aws

Guia do Desenvolvedor

AMI de deep learning

AMI de deep learning: Guia do Desenvolvedor

Table of Contents

O que é a AWS Deep Learning AMI?

Bem-vindo ao Guia do usuário da AWS Deep Learning AMI.

A AWS Deep Learning AMI (DLAMI) é o único local para aprendizado profundo na nuvem. Essa instância de máquina personalizada está disponível na maioria das regiões do Amazon EC2 para vários tipos de instância, de uma instância pequena de apenas CPU até instâncias de várias GPUs com alta potência. Ela é fornecida pré-configurada com o NVIDIA CUDA e o NVIDIA cuDNN, além das versões mais recentes das estruturas de aprendizado profundo mais populares.

Sobre este guia

Este guia ajudará você a iniciar e usar a DLAMI. Ele abrange vários casos de uso que são comuns em aprendizado profundo, para treinamento e inferência. A escolha da AMI correta para o objetivo e o tipo de instâncias de sua preferência também é abordada. A DLAMI é fornecida com vários tutoriais para cada uma das estruturas. Há também tutoriais sobre treinamento distribuído, depuração, uso do AWS Inferentia e outros conceitos-chave. Você encontrará instruções sobre como configurar o Jupyter para executar os tutoriais em seu navegador.

Pré-requisitos

Você deve estar familiarizado com ferramentas de linha de comando e o Python básico para executar a DLAMI com êxito. Tutoriais sobre como usar cada estrutura são fornecidos pelas próprias estruturas, no entanto, este guia pode mostrar como ativar cada uma delas e encontrar os tutoriais apropriados para começar a usar.

Exemplo de uso da DLAMI

Como entender o aprendizado profundo: a DLAMI é uma ótima escolha para o aprendizado ou o ensino de estruturas de machine learning e aprendizado profundo. Ela acaba com a dor de cabeça de solucionar problemas das instalações de cada estrutura e de fazê-las funcionar em conjunto no mesmo computador. A DLAMI é fornecida com um caderno Jupyter e facilita a execução dos tutoriais fornecidos pelas estruturas para pessoas não familiarizadas com machine learning e aprendizado profundo.

Desenvolvimento de aplicativos: se você for um desenvolvedor de aplicativos e estiver interessado em usar aprendizado profundo para fazer com que seus aplicativos usem os últimos avanços de AI,

a DLAMI é o campo de teste perfeito para você. Cada estrutura é fornecida com tutoriais sobre como começar a usar o deep learning, e muitas têm vários modelos que facilitam testar o aprendizado profundo sem precisar criar as redes neurais você mesmo ou fazer qualquer treinamento de modelo. Alguns exemplos mostram como criar um aplicativo de detecção de imagem em apenas alguns minutos, ou como criar um aplicativo de reconhecimento de voz para seu próprio chatbot.

Machine Learning e análise de dados: se você for um cientista de dados ou estiver interessado em processar seus dados com aprendizado profundo, descobrirá que muitas das estruturas são compatíveis com R e Spark. Você encontrará tutoriais sobre como fazer regressões simples, até a criação de sistemas de processamento de dados escaláveis para sistemas de personalização e previsões.

Pesquisa: se você for um pesquisador e quiser experimentar uma nova estrutura, testar um novo modelo ou treinar novos modelos, a DLAMI e os recursos da AWS para escala podem aliviar o problema de instalações tediosas e o gerenciamento de vários nós de treinamento.

> **Note**
>
> Embora a escolha inicial possa ser atualizar seu tipo de instância para um tipo de instância maior com mais GPUs (até 8), você também pode escalar horizontalmente criando um cluster de instâncias da DLAMI. Confira Informações relacionadas para obter mais informações sobre criações de clusters.

Recursos da DLAMI

Estruturas pré-instaladas

No momento, há dois tipos principais da DLAMI com outras variações relacionadas ao sistema operacional (SO) e às versões de software:

- AMI de deep learning com Conda – estruturas instaladas separadamente usando pacotes do `conda` e ambientes separados do Python
- AMI de deep learning base - sem estruturas instaladas; apenas NVIDIA CUDA e outras dependências

A AMI de aprendizado profundo com Conda usa ambientes `conda` para isolar cada estrutura de trabalho, para que você possa alternar entre eles conforme sua necessidade e para que não se preocupe com suas dependências conflitantes.

Esta é a lista completa de estruturas compatíveis com a AMI de deep learning com Conda:

- Apache MXNet (em incubação)
- PyTorch
- TensorFlow 2

> **Note**
>
> Não incluímos mais os ambientes CNTK, Caffe, Caffe2 e Theano, Chainer ou Keras Conda na AWS Deep Learning AMI desde a versão v28. As versões anteriores da AWS Deep Learning AMI que contêm esses ambientes continuarão disponíveis. No entanto, só forneceremos atualizações para esses ambientes se houver correções de segurança publicadas pela comunidade de código aberto dessas estruturas de trabalho.

Software de GPU pré-instalado

Mesmo que você use somente uma instância de CPU, a DLAMI terá o NVIDIA CUDA e o NVIDIA cuDNN. O software instalado é o mesmo, independentemente do tipo de instância. Lembre-se de que as ferramentas específicas à GPU só funcionam em uma instância que tenha pelo menos uma GPU. Mais informações sobre isso são fornecidas em Seleção do tipo de instância para DLAMI.

Para obter mais informações sobre a instalação do CUDA, consulte Instalações do CUDA e associações da estrutura.

Fornecimento e visualização do modelo

A AMI de deep learning com Conda vem pré-instalada com dois tipos de servidores modelo, um para MXNet e um para TensorFlow, bem como TensorBoard, para visualizações de modelos.

- Model Server para Apache MXNet (MMS)
- TensorFlow Servindo
- TensorBoard

Conceitos básicos

Como começar a usar a DLAMI

Este guia inclui dicas de como selecionar a DLAMI correta para você com a seleção de um tipo de instância que se adequa a seu caso de uso e a seu orçamento, e Informações relacionadas que descreve configurações personalizadas que podem ser de seu interesse.

Se você não estiver familiarizado com o uso da AWS ou do Amazon EC2, comece com a AMI de deep learning com Conda. Se estiver familiarizado com o Amazon EC2 e outros serviços da AWS, como o Amazon EMR, o Amazon EFS ou o Amazon S3, e estiver interessado em integrar esses serviços para projetos que precisam de treinamento distribuído ou inferência, confira Informações relacionadas para ver o que se adequa a seu caso de uso.

Recomendamos que você confira Como escolher uma DLAMI para ter uma ideia do tipo de instância que pode ser melhor para seu aplicativo.

Outra opção é este tutorial rápido: Iniciar uma AWS Deep Learning AMI (em 10 minutos).

Próxima etapa

Como escolher uma DLAMI

Como escolher uma DLAMI

Oferecemos uma variedade de opções de DLAMI. Para ajudar você a selecionar a DLAMI correta para seu caso de uso, agrupamos as imagens pelo tipo de hardware ou pela funcionalidade para a qual elas foram desenvolvidas. Nossos agrupamentos de alto nível são:

- Tipo de DLAMI: CUDA em comparação a base e estrutura única em comparação estrutura múltipla (DLAMI do Conda)
- Arquitetura de computação: AWS Graviton que usa como base x86 e com base em Arm
- Tipo de processador: GPU e CPU em comparação a Inferentia e Habana
- SDK: CUDA em comparação a AWS Neuron e SynapsesAI
- Sistema operacional: Amazon Linux em comparação com Ubuntu

O restante dos tópicos deste guia ajudam você a obter mais informações e detalhes.

Tópicos

A seguir

AMI de deep learning com Conda

Instalações do CUDA e associações da estrutura

O aprendizado profundo é de última geração, e cada estrutura oferece versões "estáveis". Essas versões estáveis podem não funcionar com a implementação e os recursos mais recentes do CUDA ou do cuDNN. O caso de uso e os recursos necessários podem ajudar você a escolher uma estrutura. Se você não tiver certeza, use a AMI de deep learning mais recente com o Conda. Ela tem binários de `pip` oficiais de todas as estruturas com CUDA 10, usando a versão mais recente compatível com cada estrutura. Se você quiser as versões mais recentes e personalizar seu ambiente de aprendizado profundo, use a AMI base de deep learning.

Consulte o nosso guia em Estabilidade e candidatos a lançamento para obter mais orientações.

Escolher uma DLAMI com o CUDA

O AMI de deep learning base tem todas as séries do CUDA 11 disponíveis, incluindo 11.0, 11.1 e 11.2.

O AMI de deep learning com Conda tem todas as séries do CUDA 11 disponíveis, incluindo 11.0, 11.1 e 11.2.

> **Note**
>
> Não incluímos mais os ambientes CNTK, Caffe, Caffe2 e Theano, Chainer ou Keras Conda na AWS Deep Learning AMI desde a versão v28. As versões anteriores da AWS

Deep Learning AMI que contêm esses ambientes continuam disponíveis. No entanto, só fornecemos atualizações para esses ambientes se houver correções de segurança publicadas pela comunidade de código aberto dessas estruturas de trabalho.

Para obter números de versão específicos da estrutura, consulte Notas de versão da DLAMI

Escolha este tipo de DLAMI ou saiba mais sobre as diferentes DLAMIs com a opção a seguir.

Escolha uma das versões do CUDA e analise a lista completa das DLAMIs que têm essa versão no Apêndice, ou saiba mais sobre as diferentes DLAMIs com a opção a seguir.

A seguir

AMI de deep learning base

Tópicos relacionados

- Para obter instruções sobre como alternar entre as versões do CUDA, consulte o tutorial Uso da AMI base de deep learning.

AMI de deep learning base

A AMI do deep learning base é como uma tela vazia para aprendizado profundo. Ela vem com tudo o que você precisa até o ponto da instalação de uma determinada estrutura e tem sua escolha de versões do CUDA.

Por que escolher a DLAMI base

Esse grupo de AMIs é útil para colaboradores de projeto que desejam usar um projeto de deep learning e criar o mais recente. É para alguém que deseja implementar seu próprio ambiente com a confiança de que o software NVIDIA mais recente está instalado e funcionando para que possa concentrar-se em escolher as estruturas e versões que deseja instalar.

Escolha este tipo de DLAMI ou saiba mais sobre as diferentes DLAMIs com a opção a seguir.

A seguir

DLAMI com o Conda

Tópicos relacionados

- Uso da AMI de deep learning base

AMI de deep learning com Conda

A DLAMI do Conda usa ambientes virtuais do `conda`. Esses ambientes são configurados para manter as instalações de estruturas diferentes separadas e simplificar a alternância entre estruturas. Isso é ótimo para aprendizado e testes com todas as estruturas que a DLAMI tem para oferecer. A maioria dos usuários acreditam que a nova AMI de deep learning com Conda é perfeita para eles.

Essas AMIs são as principais DLAMIs. Elas serão atualizadas frequentemente com as versões mais recentes das estruturas, e terão o software e os drivers de GPU mais recentes. Em geral, ela é referenciada como a AWS Deep Learning AMI na maioria dos documentos.

- A DLAMI do Ubuntu 18.04 tem as seguintes estruturas: Apache MXNet (em incubação), PyTorch e TensorFlow 2.
- A DLAMI do Amazon Linux 2 tem as seguintes estruturas: Apache MXNet (em incubação), PyTorch e TensorFlow 2.

> **Note**
>
> Não incluímos mais os ambientes CNTK, Caffe, Caffe2 e Theano, Chainer e Keras Conda na AWS Deep Learning AMI desde a versão v28. As versões anteriores da AWS Deep Learning AMI que contêm esses ambientes continuam disponíveis. No entanto, só fornecemos atualizações para esses ambientes se houver correções de segurança publicadas pela comunidade de código aberto dessas estruturas de trabalho.

Estabilidade e candidatos a lançamento

As AMIs do Conda usam binários otimizados dos lançamentos formais mais recentes de cada estrutura. Versões "release candidate" e recursos experimentais não devem ser esperados. As otimizações dependem do suporte que a estrutura oferece para tecnologias de aceleração como a DNN MKL da Intel, que acelera o treinamento e a inferência em tipos de instância de CPU C4 e C5. Os binários também estão compilados para oferecer suporte aos conjuntos de instrução Intel avançados, incluindo, entre outros, AVX, AVX-2, SSE4.1 e SSE4.2. Essas operações de vetor de

aceleração e de ponto de flutuação nas arquiteturas de Intel CPU. Além disso, para tipos de instância de GPU, o CUDA e o cuDNN são atualizados com qualquer versão compatível com o release oficial mais recente.

A AMI de deep learning com Conda instala automaticamente a versão mais otimizada da estrutura para sua instância do Amazon EC2 após a primeira ativação da estrutura. Para obter mais informações, consulte Uso da AMI de deep learning com o Conda.

Se você deseja realizar a instalação da origem usando opções de compilação personalizadas ou otimizadas, a de AMI de deep learning bases pode ser a melhor opção para você.

Defasagem do Python 2

A comunidade de código aberto Python encerrou oficialmente o suporte para Python 2 em 1 de janeiro de 2020. As comunidades TensorFlow e PyTorch anunciaram que as versões TensorFlow 2.1 e PyTorch 1.4 são as últimas compatíveis com Python 2. As versões anteriores da DLAMI (v26, v25 etc.) que contêm ambientes Python 2 Conda continuarão disponíveis. No entanto, fornecemos atualizações para os ambientes Python 2 Conda em versões da DLAMI publicadas anteriormente somente se houver correções de segurança publicadas pela comunidade de código aberto para essas versões. As DLAMIs com as versões mais recentes das estruturas TensorFlow e PyTorch não contêm os ambientes Python 2 do Conda.

CUDA Support

Os números de versão específicos do CUDA podem ser encontrados nas notas de versão da DLAMI da GPU.

A seguir

Opções de arquitetura de CPU da DLAMI

Tópicos relacionados

- Para ver um tutorial sobre como usar uma AMI de deep learning com Conda, consulte o tutorial do Uso da AMI de deep learning com o Conda.

Opções de arquitetura de CPU da DLAMI

As AWS Deep Learning AMIs são oferecidos com arquiteturas de CPU do AWSGraviton2 com base em x86 ou Arm.

Escolha uma das DLAMIs da GPU do Graviton e trabalhe com uma arquitetura de CPU que usa como base Arm. Todas as outras DLAMIs de GPU são atualmente baseadas em x86.

- GPU do AWS Graviton para AMI de deep learning do CUDA 11.4 (Ubuntu 20.04)
- GPU do AWS Graviton com AMI de deep learning do TensorFlow 2.6 (Ubuntu 20.04)
- GPU do AWSGraviton para AMI de deep learning do PyTorch 1.10 (Ubuntu 20.04)

Para obter mais informações sobre a DLAMI da GPU do Graviton, consulte A DLAMI do Graviton. Para obter mais detalhes sobre os tipos de instância disponíveis, consulte Seleção do tipo de instância para DLAMI.

A seguir

Opções do sistema operacional da DLAMI

Opções do sistema operacional da DLAMI

As DLAMIs são oferecidas nos seguintes sistemas operacionais.

- Amazon Linux 2
- Ubuntu 20.04
- Ubuntu 18.04

Há versões mais antigas dos sistemas operacionais disponíveis em DLAMIs descontinuadas. Para obter mais informações sobre a suspensão de uso da DLAMI, consulte Descontinuação da DLAMI

Antes de escolher uma DLAMI, avalie o tipo de instância que você precisa e identifique sua região da AWS.

A seguir

Seleção do tipo de instância para DLAMI

Seleção do tipo de instância para DLAMI

Consulte o Catálogo da AWS Deep Learning AMI para ver as famílias de instâncias recomendadas do Amazon EC2 que são compatíveis com DLAMIs específicas.

De forma geral, considere o seguinte ao selecionar um tipo de instância para uma DLAMI.

- Se você é novo no aprendizado profundo, uma instância com uma única GPU pode atender às suas necessidades.
- Caso se preocupe com o orçamento, é possível usar instâncias somente de CPU.
- Se você deseja otimizar o alto desempenho e a eficiência de custos para inferência de modelos de aprendizado profundo, é possível usar instâncias com chips AWS Inferentia.
- Se você deseja otimizar o alto desempenho e a eficiência de custos para treinar modelos de aprendizado profundo, é possível usar instâncias com aceleradores Habana.
- Se você está procurando uma instância de GPU de alto desempenho com uma arquitetura de CPU que usa como base ARM, é possível usar o tipo de instância G5g.
- Se tiver interesse em executar um modelo pré-treinado para inferência e previsões, é possível associar um Amazon Elastic Inference à sua instância do Amazon EC2. O Amazon Elastic Inference fornece acesso a um acelerador com uma fração de uma GPU.
- Para serviços de inferência de alto volume, uma única instância de CPU com muita memória, ou um cluster dessas instâncias, pode ser uma solução melhor.
- Se estiver usando um modelo grande com muitos dados ou um tamanho de lote amplo, você precisará de uma instância maior com mais memória. Você também pode distribuir seu modelo para um cluster de GPUs. Usar uma instância com menos memória talvez seja a melhor solução se você diminuir o tamanho do lote. Isso pode afetar sua precisão e velocidade de treinamento.
- Se você estiver interessado em executar aplicativos de machine learning usando NVIDIA Collective Communications Library (NCCL) que exigem altos níveis de comunicação entre nós em escala, talvez você queira usar o Elastic Fabric Adapter (EFA).

Para obter mais detalhes sobre instâncias, consulte Tipos de instância do EC2.

Os tópicos a seguir fornecem informações sobre as considerações relacionadas aos tipos de instância.

> **Important**
>
> As Deep Learning AMIs incluem drivers, softwares ou toolkits desenvolvidos, propriedades da NVIDIA Corporation ou fornecidos por ela. Você concorda em usar esses drivers, softwares ou toolkits NVIDIA apenas em instâncias do Amazon EC2 que incluam hardware NVIDIA.

Tópicos

Definição de preço para a DLAMI

As estruturas de aprendizado profundo incluídas na DLAMI são gratuitas, e cada uma tem suas próprias licenças de código aberto. Embora o software incluído na DLAMI seja gratuito, você ainda precisa pagar pelo hardware subjacente da instância do Amazon EC2.

Alguns tipos de instância no Amazon EC2 são identificados como gratuitos. É possível executar a DLAMI em uma dessas instâncias gratuitas. Isso significa que isso é totalmente gratuito quando você usa essa capacidade de instância. Se você precisar de uma instância mais robusta, com mais núcleos de CPU, mais espaço em disco, mais RAM e uma ou mais GPUs, é necessária uma instância que não está na classe da camada gratuita.

Para obter mais informações sobre a seleção e a definição de preços das instâncias, consulte Definição de preço do Amazon EC2.

Disponibilidade de regiões da DLAMI

Cada região oferece suporte a uma diferente variedade de tipos de instância e, muitas vezes, um tipo de instância tem um custo um pouco diferente em regiões diferentes. As DLAMIs não estão disponíveis em todas as regiões, mas é possível copiar DLAMIs para a região de sua escolha. Para obter mais informações, consulte Copiar uma AMI. Observe a lista de seleções de regiões e escolha a que esteja próxima de você ou de seus clientes. Se você planeja usar mais de uma DLAMI e potencialmente criar um cluster, use a mesma região para todos os nós do cluster.

Para obter mais informações sobre regiões, acesse Regiões do EC2.

A seguir

Instâncias de GPU recomendadas

Instâncias de GPU recomendadas

Recomendamos uma instância de GPU para a maioria dos objetivos de aprendizado profundo. O treinamento de novos modelos é mais rápido em uma instância de GPU do que em uma instância de CPU. Você pode dimensionar de forma sublinear quando tem instâncias de várias GPUs ou usa treinamento distribuído em muitas instâncias com GPUs. Para configurar o treinamento distribuído, consulte Treinamento distribuído.

Os tipos de instância a seguir são compatíveis com a DLAMI. Para obter informações sobre as opções de tipo de instância de GPU e seus usos, consulte , Tipos de instância de EC2 e selecione Computação acelerada.

> **Note**
>
> O tamanho do seu modelo deve ser um fator ao selecionar uma instância. Se o modelo exceder a RAM disponível de uma instância, selecione um tipo de instância diferente com memória suficiente para o aplicativo.

- As instâncias P3 do Amazon EC2 têm até 8 GPUs NVIDIA Tesla V100.
- As instâncias P4 do Amazon EC2 têm até 8 GPUs NVIDIA Tesla A100.
- As instâncias G3 do Amazon EC2 têm até 4 GPUs NVIDIA Tesla M60.
- As instâncias G4 do Amazon EC2 têm até 4 GPUs NVIDIA T4.
- As instâncias G5 do Amazon EC2 têm até 8 GPUs NVIDIA A10G.
- As instâncias G5g do Amazon EC2 têm processadores AWS Graviton2 com base em ARM.

As instâncias da DLAMI oferecem ferramentas para monitorar e otimizar seus processos da GPU. Para obter mais informações sobre o monitoramento dos processos da GPU, consulte Monitoramento e otimização de GPU.

Para ver tutoriais específicos sobre como trabalhar com instâncias G5g, consulte A DLAMI do Graviton.

A seguir

Instâncias de CPU recomendadas

Instâncias de CPU recomendadas

Independentemente de você ter um orçamento, estar aprendendo sobre o deep learning ou apenas querer executar um serviço de previsão, você tem muitas opções acessíveis na categoria de CPU. Algumas estruturas usam a DNN MKL da Intel, o que acelera o treinamento e a inferência em tipos de instância de CPU C3, C4 e C5 (esta não disponível em todas as regiões). Para obter informações sobre os tipos de instância de CPU, consulte Tipos de instância do EC2 e selecione Otimizado para computação.

> **Note**
>
> O tamanho do seu modelo deve ser um fator ao selecionar uma instância. Se o modelo exceder a RAM disponível de uma instância, selecione um tipo de instância diferente com memória suficiente para o aplicativo.

- As instâncias C5 do Amazon EC2 têm até 72 vCPUs Intel. As instâncias C5 são excelentes para modelagem científica, processamento em lotes, análise distribuída, computação de alta performance (HPC) e inferência de machine learning e aprendizado profundo.
- As instâncias C4 do Amazon EC2 têm até 36 vCPUs Intel.

A seguir

Instâncias recomendadas do Inferentia

Instâncias recomendadas do Inferentia

As instâncias do AWS Inferentia foram projetadas para fornecer alto desempenho e economia para o treinamento de modelos de aprendizado profundo. Especificamente, os tipos de instância Inf2 usam chips AWS Inferentia e o SDK do AWS Neuron, que é integrado a estruturas populares de machine learning, como TensorFlow e PyTorch.

Os clientes podem usar instâncias Inf2 para executar aplicativos de inferência de machine learning em grande escala, como pesquisa, mecanismos de recomendação, visão computacional, reconhecimento de fala, processamento de linguagem natural, personalização e detecção de fraudes, com o menor custo na nuvem.

> **Note**
>
> O tamanho do seu modelo deve ser um fator ao selecionar uma instância. Se o modelo exceder a RAM disponível de uma instância, selecione um tipo de instância diferente com memória suficiente para o aplicativo.

- As instâncias Inf2 do Amazon EC2 têm até 16 chips AWS Inferentia e 100 Gbps de throughput de rede.

Para obter mais informações sobre como usar as DLAMIs do AWS Inferentia, consulte O chip de AWS inferência com DLAMI.

A seguir

Instâncias recomendadas do Trainium

Instâncias recomendadas do Trainium

As instâncias do AWS Trainium foram projetadas para fornecer alto desempenho e economia para o treinamento de modelos de aprendizado profundo. Especificamente, os tipos de instância Trn1 usam chips AWS Trainium e o SDK do AWS Neuron, que é integrado a estruturas populares de machine learning, como TensorFlow e PyTorch.

Os clientes podem usar instâncias Trn1 para executar aplicativos de inferência de machine learning em grande escala, como pesquisa, mecanismos de recomendação, visão computacional, reconhecimento de fala, processamento de linguagem natural, personalização e detecção de fraudes, com o menor custo na nuvem.

> **Note**
>
> O tamanho do seu modelo deve ser um fator ao selecionar uma instância. Se o modelo exceder a RAM disponível de uma instância, selecione um tipo de instância diferente com memória suficiente para o aplicativo.

- As instâncias Trn1 do Amazon EC2 têm até 16 chips AWS Trainium e 100 Gbps de throughput de rede.

A seguir

Instâncias recomendadas do Habana

Instâncias recomendadas do Habana

As instâncias com aceleradores Habana foram projetadas para fornecer alto desempenho e economia para workloads de treinamento dos modelos de aprendizado profundo. Especificamente, os tipos de instância DL1 usam aceleradores Habana Gaudi da Habana Labs, uma empresa da Intel. As instâncias DL1 são ideais para treinar modelos de machine learning usados em aplicações como processamento de linguagem natural, detecção e classificação de objetos, mecanismos de recomendação e percepção de veículos autônomos.

As instâncias com aceleradores Habana são configuradas com o software Habana SynapseAI e pré-integradas a estruturas populares de machine learning, como TensorFlow e PyTorch. Se você está procurando uma combinação ideal de desempenho e preço para treinar modelos de aprendizado profundo, use as instâncias com aceleradores Habana para obter o menor custo de treinamento.

> **Note**
>
> O tamanho do seu modelo deve ser um fator ao selecionar uma instância. Se o modelo exceder a RAM disponível de uma instância, selecione um tipo de instância diferente com memória suficiente para o aplicativo.

- As instâncias DL1 do Amazon EC2 têm até oito aceleradores Habana Gaudi, 256 GB de memória aceleradora, 4 TB de armazenamento NVMe local e 400 Gbps de throughput de rede.

Para obter mais informações sobre como começar a usar as DLAMIs do Habana, consulte A DLAMI Habana.

Política de suporte da estrutura

As AWS Deep Learning AMIs (DLAMIS) simplificam a configuração de imagens para workloads de aprendizado profundo e são otimizados com as estruturas, o hardware, os drivers, as bibliotecas e os sistemas operacionais mais recentes. Esta página detalha a política de suporte da estrutura para DLAMIs. Para obter uma lista das DLAMIs disponíveis, consulte as Notas de versão da DLAMI.

Estruturas compatíveis

Consulte a tabela de Políticas de suporte da estrutura da AWS Deep Learning AMI a seguir para verificar quais estruturas e versões são compatíveis.

Consulte Fim do patch para verificar por quanto tempo AWS oferece suporte às versões atuais que são ativamente apoiadas pela equipe de manutenção da estrutura de origem. As estruturas e versões estão disponíveis em DLAMIs de estrutura única ou de várias estruturas.

> **Note**
>
> Na versão do de estrutura x.y.z, x se refere à versão principal, y se refere à versão secundária e z se refere à versão do patch. Por exemplo, para TensorFlow 2.6.5, a versão principal é 2, a versão secundária é 6 e a versão do patch é 5.

Consulte as notas de versão para obter mais detalhes sobre imagens específicas:

- Notas de versão da DLAMI de estrutura única
- Notas de versão da DLAMI de várias estruturas

Perguntas frequentes

- Quais versões da estrutura recebem patches de segurança?
- Quais imagens da AWS são publicadas quando novas versões da estrutura são lançadas?
- Quais imagens recebem SageMaker AWS novos/recursos?
- Como a versão atual é definida na tabela de Estruturas compatíveis?
- E se eu executar uma versão que não está na tabela Estruturas compatíveis?
- O DLamis oferece suporte a versões anteriores do? TensorFlow

- Como posso encontrar a imagem com patch aplicado mais recente de uma versão compatível da estrutura?
- Com que frequência novas imagens são lançadas?
- Minha instância receberá um patch enquanto a workload estiver em execução?
- O que acontece quando uma nova versão com patch aplicado ou atualizada da estrutura está disponível?
- As dependências são atualizadas sem alterar a versão da estrutura?
- Quando o suporte ativo para minha versão da estrutura termina?
- As imagens com versões de estrutura que não são mais mantidas ativamente receberão um patch?
- Como usar uma versão de estrutura mais antiga?
- Como faço para up-to-date acompanhar as mudanças de suporte nas estruturas e suas versões?
- Preciso de uma licença comercial para usar o repositório Anaconda?

Quais versões da estrutura recebem patches de segurança?

Se a versão da estrutura for identificada como tendo Suporte na tabela de Política de suporte da estrutura da AWS Deep Learning AMI, ela receberá patches de segurança.

Quais imagens da AWS são publicadas quando novas versões da estrutura são lançadas?

Publicamos novos DLAMis logo após o lançamento de novas versões do TensorFlow . PyTorch Isso inclui versões principais, versões principais e secundárias e major-minor-patch versões de estruturas. Também atualizamos as imagens quando novas versões de drivers e bibliotecas são disponibilizadas. Para obter mais informações sobre a manutenção da imagem, consulte Quando o suporte ativo para minha versão da estrutura termina?

Quais imagens recebem SageMaker AWS novos/recursos?

Os novos recursos geralmente são lançados na versão mais recente do DLamis para e. PyTorch TensorFlow Consulte as notas de lançamento de uma imagem específica para obter detalhes sobre novidades SageMaker ou AWS recursos. Para obter uma lista das DLAMIs disponíveis, consulte as Notas de versão da DLAMI. Para obter mais informações sobre a manutenção da imagem, consulte Quando o suporte ativo para minha versão da estrutura termina?

Como a versão atual é definida na tabela de Estruturas compatíveis?

A versão atual na tabela AWS Deep Learning AMI Framework Support Policy se refere à versão mais recente da estrutura AWS disponibilizada em GitHub. Cada versão mais recente inclui atualizações de drivers, bibliotecas e pacotes relevantes na DLAMI. Para obter informações sobre a manutenção de imagens, consulte Quando o suporte ativo para minha versão da estrutura termina?

E se eu executar uma versão que não está na tabela Estruturas compatíveis?

Se você estiver executando uma versão que não está na AWS Deep Learning AMI tabela de Política de estruturas compatíveis, talvez você não tenha os drivers, as bibliotecas e os pacotes relevantes mais atualizados. Para uma up-to-date versão adicional, recomendamos que você atualize para uma das estruturas suportadas disponíveis usando a DLAMI mais recente de sua escolha. Para obter uma lista das DLAMIs disponíveis, consulte as Notas de versão da DLAMI.

O DLamis oferece suporte a versões anteriores do? TensorFlow

Não. Oferecemos suporte à versão de patch mais recente da versão principal mais recente de cada estrutura, lançada 365 dias após o GitHub lançamento inicial, conforme declarado na tabela de políticas de suporte da AWS Deep Learning AMI estrutura. Para obter mais informações, consulte E se eu executar uma versão que não está na tabela Estruturas compatíveis?.

Como posso encontrar a imagem com patch aplicado mais recente de uma versão compatível da estrutura?

Para usar uma DLAMI com a versão mais recente da estrutura, recupere o ID da DLAMI e use-a para iniciar a DLAMI usando o console EC2. Para exemplos de comandos de CLI da AWS para recuperar o ID da AWS Deep Learning AMI, consulte a seção Estruturas de deep learning no Catálogo da AWS Deep Learning AMI. AWS As consultas de ID da AMI para CLI também estão incluídas nas notas de versão da DLAMI de estrutura única. A versão da estrutura que você escolher deve ser identificada como tendo Suporte na tabela de Política de suporte da estrutura da AWS Deep Learning AMI.

Com que frequência novas imagens são lançadas?

Nossa maior prioridade é fornecer versões de patch atualizadas. Criamos rotineiramente imagens com patch aplicado na primeira oportunidade. Monitoramos as versões recém-corrigidas da estrutura

(ex. TensorFlow 2.9 a TensorFlow 2.9.1) e novas versões secundárias (ex. TensorFlow 2.9 a TensorFlow 2.10) e disponibilize-os o mais rápido possível. Quando uma versão existente do TensorFlow é lançada com uma nova versão do CUDA, lançamos um novo DLAMI para essa versão TensorFlow do com suporte para a nova versão do CUDA.

Minha instância receberá um patch enquanto a workload estiver em execução?

Não. As atualizações de patch para a DLAMI não são atualizações “locais”.

Você deve ativar uma nova instância do EC2, migrar as cargas de trabalho e scripts. Em seguida, desativar a instância anterior.

O que acontece quando uma nova versão com patch aplicado ou atualizada da estrutura está disponível?

Verifique regularmente a imagem na página das notas de versão. É recomendado atualizar para novas estruturas com patch aplicado ou mais recentes quando estiverem disponíveis. Para obter uma lista das DLAMIs disponíveis, consulte as Notas de versão da DLAMI.

As dependências são atualizadas sem alterar a versão da estrutura?

Atualizamos as dependências sem alterar a versão da estrutura. No entanto, se uma atualização de dependência causar uma incompatibilidade, criaremos uma imagem com uma versão diferente. Verifique as notas de versão da DLAMI para obter informações atualizadas sobre as dependências.

Quando o suporte ativo para minha versão da estrutura termina?

As imagens da DLAMI são imutáveis. Depois de criadas, elas não mudam. Há quatro motivos principais para o fim do suporte ativo de uma versão da estrutura:

- Atualizações da versão da estrutura (patch)
- Patches de segurança da AWS
- Data de fim do patch (descontinuação)
- Dependência end-of-support

> **Note**
>
> Devido à frequência de atualizações de patches de versão e de segurança, é recomendado verificar a página das notas de versão da DLAMI com frequência e atualizá-la quando houver alterações.

Atualizações da versão da estrutura (patch)

Se você tem uma carga de trabalho de DLAMI TensorFlow baseada em 2.7.0 TensorFlow e libera a versão 2.7.1, então libera uma nova DLAMI GitHub com 2.7.1. AWS TensorFlow As imagens anteriores com 2.7.0 não são mais mantidas ativamente depois que a nova imagem com TensorFlow 2.7.1 é lançada. O DLAMI TensorFlow com 2.7.0 não recebe mais patches. A página de notas de lançamento do DLAMI TensorFlow para 2.7 é então atualizada com as informações mais recentes. Não há uma página individual das notas de versão para cada patch secundário.

As novas DLAMIs criadas devido a atualizações de patches são designadas com um novo ID da AMI.

