Amazon Elastic Inference

Developer Guide

Amazon Elastic Inference: Developer Guide

Copyright © 2019 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What Is Amazon Elastic Inference?

Amazon Elastic Inference (EI) is a resource you can attach to your Amazon EC2 CPU instances to accelerate your deep learning (DL) inference workloads. Elastic Inference accelerators come in multiple sizes and are a cost-effective method to build intelligent capabilities into applications running on Amazon EC2 instances.

Elastic Inference distributes model operations defined by TensorFlow, Apache MXNet, and the Open Neural Network Exchange (ONNX) format through MXNet between low-cost, DL inference accelerators and the CPU of the instance.

Prerequisites

You will need an Amazon Web Services account and should be familiar with launching an EC2 instance to successfully run Amazon Elastic Inference. To launch an EC2 instance, complete the steps in Setting up with Amazon EC2. Amazon S3 resources are required for installing packages via pip. For more information about setting up Amazon S3 resources, see the Amazon Simple Storage Service Getting Started Guide.

Pricing for Amazon Elastic Inference

You are charged for each second that an Elastic Inference accelerator is attached to an instance in the `running` state. You are not charged for an accelerator attached to an instance that is in the `pending`, `stopping`, `stopped`, `shutting-down`, or `terminated` state. You are also not charged when an Elastic Inference accelerator is in the `unknown` or `impaired` state.

You do not incur AWS PrivateLink charges for VPC endpoints to the Elastic Inference service when you have accelerators provisioned in the subnet.

For more information about pricing by region for Elastic Inference, see Elastic Inference Pricing.

Next Up

Amazon Elastic Inference Basics (p. 1)

Amazon Elastic Inference Basics

When you configure an Amazon EC2 instance to launch with an Elastic Inference accelerator, AWS finds available accelerator capacity and establishes a network connection between your instance and the accelerator.

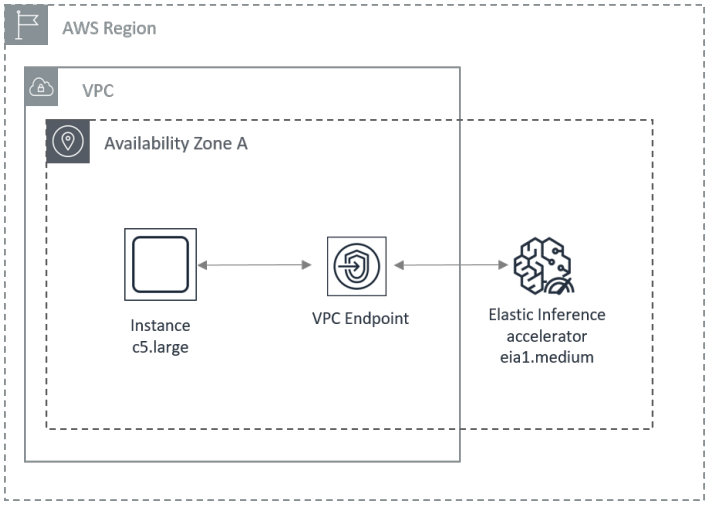
The following Elastic Inference accelerator types are available. You can attach any Elastic Inference accelerator type to any Amazon EC2 instance type.

Accelerator Type	FP32 Throughput (TFLOPS)	FP16 Throughput (TFLOPS)	Memory (GB)
eia1.medium	1	8	1

Accelerator Type	FP32 Throughput (TFLOPS)	FP16 Throughput (TFLOPS)	Memory (GB)
eia1.large	2	16	2
eia1.xlarge	4	32	4

An Elastic Inference accelerator is not part of the hardware that makes up your instance. Instead, the accelerator is attached through the network using an AWS PrivateLink endpoint service. The endpoint service routes traffic from your instance to the Elastic Inference accelerator configured with your instance.

Before you launch an instance with an Elastic Inference accelerator, you must create an AWS PrivateLink endpoint service. Only a single endpoint service is needed in every Availability Zone to connect instances with Elastic Inference accelerators. For more information, see VPC Endpoint Services (AWS PrivateLink).



You can use Amazon Elastic Inference enabled TensorFlow, TensorFlow Serving, or Apache MXNet libraries to load models and make inference calls. The modified versions of these frameworks automatically detect the presence of Elastic Inference accelerators and optimally distribute the model operations between the Elastic Inference accelerator and the CPU of the instance. The AWS Deep Learning AMIs include the latest releases of Amazon Elastic Inference enabled TensorFlow Serving and MXNet. If you are using custom AMIs or container images, you can download and install the required Amazon Elastic Inference TensorFlow and Amazon Elastic Inference Apache MXNet libraries from Amazon S3.

> **Note**
> An Elastic Inference accelerator is not visible or accessible through the management console of your instance.

Before you get started with Amazon Elastic Inference

Amazon EI Service Limits

Before you start using Elastic Inference accelerators, be aware of the following limitations:

Limit	Description		
Elastic Inference accelerator instance limit	You can only attach one Elastic Inference accelerator to each instance at a time, and only during instance launch.		
Elastic Inference Sharing	You cannot share an Elastic Inference accelerator between instances.		
Elastic Inference Transfer	You cannot detach an Elastic Inference accelerator from an instance or transfer it to another instance. If you no longer require an Elastic Inference accelerator you must terminate your instance. You cannot change the Elastic Inference accelerator type; you must terminate the instance and launch a new instance with a different Elastic Inference accelerator specification.		
Supported Libraries	Only the Amazon Elastic Inference enhanced MXNet and Amazon Elastic Inference enhanced TensorFlow libraries can make inference calls to Elastic Inference accelerators.		
Elastic Inference Attachment	Elastic Inference accelerators can only be attached to instances in a VPC.		
Reserving accelerator capacity	Pricing for Elastic Inference accelerators is available at on-demand rates only. You can attach an accelerator to a Reserved Instance, Scheduled Reserved Instance, or Spot Instance. However, the		

Limit	Description		
	on-demand price for the Elastic Inference accelerator applies. You cannot reserve or schedule Elastic Inference accelerator capacity.		

Choosing an Instance and Accelerator Type for Your Model

Demands on CPU compute resources, CPU memory, GPU-based acceleration, and GPU memory vary significantly between different types of deep learning models. The latency and throughput requirements of the application also determine the amount of instance compute and Elastic Inference acceleration you need. Consider the following when you choose an instance and accelerator type combination for your model:

- Before you evaluate the right combination of resources for your model or application, you should determine the target latency and throughput needs for your overall application stack, as well as any constraints you may have. For example, if your application needs to respond within 300 milliseconds (ms), and data retrieval (including any authentication) and pre-processing takes 200ms, you have a 100ms window to work with for the inference request. Using this analysis, you can determine the lowest cost infrastructure combination that meets these targets.
- Start with a reasonably small combination of resources. For example, a budget-friendly `c5.xlarge` CPU instance type along with an `eia1.medium` accelerator type. This combination has been tested to work well for various computer vision workloads (including a large version of ResNet: ResNet-200) and give comparable or better performance than a more costly `p2.xlarge` GPU instance. You can then upsize on the instance or accelerator type depending on your latency targets.
- I/O data transfer between instance and accelerator adds to inference latency because Elastic Inference accelerators are attached over the network.
- If you use multiple models with your accelerator (or, the same model from multiple application processes on the instance), you might need a larger accelerator size to support both the compute and memory needs on the accelerator.
- You can convert your model to mixed precision, which utilizes the higher FP16 TFLOPS of Elastic Inference (for a given size), to provide lower latency and higher performance.

Using Amazon Elastic Inference with EC2 Auto Scaling

When you create an Auto Scaling group, you can specify the information required to configure the Amazon EC2 instances. This includes Elastic Inference accelerators. To do this, specify a launch template with your instance configuration and the Elastic Inference accelerator type.

Working with Amazon Elastic Inference

After you set up and launch your EC2 instance with Elastic Inference, you can use Elastic Inference accelerators powered by the EI enabled versions of TensorFlow, TensorFlow Serving, and Apache MXNet, with few changes to your code.

Topics

Setting Up to Launch Amazon EC2 with Elastic Inference

The most convenient way to setup Amazon EC2 with Elastic Inference uses the EI setup script described in https://aws.amazon.com/blogs/machine-learning/launch-ei-accelerators-in-minutes-with-the-amazon-elastic-inference-setup-tool-for-ec2/. To manually launch an instance and associate it with an Elastic Inference accelerator, you must first configure your security groups and AWS PrivateLink endpoint services. Then, you must configure an instance role with the Elastic Inference policy.

