS Deep Learning AMI que contienen estos entornos seguirán estando disponibles. Sin embargo, solo proporcionaremos actualizaciones a estos entornos si la comunidad de código abierto publica correcciones de seguridad para estos marcos.

Información general de ONNX

El intercambio de red neuronal abierta (ONNX) es un formato abierto que se usa para representar modelos de aprendizaje profundo. ONNX es compatible con Amazon Web Services, Microsoft, Facebook y muchos otros socios. Puede diseñar, entrenar e implementar modelos de aprendizaje profundo con cualquier marco de trabajo que elija. El beneficio de los modelos ONNX es que se pueden mover entre marcos de trabajo con facilidad.

En este tutorial se muestra cómo utilizar la AMI de aprendizaje profundo con Conda con ONNX. Si sigue estos pasos, puede entrenar un modelo o cargar un modelo preentrenado desde un marco de trabajo, exportar este modelo a ONNX y, a continuación, importar el modelo en otro marco de trabajo.

Requisitos previos de ONNX

Para utilizar este tutorial de ONNX, debe disponer de acceso a una AMI de aprendizaje profundo con Conda, versión 12 o posterior. Para obtener más información sobre cómo empezar a usar una AMI de aprendizaje profundo con Conda, consulte AMI de aprendizaje profundo con Conda.

> **Important**
>
> Estos ejemplos utilizan funciones que pueden requerir hasta 8 GB de memoria (o más). Asegúrese de elegir un tipo de instancia con memoria suficiente.

Lance una sesión de terminal con su AMI de aprendizaje profundo para comenzar el siguiente tutorial.

Convierta un PyTorch modelo en ONNX y, a continuación, cárguelo en CNTK

Primero, active el PyTorch entorno:

```
$ source activate pytorch_p36
```

Cree un nuevo archivo con su editor de texto y utilice el siguiente programa en un script para entrenar un modelo simulado y, a continuación PyTorch, expórtelo al 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)
```

Después de ejecutar este script, verá el archivo .onnx que acaba de crear en el mismo directorio. Ahora, cambie al entorno CNTK Conda para cargar el modelo con CNTK.

A continuación, active el entorno CNTK:

```
$ source deactivate
$ source activate cntk_p36
```

Cree un nuevo archivo con el editor de texto y utilice el siguiente programa en un script para abrir el archivo en formato ONNX en 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)
```

Después de ejecutar este script, CNTK habrá cargado el modelo.

También puede exportar a ONNX mediante CNTK añadiendo lo siguiente a su script anterior y, a continuación, ejecutándolo.

```
# Export the model to ONNX via the CNTK export API
z.save("cntk_model.onnx", format=C.ModelFormat.ONNX)
```

Tutoriales de ONNX

- Tutorial de MXNet a ONNX a CNTK
- Tutorial de Chainer a ONNX a CNTK
- Tutorial de Chainer a ONNX a MXNet

- PyTorch Tutorial de ONNX a MXNet
- PyTorch Tutorial de ONNX a CNTK

PyTorch Tutorial de ONNX a MXNet

Información general de ONNX

El intercambio de red neuronal abierta (ONNX) es un formato abierto que se usa para representar modelos de aprendizaje profundo. ONNX es compatible con Amazon Web Services, Microsoft, Facebook y muchos otros socios. Puede diseñar, entrenar e implementar modelos de aprendizaje profundo con cualquier marco de trabajo que elija. El beneficio de los modelos ONNX es que se pueden mover entre marcos de trabajo con facilidad.

En este tutorial se muestra cómo utilizar la AMI de aprendizaje profundo con Conda con ONNX. Si sigue estos pasos, puede entrenar un modelo o cargar un modelo preentrenado desde un marco de trabajo, exportar este modelo a ONNX y, a continuación, importar el modelo en otro marco de trabajo.

Requisitos previos de ONNX

Para utilizar este tutorial de ONNX, debe disponer de acceso a una AMI de aprendizaje profundo con Conda, versión 12 o posterior. Para obtener más información sobre cómo empezar a usar una AMI de aprendizaje profundo con Conda, consulte AMI de aprendizaje profundo con Conda.

> **Important**
>
> Estos ejemplos utilizan funciones que pueden requerir hasta 8 GB de memoria (o más). Asegúrese de elegir un tipo de instancia con memoria suficiente.

Lance una sesión de terminal con su AMI de aprendizaje profundo para comenzar el siguiente tutorial.

Convierta un PyTorch modelo en ONNX y, a continuación, cargue el modelo en MXNet

Primero, active el PyTorch entorno:

```
$ source activate pytorch_p36
```

Cree un nuevo archivo con su editor de texto y utilice el siguiente programa en un script para entrenar un modelo simulado y, a continuación PyTorch, expórtelo al 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!")
```

Después de ejecutar este script, verá el archivo .onnx que acaba de crear en el mismo directorio. Ahora, cambie al entorno MXNet Conda para cargar el modelo con MXNet.

A continuación, active el entorno MXNet:

```
$ source deactivate
$ source activate mxnet_p36
```

Cree un nuevo archivo con el editor de texto y utilice el siguiente programa en un script para abrir el archivo en formato ONNX en 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())
```

Después de ejecutar este script, MXNet habrá cargado el modelo e imprimirá información básica del modelo.

Tutoriales de ONNX

- Tutorial de MXNet a ONNX a CNTK
- Tutorial de Chainer a ONNX a CNTK
- Tutorial de Chainer a ONNX a MXNet
- PyTorch Tutorial de ONNX a MXNet
- PyTorch Tutorial de ONNX a CNTK

Distribución de modelos

A continuación, se indican las opciones de distribución de modelos instaladas en la AMI de aprendizaje profundo con Conda. Haga clic en una de las opciones para obtener información acerca de cómo utilizarla.

Temas

- Model Server for Apache MXNet (MMS)
- TensorFlow Sirviendo
- TorchServe

Model Server for Apache MXNet (MMS)

Model Server for Apache MXNet (MMS) es una herramienta flexible para servir modelos de aprendizaje profundo que se han exportado de Apache MXNet (incubating) o se han exportado al formato de modelo de intercambio de red neuronal abierta (ONNX). MMS viene preinstalado con la DLAMI con Conda. En este tutorial de MMS, se muestra cómo servir un modelo de clasificación de imágenes.

Temas

- Servir un modelo de clasificación de imágenes en MMS
- Otros ejemplos
- Más información

Servir un modelo de clasificación de imágenes en MMS

En este tutorial se muestra cómo servir un modelo de clasificación de imágenes con MMS. El modelo se proporciona a través del MMS Model Zoo y se descarga automáticamente al iniciar MMS. Una vez que el servidor se está ejecutando, escucha las solicitudes de predicción. Al cargar una imagen, en este caso, una imagen de un gatito, el servidor devuelve una predicción de las 5 principales clases coincidente de las 1000 clases con las que se entrenó el modelo. Puede encontrar más información sobre los modelos, cómo se han entrenado y cómo probarlos en el MMS Model Zoo.

Para servir un modelo de clasificación de imágenes de ejemplo en MMS

1. Conéctese a una instancia de Amazon Elastic Compute Cloud (Amazon EC2) de la AMI de aprendizaje profundo con Conda.
2. Active un entorno de MXNet:
   - Para MXNet y Keras 2 en Python 3 con CUDA 9.0 y MKL-DNN, ejecute este comando:

     ```
     $ source activate mxnet_p36
     ```

   - Para MXNet y Keras 2 en Python 2 con CUDA 9.0 y MKL-DNN, ejecute este comando:

     ```
     $ source activate mxnet_p27
     ```

3. Ejecute MMS con el siguiente comando. Si añade `> /dev/null`, se silenciará la salida de registro mientras ejecuta otras pruebas.

```
$ mxnet-model-server --start > /dev/null
```

MMS ahora se está ejecutando en su host y está escuchando solicitudes de inferencia.

4. A continuación, utilice un comando curl para administrar los puntos de enlace de administración de MMS e indicarle cuál es el modelo que desea que sirva.

```
$ 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 necesita saber cuál es el número de procesos de trabajo que desea utilizar. Para esta prueba, puede utilizar 3.