Patches de segurança da AWS

Se você tiver uma carga de trabalho baseada em uma imagem com TensorFlow 2.7.0 e AWS fizer um patch de segurança, uma nova versão do DLAMI será lançada para a 2.7.0. TensorFlow A versão anterior das imagens com TensorFlow 2.7.0 não é mais mantida ativamente. Para obter mais informações, consulte Minha instância receberá um patch enquanto a workload estiver em execução? Para ver as etapas e encontrar a DLAMI mais recente, consulte Como posso encontrar a imagem com patch aplicado mais recente de uma versão compatível da estrutura?

As novas DLAMIs criadas devido a atualizações de patches são designadas com um novo ID da AMI.

Data de fim do patch (descontinuação)

O DLamis atingiu o fim da data de patch 365 dias após a data de GitHub lançamento.

Para DLAMIs de várias estruturas, quando uma das versões da estrutura é atualizada, é necessária uma nova DLAMI com a versão atualizada. A DLAMI com a versão antiga da estrutura não é mais mantida ativamente.

> **Important**
>
> A exceção é quando há uma grande atualização da estrutura. Por exemplo, se a versão TensorFlow 1.15 for atualizada para TensorFlow 2.0, continuaremos a oferecer suporte à versão mais recente da TensorFlow 1.15 por um período de dois anos a partir da data de GitHub lançamento ou seis meses após a equipe de manutenção da estrutura de origem cancelar o suporte, qualquer que seja a data anterior.

Dependência end-of-support

Se você estiver executando uma carga de trabalho em uma imagem DLAMI TensorFlow 2.7.0 com o Python 3.6 e essa versão do Python estiver marcada como sim, todas as imagens DLAMI baseadas no Python 3.6 não end-of-support serão mais mantidas ativamente. Da mesma forma, se uma versão do sistema operacional como o Ubuntu 16.04 estiver marcada para end-of-support, todas as imagens DLAMI que dependem do Ubuntu 16.04 não serão mais mantidas ativamente.

As imagens com versões de estrutura que não são mais mantidas ativamente receberão um patch?

Não. As imagens que não são mais mantidas ativamente não terão novos lançamentos.

Como usar uma versão de estrutura mais antiga?

Para usar uma DLAMI com uma versão mais antiga da estrutura, recupere o ID da DLAMI e use-a para iniciar a DLAMI usando o Console EC2. Para obter comandos de CLI da AWS com o objetivo de recuperar o ID da AMI, consulte a seção Estruturas de aprendizado profundo no catálogo da AWSAMI de aprendizado profundo. AWS As consultas de ID da AMI para CLI também estão incluídas nas notas de versão da DLAMI de estrutura única.

Como faço para up-to-date acompanhar as mudanças de suporte nas estruturas e suas versões?

Fique up-to-date com as estruturas e versões do DLAMI usando a tabela de políticas de suporte do AWS Deep Learning AMI Framework e as notas de lançamento do DLAMI.

Preciso de uma licença comercial para usar o repositório Anaconda?

O Anaconda mudou para um modelo de licenciamento comercial com foco em determinados usuários. As DLAMIs mantidas ativamente foram migradas para a versão de código aberto disponível ao público do Conda (conda-forge) do canal Anaconda

Como executar e configurar uma DLAMI

Se você está aqui, já deve ter uma boa ideia de qual AMI deseja iniciar. Caso contrário, encontre uma DLAMI e seu hardware, as estruturas e a ID de recuperação relacionados no Catálogo da AWS Deep Learning AMI ou veja as notas da versão atuais e históricas da DLAMI em Notas de versão da DLAMI.

Você também deve saber qual tipo de instância e região escolherá. Caso contrário, navegue em Seleção do tipo de instância para DLAMI.

> **Note**
>
> Usaremos p3.16xlarge como tipo de instância padrão nos exemplos. Basta substituir por qualquer tipo de instância que você tenha em mente.

> **Important**
>
> Se você planeja usar o Elastic Inference, é necessário concluir a Configuração do Elastic Inference antes de iniciar sua DLAMI.

Tópicos

Etapa 1: executar uma DLAMI

> **Note**
>
> Para esta demonstração, podemos fazer referências específicas à AMI de deep learning (Ubuntu 18.04). Mesmo que selecione outra DLAMI, você deve ser capaz de seguir este guia.

1. Encontre a ID da sua DLAMI
2. Inicie uma instância do Amazon EC2 na DLAMI

   Você usará o console do Amazon EC2. Siga as instruções detalhadas em Executar no console do Amazon EC2

   > **Tip**
   >
   > Opção da CLI: se você optar por ativar uma DLAMI usando a CLI da AWS, precisará da ID da AMI, da região e do tipo de instância, além das informações do token de segurança. Verifique se você tem a AMI e os IDs de instância prontos. Se você ainda não configurou a CLI da AWS, faça isso primeiro usando o guia de Instalação da interface de linha de comando da AWS.

3. Depois de concluir as etapas de uma dessas opções, aguarde até que a instância esteja pronta. Isso normalmente leva apenas alguns minutos. Você pode verificar o status da instância no Console do EC2.

Recuperar o ID da DLAMI

Cada AMI possui um identificador exclusivo (ID). Você pode consultar o ID da DLAMI de sua escolha com a Interface de Linha de Comando da AWS (CLI da AWS). Se você ainda não instalou a AWS CLI, consulte Introdução à CLI da AWS.

> **Note**
>
> Lembrete: você pode encontrar todas as DLAMIs e seus processadores/aceleradores relacionados, sistema operacional, arquitetura computacional, famílias de instâncias

recomendadas do Amazon EC2, status de suporte e consultas de recuperação de ID no Catálogo da AWS Deep Learning AMI. Consulte também as notas de versão da DLAMI em Notas de versão da DLAMI para obter mais informações (drivers, versões do python, tipo do Amazon EBS).

1. As suas credenciais da AWS devem estar configuradas.

```
aws configure
```

2. Use o comando a seguir para recuperar o ID da sua DLAMI ou encontrar a consulta fornecida no Catálogo da 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**
>
> Você pode especificar uma versão de lançamento para uma determinada estrutura ou obter a versão mais recente substituindo o número dela por um ponto de interrogação.

3. A saída deve ser semelhante à seguinte:

```
ami-094c089c38ed069f2
```

Copie esse ID da DLAMI e pressione q para sair do prompt.

Próxima etapa

Executar no console do Amazon EC2

Executar no console do Amazon EC2

> **Note**
>
> Consulte estas etapas para executar uma instância com o Elastic Fabric Adapter (EFA).

1. Abra o Console do EC2.
2. Observe sua região atual na navegação superior. Se essa não for a região da Região da AWS desejada, altere essa opção antes de continuar. Para obter mais informações, consulte Regiões do EC2.
3. Selecione Executar instância.
4. Insira um nome para a instância e selecione a DLAMI adequada.
   a. Encontre uma DLAMI existente em Minhas AMIs ou escolha Início rápido.
   b. Pesquise por ID da DLAMI. Navegue pelas opções e selecione a escolhida.
5. Escolha um tipo de instância. Você pode encontrar as famílias de instâncias recomendadas para sua DLAMI no Catálogo da AWS Deep Learning AMI. Para recomendações gerais sobre os tipos de instância da DLAMI, consulte Seleção de instância.

   > **Note**
   >
   > Se você quiser usar o Elastic Inference (EI), clique em Configurar detalhes da instância, selecione Adicionar um acelerador Amazon EI e selecione o tamanho dele.

6. Selecione Executar instância.

> **Tip**
>
> Confira Introdução ao aprendizado profundo usando a AMI de deep learning da AWS para ver um passo a passo com capturas de tela.

Próxima etapa

Etapa 2: conectar-se à DLAMI

Etapa 2: conectar-se à DLAMI

Conecte-se à DLAMI que você iniciou em um cliente (Windows, MacOS ou Linux). Para obter mais informações, consulte Conectar-se à sua instância do Linux no Guia do usuário do Amazon EC2 para instâncias do Linux.

Mantenha uma cópia do comando login do SSH à mão se desejar configurar o Jupyter depois de fazer login. Você usará uma variação dele para conectar-se à página da web do Jupyter.

Próxima etapa

Etapa 3: testar a DLAMI

Etapa 3: testar a DLAMI

Dependendo da sua versão de DLAMI, você terá diferentes opções de teste:

- AMI de deep learning com Conda — acesse Uso da AMI de deep learning com o Conda.
- AMI de deep learning base — consulte a documentação de instalação da estrutura desejada.

Você também pode criar um caderno Jupyter, experimentar tutoriais ou começar a codificação em Python. Para obter mais informações, consulte Configurar um servidor do caderno Jupyter.

Etapa 4: gerenciar sua instância da DLAMI

Sempre mantenha o sistema operacional e outros softwares instalados atualizados executando patches e atualizações assim que forem disponibilizados.

Se você está usando o Amazon Linux ou Ubuntu, ao efetuar login na DLAMI, você será notificado se houver atualizações disponíveis e verá as instruções para atualização. Para obter mais informações sobre a manutenção do Amazon Linux, consulte Atualizar o software de instância. Para instâncias do Ubuntu, consulte a Documentação oficial do Ubuntu.

No Windows, verifique o Windows Update regularmente para instalar atualizações de software e de segurança. Se você preferir, as atualizações podem ser instaladas automaticamente.

> **Important**
>
> Para obter mais informações sobre as vulnerabilidades Spectre e Meltdown, assim como a forma de aplicar um patch no sistema operacional para lidar com elas, consulte o Boletim de segurança da AWS-2018-013.

Limpar

Quando não precisar mais da DLAMI, você poderá interrompê-la ou fechá-la para evitar cobranças contínuas. A interrupção de uma instância a manterá disponível para que você possa retomá-la posteriormente. Suas configurações, arquivos e outras informações não voláteis são armazenados em um volume no Amazon S3. Será cobrada uma pequena taxa do S3 para reter o volume enquanto a instância estiver interrompida, mas os recursos de computação não serão cobrados enquanto ela estiver no estado interrompido. Quando você iniciar a instância novamente, ela montará esse volume e seus dados estarão disponíveis. Se você encerrar uma instância, ela será perdida e você não poderá iniciá-la novamente. Seus dados ainda residirão no S3, portanto, para evitar cobranças adicionais você precisará excluir o volume também. Para obter mais informações, consulte Terminar a instância no Guia do usuário do Amazon EC2 para instâncias do Linux.

Configurar um servidor do caderno Jupyter

Um servidor do caderno Jupyter permite que você crie e execute cadernos Jupyter a partir de sua instância da DLAMI. Com os cadernos Jupyter, você pode realizar experimentos de machine learning (ML) para treinamento e inferência enquanto usa a infraestrutura da AWS e acessa pacotes incorporados à DLAMI. Para obter mais informações sobre cadernos Jupyter, consulte a documentação do caderno Jupyter.

Para configurar um servidor para caderno Jupyter, você deve:

- Configurar o servidor do caderno Jupyter na instância da DLAMI do Amazon EC2.
- Configurar o cliente para que possa conectar-se ao servidor do caderno Jupyter. Fornecemos instruções de configuração para Windows, macOS e clientes Linux.
- Testa a configuração fazendo login no servidor do caderno Jupyter.

Para concluir as etapas de configuração do Jupyter, siga as instruções nos tópicos a seguir. Depois de configurar um servidor do caderno Jupyter, consulte Executar os tutoriais do notebook Jupyter para obter informações sobre como executar os exemplos de cadernos fornecidos na DLAMI.

Tópicos

- Proteja seu servidor do Jupyter
- Inicie o servidor do caderno Jupyter
- Configurar o cliente para se conectar ao servidor do Jupyter

- Testar fazendo login no servidor do caderno Jupyter

Proteja seu servidor do Jupyter

Aqui, configuramos o Jupyter com SSL e uma senha personalizada.

Conecte-se à instância do Amazon EC2 e, em seguida, conclua o procedimento a seguir.

Configurar o servidor do Jupyter

1. O Jupyter fornece um utilitário de senha. Execute o comando a seguir e insira sua senha preferencial no prompt.

   ```
   $ jupyter notebook password
   ```

   O resultado será semelhante ao seguinte:

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

2. Crie um certificado SSL autoassinado. Siga os prompts para preencher sua localidade, conforme julgar necessário. É necessário inserir `.` se deseja deixar um prompt em branco. Suas respostas não afetarão a funcionalidade do certificado.

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

> **Note**
>
> Você poderá estar interessado em criar um certificado SSL regular assinado por terceiros que não faz com que o navegador forneça um aviso de segurança. Esse processo é muito mais complexo. Acesse a documentação do Jupyter para obter mais informações.

Próxima etapa

Inicie o servidor do caderno Jupyter

Inicie o servidor do caderno Jupyter

Agora, é possível acionar o servidor do Jupyter fazendo login na instância e executando o seguinte comando que usa o certificado SSL criado na etapa anterior.

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

Com o servidor iniciado, agora você pode se conectar a ele por meio de um túnel SSH em seu computador cliente. Quando o servidor for executado, você verá alguma saída do Jupyter confirmando que o servidor está em execução. Nesse ponto, ignore o texto informando que você pode acessar o servidor por meio de um URL do host local, pois isso não funcionará até criar o túnel.

> **Note**
>
> O Jupyter resolverá a troca de ambientes para você ao alternar as estruturas usando a interface da web do Jupyter. Você pode encontrar mais informações sobre este assunto em Alternar ambientes com o Jupyter.

Próxima etapa

Configurar o cliente para se conectar ao servidor do Jupyter

Configurar o cliente para se conectar ao servidor do Jupyter

Depois de configurar o cliente para se conectar ao servidor do caderno Jupyter, você pode criar e acessar cadernos no servidor em sua área de trabalho e executar o código de aprendizado profundo no servidor.

Para obter informações de configuração, escolha um dos links a seguir.

Tópicos

- Configurar um cliente Windows

- Configurar um cliente Linux ou macOS

Configurar um cliente Windows

Preparação

Você deve ter as seguintes informações, que são necessárias para configurar o túnel SSH:

- O nome DNS público da instância do Amazon EC2. Você pode localizar o nome DNS público no console do EC2.
- O par de chaves do arquivo de chave privada. Para obter mais informações sobre como acessar seu par de chaves, consulte Pares de chaves do Amazon EC2 no Guia do usuário do Amazon EC2 para instâncias do Linux.

Uso de notebooks Jupyter a partir de um cliente Windows

Consulte estes guias sobre como se conectar à sua instância do Amazon EC2 a partir de um cliente Windows.

1. Resolução de problemas para se conectar à instância
2. Conexão à instância do Linux no Windows utilizando PuTTY

Para criar um túnel para um servidor do Jupyter em execução, uma abordagem recomendada é instalar o Git Bash em seu cliente Windows e seguir as instruções do cliente Linux/macOS. No entanto, você pode usar qualquer abordagem desejada para abrir um túnel SSH com mapeamento de porta. Consulte a documentação do Jupyter para obter informações adicionais.

Próxima etapa

Configurar um cliente Linux ou macOS

Configurar um cliente Linux ou macOS

1. Abra um terminal.
2. Execute o comando a seguir para encaminhar todas as solicitações da porta local 8888 para a porta 8888 na instância do Amazon EC2 remota. Atualize o comando substituindo a localização da chave para acessar a instância do Amazon EC2 e o nome DNS público da instância. Observe que, para uma AMI do Amazon Linux, o nome do usuário `ec2-user` em vez de `ubuntu`.

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

Esse comando abre um túnel entre o cliente e a instância do Amazon EC2 remota que está executando o servidor do caderno Jupyter.

Próxima etapa

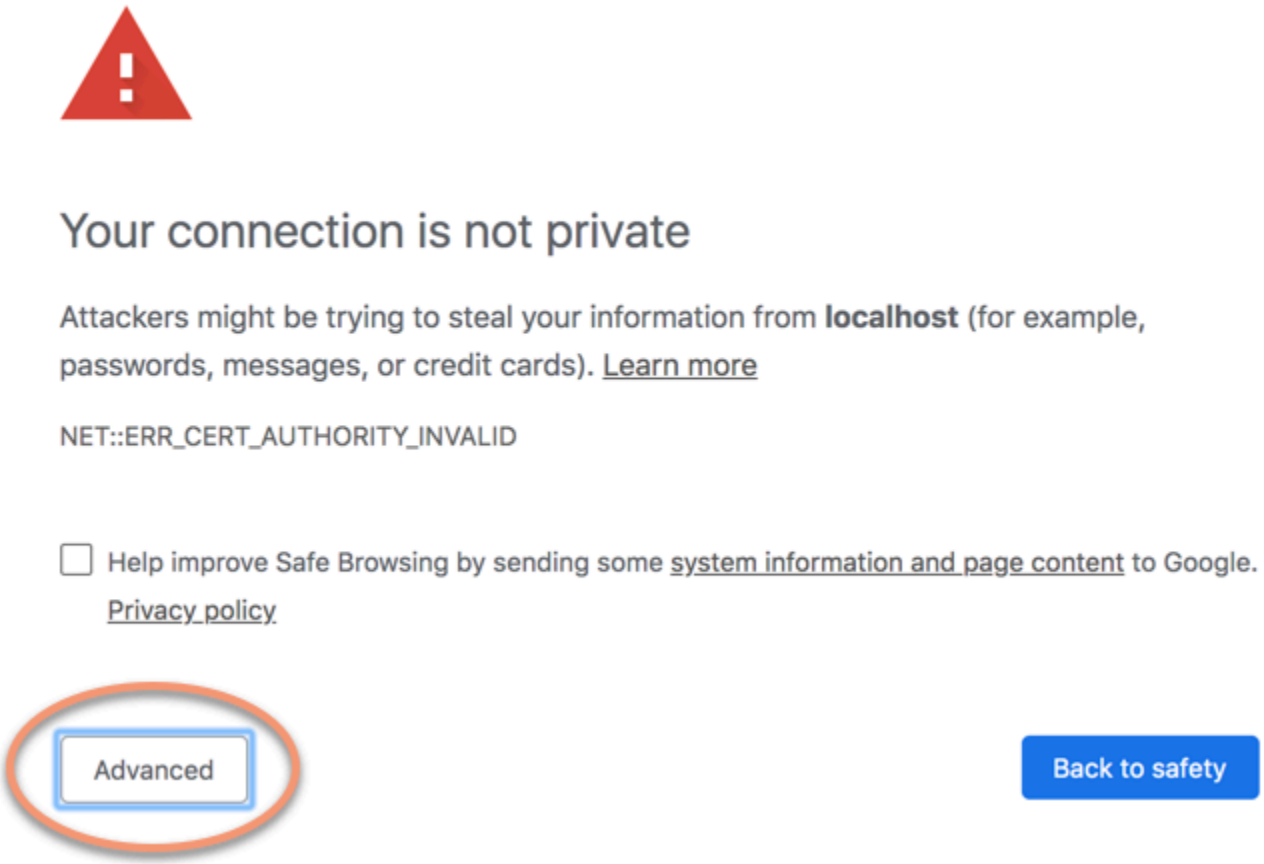
Testar fazendo login no servidor do caderno Jupyter

Testar fazendo login no servidor do caderno Jupyter

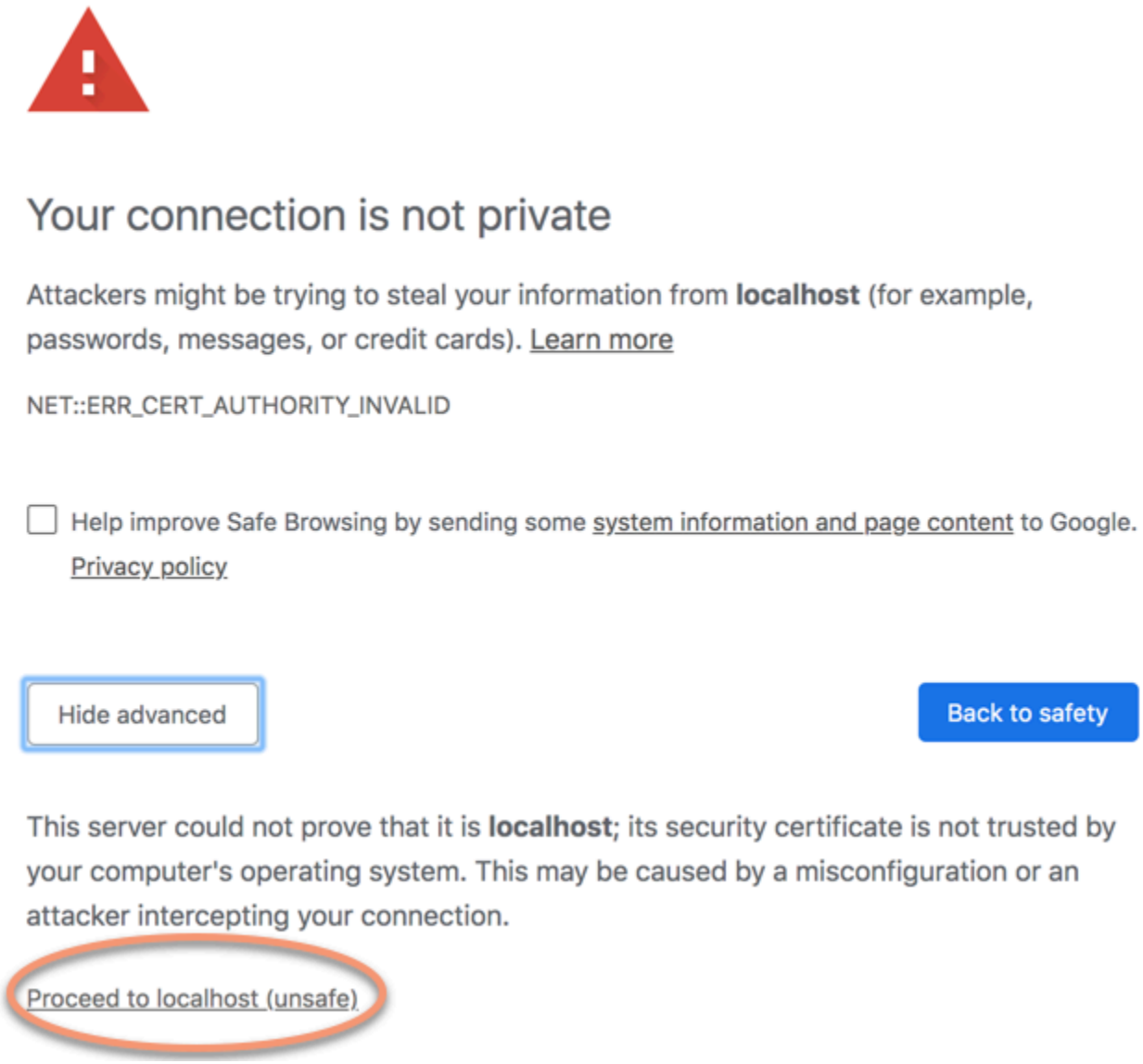
Agora, você está pronto para fazer login no servidor do caderno Jupyter.

A próxima etapa é testar a conexão ao servidor por meio de seu navegador.

1. Na barra de endereço do navegador, digite o seguinte URL ou clique neste link: https://localhost:8888
2. Com um certificado SSL autoassinado, navegador avisará você e solicitará que não continue com o acesso ao site.



Como você mesmo configurou isso, é seguro continuar. O navegador receberá um botão "avançado", "mostrar detalhes" ou algo semelhante.



Clique nele e depois clique no link "prosseguir para localhost". Se a conexão for bem-sucedida, você verá a página da web do servidor do caderno Jupyter. Nesse ponto, será solicitada a senha definida anteriormente.

Agora você tem acesso ao servidor do caderno Jupyter em execução na DLAMI. Você pode criar novos cadernos ou executar os Tutoriais fornecidos.

Utilizar uma DLAMI

Tópicos

As seções a seguir descrevem como a AMI de deep learning com Conda pode ser usada para alternar ambientes, executar o código de exemplo para cada uma das estruturas e executar o Jupyter para que você possa testar diferentes tutoriais de notebook.

Uso da AMI de deep learning com o Conda

Tópicos

Introdução à AMI de deep learning com Conda

O Conda é um sistema de gerenciamento de pacotes e de ambientes de código aberto que é executado no Windows, macOS e Linux. O Conda instala, executa e atualiza pacotes e suas dependências rapidamente. O Conda cria, salva, carrega e alterna facilmente entre ambientes em seu computador local.

A AMI de deep learning com Conda foi configurada para que você alterne facilmente entre ambientes de aprendizado profundo. As instruções a seguir orientam você na execução de alguns comandos básicos com `conda`. Elas também ajudam você a verificar se a importação básica da estrutura está

funcionando e se você pode executar algumas operações simples com a estrutura. Em seguida, você pode continuar com tutoriais mais completos fornecidos com a DLAMI ou os exemplos de estruturas encontrados em cada site de projeto das estruturas.

Faça login na DLAMI

Após fazer login em seu servidor, você verá uma "mensagem do dia" (MOTD) do servidor que descreve vários comandos do Conda que você pode usar para alternar entre as diferentes estruturas de aprendizado profundo. Abaixo está um exemplo de MOTD. Seu MOTD específico pode variar conforme novas versões da DLAMI são lançadas.

> **Note**
>
> Não incluímos mais os ambientes CNTK, Caffe, Caffe2, Theano, Chainer e Keras Conda no início da versão 28. AWS Deep Learning AMI As versões anteriores do AWS Deep Learning AMI que contêm esses ambientes continuarão disponíveis. No entanto, só forneceremos atualizações para esses ambientes se houver correções de segurança publicadas pela comunidade de código aberto dessas estruturas de trabalho.

```
=============================================================================
      __|  __|_  )
      _|  (     /   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 comando do Conda tem o seguinte padrão:

`source activate framework_python-version`

Por exemplo, você pode ver `for MXNet(+Keras1) with Python3 (CUDA 10.1) _____ source activate mxnet_p36`, o que significa que o ambiente tem o MXNet, o Keras 1, o Python 3 e o CUDA 10.1. E para ativar esse ambiente, o comando que você deve usar é:

```
$ source activate mxnet_p36
```

Inicie o TensorFlow ambiente

> **Note**
>
> Quando iniciar seu primeiro ambiente Conda, aguarde enquanto ele carrega. A AMI de deep learning com Conda instala automaticamente a versão mais otimizada da estrutura para sua instância do EC2 após a primeira ativação da estrutura. Não deve haver atrasos subsequentes.

1. Ative o ambiente TensorFlow virtual para Python 3.

   ```
   $ source activate tensorflow_p37
   ```

2. Inicie o terminal iPython.

```
(tensorflow_37)$ ipython
```

3. Execute um TensorFlow programa rápido.

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

Você deve ver "Hello, Tensorflow!"

A seguir

Executar os tutoriais do notebook Jupyter

Mude para o PyTorch ambiente Python 3

Se você ainda estiver no console do iPython, use `quit()` e, em seguida, prepare-se para alternar os ambientes.

- Ative o ambiente PyTorch virtual para Python 3.

```
$ source activate pytorch_p36
```

Teste alguns PyTorch códigos

Para testar sua instalação, use o Python para escrever um PyTorch código que cria e imprime uma matriz.

1. Inicie o terminal iPython.

```
(pytorch_p36)$ ipython
```

2. Importar PyTorch.

```
import torch
```

Você pode ver uma mensagem de aviso sobre um pacote de terceiros. Você pode ignorá-lo.

3. Crie uma matriz 5x3 com elementos inicializados de modo aleatório. Imprima a matriz.

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

Verifique o 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]])
```

Mudar para o ambiente do MXNet Python 3

Se você ainda estiver no console do iPython, use `quit()` e, em seguida, prepare-se para alternar os ambientes.

- Ative o ambiente virtual do MXNet para Python 3.

```
$ source activate mxnet_p36
```

Testar um código do MXNet

Para testar a instalação, use o Python para gravar código do MXNet que cria e imprime uma matriz usando a API `NDArray`. Para obter mais informações, consulte a API NDArray.

1. Inicie o terminal iPython.

```
(mxnet_p36)$ ipython
```

2. Importe o MXNet.

```
import mxnet as mx
```

Você pode ver uma mensagem de aviso sobre um pacote de terceiros. Você pode ignorá-lo.

3. Crie uma matriz 5x5, uma instância do `NDArray` com elementos inicializados como 0. Imprima a matriz.

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

Verifique o 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)
```

Você pode localizar mais exemplos do MXNet na seção de tutoriais do MXNet.

Remoção de ambientes

Se ficar sem espaço na DLAMI, você poderá optar por desinstalar pacotes do Conda que não estiver usando:

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

Uso da AMI base de deep learning

Uso da AMI base de deep learning

A AMI Base é fornecida com uma plataforma base de drivers de GPU e bibliotecas de aceleração para implantar seu próprio ambiente personalizado de aprendizado profundo. Por padrão, a AMI é configurada com o ambiente NVIDIA CUDA 11.0. Você também pode alternar entre diferentes versões do CUDA. Consulte as seguintes instruções para saber como fazer isso.

Configurar as versões do CUDA

É possível verificar a versão do CUDA ao executar o programa `nvcc` da NVIDIA.

```
nvcc --version
```

Você pode selecionar e verificar uma determinada versão do CUDA com o comando bash a seguir:

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

Para obter mais informações, consulte as notas de versão da DLAMI base.

Executar os tutoriais do notebook Jupyter

Tutoriais e exemplos são fornecidos com cada origem dos projetos de aprendizado profundo e, na maioria dos casos, eles são executados em qualquer DLAMI. Se você escolher a AMI de deep learning com Conda, obterá o benefício adicional de alguns tutoriais escolhidos a dedo já configurados e prontos para serem testados.

> **Important**
>
> Para executar os tutoriais instalados no caderno Jupyter na DLAMI, você precisará Configurar um servidor do caderno Jupyter.

Assim que o servidor do Jupyter estiver em execução, você poderá executar os tutoriais por meio de seu navegador da web. Se estiver executando a AMI de deep learning com Conda ou tiver configurado ambientes do Python, será possível alternar os kernels do Python na interface do caderno Jupyter. Selecione o kernel apropriado antes de tentar executar um tutorial específico à estrutura. Exemplos adicionais são fornecidos para os usuários da AMI de deep learning com Conda.

> **Note**
>
> Muitos tutoriais exigem módulos adicionais do Python que podem não estar configurados na DLAMI. Se você receber um erro, como `"xyz module not found"`, faça login na DLAMI, ative o ambiente conforme descrito acima e, em seguida, instale os módulos necessários.

> **Tip**
>
> Com frequência, os tutoriais e exemplos de aprendizado profundo dependem de uma ou mais GPUs. Se seu tipo de instância não tiver uma GPU, talvez seja necessário alterar um pouco o código de exemplo para que ele seja executado.

Navegar pelos tutoriais instalados

Depois de fazer login no servidor do Jupyter e poder ver o diretório dos tutoriais (somente na AMI de deep learning com Conda), serão apresentadas pastas de tutoriais para cada nome de estrutura. Se você não vir uma estrutura listada, os tutoriais não estarão disponíveis para essa estrutura em sua DLAMI atual. Clique no nome da estrutura para ver os tutoriais listados e, em seguida, clique em um tutorial para iniciá-lo.

Na primeira vez que executa um caderno na AMI de deep learning com Conda, você precisará informar qual ambiente deseja usar. Você poderá selecionar de uma lista. Cada ambiente é denominado de acordo com este padrão:

`Environment (conda_framework_python-version)`

Por exemplo, você pode ver `Environment (conda_mxnet_p36)`, o que significa que o ambiente tem o MXNet e o Python 3. A outra variação disso seria `Environment (conda_mxnet_p27)`, o que significa que o ambiente tem o MXNet e o Python 2.

> **Tip**
>
> Se você está preocupado sobre qual versão do CUDA está ativa, uma maneira de ver isso é no MOTD, quando você faz login pela primeira vez na DLAMI.

Alternar ambientes com o Jupyter

Se você decidir experimentar um tutorial para uma estrutura diferente, certifique-se de verificar o kernel em execução. Essas informações podem ser vistas no canto superior direito da interface do Jupyter, logo abaixo do botão de logout. Você pode alterar o kernel em qualquer notebook clicando no item de menu Kernel do Jupyter, em seguida, em Change Kernel e, em seguida, clicando no ambiente que atende ao notebook que você está executando.

Neste ponto, você precisará executar novamente todas as células porque uma alteração no kernel apagará o estado de qualquer coisa que tenha executado anteriormente.

> **Tip**
>
> Alternar entre as estruturas pode ser divertido e instrutivo, no entanto, você pode esgotar a memória. Se você começar a receber erros, examine a janela de terminal que tem o servidor

do Jupyter em execução. Há mensagens úteis e registros de erros aqui, e você pode ver um out-of-memory erro. Para corrigir isso, você pode ir para a página inicial do servidor do Jupyter, clicar na guia Running e, em seguida, em Shutdown para cada um dos tutoriais que provavelmente ainda estão em execução em segundo plano e consumindo toda a sua memória.

A seguir

Para obter mais exemplos e código de exemplo de cada estrutura, clique em Next ou continue para Apache MXNet (em incubação).

Tutoriais

Veja a seguir os tutoriais sobre como usar o software da AMI de deep learning com Conda.

Tópicos

- Tutoriais de 10 minutos
- Ativar estruturas
- Depuração e visualização
- Treinamento distribuído
- Elastic Fabric Adapter
- Monitoramento e otimização de GPU
- O chip de AWS inferência com DLAMI
- A DLAMI do Graviton
- A DLAMI Habana
- Inferência
- Como usar Estruturas de acesso com ONNX
- Fornecimento de modelos

Tutoriais de 10 minutos

- Lance um AWS Deep Learning AMI (em 10 minutos)

- Treine um modelo de aprendizado profundo com DLC no Amazon EC2 (em 10 minutos)

Ativar estruturas

Veja a seguir as estruturas de deep learning instaladas na AMI de deep learning com Conda. Clique em uma estrutura para saber como ativá-la.

Tópicos

- Apache MXNet (em incubação)
- Caffe2
- Chainer
- CNTK
- Keras
- PyTorch
- TensorFlow
- TensorFlow 2
- TensorFlow com Horovod
- TensorFlow 2 com Horovod
- Theano

Apache MXNet (em incubação)

Ativação do Apache MXNet (incubando)

Este tutorial mostra como ativar o MXNet em uma instância executando a AMI de deep learning com Conda (DLAMI no Conda) e um programa do MXNet.

Quando um pacote Conda estável de uma estrutura é lançada, ele é testado e pré-instalado na DLAMI. Se desejar executar as mais recentes compilações noturnas não testadas, você pode Como instalar a compilação noturna do MXNet (experimental) manualmente.

Para executar o MXNet na DLAMI com Conda

1. Para ativar a estrutura, abra uma instância da DLAMI do Amazon Elastic Compute Cloud (Amazon EC2) da DLAMI com o Conda.
   - Para MXNet e Keras 2 no Python 3 com CUDA 9.0 e MKL-DNN, execute este comando:

```
$ source activate mxnet_p36
```

- Para MXNet e Keras 2 no Python 2 com CUDA 9.0 e MKL-DNN, execute este comando:

```
$ source activate mxnet_p27
```

2. Inicie o terminal iPython.