Topics

Configuring Your Security Groups for Elastic Inference

You need two security groups: one for inbound and outbound traffic for the new Elastic Inference VPC endpoint and another for outbound traffic for the associated EC2 instances that you launch.

Configure Your Security Groups for Elastic Inference

To configure a security group for an Elastic Inference accelerator (console)

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the left navigation pane, choose **Security**, **Security Groups**.

3. Choose **Create Security Group**
4. Under **Create Security Group**, specify a name and description for the security group and choose the ID of the VPC. Choose **Create** and then choose **Close**.
5. Select the check box next to your security group and choose **Actions, Edit inbound rules**. Add a rule to allow HTTPS traffic on port 443 as follows:

   a. Choose **Add Rule**.

   b. For **Type**, select **HTTPS**.

   c. For **Source**, specify a CIDR block (for example, 0.0.0.0/0) or the security group for your instance.

   d. To allow traffic for port 22 to the EC2 instance, repeat the procedure. For **Type**, select **SSH**.

   e. Choose **Save rules** and then choose **Close**.
6. Choose **Edit outbound rules**. Choose **Add rule**. To allow traffic for all ports, for **Type**, select **All Traffic**.
7. Choose **Save rules**.

To configure a security group for an Elastic Inference accelerator (AWS CLI)

1. Create a security group using the create-security-group command:

```
aws ec2 create-security-group
--description insert a description for the security group
--group-name assign a name for the security group
[--vpc-id enter the VPC ID]
```

2. Create inbound rules using the authorize-security-group-ingress command:

```
aws ec2 authorize-security-group-ingress --group-id insert the security group ID --
protocol tcp --port 443 --cidr 0.0.0.0/0
```

```
aws ec2 authorize-security-group-ingress --group-id insert the security group ID --
protocol tcp --port 22 --cidr 0.0.0.0/0
```

3. The default setting for outbound rules allows all traffic from all ports for this instance.

Configuring AWS PrivateLink Endpoint Services

Elastic Inference uses VPC endpoints to privately connect the instance in your VPC with their associated Elastic Inference accelerator. You must create a VPC endpoint for Elastic Inference before you launch instances with accelerators. This needs to be done just one time per VPC. For more information, see Interface VPC Endpoints (AWS PrivateLink).

To configure an AWS PrivateLink endpoint service (console)

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the left navigation pane, choose **Endpoints**, **Create Endpoint**.
3. For **Service category**, choose **Find service by name**.
4. For **Service Name**, select **com.amazonaws.*<your-region>*.elastic-inference.runtime**.

   For example, for the us-west-2 region, select **com.amazonaws.us-west-2.elastic-inference.runtime**.
5. For **Subnets**, select one or more Availability Zones where the endpoint should be created. Where you plan to launch instances with accelerators, you must select subnets for the Availability Zone.

6. Enable the private DNS name and enter the security group for your endpoint. Choose **Create endpoint**. Note the VPC endpoint ID for later.
7. The security group that we configured for the endpoint in previous steps must allow inbound traffic to port 443.

To configure a AWS PrivateLink endpoint service (AWS CLI)

- Use the https://docs.aws.amazon.com/cli/latest/reference/ec2/create-vpc-endpoint.html command and specify the VPC ID, type of VPC endpoint (interface), service name, subnets to use the endpoint, and security groups to associate with the endpoint network interfaces. For information about how to set up a security group for your VPC endpoint, see the section called "Configuring Your Security Groups for Elastic Inference" (p. 5).

```
aws ec2 create-vpc-endpoint --vpc-id vpc-insert VPC ID --vpc-endpoint-type Interface
 --service-name com.amazonaws.us-west-2.elastic-inference.runtime --subnet-id
 subnet-insert subnet --security-group-id sg-insert security group ID
```

Configuring an Instance Role with an Elastic Inference Policy

To launch an instance with an Elastic Inference accelerator, you must provide an IAM role that allows actions on Elastic Inference accelerators.

To configure an instance role with an Elastic Inference policy (console)

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the left navigation pane, choose **Policies**, **Create Policy**.
3. Choose **JSON** and paste the following policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elastic-inference:Connect",
                "iam:List*",
                "iam:Get*",
                "ec2:Describe*",
                "ec2:Get*"
            ],
            "Resource": "*"
        }
    ]
}
```

Note
You may get a warning message about the elastic-inference service not being recognizable. This is a known issue and does not block creation of the policy.

4. Choose **Review policy** and enter a name for the policy, such as `ec2-role-trust-policy.json`, and a description.
5. Choose **Create policy**.
6. In the left navigation pane, choose **Roles**, **Create role**.

7. Choose **AWS service**, **EC2**, **Next: Permissions**.
8. Select the name of the policy that you just created (`ec2-role-trust-policy.json`). Choose **Next: Tags**.
9. Provide a role name and choose **Create Role**.

When you create your instance, select the role under **Configure Instance Details** in the launch wizard.

To configure an instance role with an Elastic Inference policy (AWS CLI)

- To configure an instance role with an Elastic Inference policy, follow the steps in Creating an IAM Role. Add the following policy to your instance:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elastic-inference:Connect",
                "iam:List*",
                "iam:Get*",
                "ec2:Describe*",
                "ec2:Get*"
            ],
            "Resource": "*"
        }
    ]
}
```

Note
You may get a warning message about the elastic-inference service not being recognizable. This is a known issue and does not block creation of the policy.

Launching an Instance with Elastic Inference

You can now configure EC2 instances with accelerators to launch within your subnet. You can choose any supported Amazon EC2 instance type and Elastic Inference accelerator size. Elastic Inference accelerators are available to all current generation instance types. There are three Elastic Inference accelerator sizes to choose from:

- `eia1.medium` with 1 GB of accelerator memory
- `eia1.large` with 2 GB of accelerator memory
- `eia1.xlarge` with 4 GB of accelerator memory

You can launch an instance with Elastic Inference automatically by using the Amazon Elastic Inference setup tool for EC2, or manually using the console or Amazon CLI.

To launch an instance with Elastic Inference (console)

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. Choose **Launch Instance**.
3. Under **Choose an Amazon Machine Image**, select an Amazon Linux or Ubuntu AMI. We recommend one of the Deep Learning AMIs.
4. Under **Choose an Instance Type**, select the hardware configuration of your instance.

5. Choose **Next: Configure Instance Details**.
6. Under **Configure Instance Details**, check the configuration settings. Ensure that you are using the VPC with the security groups for the instance and the Elastic Inference accelerator that you set up earlier. For more information, see Configuring Your Security Groups for Elastic Inference (p. 5).
7. For **IAM role**, select the role that you created in the Configuring an Instance Role with an Elastic Inference Policy (p. 7) procedure.
8. Select **Add an Elastic Inference accelerator**.
9. Select the size of the Elastic Inference accelerator. Your options are: `eia1.medium`, `eia1.large`, and `eia1.xlarge`.
10. (Optional) You can choose to add storage and tags by choosing **Next** at the bottom of the page. Or, you can let the instance wizard complete the remaining configuration steps for you.
11. In the Add Security Group step, choose the security group created previously.
12. Review the configuration of your instance and choose **Launch**.
13. You are prompted to choose an existing key pair for your instance or to create a new key pair. For more information, see Amazon EC2 Key Pairs..

    Warning
    Don't select the **Proceed without a key pair** option. If you launch your instance without a key pair, then you can't connect to it.
14. After making your key pair selection, choose **Launch Instances**.
15. A confirmation page lets you know that your instance is launching. To close the confirmation page and return to the console, choose **View Instances**.
16. Under **Instances**, you can view the status of the launch. It takes a short time for an instance to launch. When you launch an instance, its initial state is `pending`. After the instance starts, its state changes to `running`.
17. It can take a few minutes for the instance to be ready so that you can connect to it. Check that your instance has passed its status checks. You can view this information in the **Status Checks** column.

To launch an instance with Elastic Inference (AWS CLI)

To launch an instance with Elastic Inference at the command line, you need your key pair name, subnet ID, security group ID, AMI ID, and the name of the instance profile that you created in the section Configuring an Instance Role with an Elastic Inference Policy (p. 7). For the security group ID, use the one you created for your instance that contains the AWS PrivateLink endpoint. For more information, see Configuring Your Security Groups for Elastic Inference (p. 5)). For more information about the AMI ID, see Finding a Linux AMI.