```
$ curl -v -X PUT "http://localhost:8081/models/squeezenet_v1.1?min_worker=3"
```

6. Descargue una imagen de un gatito y envíela al punto de enlace de predicción de 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
```

El punto de enlace de predicción devuelve una predicción en JSON similar a las siguientes cinco predicciones principales, donde la imagen tiene una probabilidad del 94% de contener un gato egipcio, seguida de una probabilidad del 5,5% de que sea un lince o gato montés:

```
{
 "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. Pruebe algunos imágenes más o, si ya ha terminado de realizar pruebas, detenga el servidor:

```
$ mxnet-model-server --stop
```

Este tutorial se centra en el servicio de modelos básicos. MMS también admite el uso de Elastic Inference con la distribución de modelos. Para obtener más información, consulte Model Serving with Amazon Elastic Inference.

Cuando esté preparado para obtener más información sobre otras funciones del MMS, consulte la documentación del MMS en. GitHub

Otros ejemplos

MMS dispone de una serie de ejemplos que puede ejecutar en la DLAMI. Puede verlas en el repositorio del proyecto MMS.

Más información

Para obtener más documentación sobre el MMS, incluido cómo configurar el MMS con Docker, o para aprovechar las funciones más recientes del MMS, inicia la página del proyecto MMS en. GitHub

TensorFlow Sirviendo

TensorFlow Serving es un sistema de servicio flexible y de alto rendimiento para modelos de aprendizaje automático.

`tensorflow-serving-api` viene ya instalado con la AMI de aprendizaje profundo con Conda. Encontrará un script de ejemplo para entrenar, exportar y distribuir un modelo MNIST en `~/examples/tensorflow-serving/`.

Para ejecutar cualquiera de estos ejemplos, primero conéctese a su AMI de aprendizaje profundo con Conda y active el TensorFlow entorno.

```
$ source activate tensorflow_p37
```

Ahora, desplácese a los directorios que contienen la carpeta de scripts de ejemplo.

```
$ cd ~/examples/tensorflow-serving/
```

Cómo servir un modelo de incepción

A continuación, se muestra un ejemplo que puede probar para servir distintos modelos como Inception. Por regla general, necesita un modelo servible y scripts de cliente descargados previamente en la DLAMI.

Cómo servir y probar la inferencia con un modelo de incepción

1. Descargue el modelo.

   ```
   $ curl -O https://s3-us-west-2.amazonaws.com/tf-test-models/INCEPTION.zip
   ```

2. Descomprima el modelo.

   ```
   $ unzip INCEPTION.zip
   ```

3. Descargue una imagen de un perro esquimal.

   ```
   $ curl -O https://upload.wikimedia.org/wikipedia/commons/b/b5/Siberian_Husky_bi-
   eyed_Flickr.jpg
   ```

4. Lance el servidor. Tenga en cuenta que, para Amazon Linux, debe cambiar el directorio que se utiliza para `model_base_path` de `/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 el servidor ejecutándose en primer plano, tendrá que lanzar otra sesión de terminal para continuar. Abre una nueva terminal y actívala TensorFlow con`source activate tensorflow_p37`. A continuación, utilice su editor de texto preferido para crear un script que tenga el siguiente contenido. Denomínelo `inception_client.py`. Este script tomará un

nombre de archivo de imagen como parámetro y obtendrá un resultado de predicción a partir del modelo entrenado previamente.

```
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. Ahora ejecute el script pasando la ubicación y el puerto del servidor y el nombre de archivo de la foto del perro esquimal como parámetros.

```
$ python3 inception_client.py --server=localhost:9000 --image Siberian_Husky_bi-
eyed_Flickr.jpg
```

Cómo entrenar y servir un modelo de MNIST

En este tutorial vamos a exportar un modelo y después lo distribuiremos con la aplicación `tensorflow_model_server`. Por último, puede probar el servidor del modelo con un script del cliente de ejemplo.

Ejecute el script que entrenará y exportará un modelo MNIST. Como único argumento del script debe proporcionar una ubicación de carpeta para guardar el modelo. Por ahora, lo pondremos en `mnist_model`. El script creará la carpeta por usted.

```
$ python mnist_saved_model.py /tmp/mnist_model
```

Sea paciente, ya que el script puede tardar un rato en proporcionar resultados. Cuando se haya completado el entrenamiento y se haya exportado el modelo, debería ver lo siguiente:

```
Done training!
Exporting trained model to mnist_model/1
Done exporting!
```

El siguiente paso consiste en ejecutar `tensorflow_model_server` para distribuir el modelo exportado.

```
$ tensorflow_model_server --port=9000 --model_name=mnist --model_base_path=/tmp/
mnist_model
```

Se proporciona un script del cliente para probar el servidor.

Para probarlo, tendrá que abrir una nueva ventana de la terminal.

```
$ python mnist_client.py --num_tests=1000 --server=localhost:9000
```

Más características y ejemplos

Si está interesado en obtener más información sobre TensorFlow Serving, visite el TensorFlow sitio web.

También puede usar TensorFlow Serving with Amazon Elastic Inference. Consulta la guía sobre cómo usar Elastic Inference with TensorFlow Serving para obtener más información.

TorchServe

TorchServe es una herramienta flexible para ofrecer modelos de aprendizaje profundo que se han exportado desde PyTorch. TorchServe viene preinstalada con la AMI de aprendizaje profundo con Conda a partir de la versión 34.

Para obtener más información sobre su uso TorchServe, consulte Model Server para ver la documentación. PyTorch

Temas

Sirva un modelo de clasificación de imágenes en TorchServe

En este tutorial se muestra cómo crear un modelo de clasificación de imágenes con TorchServe. Utiliza un modelo DenseNet -161 proporcionado por PyTorch. Una vez que el servidor está en funcionamiento, escucha las solicitudes de predicción. Al cargar una imagen, en este caso una imagen de un gatito, el servidor devuelve una predicción de las 5 principales clases coincidentes de entre las clases con las que se haya entrenado el modelo.

A modo de ejemplo, un modelo de clasificación de imágenes en TorchServe

1. Conéctese a una instancia de Amazon Elastic Compute Cloud (Amazon EC2) con el AMI de aprendizaje profundo con Conda, v34 o posterior.
2. Active el entorno de `pytorch_latest_p36`.

   ```
   source activate pytorch_latest_p36
   ```

3. Clone el TorchServe repositorio y, a continuación, cree un directorio para almacenar sus modelos.

   ```
   git clone https://github.com/pytorch/serve.git
   ```

```
mkdir model_store
```

4. Archive el modelo utilizando el archivador de modelos. El `extra-files` parámetro usa un archivo del `TorchServe` repositorio, así que actualiza la ruta si es necesario. Para obtener más información sobre el archivador de modelos, consulte el archivador de modelos Torch para. 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. Ejecute TorchServe para iniciar un punto final. Al agregar `> /dev/null` se silencia la salida del registro.

```
torchserve --start --ncs --model-store model_store --models densenet161.mar > /dev/
null
```

6. Descarga una imagen de un gatito y envíala al punto final de TorchServe predicción:

```
curl -O https://s3.amazonaws.com/model-server/inputs/kitten.jpg
curl http://127.0.0.1:8080/predictions/densenet161 -T kitten.jpg
```

   El punto de conexión de predicción devuelve una predicción en JSON similar a las siguientes cinco predicciones principales, donde la imagen tiene una probabilidad del 47 % de contener un gato egipcio, seguida de una probabilidad del 46 % de que sea un lince o un gato montés:

```
{
 "tiger_cat": 0.46933576464653015,
 "tabby": 0.463387668132782,
 "Egyptian_cat": 0.0645613968372345,
 "lynx": 0.0012828196631744504,
 "plastic_bag": 0.00023523058849200606
}
```

7. Cuando termine la prueba, detenga el servidor.

```
torchserve --stop
```

Otros ejemplos

TorchServe tiene varios ejemplos que puede ejecutar en su instancia de DLAMI. Puede verlos en la página de ejemplos del repositorio de TorchServe proyectos.

Más información

Para obtener más TorchServe documentación, incluida la forma de configurar TorchServe con Docker y las TorchServe funciones más recientes, consulta la página del TorchServe proyecto en GitHub.

Actualización de la DLAMI

Aquí encontrará información sobre la actualización de la DLAMI y consejos sobre la actualización de software en ella.

Temas

Actualización a una nueva versión de la DLAMI

Las imágenes del sistema de la DLAMI se actualizan de forma periódica para aprovechar las nuevas versiones del marco de aprendizaje profundo, CUDA y otras actualizaciones de software, así como para el ajuste del desempeño. Si lleva un tiempo utilizando una DLAMI y desea aprovechar una actualización, tendrá que volver a lanzar una nueva instancia. Además, tendría que transferir manualmente cualquier conjunto de datos, puntos de comprobación u otros datos valiosos. En lugar de ello, puede utilizar Amazon EBS para retener los datos y asociarlos a una nueva DLAMI. De esta forma, puede actualizar a menudo, además de reducir el tiempo que se tarda en realizar la transición de los datos.

> **Note**
>
> Al adjuntar y desplazar volúmenes de Amazon EBS entre las DLAMI, debe tener tanto las DLAMI como el nuevo volumen en la misma zona de disponibilidad.

