Distributed Deep Learning Inference

using Apache MXNet* and Apache Spark

Naveen Swamy

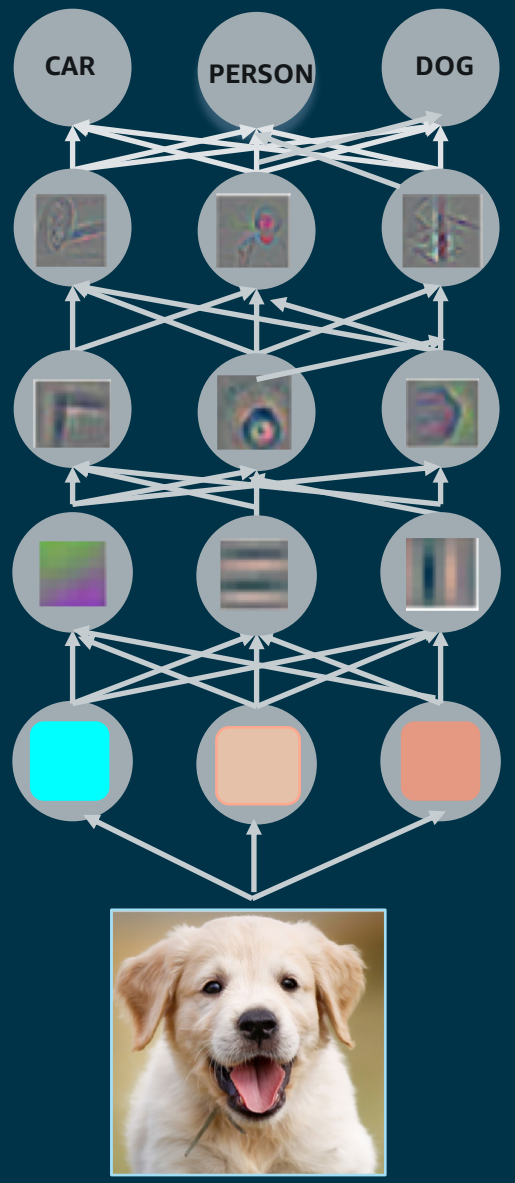
Senior Software Engineer
Amazon AI

Outline

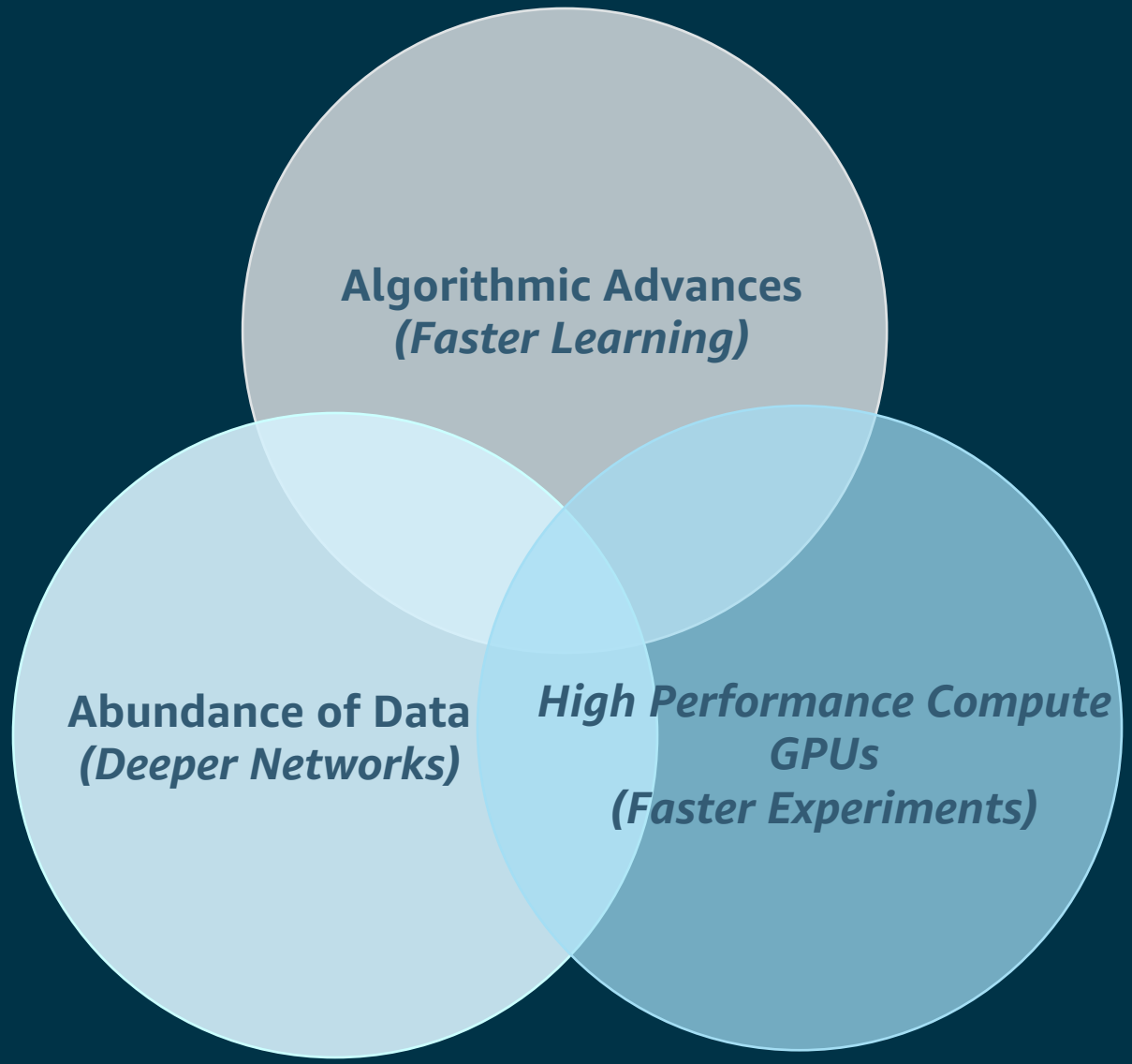
- Review of Deep Learning
- Apache MXNet Framework
- Distributed Inference using MXNet and Spark