```
(mxnet_p36)$ ipython
```

3. Execute um programa rápido do MXNet. Crie uma matriz 5x5, uma instância do `NDArray` com elementos inicializados como 0. Imprima a matriz.

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

4. Verifique o 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)
```

Como instalar a compilação noturna do MXNet (experimental)

Você pode instalar a versão mais recente do MXNet construída em um ou ambos os ambientes MXNet Conda na AMI de deep learning com Conda.

Para Instalar o MXNet a partir de uma compilação noturna

1. • Para o ambiente Pyhton 3 do MXNet, execute este comando:

```
$ source activate mxnet_p36
```

- Para o ambiente Pyhton 2 do MXNet, execute este comando:

```
$ source activate mxnet_p27
```

2. Remova o MXNet instalado.

> **Note**
>
> As etapas restantes assumem que você está usando o ambiente do `mxnet_p36`.

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

3. Instale a última compilação noturna do MXNet.

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

4. Para verificar se você instalou com sucesso a versão mais recente da compilação noturna, inicie o terminal IPython e verifique a versão do MXNet.

```
(mxnet_p36)$ ipython
```

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

A saída deve imprimir a versão estável mais recente do MXNet.

Mais tutoriais

Você pode encontrar mais tutoriais na respectiva pasta da AMI de deep learning com Conda, localizada no diretório inicial da DLAMI.

1. Use o Apache MXNet (incubação) para inferência com um modelo 5.0 ResNet
2. Usar o Apache MXNet (em incubação) para inferência com um modelo ONNX
3. Model Server para Apache MXNet (MMS)

Para ver mais tutoriais e exemplos, consulte a documentação oficial do Python referentes à estrutura, API Python para MXNet, ou acesse o site do Apache MXNet.

Caffe2

> **Note**
>
> Não incluímos mais os ambientes CNTK, Caffe, Caffe2 e Theano Conda no AWS Deep Learning AMI desde a versão v28. As versões anteriores do AWS Deep Learning AMI que contêm esses ambientes continuarão disponíveis. No entanto, só forneceremos atualizações para esses ambientes se houver correções de segurança publicadas pela comunidade de código aberto dessas estruturas de trabalho.

Tutorial do Caffe2

Para ativar a estrutura, siga estas instruções sobre a AMI de deep learning com Conda.

Há apenas o Python 2 com as opções CUDA 9 e cuDNN 7:

```
$ source activate caffe2_p27
```

Inicie o terminal iPython.

```
(caffe2_p27)$ ipython
```

Execute um programa rápido do Caffe2.

```
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)
```

Você deve ver as matrizes aleatórias iniciais do numpy impressas e as carregadas em um blob do Caffe2. Observe que, após o carregamento, elas são iguais.

Mais tutoriais

Para obter mais tutoriais e exemplos, consulte os documentos oficiais do Python referentes à estrutura, API Python para Caffe2 e o site do Caffe2.

Chainer

> **Note**
>
> Não incluímos mais os ambientes Chainer Conda na AWS Deep Learning AMI desde a versão v28. As versões anteriores do AWS Deep Learning AMI que contêm esses ambientes continuarão disponíveis. No entanto, só forneceremos atualizações para esses ambientes se houver correções de segurança publicadas pela comunidade de código aberto dessas estruturas de trabalho.

O Chainer é uma estrutura flexível baseada em Python para gravar arquiteturas de rede neurais complexas de maneira fácil e intuitiva. O Chainer facilita o uso de instâncias de várias GPUs para treinamento. O Chainer também registra automaticamente os resultados, a perda e a precisão do gráfico e produz uma saída para visualizar a rede neural com um gráfico computacional. Ele está incluído na AMI de deep learning com Conda (DLAMI com Conda).

Ativar o Chainer

1. Conecte-se à instância que executa a AMI de deep learning com o Conda. Consulte a the section called “Seleção de instância” ou a documentação do Amazon EC2 sobre como selecionar ou se conectar a uma instância.
2.
   - Ative o ambiente do Chainer Python 3:

     ```
     $ source activate chainer_p36
     ```

   - Ative o ambiente do Chainer Python 2:

     ```
     $ source activate chainer_p27
     ```

3. Inicie o terminal iPython:

   ```
   (chainer_p36)$ ipython
   ```

4. Teste a importação do Chainer a fim de verificar se ele está funcionando corretamente:

```
import chainer
```

Talvez você veja algumas mensagens de aviso, mas nenhum erro.

Mais informações

- Experimente os tutoriais para o Chainer.
- A pasta de exemplos do `Chainer` que você obteve por download anteriormente contém mais exemplos. Teste-os para ver como eles são executados.
- Para saber mais sobre o Chainer, consulte o Site de documentação do Chainer.

CNTK

> **Note**
>
> Não incluímos mais os ambientes CNTK, Caffe, Caffe2 e Theano Conda no AWS Deep Learning AMI desde a versão v28. As versões anteriores do AWS Deep Learning AMI que contêm esses ambientes continuarão disponíveis. No entanto, só forneceremos atualizações para esses ambientes se houver correções de segurança publicadas pela comunidade de código aberto dessas estruturas de trabalho.

Ativação do CNTK

Este tutorial mostra como ativar o CNTK em uma instância executando a AMI de deep learning com Conda (DLAMI no Conda) e um programa do CNTK.

Quando um pacote Conda estável de uma estrutura é lançada, ele é testado e pré-instalado na DLAMI. Se desejar executar as mais recentes compilações noturnas não testadas, você pode Instale a compilação noturna do CNTK (experimental) manualmente.

Para executar o CNTK na DLAMI com Conda

1. Para ativar o CNTK, abra uma instância da DLAMI do Amazon Elastic Compute Cloud (Amazon EC2) da DLAMI com o Conda.
   - Para o Python 3 com CUDA 9 e cuDNN 7:

```
$ source activate cntk_p36
```

- Para o Python 2 com CUDA 9 e cuDNN 7:

```
$ source activate cntk_p27
```

2. Inicie o terminal iPython.

```
(cntk_p36)$ ipython
```

3. • Se você tiver uma instância de CPU, execute esse programa CNTK rápido.

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

Você deve ver a versão do CNTK e, em seguida, a saída de uma matriz 2x3 de 3s.

- Se você tiver uma instância de GPU, poderá testá-la com o seguinte código de exemplo. Um resultado de `True` é o que você deve esperar se o CNTK puder acessar a GPU.

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

Instale a compilação noturna do CNTK (experimental)

Você pode instalar a versão mais recente do CNTK construída em um ou ambos os ambientes CNTK Conda na AMI de deep learning com Conda.

Para instalar o CTNK a partir de uma compilação noturna

1. • Para CNTK e Keras 2 no Python 3 com o CUDA 9.0 e o MKL-DNN, execute este comando:

```
$ source activate cntk_p36
```

- Para CNTK e Keras 2 no Python 2 com o CUDA 9.0 e o MKL-DNN, execute este comando:

```
$ source activate cntk_p27
```

2. As etapas restantes assumem que você está usando o ambiente do `cntk_p36`. Remova o CNTK atualmente instalado:

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

3. Para instalar a compilação noturna do CNTK, primeiro é preciso encontrar a versão que você quer instalar no site do CNTK.

4.
   - (Opção para instâncias de GPU) — Para instalar a compilação noturna, você pode usar o seguinte, substituindo a compilação desejada:

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

     Substitua o URL no comando anterior pela versão de GPU para o seu ambiente Python atual.

   - (Opção para instâncias de CPU) — Para instalar a compilação noturna, você pode usar o seguinte, substituindo a compilação desejada:

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

     Substitua o URL no comando anterior pela versão de CPU para o seu ambiente Python atual.

5. Para verificar se você instalou com sucesso a versão mais recente da compilação noturna, inicie o terminal IPython e verifique a versão do CNTK.

```
(cntk_p36)$ ipython
```

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

A saída deve imprimir algo semelhante a `2.6-rc0.dev20181015`

Mais tutoriais

Para obter mais tutoriais e exemplos, consulte os documentos oficiais do Python referentes à estrutura, API Python para CNTK ou acesse o site do CNTK.

Keras

Tutorial do Keras

1. Para ativar o framework, use estes comandos na CLI the section called “DLAMI do Conda”.
   - Para o Keras 2 com um back-end do MXNet no Python 3 com CUDA 9 e cuDNN 7:

     ```
     $ source activate mxnet_p36
     ```

   - Para o Keras 2 com um back-end do MXNet no Python 2 com CUDA 9 e cuDNN 7:

     ```
     $ source activate mxnet_p27
     ```

   - Para Keras 2 com um TensorFlow back-end em Python 3 com CUDA 9 com cuDNN 7:

     ```
     $ source activate tensorflow_p36
     ```

   - Para Keras 2 com um TensorFlow back-end em Python 2 com CUDA 9 com cuDNN 7:

     ```
     $ source activate tensorflow_p27
     ```

2. Para testar a importação do Keras a fim de verificar qual back-end está ativado, use estes comandos:

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

   O seguinte deve ser exibido na tela:

   ```
   Using MXNet backend
   ```

   Se o Keras estiver usando TensorFlow, o seguinte será exibido:

   ```
   Using TensorFlow backend
   ```

> **Note**
>
> Se você receber um erro ou se o back-end errado ainda estiver sendo usado, você poderá atualizar a configuração do Keras manualmente. Edite o arquivo `~/.keras/keras.json` e altere a configuração do back-end para `mxnet` ou `tensorflow`.

Mais tutoriais

- Para obter um tutorial de várias GPUs usando o Keras com um back-end do MXNet, tente o Tutorial de treinamento do Keras-MXNet com várias GPUs.
- Você pode encontrar exemplos para Keras com um back-end do MXNet no diretório da `~/examples/keras-mxnet` de deep learning com o Conda.
- Você pode encontrar exemplos de Keras com um TensorFlow back-end no diretório Deep Learning AMI with `~/examples/keras` Conda.
- Para obter mais tutoriais e exemplos, consulte o site do Keras.

PyTorch

Ativando PyTorch

Quando um pacote Conda estável de uma estrutura é lançada, ele é testado e pré-instalado na DLAMI. Se desejar executar as mais recentes compilações noturnas não testadas, você pode PyTorchInstall's Nightly Build (experimental) manualmente.

Para ativar a estrutura instalada no momento, siga estas instruções sobre a AMI de deep learning com Conda.

Para PyTorch Python 3 com CUDA 10 e MKL-DNN, execute este comando:

```
$ source activate pytorch_p36
```

Para PyTorch Python 2 com CUDA 10 e MKL-DNN, execute este comando:

```
$ source activate pytorch_p27
```

Inicie o terminal iPython.

```
(pytorch_p36)$ ipython
```

Execute um PyTorch programa rápido.

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

Você deve ver a matriz aleatória inicial impressa, seu tamanho e a adição de outra matriz aleatória.

PyTorchInstall's Nightly Build (experimental)

Como instalar a PyTorch partir de uma compilação noturna

Você pode instalar a PyTorch versão mais recente em um ou em ambos os ambientes PyTorch Conda em sua AMI de aprendizado profundo com o Conda.

1. • (Opção para Python 3) - Ative o ambiente Python 3: PyTorch

   ```
   $ source activate pytorch_p36
   ```

   • (Opção para Python 2) - Ative o ambiente Python 2: PyTorch

   ```
   $ source activate pytorch_p27
   ```

2. As etapas restantes assumem que você está usando o ambiente do `pytorch_p36`. Remova o atualmente instalado PyTorch:

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

3. • (Opção para instâncias de GPU) - Instale a versão noturna mais recente do CUDA 10.0 PyTorch :

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

   • (Opção para instâncias de CPU) - Instale a versão noturna mais recente de PyTorch para instâncias sem GPUs:

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

4. Para verificar se você instalou com sucesso a última compilação noturna, inicie o terminal IPython e verifique a versão do. PyTorch

   ```
   (pytorch_p36)$ ipython
   ```

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

   A saída deve imprimir algo semelhante a `1.0.0.dev20180922`

5. Para verificar se a compilação PyTorch noturna funciona bem com o exemplo do MNIST, você pode executar um script de teste no repositório de exemplos PyTorch da:

   ```
   (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
   ```

Mais tutoriais

Você pode encontrar mais tutoriais na respectiva pasta da AMI de deep learning com Conda no diretório inicial da DLAMI. Para mais tutoriais e exemplos, consulte os documentos oficiais, a PyTorch documentação e o site da estrutura. PyTorch

- PyTorch Tutorial do ONNX para o MXNet
- PyTorch Tutorial do ONNX para o CNTK

TensorFlow

Ativando TensorFlow

Este tutorial mostra como ativar TensorFlow em uma instância executando a AMI de aprendizado profundo com o Conda (DLAMI no Conda) e executar um programa. TensorFlow

Quando um pacote Conda estável de uma estrutura é lançada, ele é testado e pré-instalado na DLAMI. Se desejar executar as mais recentes compilações noturnas não testadas, você pode TensorFlowInstall's Nightly Build (experimental) manualmente.

Para rodar TensorFlow no DLAMI com Conda

1. Para ativar TensorFlow, abra uma instância do Amazon Elastic Compute Cloud (Amazon EC2) do DLAMI com o Conda.
   - Para TensorFlow e Keras 2 em Python 3 com CUDA 9.0 e MKL-DNN, execute este comando:

     ```
     $ source activate tensorflow_p36
     ```

   - Para TensorFlow e Keras 2 em Python 2 com CUDA 9.0 e MKL-DNN, execute este comando:

     ```
     $ source activate tensorflow_p27
     ```

2. Inicie o terminal iPython:

   ```
   (tensorflow_p36)$ ipython
   ```

3. Execute um TensorFlow programa para verificar se ele está funcionando corretamente:

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

   `Hello, TensorFlow!` deve aparecer na tela.

TensorFlowInstall's Nightly Build (experimental)

Você pode instalar a TensorFlow versão mais recente em um ou em ambos os ambientes TensorFlow Conda em sua AMI de aprendizado profundo com o Conda.

Para instalar a TensorFlow partir de uma compilação noturna

1. • Para o TensorFlow ambiente Python 3, execute o seguinte comando:

```
$ source activate tensorflow_p36
```

   • Para o TensorFlow ambiente Python 2, execute o seguinte comando:

```
$ source activate tensorflow_p27
```

2. Remova o atualmente instalado TensorFlow.

> **Note**
>
> As etapas restantes assumem que você está usando o ambiente do `tensorflow_p36`.

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

3. Instale a versão noturna mais recente do. TensorFlow

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

4. Para verificar se você instalou com sucesso a última compilação noturna, inicie o terminal IPython e verifique a versão do. TensorFlow

```
(tensorflow_p36)$ ipython
```

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

   A saída deve imprimir algo semelhante a `1.12.0-dev20181012`

Mais tutoriais

TensorFlow com Horovod

TensorBoard

TensorFlow Servindo

Para tutoriais, consulte a pasta denominada `Deep Learning AMI with Conda tutorials` no diretório inicial da DLAMI.

Para ver mais tutoriais e exemplos, consulte a TensorFlow documentação da API TensorFlow Python ou acesse o site. TensorFlow

TensorFlow 2

Este tutorial mostra como ativar TensorFlow 2 em uma instância executando a AMI de aprendizado profundo com o Conda (DLAMI no Conda) e executar um programa 2. TensorFlow

Quando um pacote Conda estável de uma estrutura é lançada, ele é testado e pré-instalado na DLAMI. Se desejar executar as mais recentes compilações noturnas não testadas, você pode Instale o Nightly Build do TensorFlow 2 (experimental) manualmente.

Ativando 2 TensorFlow

Para rodar TensorFlow no DLAMI com Conda

1. Para ativar TensorFlow 2, abra uma instância do Amazon Elastic Compute Cloud (Amazon EC2) do DLAMI com o Conda.
2. Para TensorFlow 2 e Keras 2 em Python 3 com CUDA 10.1 e MKL-DNN, execute este comando:

   ```
   $ source activate tensorflow2_p36
   ```

3. Inicie o terminal iPython:

   ```
   (tensorflow2_p36)$ ipython
   ```

4. Execute um programa TensorFlow 2 para verificar se ele está funcionando corretamente:

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

   `Hello, TensorFlow!` deve aparecer na tela.

Instale o Nightly Build do TensorFlow 2 (experimental)

Você pode instalar as TensorFlow 2 versões mais recentes em um ou em ambos os ambientes Conda em sua AMI de aprendizado profundo com o Conda. TensorFlow

Para instalar a TensorFlow partir de uma compilação noturna

1. Para o ambiente Python 3 TensorFlow 2, execute o seguinte comando:

```
$ source activate tensorflow2_p36
```

2. Remova o atualmente instalado TensorFlow.

> **Note**
>
> As etapas restantes assumem que você está usando o ambiente do `tensorflow2_p36`.

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

3. Instale a versão noturna mais recente do. TensorFlow

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

4. Para verificar se você instalou com sucesso a última compilação noturna, inicie o terminal IPython e verifique a versão do. TensorFlow

```
(tensorflow2_p36)$ ipython
```

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

A saída deve imprimir algo semelhante a `2.1.0-dev20191122`

Mais tutoriais

Para tutoriais, consulte a pasta denominada `Deep Learning AMI with Conda tutorials` no diretório inicial da DLAMI.

Para ver mais tutoriais e exemplos, consulte a TensorFlow documentação da API TensorFlow Python ou acesse o site. TensorFlow

TensorFlow com Horovod

Este tutorial mostra como ativar TensorFlow com Horovod em um (AWS Deep Learning AMI DLAMI) com Conda. O Horovod é pré-instalado nos ambientes Conda para. TensorFlow O ambiente Python3 é recomendado.

> **Note**
>
> Apenas os tipos de instância P3.*, P2.* e G3.* são compatíveis.

Para ativar TensorFlow e testar o Horovod no DLAMI com o Conda

1. Abra uma instância da DLAMI do Amazon Elastic Compute Cloud (Amazon EC2) da DLAMI com o Conda. Para obter ajuda para começar a trabalhar com uma DLAMI, consulte the section called “Como começar a usar a DLAMI”.
2. (Recomendado) Para TensorFlow 1.15 com Horovod em Python 3 com CUDA 11, execute o seguinte comando:

   ```
   $ source activate tensorflow_p37
   ```

3. Inicie o terminal iPython:

   ```
   (tensorflow_p37)$ ipython
   ```

4. Teste a importação TensorFlow com o Horovod para verificar se está funcionando corretamente:

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

   O seguinte pode ser exibido na tela (você pode ignorar todas as mensagens de aviso).

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

Mais informações

- TensorFlow com Horovod
- Para tutoriais, consulte a pasta `examples/horovod` no diretório inicial da DLAMI.
- Para ver ainda mais tutoriais e exemplos, consulte o projeto GitHub Horovod.

TensorFlow 2 com Horovod

Este tutorial mostra como ativar TensorFlow 2 com Horovod em um (AWS Deep Learning AMI DLAMI) com Conda. O Horovod é pré-instalado nos ambientes Conda para 2. TensorFlow O ambiente Python3 é recomendado.

> **Note**
>
> Apenas os tipos de instância P3.*, P2.* e G3.* são compatíveis.

Para ativar TensorFlow 2 e testar Horovod no DLAMI com Conda

1. Abra uma instância da DLAMI do Amazon Elastic Compute Cloud (Amazon EC2) da DLAMI com o Conda. Para obter ajuda para começar a trabalhar com uma DLAMI, consulte the section called “Como começar a usar a DLAMI”.
   - (Recomendado) Para TensorFlow 2 com Horovod em Python 3 com CUDA 10, execute este comando:

     ```
     $ source activate tensorflow2_p36
     ```

- Para TensorFlow 2 com Horovod em Python 2 com CUDA 10, execute este comando:

```
$ source activate tensorflow2_p27
```

2. Inicie o terminal iPython:

```
(tensorflow2_p36)$ ipython
```

3. Teste a importação de TensorFlow 2 com o Horovod para verificar se está funcionando corretamente:

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

Se não há nenhuma saída, então, o Horovod está funcionando corretamente. O seguinte pode ser exibido na tela (você pode ignorar todas as mensagens de aviso).

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

Mais informações

- Para tutoriais, consulte a pasta `examples/horovod` no diretório inicial da DLAMI.
- Para ver ainda mais tutoriais e exemplos, consulte o projeto GitHub Horovod.

Theano

Tutorial do Theano

> **Note**
>
> Não incluímos mais os ambientes CNTK, Caffe, Caffe2 e Theano Conda no AWS Deep Learning AMI desde a versão v28. As versões anteriores do AWS Deep Learning AMI que contêm esses ambientes continuarão disponíveis. No entanto, só forneceremos atualizações para esses ambientes se houver correções de segurança publicadas pela comunidade de código aberto dessas estruturas de trabalho.

Para ativar a estrutura, siga estas instruções sobre a AMI de deep learning com Conda.

Para o Theano + Keras no Python 3 com CUDA 9 e cuDNN 7:

```
$ source activate theano_p36
```

Para o Theano + Keras no Python 2 com CUDA 9 e cuDNN 7:

```
$ source activate theano_p27
```

Inicie o terminal iPython.

```
(theano_p36)$ ipython
```

Execute um programa rápido do 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)
```

Você deve ver um Theano calculando um gradiente simbólico.

Mais tutoriais

Para obter mais tutoriais e exemplos, consulte os documentos oficiais da estrutura API do Python Theano e o site do Theano.

Depuração e visualização

Conheça as opções de visualização e depuração para a DLAMI. Clique em uma das opções para saber como usá-la.

Tópicos

- MXBoard
- TensorBoard

MXBoard

O MXBoard permite que você inspecione e interprete visualmente suas execuções e gráficos do MXNet usando o software. TensorBoard Ele executa um servidor web que fornece uma página da web para exibir e interagir com as visualizações do MXBoard.

O MXNet, TensorBoard, e o MXBoard são pré-instalados com a AMI de aprendizado profundo com Conda (DLAMI com Conda). Neste tutorial, você usa uma função MxBoard para gerar registros compatíveis com o. TensorBoard

Tópicos

- Como usar o MXNet com MXBoard
- Mais informações

Como usar o MXNet com MXBoard

Gere dados de registro do MxBoard compatíveis com TensorBoard

1. Conecte uma instância do Amazon Elastic Compute Cloud (Amazon EC2) da DLAMI com o Conda.
2. Ative o ambiente do MXNet Python 3.

```
$ source activate mxnet_p36
```

3. Prepare um script Python para gravar dados gerados pelo operador para um evento de arquivo normal. Os dados são gerados dez vezes com desvio padrão decrescente, depois gravados para o arquivo de evento um por vez. Você verá a distribuição de dados se tornarem gradualmente mais centralizadas em torno do valor médio. Observe que você especificará o arquivo de evento na pasta de logs. Você passa esse caminho de pasta para o TensorBoard binário.

```
$ vi mxboard_normal.py
```

4. Cole o seguinte código no arquivo e salve-o:

```
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. Executar o script. Isso gera logs em uma pasta `logs` que você pode usar para visualizações.

```
$ python mxboard_normal.py
```

6. Agora você deve alternar para o TensorFlow ambiente a ser usado TensorBoard e para o MxBoard para visualizar os registros. Essa é uma dependência necessária para o MxBoard e. TensorBoard

```
$ source activate tensorflow_p36
```

7. Passe a localização dos logs para o `tensorboard`:

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

TensorBoard inicia o servidor web de visualização na porta 8888.

8. Para facilitar o acesso do navegador local, você pode alterar a porta do servidor web para a porta 80 ou outra. Qualquer que seja a porta usada, você precisará abrir essa porta no grupo de segurança do EC2 para a DLAMI. Você também pode usar o encaminhamento de porta. Para instruções sobre alterar suas configurações do grupo de segurança e encaminhamento de porta, consulte Configurar um servidor do caderno Jupyter. As configurações padrão são descritas na próxima etapa.

   > **Note**
   >
   > Se deseja executar o servidor Jupyter e um servidor MXBoard, use uma porta diferente para cada um deles.

9. Abra a porta 8888 (ou a porta atribuída ao servidor web de visualização) na instância do EC2.

   a. Abra a instância do EC2 no console do Amazon EC2 em https://console.aws.amazon.com/ec2/.

   b. No console do Amazon EC2, escolha Rede e segurança e, em seguida, escolha Grupos de segurança.

   c. Para Security Group (Grupo de segurança), escolha o que foi criado mais recentemente, consulte a data e a hora na descrição.

   d. Escolha a guia Inbound (Entrada) e Edit (Editar).

   e. Escolha Add Rule.

   f. Na nova linha, digite o seguinte:

      Tipo : Personalizado **`TCP Rule`**

      Protocolo: **`TCP`**

      Port Range (Intervalo de portas): **`8888`** (ou a porta que você atribuiu ao servidor de visualização)

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

10. Se você deseja visualizar os dados do navegador local, digite o comando a seguir para encaminhar os dados que são renderizados na instância do EC2 para sua máquina local.

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

11. Abra a página da web para as visualizações do MXBoard usando o endereço IP ou DNS público da instância do EC2 que está executando a DLAMI com Conda e a porta que você abriu para o MXBoard:

    `http://127.0.0.1:8888`

Mais informações

Para saber mais sobre o MXBoard, consulte site do MXBoard.

TensorBoard

TensorBoardpermite que você inspecione e interprete visualmente seus TensorFlow ensaios e gráficos. Ele executa um servidor web que serve uma página da web para visualizar e interagir com as TensorBoard visualizações.

TensorFlow e TensorBoard são pré-instalados com a AMI de aprendizado profundo com Conda (DLAMI com Conda). O DLAMI com Conda também inclui um script de exemplo TensorFlow usado para treinar um modelo MNIST com recursos extras de registro habilitados. O MNIST é um banco de dados de números manuscritos que é comumente usado para treinar modelos de reconhecimento de imagem. Neste tutorial, você usa o script para treinar um modelo MNIST TensorBoard e os registros para criar visualizações.

Tópicos

- Treine um modelo MNIST e visualize o treinamento com TensorBoard
- Mais informações

Treine um modelo MNIST e visualize o treinamento com TensorBoard

Visualize o treinamento do modelo MNIST com TensorBoard

1. Conecte uma instância do Amazon Elastic Compute Cloud (Amazon EC2) da DLAMI com o Conda.
2. Ative o TensorFlow ambiente Python 2.7 e navegue até o diretório que contém a pasta com os scripts de TensorBoard exemplo:

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

3. Execute o script que treina um modelo MNIST com o log expandido habilitado:

```
$ python mnist_with_summaries.py
```

   O script grava os logs em `/tmp/tensorflow/mnist`.

4. Passe a localização dos logs para o `tensorboard`:

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

   TensorBoard inicia o servidor web de visualização na porta 6006.

5. Para facilitar o acesso do navegador local, você pode alterar a porta do servidor web para a porta 80 ou outra. Qualquer que seja a porta usada, você precisará abrir essa porta no grupo de segurança do EC2 para a DLAMI. Você também pode usar o encaminhamento de porta. Para instruções sobre alterar suas configurações do grupo de segurança e encaminhamento de porta, consulte Configurar um servidor do caderno Jupyter. As configurações padrão são descritas na próxima etapa.

   > **Note**
   >
   > Se você precisar executar o servidor Jupyter e um TensorBoard servidor, use uma porta diferente para cada um.

6. Abra a porta 6006 (ou a porta atribuída ao servidor web de visualização) na instância do EC2.

   a. Abra a instância do EC2 no console do Amazon EC2 em https://console.aws.amazon.com/ec2/.

   b. No console do Amazon EC2, escolha Rede e segurança e, em seguida, escolha Grupos de segurança.

   c. Em Security Group, escolha o que foi criado mais recentemente (consulte o time stamp na descrição).

   d. Escolha a guia Inbound (Entrada) e Edit (Editar).

   e. Escolha Add Rule.

   f. Na nova linha, digite o seguinte:

      Tipo : Personalizado **`TCP Rule`**

      Protocolo: **`TCP`**

Port Range (Intervalo de portas): **`6006`** (ou a porta que você atribuiu ao servidor de visualização)

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

7. Abra a página da web para TensorBoard as visualizações usando o endereço IP ou DNS público da instância EC2 que está executando o DLAMI com o Conda e a porta para a qual você abriu: TensorBoard

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

Mais informações

Para saber mais sobre isso TensorBoard, consulte o TensorBoardsite.

Treinamento distribuído

Conheça as opções que a DLAMI tem para treinamento com várias GPUs. Para obter mais desempenho, consulte Elastic Fabric Adapter Clique em uma das opções para saber como usá-la.

Tópicos

- Chainer
- Keras com MXNet
- TensorFlow com Horovod

Chainer

> **Note**
>
> Não incluímos mais os ambientes Chainer Conda na AWS Deep Learning AMI desde a versão v28. As versões anteriores do AWS Deep Learning AMI que contêm esses ambientes continuarão disponíveis. No entanto, só forneceremos atualizações para esses ambientes se houver correções de segurança publicadas pela comunidade de código aberto dessas estruturas de trabalho.

O Chainer é uma estrutura flexível baseada em Python para gravar arquiteturas de rede neurais complexas de maneira fácil e intuitiva. O Chainer facilita o uso de instâncias de várias GPUs para

treinamento. O Chainer também registra automaticamente os resultados, a perda e a precisão do gráfico e produz uma saída para visualizar a rede neural com um gráfico computacional. Ele está incluído na AMI de deep learning com Conda (DLAMI com Conda).

Os tópicos a seguir mostram como treinar em várias GPUs, uma única GPU e uma CPU, criar visualizações e testar a instalação do Chainer.

Tópicos

Treinar um modelo com o Chainer

Este tutorial mostra como usar scripts de exemplo do Chainer para treinar um modelo com o conjunto de dados do MNIST. O MNIST é um banco de dados de números manuscritos que é comumente usado para treinar modelos de reconhecimento de imagem. O tutorial também mostra a diferença na velocidade de treinamento entre o treinamento em uma CPU e em uma ou mais GPUs.

Use o Chainer para treinar em várias GPUs

Para treinar em várias GPUs

1. Conecte-se à instância que executa a AMI de deep learning com o Conda. Consulte a the section called “Seleção de instância” ou a documentação do Amazon EC2 sobre como selecionar ou se conectar a uma instância. Para executar este tutorial, você desejará usar uma instância com, pelo menos, duas GPUs.
2. Ative o ambiente do Chainer Python 3:

   ```
   $ source activate chainer_p36
   ```

3. Para obter os tutoriais mais recentes, clone o repositório do Chainer, e navegue até a pasta de exemplos:

```
(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. Execute o exemplo no script `train_mnist_data_parallel.py`. Por padrão, o script usa as GPUs em execução na instância da AMI de deep learning com Conda. O script pode ser executado em duas GPUs no máximo. Ele ignora qualquer GPU após as duas primeiras. Ele detecta uma ou as duas automaticamente. Se você estiver executando uma instância sem GPUs, vá para Usar o Chainer para treinar com CPUs, mais adiante neste tutorial.

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

> **Note**
>
> Este exemplo retornará o seguinte erro devido à inclusão de um recurso beta não incluído na DLAMI.
> `chainerx ModuleNotFoundError: No module named 'chainerx'`

Enquanto o script do Chainer treina um modelo usando o banco de dados do MNIST, você verá os resultados para cada epoch.

Em seguida, você verá o exemplo de saída enquanto o script é executado. O exemplo de saída a seguir foi executado em uma instância p3.8xlarge. A saída do script mostra "GPU: 0, 1", que indica que está usando as duas primeiras das quatro GPUs disponíveis. Normalmente, os scripts usam um índice de GPUs começando com zero, em vez de uma contagem 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. Enquanto o treinamento está sendo executado, é útil examinar a utilização da GPU. Você pode verificar quais GPUs estão ativas e visualizar suas cargas. O NVIDIA fornece uma ferramenta

para isso, que pode ser executada com o comando `nvidia-smi`. No entanto, ela apenas mostra um snapshot da utilização, portanto, é mais informativo combiná-la com o comando `watch` do Linux. O comando a seguir usa `watch` com `nvidia-smi` para atualizar a utilização da GPU atual a cada décimo de segundo. Abra uma sessão de terminal para a DLAMI e execute o comando a seguir:

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

Você verá uma saída semelhante ao resultado a seguir. Use `ctrl-c` para fechar a ferramenta ou apenas para mantê-la em execução enquanto você testa outros exemplos na primeira sessão 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 |
```

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

Neste exemplo, a GPU 0 e a GPU 1 estão ativas e a GPU 2 e 3 não estão. Também é possível ver a utilização da memória por GPU.

6. Conforme o treinamento é concluído, observe o tempo decorrido na primeira sessão de terminal. No exemplo, o tempo decorrido é de 38,1389 segundos.

Use o Chainer para treinar em uma única GPU

Este exemplo mostra como treinar em uma única GPU. Você pode fazer isso se tiver apenas uma GPU disponível ou apenas para ver como o treinamento de várias GPUs pode ser escalado com o Chainer.