1. Use the run-instances command to launch your instance and accelerator:

```
aws ec2 run-instances --image-id ami-image ID --instance-type m5.large --subnet-id
 subnet-subnet ID --elastic-inference-accelerator Type=eia1.large --key-name key
 pair name --security-group-ids sg-security group ID --iam-instance-profile
 Name="accelerator profile name"
```

2. When the `run-instances` operation succeeds, your output is similar to the following. The ElasticInferenceAcceleratorArn identifies the Elastic Inference accelerator.

```
"ElasticInferenceAcceleratorAssociations": [
 {
 "ElasticInferenceAcceleratorArn": "arn:aws:elastic-
inference:us-west-2:204044812891:elastic-inference-accelerator/
eia-3e1de7c2f64a4de8b970c205e838af6b",
 "ElasticInferenceAcceleratorAssociationId": "eia-assoc-031f6f53ddcd5f260",
 "ElasticInferenceAcceleratorAssociationState": "associating",
```

```
"ElasticInferenceAcceleratorAssociationTime": "2018-10-05T17:22:20.000Z"
}
],
```

You are now ready to run your models using either TensorFlow or MXNet on the provided AMI.

Using TensorFlow Models with Elastic Inference

Amazon Elastic Inference(EI) is available only on instances that were launched with an Elastic Inference Accelerator.

The Elastic Inference enabled version of TensorFlow allows you to use Elastic Inference accelerators with minimal changes to your TensorFlow code.

Topics

Elastic Inference Enabled TensorFlow

Preinstalled EI Enabled TensorFlow

The Elastic Inference enabled packages are available in the AWS Deep Learning AMI. You also have Docker container options.

Installing EI Enabled TensorFlow

If you're not using a DLAMI instance, you can download the packages from the Amazon S3 bucket to build it in to your own Amazon Linux or Ubuntu AMIs.

Additional Requirements and Considerations

Model Formats Supported

Elastic Inference supports the TensorFlow saved_model format via TensorFlow Serving.

OpenSSL Requirement

Elastic Inference TensorFlow Serving requires OpenSSL for IAM authentication. OpenSSL is pre-installed in the AWS Deep Learning AMI. If you are building your own AMI or Docker container, you must install OpenSSL .

- Command to install OpenSSL for Ubuntu:

```
sudo apt-get install libssl-dev
```

- Command to install OpenSSL for Amazon Linux:

```
sudo yum install openssl-devel
```

Warmup

Elastic Inference TensorFlow Serving provides a warmup feature to preload models and reduce the delay that is typical of the first inference request. Amazon Elastic Inference TensorFlow Serving only supports warming up the "fault-finders" signature definition.

Signature Definitions

Using multiple signature definitions can have a multiplicative effect on the amount of accelerator memory consumed. If you plan to exercise more than one signature definition for your inference calls, you should test these scenarios as you determine the accelerator type for your application.

For large models, EI tends to have larger memory overhead. This may lead to an out-of-memory error. If you receive this error, try switching to a higher EI Accelerator type.

TensorFlow Elastic Inference with Python

With Elastic Inference TensorFlow Serving, the standard TensorFlow Serving interface remains unchanged. The only difference is that the entry point is a different binary named `amazonei_tensorflow_model_server`.

TensorFlow Serving and Predictor are the only inference modes that EI supports. If you haven't tried TensorFlow Serving before, we recommend that you try the TensorFlow Serving tutorial first.

This release of Elastic Inference TensorFlow Serving has been tested to perform well and provide cost-saving benefits with the following deep learning use cases and network architectures (and similar variants):

Use Case	Example Network Topology
Image Recognition	Inception, ResNet, MVCNN
Object Detection	SSD, RCNN
Neural Machine Translation	GNMT

Topics

Activate the Tensorflow Amazon EI Environment

1. **Note**
   This tutorial assumes usage of a DLAMI with Tensorflow Amazon EI.
   - (Option for Python 3) - Activate the Python 3 TensorFlow EI environment:

     ```
     $ source activate amazonei_tensorflow_p36
     ```

   - (Option for Python 2) - Activate the Python 2.7 TensorFlow EI environment:

     ```
     $ source activate amazonei_tensorflow_p27
     ```

2. The remaining parts of this guide assume you are using the `amazonei_tensorflow_p27` environment.

If you are switching between MXNet or TensorFlow Elastic Inference environments, you must Stop and then Start your instance to reattach the Elastic Inference Accelerator. Rebooting is not sufficient since the process requires a complete shutdown.

Use Elastic Inference with TensorFlow Serving

The following is an example of serving a Single Shot Detector (SSD) with a Resnet backbone. As a general rule, you need a servable model and client scripts downloaded to your DLAMI.

Serve and Test Inference with an Inception Model

1. Download the model.

```
curl -O https://s3-us-west-2.amazonaws.com/aws-tf-serving-ei-example/ssd_resnet.zip
```

2. Unzip the model.

```
unzip ssd_resnet.zip -d /tmp
```

3. Download a picture of three dogs to your home directory.

```
curl -O https://raw.githubusercontent.com/awslabs/mxnet-model-server/master/docs/
images/3dogs.jpg
```

4. Navigate to the folder where `AmazonEI_TensorFlow_Serving` is installed and run the following command to launch the server. Note, "model_base_path" must be an absolute path.

```
amazonei_tensorflow_model_server --model_name=ssdresnet --model_base_path=/tmp/
ssd_resnet50_v1_coco --port=9000
```

5. While the server is running in the foreground, launch another terminal session. Open a new terminal and activate the TensorFlow environment.

```
source activate amazonei_tensorflow_p27
```

6. Use your preferred text editor to create a script that has the following content. Name it `ssd_resnet_client.py`. This script will take an image filename as a parameter and get a prediction result from the pre-trained model.

```
from __future__ import print_function

import grpc
import tensorflow as tf
from PIL import Image
import numpy as np
import time
import os
from tensorflow_serving.apis import predict_pb2
from tensorflow_serving.apis import prediction_service_pb2_grpc

tf.app.flags.DEFINE_string('server', 'localhost:9000',
                           'PredictionService host:port')
tf.app.flags.DEFINE_string('image', '', 'path to image in JPEG format')
FLAGS = tf.app.flags.FLAGS
```

```
coco_classes_txt = "https://raw.githubusercontent.com/amikelive/coco-labels/master/
coco-labels-paper.txt"
local_coco_classes_txt = "/tmp/coco-labels-paper.txt"
# it's a file like object and works just like a file
os.system("curl -o %s -O %s"%(local_coco_classes_txt, coco_classes_txt))
NUM_PREDICTIONS = 5
with open(local_coco_classes_txt) as f:
  classes = ["No Class"] + [line.strip() for line in f.readlines()]


def main(_):
  channel = grpc.insecure_channel(FLAGS.server)
  stub = prediction_service_pb2_grpc.PredictionServiceStub(channel)

  # Send request
  with Image.open(FLAGS.image) as f:
    f.load()
    # See prediction_service.proto for gRPC request/response details.
    data = np.asarray(f)
    data = np.expand_dims(data, axis=0)

    request = predict_pb2.PredictRequest()
    request.model_spec.name = 'ssdresnet'
    request.inputs['inputs'].CopyFrom(
        tf.contrib.util.make_tensor_proto(data, shape=data.shape))
    result = stub.Predict(request, 60.0)  # 10 secs timeout
    outputs = result.outputs
    detection_classes = outputs["detection_classes"]
    detection_classes = tf.make_ndarray(detection_classes)
    num_detections = int(tf.make_ndarray(outputs["num_detections"])[0])
    print("%d detection[s]" % (num_detections))
    class_label = [classes[int(x)]
                   for x in detection_classes[0][:num_detections]]
    print("SSD Prediction is ", class_label)


if __name__ == '__main__':
  tf.app.run()
```

7. Now run the script passing the server location, port, and the dog photo's filename as the parameters.

```
python ssd_resnet_client.py  --server=localhost:9000 --image 3dogs.jpg
```

Use Elastic Inference with the TensorFlow EIPredictor API

Elastic Inference TensorFlow packages for Python 2 and 3 provide an EIPredictor API. This API function provides you with a flexible way to run models on EI as an alternative to using TensorFlow Serving. The EIPredictor API provides a simple interface to perform repeated inference on a pre-trained model. The following code sample shows the available parameters.

```
ei_predictor = EIPredictor(model_dir,
           signature_def_key=None,
           signature_def=None,
           input_names=None,
           output_names=None,
           tags=None,
           graph=None,
           config=None,
           use_ei=True)
```

```
output_dict = ei_predictor(feed_dict)
```