Deep Learning

- Originally inspired by our biological neural systems.
- A System that learns important features from experience.
- Layers of Neurons learning concepts.
- Deep learning != deep understanding



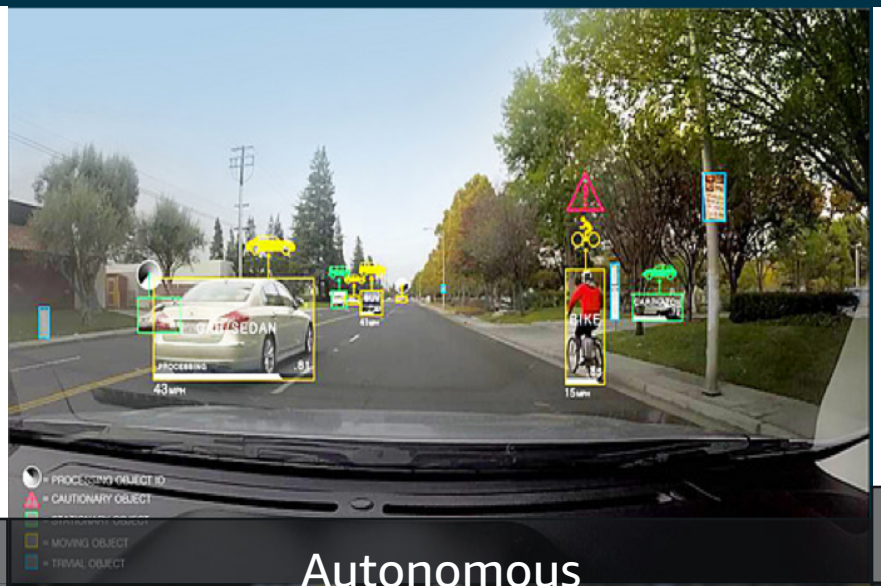
Credit: Ian Goodfellow etal., Deep Learning Book