1. Utilice la consola de Amazon EC2 para crear un nuevo volumen de Amazon EBS. Para obtener instrucciones detalladas, consulte Creación de un volumen de Amazon EBS.
2. Adjunte el volumen de Amazon EB; recién creado a la DLAMI existente. Para obtener instrucciones detalladas, consulte Attaching an Amazon EBS Volume.
3. Transfiera sus datos, por ejemplo, conjuntos de datos, puntos de comprobación y archivos de configuración.
4. Lanzamiento de una DLAMI. Para obtener indicaciones detalladas, consulte Lanzamiento y configuración de una DLAMI.

5. Separe el volumen de Amazon EBS de la DLAMI antigua. Para obtener instrucciones detalladas, consulte Detaching an Amazon EBS Volume.
6. Adjunte el volumen de Amazon EBS a la nueva DLAMI. Siga las instrucciones desde el Paso 2 para adjuntar el volumen.
7. Después de verificar que los datos están disponibles en su nueva DLAMI, detenga y termine la DLAMI antigua. Para obtener instrucciones de limpieza detalladas, consulte Eliminación.

Sugerencias para actualizaciones de software

De vez en cuando, es posible que desee actualizar manualmente el software en la DLAMI. En general, se recomienda que utilice `pip` para actualizar paquetes de Python. También debería utilizar `pip` para actualizar paquetes dentro de un entorno de Conda en la AMI de aprendizaje profundo con Conda. Consulte el sitio web del marco de trabajo o del software para obtener las instrucciones de actualización y de instalación.

> **Note**
>
> No podemos garantizar que la actualización de un paquete se realice correctamente. Si se intenta actualizar un paquete en un entorno con dependencias incompatibles, se puede producir un error. En tal caso, debe ponerse en contacto con el responsable de la biblioteca para ver si es posible actualizar las dependencias del paquete. Como alternativa, puede intentar modificar el entorno de tal manera que permita la actualización. Sin embargo, es probable que esta modificación implique eliminar o actualizar los paquetes existentes, lo que significa que ya no podemos garantizar la estabilidad de este entorno.

Si le interesa ejecutar la última ramificación principal de un paquete determinado, active el entorno adecuado y, a continuación, añada `--pre` al final del comando `pip install --upgrade`. Por ejemplo:

```
source activate mxnet_p36
pip install --upgrade mxnet --pre
```

La AWS Deep Learning AMI viene con muchos entornos de Conda y muchos paquetes preinstalados. Debido a la cantidad de paquetes preinstalados, es difícil encontrar un conjunto de paquetes que garanticen su compatibilidad. Es posible que aparezca una advertencia que diga: “The

environment is inconsistent, please check the package plan carefully”. La DLAMI garantiza que todos los entornos proporcionados por ella sean correctos, pero no puede garantizar que los paquetes instalados por el usuario funcionen correctamente.

Seguridad en AWS Deep Learning AMI

La seguridad en la nube AWS es la máxima prioridad. Como AWS cliente, usted se beneficia de una arquitectura de centro de datos y red diseñada para cumplir con los requisitos de las organizaciones más sensibles a la seguridad.

La seguridad es una responsabilidad compartida entre usted AWS y usted. El modelo de responsabilidad compartida la describe como seguridad de la nube y seguridad en la nube:

- Seguridad de la nube: AWS es responsable de proteger la infraestructura que ejecuta AWS los servicios en la AWS nube. AWS también le proporciona servicios que puede utilizar de forma segura. Los auditores externos prueban y verifican periódicamente la eficacia de nuestra seguridad como parte de los AWS programas de de . Para obtener más información sobre los programas de cumplimiento que se aplican a la DLAMI, AWS consulte Servicios dentro del alcance por programa de cumplimiento Servicios incluidos en el alcance por AWS de cumplimiento.
- Seguridad en la nube: su responsabilidad viene determinada por el AWS servicio que utilice. Usted también es responsable de otros factores, incluida la confidencialidad de los datos, los requisitos de la empresa y la legislación y los reglamentos aplicables.

Esta documentación le ayuda a comprender cómo aplicar el modelo de responsabilidad compartida cuando se utiliza DLAMI. En los siguientes temas, se le mostrará cómo configurar para satisfacer sus objetivos de seguridad y conformidad. También aprenderá a utilizar otros AWS servicios que le ayudan a supervisar y proteger sus recursos de DLAMI.

Para obtener más información, consulte Seguridad en Amazon EC2.

Temas

Protección de datos en AWS Deep Learning AMI

El modelo de se aplica a protección de datos en la AMI de AWS Deep Learning. Como se describe en este modelo, AWS es responsable de proteger la infraestructura global en la que se ejecutan todos los Nube de AWS. Usted es responsable de mantener el control sobre el contenido alojado en esta infraestructura. Usted también es responsable de las tareas de administración y configuración de seguridad para los Servicios de AWS que utiliza. Para obtener más información sobre la privacidad de los datos, consulte las Preguntas frecuentes sobre la privacidad de datos. Para obtener información sobre la protección de datos en Europa, consulte la publicación de blog sobre el Modelo de responsabilidad compartida de AWS y GDPR en el Blog de seguridad de AWS.

Con fines de protección de datos, le recomendamos que proteja Cuenta de AWS las credenciales y configure los usuarios individuales con AWS IAM Identity Center o AWS Identity and Access Management (IAM). De esta manera, solo se otorgan a cada usuario los permisos necesarios para cumplir sus obligaciones laborales. También recomendamos proteger sus datos de la siguiente manera:

- Utilice autenticación multifactor (MFA) en cada cuenta.
- Utilice SSL/TLS para comunicarse con los recursos. AWS Se recomienda el uso de TLS 1.2 y recomendamos TLS 1.3.
- Configure la API y el registro de actividad de los usuarios con. AWS CloudTrail
- Utilice soluciones de AWS cifrado, junto con todos los controles de seguridad predeterminados Servicios de AWS.
- Utilice servicios de seguridad administrados avanzados, como Amazon Macie, que lo ayuden a detectar y proteger los datos confidenciales almacenados en Amazon S3.
- Si necesita módulos criptográficos validados por FIPS 140-2 para acceder a AWS través de una interfaz de línea de comandos o una API, utilice un punto final FIPS. Para obtener más información sobre los puntos de conexión de FIPS disponibles, consulte Estándar de procesamiento de la información federal (FIPS) 140-2.

Se recomienda encarecidamente no introducir nunca información confidencial o sensible, como, por ejemplo, direcciones de correo electrónico de clientes, en etiquetas o campos de formato libre, tales como el campo Nombre. Esto incluye cuando trabajas con DLAMI u Servicios de AWS otro tipo mediante la consola, la API AWS CLI o los SDK. AWS Cualquier dato que ingrese en etiquetas o campos de formato libre utilizados para nombres se puede emplear para los registros de facturación

o diagnóstico. Si proporciona una URL a un servidor externo, recomendamos encarecidamente que no incluya información de credenciales en la URL a fin de validar la solicitud para ese servidor.

Identity and Access Management en AWS Deep Learning AMI

AWS Identity and Access Management (IAM) es una herramienta Servicio de AWS que ayuda al administrador a controlar de forma segura el acceso a AWS los recursos. Los administradores de IAM controlan quién puede estar autenticado (ha iniciado sesión) y autorizado (tiene permisos) para utilizar recursos. La IAM es una Servicio de AWS herramienta que puede utilizar sin coste adicional.

Para obtener más información sobre Identity and Access Management, consulte Identity and Access Management para Amazon EC2.

Temas

- Autenticación con identidades
- Administración de acceso mediante políticas
- IAM con Amazon EMR

Autenticación con identidades

La autenticación es la forma en que inicias sesión AWS con tus credenciales de identidad. Debe estar autenticado (con quien haya iniciado sesión AWS) como usuario de IAM o asumiendo una función de IAM. Usuario raíz de la cuenta de AWS

Puede iniciar sesión AWS como una identidad federada mediante las credenciales proporcionadas a través de una fuente de identidad. AWS IAM Identity Center Los usuarios (IAM Identity Center), la autenticación de inicio de sesión único de su empresa y sus credenciales de Google o Facebook son ejemplos de identidades federadas. Al iniciar sesión como una identidad federada, su administrador habrá configurado previamente la federación de identidades mediante roles de IAM. Cuando accedes AWS mediante la federación, estás asumiendo un rol de forma indirecta.

Según el tipo de usuario que sea, puede iniciar sesión en el portal AWS Management Console o en el de AWS acceso. Para obtener más información sobre cómo iniciar sesión AWS, consulte Cómo iniciar sesión Cuenta de AWS en su Guía del AWS Sign-In usuario.