Thus use of EIPredictor is similar to TensorFlow Predictor for a saved model . EIPredictor can be used in the following ways:

```
//EIPredictor class picks inputs and outputs from default serving signature def with tag
 “serve”. (similar to TF predictor)
ei_predictor = EIPredictor(model_dir)

//EI Predictor class picks inputs and outputs from the signature def picked using the
 signtaure_def_key (similar to TF predictor)
ei_predictor = EIPredictor(model_dir, signature_def_key='predict')

// Signature_def can be provided directly (similar to TF predictor)
ei_predictor = EIPredictor(model_dir, signature_def= sig_def)

// You provide the input_names and output_names dict.
// similar to TF predictor

ei_predictor = EIPredictor(model_dir,
                           input_names,
                           output_names)


// tag is used to get the correct signature def. (similar to TF predictor)

ei_predictor = EIPredictor(model_dir, tags='serve')
```

Additional EI Predictor functionality includes:

- Support for frozen models.

```
// For Frozen graphs, model_dir takes a file name , input_names and output_names
// input_names and output_names should be provided in this case.

ei_predictor = EIPredictor(model_dir,
                           input_names=None,
                           output_names=None )
```

- Ability to disable use of EI by using the `use_ei` flag, which is defaulted to `True`. This is useful for testing EIPredictor against TensorFlow Predictor.
- EIPredictor can also be created from a TensorFlow Estimator. Given a trained Estimator, you can first export a SavedModel. See the SavedModel documentation for more details. The following shows example usage:

```
saved_model_dir = estimator.export_savedmodel(my_export_dir, serving_input_fn)
ei_predictor = EIPredictor(export_dir=saved_model_dir)

// Once the EIPredictor is created, inference is done using the following:
output_dict = ei_predictor(feed_dict)
```

Use Elastic Inference with TensorFlow Predictor

Installing Elastic Inference TensorFlow

EI enabled TensorFlow comes bundled in the Deep Learning AMIs. You can also download pip wheels for Python 2 and 3 from the Elastic Inference S3 bucket. Follow these instructions to download and install the pip package:

Choose the tar file for the Python version and operating system of your choice from the S3 bucket. Copy the path to the tar file and run the following command:

```
curl -O [URL of the tar file of your choice]
```

To untar the file:

```
tar -xvzf [name of tar file]
```

Try the following example to serve different models, such as ResNet, using a Single Shot Detector (SSD). As a general rule, you need a servable model and client scripts downloaded to your Deep Learning AMI (DLAMI) before proceeding.

Serve and Test Inference with an SSD Model

1. Download the model. If you already downloaded the model in the Serving example, skip this step.

   ```
   curl -O https://s3-us-west-2.amazonaws.com/aws-tf-serving-ei-example/ssd_resnet.zip
   ```

2. Unzip the model. Again, you may skip this step if you already have the model.

   ```
   unzip ssd_resnet.zip -d /tmp
   ```

3. Download a picture of three dogs to your current directory.

   ```
   curl -O https://raw.githubusercontent.com/awslabs/mxnet-model-server/master/docs/
   images/3dogs.jpg
   ```

4. Open a text editor, such as vim, and paste the following inference script. Save the file as `ssd_resnet_predictor.py`.

   ```
   from __future__ import absolute_import
   from __future__ import division
   from __future__ import print_function

   import os
   import sys
   import numpy as np
   import tensorflow as tf
   import matplotlib.image as mpimg
   from tensorflow.contrib.ei.python.predictor.ei_predictor import EIPredictor

   tf.app.flags.DEFINE_string('image', '', 'path to image in JPEG format')
   FLAGS = tf.app.flags.FLAGS

   coco_classes_txt = "https://raw.githubusercontent.com/amikelive/coco-labels/master/
   coco-labels-paper.txt"
   local_coco_classes_txt = "/tmp/coco-labels-paper.txt"
   # it's a file like object and works just like a file
   os.system("curl -o %s -O %s"%(local_coco_classes_txt, coco_classes_txt))
   NUM_PREDICTIONS = 5
   with open(local_coco_classes_txt) as f:
     classes = ["No Class"] + [line.strip() for line in f.readlines()]


   def get_output(eia_predictor, test_input):
     pred = None
     for curpred in range(NUM_PREDICTIONS):
       pred = eia_predictor(test_input)
   ```

```
  num_detections = int(pred["num_detections"])
  print("%d detection[s]" % (num_detections))
  detection_classes = pred["detection_classes"][0][:num_detections]
  print([classes[int(i)] for i in detection_classes])


def main(_):

  img = mpimg.imread(FLAGS.image)
  img = np.expand_dims(img, axis=0)
  ssd_resnet_input = {'inputs': img}

  print('Running SSD Resnet on EIPredictor using specified input and outputs')
  eia_predictor = EIPredictor(
      model_dir='/tmp/ssd_resnet50_v1_coco/1/',
      input_names={"inputs": "image_tensor:0"},
      output_names={"detection_classes": "detection_classes:0", "num_detections":
 "num_detections:0",
                    "detection_boxes": "detection_boxes:0"},
  )
  get_output(eia_predictor, ssd_resnet_input)

  print('Running SSD Resnet on EIPredictor using default Signature Def')
  eia_predictor = EIPredictor(
      model_dir='/tmp/ssd_resnet50_v1_coco/1/',
  )
  get_output(eia_predictor, ssd_resnet_input)


if __name__ == "__main__":
  tf.app.run()
```

5. Run the inference script.

```
python ssd_resnet_predictor.py --image 3dogs.jpg
```

For more tutorials and examples, see the TensorFlow Python API.

Using MXNet Models with Elastic Inference

This release of Elastic Inference Apache MXNet has been tested to perform well and provide cost-saving benefits with the following deep learning use cases and network architectures (and similar variants).

Use Case	Example Network Topology
Image Recognition	Inception, ResNet, VGG, ResNext
Object Detection	SSD
Text to Speech	WaveNet

Topics

More Models and Resources

Here are some more pre-trained models and examples to try with EI.

1. MXNet Model Zoo - these Gluon models can be exported to the Symbol format and used with EI.
2. Open Neural Network Exchange (ONNX) Models with MXNet - MXNet natively supports the ONNX model format, so you can use EI with ONNX models that were exported from other frameworks.

For more tutorials and examples, see the framework's official Python documentation, the Python API for MXNet, or the website.

MXNet Elastic Inference with Python

The Amazon Elastic Inference(EI) enabled version of Apache MXNet lets you use Elastic Inference seamlessly, with few changes to your MXNet code. To use an existing MXNet inference script, make one change in the code. Wherever you set the context to bind your model, such as `mx.cpu()` or `mx.gpu()`, update this to use `mx.eia()` instead. You can use Elastic Inference with the following MXNet API operations:

- MXNet Python Symbol API
- MXNet Python Module API

Topics

Elastic Inference Enabled Apache MXNet

For more information on MXNet setup, see Apache MXNet on AWS.

Preinstalled EI Enabled MXNet

Elastic Inference enabled Apache MXNet is available in the AWS Deep Learning AMI.

Installing EI Enabled MXNet

If you're not using a DLAMI instance, a 'pip' package is available on Amazon S3 so you can build it in to your own Amazon Linux or Ubuntu AMIs using the following command:

```
pip install "latest-wheel"
```

Activate the MXNet Elastic Inference Environment

If you are using the AWS Deep Learning AMI, activate the Python 3 MXNet Elastic Inference environment or Python 2 MXNet Elastic Inference environment, depending on your version of Python.

For Python 3:

```
source activate amazonei_mxnet_p36
```

For Python 2:

```
source activate amazonei_mxnet_p27
```

If you are using a different AMI or a container, access the environment where MXNet is installed.

Test MXNet

Verify that you've properly set up your instance with EI.

```
$ python ~/anaconda3/bin/EISetupValidator.py
```

If your instance is not properly set up with an accelerator, running any of the examples in this section will result in the following error:

```
Error: Failed to query accelerator metadata.
Failed to detect any accelerator
```

For detailed instructions on how to launch a DLAMI with an Elastic Inference Accelerator, see the Elastic Inference documentation.

You can verify that MXNet is available to use and check the current version with the following code from the Python terminal:

```
>>> import mxnet
>>> mxnet.__version__
'1.4.0'
```

Use Elastic Inference with the MXNet Symbol API

Pass `mx.eia()` as the context in a call to either the `simple_bind()` or the `bind()` methods. For information, see MXNet Symbol API.