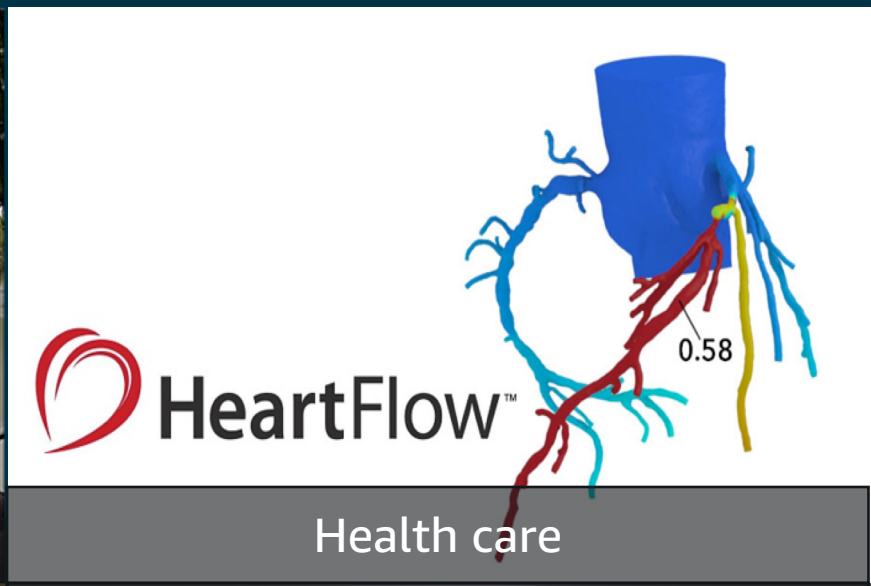
Bigger and Better Models = Better AI Products

Why does Deep Learning matter?