Si accede AWS mediante programación, AWS proporciona un kit de desarrollo de software (SDK) y una interfaz de línea de comandos (CLI) para firmar criptográficamente sus solicitudes con sus credenciales. Si no utilizas AWS herramientas, debes firmar las solicitudes tú mismo. Para obtener

más información sobre cómo usar el método recomendado para firmar las solicitudes usted mismo, consulte Firmar las solicitudes de la AWS API en la Guía del usuario de IAM.

Independientemente del método de autenticación que use, es posible que deba proporcionar información de seguridad adicional. Por ejemplo, le AWS recomienda que utilice la autenticación multifactor (MFA) para aumentar la seguridad de su cuenta. Para obtener más información, consulte Autenticación multifactor en la Guía del usuario de AWS Single Sign-On y Uso de la autenticación multifactor (MFA) en AWS en la Guía del usuario de IAM.

Cuenta de AWS usuario root

Al crear una Cuenta de AWS, comienza con una identidad de inicio de sesión que tiene acceso completo a todos Servicios de AWS los recursos de la cuenta. Esta identidad se denomina usuario Cuenta de AWS raíz y se accede a ella iniciando sesión con la dirección de correo electrónico y la contraseña que utilizaste para crear la cuenta. Recomendamos encarecidamente que no utilice el usuario raíz para sus tareas diarias. Proteja las credenciales del usuario raíz y utilícelas solo para las tareas que solo el usuario raíz pueda realizar. Para ver la lista completa de las tareas que requieren que inicie sesión como usuario raíz, consulte Tareas que requieren credenciales de usuario raíz en la Guía del usuario de IAM.

Usuarios y grupos de IAM

Un usuario de IAM es una identidad propia Cuenta de AWS que tiene permisos específicos para una sola persona o aplicación. Siempre que sea posible, recomendamos emplear credenciales temporales, en lugar de crear usuarios de IAM que tengan credenciales de larga duración como contraseñas y claves de acceso. No obstante, si tiene casos de uso específicos que requieran credenciales de larga duración con usuarios de IAM, recomendamos rotar las claves de acceso. Para más información, consulte Rotar las claves de acceso periódicamente para casos de uso que requieran credenciales de larga duración en la Guía del usuario de IAM.

Un grupo de IAM es una identidad que especifica un conjunto de usuarios de IAM. No puede iniciar sesión como grupo. Puede usar los grupos para especificar permisos para varios usuarios a la vez. Los grupos facilitan la administración de los permisos de grandes conjuntos de usuarios. Por ejemplo, podría tener un grupo cuyo nombre fuese IAMAdmins y conceder permisos a dicho grupo para administrar los recursos de IAM.

Los usuarios son diferentes de los roles. Un usuario se asocia exclusivamente a una persona o aplicación, pero la intención es que cualquier usuario pueda asumir un rol que necesite. Los usuarios tienen credenciales permanentes a largo plazo y los roles proporcionan credenciales temporales.

Para más información, consulte Cuándo crear un usuario de IAM (en lugar de un rol) en la Guía del usuario de IAM.

Roles de IAM

Un rol de IAM es una identidad dentro de usted Cuenta de AWS que tiene permisos específicos. Es similar a un usuario de IAM, pero no está asociado a una determinada persona. Puede asumir temporalmente una función de IAM en el AWS Management Console cambiando de función. Puede asumir un rol llamando a una operación de AWS API AWS CLI o utilizando una URL personalizada. Para más información sobre los métodos para el uso de roles, consulte Uso de roles de IAM en la Guía del usuario de IAM.

Los roles de IAM con credenciales temporales son útiles en las siguientes situaciones:

- Acceso de usuario federado: para asignar permisos a una identidad federada, puede crear un rol y definir sus permisos. Cuando se autentica una identidad federada, se asocia la identidad al rol y se le conceden los permisos define el rol. Para obtener información acerca de roles para federación, consulte Creación de un rol para un proveedor de identidades de terceros en la Guía del usuario de IAM. Si utiliza el IAM Identity Center, debe configurar un conjunto de permisos. El IAM Identity Center correlaciona el conjunto de permisos con un rol en IAM para controlar a qué pueden acceder las identidades después de autenticarse. Para obtener información acerca de los conjuntos de permisos, consulte Conjuntos de permisos en la Guía del usuario de AWS Single Sign-On.
- Permisos de usuario de IAM temporales: un usuario de IAM puede asumir un rol de IAM para recibir temporalmente permisos distintos que le permitan realizar una tarea concreta.
- Acceso entre cuentas: puede utilizar un rol de IAM para permitir que alguien (una entidad principal de confianza) de otra cuenta acceda a los recursos de la cuenta. Los roles son la forma principal de conceder acceso entre cuentas. Sin embargo, con algunas Servicios de AWS, puedes adjuntar una política directamente a un recurso (en lugar de usar un rol como proxy). Para obtener información acerca de la diferencia entre los roles y las políticas basadas en recursos para el acceso entre cuentas, consulte Cómo los roles de IAM difieren de las políticas basadas en recursos en la Guía del usuario de IAM.
- Acceso entre servicios: algunos Servicios de AWS utilizan funciones en otros Servicios de AWS. Por ejemplo, cuando realiza una llamada en un servicio, es común que ese servicio ejecute aplicaciones en Amazon EC2 o almacene objetos en Amazon S3. Es posible que un servicio haga esto usando los permisos de la entidad principal, usando un rol de servicio o usando un rol vinculado al servicio.

- Sesiones de acceso directo (FAS): cuando utilizas un usuario o un rol de IAM para realizar acciones en ellas AWS, se te considera director. Cuando utiliza algunos servicios, es posible que realice una acción que desencadene otra acción en un servicio diferente. El FAS utiliza los permisos del principal que llama Servicio de AWS y los solicita Servicio de AWS para realizar solicitudes a los servicios descendentes. Las solicitudes de FAS solo se realizan cuando un servicio recibe una solicitud que requiere interacciones con otros Servicios de AWS recursos para completarse. En este caso, debe tener permisos para realizar ambas acciones. Para obtener información sobre las políticas a la hora de realizar solicitudes de FAS, consulte Reenviar sesiones de acceso.
- Rol de servicio: un rol de servicio es un rol de IAM que adopta un servicio para realizar acciones en su nombre. Un administrador de IAM puede crear, modificar y eliminar un rol de servicio desde IAM. Para obtener más información, consulte Creación de un rol para delegar permisos a un Servicio de AWS en la Guía del usuario de IAM.
- Función vinculada al servicio: una función vinculada a un servicio es un tipo de función de servicio que está vinculada a un. Servicio de AWS El servicio puede asumir el rol para realizar una acción en su nombre. Los roles vinculados al servicio aparecen en usted Cuenta de AWS y son propiedad del servicio. Un administrador de IAM puede ver, pero no editar, los permisos de los roles vinculados a servicios.

- Aplicaciones que se ejecutan en Amazon EC2: puede usar un rol de IAM para administrar las credenciales temporales de las aplicaciones que se ejecutan en una instancia EC2 y realizan AWS CLI solicitudes a la API. AWS Es preferible hacerlo de este modo a almacenar claves de acceso en la instancia EC2. Para asignar un AWS rol a una instancia EC2 y ponerlo a disposición de todas sus aplicaciones, debe crear un perfil de instancia adjunto a la instancia. Un perfil de instancia contiene el rol y permite a los programas que se ejecutan en la instancia EC2 obtener credenciales temporales. Para más información, consulte Uso de un rol de IAM para conceder permisos a aplicaciones que se ejecutan en instancias Amazon EC2 en la Guía del usuario de IAM.

Para obtener información sobre el uso de los roles de IAM, consulte Cuándo crear un rol de IAM (en lugar de un usuario) en la Guía del usuario de IAM.

Administración de acceso mediante políticas

El acceso se controla AWS creando políticas y adjuntándolas a AWS identidades o recursos. Una política es un objeto AWS que, cuando se asocia a una identidad o un recurso, define sus permisos. AWS evalúa estas políticas cuando un director (usuario, usuario raíz o sesión de rol) realiza una solicitud. Los permisos en las políticas determinan si la solicitud se permite o se deniega. La mayoría

de las políticas se almacenan AWS como documentos JSON. Para obtener más información sobre la estructura y el contenido de los documentos de política JSON, consulte Información general de políticas JSON en la Guía del usuario de IAM.

Los administradores pueden usar las políticas de AWS JSON para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

De forma predeterminada, los usuarios y los roles no tienen permisos. Para conceder permiso a los usuarios para realizar acciones en los recursos que necesiten, un administrador de IAM puede crear políticas de IAM. A continuación, el administrador puede añadir las políticas de IAM a roles y los usuarios pueden asumirlos.

Las políticas de IAM definen permisos para una acción independientemente del método que se utilice para realizar la operación. Por ejemplo, suponga que dispone de una política que permite la acción `iam:GetRole`. Un usuario con esa política puede obtener información sobre el rol de la API AWS Management Console AWS CLI, la o la AWS API.