Para usar o Chainer para treinar em uma única GPU

- Para este exemplo, você usa outro script, `train_mnist.py`, e indica que ele use apenas a GPU 0 com o argumento `--gpu=0`. Para ver como GPUs diferentes são ativadas no console `nvidia-smi`, você pode indicar que o script use a GPU número 1 usando `--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
```

Neste exemplo, a execução em uma única GPU levou quase o dobro do tempo! Treinar modelos maiores ou conjuntos de dados maiores produz resultados diferentes deste exemplo, portanto, faça testes para avaliar melhor o desempenho da GPU.

Usar o Chainer para treinar com CPUs

Agora, tente o treinamento em um modo somente CPU. Execute o mesmo script, `python train_mnist.py`, sem argumentos:

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

Na saída, `GPU: -1` indica que nenhuma GPU está sendo usada:

```
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
```

Neste exemplo, o MNIST foi treinado em 323 segundos, que é um tempo mais de 11x maior que o tempo do treinamento com duas GPUs. Se você alguma vez duvidou do poder das GPUs, este exemplo mostra o quanto elas são mais eficientes.

Resultados de gráficos

O Chainer também registra automaticamente os resultados, a perda e a precisão do gráfico e produz uma saída para plotar o gráfico computacional.

Para gerar o gráfico computacional

1. Depois que a execução de qualquer treinamento for concluída, você pode navegar até o diretório `result` e visualizar a precisão e a perda da execução na forma de duas imagens geradas automaticamente. Navegue até lá agora e liste o conteúdo:

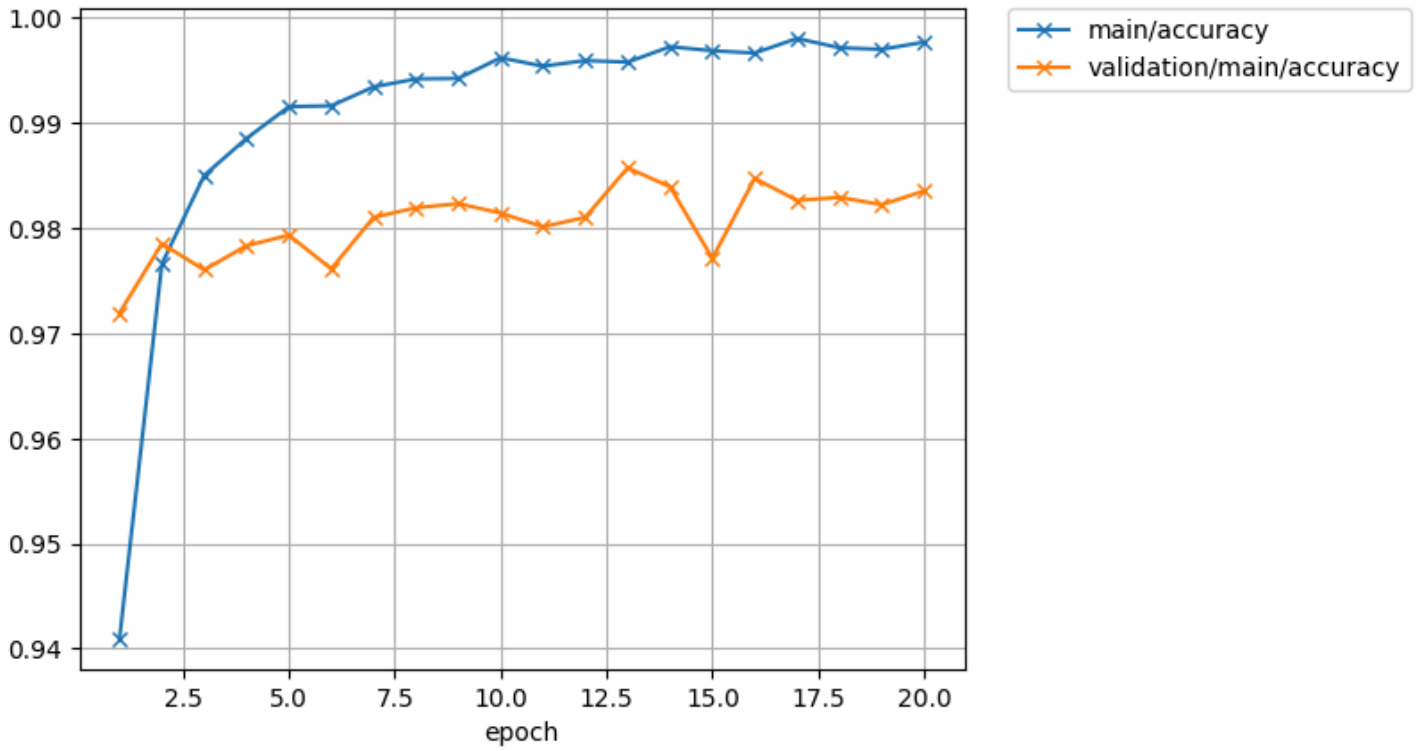
   ```
   (chainer_p36) :~/src/chainer/examples/mnist$ cd result
   (chainer_p36) :~/src/chainer/examples/mnist/result$ ls
   ```

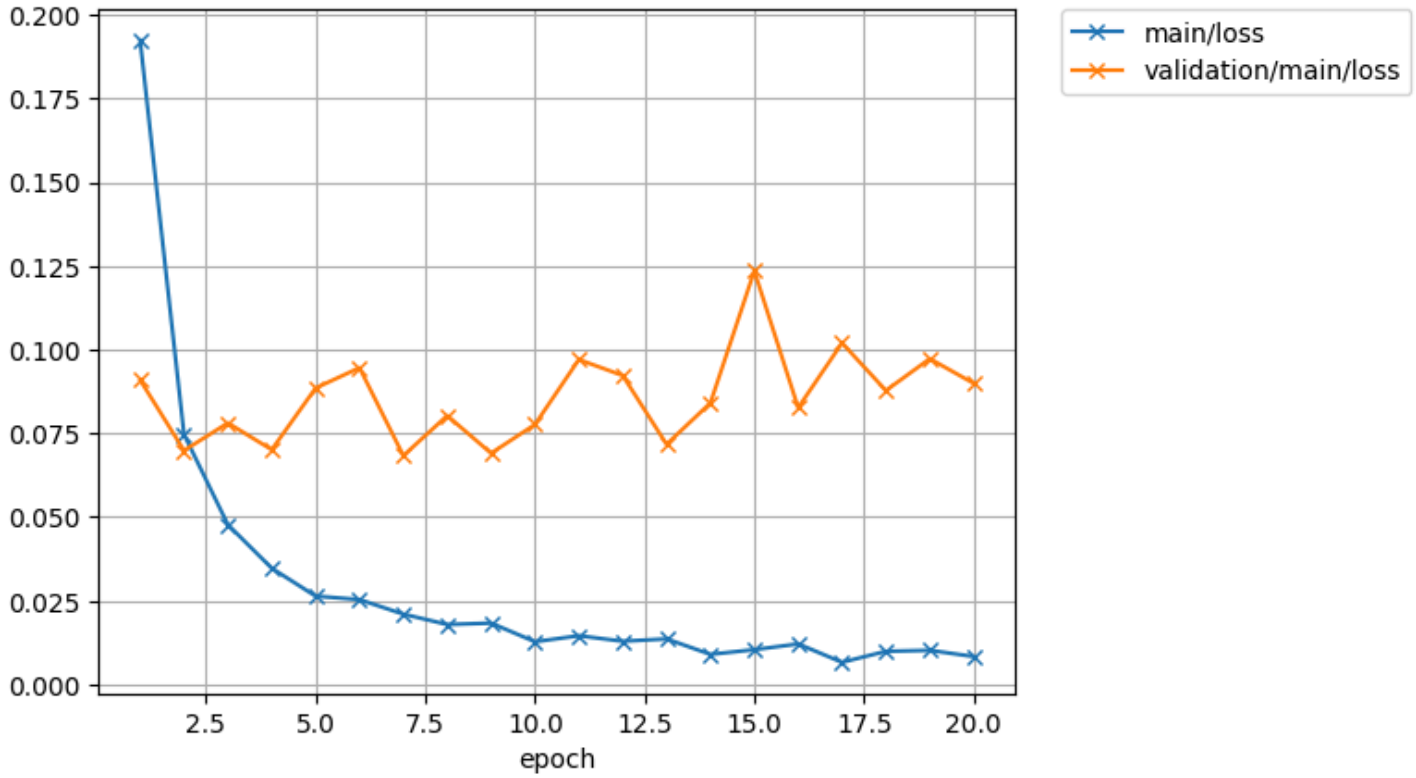
   O diretório `result` contém dois arquivos em formato .png: `accuracy.png` e `loss.png`.
2. Para visualizar os gráficos, use o comando `scp` para copiá-los para o computador local.

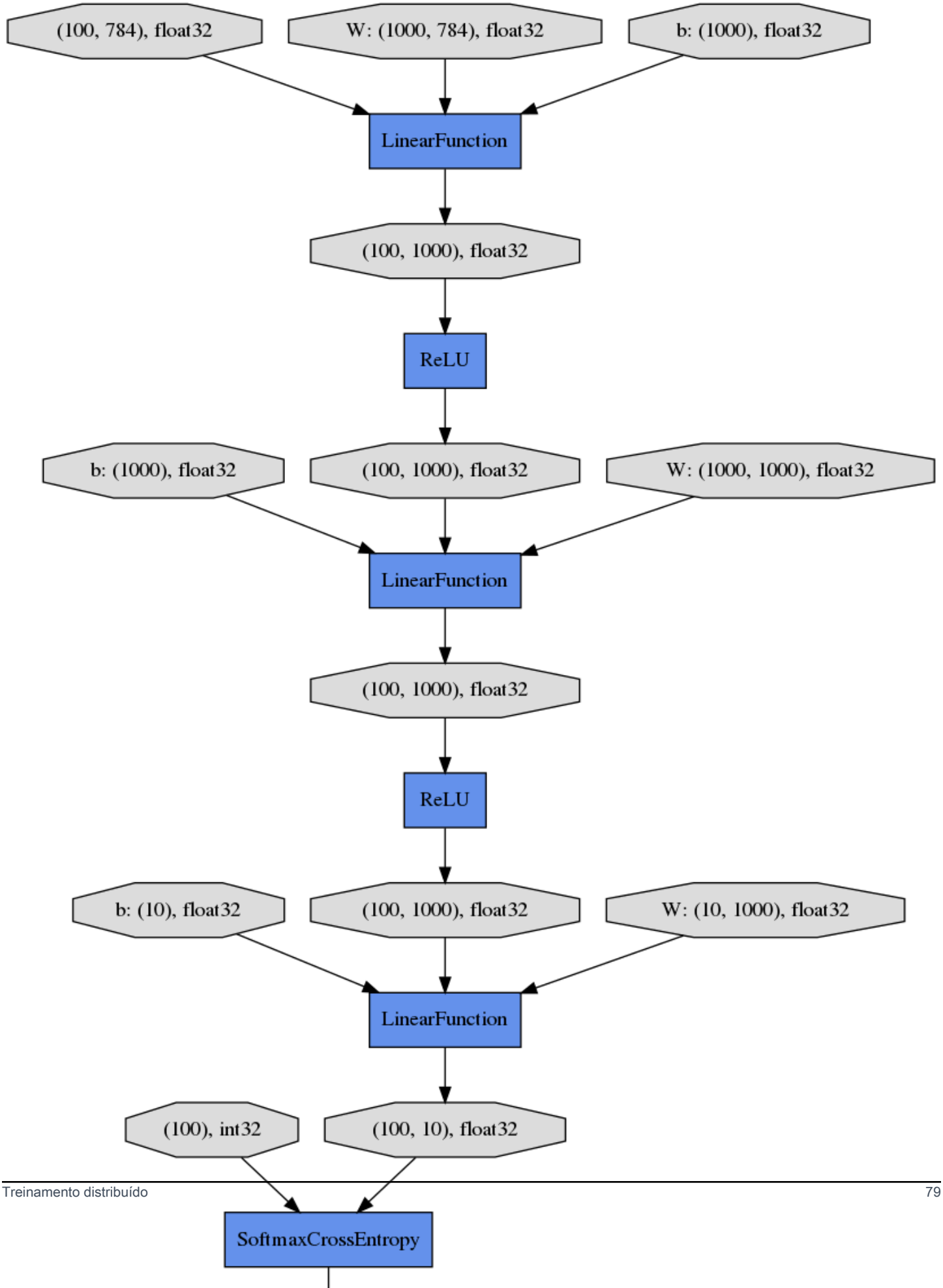
   Em um terminal macOS, a execução do seguinte comando `scp` faz download de todos os três arquivos na pasta `Downloads`. Substitua os espaços reservados do local do arquivo principal e do endereço do servidor com suas informações. Para outros sistemas operacionais, use o formato de comando `scp` adequado. Observe que, para uma AMI do Amazon Linux, o nome do usuário é 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
   ```

As imagens a seguir são exemplos de precisão, perda e gráficos computacionais, respectivamente.


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


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


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

Testar o Chainer

Para testar o Chainer e verificar a GPU compatível com um script de teste pré-instalado, execute o comando a seguir:

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

Isso faz download do código-fonte do Chainer e executa o MNIST de exemplo nas várias GPUs do Chainer.

Mais informações

Para saber mais sobre o Chainer, consulte o Site de documentação do Chainer. A pasta de exemplos `Chainer` contém mais exemplos. Teste-os para ver como eles são executados.

Keras com MXNet

Este tutorial mostra como ativar e usar o Keras 2 com o back-end do MXNet em uma AMI de deep learning com Conda.

Ative o Keras com o back-end do MXNet e teste-o na DLAMI com Conda

1. Para ativar o Keras com o MXNet de back-end, abra uma instância da DLAMI do Amazon Elastic Compute Cloud (Amazon EC2) da DLAMI com o Conda.
   - Para Python 3, execute este comando:

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

     ```
     $ source activate mxnet_p27
     ```
2. Inicie o terminal iPython:

   ```
   (mxnet_p36)$ ipython
   ```
3. Teste a importação do Keras com MXNet a fim de verificar se ela está funcionando corretamente:

```
import keras as k
```

O seguinte deve ser exibido na tela (possivelmente após algumas mensagens de aviso).

```
Using MXNet backend
```

> **Note**
>
> Se você receber um erro ou se o TensorFlow back-end ainda estiver sendo usado, você precisará atualizar a configuração do Keras manualmente. Edite o arquivo `~/.keras/keras.json` e altere a configuração do back-end para `mxnet`.

Tutorial de treinamento do Keras-MXNet com várias GPUs

Treinar uma rede neural convolucional (CNN)

1. Abra um terminal e SSH em sua DLAMI.
2. Navegue para a pasta `~/examples/keras-mxnet/`.
3. Execute `nvidia-smi` na janela do terminal para determinar o número de GPUs disponíveis no seu DLAMI. Na próxima etapa, você irá executar o script no estado em que se encontra se tiver quatro GPUs.
4. (Opcional) Execute o seguinte comando para abrir o script para edição.

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

5. (Opcional) O script tem a seguinte linha que define o número de GPUs. Atualize-o, se necessário.

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

6. Agora, rode o treinamento.

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

> **Note**
>
> Keras-MXNet é executado até duas vezes mais rápido com a definição `channels_first` image_data_format. Para alterar a `channels_first`, edite o arquivo de configuração (Keras`~/.keras/keras.json`) e defina o seguinte: `"image_data_format": "channels_first"`.
> Para obter mais técnicas de ajuste de desempenho, consulte Guia de ajuste de desempenho do Keras-MXNet.

Mais informações

- Você pode encontrar exemplos para Keras com um back-end do MXNet no diretório da `~/examples/keras-mxnet` de deep learning com o Conda.
- Para ver ainda mais tutoriais e exemplos, consulte o projeto GitHub Keras-MXNet.

TensorFlow com Horovod

Este tutorial mostra como usar TensorFlow o Horovod em uma AMI de aprendizado profundo com o Conda. O Horovod é pré-instalado nos ambientes Conda para. TensorFlow O ambiente Python 3 é recomendado. Para seguir estas instruções, é necessário conhecer a instância DLAMI com um ou mais GPUs. Para ter mais informações, consulte Como começar a usar a DLAMI.

> **Note**
>
> Apenas os tipos de instância P3.*, P2.* e G3.* são compatíveis.

> **Note**
>
> Há dois locais onde o mpirun (via OpenMPI) está disponível. Ele está disponível em `/usr/bin` e `/home/ubuntu/anaconda3/envs/<env>/bin`. env é um ambiente que corresponde à estrutura, como o Tensorflow e o Apache MXNet. As versões mais recentes do OpenMPI estão disponíveis nos ambientes conda. Recomendamos usar o caminho absoluto do binário mpirun ou o sinalizador --prefix para executar cargas de trabalho mpi. Por exemplo, com o ambiente python36 do Tensorflow, use:

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

or

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

Ative e teste TensorFlow com Horovod

1. Verifique se sua instância tem GPUs ativas. O NVIDIA fornece uma ferramenta para isso:

   ```
   $ nvidia-smi
   ```

2. Ative o ambiente Python 3: TensorFlow

   ```
   $ source activate tensorflow_p36
   ```

3. Inicie o terminal iPython:

   ```
   (tensorflow_p36)$ ipython
   ```

4. Teste a importação TensorFlow com o Horovod para verificar se está funcionando corretamente:

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

   O seguinte pode ser exibido na tela (possivelmente após algumas mensagens de aviso).

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

Configure o arquivo de hosts do Horovod

Você pode usar o Horovod para o treinamento de vários GPUs e um único nó, ou várias GPUs e vários nós. Se você planeja usar vários nós para treinamento distribuído, adicione cada endereço IP privado da DLAMI a um arquivo de hosts. A DLAMI a qual você está conectado no momento é chamada de líder. Outras instâncias da DLAMI que fazem parte do cluster são chamadas de membros.

Antes de começar esta seção, inicie uma ou mais DLAMIs e aguarde até que todas estejam no estado Pronta. Os scripts de exemplo esperam um arquivo de hosts. Portanto, mesmo que você planeje usar apenas uma DLAMI, crie um arquivo de hosts com apenas uma entrada. Se você editar o arquivo de hosts após o início do treinamento, será necessário reiniciar o treinamento para que tenha efeito a adição ou remoção de hosts.

Para configurar o Horovod para treinamento

1. Altere os diretórios para os locais em que os scripts de treinamento estão.

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

2. Use o vim para editar um arquivo no diretório inicial do líder.

   ```
   vim hosts
   ```

3. Selecione um dos membros no console do Amazon Elastic Compute Cloud, e o painel de descrição do console será exibido. Encontre o campo Private IPs (IPs privados), copie o IP e cole-o em um arquivo de texto. Copie o IP privado de cada membro em uma nova linha. Em seguida, ao lado de cada IP, adicione um espaço e, em seguida, o texto `slots=8` conforme mostrado abaixo. Isso representa quantas GPUs há em cada instância. As instâncias p3.16xlarge têm 8 GPUs. Se você escolher um tipo de instância diferente, deverá fornecer o número real de GPUs para cada instância. Para o líder, você pode usar `localhost`. Com um cluster de 4 nós, ele deve ser semelhante ao seguinte:

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

   Salve o arquivo e volte para o terminal do líder.

4. Adicione a chave SSH usada pelas instâncias membros ao ssh-agent.

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

5. Agora o líder sabe como alcançar cada membro. Isso tudo acontecerá nas interfaces de rede privada. Em seguida, use uma função bash curta para ajudar a enviar comandos a cada membro.

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

6. Diga aos outros membros que não façam “StrickHostKeyChecking”, pois isso pode fazer com que o treinamento pare de responder.

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

Treinamento com dados sintéticos

A DLAMI é fornecida com um script de exemplo para treinar um modelo com dados sintéticos. Ele testa se o líder pode se comunicar com os membros do cluster. Um arquivo de hosts é necessário. Consulte Configure o arquivo de hosts do Horovod para ver as instruções.

Para testar o treinamento do Horovod com dados de exemplo.

1. O `~/examples/horovod/tensorflow/train_synthetic.sh` tem como padrão 8 GPUs, mas você pode fornecer o número de GPUs que quiser executar. O exemplo a seguir executa o script, transmitindo 4 como um parâmetro para 4 GPUs.

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

   Após algumas mensagens de aviso, você verá a saída a seguir que verifica se o Horovod está usando 4 GPUs.

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

Em seguida, depois de alguns outros avisos, você verá o início de uma tabela e alguns pontos de dados. Se você não quiser ver os 1.000 lotes, saia do treinamento.

```
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. O Horovod usa todos os GPUs locais antes de tentar usar os GPUs dos membros do cluster. Portanto, para se certificar de que o treinamento distribuído no cluster esteja funcionando, teste todos os GPUs que você pretende usar. Se, por exemplo, você tiver 4 membros do tipo de instância p3.16xlarge, você terá 32 GPUs no cluster. Este é o local em que você deseja testar os 32 GPUs.

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

   O resultado será semelhante ao do teste anterior. O tamanho do Horovod é 32 e cerca de quatro vezes a velocidade. Com essa experimentação concluída, você testou o seu líder e a capacidade dele de se comunicar com os membros. Se tiver problemas, consulte a seção Solução de problemas.

Prepare o ImageNet conjunto de dados

Nesta seção, você baixa o ImageNet conjunto de dados e gera um conjunto de dados no formato TFRecord a partir do conjunto de dados bruto. Um conjunto de scripts de pré-processamento é fornecido no DLAMI para ImageNet o conjunto de dados que você pode usar para um ImageNet ou como modelo para outro conjunto de dados. Os principais scripts de treinamento configurados também ImageNet são fornecidos. Para acompanhar a seção a seguir, é preciso já ter executado uma DLAMI com uma instância do EC2 com 8 GPUs. Recomendamos o tipo de instância p3.16xlarge.

No diretório `~/examples/horovod/tensorflow/utils` na sua DLAMI, você encontra os seguintes scripts:

- `utils/preprocess_imagenet.py`- Use isso para converter o ImageNet conjunto de dados bruto para o `TFRecord` formato.
- `utils/tensorflow_image_resizer.py`- Use isso para redimensionar o `TFRecord` conjunto de dados conforme recomendado para ImageNet treinamento.

Prepare o ImageNet conjunto de dados

1. Visite image-net.org, crie uma conta, adquira uma chave de acesso e faça download do conjunto de dados. image-net.org hospeda o conjunto de dados bruto. Para baixá-lo, você precisa ter uma ImageNet conta e uma chave de acesso. A conta é gratuita e, para obter a chave de acesso gratuita, você deve concordar com a ImageNet licença.
2. Use o script de pré-processamento de imagem para gerar um conjunto de dados no formato TFRecord a partir do conjunto de dados bruto. ImageNet No diretório `~/examples/horovod/tensorflow/utils`:

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

3. Use o script de redimensionamento de imagem. Se você redimensionar as imagens, o treinamento será executado mais rapidamente e se alinhará melhor ao ResNet papel de referência. No diretório `~/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
```

Treine um ImageNet modelo ResNet -50 em um único DLAMI

> **Note**
>
> - O script neste tutorial espera que os dados de treinamento pré-processados estejam na pasta `~/data/tf-imagenet/`. Consulte Prepare o ImageNet conjunto de dados para ver as instruções.
> - Um arquivo de hosts é necessário. Consulte Configure o arquivo de hosts do Horovod para ver as instruções.

Use o Horovod para treinar uma CNN de ResNet 50 no conjunto de dados ImageNet

1. Navegue para a pasta `~/examples/horovod/tensorflow`.

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

2. Verifique a configuração e defina o número de GPUs para uso no treinamento. Primeiro, revise o `hosts` que está na mesma pasta que os scripts. Esse arquivo deverá ser atualizado se você usar uma instância com menos de 8 GPUs. Por padrão, ele diz `localhost slots=8`. Atualize o número 8 para o número de GPUs que você deseja usar.
3. É fornecido um script de shell que considera o número de GPUs que você planeja usar como o único parâmetro. Execute esse script para iniciar o treinamento. O exemplo a seguir usa 4 para quatro GPUs.

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

4. Pode levar algumas horas para ser concluído. Ele usa `mpirun` para distribuir o treinamento por suas GPUs.

Treine um ImageNet modelo ResNet -50 em um cluster de DLamis

> **Note**
>
> - O script neste tutorial espera que os dados de treinamento pré-processados estejam na pasta `~/data/tf-imagenet/`. Consulte Prepare o ImageNet conjunto de dados para ver as instruções.
> - Um arquivo de hosts é necessário. Consulte Configure o arquivo de hosts do Horovod para ver as instruções.

Este exemplo mostra como treinar um modelo ResNet -50 em um conjunto de dados preparado em vários nós em um cluster de DLamis.

- Para um desempenho mais rápido, recomendamos que você tenha o conjunto de dados localmente em cada membro do cluster.

  Use a função `copyclust` para copiar os dados para outros membros.

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

Ou, se os arquivos estiverem em um bucket do S3, use a função `runclust` para fazer download dos arquivos para cada membro diretamente.

```
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/\""
```

Com as ferramentas para gerenciar vários nós de uma só vez, a economia de tempo é enorme. Você pode aguardar cada etapa e gerenciar cada instância separadamente ou usar ferramentas como tmux ou screen para poder desconectar e retomar sessões.

Depois que a cópia for concluída, você está pronto para iniciar o treinamento. Execute o script, transmitindo 32 como um parâmetro para os 32 GPUs que estamos usando nesta execução. Use o tmux ou uma ferramenta similar se você estiver preocupado com a desconexão e encerramento da sua sessão, que deve encerrar a execução do treinamento.

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

O resultado a seguir é o que você vê ao executar o treinamento ImageNet com 32 GPUs. Trinta e duas GPUs levam de 90 a 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
```

Após concluída uma execução do treinamento, o script prosseguirá com uma execução de avaliação. Ele é executado no líder, pois é executado de maneira tão veloz que não precisa distribuir o trabalho para outros membros. Veja a seguir o resultado da execução de avaliação.

```
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
```

Veja a seguir um exemplo de resultado retornado quando o script é executado com 256 GPUs onde o tempo de execução foi entre 14 e 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
```

Solução de problemas

O comando a seguir pode ajudar a identificar erros passados que surgem quando você testa o Horovod.

- Se o treinamento falhar por algum motivo, o mpirun pode falhar para limpar todos os processos do python em cada máquina. Nesse caso, antes de iniciar a próxima tarefa, interrompa os processos do python em todas as máquinas da seguinte forma:

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

- Se o processo encerrar abruptamente sem erros, exclua a pasta de registros.

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

- Se ocorrerem outros problemas inexplicáveis, verifique o espaço em disco. Se você estiver fora, remova a pasta de registros, pois ela está cheia de pontos de verificação e de dados. Você também pode aumentar o tamanho do volume para cada membro.

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

- Como último recurso, você também pode reinicializar.

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

Você pode receber o seguinte código de erro se tentar usar TensorFlow com o Horovod em um tipo de instância não compatível:

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

Mais informações

Para utilitários e exemplos, consulte a pasta `~/examples/horovod` no diretório inicial da DLAMI.

Para ver ainda mais tutoriais e exemplos, consulte o projeto GitHub Horovod.

Elastic Fabric Adapter

O Elastic Fabric Adapter (EFA) é um dispositivo de rede que é possível anexar à instância da DLAMI para acelerar as aplicações de Computação de Alta Performance (HPC). O EFA permite que você alcance o desempenho do aplicativo de um cluster de HPC local, com a escalabilidade, a flexibilidade e a elasticidade fornecidas pela nuvem. AWS

Os tópicos a seguir mostram como começar a usar o EFA com a DLAMI.

> **Note**
>
> Escolha sua DLAMI nesta lista de DLAMI de GPU base

Tópicos

- Iniciando uma AWS Deep Learning AMI instância com o EFA
- Uso do EFA na DLAMI

Iniciando uma AWS Deep Learning AMI instância com o EFA

A DLAMI base mais recente está pronta para usar o EFA e vem com os drivers, os módulos do kernel, libfabric, openmpi e o plug-in OFI NCCL necessários para instâncias de GPU.

Você pode encontrar as versões CUDA compatíveis de uma DLAMI base nas notas de versão.

Nota:

- Ao executar um aplicativo NCCL usando `mpirun` no EFA, será necessário especificar o caminho completo para as instalações do EFA com suporte como:

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

- Para habilitar seu aplicativo para usar o EFA, adicione `FI_PROVIDER="efa"` ao comando `mpirun` como mostrado em Uso do EFA na DLAMI.

Tópicos

Preparar um grupo de segurança habilitado para o EFA

O EFA exige um grupo de segurança que permita todo o tráfego de entrada e saída de e para o próprio grupo de segurança. Para obter mais informações, consulte a documentação do EFA.

1. Abra o console do Amazon EC2 em https://console.aws.amazon.com/ec2/.
2. No painel de navegação, escolha Security Groups (Grupos de segurança) e, em seguida, Create Security Group (Criar grupo de segurança).
3. Na janela Security group, faça o seguinte:
   - Em Security group name (Nome do grupo de segurança), insira um nome descritivo para o grupo de segurança, como `EFA-enabled security group`.
   - (Opcional) Em Description (Descrição), insira uma breve descrição do grupo de segurança.
   - Em VPC, selecione a VPC na qual você pretende executar suas instâncias habilitadas para EFA.
   - Escolha Create (Criar).
4. Selecione o grupo de segurança que você criou e, na guia Description (Descrição), copie o Group ID (ID do grupo).
5. Nas guias Entrada e Saída, faça o seguinte:
   - Selecione Edit.
   - Para Type (Tipo), escolha All traffic (Todo o tráfego).

- Em Source, escolha Custom.
- Cole o ID do grupo de segurança que você copiou no campo.
- Escolha Salvar.

6. Habilite o tráfego de entrada fazendo referência a Autorizar tráfego de entrada para as instâncias do Linux. Se ignorar esta etapa, você não poderá se comunicar com sua instância da DLAMI.

Executar sua instância

Atualmente, o EFA no AWS Deep Learning AMI é compatível com os seguintes tipos de instância e sistemas operacionais:

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

A seção a seguir mostra como iniciar uma instância da DLAMI habilitada para EFA. Para obter mais informações sobre executar uma instância habilitada para EFA, consulte Executar instâncias habilitadas para EFA em um grupo com posicionamento em cluster.

1. Abra o console do Amazon EC2 em https://console.aws.amazon.com/ec2/.
2. Escolha Executar instância.
3. Na página Escolha uma AMI, selecione uma DLAMI compatível encontrada na página de notas de versão da DLAMI
4. Na página Escolher um tipo de instância, selecione um dos seguintes tipos de instância com suporte e escolha Próximo: Configurar os detalhes da instância. Consulte este link para ver a lista de instâncias compatíveis: Comece a usar o EFA e o MPI
5. Na página Configurar detalhes da instância, faça o seguinte:

   - Em Number of instances (Número de instâncias), insira o número de instâncias habilitadas para EFA que você deseja executar.
   - Em Rede e Sub-rede, selecione a VPC e a sub-rede na qual executar as instâncias.
   - [Opcional] Em Grupo de posicionamento, selecione Adicionar instância ao grupo de posicionamento. Para obter o melhor desempenho, execute as instâncias dentro de um placement group.

- [Opcional] Em Nome do grupo de posicionamento, selecione Adicionar a um novo grupo de posicionamento, insira um nome descritivo para o grupo de posicionamento e, em seguida, em Estratégia do grupo de posicionamento, selecione cluster.
- Ative o “Elastic Fabric Adapter” nesta página. Se esta opção estiver desabilitada, altere a sub-rede para uma que ofereça suporte ao tipo de instância selecionado.
- Na seção Interfaces de rede, para dispositivo eth0, escolha Nova interface de rede. Como opção, é possível especificar um endereço IPv4 principal e um ou mais endereços IPv4 secundários. Se estiver executando a instância em uma sub-rede que tenha um bloco CIDR IPv6 associado, será possível especificar opcionalmente um endereço IPv6 principal e um ou mais endereços IPv6 secundários.
- Escolha Next: Add Storage.

6. Na página Add Storage (Adicionar armazenamento), especifique os volumes para anexar às instâncias em associação aos volumes especificados pela AMI (como o volume do dispositivo raiz) e escolha Next: Add Tags (Próximo: adicionar tags).
7. Na página Adicionar tags, especifique tags para as instâncias, como nome amigável, e selecione Próximo: Configurar grupo de segurança.
8. Na página Configurar grupo de segurança, em Atribuir um grupo de segurança, selecione Selecionar um grupo de segurança existente e, em seguida, selecione o grupo de segurança que você criou anteriormente.
9. Escolha revisar e iniciar.
10. Na página Revisar execução da instância, reveja as configurações, e escolha Executar para escolher um par de chaves e executar a instâncias.

Verificar anexo do EFA

No console

Depois de iniciar a instância, verifique os detalhes da instância no AWS console. Para fazer isso, selecione a instância no console do EC2 e observe a guia "Descrição" no painel inferior da página. Encontre o parâmetro "Network Interfaces: eth0" e clique em eth0 e um pop-up será aberto. O “Elastic Fabric Adapter” deve estar ativado.

Se o EFA não estiver habilitado, você poderá corrigir isso da seguinte forma:

- Encerre a instância do EC2 e execute uma nova com as mesmas etapas. Verifique se o EFA está associado.

- Associe o EFA a uma instância existente.

1. No console do EC2, acesse "Interfaces de rede".
2. Clique em "Criar uma interface de rede".
3. Selecione a mesma sub-rede na qual está sua instância.
4. Habilite o "Elastic Fabric Adapter" e clique em Criar.
5. Volte para a guia "Instâncias do EC2" e selecione sua instância.
6. Acesse Ações: estado da instância e interrompa a instância antes de vincular o EFA.
7. Em "Ações", selecione "Rede: Anexar Interface de Rede".
8. Selecione a interface que você acabou de criar e clique em associar.
9. Reinicie sua instância.

Na instância

O script de teste a seguir já está presente na DLAMI. Execute-o para garantir que os módulos do kernel sejam carregados corretamente.

```
$ fi_info -p efa
```

O resultado deve ser semelhante ao seguinte:

```
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
```

Verificar a configuração do grupo de segurança

O script de teste a seguir já está presente na DLAMI. Execute-o para garantir que o grupo de segurança criado esteja configurado corretamente.

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

O resultado deve ser semelhante ao seguinte:

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

Se ele parar de responder ou não for concluído, verifique se o grupo de segurança tem as regras corretas de entrada/saída.

Uso do EFA na DLAMI

A seção a seguir descreve como usar o EFA para executar aplicativos de vários nós na AWS Deep Learning AMI.