Personal Assistants

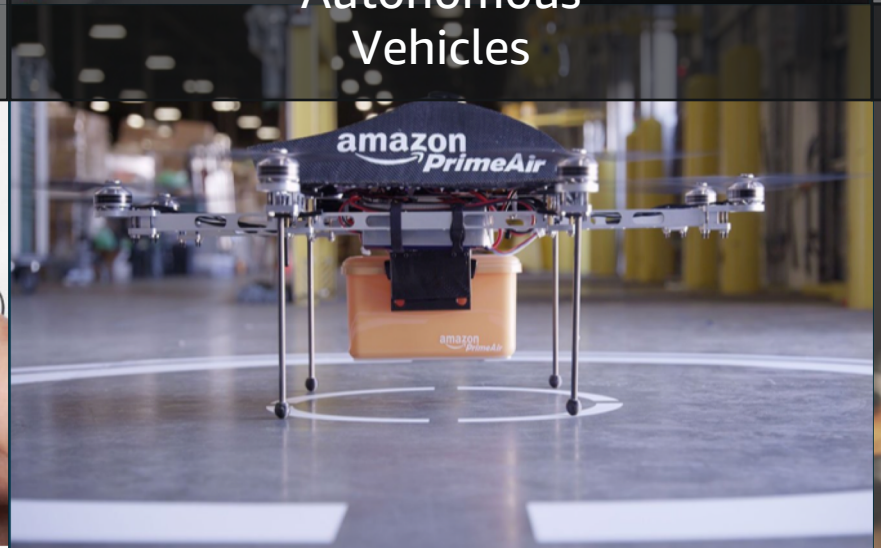


Autonomous Vehicles

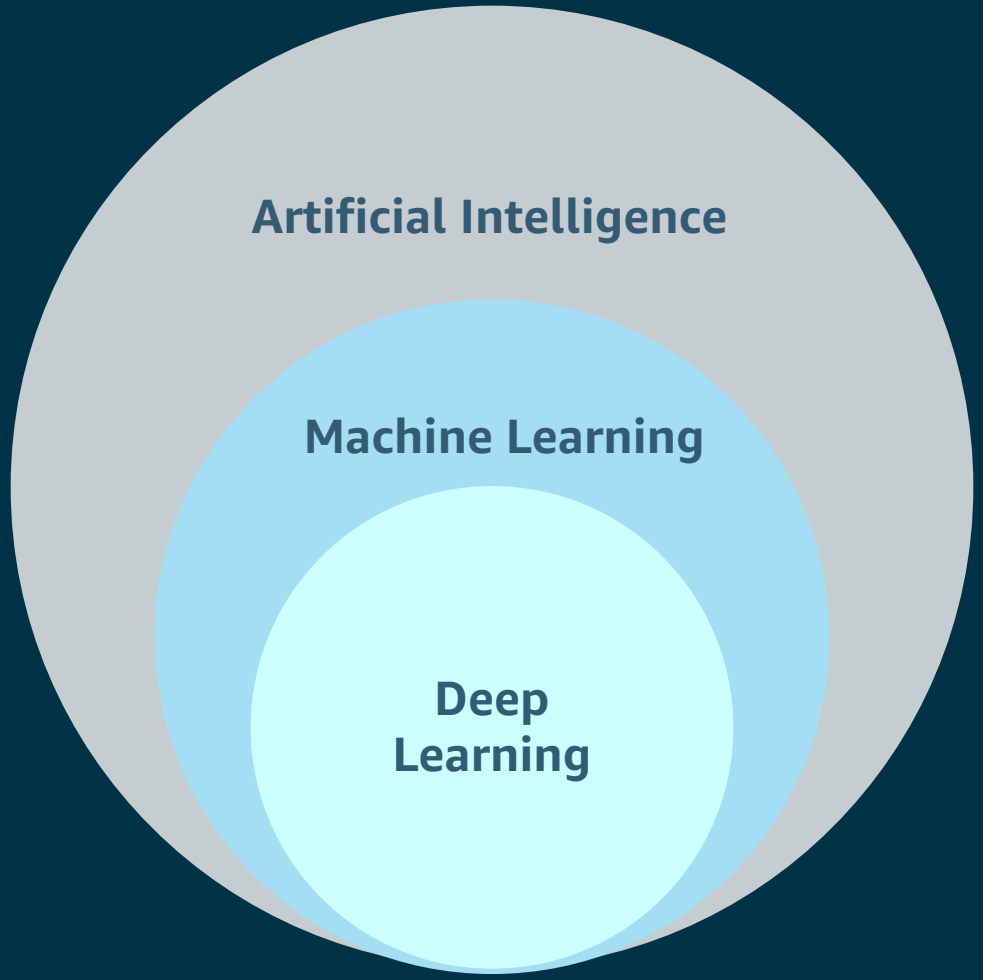


Health care

Solve Intelligence ???



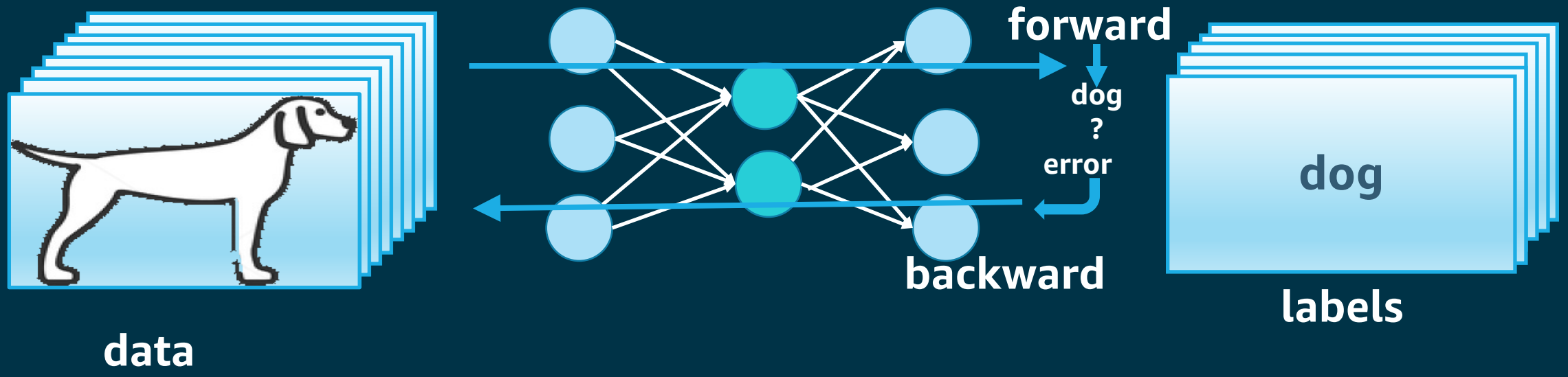
Deep Learning & AI, Limitations



DL Limitations:

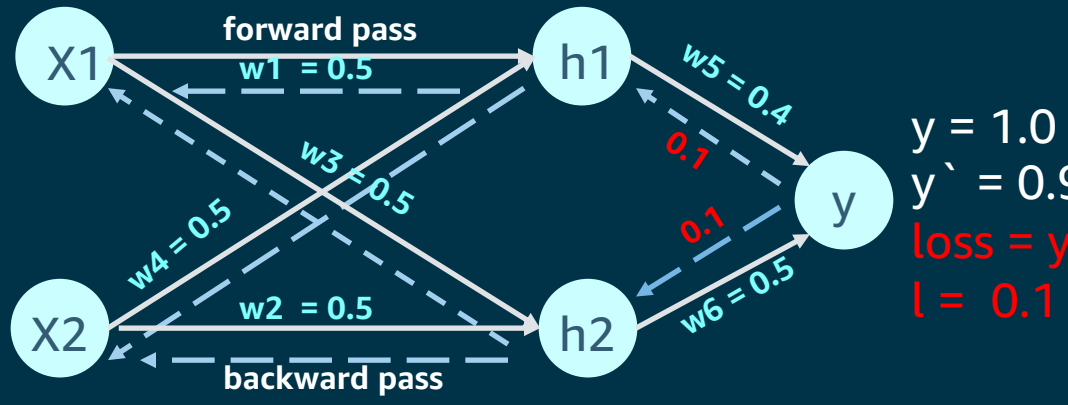
- Requires lots of data and compute power.
- Cannot detect Inherent bias in data - Transparency.
- Uninterpretable Results.

Deep Learning Training

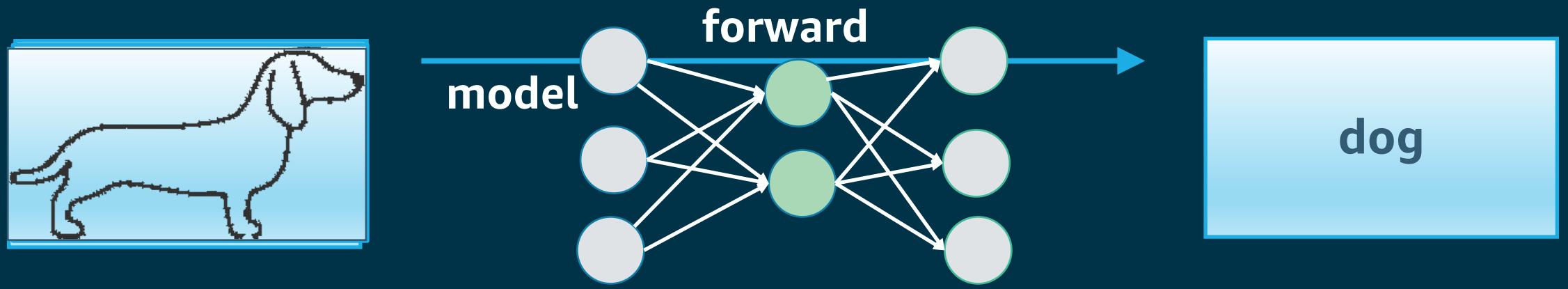


- Pass data through the network – forward pass
- Define an objective – Loss function
- Send the error back – backward pass

Model: Output of Training a neural network



Deep Learning Inference



- **Real time Inference:** Tasks that require immediate result.

- **Batch Inference:** Tasks where you need to run on a large data sets.
  - Pre-computations are necessary - Recommender Systems.
  - Backfilling with state-of-the art models.
  - Testing new models on historic data.

Types of Learning

- **Supervised Learning** – Uses labeled training data learning to associate input data to output.

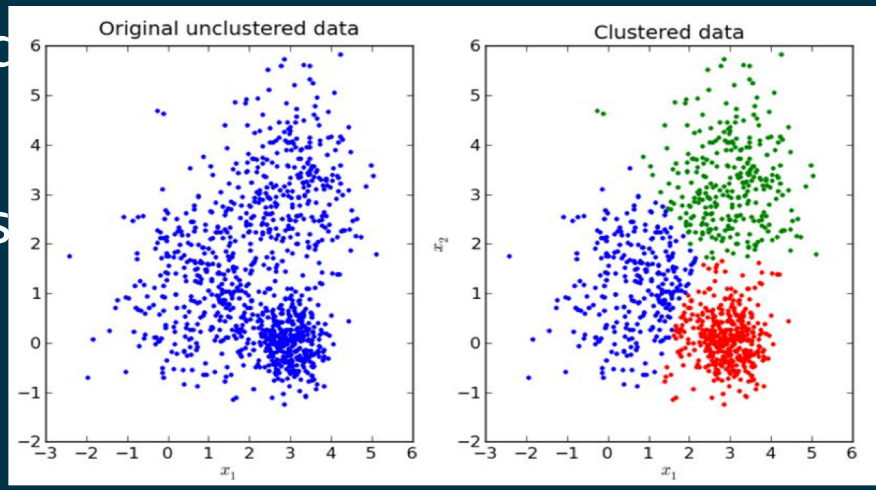
Example: Image classification, Speech Recognition, Machine translation

- **Unsupervised Learning -** Learns patterns from Unlabeled data.