Políticas basadas en identidad

Las políticas basadas en identidad son documentos de políticas de permisos JSON que puede asociar a una identidad, como un usuario de IAM, un grupo de usuarios o un rol. Estas políticas controlan qué acciones pueden realizar los usuarios y los roles, en qué recursos y en qué condiciones. Para obtener más información sobre cómo crear una política basada en identidad, consulte Creación de políticas de IAM en la Guía del usuario de IAM.

Las políticas basadas en identidades pueden clasificarse además como políticas insertadas o políticas administradas. Las políticas insertadas se integran directamente en un único usuario, grupo o rol. Las políticas administradas son políticas independientes que puede adjuntar a varios usuarios, grupos y roles de su Cuenta de AWS empresa. Las políticas administradas incluyen políticas AWS administradas y políticas administradas por el cliente. Para más información sobre cómo elegir una política administrada o una política insertada, consulte Elegir entre políticas administradas y políticas insertadas en la Guía del usuario de IAM.

Políticas basadas en recursos

Las políticas basadas en recursos son documentos de política JSON que se asocian a un recurso. Ejemplos de políticas basadas en recursos son las políticas de confianza de roles de IAM y las políticas de bucket de Amazon S3. En los servicios que admiten políticas basadas en recursos, los administradores de servicios pueden utilizarlos para controlar el acceso a un recurso específico.

Para el recurso al que se asocia la política, la política define qué acciones puede realizar una entidad principal especificada en ese recurso y en qué condiciones. Debe especificar una entidad principal en una política en función de recursos. Los principales pueden incluir cuentas, usuarios, roles, usuarios federados o. Servicios de AWS

Las políticas basadas en recursos son políticas insertadas que se encuentran en ese servicio. No puedes usar políticas AWS gestionadas de IAM en una política basada en recursos.

Listas de control de acceso (ACL)

Las listas de control de acceso (ACL) controlan qué entidades principales (miembros de cuentas, usuarios o roles) tienen permisos para acceder a un recurso. Las ACL son similares a las políticas basadas en recursos, aunque no utilizan el formato de documento de políticas JSON.

Amazon S3 y Amazon VPC son ejemplos de servicios que admiten las ACL. AWS WAF Para obtener más información sobre las ACL, consulte Información general de Lista de control de acceso (ACL) en la Guía para desarrolladores de Amazon Simple Storage Service.

Otros tipos de políticas

AWS admite tipos de políticas adicionales y menos comunes. Estos tipos de políticas pueden establecer el máximo de permisos que los tipos de políticas más frecuentes le conceden.

- Límites de permisos: un límite de permisos es una característica avanzada que le permite establecer los permisos máximos que una política basada en identidad puede conceder a una entidad de IAM (usuario o rol de IAM). Puede establecer un límite de permisos para una entidad. Los permisos resultantes son la intersección de las políticas basadas en la identidad de la entidad y los límites de permisos. Las políticas basadas en recursos que especifique el usuario o rol en el campo `Principal` no estarán restringidas por el límite de permisos. Una denegación explícita en cualquiera de estas políticas anulará el permiso. Para obtener más información sobre los límites de los permisos, consulte Límites de permisos para las entidades de IAM en la Guía del usuario de IAM.
- Políticas de control de servicios (SCP): las SCP son políticas de JSON que especifican los permisos máximos para una organización o unidad organizativa (OU). AWS Organizations AWS Organizations es un servicio para agrupar y gestionar de forma centralizada varios de los Cuentas de AWS que son propiedad de su empresa. Si habilita todas las características en una organización, entonces podrá aplicar políticas de control de servicio (SCP) a una o a todas sus cuentas. El SCP limita los permisos de las entidades en las cuentas de los miembros, incluidas las de cada una. Usuario raíz de la cuenta de AWS Para obtener más información acerca de

Organizations y las SCP, consulte Funcionamiento de las SCP en la Guía del usuario de AWS Organizations.

- Políticas de sesión: las políticas de sesión son políticas avanzadas que se pasan como parámetro cuando se crea una sesión temporal mediante programación para un rol o un usuario federado. Los permisos de la sesión resultantes son la intersección de las políticas basadas en identidades del rol y las políticas de la sesión. Los permisos también pueden proceder de una política en función de recursos. Una denegación explícita en cualquiera de estas políticas anulará el permiso. Para más información, consulte Políticas de sesión en la Guía del usuario de IAM.

Varios tipos de políticas

Cuando se aplican varios tipos de políticas a una solicitud, los permisos resultantes son más complicados de entender. Para saber cómo AWS determinar si se debe permitir una solicitud cuando se trata de varios tipos de políticas, consulte la lógica de evaluación de políticas en la Guía del usuario de IAM.

IAM con Amazon EMR

Puede usarlo AWS Identity and Access Management con Amazon EMR para definir usuarios, AWS recursos, grupos, funciones y políticas. También puede controlar a qué AWS servicios pueden acceder estos usuarios y roles.

Para obtener más información sobre el uso de IAM con Amazon EMR, consulte AWS Identity and Access Management para Amazon EMR.

Inicio de sesión y supervisión AWS Deep Learning AMI

La AWS Deep Learning AMI instancia incluye varias herramientas de supervisión de la GPU, incluida una utilidad que informa a Amazon de las estadísticas de uso de la GPU CloudWatch. Para obtener más información, consulte Monitorización y optimización de GPU y Monitorear Amazon EC2.

Seguimiento de uso

Las siguientes distribuciones de sistemas AWS Deep Learning AMI operativos incluyen código que permite AWS recopilar información sobre el tipo de instancia, el ID de instancia, el tipo de DLAMI y el sistema operativo. No se recopila ni se conserva información sobre los comandos utilizados en la DLAMI. No se recopila ni se conserva ninguna otra información sobre la DLAMI.

- Ubuntu 16.04
- Ubuntu 18.04
- Ubuntu 20.04
- Amazon Linux 2

Para excluirse del seguimiento del uso de su DLAMI, añada una etiqueta a su instancia de Amazon EC2 durante el lanzamiento. La etiqueta debe usar la clave `OPT_OUT_TRACKING` con el valor asociado definido como `true`. Para obtener más información, consulte Etiquetar los recursos de Amazon EC2.

Validación de conformidad para AWS Deep Learning AMI

Los auditores externos evalúan la seguridad y el cumplimiento AWS Deep Learning AMI como parte de varios programas de AWS cumplimiento. Para obtener información sobre los programas de conformidad admitidos, consulte Validación de conformidad para Amazon EC2.

Para obtener una lista de AWS los servicios incluidos en el ámbito de los programas de cumplimiento específicos, consulte los AWS servicios incluidos en el ámbito de aplicación por programa de cumplimiento y . Para obtener información general, consulte Programas de AWS cumplimiento > Programas AWS .

Puede descargar informes de auditoría de terceros utilizando AWS Artifact. Para obtener más información, consulte Descarga de informes en AWS Artifact .

Su responsabilidad de cumplimiento al utilizar DLAMI está determinada por la confidencialidad de sus datos, los objetivos de cumplimiento de su empresa y las leyes y reglamentos aplicables. AWS proporciona los siguientes recursos para ayudar con el cumplimiento:

- Security and Compliance Quick Start Guides (Guías de inicio rápido de seguridad y conformidad) (Guías de inicio rápido de seguridad y conformidad): Estas guías de implementación analizan las consideraciones en materia de arquitectura y proporcionan los pasos para implementar los entornos de referencia centrados en la seguridad y la conformidad en AWS.
- AWS Recursos de de cumplimiento: esta colección de libros de trabajo y guías puede aplicarse a su sector y ubicación.
- Evaluación de los recursos con las reglas de la guía para AWS Config desarrolladores: el AWS Config servicio evalúa en qué medida las configuraciones de los recursos cumplen con las prácticas internas, las directrices del sector y las normas.

- AWS Security Hub— Este AWS servicio proporciona una visión integral del estado de su seguridad AWS que le ayuda a comprobar su conformidad con los estándares y las mejores prácticas del sector de la seguridad.

Resiliencia en AWS Deep Learning AMI

La infraestructura AWS global se basa en AWS regiones y zonas de disponibilidad. AWS Las regiones proporcionan varias zonas de disponibilidad aisladas y separadas físicamente, que están conectadas mediante redes de baja latencia, alto rendimiento y alta redundancia. Con las zonas de disponibilidad, puede diseñar y utilizar aplicaciones y bases de datos que realizan una conmutación por error automática entre las zonas sin interrupciones. Las zonas de disponibilidad tienen una mayor disponibilidad, tolerancia a errores y escalabilidad que las infraestructuras tradicionales de uno o varios centros de datos.