The EI environment is not compiled with CUDA, so GPU context is not supported. Check for this in your code to make sure that you don't accidentally use the GPU context. For example, NDArray computations should be handled by the `mx.cpu()` context when using EI. You use the `mx.eia()` context only with the bind call. The following example calls the `simple_bind()` method with the `mx.eia()` context:

```
import mxnet as mx

data = mx.sym.var('data', shape=(1,))
sym = mx.sym.exp(data)

# Pass mx.eia() as context during simple bind operation

executor = sym.simple_bind(ctx=mx.eia(), grad_req='null')
for i in range(10):

  # Forward call is performed on remote accelerator
  executor.forward(data=mx.nd.ones((1,)))
```

```
  print('Inference %d, output = %s' % (i, executor.outputs[0]))
```

The following example calls the `bind()` method:

```
import mxnet as mx
a = mx.sym.Variable('a')
b = mx.sym.Variable('b')
c = 2 * a + b
# Even for execution of inference workloads on eia,
# context for input ndarrays to be mx.cpu()
a_data = mx.nd.array([1,2], ctx=mx.cpu())
b_data = mx.nd.array([2,3], ctx=mx.cpu())
# Then in the bind call, use the mx.eia() context
e = c.bind(mx.eia(), {'a': a_data, 'b': b_data})

# Forward call is performed on remote accelerator
e.forward()
print('1st Inference, output = %s' % (e.outputs[0]))
# Subsequent calls can pass new data in a forward call
e.forward(a=mx.nd.ones((2,)), b=mx.nd.ones((2,)))
print('2nd Inference, output = %s' % (e.outputs[0]))
```

The following example calls the `bind()` method on a pre-trained real model (Resnet-50) from the Symbol API. Use your preferred text editor to create a script called mxnet_resnet50.py that has the following content. This script downloads the ResNet-50 model files (resnet-50-0000.params and resnet-50-symbol.json), list of labels(synset.txt) and an image of a cat. The cat image is used to get a prediction result from the pre-trained model. This result is looked up in the list of labels, returning a prediction result.

```
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.eia()

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(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]))
```

Then run the script, and you should see something similar to the following output. MXNet will optimize the model graph for Elastic Inference, load it on Elastic Inference Accelerator, and then run inference against it:

```
(amazonei_mxnet_p36) ubuntu@ip-172-31-42-83:~$ python mxnet_resnet50.py
  [23:12:03] src/nnvm/legacy_json_util.cc:209: Loading symbol saved by previous version
 v0.8.0. Attempting to upgrade...
  [23:12:03] src/nnvm/legacy_json_util.cc:217: Symbol successfully upgraded!
  Using Amazon Elastic Inference Client Library Version: 1.2.8
  Number of Elastic Inference Accelerators Available: 1
  Elastic Inference Accelerator ID: eia-95ae5a472b2241769656dbb5d344a80e
  Elastic Inference Accelerator Type: eia1.large

  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
```

Use Elastic Inference with the MXNet Module API

When you create the `Module` object, pass `mx.eia()` as the context. For more information, see Module API.

To use the MXNet Module API, you can use the following commands:

```
# Load saved model
sym, arg_params, aux_params = mx.model.load_checkpoint(model_path, EPOCH_NUM)

# Pass mx.eia() as context while creating Module object
mod = mx.mod.Module(symbol=sym, context=mx.eia())

# Only for_training = False is supported for eia
mod.bind(for_training=False, data_shapes=data_shape)
mod.set_params(arg_params, aux_params)

# forward call is performed on remote accelerator
mod.forward(data_batch)
```

The following script downloads two ResNet-152 model files (resnet-152-0000.params and resnet-152-symbol.json) and a labels list (synset.txt). It also downloads a cat image to get a prediction result from the pre-trained model, then looks this up in the result in labels list, returning a prediction result. Use your preferred text editor to create a script using the following content:

```
import mxnet as mx
import numpy as np
from collections import namedtuple

Batch = namedtuple('Batch', ['data'])


path='http://data.mxnet.io/models/imagenet/'
[mx.test_utils.download(path+'resnet/152-layers/resnet-152-0000.params'),
mx.test_utils.download(path+'resnet/152-layers/resnet-152-symbol.json'),
```

```
mx.test_utils.download(path+'synset.txt')]


ctx = mx.eia()


sym, arg_params, aux_params = mx.model.load_checkpoint('resnet-152', 0)
mod = mx.mod.Module(symbol=sym, context=ctx, label_names=None)
mod.bind(for_training=False, data_shapes=[('data', (1,3,224,224))],
     label_shapes=mod._label_shapes)
mod.set_params(arg_params, aux_params, allow_missing=True)

with open('synset.txt', 'r') as f:
  labels = [l.rstrip() for l in f]

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


mod.forward(Batch([img]))
prob = mod.get_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]))
```

Save this script as test.py

Additional Requirements and Considerations

- MXNet EI is built with MKL-DNN, so all operations using mx.cpu() are supported and will run with the same performance as the standard release. MXNet EI does not support mx.gpu(), so all operations using that context will throw an error. Sample error message:

```
>>> mx.nd.ones((1),ctx=mx.gpu())
[20:35:32] src/imperative/./ imperative_utils.h:90: GPU support is disabled. Compile
 MXNet with USE_CUDA=1 to enable GPU support.
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/ubuntu/deps/MXNetECL /python/mxnet/ndarray/ndarray.py", line 2421, in ones
    return _internal._ones(shape=shape, ctx=ctx, dtype=dtype, **kwargs)
  File "<string>", line 34, in _ones
  File "/home/ubuntu/deps/MXNetECL /python/mxnet/_ctypes/ndarray.py", line 92, in
 _imperative_invoke
    ctypes.byref(out_stypes)))
  File "/home/ubuntu/deps/MXNetECL /python/mxnet/base.py", line 252, in check_call
    raise MXNetError(py_str(_LIB.MXGetLastError()))
mxnet.base.MXNetError: [20:35:32] src/imperative/imperative.cc:79: Operator _ones is not
 implemented for GPU.
```

- You cannot allocate memory for NDArray on the remote accelerator by writing something like this:

```
 x = mx.nd.array([[1, 2], [3, 4]],
                                ctx=mx.eia())
```

This throws an error. Instead you should use `mx.cpu()`. Look at the previous `bind()` example to see how MXNet automatically transfers your data to the accelerator as necessary. Sample error message:

```
>>> mx.nd.array([1,2],ctx=mx.eia())
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/ec2-user/.local/lib/python3.6/site-packages/mxnet/ndarray/utils.py", line
 146, in array
    return _array(source_array, ctx=ctx, dtype=dtype)
  File "/home/ec2-user/.local/lib/python3.6/site-packages/mxnet/ndarray/ndarray.py", line
 2434, in array
    arr = empty(source_array.shape, ctx, dtype)
  File "/home/ec2-user/.local/lib/python3.6/site-packages/mxnet/ndarray/ndarray.py", line
 3820, in empty
    return NDArray(handle=_new_alloc_handle(shape, ctx, False, dtype))
  File "/home/ec2-user/.local/lib/python3.6/site-packages/mxnet/ndarray/ndarray.py", line
 139, in _new_alloc_handle
    ctypes.byref(hdl)))
  File "/home/ec2-user/.local/lib/python3.6/site-packages/mxnet/base.py", line 252, in
 check_call
    raise MXNetError(py_str(_LIB.MXGetLastError()))
mxnet.base.MXNetError: [21:51:47] src/storage/storage.cc:145: Unimplemented device 6

Stack trace returned 10 entries:
[bt] (0) /home/ec2-user/.local/lib/python3.6/site-packages/mxnet/
libmxnet.so(dmlc::StackTrace()+0x44) [0x7fc9ac6a8b34]
[bt] (1) /home/ec2-user/.local/lib/python3.6/site-packages/mxnet/
libmxnet.so(dmlc::LogMessageFatal::~LogMessageFatal()+0x21) [0x7fc9ac6a8f11]
[bt] (2) /home/ec2-user/.local/lib/python3.6/site-packages/mxnet/libmxnet.so(+0x366e697)
 [0x7fc9af789697]
[bt] (3) /home/ec2-user/.local/lib/python3.6/site-packages/mxnet/libmxnet.so(+0x367031f)
 [0x7fc9af78b31f]
[bt] (4) /home/ec2-user/.local/lib/python3.6/site-packages/mxnet/
libmxnet.so(mxnet::StorageImpl::Alloc(mxnet::Storage::Handle*)+0x3f) [0x7fc9af78c22f]
[bt] (5) /home/ec2-user/.local/lib/python3.6/site-packages/mxnet/libmxnet.so(+0x595cf9)
 [0x7fc9ac6b0cf9]
[bt] (6) /home/ec2-user/.local/lib/python3.6/site-packages/mxnet/
libmxnet.so(mxnet::NDArray::NDArray(nnvm::TShape const, mxnet::Context, bool, int)+0x7fa)
 [0x7fc9ac6bc76a]
[bt] (7) /home/ec2-user/.local/lib/python3.6/site-packages/mxnet/
libmxnet.so(MXNDArrayCreateEx+0x17d) [0x7fc9aef87e7d]
[bt] (8) /usr/lib64/libffi.so.6(ffi_call_unix64+0x4c) [0x7fca559bbcec]
[bt] (9) /usr/lib64/libffi.so.6(ffi_call+0x1f5) [0x7fca559bb615]
```