Example: Clustering, Association discovery.

- **Active Learning** – Semi-supervised, human in the midd

- **Reinforcement Learning** – learn from environment, us feedback.



Outline

- Review of Deep Learning
- Apache MXNet Framework
- Distributed Inference using MXNet and Spark

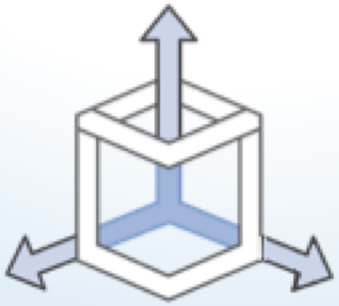
Apache MXNet - Background

- Apache (incubating) open source project
- Framework for building and training DNNs
- Created by academia (CMU and UW)
- Adopted by AWS as DNN framework of choice, Nov 2016

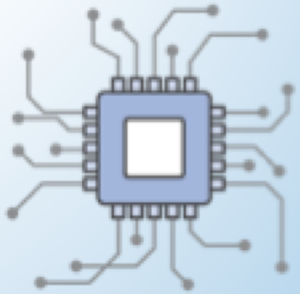
http://mxnet.apache.org

Why MXNet

Programmable
Simple Syntax
Imperative/Declarative
Multiple languages



Portable
Highly efficient models
for Mobile and IOT



High Performance
Near linear scaling across
hundreds of GPUs

Open Source
Incubating at Apache

ONNX Support

GLUON

Easily and quickly build high performance models with **Imperative APIs**

mxnet

chik-fil-A waffle-fry freshness with Apache MXNet Computer Vision

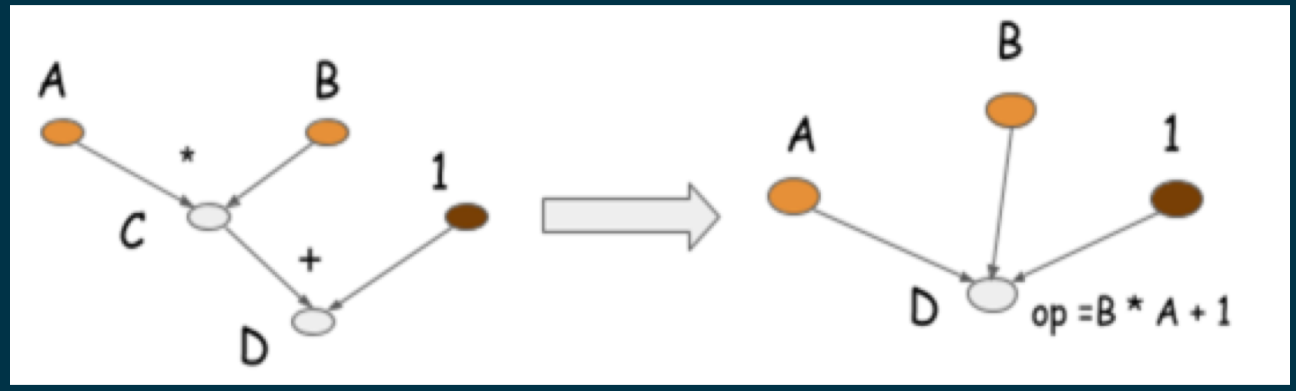
Apache MXNet Customer Momentum

MXNet – NDArray & Symbol

- **NDArray**– Imperative Tensor Operations that work on both CPU and GPUs.
- **Symbol APIs** – similar to NDArray but adopts declarative programming for optimization.

```
A = Variable('A')
B = Variable('B')
C = B * A
D = C + Constant(1)
# compiles the function
f = compile(D)
d = f(A=np.ones(10), B=np.ones(10)*2)
```