Para obtener más información sobre AWS las regiones y las zonas de disponibilidad, consulte Infraestructura global.AWS

Para obtener información sobre las características que ayudan a respaldar sus necesidades de resiliencia de datos y de copia de seguridad, consulte Resiliencia en Amazon EC2.

Seguridad de la infraestructura en AWS Deep Learning AMI

La seguridad de la infraestructura de AWS Deep Learning AMI está respaldada por Amazon EC2. Para obtener más información, consulte Seguridad de la infraestructura en Amazon EC2.

Cambios importantes en el DLAMI

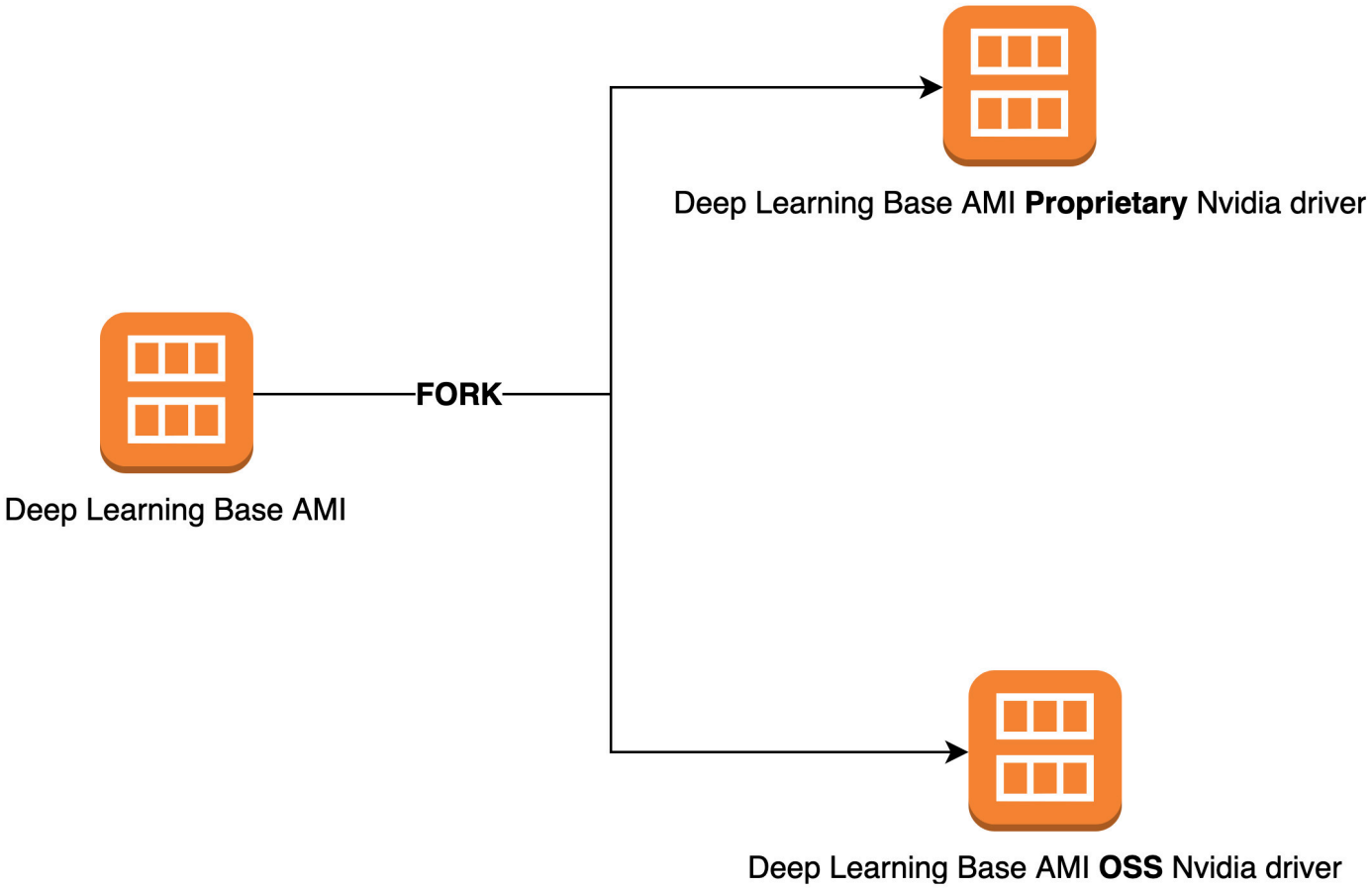
Preguntas frecuentes

¿Qué está cambiando?

El 15 de noviembre de 2023 AWS Deep Learning AMI (DLAMis) se dividirá en dos grupos separados:

- Los DLAM utilizan el controlador propietario de Nvidia (para admitir P3, P3dn y G3).
- DLAMI que utilizan el controlador OSS de Nvidia (para admitir G4dn, G5, P4, P5).

Como resultado, se crearán nuevas DLAMI para cada una de las dos categorías con nuevos nombres y nuevos ID de AMI. Estos DLAMI no serán intercambiables, es decir, los DLAMI de un grupo no admitirán instancias compatibles con el otro grupo, por ejemplo, los DLAMI que admiten p5 no admitirán g3 y viceversa.



¿Por qué es necesario este cambio?

Actualmente, las DLAMI para las GPU NVIDIA incluyen un controlador de núcleo patentado por NVIDIA. Sin embargo, recientemente, la comunidad especializada en el núcleo de Linux ha aceptado un cambio que impide que los controladores de núcleo propietarios, como el controlador de GPU de NVIDIA, se comuniquen con otros controladores de núcleo. Este cambio inhabilita GPUDirect RDMA en las instancias de las series P4/P5, que es el mecanismo que permite a las GPU utilizar el EFA de forma eficiente para la formación distribuida. Como resultado, DLAMis utilizará el controlador OpenRM (controlador de código abierto de NVIDIA), vinculado a los controladores EFA de código abierto para admitir G4dn, G5, P4 y P5. Sin embargo, este controlador OpenRM no es compatible con instancias más antiguas (P3, G3, etc.) Por lo tanto, para asegurarnos de seguir proporcionando DLAMI actuales, seguros y de alto rendimiento que admitan ambos tipos de instancias, dividiremos las DLAMI en dos grupos: uno con el controlador OpenRM (compatible con G4dn, G5, P4 y P5) y otro con el controlador propietario más antiguo (compatible con las instancias más antiguas P3, P3dn, G3).

¿Qué DLAMI se ven afectados por este cambio?

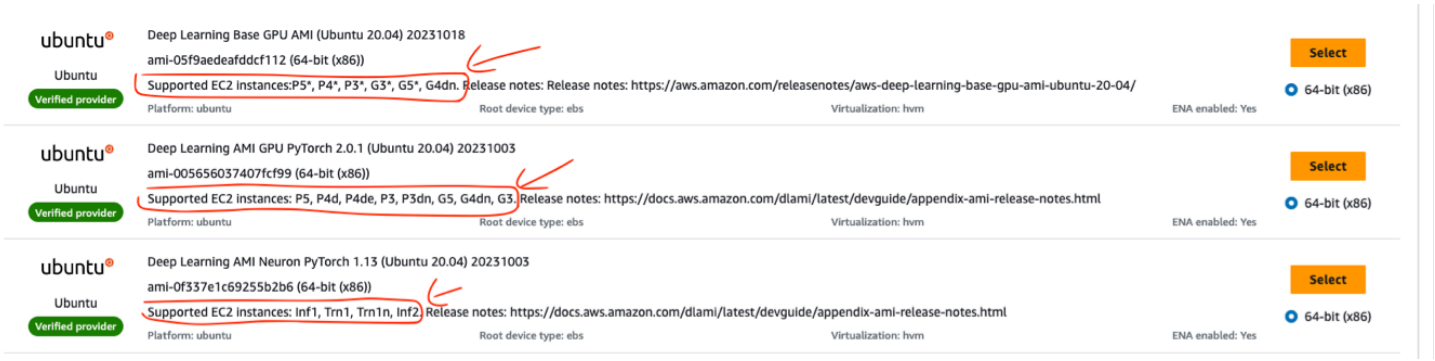
Todos los DLAMI se ven afectados por este cambio.

¿Qué significa esto para ti?

Las nuevas DLAMI seguirán ofreciendo la funcionalidad, el rendimiento y la seguridad de las DLAMI actuales siempre que se ejecuten en un tipo de instancia compatible. Si utiliza DLAMI, tendrá que asegurarse de lanzar una DLAMI en una de las instancias compatibles mencionadas en las notas de la versión de cada DLAMI (consulte aquí). Por ejemplo: tendrá que adaptar este cambio a:

- Invoque DLAMIs con las consultas CLI correctas (consulte a continuación)
- Inicie DLAMIs desde la consola y la CLI en un tipo de instancia compatible

Si lanza DLAMI desde la consola EC2, Inicio rápido: cada descripción de DLAMI muestra los tipos de instancias compatibles con la consola EC2. Debe lanzar los DLAMis en instancias compatibles.