- Elastic Inference is only for production inference use cases and does not support any model training. When you use either the Symbol API or the Module API, do not call the `backward()` method or call `bind()` with `for_training=True`. This throws an error. Because the default value of `for_training` is `True`, make sure you set `for_training=False` manually in cases such as the example in Use Elastic Inference with the MXNet Module API (p. 20). Sample error using test.py:

```
Traceback (most recent call last):
  File "test.py", line 16, in <module>
    label_shapes=mod._label_shapes)
  File "/home/ec2-user/.local/lib/python3.6/site-packages/mxnet/module/module.py", line
 402, in bind
    raise ValueError("for training cannot be set to true with EIA context")
ValueError: for training cannot be set to true with EIA context
```

- Because training is not allowed, there is no point of initializing an optimizer for inference.
- You are limited to using only a single Elastic Inference Accelerator. Therefore, you can not create multiple eia devices (e.g. `mx.context.eia(1)` and `mx.context.eia(2)`) and should only use `mx.context.eia()`with the default device everywhere.

- A model trained on an earlier version of MXNet will work on a later version of MXNet EI because it is backwards compatible (e.g. train model on MXNet 1.3 and run on MXNet EI 1.4). However, you may run into undefined behavior if you train on a later version of MXNet (e.g. train model on MXNet Master and run on MXNet EI 1.4)
- Different sizes of EI accelerators have different amounts of GPU memory. If your model requires more GPU memory than is available in your accelerator, you get a message that looks like the log below. If you run into this message, you should use a larger accelerator size with more memory. Stop and restart your instance with a larger accelerator.

```
mxnet.base.MXNetError: [06:16:17] src/operator/subgraph/eia/eia_subgraph_op.cc:206: Last
 Error:
    EI Error Code: [51, 8, 31]
    EI Error Description: Accelerator out of memory. Consider using a larger accelerator.
    EI Request ID: MX-A19B0DE6-7999-4580-8C49-8EA 7ADSFFCB  --  EI Accelerator ID: eia-
cb0aasdfdfsdf2a acab7
    EI Client Version: 1.2.12
```

- Calling `reshape` explicitly by using either the Module or the Symbol API, or implicitly using different shapes for input `NDArrays` in different forward passes can lead to OOM errors. Before being reshaped, the model is not cleaned up on the accelerator until the session is destroyed.

MXNet Elastic Inference with Java

Starting from Apache MXNet version 1.4, the Java API can now integrate with Amazon Elastic Inference. You can use Elastic Inference with the following MXNet Java API operations:

- MXNet Java Infer API

Topics

Install Amazon EI Enabled Apache MXNet

Amazon Elastic Inference enabled Apache MXNet is available in the AWS Deep Learning AMI. A maven repository is also available on Amazon S3 so you can build it in to your own Amazon Linux or Ubuntu AMIs, or Docker containers.

For Maven projects, Elastic Inference Java can be included by adding the following to your project's `pom.xml`:

```
<repositories>
    <repository>
      <id>Amazon Elastic Inference</id>
      <url>https://s3.amazonaws.com/amazonei-apachemxnet/scala</url>
    </repository>
</repositories>
```

In addition, add the Elastic Inference flavor of MXNet as a dependency using:

```
        <dependency>
```

```
            <groupId>com.amazonaws.ml.mxnet</groupId>
            <artifactId>mxnet-full_2.11-linux-x86_64-eia</artifactId>
            <version>[1.4.0,)</version>
        </dependency>
```

Use Amazon Elastic Inference with the MXNet Java Infer API

To use Amazon Elastic Inference with the MXNet Java Infer API, pass `Context.eia()` as the context when creating the Infer Predictor object. See the MXNet Infer Reference for more information. The following example uses the pre-trained real model (Resnet-152)

```
package mxnet;

import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.net.URL;
import java.util.Arrays;
import java.util.Comparator;
import java.util.List;
import java.util.stream.IntStream;
import org.apache.commons.io.FileUtils;
import org.apache.mxnet.infer.javaapi.ObjectDetector;
import org.apache.mxnet.infer.javaapi.Predictor;
import org.apache.mxnet.javaapi.*;

public class Example {
    public static void main(String[] args) throws IOException {
        String urlPath = "http://data.mxnet.io/models/imagenet";
        String filePath = System.getProperty("java.io.tmpdir");

        // Download Model and Image
        FileUtils.copyURLToFile(new URL(urlPath + "/resnet/152-layers/
resnet-152-0000.params"),
                new File(filePath + "resnet-152/resnet-152-0000.params"));
        FileUtils.copyURLToFile(new URL(urlPath + "/resnet/152-layers/resnet-152-
symbol.json"),
                new File(filePath + "resnet-152/resnet-152-symbol.json"));
        FileUtils.copyURLToFile(new URL(urlPath + "/synset.txt"),
                new File(filePath + "resnet-152/synset.txt"));
        FileUtils.copyURLToFile(new URL("https://github.com/dmlc/web-data/blob/master/
mxnet/doc/tutorials/python/predict_image/cat.jpg?raw=true"),
                new File(filePath + "cat.jpg"));

        List<Context> contexts = Arrays.asList(Context.eia());
        Shape inputShape = new Shape(new int[]{1, 3, 224, 224});
        List<DataDesc> inputDesc = Arrays.asList(new DataDesc("data", inputShape,
 DType.Float32(), "NCHW"));
        Predictor predictor = new Predictor(filePath + "resnet-152/resnet-152", inputDesc,
 contexts, 0);

        BufferedImage originalImg = ObjectDetector.loadImageFromFile(filePath + "cat.jpg");
        BufferedImage resizedImg = ObjectDetector.reshapeImage(originalImg, 224, 224);
        NDArray img = ObjectDetector.bufferedImageToPixels(resizedImg, new Shape(new int[]
{1, 3, 224, 224}));

        List<NDArray> predictResults = predictor.predictWithNDArray(Arrays.asList(img));
        float[] results = predictResults.get(0).toArray();

        List<String> synsetLines = FileUtils.readLines(new File(filePath + "resnet-152/
synset.txt"));

        int[] best = IntStream.range(0, results.length)
```

```
                .boxed().sorted(Comparator.comparing(i -> -results[i]))
                .mapToInt(ele -> ele).toArray();

        for (int i = 0; i < 5; i++) {
            int ind = best[i];
            System.out.println(i + ": " + synsetLines.get(ind) + " - " + best[ind]);
        }

    }
}
```

More Models and Resources

For more tutorials and examples, see the framework's official Java documentation, the Java API Reference, or the Apache MXNet website.

Additional Requirements and Considerations

- MXNet EI is built with MKL-DNN, so all operations using Context.cpu() are supported and will run with the same performance as the standard release. MXNet EI does not support Context.gpu(), so all operations using that context will throw an error.
- You cannot allocate memory for NDArray on the remote accelerator by writing something like this:

```
x = NDArray.array(Array(1,2,3),
            ctx=Context.eia())
```

  This throws an error. Instead you should use `Context.cpu()`. Look at the previous `bind()` example to see how MXNet automatically transfers your data to the accelerator as necessary. Sample error message:
- Elastic Inference is only for production inference use cases and does not support any model training. When you use either the Symbol API or the Module API, do not call the `backward()` method or call `bind()` with `forTraining=True`. This throws an error. Because the default value of `forTraining` is `True`, make sure you set `for_training=False` manually in cases such as the example in Use Elastic Inference with the MXNet Module API (p. 20). Sample error using test.py:
- Because training is not allowed, there is no point of initializing an optimizer for inference.
- You are limited to using only a single Elastic Inference Accelerator. Therefore, you can not create multiple eia devices (e.g. `Context.eia(1)` and `Context.eia(2)`) and should only use `Context.eia()`with the default device everywhere.
- A model trained on an earlier version of MXNet will work on a later version of MXNet EI because it is backwards compatible (e.g. train model on MXNet 1.3 and run on MXNet EI 1.4). However, you may run into undefined behavior if you train on a later version of MXNet (e.g. train model on MXNet Master and run on MXNet EI 1.4)
- Different sizes of EI accelerators have different amounts of GPU memory. If your model requires more GPU memory than is available in your accelerator, you get a message that looks like the log below. If you run into this message, you should use a larger accelerator size with more memory. Stop and restart your instance with a larger accelerator.
- Calling `reshape` explicitly by using either the Module or the Symbol API, or implicitly using different shapes for input `NDArrays` in different forward passes can lead to OOM errors. Before being reshaped, the model is not cleaned up on the accelerator until the session is destroyed.