Symbolic Program



Computation Graph

MXNet - Module

High level APIs to work with Symbol

1) Create Graph

```
>>> data = mx.sym.Variable('data')
>>> fc1  = mx.sym.FullyConnected(data, name='fc1', num_hidden=128)
>>> act1 = mx.sym.Activation(fc1, name='relu1', act_type="relu")
>>> fc2  = mx.sym.FullyConnected(act1, name='fc2', num_hidden=10)
>>> out  = mx.sym.SoftmaxOutput(fc2, name = 'softmax')
>>> mod = mx.mod.Module(out)  # create a module by given a Symbol
```

2) Bind

```
>>> mod.bind(data_shapes=nd_iter.provide_data,
>>>          label_shapes=nd_iter.provide_label) # create memory by given input shapes
>>> mod.init_params()  # initial parameters with the default random initializer
```

3) Pass data

```
>>> mod.fit(nd_iter, num_epoch=10, ...)  # train
>>> mod.predict(new_nd_iter)  # predict on new data
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 2

In [ ]:
```
import mxnet as mx
import numpy as np
from collections import namedtuple
```

In [ ]:
```
symbol_fname = "/Users/wamy/nswamy/deepengine/workspace/emr-integration/resnet-18/resnet-18-symb
params_fname = "/Users/wamy/nswamy/deepengine/workspace/emr-integration/resnet-18/resnet-18-000(
synset_fname = "/Users/wamy/nswamy/deepengine/workspace/emr-integration/resnet-18/synset.txt"
```

In [ ]:
```
def load_model(s_fname, p_fname):
    sym = mx.symbol.load(s_fname)
    save_dict = mx.nd.load(p_fname)
    arg_params = {}
    aux_params = {}
    for k, v in save_dict.items():
        tp, name = k.split(':', 1)
        if tp == 'arg':
            arg_params[name] = v
        if tp == 'aux':
            aux_params[name] = v
    return sym, arg_params, aux_params
```

In [ ]:
```
symbol,arg_params, aux_params = load_model(symbol_fname, params_fname)
```

In [ ]:
```
mx.viz.plot_network(symbol)
```

In [ ]:
```
arg_params
```

In [ ]:
```
synset_fname
```

In [ ]:
```
with open(synset_fname, 'r') as f:
```

MXNet Gluon – Imperative & Fast

Simple, Easy-to-Understand Code