Si lanza DLAMis mediante CLI, tendrá que modificar sus consultas. Por ejemplo:

Actualmente, la siguiente consulta CLI se utiliza para las DLAMI base que admiten todas las instancias [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
```

Las nuevas consultas CLI serán:

Para el DLAMI básico compatible con P3, P3dn y 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
```

Para el DLAMI básico compatible con G4dn, G5, P4 y 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
```

Consulte las notas de la versión actualizada de las nuevas AMI aquí. Para saber cómo lanzar las AMI en instancias EC2, consulte las instrucciones aquí.

¿Cuándo debería empezar a utilizar las nuevas DLAMIs?

Debería empezar a utilizar los nuevos DLAMIs lo antes posible para disponer de los marcos, las dependencias, los parches y las funcionalidades más recientes. Opcionalmente, si utiliza las DLAMI de Amazon Linux 2 publicadas antes del 8/11/2023, puede optar por seguir aplicando parches en directo a sus DLAMIs (consulte las instrucciones aquí) hasta el 30 de noviembre de 2023.

¿Se producirá alguna pérdida de funcionalidad con los nuevos DLAMIs?

No, no hay pérdida de funcionalidad con los nuevos DLAMis. Tras la división, las nuevas DLAMI seguirán ofreciendo toda la funcionalidad, el rendimiento y la seguridad de las antiguas DLAMI anteriores a la división, siempre y cuando se ejecuten en una instancia compatible. Vamos a dividir los DLAMI en dos grupos para seguir ofreciendo DLAMI actuales, eficaces y seguros para su uso en una amplia gama de instancias.

¿Qué pasa con los DLC?

Los DLC no incluyen el controlador NVIDIA, por lo que no se ven afectados por este cambio. Sin embargo, debe asegurarse de que los DLC se ejecuten en AMI que sean compatibles con las instancias subyacentes.

Información relacionada

Temas

- Foros
- Entradas de blog relacionadas
- Preguntas frecuentes

Foros

- Foro: AMI de aprendizaje profundo de AWS

Entradas de blog relacionadas

- Lista actualizada de artículos relacionados con las AMI de aprendizaje profundo
- Lance una AWS Deep Learning AMI (en 10 minutos)
- Entrenamiento más rápido con TensorFlow 1.6 optimizado en instancias C5 y P3 de Amazon EC2
- Nuevas AMI de aprendizaje AWS profundo para profesionales de machine learning
- Nuevos cursos de formación disponibles: Introducción a machine learning y aprendizaje profundo en AWS
- Adéntrate en el aprendizaje profundo con AWS

Preguntas frecuentes

- P: ¿Cómo puedo realizar un seguimiento de los anuncios de productos relacionados con DLAMI?

  Puede hacerlo de dos formas distintas:

  - Marque esta categoría de blog, “AMI de aprendizaje profundo de AWS“, que se encuentra aquí: Lista actualizada de artículos relacionados con las AMI de aprendizaje profundo.
  - “Consulte” el foro: AMI de aprendizaje profundo de AWS
- P: ¿Están instalados los controladores de NVIDIA y CUDA?

  Sí Algunas DLAMI tienen versiones diferentes. La AMI de aprendizaje profundo con Conda tiene la versión más reciente de cualquier DLAMI. Esto se explica de forma más detallada en

Instalaciones de CUDA y enlaces de marco de trabajo. También puede consultar las notas de versión específicas de la AMI para confirmar qué es lo que está instalado.

- P: ¿Está instalado cuDNN?

  Sí

- P: ¿Cómo puedo saber si se han detectado las GPU y su estado actual?

  Ejecute `nvidia-smi`. Esto mostrará una o varias GPU, dependiendo del tipo de instancia, junto con su consumo de memoria actual.

- P: ¿Están ya configurados los entornos virtuales?

  Sí, pero solo en la AMI de aprendizaje profundo con Conda.

- P: ¿Qué versión de Python está instalada?

  Cada DLAMI tiene Python 2 y 3. Las AMI de aprendizaje profundo con Conda tienen entornos para ambas versiones y para cada marco de trabajo.

- P: ¿Está instalado Keras?

  Esto depende de la AMI. La AMI de aprendizaje profundo con Conda tiene Keras disponible como front-end para cada marco de trabajo. La versión de Keras depende de su compatibilidad con el marco de trabajo.

- P: ¿Es gratis?

  Todas las DLAMI son gratuitas. Sin embargo, dependiendo del tipo de instancia que elija, es posible que esta no sea gratuita. Consulte Precios de la DLAMI para obtener más información.

- P: Aparecen errores de CUDA o mensajes relacionados con la GPU en mi marco de trabajo. ¿Por qué?

  Compruebe el tipo de instancia que ha utilizado. Muchos de los ejemplos y tutoriales requieren que la instancia tenga una GPU. Si la ejecución de `nvidia-smi` muestra que no hay ninguna GPU, deberá activar otra DLAMI utilizando una instancia que tenga al menos una GPU. Consulte Selección del tipo de instancia para DLAMI para obtener más información.

- P: ¿Puedo usar Docker?

  Docker se ha preinstalado a partir de la versión 14 de la AMI de aprendizaje profundo con Conda. Tenga en cuenta que deberá utilizar nvidia-docker en instancias de GPU para hacer uso de la GPU.

- P: ¿En qué regiones están disponibles las DLAMI de Linux?

Región	Code
US East (Ohio)	us-east-2
US East (N. Virginia)	us-east-1
GovCloud	us-gov-oeste-1
EE. UU. Oeste (Norte de California)	us-west-1
US West (Oregon)	us-west-2
China (Pekín)	cn-north-1
China (Ningxia)	cn-northwest-1
Asia Pacífico (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 (Fráncfort)	eu-central-1
UE (Irlanda)	eu-west-1
UE (Londres)	eu-west-2
UE (París)	eu-west-3
SA (São Paulo)	sa-east-1

- P: ¿En qué regiones están disponibles las DLAMI de Windows?

Región	Code
US East (Ohio)	us-east-2
US East (N. Virginia)	us-east-1
GovCloud	us-gov-oeste-1
EE. UU. Oeste (Norte de California)	us-west-1
US West (Oregon)	us-west-2
China (Pekín)	cn-north-1
Asia Pacífico (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 (Fráncfort)	eu-central-1
UE (Irlanda)	eu-west-1
UE (Londres)	eu-west-2
UE (París)	eu-west-3
SA (São Paulo)	sa-east-1

Notas de la versión de DLAMI

> **Note**
>
> Las AWS Deep Learning AMI tienen parches de seguridad que se publican cada noche. Estos parches de seguridad incrementales no están incluidos en las notas de lanzamiento oficiales.

Consulte la página de la política de soporte de DLAMI para ver cualquier nota de versión del marco no compatible.

DLAMI base

GPU

- AWS AMI de Deep Learning Base (Amazon Linux 2)
- AWS AMI de Deep Learning Base (Ubuntu 20.04)

AWS Neuron

- AWS Base de aprendizaje profundo AMI Neuron (Amazon Linux 2)
- AWS Base de aprendizaje profundo AMI Neuron (Ubuntu 20.04)

Qualcomm

- AWS Base de aprendizaje profundo (AMI de Qualcomm) (Amazon Linux 2)

DLAMI de marco único

PyTorch-AMI específica

- GPU
  - AWS GPU AMI PyTorch 2.1 de aprendizaje profundo (Ubuntu 20.04)
  - AWS GPU AMI de aprendizaje profundo PyTorch 1.13 (Amazon Linux 2)

  - AWS GPU AMI de aprendizaje profundo PyTorch 1.13 (Ubuntu 20.04)
- AWS Neuron
  - AWS Aprendizaje profundo AMI Neuron PyTorch 1.13 (Amazon Linux 2)
  - AWS Aprendizaje profundo AMI Neuron PyTorch 1.13 (Ubuntu 20.04)

TensorFlow-AMI específica

- GPU
  - AWS GPU AMI de aprendizaje profundo TensorFlow 2.15 (Amazon Linux 2)
  - AWS GPU AMI de aprendizaje profundo TensorFlow 2.15 (Ubuntu 20.04)
  - AWS GPU AMI de aprendizaje profundo TensorFlow 2.13 (Amazon Linux 2)
  - AWS GPU AMI de aprendizaje profundo TensorFlow 2.13 (Ubuntu 20.04)
- AWS Neuron
  - AWS Aprendizaje profundo AMI Neuron TensorFlow 2.10 (Amazon Linux 2)
  - AWS Aprendizaje profundo AMI Neuron TensorFlow 2.10 (Ubuntu 20.04)

DLAMI multimarco

GPU

> **Note**
>
> Si solo utiliza un marco de aprendizaje automático, le recomendamos un DLAMI de marco único

- AWS AMI de aprendizaje profundo (Amazon Linux 2)

AWS Neuron