Como executar aplicativos de vários nós com o EFA

Para executar um aplicativo em um cluster de nós, é necessária a seguinte configuração:

Tópicos

- Permitir SSH sem senha
- Criar arquivo de hosts
- Testes NCCL

Permitir SSH sem senha

Selecione um nó no cluster como o nó líder. Os nós restantes são referidos como nós membros.

1. No nó líder, gere o par de chaves RSA.

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

2. Altere as permissões da chave privada no nó líder.

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

3. Copie `~/.ssh/id_rsa.pub` a chave pública e anexe-a aos nós membros `~/.ssh/authorized_keys` do cluster.
4. Agora, você poderá fazer login diretamente nos nós membros a partir do nó líder usando o ip privado.

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

5. Desative a strictHostKey verificação e habilite o encaminhamento de agentes no nó principal adicionando o seguinte ao arquivo ~/.ssh/config no nó líder:

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

6. Nas instâncias do Amazon Linux 2, execute o seguinte comando no nó líder para fornecer as permissões corretas ao arquivo de configuração:

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

Criar arquivo de hosts

No nó líder, crie um arquivo de hosts para identificar os nós no cluster. O arquivo de hosts deve ter uma entrada para cada nó no cluster. Crie um arquivo ~/hosts e adicione cada nó usando o ip privado da seguinte forma:

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

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

Testes NCCL

> **Note**
>
> Esses testes foram executados usando o EFA versão 1.30.0 e o OFI NCCL Plugin 1.7.4.

Abaixo está listado um subconjunto de testes NCCL fornecidos pela Nvidia para testar a funcionalidade e o desempenho em vários nós de computação

Instâncias suportadas: P3dn, P4, P5

Testes de funcionalidade

Teste de vários nós de transferência de mensagens NCCL

O nccl_message_transfer é um teste simples para garantir que o plug-in OFI NCCL esteja funcionando como esperado. O teste valida a funcionalidade das APIs de estabelecimento de conexão e transferência de dados do NCCL. Use o caminho completo para mpirun como mostrado no exemplo ao executar aplicativos NCCL com o EFA. Altere os parâmetros `np` e `N` com base no número de instâncias e GPUs em seu cluster. Para obter mais informações, consulte a Documentação do AWS OFI NCCL.

O seguinte teste nccl_message_transfer é para a versão genérica do CUDA xx.x. Você pode executar os comandos para qualquer versão do CUDA disponível em sua instância do Amazon EC2 substituindo a versão do CUDA no 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
```

A saída será semelhante a: É possível verificar a saída para ver se o EFA está sendo usado como o provedor 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
```

Testes de desempenho

Teste de desempenho de vários nós do NCCL em P4d.24xlarge

Para verificar o desempenho do NCCL com o EFA, execute o teste de desempenho do NCCL padrão que está disponível no NCCL-Tests Repo. O DLAMI vem com esse teste já criado para CUDA XX.X. Você também pode executar seu próprio script com o EFA.

Ao criar seu próprio script, consulte as seguintes orientações:

- Use o caminho completo para mpirun como mostrado no exemplo ao executar aplicativos NCCL com o EFA.
- Altere os parâmetros np e N com base no número de instâncias e GPUs em seu cluster.
- Adicione o sinalizador NCCL_DEBUG=INFO e verifique se os registros indicam o uso do EFA como “O provedor selecionado é EFA”.
- Defina o local do registro de treinamento a ser analisado para validação

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

Use o comando `watch nvidia-smi` em qualquer um dos nós membros para monitorar o uso da GPU. Os comandos `watch nvidia-smi` a seguir são para o CUDA versão xx.x e dependem do sistema operacional da instância. Você pode executar os comandos para qualquer versão do CUDA disponível em sua instância do Amazon EC2 substituindo a versão do CUDA no 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}
```

A saída será semelhante a:

```
# 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
```

Testes de validação

Para validar se os testes EFA retornaram um resultado válido, use os seguintes testes para confirmar:

- Obtenha o tipo de instância usando os metadados de instância do 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)
```

- Execute a Testes de desempenho
- Defina os seguintes parâmetros

```
CUDA_VERSION
CUDA_RUNTIME_VERSION
NCCL_VERSION
```

- Valide os resultados conforme mostrado:

```
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 acessar os dados do benchmark, podemos analisar a linha final da saída da tabela do teste Multi Node all_reduce:

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

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

Monitoramento e otimização de GPU

A seção a seguir orientará você durante a otimização de GPU e opções de monitoramento. Esta seção é organizada como um fluxo de trabalho típico com monitoramento, supervisão, pré-processamento e treinamento.

- Monitoramento
  - Monitore GPUs com CloudWatch
- Otimização
  - Pré-processamento
  - Treinamento

Monitoramento

A DLAMI vem pré-instalada com várias ferramentas de monitoramento de GPU. Este guia menciona também ferramentas que estão disponíveis para download e instalação.

- Monitore GPUs com CloudWatch- um utilitário pré-instalado que reporta estatísticas de uso da GPU para a Amazon. CloudWatch
- CLI nvidia-smi – um utilitário para monitorar a utilização geral de computação e memória de GPU. Isso está pré-instalado no seu AWS Deep Learning AMI (DLAMI).
- Biblioteca NVML C – uma API baseada em C para acessar diretamente funções de monitoramento e gerenciamento de GPU. Isso é usado pela CLI nvidia-smi nos bastidores e é pré-instalado na DLAMI. Também tem associações Python e Perl para facilitar o desenvolvimento nessas linguagens. O utilitário gpumon.py pré-instalado em seu DLAMI usa o pacote pynvml do. nvidia-ml-py
- NVIDIA DCGM – uma ferramenta de gerenciamento de cluster. Visite a página do desenvolvedor para saber como instalar e configurar essa ferramenta.

> **Tip**
>
> Confira o blog do desenvolvedor de NVIDIA para obter as informações mais recentes sobre como usar as ferramentas do CUDA instaladas na DLAMI:
>
> - Monitorando TensorCore a utilização usando o Nsight IDE e o nvprof.

Monitore GPUs com CloudWatch

Ao usar a DLAMI com uma GPU, talvez você descubra que está procurando maneiras de controlar o uso durante o treinamento ou a inferência. Isso pode ser útil para otimizar o pipeline de dados e ajustar sua rede de aprendizado profundo.

Há duas maneiras de configurar as métricas da GPU com CloudWatch:

- Configurar métricas com o AWS CloudWatch agente (recomendado)
- Configurar métricas com o script `gpumon.py` pré-instalado

Configurar métricas com o AWS CloudWatch agente (recomendado)

Integre seu DLAMI com o agente CloudWatch unificado para configurar métricas de GPU e monitorar a utilização de coprocessos de GPU em instâncias aceleradas do Amazon EC2.

Há quatro maneiras de configurar métricas de GPU com a DLAMI:

- Configurar métricas de GPU mínimas
- Configurar métricas de GPU parciais
- Configurar todas as métricas de GPU disponíveis
- Configurar métricas de GPU personalizadas

Para obter mais informações sobre atualizações e patches de segurança, consulte Patches de segurança para o agente AWS CloudWatch

Pré-requisitos

Para começar, você deve configurar as permissões IAM da instância do Amazon EC2 que permitam que sua instância envie métricas para. CloudWatch Para ver etapas detalhadas, consulte Criar funções e usuários do IAM para uso com o CloudWatch agente.

Configurar métricas de GPU mínimas

Configure métricas mínimas de GPU usando o serviço `dlami-cloudwatch-agent@minimal systemd`. Esse serviço configura as seguintes métricas:

- `utilization_gpu`
- `utilization_memory`

Você pode encontrar o serviço `systemd` para métricas mínimas de GPU pré-configuradas no seguinte local:

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

Ative e inicie o serviço `systemd` com os seguintes comandos:

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

Configurar métricas de GPU parciais

Configure métricas de GPU parciais usando o serviço `dlami-cloudwatch-agent@partial` `systemd`. Esse serviço configura as seguintes métricas:

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

Você pode encontrar o serviço `systemd` para métricas parciais de GPU pré-configuradas no seguinte local:

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

Ative e inicie o serviço `systemd` com os seguintes comandos:

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

Configurar todas as métricas de GPU disponíveis

Configure todas as métricas de GPU disponíveis usando o serviço `dlami-cloudwatch-agent@all` `systemd`. Esse serviço configura as seguintes 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

Você pode encontrar o serviço systemd para todas as métricas disponíveis de GPU pré-configuradas no seguinte local:

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

Ative e inicie o serviço systemd com os seguintes comandos:

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

Configurar métricas de GPU personalizadas

Se as métricas pré-configuradas não atenderem aos seus requisitos, você poderá criar um arquivo personalizado de configuração do CloudWatch agente.

Criar um arquivo de configuração personalizada

Para criar um arquivo de configuração personalizado, consulte as etapas detalhadas em Criar ou editar manualmente o arquivo de configuração do CloudWatch agente.

Neste exemplo, suponha que a definição do esquema esteja localizada em `/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json`.

Configurar métricas com seu arquivo personalizado

Execute o comando a seguir para configurar o CloudWatch agente de acordo com seu arquivo 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
```

Patches de segurança para o agente AWS CloudWatch

Os DLAMIs recém-lançados são configurados com os patches de segurança de AWS CloudWatch agentes mais recentes disponíveis. Consulte as seções a seguir para atualizar a DLAMI atual com os patches de segurança mais recentes, dependendo do sistema operacional escolhido.

Amazon Linux 2

Use `yum` para obter os patches de segurança de AWS CloudWatch agentes mais recentes para um Amazon Linux 2 DLAMI.

```
sudo yum update
```

Ubuntu

Para obter os patches de AWS CloudWatch segurança mais recentes para um DLAMI com Ubuntu, é necessário reinstalar o agente usando um link de download AWS CloudWatch do Amazon S3.

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

Para obter mais informações sobre como instalar o AWS CloudWatch agente usando os links de download do Amazon S3, consulte Instalando e executando o CloudWatch agente em seus servidores.

Configurar métricas com o script **`gpumon.py`** pré-instalado

Um utilitário chamado gpumon.py é pré-instalado na DLAMI. Ele se integra CloudWatch e oferece suporte ao monitoramento do uso por GPU: memória da GPU, temperatura da GPU e potência da

GPU. O script envia periodicamente os dados monitorados para CloudWatch o. Você pode configurar o nível de granularidade dos dados enviados CloudWatch alterando algumas configurações no script. Antes de iniciar o script, no entanto, você precisará configurar CloudWatch para receber as métricas.

Como configurar e executar o monitoramento de GPU com CloudWatch

1. Crie um usuário do IAM ou modifique um existente para ter uma política para publicar a métrica CloudWatch. Se você criar um novo usuário, anote as credenciais, pois elas serão necessárias na próxima etapa.

   A política do IAM a ser pesquisada é “cloudwatch:PutMetricData”. A política que é adicionada é a seguinte:

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

   > **Tip**
   >
   > Para obter mais informações sobre como criar um usuário do IAM e adicionar políticas para CloudWatch, consulte a CloudWatch documentação.

2. Em sua DLAMI, execute AWS configure e especifique as credenciais de usuário do IAM.

   ```
   $ aws configure
   ```

3. Talvez você precise fazer algumas modificações no utilitário gpumon antes de executá-lo. Você pode encontrar o utilitário gpumon e o README no seguinte local definido no seguinte bloco de código. Para obter mais informações sobre o script `gpumon.py`, consulte a localização do script no Amazon S3.

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

Opções:

- Altere a região no gpumon.py se sua instância NÃO estiver em us-east-1.
- Altere outros parâmetros, como o período do relatório CloudWatch `namespace` ou o período do relatório, com`store_reso`.

4. No momento, o script oferece suporte apenas ao Python 3. Ative o ambiente do Python 3 da estrutura de trabalho preferencial ou ative o ambiente geral do Python 3 da DLAMI.

```
$ source activate python3
```

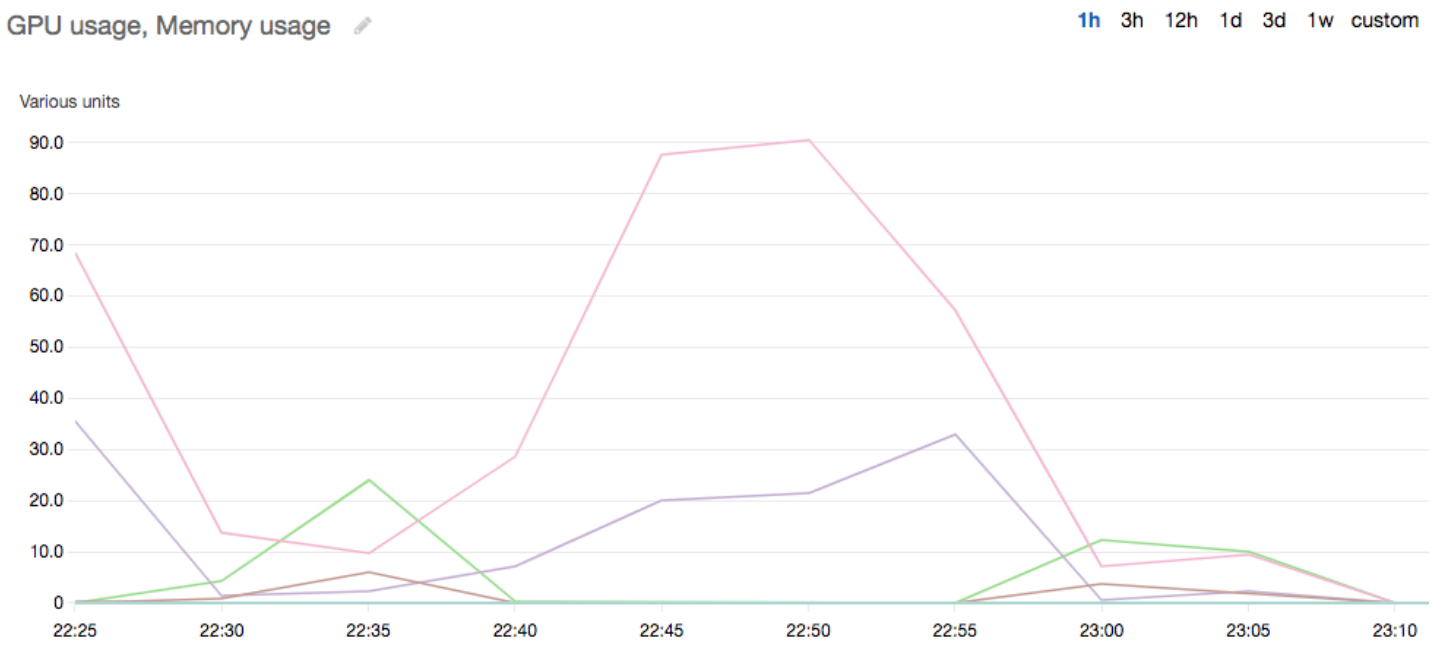
5. Execute o utilitário gpumon em segundo plano.

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

6. Abra seu navegador em https://console.aws.amazon.com/cloudwatch/ e selecione a métrica. Ele terá um namespace ". DeepLearningTrain

> **Tip**
>
> Você pode alterar o namespace modificando o gpumon.py. Você também pode modificar o intervalo de relatório ajustando `store_reso`.

Veja a seguir um exemplo de CloudWatch gráfico relatando uma execução do gpumon.py monitorando um trabalho de treinamento na instância p2.8xlarge.



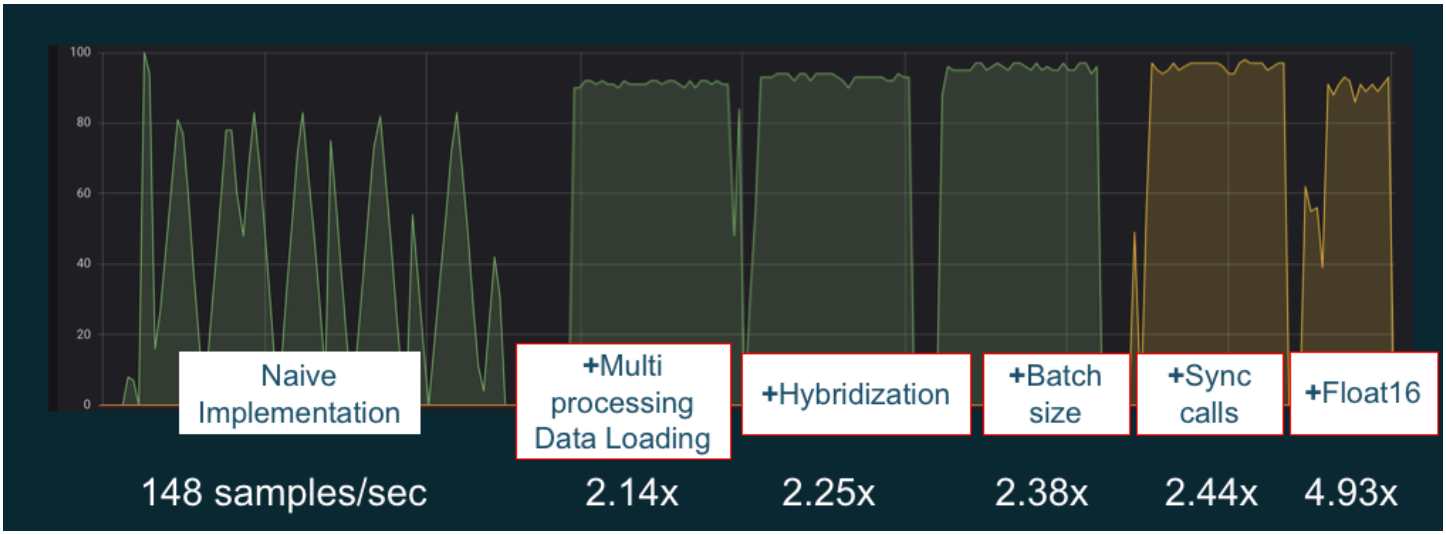
Você pode estar interessado nesses outros tópicos sobre monitoramento e otimização de GPU:

- Monitoramento
  - Monitore GPUs com CloudWatch
- Otimização
  - Pré-processamento
  - Treinamento

Otimização

Para obter o máximo de suas GPUs, você pode otimizar seu pipeline de dados e ajustar a rede de aprendizado profundo. Como o gráfico a seguir descreve, uma implementação simples ou básica de uma rede neural pode usar a GPU de maneira inconsistente e não aproveitar o potencial máximo. Ao otimizar o pré-processamento e o carregamento de dados, você pode reduzir o gargalo da CPU para a GPU. Você pode ajustar a rede neural em si usando hibridização (quando compatível com a estrutura), ajustando o tamanho do lote e sincronizando as chamadas. Você também pode usar treinamento de precisão múltipla (float16 ou int8) na maioria das estruturas, o que pode causar um efeito enorme na melhoria da taxa de transferência.

A tabela a seguir mostra os ganhos de desempenho cumulativos ao aplicar otimizações diferentes. Seus resultados dependerão dos dados que você está processando e da rede que está otimizando.



Exemplos de otimização de desempenho de GPUs. Origem do gráfico: Truques de desempenho com o MXNet Gluon

Os seguintes guias apresentam opções que funcionarão com a DLAMI e ajudarão a aumentar o desempenho da GPU.

Tópicos

- Pré-processamento
- Treinamento

Pré-processamento

O pré-processamento de dados durante transformações ou aumento muitas vezes pode ser um processo vinculado à CPU e isso pode ser o gargalo em todo o seu pipeline. Estruturas têm operadores integrados para processamento de imagens, mas a DALI (Data Augmentation Library) demonstra um melhor desempenho em relação às opções integradas das estruturas.

- NVIDIA Data Augmentation Library (DALI): a DALI minimiza o aumento dos dados para a GPU. Não é pré-instalada na DLAMI, mas você pode ter acesso instalando-a ou carregando um contêiner de estrutura compatível na sua DLAMI ou em outra instância do Amazon Elastic Compute Cloud. Consulte a página do projeto DALI no site do NVIDIA para obter detalhes. Para ver um exemplo de caso de uso e baixar amostras de código, consulte a amostra de desempenho do treinamento SageMaker de pré-processamento.
- nvJPEG: uma biblioteca de decodificadores JPEG acelerados para GPU para programadores de C. Ela oferece suporte à decodificação de imagens ou lotes únicos, bem como operações

de transformação subsequentes que são comuns em aprendizado profundo. A nvJPEG vem integrada à DALI, ou você pode fazer download da página nvjpeg do site da NVIDIA e usá-la separadamente.

Você pode estar interessado nesses outros tópicos sobre monitoramento e otimização de GPU:

- Monitoramento
  - Monitore GPUs com CloudWatch
- Otimização
  - Pré-processamento
  - Treinamento

Treinamento

Com treinamento de precisão mista, você pode implantar redes maiores com a mesma quantidade de memória ou reduzir o uso de memória em comparação com sua rede de precisão única ou dupla, e você verá aumentos de desempenho de computação. Você também pode obter o benefício de transferências de dados menores e mais rápidas, um fator importante em um treinamento distribuído em vários nós. Para aproveitar o treinamento de precisão mista, você precisa ajustar a conversão de dados e a escalabilidade de perdas. Veja a seguir os guias que descrevem como fazer isso para as estruturas que oferecem suporte à precisão mista.

- SDK NVIDIA Deep Learning - documentos no site da NVIDIA descrevendo a implementação de precisão mista para MXNet, e. PyTorch TensorFlow

> **Tip**
>
> Verifique se o site oferece a estrutura de sua preferência e procure por "precisão mista" ou "fp16" para encontrar as técnicas de otimização mais recentes. Veja a seguir alguns guias sobre precisão mista que podem ser úteis:
>
> - Treinamento de precisão mista com TensorFlow (vídeo) - no site do blog da NVIDIA.
> - Treinamento de precisão mista usando float16 com MXNet - um artigo de perguntas frequentes no site do MXNet.

- NVIDIA Apex: uma ferramenta para treinamento fácil de precisão mista com PyTorch - um artigo de blog no site da NVIDIA.

Você pode estar interessado nesses outros tópicos sobre monitoramento e otimização de GPU:

- Monitoramento
  - Monitore GPUs com CloudWatch
- Otimização
  - Pré-processamento
  - Treinamento

O chip de AWS inferência com DLAMI

AWS O Inferentia é um chip de aprendizado de máquina personalizado projetado por AWS ele que você pode usar para previsões de inferência de alto desempenho. Para usar o chip, configure uma instância Amazon Elastic Compute Cloud e use o kit de desenvolvimento de software (SDK) AWS Neuron para invocar o chip Inferentia. Para fornecer aos clientes a melhor experiência no Inferentia, o Neuron foi integrado à AWS Deep Learning AMI (DLAMI).

Os tópicos a seguir mostram como começar a usar o Inferentia com a DLAMI.

Conteúdo

- Lançamento de uma instância DLAMI com Neuron AWS
- Usando o DLAMI com Neuron AWS

Lançamento de uma instância DLAMI com Neuron AWS

O DLAMI mais recente está pronto para uso AWS com o Inferentia e vem com AWS o pacote Neuron API. Para iniciar uma instância da DLAMI, consulte Iniciar e configurar uma DLAMI. Depois de ter um DLAMI, use as etapas aqui para garantir que AWS seu chip de inferência AWS e os recursos do Neuron estejam ativos.

Conteúdo

- Verifique a instância
- Identificação de AWS dispositivos de inferência

- Exibir o uso de recursos
- Como usar o Monitor do Neuron
- Atualização do software Neuron

Verifique a instância

Antes de usar a instância, verifique se ela está corretamente definida e configurada com o Neuron.

Identificação de AWS dispositivos de inferência

Para identificar o número de dispositivos do Inferentia na sua instância, use o seguinte comando:

```
neuron-ls
```

Se a instância tiver dispositivos do Inferentia conectados a ela, a saída será semelhante à seguinte:

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

O resultado fornecido é obtida de uma instância INF1.6xlarge e inclui as seguintes colunas:

- DISPOSITIVO NEURONAL: O ID lógico atribuído ao NeuronDevice. Esse ID é usado ao configurar vários tempos de execução para usar diferentes. NeuronDevices
- NÚCLEOS DE NEURÔNIOS: O número de NeuronCores presentes no NeuronDevice.
- MEMÓRIA NEURONAL: A quantidade de memória DRAM no. NeuronDevice
- DISPOSITIVOS CONECTADOS: Outros NeuronDevices conectados ao NeuronDevice.
- PCI BDF: O ID da função de dispositivo de barramento PCI (BDF) do. NeuronDevice

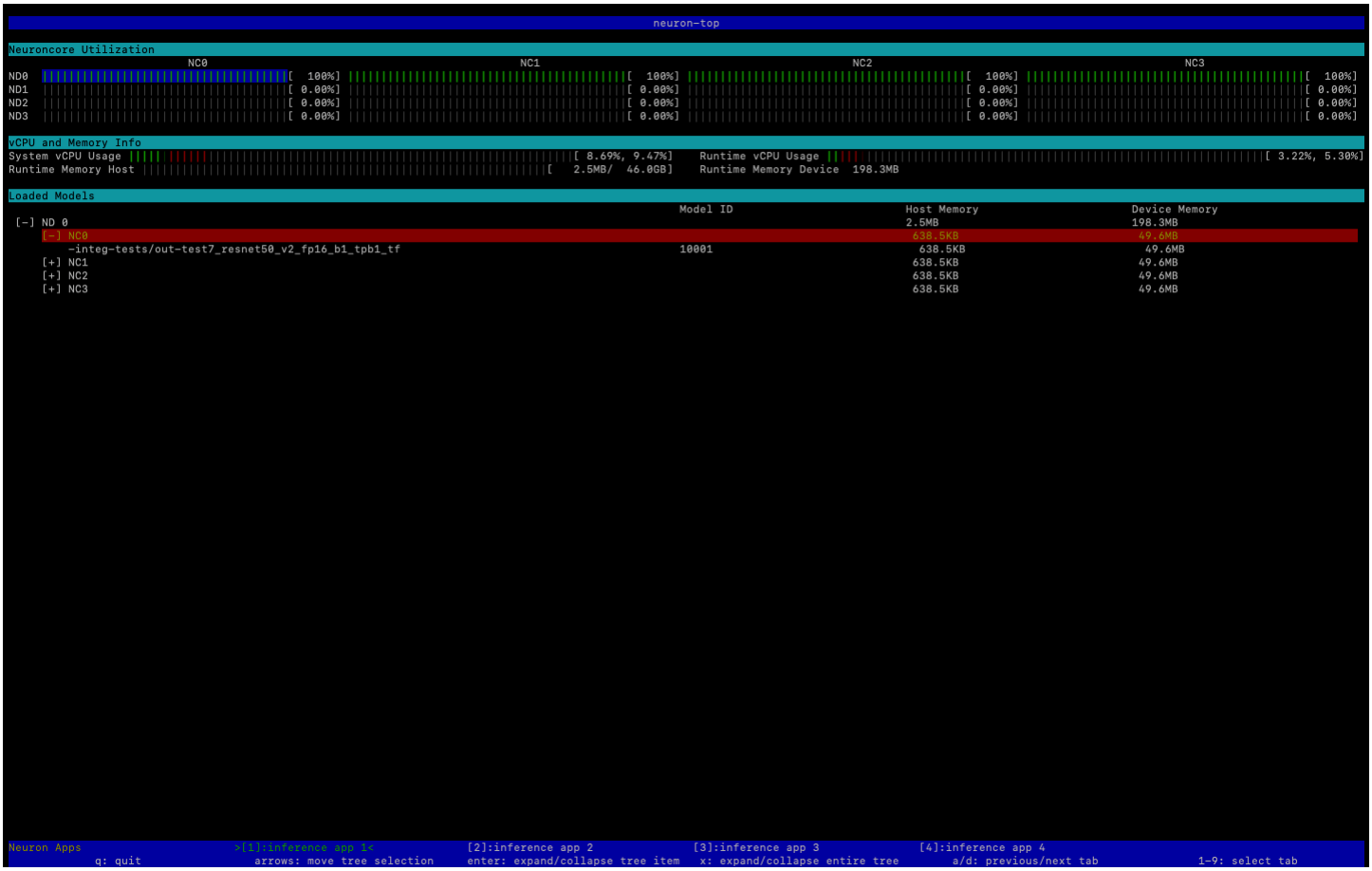
Exibir o uso de recursos

Visualize informações úteis sobre a NeuronCore utilização da vCPU, o uso da memória, os modelos carregados e os aplicativos Neuron com o comando. `neuron-top` O lançamento `neuron-top` sem

argumentos mostrará os dados de todos os aplicativos de aprendizado de máquina que utilizam NeuronCores.

```
neuron-top
```

Quando um aplicativo está usando quatro NeuronCores, a saída deve ser semelhante à imagem a seguir:



Para obter mais informações sobre recursos para monitorar e otimizar aplicativos de inferência que usam como base o Neuron, consulte Ferramentas do Neuron.

Como usar o Monitor do Neuron

O Monitor do Neuron coleta métricas dos runtimes do Neuron em execução no sistema e transmite os dados coletados para stdout no formato JSON. Elas são organizadas em grupos de métricas que você configura fornecendo um arquivo de configuração. Para obter mais informações sobre o Monitor do Neuron, consulte o Guia do usuário do monitor do Neuron.

Atualização do software Neuron

Para obter informações sobre como atualizar o software Neuron SDK no DLAMI, consulte o Guia de configuração do Neuron. AWS

Próxima etapa

Usando o DLAMI com Neuron AWS

Usando o DLAMI com Neuron AWS

Um fluxo de trabalho típico com o AWS Neuron SDK é compilar um modelo de aprendizado de máquina previamente treinado em um servidor de compilação. Depois disso, distribua os artefatos para as instâncias Inf1 para execução. AWS Deep Learning AMI (DLAMI) vem pré-instalado com tudo o que você precisa para compilar e executar inferência em uma instância Inf1 que usa Inferentia.

As seções a seguir descrevem como usar a DLAMI com o Inferentia.

Conteúdo

- Usando TensorFlow -Neuron e o compilador Neuron AWS
- Usando AWS Neuron Serving TensorFlow
- Usando o MXNet-Neuron e o Neuron Compiler AWS
- Uso do fornecimento de modelos MXNet-Neuron
- Usando PyTorch -Neuron e o compilador Neuron AWS

Usando TensorFlow -Neuron e o compilador Neuron AWS

Este tutorial mostra como usar o compilador AWS Neuron para compilar o modelo Keras ResNet -50 e exportá-lo como um modelo salvo em formato. SavedModel Esse formato é um formato típico de TensorFlow modelo intercambiável. Você também aprenderá a executar a inferência em uma instância do Inf1 com exemplo de entrada.

Para obter mais informações sobre o SDK do Neuron, consulte a Documentação do SDK do AWS Neuron.

Conteúdos

- Pré-requisitos

- Ative o ambiente Conda
- Compilação ResNet50
- ResNet50 Inferência

Pré-requisitos

Antes de usar este tutorial, você precisa ter concluído os passos da configuração em Lançamento de uma instância DLAMI com Neuron AWS. Também é necessário conhecer a aprendizagem profunda e o uso da DLAMI.

Ative o ambiente Conda

Ative o ambiente TensorFlow -Neuron conda usando o seguinte comando:

```
source activate aws_neuron_tensorflow_p36
```

Para sair do ambiente Conda atual, execute o seguinte comando:

```
source deactivate
```

Compilação ResNet50

Crie um script Python chamado **`tensorflow_compile_resnet50.py`** com o seguinte conteúdo. Esse script Python compila o modelo Keras ResNet 50 e o exporta como um modelo salvo.

```
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 o modelo usando o seguinte comando:

```
python tensorflow_compile_resnet50.py
```

O processo de compilação leva alguns minutos. Quando concluído, sua saída será semelhante a:

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

Após a compilação, o modelo salvo será compactado em **`ws_resnet50/resnet50_neuron.zip`**. Descompacte o modelo e faça download da imagem de exemplo para a inferência, usando os seguintes 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 Inferência

Crie um script Python chamado **`tensorflow_infer_resnet50.py`** com o seguinte conteúdo. Esse script executa a inferência no modelo obtido por download usando um modelo de inferência previamente compilado.

```
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])
```

Execute a inferência no modelo usando o seguinte comando:

```
python tensorflow_infer_resnet50.py
```

A saída será semelhante a:

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

Próxima etapa

Usando AWS Neuron Serving TensorFlow

Usando AWS Neuron Serving TensorFlow

Este tutorial mostra como construir um gráfico e adicionar uma etapa de compilação do AWS Neuron antes de exportar o modelo salvo para uso com o Serving. TensorFlow TensorFlow Serving é um sistema de atendimento que permite ampliar a inferência em uma rede. O Neuron TensorFlow Serving usa a mesma API do TensorFlow Serving normal. A única diferença é que um modelo salvo deve ser compilado para AWS Inferentia e o ponto de entrada é um binário diferente chamado`tensorflow_model_server_neuron`. O binário é encontrado em `/usr/local/bin/tensorflow_model_server_neuron` e é pré-instalado na DLAMI.

Para obter mais informações sobre o SDK do Neuron, consulte a Documentação do SDK do AWS Neuron.

Conteúdos

- Pré-requisitos
- Ative o ambiente Conda
- Compile e exporte o modelo salvo
- Fornecer o modelo salvo
- Gerar solicitações de inferência para o modelo de servidor

Pré-requisitos

Antes de usar este tutorial, você precisa ter concluído os passos da configuração em Lançamento de uma instância DLAMI com Neuron AWS. Também é necessário conhecer a aprendizagem profunda e o uso da DLAMI.

Ative o ambiente Conda

Ative o ambiente TensorFlow -Neuron conda usando o seguinte comando:

```
source activate aws_neuron_tensorflow_p36
```

Se você precisar sair do ambiente Conda atual, execute:

```
source deactivate
```

Compile e exporte o modelo salvo

Crie um script Python chamado `tensorflow-model-server-compile.py` com o conteúdo a seguir. Ele constrói um gráfico e o compila usando o Neuron. Depois, exporta o gráfico compilado como modelo salvo.

```
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 o modelo usando o seguinte comando:

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

A saída será semelhante a:

```
...
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
```

Fornecer o modelo salvo

Depois que o modelo foi compilado, você pode usar o seguinte comando para fornecer o modelo salvo com o binário tensorflow_model_server_neuron:

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

A saída será semelhante a: O modelo compilado é preparado na DRAM do dispositivo do Inferentia, pelo servidor, para preparar para a inferência.