MXNet Elastic Inference with Scala

Starting from Apache MXNet version 1.4, the Scala API can now integrate with Amazon Elastic Inference. You can use Elastic Inference with the following MXNet Scala API operations:

- MXNet Scala Symbol API
- MXNet Scala Module API
- MXNet Scala Infer API

Topics

Install Elastic Inference Enabled Apache MXNet

Elastic Inference enabled Apache MXNet is available in the AWS Deep Learning AMI. A maven repository is also available on Amazon S3 so you can build it in to your own Amazon Linux or Ubuntu AMIs, or Docker containers.

For Maven projects, Elastic Inference with Scala can be included by adding the following to your project's `pom.xml`:

```
<repositories>
    <repository>
      <id>Amazon Elastic Inference</id>
      <url>https://s3.amazonaws.com/amazonei-apachemxnet/scala</url>
    </repository>
</repositories>
```

In addition, add the Elastic Inference flavor of MXNet as a dependency using:

```
        <dependency>
            <groupId>com.amazonaws.ml.mxnet</groupId>
            <artifactId>mxnet-full_2.11-linux-x86_64-eia</artifactId>
            <version>[1.4.0,)</version>
        </dependency>
```

Use Amazon Elastic Inference with the MXNet Symbol API

To use Elastic Inference with the MXNet Symbol API, pass `Context.eia()` as the context in a call to either the `Symbol.bind` or `Symbol.simpleBind` methods. See the MXNet Symbol Reference for more information.

The following is an example using `Context.eia()` in a call to simpleBind:

```
import org.apache.mxnet._

object Example {

  def main(args: Array[String]): Unit = {
    val data = Symbol.Variable("data", shape=Shape(1))
    val sym = Symbol.api.exp(Some(data))
```

```
    // Pass mx.eia() as context during simple bind operation
    val executor = sym.simpleBind(Context.eia(), gradReq = "null", shapeDict = Map("data" -
> Shape(1)))
    for( i  <- 1 to 10) {
      executor.forward(false, ("data", NDArray.ones(1)))
      println(s"Inference ${i}, output = ${executor.outputs.head}")
    }
  }
}
```

Note, the GPU context is not supported. All non-EIA values and computations should use the CPU context. The EIA context should only be used with the bind call.

The following is an example using bind. Note, you cannot use the EIA context to allocate memory or it will throw an error.

```
import org.apache.mxnet._

object Example {

  def main(args: Array[String]): Unit = {
    val a = Symbol.Variable("a")
    val b = Symbol.Variable("b")
    val c = a + b

    // Even for EIA workloads, declare NDArrays on the CPU
    val aData = NDArray.array(Array(1f,2f), Shape(2), Context.cpu())
    val bData = NDArray.array(Array(2f,3f), Shape(2), Context.cpu())

    // Then in the bind call, use Context.eia()
    val executor = c.bind(Context.eia(), Map("a" -> aData, "b" -> bData))

    // The forward call is performed on the remote accelerator
    executor.forward()
    println(s"1st Inference, output = ${executor.outputs.head}")

    // Subsequent calls can pass new data in a forward call
    executor.forward(false, ("a", NDArray.ones((2))), ("b", NDArray.ones((2))))
    println(s"2nd Inference, output = ${executor.outputs.head}")
  }
}
```

Use Amazon Elastic Inference with the MXNet Module API

To use Elastic Inference with the MXNet Module API, pass `Context.eia()` as the context when creating the `Module` object. See the MXNet Module Reference for more information.

The following is an example using Elastic Inference with the Module API on a pre-trained real model (Resnet-152).

```
import java.io.File
import java.net.URL

import org.apache.commons.io.FileUtils
import org.apache.mxnet._
import org.apache.mxnet.infer.ImageClassifier
import org.apache.mxnet.module.Module
```

```
import scala.io.Source

object Example {

  def main(args: Array[String]): Unit = {
    val urlPath = "http://data.mxnet.io/models/imagenet"
    val filePath = System.getProperty("java.io.tmpdir")

    // Download Model and Image
    FileUtils.copyURLToFile(new URL(s"${urlPath}/resnet/152-layers/
resnet-152-0000.params"),
      new File(s"${filePath}resnet-152/resnet-152-0000.params"))
    FileUtils.copyURLToFile(new URL(s"${urlPath}/resnet/152-layers/resnet-152-
symbol.json"),
      new File(s"${filePath}resnet-152/resnet-152-symbol.json"))
    FileUtils.copyURLToFile(new URL(s"${urlPath}/synset.txt"),
      new File(s"${filePath}resnet-152/synset.txt"))
    FileUtils.copyURLToFile(new URL("https://github.com/dmlc/web-data/blob/master/mxnet/
doc/tutorials/python/predict_image/cat.jpg?raw=true"),
      new File(s"${filePath}cat.jpg"))

    // Load model
    val (symbol, argParams, auxParams) = Model.loadCheckpoint(s"${filePath}resnet-152/
resnet-152", 0)
    val mod = new Module(symbol, contexts = Context.eia(), labelNames = IndexedSeq())
    mod.bind(dataShapes=IndexedSeq(DataDesc("data", Shape(1, 3, 224, 224))), forTraining =
 false)
    mod.setParams(argParams, auxParams, allowMissing = true)
    val labels = Source.fromFile(s"${filePath}resnet-152/
synset.txt").getLines().map(_.trim).toIndexedSeq

    // Load image
    val originalImg = ImageClassifier.loadImageFromFile(s"${filePath}cat.jpg")
    val resizedImg = ImageClassifier.reshapeImage(originalImg, 224, 224)
    val img = ImageClassifier.bufferedImageToPixels(resizedImg, Shape(1, 3, 224, 224))

    mod.forward(new DataBatch(IndexedSeq(img), IndexedSeq(), IndexedSeq(), 0))

    val probabilities = mod.getOutputs().head.head.toArray
    val best = probabilities.zipWithIndex.sortBy(-_._1).take(5)
    best.zipWithIndex.foreach {
      case ((prob, nameIndex), i) => println(s"Option ${i}: ${labels(nameIndex)} -
 ${prob}")
    }
  }
}
```

Use Amazon Elastic Inference with the MXNet Infer API

To use Elastic Inference with the MXNet Infer API, pass `Context.eia()` as the context when creating the Infer Predictor object. See the MXNet Infer Reference for more information. The following example also uses the pre-trained real model (Resnet-152) .

```
import java.io.File
import java.net.URL

import org.apache.commons.io.FileUtils
import org.apache.mxnet._
import org.apache.mxnet.infer.ImageClassifier

object Example {
```

```
  def main(args: Array[String]): Unit = {
    val urlPath = "http://data.mxnet.io/models/imagenet"
    val filePath = System.getProperty("java.io.tmpdir")

    // Download Model and Image
    FileUtils.copyURLToFile(new URL(s"${urlPath}/resnet/152-layers/
resnet-152-0000.params"),
      new File(s"${filePath}resnet-152/resnet-152-0000.params"))
    FileUtils.copyURLToFile(new URL(s"${urlPath}/resnet/152-layers/resnet-152-
symbol.json"),
      new File(s"${filePath}resnet-152/resnet-152-symbol.json"))
    FileUtils.copyURLToFile(new URL(s"${urlPath}/synset.txt"),
      new File(s"${filePath}resnet-152/synset.txt"))
    FileUtils.copyURLToFile(new URL("https://github.com/dmlc/web-data/blob/master/mxnet/
doc/tutorials/python/predict_image/cat.jpg?raw=true"),
      new File(s"${filePath}cat.jpg"))

    val inputShape = Shape(1, 3, 224, 224)
    val inputDesc = IndexedSeq(DataDesc("data", inputShape, DType.Float32, "NCHW"))
    val imgClassifier = new ImageClassifier(s"${filePath}resnet-152/resnet-152", inputDesc,
 Context.eia())

    val img = ImageClassifier.loadImageFromFile(s"${filePath}cat.jpg")
    val topK = 5
    val output = imgClassifier.classifyImage(img, Some(topK)).head

    output.zipWithIndex.foreach{
      case ((name, prob), i) => println(s"Option ${i}: ${name} - ${prob}")
    }
  }
}
```

More Models and Resources

For more tutorials and examples, see the framework's official Scala documentation, the Scala API Reference, or the Apache MXNet website.