- AWS AMI Neuron de aprendizaje profundo (Ubuntu 22.04)

Avisos de obsolescencia de DLAMI

En la tabla siguiente se muestra información sobre las características obsoletas en AWS Deep Learning AMI.

Características obsoletas	Fecha de obsolescencia	Aviso de obsolescencia
Ubuntu 16.04	07/10/2021	Ubuntu Linux 16.04 LTS finalizó su período de cinco años de LTS el 30 de abril de 2021 y su proveedor ya no lo ofrece. A partir de octubre de 2021, ya no hay actualizaciones de la AMI base de aprendizaje profundo (Ubuntu 16.04) en las nuevas versiones. Las versiones anteriores seguirán estando disponibl es.
Amazon Linux	07/10/2021	Amazon Linux llegó al final de su vida útil en diciembre de 2020. A partir de octubre de 2021, ya no hay actualizaciones de la AMI de aprendizaje profundo (Amazon Linux) en las nuevas versiones. Las versiones anteriores de la AMI de aprendizaje profundo (Amazon Linux) seguirán estando disponibl es.
Chainer	01/07/2020	Chainer ha anunciado el final de los principal

Características obsoletas	Fecha de obsolescencia	Aviso de obsolescencia
		es lanzamientos a partir de diciembre de 2019. En consecuencia, ya no incluiremos entornos Chainer Conda en DLAMI a partir de julio de 2020. Las versiones anteriores de DLAMI que contienen estos entornos seguirán estando disponibles. Proporcionaremos actualizaciones a estos entornos solo si la comunidad de código abierto publica correcciones de seguridad para estos marcos.
Python 3.6	15/06/2020	Debido a las solicitudes de los clientes, vamos a pasar a Python 3.7 para nuevas versiones de TF/MX/PT.

Características obsoletas	Fecha de obsolescencia	Aviso de obsolescencia
Python 2	01/01/2020	La comunidad de código abierto de Python ha terminado oficialmente la compatibilidad con Python 2. Las comunidades de TensorFlow, PyTorch y MXNet también han anunciado que las versiones TensorFlow 1.15, TensorFlow 2.1, PyTorch 1.4 y MXNet 1.6.0 serán las últimas compatibles con Python 2.

Historial de revisión de la Guía para desarrolladores de AWS Deep Learning AMI

Cambio	Descripción	Fecha
DLAMI de Graviton	Ahora AWS Deep Learning AMI admite imágenes en las GPU de Graviton basadas en el procesador Arm.	29 de noviembre de 2021
DLAMI de Habana	Ahora AWS Deep Learning AMI es compatible con el hardware de Habana Gaudi y el SDK de Habana SynapseAI.	25 de octubre de 2021
TensorFlow 2	La AMI de aprendiza je profundo con Conda se suministra ahora con TensorFlow 2 con CUDA 10.	3 de diciembre de 2019
AWS Inferentia	La AMI de aprendizaje profundo ahora admite el hardware de AWS Inferentia y el SDK AWS Neuron.	3 de diciembre de 2019
Uso de TensorFlow Serving con un modelo de incepción	Se ha añadido un ejemplo para utilizar la inferencia con un modelo de incepción para TensorFlow Serving, con y sin Elastic Inference.	28 de noviembre de 2018
Entrenamiento con 256 GPU con TensorFlow y Horovod	El tutorial de TensorFlow con Horovod se ha actualiza do para añadir un ejemplo de entrenamiento en varios nodos.	28 de noviembre de 2018

Elastic Inference	Se han añadido los requisitos previos e información relacionada con Elastic Inference a la guía de configuración.	28 de noviembre de 2018
MMS v1.0 está disponible en la DLAMI.	El tutorial de MMS se ha actualizado para utilizar el nuevo modelo formato de archivo para modelos (.mar) y para mostrar las nuevas características de iniciar y detener.	15 de noviembre de 2018
Instalación de TensorFlow desde una compilación nocturna	Se ha añadido un tutorial que explica cómo puede desinstalar TensorFlow y después instalar una compilación nocturna de TensorFlow en una AMI de aprendizaje profundo con Conda.	16 de octubre de 2018
Instalación de CNTK desde una compilación nocturna	Se ha añadido un tutorial que explica cómo puede desinstalar CNTK y después instalar una compilación nocturna de CNTK en su AMI de aprendizaje profundo con Conda.	16 de octubre de 2018
Instalación de Apache MXNet (Incubating) desde una compilación nocturna	Se ha añadido un tutorial que explica cómo puede desinstalar MXNet y después instalar una compilación nocturna de MXNet en su AMI de aprendizaje profundo con Conda.	16 de octubre de 2018

Instalación de PyTorch desde una compilación nocturna	Se ha añadido un tutorial que explica cómo puede desinstalar PyTorch y después instalar una compilación nocturna de PyTorch en su AMI de aprendizaje profundo con Conda.	25 de septiembre de 2018
Ahora Docker está preinstalado en su DLAMI	A partir de la versión 14 de la AMI de aprendizaje profundo con Conda, se han preinstalado Docker y la versión de NVIDIA de Docker para las GPU.	25 de septiembre de 2018
Tutorial de TensorBoard	El ejemplo se ha movido a ~/examples/tensorboard. Rutas del tutorial actualizadas.	23 de julio de 2018
Tutorial de MXBoard	Se ha añadido un tutorial sobre cómo utilizar MXBoard para la visualización de modelos MXNet.	23 de julio de 2018
Tutoriales de entrenamiento distribuido	Se ha añadido un tutorial sobre cómo utilizar Keras-MXNet para el entrenamiento en varias GPU. Se ha actualizado el tutorial de Chainer para v4.2.0.	23 de julio de 2018
Tutorial de Conda	Se ha actualizado el ejemplo MOTD para reflejar una versión más reciente.	23 de julio de 2018

Tutorial de Chainer	Se ha actualizado el tutorial para utilizar los últimos ejemplos del origen de Chainer.	23 de julio de 2018

Actualizaciones anteriores:

En la siguiente tabla se describen los cambios importantes de cada versión de la AWS Deep Learning AMI anteriores a julio de 2018.

Cambio	Descripción	Fecha
TensorFlow con Horovod	Se ha añadido un tutorial para capacitación de ImageNet con TensorFlow y Horovod.	6 de junio de 2018
Actualización de guía	Se ha añadido la guía de actualización.	15 de mayo de 2018
Nueva regiones y nuevo tutorial de 10 minutos	Se han añadido más regiones: EE.UU. Oeste (Norte de California), América del Sur, Canadá (Central), UE (Londres) y UE (París). Además, la primera versión de un tutorial de 10 minutos titulado: "Introducción a Deep Learning AMI".	26 de abril de 2018
Tutorial de Chainer	Se añadió un tutorial para utilizar Chainer en modos de varias GPU, GPU única y CPU. La integración de CUDA se actualizó de CUDA 8 a CUDA 9 para varios marcos de trabajo.	28 de febrero de 2018

Cambio	Descripción	Fecha
AMI de Linux v3.0, además de la introducción de MXNet Model Server, TensorFlow Serving y TensorBoard	Se han añadido tutoriales para las AMI de Conda con nuevas funciones de distribución de modelos y visualizaciones que utilizan MXNet Model Server v0.1.5, TensorFlow Serving v1.4.0 y TensorBoard v0.4.0. Las funciones CUDA de las AMI y las plataformas se describen en la información general sobre Conda y CUDA. Las notas de la versión más recientes se han movido a https://aws.amazon.com/releasenotes/	25 de enero de 2018
AMI de Linux v2.0	AMI Base, Source y Conda actualizadas con NCCL 2.1. AMI Source y Conda actualizadas con MXNet v1.0, PyTorch 0.3.0 y Keras 2.0.9.	11 de diciembre de 2017
Se han añadido dos opciones de AMI de Windows	Se han publicado las AMI para Windows 2012 R2 y 2016. Se han añadido a la guía de selección de AMI y a las notas de la versión.	30 de noviembre de 2017
Publicación inicial de la documentación	Descripción detallada del cambio con un enlace al tema o la sección que se ha modificado.	15 de noviembre de 2017

Glosario de AWS

Para ver la terminología más reciente de AWS, consulte el Glosario de AWS en la Referencia de Glosario de AWS.

Las traducciones son generadas a través de traducción automática. En caso de conflicto entre la traducción y la version original de inglés, prevalecerá la version en inglés.