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

Gerar solicitações de inferência para o modelo de servidor

Crie um script Python chamado `tensorflow-model-server-infer.py` com o conteúdo a seguir. Esse script executa a inferência via gRPC, que é um framework de serviço.

```
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))
```

Execute a inferência no modelo usando gRPC com o seguinte comando:

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

A saída será semelhante a:

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

Usando o MXNet-Neuron e o Neuron Compiler AWS

A API de compilação MXNet-Neuron fornece um método para compilar um gráfico de modelo que você pode executar em um dispositivo Inferentia. AWS

Neste exemplo, você usa a API para compilar um modelo ResNet -50 e usá-lo para executar inferência.

Para obter mais informações sobre o SDK do Neuron, consulte a Documentação do SDK do AWS Neuron.

Conteúdos

Pré-requisitos

Antes de usar este tutorial, você precisa ter concluído os passos da configuração em Lançamento de uma instância DLAMI com Neuron AWS. Também é necessário conhecer a aprendizagem profunda e o uso da DLAMI.

Ative o ambiente Conda

Ative o ambiente Conda do MXNet-Neuron usando o seguinte comando:

```
source activate aws_neuron_mxnet_p36
```

Para sair do ambiente Conda atual, execute:

```
source deactivate
```

Compilação ResNet50

Crie um script Python chamado **`mxnet_compile_resnet50.py`** com o conteúdo a seguir. Esse script usa a API Python de compilação MXNet-Neuron para compilar um modelo -50. ResNet

```
import mxnet as mx
import numpy as np

print("downloading...")
path='http://data.mxnet.io/models/imagenet/'
mx.test_utils.download(path+'resnet/50-layers/resnet-50-0000.params')
mx.test_utils.download(path+'resnet/50-layers/resnet-50-symbol.json')
```

```
print("download finished.")

sym, args, aux = mx.model.load_checkpoint('resnet-50', 0)

print("compile for inferentia using neuron... this will take a few minutes...")
inputs = { "data" : mx.nd.ones([1,3,224,224], name='data', dtype='float32') }

sym, args, aux = mx.contrib.neuron.compile(sym, args, aux, inputs)

print("save compiled model...")
mx.model.save_checkpoint("compiled_resnet50", 0, sym, args, aux)
```

Compile o modelo usando o seguinte comando:

```
python mxnet_compile_resnet50.py
```

A compilação demora alguns minutos. Quando ela terminar, os seguintes arquivos estarão no diretório atual:

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

ResNet50 Inferência

Crie um script Python chamado **`mxnet_infer_resnet50.py`** com o conteúdo a seguir. Esse script faz download de uma imagem de amostra e a usa para executar a inferência com o 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]))
```

Execute a inferência com o modelo compilado usando o seguinte comando:

```
python mxnet_infer_resnet50.py
```

A saída será semelhante a:

```
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
```

Próxima etapa

Uso do fornecimento de modelos MXNet-Neuron

Uso do fornecimento de modelos MXNet-Neuron

Neste tutorial, você aprenderá a usar um modelo MXNet pré-treinado para executar a classificação de imagens em tempo real com o Multi Model Server (MMS). O MMS é uma easy-to-use ferramenta flexível para servir modelos de aprendizado profundo que são treinados usando qualquer estrutura de aprendizado de máquina ou aprendizado profundo. Este tutorial inclui uma etapa de compilação usando o AWS Neuron e uma implementação do MMS usando o MXNet.

Para obter mais informações sobre o SDK do Neuron, consulte a Documentação do SDK do AWS Neuron.

Conteúdos

Pré-requisitos

Antes de usar este tutorial, você precisa ter concluído os passos da configuração em Lançamento de uma instância DLAMI com Neuron AWS. Também é necessário conhecer a aprendizagem profunda e o uso da DLAMI.

Ative o ambiente Conda

Para ativar o ambiente Conda do MXNet-Neuron, use o seguinte comando:

```
source activate aws_neuron_mxnet_p36
```

Para sair do ambiente Conda atual, execute:

```
source deactivate
```

Faça download do código de exemplo

Para executar o exemplo, faça download do código de exemplo usando os seguintes comandos:

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

Compile o modelo.

Crie um script Python chamado `multi-model-server-compile.py` com o conteúdo a seguir. Esse script compila o modelo ResNet 50 para o alvo do 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 o modelo, use o seguinte comando:

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

A saída será semelhante a:

```
...
```

```
[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!
```

Crie um arquivo chamado `signature.json` com o seguinte conteúdo para configurar o nome e a forma de entrada:

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

Faça download do arquivo `synset.txt` usando o seguinte comando: Esse arquivo é uma lista de nomes para classes de ImageNet predição.

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

Crie uma classe de serviço personalizada seguindo o modelo na pasta `model_server_template`. Copie o modelo para o diretório de trabalho atual usando o seguinte comando:

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

Edite o módulo `mxnet_model_service.py` para substituir o contexto `mx.cpu()` pelo contexto `mx.neuron()`, da seguinte forma. Também é preciso comentar a cópia de dados desnecessária para o `model_input`, porque o MXNet-Neuron não é compatível com as APIs NDArray e Gluon.

```
...
self.mxnet_ctx = mx.neuron() if gpu_id is None else mx.gpu(gpu_id)
...
#model_input = [item.as_in_context(self.mxnet_ctx) for item in model_input]
```

Empacote o modelo com arquivador de modelos, usando os seguintes comandos:

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

Execute a inferência

Inicie o Multi Model Server e carregue o modelo que usa a API RESTful, usando os seguintes comandos. Certifique-se de que o neuron-rtd está sendo executado com as configurações padrão.

```
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
```

Execute a inferência usando uma imagem de exemplo com os seguintes 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
```

A saída será semelhante a:

```
[
  {
    "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 fazer a limpeza após o teste, dê um comando "delete" por meio da API RESTful e pare o servidor modelo usando os seguintes comandos:

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

multi-model-server --stop
```

A seguinte saída deverá ser mostrada:

```
{
  "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
```

Usando PyTorch -Neuron e o compilador Neuron AWS

A API de compilação PyTorch -Neuron fornece um método para compilar um gráfico de modelo que você pode executar em um dispositivo Inferentia. AWS

Um modelo treinado deve ser compilado para um destino Inferentia antes que ele possa ser implantado em instâncias Inf1. O tutorial a seguir compila o modelo torchvision ResNet 50 e o exporta como um módulo salvo. TorchScript Esse modelo é, assim sendo, usado para executar inferência.

Por conveniência, este tutorial usa uma instância Inf1 para compilação e inferência. Na prática, você pode compilar o modelo usando outro tipo de instância, como a família de instâncias c5. Depois,

você deve implantar o modelo compilado no servidor de inferência Inf1. Para obter mais informações, consulte a documentação do AWS Neuron PyTorch SDK.

Conteúdos

Pré-requisitos

Antes de usar este tutorial, você precisa ter concluído os passos da configuração em Lançamento de uma instância DLAMI com Neuron AWS. Também é necessário conhecer a aprendizagem profunda e o uso da DLAMI.

Ative o ambiente Conda

Ative o ambiente PyTorch -Neuron conda usando o seguinte comando:

```
source activate aws_neuron_pytorch_p36
```

Para sair do ambiente Conda atual, execute:

```
source deactivate
```

Compilação ResNet50

Crie um script Python chamado **`pytorch_trace_resnet50.py`** com o conteúdo a seguir. Esse script usa a API Python de compilação PyTorch -Neuron para compilar um modelo -50. ResNet

> **Note**
>
> Há uma dependência entre as versões do torchvision e do pacote torch que você deve conhecer ao compilar os modelos do torchvision. Essas regras de dependência podem ser gerenciadas por meio do pip. Torchvision==0.6.1 corresponde à versão torch==1.5.1, enquanto torchvision==0.8.2 corresponde à versão 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")
```

Execute o script de compilação.

```
python pytorch_trace_resnet50.py
```

A compilação demora alguns minutos. Quando terminar, o modelo compilado será salvo como `resnet50_neuron.pt` no diretório local.

ResNet50 Inferência

Crie um script Python chamado **`pytorch_infer_resnet50.py`** com o conteúdo a seguir. Esse script faz download de uma imagem de amostra e a usa para executar a inferência com o 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) )
```

Execute a inferência com o modelo compilado usando o seguinte comando:

```
python pytorch_infer_resnet50.py
```

A saída será semelhante a:

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

A DLAMI do Graviton

AWS As DLAMIs de GPU Graviton foram projetadas para fornecer alto desempenho e economia para cargas de trabalho de aprendizado profundo. Especificamente, o tipo de instância G5g apresenta o processador AWS Graviton2 baseado em ARM, que foi desenvolvido do zero AWS e otimizado para a forma como os clientes executam suas cargas de trabalho na nuvem. AWS As DLAMIs de GPU Graviton são pré-configuradas com Docker, NVIDIA Docker, NVIDIA Driver, CUDA, cuDNN, NCCL e TensorRT, além de estruturas populares de aprendizado de máquina, como e. TensorFlow PyTorch

Com o tipo de instância G5g, você pode aproveitar os benefícios de preço e desempenho do Graviton2 para implantar modelos de aprendizado profundo acelerados por GPU a um custo significativamente menor em comparação com instâncias que usam como base x86 com aceleração de GPU.

Selecionar uma DLAMI do Graviton

Execute uma instância G5g com a DLAMI do Graviton de sua escolha.

Para step-by-step obter instruções sobre como iniciar uma DLAMI, consulte Iniciando e configurando uma DLAMI.

Para obter uma lista das DLAMIs mais recentes do Graviton, consulte as Notas de versão da DLAMI.

Conceitos básicos

Os tópicos a seguir mostram como começar a usar a DLAMI do Graviton.

Conteúdo

Como usar uma DLAMI de GPU do Graviton

AWS Deep Learning AMI Está pronto para uso com as GPUs Graviton baseadas no processador Arm. A DLAMI de GPU do Graviton é fornecida com uma plataforma base de drivers de GPU e bibliotecas de aceleração para implantar seu próprio ambiente personalizado de aprendizado profundo. O Docker e o NVIDIA Docker são pré-configurados na DLAMI da GPU do Graviton, permitindo que você implante aplicativos em contêineres. Confira as notas de versão para obter detalhes adicionais sobre a DLAMI da GPU do Graviton.

Conteúdo

Verificar o status da GPU

Use a interface de gerenciamento do sistema NVIDIA para verificar o status da GPU do Graviton.

```
nvidia-smi
```

A saída do comando `nvidia-smi` será semelhante ao mostrado a seguir:

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

Verificar a versão do CUDA

Execute o comando a seguir para verificar a versão do CUDA:

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

Sua saída deve ser semelhante à seguinte:

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

Verificar Docker

Execute um contêiner CUDA DockerHubpara verificar a funcionalidade do Docker em sua GPU Graviton:

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

Sua saída deve ser semelhante à seguinte:

```
+-----------------------------------------------------------------------------+
| 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 o comando a seguir para acessar a ferramenta de linha de comando do TensorRT:

```
trtexec
```

Sua saída deve ser semelhante à seguinte:

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

Existem wheels do Python para TensorRT disponíveis, com instalação sob demanda. Você pode encontrar os wheels nos seguintes locais de arquivos:

```
/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 obter detalhes adicionais, consulte a documentação do NVIDIA TensorRT.

Executar amostras do CUDA

A DLAMI da GPU do Graviton fornece amostras do CUDA pré-compiladas para ajudar você a verificar diferentes funcionalidades do CUDA.

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

Por exemplo, execute a amostra do `vectorAdd` com o seguinte comando:

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

Sua saída deve ser semelhante à seguinte:

```
[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
```

Execute a amostra do `transpose`:

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

Sua saída deve ser semelhante à seguinte:

```
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
```

A seguir

Usando a GPU Graviton DLAMI TensorFlow

Usando a GPU Graviton DLAMI TensorFlow

O AWS Deep Learning AMI está pronto para uso com as GPUs Graviton baseadas no processador Arm e é otimizado para o. TensorFlow O Graviton GPU TensorFlow DLAMI inclui um ambiente Python pré-configurado TensorFlow com o Serving para casos de uso de inferência de aprendizado profundo. Confira as notas de lançamento para obter detalhes adicionais sobre a DLAMI da TensorFlow GPU Graviton.

Conteúdo

- Verifique a disponibilidade do TensorFlow serviço
- Verificar TensorFlow e fornecer TensorFlow a disponibilidade da API
- Execute um exemplo de inferência com TensorFlow o Serving

Verifique a disponibilidade do TensorFlow serviço

Execute o comando a seguir para verificar a disponibilidade e a versão do TensorFlow Serving:

```
tensorflow_model_server --version
```

Sua saída deve ser semelhante à seguinte:

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

```
TensorFlow Library: 2.8.0
```

Verificar TensorFlow e fornecer TensorFlow a disponibilidade da API

Execute o comando a seguir para verificar a disponibilidade TensorFlow e a TensorFlow Serving API:

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

Se o comando for bem-sucedido, não haverá resultado.

Execute um exemplo de inferência com TensorFlow o Serving

Use os comandos a seguir para baixar um modelo ResNet 50 pré-treinado e executar inferência usando o TensorFlow Serving:

```
# 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" &
```

Sua saída deve ser semelhante à seguinte:

```
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 ...
```

Use o `resnet_client` exemplo de TensorFlow Serving para executar a inferência:

```
python3 serving/tensorflow_serving/example/resnet_client.py
```

Sua saída deve ser semelhante à seguinte:

```
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
```

Pare de TensorFlow servir com o seguinte comando:

```
kill $(pidof tensorflow_model_server)
```

A seguir

Usando a GPU Graviton DLAMI PyTorch

Usando a GPU Graviton DLAMI PyTorch

O AWS Deep Learning AMI está pronto para uso com as GPUs Graviton baseadas no processador Arm e é otimizado para o. PyTorch O Graviton GPU PyTorch DLAMI inclui um ambiente Python pré-configurado PyTorchcom TorchVision, e para casos de uso de treinamento TorchServee inferência de aprendizado profundo. Confira as notas de lançamento para obter detalhes adicionais sobre a DLAMI da PyTorch GPU Graviton.

Conteúdo

- Verifique o PyTorch ambiente Python
- Execute uma amostra de treinamento com PyTorch
- Execute uma amostra de inferência com PyTorch

Verifique o PyTorch ambiente Python

Conecte-se à sua instância G5g e ative o ambiente básico do Conda com o seguinte comando:

```
source activate base
```

Seu prompt de comando deve indicar que você está trabalhando no ambiente básico do Conda, que contém PyTorch TorchVision, e outras bibliotecas.

```
(base) $
```

Verifique os caminhos de ferramentas padrão do PyTorch ambiente:

```
(base) $ which python
/opt/conda/bin/python

(base) $ which pip
/opt/conda/bin/pip

(base) $ which conda
/opt/conda/bin/conda

(base) $ which mamba
/opt/conda/bin/mamba
```

Verifique se o Torch e o Torch TorchVersion estão disponíveis, verifique suas versões e teste a funcionalidade 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)
```

Execute uma amostra de treinamento com PyTorch

Execute uma amostra de trabalho de treinamento do MNIST:

```
git clone https://github.com/pytorch/examples.git
cd examples/mnist
python main.py
```

Sua saída deve ser semelhante à seguinte:

```
...
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%)
```

Execute uma amostra de inferência com PyTorch

Use os comandos a seguir para baixar um modelo densenet161 pré-treinado e executar a inferência usando: 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
```

Sua saída deve ser semelhante à seguinte:

```
{
  "tiger_cat": 0.4693363308906555,
  "tabby": 0.4633873701095581,
  "Egyptian_cat": 0.06456123292446136,
  "lynx": 0.0012828150065615773,
  "plastic_bag": 0.00023522898778133094
}
```

Use os comandos a seguir para cancelar o registro do modelo densenet161 e parar o servidor:

```
curl -X DELETE http://localhost:8081/models/densenet161/1.0
torchserve --stop
```

Sua saída deve ser semelhante à seguinte:

```
{
  "status": "Model \"densenet161\" unregistered"
}
TorchServe has stopped.
```

A DLAMI Habana

As instâncias com aceleradores Habana foram projetadas para fornecer alto desempenho e economia para workloads de treinamento dos modelos de aprendizado profundo. Especificamente, os tipos de instância DL1 usam aceleradores Habana Gaudi da Habana Labs, uma empresa da Intel. As instâncias com aceleradores Habana são configuradas com o software Habana SynapseAI e pré-integradas a estruturas populares de aprendizado de máquina, como e. TensorFlow PyTorch

Os tópicos a seguir mostram como começar a usar o hardware do Habana Gaudi com a DLAMI.

Conteúdo

Como iniciar uma DLAMI

A DLAMI mais recente é pronta para uso com os aceleradores Habana Gaudi. Use as etapas a seguir para iniciar a DLAMI Habana e garantir que os recursos específicos do Python e da estrutura estejam ativos. Para obter recursos adicionais de configuração, consulte o repositório de Configuração e instalação do Habana Gaudi.

Conteúdo

Selecionar uma DLAMI Habana

Execute uma instância DL1 com a DLAMI Habana de sua escolha.

Para step-by-step obter instruções sobre como iniciar uma DLAMI, consulte Iniciando e configurando uma DLAMI.

Para obter uma lista dos DLAMIs mais recentes do Habana, consulte as notas de versão da DLAMI.

Ativar o ambiente do Python

Conecte-se à sua instância DL1 e ative o ambiente Python recomendado para a DLAMI Habana. Para verificar seu ambiente Python recomendado, selecione a DLAMI nas Notas de versão.

Importar a estrutura de machine learning

As instâncias com aceleradores Habana são pré-integradas a estruturas populares de aprendizado de máquina, como e. TensorFlow PyTorch Importe a estrutura de machine learning escolhida.

Importar TensorFlow

Para usar TensorFlow em seu Habana DLAMI, navegue até a pasta do ambiente Python que você ativou e importou. TensorFlow

```
/usr/bin/$PYTHON_VERSION
import tensorflow
```

```
tensorflow.__version__
```

Para verificar a TensorFlow versão compatível com seu Habana DLAMI, selecione seu DLAMI nas notas de lançamento.

Importar PyTorch

Para usar PyTorch em seu Habana DLAMI, navegue até a pasta do ambiente Python que você ativou e importe a versão apropriada. PyTorch

```
/usr/bin/$PYTHON_VERSION
import torch
torch.__version__
```

Para verificar a PyTorch versão compatível com seu Habana DLAMI, selecione seu DLAMI nas notas de lançamento.

Para obter mais informações sobre como executar e treinar modelos de aprendizado de máquina em TensorFlow e PyTorch usando seu Habana DLAMI, consulte o repositório Habana Model References. GitHub Para ver mais recursos sobre como trabalhar com a DLAMI Habana, acesse a documentação do Habana Gaudi.

Inferência

Nesta seção você encontra tutoriais sobre como executar inferências usando as estruturas e ferramentas da DLAMI.

Para ver tutoriais sobre o uso do Elastic Inference, consulte Como trabalhar com o Amazon Elastic Inference

Inferências com estruturas

- Usar o Apache MXNet (em incubação) para inferência com um modelo ONNX
- Use o Apache MXNet (incubação) para inferência com um modelo 5.0 ResNet
- Usar o CNTK para inferência com um modelo ONNX

Ferramentas de inferência

- Model Server para Apache MXNet (MMS)

- TensorFlow Servindo

Usar o Apache MXNet (em incubação) para inferência com um modelo ONNX

Como usar o modelo do ONNX para inferência de imagem com o Apache MXNet (em incubação)

1.
   - (Opção para Python 3): ative o ambiente Python 3 do Apache MXNet (em incubação):

     ```
     $ source activate mxnet_p36
     ```

   - (Opção para Python 2): ative o ambiente Python 2 do Apache MXNet (em incubação):

     ```
     $ source activate mxnet_p27
     ```

2. As etapas restantes assumem que você está usando o ambiente do `mxnet_p36`.
3. Faça download de uma imagem de um husky.

   ```
   $ curl -O https://upload.wikimedia.org/wikipedia/commons/b/b5/Siberian_Husky_bi-
   eyed_Flickr.jpg
   ```

4. Faça download de uma lista de classes que funcionarão com esse modelo.

   ```
   $ curl -O https://gist.githubusercontent.com/yrevar/6135f1bd8dcf2e0cc683/raw/
   d133d61a09d7e5a3b36b8c111a8dd5c4b5d560ee/imagenet1000_clsid_to_human.pkl
   ```

5. Faça download do modelo VGG 16 pré-treinado no formato ONNX.

   ```
   $ wget -O vgg16.onnx https://github.com/onnx/models/raw/master/vision/
   classification/vgg/model/vgg16-7.onnx
   ```

6. Use um editor de texto de sua preferência para criar um script com o conteúdo a seguir. Esse script usará a imagem do husky, obterá um resultado de previsão do modelo pré-treinado e o analisará no arquivo de classes, retornando um resultado de classificação de imagem.

   ```
   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. Em seguida, execute o script e você deve ver um resultado da seguinte forma:

```
248
Eskimo dog, husky
```

Use o Apache MXNet (incubação) para inferência com um modelo 5.0 ResNet

Como usar um modelo Apache MXNet (em incubação) pré-treinado com a API Symbol para inferência de imagem com MXNet

1. • (Opção para Python 3): ative o ambiente Python 3 do Apache MXNet (em incubação):

   ```
   $ source activate mxnet_p36
   ```

   • (Opção para Python 2): ative o ambiente Python 2 do Apache MXNet (em incubação):

   ```
   $ source activate mxnet_p27
   ```

2. As etapas restantes assumem que você está usando o ambiente do `mxnet_p36`.

3. Use um editor de texto de sua preferência para criar um script com o conteúdo a seguir. Esse script baixará os arquivos de modelo ResNet -50 (resnet-50-0000.params e resnet-50-symbol.json) e a lista de rótulos (synset.txt), baixará uma imagem de gato para obter um resultado de previsão do modelo pré-treinado e, em seguida, pesquisará o resultado na lista de rótulos, retornando um resultado de previsão.

   ```
   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. Em seguida, execute o script e você deve ver um resultado da seguinte forma:

```
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
```

Usar o CNTK para inferência com um modelo ONNX

> **Note**
>
> Não incluímos mais os ambientes CNTK, Caffe, Caffe2 e Theano Conda no AWS Deep Learning AMI desde a versão v28. As versões anteriores do AWS Deep Learning AMI que contêm esses ambientes continuarão disponíveis. No entanto, só forneceremos atualizações para esses ambientes se houver correções de segurança publicadas pela comunidade de código aberto dessas estruturas de trabalho.

> **Note**
>
> O modelo VGG-16 usado neste tutorial consome uma grande quantidade de memória. Ao selecionar sua AWS Deep Learning AMI instância, você pode precisar de uma instância com mais de 30 GB de RAM.

Como usar um modelo ONNX para inferência com CNTK

1.
   - (Opção para Python 3) - ative o ambiente Python 3 CNTK:

     ```
     $ source activate cntk_p36
     ```

   - (Opção para Python 2) - ative o ambiente Python 2 CNTK:

     ```
     $ source activate cntk_p27
     ```

2. As etapas restantes assumem que você está usando o ambiente do `cntk_p36`.
3. Crie um novo arquivo com o editor de texto e use o programa a seguir em um script para abrir o arquivo no formato ONNX no 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!")
   ```

   Depois de executar esse script, CNTK terá carregado o modelo.
4. Você também pode experimentar a execução de inferência com CNTK. Primeiro, faça download de uma imagem de um husky.

   ```
   $ curl -O https://upload.wikimedia.org/wikipedia/commons/b/b5/Siberian_Husky_bi-
   eyed_Flickr.jpg
   ```

5. Em seguida, faça download de uma lista de classes que funcionarão com esse modelo.

   ```
   $ curl -O https://gist.githubusercontent.com/yrevar/6135f1bd8dcf2e0cc683/raw/
   d133d61a09d7e5a3b36b8c111a8dd5c4b5d560ee/imagenet1000_clsid_to_human.pkl
   ```

6. Edite o script criado anteriormente para ter o seguinte conteúdo. Esta nova versão usará a imagem do husky, obtenha um resultado de previsão e, em seguida, procure isso no arquivo de classes, retornando um resultado de previsão.

```
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. Em seguida, execute o script e você deve ver um resultado da seguinte forma:

```
248
Eskimo dog, husky
```

Como usar Estruturas de acesso com ONNX

A AMI de deep learning com Conda agora oferece suporte a modelos Open Neural Network Exchange (ONNX) para algumas estruturas de acesso. Escolha um dos tópicos listados abaixo para saber como usar ONNX em sua AMI de deep learning com Conda.

Se você quiser usar um modelo ONNX existente em uma DLAMI, consulte Usar o Apache MXNet (em incubação) para inferência com um modelo ONNX.

Sobre as ONNX

O Open Neural Network Exchange (ONNX) é um formato aberto usado para representar modelos de aprendizado profundo. O ONNX tem suporte da Amazon Web Services, da Microsoft, do Facebook e de vários outros parceiros. Você pode projetar, treinar e implantar modelos de aprendizado profundo com qualquer estrutura que você escolher. O benefício de modelos ONNX é que eles podem ser movidos entre as estruturas com facilidade.

A AMI de deep learning com Conda atualmente destaca alguns do recursos de ONNX na seguinte coleção de tutoriais.

- Tutorial do Apache MXNet para ONNX para CNTK
- Tutorial do Chainer para ONNX para CNTK
- Tutorial do Chainer para ONNX para MXNet
- PyTorch Tutorial do ONNX para o CNTK
- PyTorch Tutorial do ONNX para o MXNet

Você também pode consultar a documentação e tutoriais do projeto ONNX:

- Projeto ONNX em GitHub
- Tutoriais do ONNX

Tutorial do Apache MXNet para ONNX para CNTK

> **Note**
>
> Não incluímos mais os ambientes CNTK, Caffe, Caffe2 e Theano Conda no AWS Deep Learning AMI desde a versão v28. As versões anteriores do AWS Deep Learning AMI que contêm esses ambientes continuarão disponíveis. No entanto, só forneceremos atualizações para esses ambientes se houver correções de segurança publicadas pela comunidade de código aberto dessas estruturas de trabalho.

Visão geral da ONNX

O Open Neural Network Exchange (ONNX) é um formato aberto usado para representar modelos de aprendizado profundo. O ONNX tem suporte da Amazon Web Services, da Microsoft, do Facebook e

de vários outros parceiros. Você pode projetar, treinar e implantar modelos de aprendizado profundo com qualquer estrutura que você escolher. O benefício de modelos ONNX é que eles podem ser movidos entre as estruturas com facilidade.

Este tutorial mostra como usar a AMI de deep learning com o Conda e o ONNX. Ao seguir essas etapas, você poderá treinar um modelo ou carregar um modelo pré-treinado de um framework, exportar esse modelo para ONNX e, em seguida, importar o modelo em outro framework.

Pré-requisitos do ONNX

Para usar esse tutorial de ONNX, você deve ter acesso à versão 12 ou posterior da AMI de deep learning com Conda. Para obter mais informações sobre como começar a usar a AMI de deep learning com Conda, consulte AMI de deep learning com Conda.

> **Important**
>
> Esses exemplos usam funções que podem exigir até 8 GB de memória (ou mais). Certifique-se de escolher um tipo de instância com memória suficiente.

Inicie uma sessão de terminal com sua AMI de deep learning com Conda para iniciar o tutorial a seguir.

Converta um modelo Apache MXNet (em incubação) para ONNX, em seguida carregue o modelo no CNTK

Como exportar um modelo do Apache MXNet (em incubação)

Você pode instalar a versão mais recente do MXNet construída em um ou ambos os ambientes MXNet Conda na AMI de deep learning com Conda.

1.
   - (Opção para Python 3) - ative o ambiente Python 3 MXNet:

     ```
     $ source activate mxnet_p36
     ```

   - (Opção para Python 2) - ative o ambiente Python 2 MXNet:

     ```
     $ source activate mxnet_p27
     ```

2. As etapas restantes assumem que você está usando o ambiente do `mxnet_p36`.

3. Fazer download dos arquivos de 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 os arquivos de modelo do MXNet para o formato ONNX, crie um novo arquivo com o editor de texto e use o programa a seguir em um 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)
```

   Algumas mensagens de aviso podem ser exibidas, mas você pode ignorá-las por enquanto. Depois de executar esse script, você verá o arquivo .onnx recém-criado no mesmo diretório.

5. Agora que você tem um arquivo ONNX, execute a inferência com ele usando o seguinte exemplo:

   - Usar o CNTK para inferência com um modelo ONNX

Tutoriais do ONNX

- Tutorial do Apache MXNet para ONNX para CNTK
- Tutorial do Chainer para ONNX para CNTK
- Tutorial do Chainer para ONNX para MXNet
- PyTorch Tutorial do ONNX para o MXNet
- PyTorch Tutorial do ONNX para o CNTK

Tutorial do Chainer para ONNX para CNTK

> **Note**
>
> Não incluímos mais os ambientes CNTK, Caffe, Caffe2 e Theano Conda no AWS Deep Learning AMI desde a versão v28. As versões anteriores do AWS Deep Learning AMI que contêm esses ambientes continuarão disponíveis. No entanto, só forneceremos atualizações para esses ambientes se houver correções de segurança publicadas pela comunidade de código aberto dessas estruturas de trabalho.

Visão geral da ONNX

O Open Neural Network Exchange (ONNX) é um formato aberto usado para representar modelos de aprendizado profundo. O ONNX tem suporte da Amazon Web Services, da Microsoft, do Facebook e de vários outros parceiros. Você pode projetar, treinar e implantar modelos de aprendizado profundo com qualquer estrutura que você escolher. O benefício de modelos ONNX é que eles podem ser movidos entre as estruturas com facilidade.

Este tutorial mostra como usar a AMI de deep learning com o Conda e o ONNX. Ao seguir essas etapas, você poderá treinar um modelo ou carregar um modelo pré-treinado de um framework, exportar esse modelo para ONNX e, em seguida, importar o modelo em outro framework.

Pré-requisitos do ONNX

Para usar esse tutorial de ONNX, você deve ter acesso à versão 12 ou posterior da AMI de deep learning com Conda. Para obter mais informações sobre como começar a usar a AMI de deep learning com Conda, consulte AMI de deep learning com Conda.

> **Important**
>
> Esses exemplos usam funções que podem exigir até 8 GB de memória (ou mais). Certifique-se de escolher um tipo de instância com memória suficiente.

Inicie uma sessão de terminal com sua AMI de deep learning com Conda para iniciar o tutorial a seguir.

Converta um Modelo Chainer para ONNX, depois carregue o modelo no CNTK

Primeiro, ative o ambiente Chainer:

```
$ source activate chainer_p36
```

Crie um novo arquivo com o editor de texto e use o programa a seguir em um script para obter um modelo de Zoo do modelo Chainer e, em seguida, exporte para o 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)
```

Depois de executar esse script, você verá o arquivo .onnx recém-criado no mesmo diretório.

Agora que você tem um arquivo ONNX, execute a inferência com ele usando o seguinte exemplo:

- Usar o CNTK para inferência com um modelo ONNX

Tutoriais do ONNX

- Tutorial do Apache MXNet para ONNX para CNTK

- Tutorial do Chainer para ONNX para CNTK
- Tutorial do Chainer para ONNX para MXNet
- PyTorch Tutorial do ONNX para o MXNet
- PyTorch Tutorial do ONNX para o CNTK

Tutorial do Chainer para ONNX para MXNet

Visão geral da ONNX

O Open Neural Network Exchange (ONNX) é um formato aberto usado para representar modelos de aprendizado profundo. O ONNX tem suporte da Amazon Web Services, da Microsoft, do Facebook e de vários outros parceiros. Você pode projetar, treinar e implantar modelos de aprendizado profundo com qualquer estrutura que você escolher. O benefício de modelos ONNX é que eles podem ser movidos entre as estruturas com facilidade.

Este tutorial mostra como usar a AMI de deep learning com o Conda e o ONNX. Ao seguir essas etapas, você poderá treinar um modelo ou carregar um modelo pré-treinado de um framework, exportar esse modelo para ONNX e, em seguida, importar o modelo em outro framework.

Pré-requisitos do ONNX

Para usar esse tutorial de ONNX, você deve ter acesso à versão 12 ou posterior da AMI de deep learning com Conda. Para obter mais informações sobre como começar a usar a AMI de deep learning com Conda, consulte AMI de deep learning com Conda.

> **Important**
>
> Esses exemplos usam funções que podem exigir até 8 GB de memória (ou mais). Certifique-se de escolher um tipo de instância com memória suficiente.

Inicie uma sessão de terminal com sua AMI de deep learning com Conda para iniciar o tutorial a seguir.