Additional Requirements and Considerations

- MXNet EI is built with MKL-DNN, so all operations using Context.cpu() are supported and will run with the same performance as the standard release. MXNet EI does not support Context.gpu(), so all operations using that context will throw an error.
- You cannot allocate memory for NDArray on the remote accelerator by writing something like this:

  ```
  x = NDArray.array(Array(1,2,3),
              ctx=Context.eia())
  ```

  This throws an error. Instead you should use `Context.cpu()`. Look at the previous `bind()` example to see how MXNet automatically transfers your data to the accelerator as necessary. Sample error message:
- Elastic Inference is only for production inference use cases and does not support any model training. When you use either the Symbol API or the Module API, do not call the `backward()` method or call `bind()` with `forTraining=True`. This throws an error. Because the default value of `forTraining` is `True`, make sure you set `for_training=False` manually in cases such as the example in Use Elastic Inference with the MXNet Module API (p. 20). Sample error using test.py:
- Because training is not allowed, there is no point of initializing an optimizer for inference.
- You are limited to using only a single Elastic Inference Accelerator. Therefore, you can not create multiple eia devices (e.g. `Context.eia(1)` and `Context.eia(2)`) and should only use `Context.eia()`with the default device everywhere.

- A model trained on an earlier version of MXNet will work on a later version of MXNet EI because it is backwards compatible (e.g. train model on MXNet 1.3 and run on MXNet EI 1.4). However, you may run into undefined behavior if you train on a later version of MXNet (e.g. train model on MXNet Master and run on MXNet EI 1.4)
- Different sizes of EI accelerators have different amounts of GPU memory. If your model requires more GPU memory than is available in your accelerator, you get a message that looks like the log below. If you run into this message, you should use a larger accelerator size with more memory. Stop and restart your instance with a larger accelerator.
- Calling `reshape` explicitly by using either the Module or the Symbol API, or implicitly using different shapes for input `NDArrays` in different forward passes can lead to OOM errors. Before being reshaped, the model is not cleaned up on the accelerator until the session is destroyed.

Using CloudWatch Metrics to Monitor Elastic Inference

You can monitor your Elastic Inference accelerators using Amazon CloudWatch, which collects metrics about your usage and performance. These statistics are recorded for a period of two weeks so that you can access historical information and gain a better perspective of how your service is performing.

By default, Elastic Inference sends metric data to CloudWatch in 5-minute periods.

For more information, see the Amazon CloudWatch User Guide.

Topics

Elastic Inference Metrics and Dimensions

Metrics are grouped first by the service namespace, and then by the various dimension combinations within each namespace. You can use the following procedures to view the metrics for Elastic Inference.

To view metrics using the CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. If necessary, change the Region. From the navigation bar, select the region where Elastic Inference resides. For more information, see Regions and Endpoints.
3. In the navigation pane, choose **Metrics**.
4. Under **All metrics**, select a metrics category, and then scroll down to view the full list of metrics.

To view metrics (AWS CLI)

- At a command prompt, enter the following command:

```
aws cloudwatch list-metrics --namespace " AWS/ElasticInference "
```

CloudWatch displays the following metrics for Elastic Inference.

Metric	Description
AcceleratorHealthCheckFailed	Reports whether the Elastic Inference accelerator has passed a status health check in the last minute. A value of zero (0) indicates that the status check passed. A value of one (1) indicates a status check failure. Units: Count

Metric	Description
ConnectivityCheckFailed	Reports whether connectivity to the Elastic Inference accelerator is active or has failed in the last minute. A value of zero (0) indicates that the connection is active. A value of one (1) indicates a connectivity failure. Units: Count
AcceleratorMemoryUsage	The memory of the Elastic Inference accelerator used in the last minute. Units: Bytes

You can filter the Elastic Inference data using the following dimensions.

Dimension	Description
ElasticInferenceAcceleratorId	This dimension filters the data by the Elastic Inference accelerator.
InstanceId	This dimension filters the data by instance to which the Elastic Inference accelerator is attached.

Creating CloudWatch Alarms to Monitor Elastic Inference

You can create a CloudWatch alarm that sends an Amazon SNS message when the alarm changes state. An alarm watches a single metric over a time period that you specify. It sends a notification to an SNS topic based on the value of the metric relative to a given threshold over a number of time periods.

For example, you can create an alarm that monitors the health of an Elastic Inference accelerator. It sends a notification when the Elastic Inference accelerator fails a status health check for three consecutive 5-minute periods.

To create an alarm for Elastic Inference accelerator health status

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**, **Create Alarm**.
3. Choose **Amazon EI Metrics**.
4. Select the **Amazon EI** and the **AcceleratorHealthCheckFailed** metric and choose **Next**.
5. Configure the alarm as follows, and then choose **Create Alarm**:

   - Under **Alarm Threshold**, enter a name and description. For **Whenever**, choose **=>** and enter **1**. For the consecutive periods, enter **3**.
   - Under **Actions**, select an existing notification list or choose **New list**.
   - Under **Alarm Preview**, select a period of 5 minutes.

Troubleshooting

The following are common Amazon Elastic Inference errors and troubleshooting steps.

Topics

Issues Launching Accelerators

Ensure that you are launching in a Region where Elastic Inference accelerators are available. For more information, see the Region Table.

Resolving Configuration Issues

If you launched your instance with the Deep Learning AMI (DLAMI), run `python ~/anaconda3/bin/EISetupValidator.py` to verify that the instance is correctly configured to use the Elastic Inference service. You can also download the EISetupValidator.py script and execute `'python EISetupValidator.py`.

Resolving Connectivity Issues

If you are unable to successfully connect to accelerators, verify that you have completed the following:

- Set up a Virtual Private Cloud (VPC) endpoint for Elastic Inference for the subnet in which you have launched your instance.
- Configure security groups for the instance and VPC endpoints with outbound rules that allow communications for HTTPS (Port 443). Configure the VPC endpoint security group with an inbound rule that allows HTTPS traffic.
- Add an IAM instance role with the **elastic-inference:Connect** permission to the instance from which you are connecting to the accelerator.
- Check CloudWatch Logs to verify that your accelerator is healthy. The Amazon EC2 instance details from the Amazon EC2 console contain a link to CloudWatch, which allows you to view the health of its associated accelerator.

Stop and Start the Instance

If your Elastic Inference accelerator is in an unhealthy state, stopping and starting it again is the simplest option. For more information, see Stopping and Starting Your Instances.

Warning
When you stop an instance, the data on any instance store volumes is erased. If you have any data to preserve on instance store volumes, make sure to back it up to persistent storage.

Troubleshooting Model Performance

Elastic Inference accelerates operations defined by frameworks like TensorFlow and MXNet. While Elastic Inference accelerates most neural network, math, array manipulation, and control flow operators, there are many operators that Elastic Inference does not accelerate. These include training-related operators, input/output operators, and some operators in contrib.

When a model contains operators that Elastic Inference does not accelerate, the framework runs them on the instance. The frequency and location of these operators within a model graph can have an impact on the model's inference performance with Elastic Inference accelerators. If your model is known to benefit from GPU acceleration and does not perform well on Elastic Inference, contact AWS Support or amazon-ei-feedback@amazon.com.

Submitting Feedback

Contact AWS Support or send feedback to: amazon-ei-feedback@amazon.com.

Document History for Developer Guide

The following table describes the documentation for this release of Amazon Elastic Inference.

- **API version: latest**
- **Latest documentation update:** April 16, 2019

update-history-change	update-history-description	update-history-date
Elastic Inference (p. 1)	Elastic inference prerequisites and related info were added to the setup guide.	April 16, 2019

AWS Glossary

For the latest AWS terminology, see the AWS Glossary in the *AWS General Reference*.