Converta um Modelo Chainer para ONNX, depois carregue o modelo no MXNet

Primeiro, ative o ambiente Chainer:

```
$ source activate chainer_p36
```

Crie um novo arquivo com o editor de texto e use o programa a seguir em um script para obter um modelo de Zoo do modelo Chainer e, em seguida, exporte para o 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)
```

Depois de executar esse script, você verá o arquivo .onnx recém-criado no mesmo diretório.

Agora que você tem um arquivo ONNX, execute a inferência com ele usando o seguinte exemplo:

- Usar o Apache MXNet (em incubação) para inferência com um modelo ONNX

Tutoriais do ONNX

- Tutorial do Apache MXNet para ONNX para CNTK
- Tutorial do Chainer para ONNX para CNTK
- Tutorial do Chainer para ONNX para MXNet
- PyTorch Tutorial do ONNX para o MXNet

- PyTorch Tutorial do ONNX para o CNTK

PyTorch Tutorial do ONNX para o CNTK

> **Note**
>
> Não incluímos mais os ambientes CNTK, Caffe, Caffe2 e Theano Conda no AWS Deep Learning AMI desde a versão v28. As versões anteriores do AWS Deep Learning AMI que contêm esses ambientes continuarão disponíveis. No entanto, só forneceremos atualizações para esses ambientes se houver correções de segurança publicadas pela comunidade de código aberto dessas estruturas de trabalho.

Visão geral da ONNX

O Open Neural Network Exchange (ONNX) é um formato aberto usado para representar modelos de aprendizado profundo. O ONNX tem suporte da Amazon Web Services, da Microsoft, do Facebook e de vários outros parceiros. Você pode projetar, treinar e implantar modelos de aprendizado profundo com qualquer estrutura que você escolher. O benefício de modelos ONNX é que eles podem ser movidos entre as estruturas com facilidade.

Este tutorial mostra como usar a AMI de deep learning com o Conda e o ONNX. Ao seguir essas etapas, você poderá treinar um modelo ou carregar um modelo pré-treinado de um framework, exportar esse modelo para ONNX e, em seguida, importar o modelo em outro framework.

Pré-requisitos do ONNX

Para usar esse tutorial de ONNX, você deve ter acesso à versão 12 ou posterior da AMI de deep learning com Conda. Para obter mais informações sobre como começar a usar a AMI de deep learning com Conda, consulte AMI de deep learning com Conda.

> **Important**
>
> Esses exemplos usam funções que podem exigir até 8 GB de memória (ou mais). Certifique-se de escolher um tipo de instância com memória suficiente.

Inicie uma sessão de terminal com sua AMI de deep learning com Conda para iniciar o tutorial a seguir.

Converta um PyTorch modelo em ONNX e, em seguida, carregue o modelo em CNTK

Primeiro, ative o PyTorch ambiente:

```
$ source activate pytorch_p36
```

Crie um novo arquivo com seu editor de texto e use o seguinte programa em um script para treinar um modelo simulado e PyTorch, em seguida, exporte-o para o 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)
```

Depois de executar esse script, você verá o arquivo .onnx recém-criado no mesmo diretório. Agora, alterne para o ambiente Conda CNTK para carregar o modelo com CNTK.

Depois, ative o ambiente CNTK:

```
$ source deactivate
$ source activate cntk_p36
```

Crie um novo arquivo com o editor de texto e use o programa a seguir em um script para abrir o arquivo no formato ONNX no 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)
```

Depois de executar esse script, CNTK terá carregado o modelo.

Você também pode exportar para ONNX usando CNTK anexando o seguinte ao script anterior e, em seguida, executando-o.

```
# Export the model to ONNX via the CNTK export API
z.save("cntk_model.onnx", format=C.ModelFormat.ONNX)
```

Tutoriais do ONNX

- Tutorial do Apache MXNet para ONNX para CNTK
- Tutorial do Chainer para ONNX para CNTK
- Tutorial do Chainer para ONNX para MXNet
- PyTorch Tutorial do ONNX para o MXNet
- PyTorch Tutorial do ONNX para o CNTK

PyTorch Tutorial do ONNX para o MXNet

Visão geral da ONNX

O Open Neural Network Exchange (ONNX) é um formato aberto usado para representar modelos de aprendizado profundo. O ONNX tem suporte da Amazon Web Services, da Microsoft, do Facebook e

de vários outros parceiros. Você pode projetar, treinar e implantar modelos de aprendizado profundo com qualquer estrutura que você escolher. O benefício de modelos ONNX é que eles podem ser movidos entre as estruturas com facilidade.

Este tutorial mostra como usar a AMI de deep learning com o Conda e o ONNX. Ao seguir essas etapas, você poderá treinar um modelo ou carregar um modelo pré-treinado de um framework, exportar esse modelo para ONNX e, em seguida, importar o modelo em outro framework.

Pré-requisitos do ONNX

Para usar esse tutorial de ONNX, você deve ter acesso à versão 12 ou posterior da AMI de deep learning com Conda. Para obter mais informações sobre como começar a usar a AMI de deep learning com Conda, consulte AMI de deep learning com Conda.

> **Important**
>
> Esses exemplos usam funções que podem exigir até 8 GB de memória (ou mais). Certifique-se de escolher um tipo de instância com memória suficiente.

Inicie uma sessão de terminal com sua AMI de deep learning com Conda para iniciar o tutorial a seguir.

Converter um PyTorch modelo em ONNX e, em seguida, carregar o modelo no MXNet

Primeiro, ative o PyTorch ambiente:

```
$ source activate pytorch_p36
```

Crie um novo arquivo com seu editor de texto e use o seguinte programa em um script para treinar um modelo simulado e PyTorch, em seguida, exporte-o para o 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!")
```

Depois de executar esse script, você verá o arquivo .onnx recém-criado no mesmo diretório. Agora, alterne para o ambiente Conda MXNet para carregar o modelo com MXNet.

Depois, ative o ambiente MXNet:

```
$ source deactivate
$ source activate mxnet_p36
```

Crie um novo arquivo com o editor de texto e use o programa a seguir em um script para abrir o arquivo no formato ONNX no 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())
```

Depois de executar esse script, o MXNet terá carregado o modelo e imprimirá algumas informações básicas de modelo.

Tutoriais do ONNX

- Tutorial do Apache MXNet para ONNX para CNTK
- Tutorial do Chainer para ONNX para CNTK
- Tutorial do Chainer para ONNX para MXNet
- PyTorch Tutorial do ONNX para o MXNet
- PyTorch Tutorial do ONNX para o CNTK

Fornecimento de modelos

Veja a seguir as opções de fornecimento de modelos instaladas na AMI de deep learning com Conda. Clique em uma das opções para saber como usá-la.

Tópicos

- Model Server para Apache MXNet (MMS)
- TensorFlow Servindo
- TorchServe

Model Server para Apache MXNet (MMS)

O Model Server for Apache MXNet (MMS) é uma ferramenta flexível que fornece modelos de aprendizado profundos exportados do Apache MXNet (incubadora) ou exportados para um formato de modelo Open Neural Network Exchange (ONNX) O MMS vem pré-instalado com a DLAMI com Conda. Este tutorial para MMS demonstrará como fornecer um modelo de classificação de imagem.

Tópicos

- Fornecer um modelo de classificação de imagem no MMS
- Outros exemplos
- Mais informações

Fornecer um modelo de classificação de imagem no MMS

Este tutorial mostra como fornecer um modelo de classificação de imagem com o MMS. O modelo é fornecido por meio do MMS Model Zoo e é baixado automaticamente quando você inicia o MMS. Quando o servidor está em execução, ele recebe solicitações de previsão. Quando você carrega uma imagem, neste caso, uma imagem de um gatinho, o servidor retorna uma estimativa das cinco principais classes correspondentes das 1.000 classes em que o modelo foi treinado. Mais informações sobre os modelos, como eles foram treinados e como testá-los podem ser encontradas no MMS Model Zoo.

Para fornecer um exemplo de modelo de classificação de imagem no MMS

1. Conecte-se a uma instância do Amazon Elastic Compute Cloud (Amazon EC2) da AMI de deep learning com o Conda.
2. Ative um ambiente do MXNet:
   - Para MXNet e Keras 2 no Python 3 com CUDA 9.0 e MKL-DNN, execute este comando:

     ```
     $ source activate mxnet_p36
     ```

   - Para MXNet e Keras 2 no Python 2 com CUDA 9.0 e MKL-DNN, execute este comando:

     ```
     $ source activate mxnet_p27
     ```

3. Execute o MMS com o comando a seguir. Adicione `> /dev/null` para silenciar a saída de registro enquanto você executa outros testes.

   ```
   $ mxnet-model-server --start > /dev/null
   ```

   Agora, o MMS está em execução no host e está recebendo solicitações de inferência.
4. Em seguida, use um comando curl para administrar endpoints de gerenciamento do MMS e informe ao comando qual modelo você quer que ele forneça.

   ```
   $ 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. O MMS precisa saber o número de operadores que você pretende usar. Para este teste, use 3.

   ```
   $ curl -v -X PUT "http://localhost:8081/models/squeezenet_v1.1?min_worker=3"
   ```

6. Faça download de uma imagem de um gatinho e a envie para o endpoint de previsão 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
```

O endpoint de previsão retorna uma previsão em JSON semelhante às cinco principais previsões a seguir, em que a imagem tem uma probabilidade de 94% de conter um gato egípcio, seguida por uma chance de 5,5% de ter um lince ou uma onça-parda:


```
{
 "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. Teste algumas outras imagens ou, se você tiver concluído o teste, interrompa o servidor:

```
$ mxnet-model-server --stop
```

Este tutorial foca no fornecimento do modelo básico. O MMS também é compatível usando inferência elástica com fornecimento do modelo. Para obter mais informações, consulte Fornecimento do modelo com o Amazon Elastic Inference

Quando você estiver pronto para aprender mais sobre outros recursos do MMS, consulte a documentação do MMS em. GitHub

Outros exemplos

O MMS tem uma variedade de exemplos que você pode executar na DLAMI. Você pode visualizá-los no repositório do projeto do MMS.

Mais informações

Para obter mais documentação do MMS, incluindo como configurar o MMS com o Docker, ou para aproveitar os recursos mais recentes do MMS, marque a página do projeto MMS com uma estrela. GitHub

TensorFlow Servindo

TensorFlow O Serving é um sistema de atendimento flexível e de alto desempenho para modelos de aprendizado de máquina.

O `tensorflow-serving-api` vem pré-instalado com a AMI de deep learning com o Conda. Você encontrará scripts de exemplo para treinar, exportar e fornecer um modelo MNIST em `~/examples/tensorflow-serving/`.

Para executar qualquer um desses exemplos, primeiro conecte-se à sua AMI de aprendizado profundo com o Conda e ative o TensorFlow ambiente.

```
$ source activate tensorflow_p37
```

Agora, altere os diretórios para a pasta de scripts de exemplo de fornecimento.

```
$ cd ~/examples/tensorflow-serving/
```

Fornecer um modelo de origem pré-treinado

Veja a seguir um exemplo que você pode seguir para fornecer diferentes modelos como origem. Como regra geral, você precisa de um script de modelo e cliente que pode ser fornecido que já possa ser obtido por download na DLAMI.

Forneça e teste a inferência com um modelo de início

1. Faça download do modelo.

```
$ curl -O https://s3-us-west-2.amazonaws.com/tf-test-models/INCEPTION.zip
```

2. Descompacte o modelo.

```
$ unzip INCEPTION.zip
```

3. Faça download de uma imagem de um husky.

```
$ curl -O https://upload.wikimedia.org/wikipedia/commons/b/b5/Siberian_Husky_bi-
eyed_Flickr.jpg
```

4. Inicie o servidor. Observe que, para o Amazon Linux, você deve alterar o diretório usado para `model_base_path`, de `/home/ubuntu` para `/home/ec2-user`.

```
$ tensorflow_model_server --model_name=INCEPTION --model_base_path=/home/ubuntu/
examples/tensorflow-serving/INCEPTION/INCEPTION --port=9000
```

5. Com o servidor em execução em primeiro plano, é necessário iniciar outra sessão de terminal para continuar. Abra um novo terminal e ative TensorFlow com`source activate tensorflow_p37`. Em seguida, use um editor de texto de sua preferência para criar um script com o conteúdo a seguir. Chame-o de `inception_client.py`. Esse script usará um nome de arquivo de imagem como um parâmetro e obterá o resultado da previsão do modelo pré-treinado.

```
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. Agora execute o script passando o local do servidor, a porta e o nome do arquivo da fotografia do husky como parâmetros.

```
$ python3 inception_client.py --server=localhost:9000 --image Siberian_Husky_bi-
eyed_Flickr.jpg
```

Treinar e fornecer um modelo do MNIST

Neste tutorial, vamos exportar um modelo e, em seguida, fornecê-lo com o aplicativo `tensorflow_model_server`. Finalmente, você pode testar o servidor modelo com um exemplo de script do cliente.

Execute o script que treina e exporta um modelo MNIST. Como único argumento do script, você precisa fornecer uma pasta local para salvar o modelo. Por enquanto, podemos colocá-lo em `mnist_model`. O script criará a pasta para você.

```
$ python mnist_saved_model.py /tmp/mnist_model
```

Aguarde. Este script pode demorar um pouco antes de fornecer resultados. Ao concluir o treinamento e, por fim, exportar o modelo, você verá o seguinte:

```
Done training!
Exporting trained model to mnist_model/1
Done exporting!
```

A próxima etapa é executar o `tensorflow_model_server` para fornecer o modelo exportado.

```
$ tensorflow_model_server --port=9000 --model_name=mnist --model_base_path=/tmp/
mnist_model
```

Um script de cliente é disponibilizado para testar o servidor.

Para testá-lo, você precisa abrir uma nova janela no terminal.

```
$ python mnist_client.py --num_tests=1000 --server=localhost:9000
```

Outros recursos e exemplos

Se você estiver interessado em saber mais sobre o TensorFlow Serving, confira o TensorFlow site.

Você também pode usar o TensorFlow Serving with Amazon Elastic Inference. Confira o guia sobre como usar o Elastic Inference with TensorFlow Serving para obter mais informações.

TorchServe

TorchServe é uma ferramenta flexível para servir modelos de aprendizado profundo que foram exportados do PyTorch. TorchServe vem pré-instalado com a AMI de aprendizado profundo com Conda a partir da v34.

Para obter mais informações sobre o uso TorchServe, consulte Model Server for PyTorch Documentation.

Tópicos

Ofereça um modelo de classificação de imagens em TorchServe

Este tutorial mostra como servir um modelo de classificação de imagens com TorchServe o. Ele usa um modelo DenseNet -161 fornecido pela. PyTorch Quando o servidor está em execução, ele escuta as solicitações de previsão. Quando você carrega uma imagem, neste caso, uma imagem de um gatinho, o servidor retorna uma estimativa das cinco principais classes correspondentes das classes em que o modelo foi treinado.

Para fornecer um exemplo de modelo de classificação de imagens em TorchServe

1. Conecte-se a uma instância do Amazon Elastic Compute Cloud (Amazon EC2) da AMI de deep learning com o Conda v34 ou posterior.
2. Ative o ambiente `pytorch_latest_p36`.

   ```
   source activate pytorch_latest_p36
   ```

3. Clone o TorchServe repositório e crie um diretório para armazenar seus modelos.

   ```
   git clone https://github.com/pytorch/serve.git
   mkdir model_store
   ```

4. Arquive o modelo usando o arquivador de modelos. O `extra-files` parâmetro usa um arquivo do `TorchServe` repositório, portanto, atualize o caminho, se necessário. Para obter mais informações sobre o arquivador de modelos, consulte Torch Model archiver for. TorchServe

   ```
   wget https://download.pytorch.org/models/densenet161-8d451a50.pth
   torch-model-archiver --model-name densenet161 --version 1.0 --model-file ./
   serve/examples/image_classifier/densenet_161/model.py --serialized-file
    densenet161-8d451a50.pth --export-path model_store --extra-files ./serve/examples/
   image_classifier/index_to_name.json --handler image_classifier
   ```

5. Execute TorchServe para iniciar um endpoint. A adição de `> /dev/null` silencia a saída do log.

```
torchserve --start --ncs --model-store model_store --models densenet161.mar > /dev/
null
```

6. Baixe uma imagem de um gatinho e envie-a para o endpoint de TorchServe previsão:

```
curl -O https://s3.amazonaws.com/model-server/inputs/kitten.jpg
curl http://127.0.0.1:8080/predictions/densenet161 -T kitten.jpg
```

   O endpoint de previsão retorna uma previsão em JSON semelhante às cinco principais previsões a seguir, em que a imagem tem uma probabilidade de 47% de conter um gato egípcio, seguida por uma chance de 46% de ter um gato malhado.

```
{
 "tiger_cat": 0.46933576464653015,
 "tabby": 0.463387668132782,
 "Egyptian_cat": 0.0645613968372345,
 "lynx": 0.0012828196631744504,
 "plastic_bag": 0.00023323058849200606
}
```

7. Ao terminar o teste, interrompa o servidor.

```
torchserve --stop
```

Outros exemplos

TorchServe tem vários exemplos que você pode executar em sua instância DLAMI. Você pode visualizá-los na página de exemplos TorchServe do repositório do projeto.

Mais informações

Para obter mais TorchServe documentação, incluindo como configurar TorchServe o Docker e os TorchServe recursos mais recentes, consulte a página do TorchServe projeto em GitHub.

Como atualizar a DLAMI

Aqui você encontrará informações sobre como atualizar a DLAMI e dicas sobre como atualizar o software na DLAMI.

Tópicos

Atualização para a nova versão da DLAMI

As imagens do sistema da DLAMI são atualizadas regularmente para aproveitar as novas versões da estrutura de aprendizado profundo, CUDA e outras atualizações de softwares e ajustes de desempenho. Se estiver usando uma DLAMI há um certo tempo e deseja aproveitar uma atualização, você precisaria executar uma nova instância. Também seria necessário transferir manualmente todos os conjuntos de dados, pontos de verificação ou outros dados importantes. Em vez disso, você pode usar o Amazon EBS para reter seus dados e associá-los a uma nova DLAMI. Dessa forma, você pode fazer atualizações com frequência e, ao mesmo tempo, minimizar o tempo necessário para transferir os dados.

> **Note**
>
> Ao associar e mover volumes do Amazon EBS entre DLAMIs, você deve ter as DLAMIs e o novo volume na mesma zona de disponibilidade.

1. Use o console do Amazon EC2 para criar um novo volume do Amazon EBS. Para obter instruções detalhadas, consulte Criação de um volume do Amazon EBS.
2. Associe o volume do Amazon EBS recém-criado à sua DLAMI existente. Para obter instruções detalhadas, consulte Associação de um volume do Amazon EBS.
3. Transfira seus dados, como conjuntos de dados, pontos de verificação e arquivos de configuração.
4. Execute uma DLAMI. Para obter instruções detalhadas, consulte Como executar e configurar uma DLAMI.

5. Desassocie o volume do Amazon EBS da sua antiga DLAMI. Para obter instruções detalhadas, consulte Desassociação de um volume do Amazon EBS.
6. Associe o volume do Amazon EBS à sua nova DLAMI. Siga as instruções da etapa 2 para associar o volume.
7. Após verificar se seus dados estão disponíveis na nova DLAMI, interrompa e encerre a antiga. Para obter instruções detalhadas para limpeza, consulte Limpar.

Dicas para as atualizações de software

Periodicamente, você pode atualizar manualmente o software em sua DLAMI. É recomendável usar `pip` para atualizar pacotes Python. Você também deve usar `pip` para atualizar pacotes em um ambiente Conda na AMI de deep learning com Conda. Consulte o site do software ou do framework específico para obter instruções de atualização e instalação.

> **Note**
>
> Não podemos garantir que uma atualização de pacote será bem-sucedida. A tentativa de atualizar um pacote em um ambiente com dependências incompatíveis pode resultar em uma falha. Nesse caso, você deve entrar em contato com o responsável pela biblioteca para conferir se é possível atualizar as dependências do pacote. Como alternativa, é possível tentar modificar o ambiente para permitir a atualização. No entanto, essa modificação provavelmente significará remover ou atualizar os pacotes existentes, ou seja, não poderemos mais garantir a estabilidade desse ambiente.

Se tiver interesse em executar a ramificação principal mais recente de um determinado pacote, ative o ambiente apropriado e, em seguida, adicione `--pre` ao final do comando `pip install --upgrade`. Por exemplo:

```
source activate mxnet_p36
pip install --upgrade mxnet --pre
```

Há muitos ambientes Conda e pacotes pré-instalados na AWS Deep Learning AMI. Devido ao número de pacotes pré-instalados, é difícil encontrar um conjunto de pacotes com garantia de compatibilidade. Você pode ver um aviso “O ambiente é inconsistente, verifique o plano do pacote com cuidado”. A DLAMI garante que todos os ambientes fornecidos por ela estejam corretos, mas não pode garantir que nenhum pacote instalado pelo usuário funcione corretamente.

Segurança em AWS Deep Learning AMI

A segurança na nuvem AWS é a maior prioridade. Como AWS cliente, você se beneficia de uma arquitetura de data center e rede criada para atender aos requisitos das organizações mais sensíveis à segurança.

A segurança é uma responsabilidade compartilhada entre você AWS e você. O modelo de responsabilidade compartilhada descreve isso como a segurança da nuvem e segurança na nuvem:

- Segurança da nuvem — AWS é responsável por proteger a infraestrutura que executa AWS os serviços na AWS nuvem. AWS também fornece serviços que você pode usar com segurança. Auditores terceirizados testam e verificam regularmente a eficácia de nossa segurança como parte dos Programas de Conformidade Programas de AWS de . Para saber mais sobre os programas de conformidade que se aplicam ao DLAMI, AWS consulte Serviços no escopo do programa de conformidade Serviços no escopo AWS de conformidade.
- Segurança na nuvem — Sua responsabilidade é determinada pelo AWS serviço que você usa. Você também é responsável por outros fatores, incluindo a confidencialidade de seus dados, os requisitos da empresa e as leis e regulamentos aplicáveis.

Esta documentação ajuda a entender como aplicar o modelo de responsabilidade compartilhada ao usar a DLAMI. Os tópicos a seguir mostram como configurar a DLAMI para atender aos seus objetivos de segurança e conformidade. Você também aprende a usar outros AWS serviços que ajudam a monitorar e proteger seus recursos de DLAMI.

Para obter mais informações, consulte Segurança no Amazon EC2.

Tópicos

Proteção de dados em AWS Deep Learning AMI

O modelo de responsabilidade AWS compartilhada se aplica à proteção de dados no AWS Deep Learning AMI. Conforme descrito neste modelo, AWS é responsável por proteger a infraestrutura global que executa todos os Nuvem AWS. Você é responsável por manter o controle sobre seu conteúdo hospedado nessa infraestrutura. Você também é responsável pelas tarefas de configuração e gerenciamento de segurança dos Serviços da AWS que usa. Para ter mais informações sobre a privacidade de dados, consulte as Perguntas frequentes sobre privacidade de dados. Para ter mais informações sobre a proteção de dados na Europa, consulte a postagem do blog AWS Shared Responsibility Model and GDPR no Blog de segurança da AWS .

Para fins de proteção de dados, recomendamos que você proteja Conta da AWS as credenciais e configure usuários individuais com AWS IAM Identity Center ou AWS Identity and Access Management (IAM). Dessa maneira, cada usuário receberá apenas as permissões necessárias para cumprir suas obrigações de trabalho. Recomendamos também que você proteja seus dados das seguintes formas:

- Use uma autenticação multifator (MFA) com cada conta.
- Use SSL/TLS para se comunicar com os recursos. AWS Exigimos TLS 1.2 e recomendamos TLS 1.3.
- Configure a API e o registro de atividades do usuário com AWS CloudTrail.
- Use soluções de AWS criptografia, juntamente com todos os controles de segurança padrão Serviços da AWS.
- Use serviços gerenciados de segurança avançada, como o Amazon Macie, que ajuda a localizar e proteger dados sigilosos armazenados no Amazon S3.
- Se você precisar de módulos criptográficos validados pelo FIPS 140-2 ao acessar AWS por meio de uma interface de linha de comando ou de uma API, use um endpoint FIPS. Para ter mais informações sobre endpoints do FIPS, consulte Federal Information Processing Standard (FIPS) 140-2.

É altamente recomendável que nunca sejam colocadas informações de identificação confidenciais, como endereços de e-mail dos seus clientes, em marcações ou campos de formato livre, como um campo Nome. Isso inclui quando você trabalha com o DLAMI ou Serviços da AWS outro usando o console, a API AWS CLI ou os SDKs. AWS Quaisquer dados inseridos em tags ou campos de texto de formato livre usados para nomes podem ser usados para logs de faturamento ou de diagnóstico.

Se você fornecer um URL para um servidor externo, recomendamos fortemente que não sejam incluídas informações de credenciais no URL para validar a solicitação a esse servidor.

Identity and Access Management em AWS Deep Learning AMI

AWS Identity and Access Management (IAM) é uma ferramenta AWS service (Serviço da AWS) que ajuda o administrador a controlar com segurança o acesso aos AWS recursos. Os administradores do IAM controlam quem pode ser autenticado (fazer login) e autorizado (ter permissões) para usar os recursos da DLAMI. O IAM é um AWS service (Serviço da AWS) que você pode usar sem custo adicional.

Para obter mais informações sobre Identity and Access Management, consulte Identity and Access Management para o Amazon EC2.

Tópicos

- Autenticação com identidades
- Gerenciamento do acesso usando políticas
- IAM com o Amazon EMR

Autenticação com identidades

A autenticação é a forma como você faz login AWS usando suas credenciais de identidade. Você deve estar autenticado (conectado AWS) como o Usuário raiz da conta da AWS, como usuário do IAM ou assumindo uma função do IAM.

Você pode entrar AWS como uma identidade federada usando credenciais fornecidas por meio de uma fonte de identidade. AWS IAM Identity Center Usuários (IAM Identity Center), a autenticação de login único da sua empresa e suas credenciais do Google ou do Facebook são exemplos de identidades federadas. Quando você faz login como uma identidade federada, o administrador já configurou anteriormente a federação de identidades usando perfis do IAM. Ao acessar AWS usando a federação, você está assumindo indiretamente uma função.

Dependendo do tipo de usuário que você é, você pode entrar no AWS Management Console ou no portal de AWS acesso. Para obter mais informações sobre como fazer login AWS, consulte Como fazer login Conta da AWS no Guia do Início de Sessão da AWS usuário.

Se você acessar AWS programaticamente, AWS fornece um kit de desenvolvimento de software (SDK) e uma interface de linha de comando (CLI) para assinar criptograficamente suas solicitações

usando suas credenciais. Se você não usa AWS ferramentas, você mesmo deve assinar as solicitações. Para obter mais informações sobre como usar o método recomendado para assinar solicitações por conta própria, consulte Assinatura de solicitações de AWS API no Guia do usuário do IAM.

Independentemente do método de autenticação usado, também pode ser exigido que você forneça informações adicionais de segurança. Por exemplo, AWS recomenda que você use a autenticação multifator (MFA) para aumentar a segurança da sua conta. Para saber mais, consulte Autenticação multifator no Guia AWS IAM Identity Center do usuário. Usar a autenticação multifator (MFA) na AWS no Guia do usuário do IAM.

Conta da AWS usuário root

Ao criar uma Conta da AWS, você começa com uma identidade de login que tem acesso completo a todos Serviços da AWS os recursos da conta. Essa identidade é chamada de usuário Conta da AWS raiz e é acessada fazendo login com o endereço de e-mail e a senha que você usou para criar a conta. É altamente recomendável não usar o usuário raiz para tarefas diárias. Proteja as credenciais do usuário raiz e use-as para executar as tarefas que somente ele pode executar. Para obter a lista completa das tarefas que exigem login como usuário raiz, consulte Tarefas que exigem credenciais de usuário raiz no Guia do usuário do IAM.

Grupos e usuários do IAM

Um usuário do IAM é uma identidade dentro da sua Conta da AWS que tem permissões específicas para uma única pessoa ou aplicativo. Sempre que possível, recomendamos depender de credenciais temporárias em vez de criar usuários do IAM com credenciais de longo prazo, como senhas e chaves de acesso. No entanto, se você tiver casos de uso específicos que exijam credenciais de longo prazo com usuários do IAM, recomendamos alternar as chaves de acesso. Para mais informações, consulte Alterne as chaves de acesso regularmente para casos de uso que exijam credenciais de longo prazo no Guia do usuário do IAM.

Um grupo do IAM é uma identidade que especifica uma coleção de usuários do IAM. Não é possível fazer login como um grupo. É possível usar grupos para especificar permissões para vários usuários de uma vez. Os grupos facilitam o gerenciamento de permissões para grandes conjuntos de usuários. Por exemplo, você pode ter um grupo chamado IAMAdmins e atribuir a esse grupo permissões para administrar recursos do IAM.

Usuários são diferentes de perfis.. Um usuário é exclusivamente associado a uma pessoa ou a uma aplicação, mas um perfil pode ser assumido por qualquer pessoa que precisar dele. Os usuários

têm credenciais permanentes de longo prazo, mas os perfis fornecem credenciais temporárias. Para saber mais, consulte Quando criar um usuário do IAM (em vez de um perfil) no Guia do usuário do IAM.

Perfis do IAM

Uma função do IAM é uma identidade dentro da sua Conta da AWS que tem permissões específicas. Ele é semelhante a um usuário do IAM, mas não está associado a uma pessoa específica. Você pode assumir temporariamente uma função do IAM no AWS Management Console trocando de funções. Você pode assumir uma função chamando uma operação de AWS API AWS CLI ou usando uma URL personalizada. Para mais informações sobre métodos para o uso de perfis, consulte Usar perfis do IAM no Guia do usuário do IAM.

Perfis do IAM com credenciais temporárias são úteis nas seguintes situações:

- Acesso de usuário federado: para atribuir permissões a identidades federadas, você pode criar um perfil e definir permissões para ele. Quando uma identidade federada é autenticada, essa identidade é associada ao perfil e recebe as permissões definidas pelo mesmo. Para mais informações sobre perfis para federação, consulte Criar um perfil para um provedor de identidades de terceiros no Guia do usuário do IAM. Se você usar o IAM Identity Center, configure um conjunto de permissões. Para controlar o que suas identidades podem acessar após a autenticação, o IAM Identity Center correlaciona o conjunto de permissões a um perfil no IAM. Para obter informações sobre conjuntos de permissões, consulte Conjuntos de permissões no Guia do Usuário do AWS IAM Identity Center .
- Permissões temporárias para usuários do IAM: um usuário ou um perfil do IAM pode assumir um perfil do IAM para obter temporariamente permissões diferentes para uma tarefa específica.
- Acesso entre contas – É possível usar um perfil do IAM para permitir que alguém (uma entidade principal confiável) em outra conta acesse recursos em sua conta. Os perfis são a principal forma de conceder acesso entre contas. No entanto, com alguns Serviços da AWS, você pode anexar uma política diretamente a um recurso (em vez de usar uma função como proxy). Para saber a diferença entre perfis e políticas baseadas em recurso para acesso entre contas, consulte Como os perfis do IAM diferem das políticas baseadas em recurso no Guia do usuário do IAM.
- Acesso entre serviços — Alguns Serviços da AWS usam recursos em outros Serviços da AWS. Por exemplo, quando você faz uma chamada em um serviço, é comum que esse serviço execute aplicações no Amazon EC2 ou armazene objetos no Amazon S3. Um serviço pode fazer isso usando as permissões do principal de chamada, usando um perfil de serviço ou um perfil vinculado ao serviço.

- Sessões de acesso direto (FAS) — Quando você usa um usuário ou uma função do IAM para realizar ações AWS, você é considerado principal. Ao usar alguns serviços, você pode executar uma ação que inicia outra ação em um serviço diferente. O FAS usa as permissões do diretor chamando um AWS service (Serviço da AWS), combinadas com a solicitação AWS service (Serviço da AWS) para fazer solicitações aos serviços posteriores. As solicitações do FAS são feitas somente quando um serviço recebe uma solicitação que requer interações com outros Serviços da AWS ou com recursos para ser concluída. Nesse caso, você precisa ter permissões para executar ambas as ações. Para obter detalhes da política ao fazer solicitações de FAS, consulte Encaminhar sessões de acesso.
- Perfil de serviço: um perfil de serviço é um perfil do IAM que um serviço assume para realizar ações em seu nome. Um administrador do IAM pode criar, modificar e excluir um perfil de serviço do IAM. Para mais informações, consulte Criar um perfil para delegar permissões a um AWS service (Serviço da AWS) no Guia do Usuário do IAM.
- Função vinculada ao serviço — Uma função vinculada ao serviço é um tipo de função de serviço vinculada a um. AWS service (Serviço da AWS) O serviço pode assumir o perfil para executar uma ação em seu nome. As funções vinculadas ao serviço aparecem em você Conta da AWS e são de propriedade do serviço. Um administrador do IAM pode visualizar, mas não pode editar as permissões para perfis vinculados ao serviço.

- Aplicativos em execução no Amazon EC2 — Você pode usar uma função do IAM para gerenciar credenciais temporárias para aplicativos que estão sendo executados em uma instância do EC2 e fazendo AWS CLI solicitações de API. AWS É preferível fazer isso a armazenar chaves de acesso na instância do EC2. Para atribuir uma AWS função a uma instância do EC2 e disponibilizá-la para todos os seus aplicativos, você cria um perfil de instância anexado à instância. Um perfil de instância contém o perfil e permite que os programas em execução na instância do EC2 obtenham credenciais temporárias. Para mais informações, consulte Usar um perfil do IAM para conceder permissões a aplicações em execução nas instâncias do Amazon EC2 no Guia do usuário do IAM.

Para saber se deseja usar os perfis do IAM, consulte Quando criar um perfil do IAM (em vez de um usuário) no Guia do usuário do IAM.

Gerenciamento do acesso usando políticas

Você controla o acesso AWS criando políticas e anexando-as a AWS identidades ou recursos. Uma política é um objeto AWS que, quando associada a uma identidade ou recurso, define suas permissões. AWS avalia essas políticas quando um principal (usuário, usuário raiz ou sessão de função) faz uma solicitação. As permissões nas políticas determinam se a solicitação será permitida

ou negada. A maioria das políticas é armazenada AWS como documentos JSON. Para mais informações sobre a estrutura e o conteúdo de documentos de políticas JSON, consulte Visão geral das políticas JSON no Guia do usuário do IAM.

Os administradores podem usar políticas AWS JSON para especificar quem tem acesso ao quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

Por padrão, usuários e funções não têm permissões. Para conceder aos usuários permissão para executar ações nos recursos de que eles precisam, um administrador do IAM pode criar políticas do IAM. O administrador pode então adicionar as políticas do IAM a perfis, e os usuários podem assumir os perfis.

As políticas do IAM definem permissões para uma ação, independentemente do método usado para executar a operação. Por exemplo, suponha que você tenha uma política que permite a ação `iam:GetRole`. Um usuário com essa política pode obter informações de função da AWS Management Console AWS CLI, da ou da AWS API.

Políticas baseadas em identidade

As políticas baseadas em identidade são documentos de políticas de permissões JSON que você pode anexar a uma identidade, como usuário, grupo de usuários ou perfil do IAM. Essas políticas controlam quais ações os usuários e funções podem realizar, em quais recursos e em que condições. Para saber como criar uma política baseada em identidade, consulte Criação de política do IAM no Guia do usuário do IAM.

As políticas baseadas em identidade podem ser categorizadas ainda mais como políticas em linha ou políticas gerenciadas. As políticas em linha são anexadas diretamente a um único usuário, grupo ou perfil. As políticas gerenciadas são políticas autônomas que você pode associar a vários usuários, grupos e funções em seu Conta da AWS. As políticas AWS gerenciadas incluem políticas gerenciadas e políticas gerenciadas pelo cliente. Para saber como escolher entre uma política gerenciada ou uma política em linha, consulte Escolher entre políticas gerenciadas e políticas em linha no Guia do usuário do IAM.

Políticas baseadas em recurso

Políticas baseadas em recurso são documentos de políticas JSON que você anexa a um recurso. São exemplos de políticas baseadas em recursos as políticas de confiança de função do IAM e as políticas de bucket do Amazon S3. Em serviços compatíveis com políticas baseadas em recursos, os administradores de serviço podem usá-las para controlar o acesso a um recurso específico. Para o recurso ao qual a política está anexada, a política define quais ações uma entidade principal

especificada pode executar nesse recurso e em que condições. Você deve especificar uma entidade principal em uma política baseada em recursos. Os diretores podem incluir contas, usuários, funções, usuários federados ou. Serviços da AWS

Políticas baseadas em recursos são políticas em linha que estão localizadas nesse serviço. Você não pode usar políticas AWS gerenciadas do IAM em uma política baseada em recursos.

Listas de controle de acesso (ACLs)

As listas de controle de acesso (ACLs) controlam quais entidades principais (membros, usuários ou funções da conta) têm permissões para acessar um recurso. As ACLs são semelhantes às políticas baseadas em recursos, embora não usem o formato de documento de política JSON.

O Amazon S3 e o Amazon VPC são exemplos de serviços que oferecem suporte a ACLs. AWS WAF Para saber mais sobre ACLs, consulte Visão geral da lista de controle de acesso (ACL) no Guia do desenvolvedor do Amazon Simple Storage Service.

Outros tipos de política

AWS oferece suporte a tipos de políticas adicionais menos comuns. Esses tipos de política podem definir o máximo de permissões concedidas a você pelos tipos de política mais comuns.

- Limites de permissões: um limite de permissões é um atributo avançado no qual você define o máximo de permissões que uma política baseada em identidade pode conceder a uma entidade do IAM (usuário ou perfil do IAM). É possível definir um limite de permissões para uma entidade. As permissões resultantes são a interseção das políticas baseadas em identidade de uma entidade e dos seus limites de permissões. As políticas baseadas em recurso que especificam o usuário ou a função no campo `Principal` não são limitadas pelo limite de permissões. Uma negação explícita em qualquer uma dessas políticas substitui a permissão. Para mais informações sobre limites de permissões, consulte Limites de permissões para identidades do IAM no Guia do usuário do IAM.
- Políticas de controle de serviço (SCPs) — SCPs são políticas JSON que especificam as permissões máximas para uma organização ou unidade organizacional (OU) em. AWS Organizations AWS Organizations é um serviço para agrupar e gerenciar centralmente vários Contas da AWS que sua empresa possui. Se você habilitar todos os recursos em uma organização, poderá aplicar políticas de controle de serviço (SCPs) a qualquer uma ou a todas as contas. O SCP limita as permissões para entidades nas contas dos membros, incluindo cada uma Usuário raiz da conta da AWS. Para mais informações sobre Organizações e SCPs, consulte Como os SCPs funcionam no AWS Organizations Guia do Usuário.

- Políticas de sessão: são políticas avançadas que você transmite como um parâmetro quando cria de forma programática uma sessão temporária para um perfil ou um usuário federado. As permissões da sessão resultante são a interseção das políticas baseadas em identidade do usuário ou do perfil e das políticas de sessão. As permissões também podem ser provenientes de uma política baseada em recurso. Uma negação explícita em qualquer uma dessas políticas substitui a permissão. Para mais informações, consulte Políticas de sessão no Guia do usuário do IAM.

Vários tipos de política

Quando vários tipos de política são aplicáveis a uma solicitação, é mais complicado compreender as permissões resultantes. Para saber como AWS determinar se uma solicitação deve ser permitida quando vários tipos de políticas estão envolvidos, consulte Lógica de avaliação de políticas no Guia do usuário do IAM.

IAM com o Amazon EMR

Você pode usar AWS Identity and Access Management com o Amazon EMR para definir usuários, AWS recursos, grupos, funções e políticas. Você também pode controlar quais AWS serviços esses usuários e funções podem acessar.

Para obter mais informações sobre como usar o IAM com o Amazon EMR, consulte AWS Identity and Access Management para Amazon EMR.

Registro e monitoramento em AWS Deep Learning AMI

Sua AWS Deep Learning AMI instância vem com várias ferramentas de monitoramento de GPU, incluindo um utilitário que reporta estatísticas de uso da GPU para a Amazon. CloudWatch Para obter mais informações, consulte Monitoramento e otimização de GPU e Monitoramento do Amazon EC2.

Monitoramento de uso

As distribuições de sistema AWS Deep Learning AMI operacional a seguir incluem código que permite AWS coletar o tipo de instância, ID da instância, tipo de DLAMI e informações do sistema operacional. Nenhuma informação sobre os comandos usados na DLAMI é coletada ou retida. Nenhuma outra informação sobre a DLAMI é coletada ou retida.

- Ubuntu 16.04
- Ubuntu 18.04
- Ubuntu 20.04
- Amazon Linux 2

Para desativar o monitoramento de uso da DLAMI, adicione uma tag à instância do Amazon EC2 durante a execução. A tag deve usar a chave `OPT_OUT_TRACKING` com o valor associado definido como `true`. Para obter mais informações, consulte Marcar recursos do Amazon EC2.

Validação de conformidade para AWS Deep Learning AMI

Auditores terceirizados avaliam a segurança e a conformidade AWS Deep Learning AMI como parte de vários programas de AWS conformidade. Para obter informações sobre os programas de conformidade compatíveis, consulte Validação de conformidade para o Amazon EC2.

Para obter uma lista de AWS serviços no escopo de programas de conformidade específicos, consulte AWS Serviços no escopo do programa de conformidade AWS . Para obter informações gerais, consulte Programas de AWS conformidade Programas AWS de .

Você pode baixar relatórios de auditoria de terceiros usando AWS Artifact. Para obter mais informações, consulte Baixando relatórios no AWS Artifact .

Sua responsabilidade de conformidade ao usar o DLAMI é determinada pela sensibilidade de seus dados, pelos objetivos de conformidade de sua empresa e pelas leis e regulamentos aplicáveis. AWS fornece os seguintes recursos para ajudar na conformidade:

- Guias de início rápido de segurança e compatibilidade: estes guias de implantação abordam as considerações de arquitetura e fornecem etapas para implantação de ambientes de linha de base focados em compatibilidade e segurança na AWS.
- AWS Recursos de https://aws.amazon.com/compliance/resources/ de conformidade — Essa coleção de pastas de trabalho e guias pode ser aplicada ao seu setor e local.
- Avaliação de recursos com regras no Guia do AWS Config desenvolvedor — O AWS Config serviço avalia o quão bem suas configurações de recursos estão em conformidade com as práticas internas, as diretrizes e os regulamentos do setor.
- AWS Security Hub— Esse AWS serviço fornece uma visão abrangente do seu estado de segurança interno, AWS que ajuda você a verificar sua conformidade com os padrões e as melhores práticas do setor de segurança.

Resiliência em AWS Deep Learning AMI

A infraestrutura AWS global é construída em torno de AWS regiões e zonas de disponibilidade. AWS As regiões fornecem várias zonas de disponibilidade fisicamente separadas e isoladas, conectadas a redes de baixa latência, alta taxa de transferência e alta redundância. Com as zonas de disponibilidade, é possível projetar e operar aplicações e bancos de dados que automaticamente executam o failover entre as zonas sem interrupção. As zonas de disponibilidade são mais altamente disponíveis, tolerantes a falhas e escaláveis que uma ou várias infraestruturas de datacenter tradicionais.

Para obter mais informações sobre AWS regiões e zonas de disponibilidade, consulte Infraestrutura AWS global.

Para obter informações sobre recursos para ajudar a satisfazer suas necessidades de resiliência e backup de dados, consulte Resiliência no Amazon EC2.

Segurança de infraestrutura em AWS Deep Learning AMI

A segurança da infraestrutura do AWS Deep Learning AMI é apoiada pelo Amazon EC2. Para obter mais informações, consulte Segurança da infraestrutura no Amazon EC2.

Mudanças importantes no DLAMI

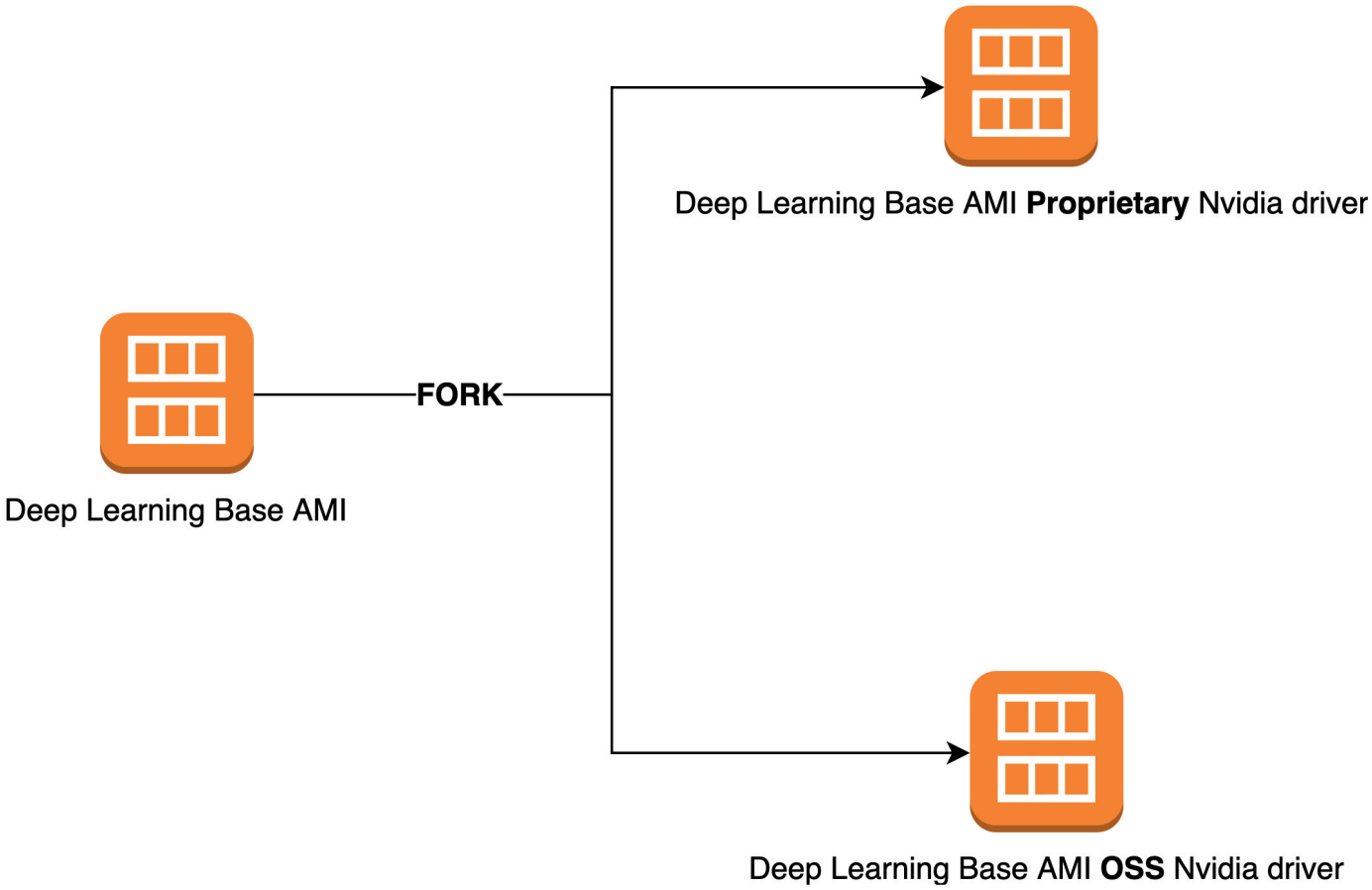
Perguntas frequentes

O que está mudando?

Em 15/11/2023 AWS Deep Learning AMI (DLamis) será dividido em dois grupos separados:

- DLAMIs que usam o driver proprietário da Nvidia (para suportar P3, P3dn, G3).
- DLAMIs que usam o driver Nvidia OSS (para suportar G4dn, G5, P4, P5).

Como resultado, novas DLAMIs serão criadas para cada uma das duas categorias com novos nomes e novas IDs de AMI. Esses DLAMIs não serão intercambiáveis - ou seja, os DLAMIs de um grupo não suportarão instâncias suportadas pelo outro grupo, por exemplo, o DLAMI que suporta p5 não suportará g3 e vice-versa.



Por que essa mudança é necessária?

Atualmente, os DLAMiS para GPUs NVIDIA incluem um driver de kernel proprietário da NVIDIA. No entanto, recentemente, a comunidade upstream do kernel Linux aceitou uma mudança que isola os drivers proprietários do kernel, como o driver da GPU NVIDIA, da comunicação com outros drivers do kernel. Essa alteração desativa o GPUDirect RDMA nas instâncias da série P4/P5, que é o mecanismo que permite que as GPUs usem o EFA de forma eficiente para treinamento distribuído. Como resultado, o DLamis usará o driver OpenRM (driver de código aberto NVIDIA), vinculado aos drivers EFA de código aberto para suportar G4dn, G5, P4 e P5. No entanto, esse driver OpenRM não suportará instâncias mais antigas (P3, G3 etc.) Portanto, para garantir que continuemos a fornecer DLAMis atuais, de alto desempenho e seguros que suportem os dois tipos de instâncias, dividiremos os DLAMis em dois grupos: um com o driver OpenRM (compatível com G4dn, G5, P4 e P5) e outro com o driver proprietário mais antigo (compatível com instâncias mais antigas P3, P3dn, G3).

Quais DLAMIs são afetadas por essa mudança?

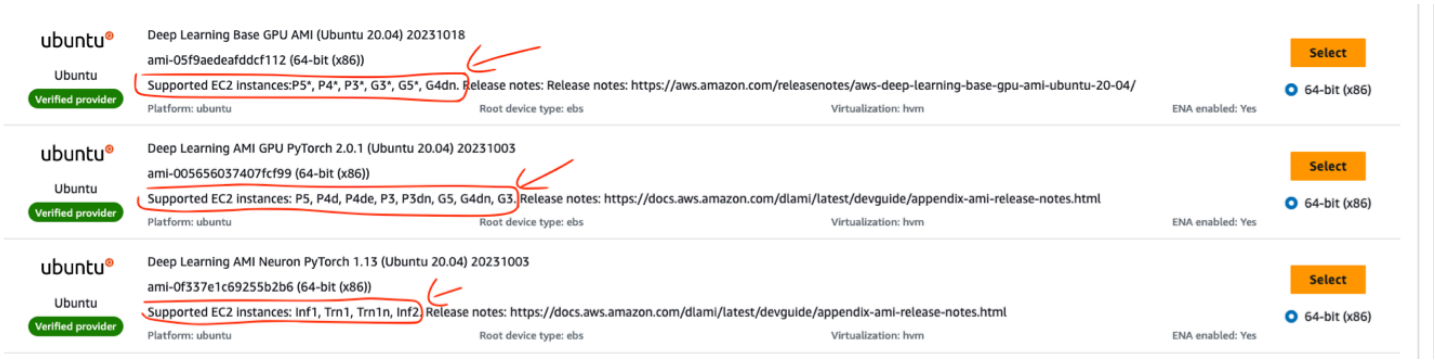
Todas as DLAMIs são afetadas por essa alteração.

O que isso significa para você?

Os novos DLAMIs continuarão a fornecer funcionalidade, desempenho e segurança dos DLAMIs atuais, desde que sejam executados em um tipo de instância compatível. Se você estiver usando o DLAMI, precisará garantir que um DLAMI seja lançado em uma das instâncias compatíveis mencionadas nas notas de lançamento de cada DLAMI (veja aqui). Por exemplo: você precisará acomodar essa alteração para:

- Invoque DLamis com as consultas CLI corretas (veja abaixo)
- Inicie o DLamis a partir do console e da CLI em um tipo de instância compatível

Se você estiver iniciando o DLAMI a partir do console EC2 Quickstart: cada descrição do DLAMI lista os tipos de instâncias compatíveis com o console EC2. Você deve iniciar o DLamis em instâncias compatíveis.



Se você estiver lançando o DLamis usando CLI, precisará modificar suas consultas. Por exemplo: .

Atualmente, a seguinte consulta CLI é usada para DLAMIs básicas que oferecem suporte a todas as instâncias [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
```

As novas consultas da CLI serão:

Para DLAMI básico compatível com P3, P3dn e G3:

```
aws ec2 describe-images --region us-east-1 --owners amazon \
--filters 'Name=name,Values=Deep Learning Base Proprietary Nvidia Driver AMI (Amazon
 Linux 2) Version ??.?' 'Name=state,Values=available' \
--query 'reverse(sort_by(Images, &CreationDate))[:1].ImageId' --output text
```

Para DLAMI básico compatível com G4dn, G5, P4 e P5:

```
aws ec2 describe-images --region us-east-1 --owners amazon \
--filters 'Name=name,Values=Deep Learning Base OSS Nvidia Driver AMI (Amazon Linux 2)
 Version ??.?' 'Name=state,Values=available' \
--query 'reverse(sort_by(Images, &CreationDate))[:1].ImageId' --output text
```

Consulte as notas de versão atualizadas para ver as novas AMIs aqui. Para saber como iniciar AMIs em instâncias do EC2, consulte as instruções aqui.

Quando você deve começar a usar o novo DLamis?

Você deve começar a usar o novo DLAMis o mais rápido possível para obter as estruturas, dependências, patches e funcionalidades mais recentes. Opcionalmente, se você estiver usando Amazon Linux 2 DLAMis lançados antes de 11/8/2023, então você pode optar por continuar aplicando patches ativos para seus DLAMis (veja as instruções aqui) até 30/11/2023.

Haverá alguma perda de funcionalidade com o novo DLamis?

Não, não há perda de funcionalidade com o novo DLamis. Os novos DLAMis após a divisão continuarão a fornecer todas as funcionalidades, desempenho e segurança dos antigos DLAMis antes da divisão, desde que sejam executados em uma instância compatível. Estamos dividindo os DLAMis em dois grupos para continuarmos a oferecer DLAMis atuais, eficientes e seguros para seu uso em uma ampla variedade de instâncias.

E quanto aos DLCs?

Os DLCs não incluem o driver NVIDIA, portanto, não são afetados por essa alteração. Mas você deve garantir que os DLCs sejam executados em AMIs compatíveis com as instâncias subjacentes.

Informações relacionadas

Tópicos

- Fóruns
- Publicações em blogs da relacionadas
- Perguntas frequentes

Fóruns

- Fórum: AMIs de deep learning da AWS

Publicações em blogs da relacionadas

- Lista atualizada de artigos relacionados às AMIs de deep learning
- Executar uma AWS Deep Learning AMI (em 10 minutos)
- Treinamento mais rápido com o TensorFlow 1.6 otimizado nas instâncias C5 e P3 do Amazon EC2
- Novas AMIs de deep learning da AWS para profissionais de machine learning
- Novos cursos de treinamento disponíveis: Introdução ao Machine Learning e ao aprendizado profundo na AWS
- Jornada de aprendizado profundo com a AWS

Perguntas frequentes

- P: Como faço para acompanhar os anúncios de produtos relacionados à DLAMI?

  Confira duas sugestões para isso:

  - Adicione esta categoria de blog aos favoritos, “AMIs de deep learning da AWS”, encontrada aqui: Lista atualizada de artigos relacionados a AMIs de deep learning.
  - “Assista” ao fórum: AMIs de deep learning da AWS
- P: Os drivers NVIDIA e o CUDA estão instalados?

  Sim. Algumas DLAMIs têm versões diferentes. O AMI de deep learning com Conda tem as versões mais recentes de qualquer DLAMI. Isso é abordado com mais detalhes em Instalações do CUDA

e associações da estrutura. Você também pode consultar as notas de versão específicas da AMI para confirmar o que está instalado.

- P: O cuDNN está instalado?

  Sim.

- P: Como faço para ver se as GPUs foram detectadas e seu status atual?

  Execute `nvidia-smi`. Isso mostrará uma ou mais GPUs, dependendo do tipo de instância, juntamente com o seu consumo de memória atual.

- P. Há ambientes virtuais configurados para mim?

  Sim, mas apenas o AMI de deep learning com Conda.

- P. Qual versão do Python está instalada?

  Cada DLAMI tem Python 2 e 3. A AMI de deep learning com Conda tem ambientes para as duas versões de cada estrutura.

- P: O Keras está instalado?

  Isso depende da AMI. A AMI de deep learning com Conda tem o Keras disponível como um front-end para cada estrutura. A versão do Keras depende do suporte que a estrutura oferece para ele.

- P: É gratuito?

  Todas as DLAMIs são gratuitas. No entanto, dependendo do tipo de instância que você escolher, a instância poderá não ser gratuita. Consulte Definição de preço para a DLAMI para obter mais informações.

- P: Estou recebendo erros do CUDA ou mensagens relacionadas à GPU em minha estrutura. O que está errado?

  Verifique o tipo de instância que você usou. Para funcionar, ela precisa ter uma GPU para muitos exemplos e tutoriais. Se a execução do `nvidia-smi` não mostrar nenhuma GPU, você precisará criar outra DLAMI usando uma instância com uma ou mais GPUs. Consulte Seleção do tipo de instância para DLAMI para obter mais informações.

- P: Posso usar o Docker?

  O Docker foi pré-instalado desde a versão 14 da AMI de deep learning com o Conda. Observe que você desejará usar o nvidia-docker em instâncias para fazer uso da GPU.

- P: Em quais regiões as DLAMIs do Linux estão disponíveis?

região	Código
Leste dos EUA (Ohio)	us-east-2
Leste dos EUA (N. da Virgínia)	us-east-1
GovCloud	us-gov-west-1
Oeste dos EUA (N. da Califórnia)	us-west-1
Oeste dos EUA (Oregon)	us-west-2
Pequim (China)	cn-north-1
Ningxia (China)	cn-northwest-1
Ásia-Pacífico (Mumbai)	ap-south-1
Ásia-Pacífico (Seul)	ap-northeast-2
Ásia-Pacífico (Singapura)	ap-southeast-1
Ásia-Pacífico (Sydney)	ap-southeast-2
Ásia-Pacífico (Tóquio)	ap-northeast-1
Canadá (Central)	ca-central-1
UE (Frankfurt)	eu-central-1
UE (Irlanda)	eu-west-1
UE (Londres)	eu-west-2
UE (Paris)	eu-west-3
SA (São Paulo)	sa-east-1

- P: Em quais regiões as DLAMIs do Windows estão disponíveis?

região	Código
Leste dos EUA (Ohio)	us-east-2
Leste dos EUA (N. da Virgínia)	us-east-1
GovCloud	us-gov-west-1
Oeste dos EUA (N. da Califórnia)	us-west-1
Oeste dos EUA (Oregon)	us-west-2
Pequim (China)	cn-north-1
Ásia-Pacífico (Mumbai)	ap-south-1
Ásia-Pacífico (Seul)	ap-northeast-2
Ásia-Pacífico (Singapura)	ap-southeast-1
Ásia-Pacífico (Sydney)	ap-southeast-2
Ásia-Pacífico (Tóquio)	ap-northeast-1
Canadá (Central)	ca-central-1
UE (Frankfurt)	eu-central-1
UE (Irlanda)	eu-west-1
UE (Londres)	eu-west-2
UE (Paris)	eu-west-3
SA (São Paulo)	sa-east-1

Notas de versão da DLAMI

> **Note**
>
> As AWS Deep Learning AMI s têm uma cadência de lançamentos noturnos de patches de segurança. Esses patches de segurança incrementais não estão incluídos nas notas de versão oficiais.

Consulte a página da Política de Suporte do DLAMI para ver as notas de lançamento de qualquer estrutura não suportada.

DLAMI base

GPU

- AWS AMI de base de aprendizado profundo (Amazon Linux 2)
- AWS AMI de base de aprendizado profundo (Ubuntu 20.04)

AWS Neuron

- AWS Base de aprendizado profundo AMI Neuron (Amazon Linux 2)
- AWS Base de aprendizado profundo AMI Neuron (Ubuntu 20.04)

Qualcomm

- AWS Base de aprendizado profundo Qualcomm AMI (Amazon Linux 2)

DLAMI de estrutura única

PyTorch-AMI específica

- GPU
  - AWS AMI GPU PyTorch 2.1 de aprendizado profundo (Ubuntu 20.04)
  - AWS GPU AMI PyTorch 1.13 de aprendizado profundo (Amazon Linux 2)

  - AWS GPU AMI de aprendizado profundo PyTorch 1.13 (Ubuntu 20.04)
- AWS Neuron
  - AWS AMI de aprendizado profundo Neuron PyTorch 1.13 (Amazon Linux 2)
  - AWS AMI de aprendizado profundo Neuron PyTorch 1.13 (Ubuntu 20.04)

TensorFlow-AMI específica

- GPU
  - AWS GPU AMI de aprendizado profundo TensorFlow 2.15 (Amazon Linux 2)
  - AWS GPU AMI de aprendizado profundo TensorFlow 2.15 (Ubuntu 20.04)
  - AWS GPU AMI de aprendizado profundo TensorFlow 2.13 (Amazon Linux 2)
  - AWS GPU AMI de aprendizado profundo TensorFlow 2.13 (Ubuntu 20.04)
- AWS Neuron
  - AWS AMI de aprendizado profundo Neuron TensorFlow 2.10 (Amazon Linux 2)
  - AWS AMI Neuron TensorFlow 2.10 de aprendizado profundo (Ubuntu 20.04)

DLAMI de várias estruturas

GPU

> **Note**
>
> Se você usa apenas uma estrutura de aprendizado de máquina, recomendamos uma DLAMI de estrutura única

- AWS AMI de aprendizado profundo (Amazon Linux 2)

AWS Neuron

- AWS AMI Neuron de aprendizado profundo (Ubuntu 22.04)

Avisos de descontinuação da DLAMI

A tabela a seguir lista informações sobre recursos defasados na AWS Deep Learning AMI.

Recurso defasado	Data da defasagem	Aviso de defasagem
Ubuntu 16.04	07/10/2021	O Ubuntu Linux 16.04 LTS chegou ao fim da janela de LTS de cinco anos em 30 de abril de 2021 e não o fornecedor não fornece mais suporte. Não há mais atualizações na AMI de deep learning base (Ubuntu 16.04) em novas versões a partir de outubro de 2021. As versões anteriores continuarão disponíveis.
Amazon Linux	07/10/2021	O Amazon Linux estará no fim de vida útil a partir de dezembro de 2020. Não há mais atualizações para novas versões da AMI de deep learning (Amazon Linux) a partir de outubro de 2021. As versões anteriores da AMI de deep learning (Amazon Linux) continuarão disponíveis.
Chainer	01/07/2020	A Chainer anunciou o fim dos grandes lançamentos a partir de dezembro de 2019. Consequentemente, deixaremos de incluir ambientes

Recurso defasado	Data da defasagem	Aviso de defasagem
		Chainer Conda na DLAMI a partir de julho de 2020. As versões anteriores da DLAMI que contêm esses ambientes continuar ão disponíveis. Nós só forneceremos atualizaç ões para esses ambientes se houver correções de segurança publicadas pela comunidade de código aberto para essas estrutura s de trabalho.
Python 3.6	15/06/2020	Devido a solicitações dos clientes, estamos mudando para o Python 3.7 para as novas versões TF/MX/PT.
Python 2	01/01/2020	A comunidade de código aberto Python encerrou oficialmente o suporte a Python 2. As comunidades de TensorFlow, PyTorch e MXNet também anunciaram que as versões TensorFlo w 1.15, TensorFlow 2.1, PyTorch 1.4 e MXNet 1.6.0 serão as últimas com suporte ao Python 2.

Histórico do documento do Guia do desenvolvedor do AWS Deep Learning AMI

Alteração	Descrição	Data
DLAMI do Graviton	Agora, a AWS Deep Learning AMI oferece suporte para imagens em GPUs Graviton que usam como base o processador Arm.	29 de novembro de 2021
DLAMI Habana	Agora a AWS Deep Learning AMI é compatível com o hardware Habana Gaudi e o SDK do Habana SynapseAI.	25 de outubro de 2021
TensorFlow 2	A Deep Learning AMI com Conda agora vem com o TensorFlow 2 com CUDA 10.	3 de dezembro de 2019
AWS Inferentia	A AMI de deep learning agora oferece suporte ao hardware do AWS Inferentia e ao SDK do AWS Neuron.	3 de dezembro de 2019
Como usar o TensorFlow Serving com um modelo de início	Um exemplo para usar a inferência com um modelo de início foi adicionado ao TensorFlow Serving, para com ou sem inferência elástica.	28 de novembro de 2018
Treinamento com 256 GPUs com TensorFlow e Horovod	O tutorial TensorFlow com Horovod foi atualizado para adicionar um exemplo de treinamento de vários nós.	28 de novembro de 2018

Elastic Inference	Pré-requisitos de inferênci a elástica e informações relacionadas à inferência foram adicionadas ao guia de configuração.	28 de novembro de 2018
MMS v1.0 lançado na DLAMI.	O tutorial do MMS foi atualizad o para usar o novo formato de arquivo de modelo (.mar) e demonstra os novos recursos de iniciar e interromper.	15 de novembro de 2018
Como instalar o TensorFlow a partir de uma compilação noturna	Um tutorial que abrange como você pode desinstalar o TensorFlow e, em seguida, instalar uma compilação noturna do TensorFlow na sua AMI de deep learning com o Conda.	16 de outubro de 2018
Como instalar o CTNK a partir de uma compilação noturna	Um tutorial que abrange como você pode desinstal ar o CNTK foi adicionado e, em seguida, instale uma compilação noturna do CNTK na AMI de deep learning com Conda.	16 de outubro de 2018
Como instalar o Apache MXNet (em incubação) a partir de uma compilação noturna	Um tutorial que abrange como você pode desinstal ar o MXNet foi adicionado e, em seguida, instalar uma compilação noturna do MXNet na AMI de deep learning com Conda.	16 de outubro de 2018

Como instalar o PyTorch a partir de uma compilação noturna	Um tutorial que abrange como você pode desinstalar o PyTorch foi adicionado e, em seguida, instale uma compilação noturna do PyTorch na AMI de deep learning com o Conda.	25 de setembro de 2018
Docker pré-instalado na DLAMI	Desde a v14 da AMI de deep learning com o Conda, o Docker e a versão do Docker para GPUs da NVIDIA foram pré-instaladas.	25 de setembro de 2018
Tutorial do TensorBoard	Exemplo foi movido para ~ /exemplos/tensorboard. Caminhos de tutorial atualizados.	23 de julho de 2018
Tutorial do MXBoard	Foi adicionado um tutorial sobre como usar o MXBoard para ver os modelos MXNet.	23 de julho de 2018
Tutoriais de treinamento distribuído	Um tutorial sobre como usar Keras-MXNet para treinamento de várias GPUs foi adicionado. O tutorial de Chainer foi atualizado para v4.2.0.	23 de julho de 2018
Tutorial do Conda	O exemplo MOTD foi atualizado para refletir uma versão mais recente.	23 de julho de 2018
Tutorial do Chainer	O tutorial foi atualizado para usar os exemplos mais recentes da fonte Chainer.	23 de julho de 2018

Atualizações anteriores:

A tabela a seguir descreve as alterações importantes em cada versão do AWS Deep Learning AMI antes de julho de 2018.

Alteração	Descrição	Data
TensorFlow com Horovod	Adicionado um tutorial para treinamento do ImageNet com TensorFlow e Horovod.	6 de junho de 2018
Guia de atualização	Adicionado o guia de atualizaç ão.	15 de maio de 2018
Novas regiões e novo tutorial de 10 minutos	Novas regiões adicionad as: Oeste dos EUA (Norte da Califórnia), América do Sul, Canadá (Central), UE (Londres) e UE (Paris). Além disso, o primeiro lançamento de um tutorial de 10 minutos chamado: "Conceitos básicos da Deep Learning AMI".	26 de abril de 2018
Tutorial do Chainer	Um tutorial para usar o Chainer nos modos várias GPUs, uma única GPU e CPU foi adicionado. A integraçã o do CUDA foi atualizada do CUDA 8 para o CUDA 9 para várias estruturas.	28 de fevereiro de 2018
AMIs do Linux v3.0, além da apresentação do MXNet Model Server, TensorFlow Serving e TensorBoard	Tutoriais para AMIs do Conda com novos recursos de fornecimento e visualiza ção de modelos usando MXNet Model Server v0.1.5, TensorFlow Serving v1.4.0 e	25 de janeiro de 2018

Alteração	Descrição	Data
	TensorBoard v0.4.0. Funcionalidades de AMI e CUDA de estrutura descritas nas visões gerais de Conda e CUDA. Notas de versão mais recentes movidas para https://aws.amazon.com/releasenotes/	
AMIs do Linux v2.0	Base, Fonte e AMIs de Conda atualizados com NCCL 2.1. Fonte e AMIs de Conda atualizados com MXNet v1.0, PyTorch 0.3.0 e Keras 2.0.9.	11 de dezembro de 2017
Duas opções de AMI do Windows adicionadas	AMIs do Windows 2012 R2 e 2016 lançadas: adicionadas ao guia de seleção de AMI e adicionadas às notas de release.	30 de novembro de 2017
Versão da documentação inicial	Descrição detalhada da alteração com o link para o tópico ou a seção que sofreu a alteração.	15 de novembro de 2017

AWS Glossário

Para obter a terminologia mais recente da AWS, consulte o glossário da AWS na Referência do Glossário da AWS.

As traduções são geradas por tradução automática. Em caso de conflito entre o conteúdo da tradução e da versão original em inglês, a versão em inglês prevalecerá.