- Neural networks can be defined using simple, clear, concise code
- Plug-and-play neural network building blocks – including predefined layers, optimizers, and initializers

Flexible, Imperative Structure

- Eliminates rigidity of neural network model definition and brings together the model with the training algorithm
- Intuitive, easy-to-debug, familiar code

Dynamic Graphs

- Neural networks can change in shape or size during the training process to address advanced use cases where the size of data fed is variable
- Important area of innovation in Natural Language Processing (NLP)

High Performance

- There is no sacrifice with respect to training speed
- When it is time to move from prototyping to production, easily cache neural networks for high performance and a reduced memory footprint

Credits: Cyrus Vahid

Outline

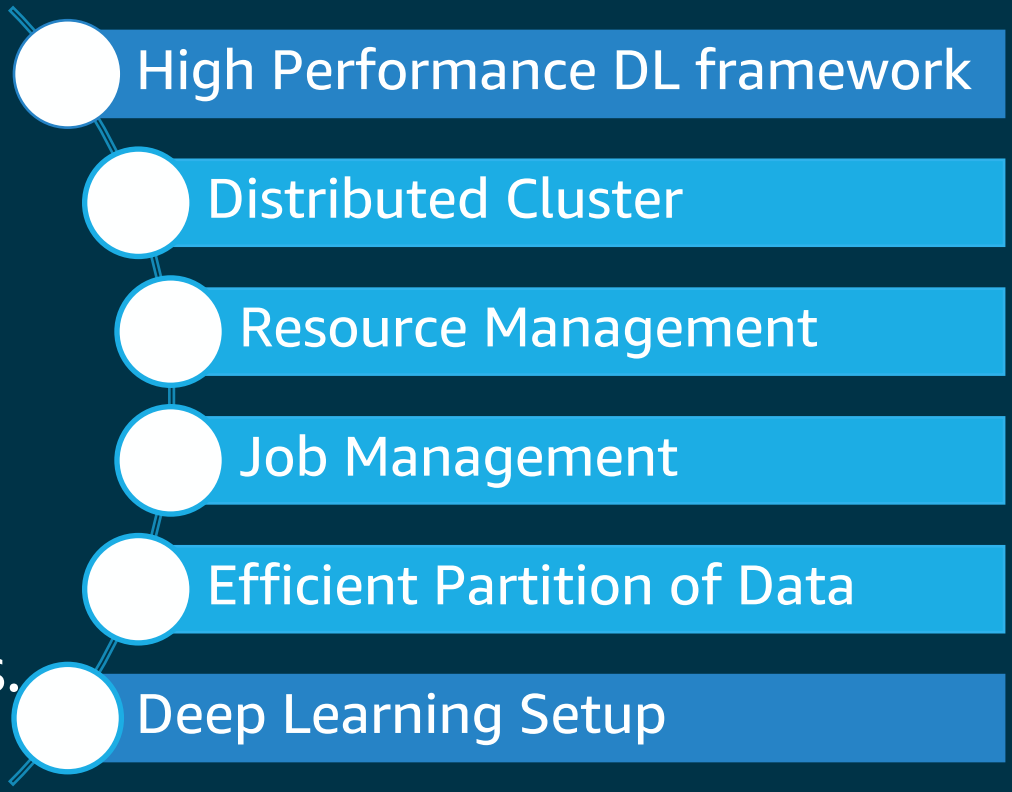
- Review of Deep Learning
- Apache MXNet Framework
- Distributed Inference using MXNet and Spark

Distributed Inference Challenges

- Similar to large scale data processing systems

Apache Spark:

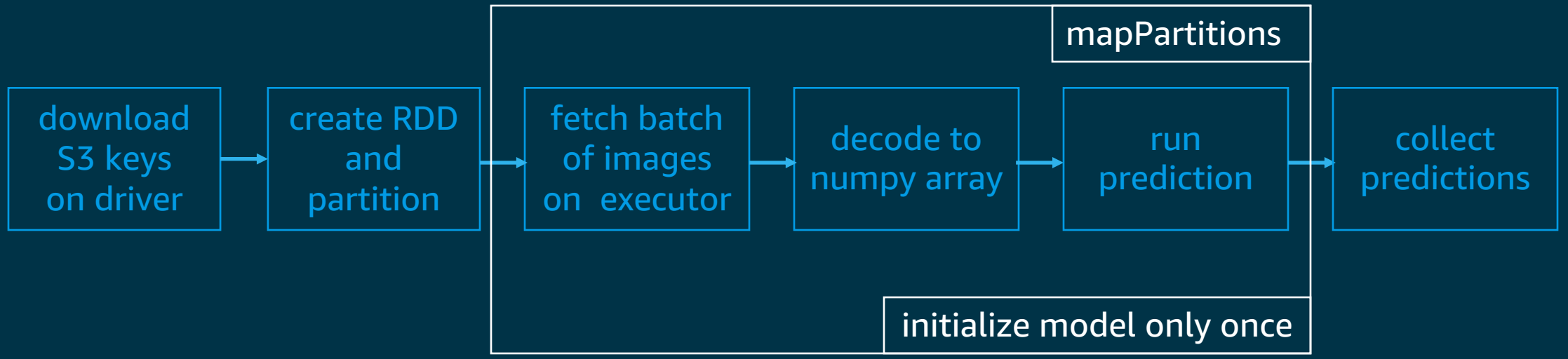
- Multiple Cluster Managers
- Works well with MXNet.
- Integrates with Hadoop & big data tools.



MXNet + Spark for Inference.

- ImageNet trained ResNet-18 classifier.
- For demo, CIFAR-10 test dataset with 10K Images.
- PySpark on Amazon EMR, MXNet is also available in Scala.
- Inference on CPUs, can be extended to use GPUs.

Distributed Inference Pipeline



MXNet + Spark for Inference.

On the driver

```
conf = SparkConf().setAppName("Distributed Inference using MXNet and Spark")
conf.set('spark.executor.cores', '1')
  n_partitions = len(keys) // args['batch']

rdd = sc.parallelize(keys, num_slices=n_partitions)
sc.broadcast(args['bucket'])
rdd = rdd.mapPartitions(lambda k : download_objects(args['bucket'], k))
rdd = rdd.mapPartitions(load_images)
sc.broadcast(args)
rdd = rdd.mapPartitions(lambda imgs: predict(imgs, args))

output = rdd.collect()
```

On the executor

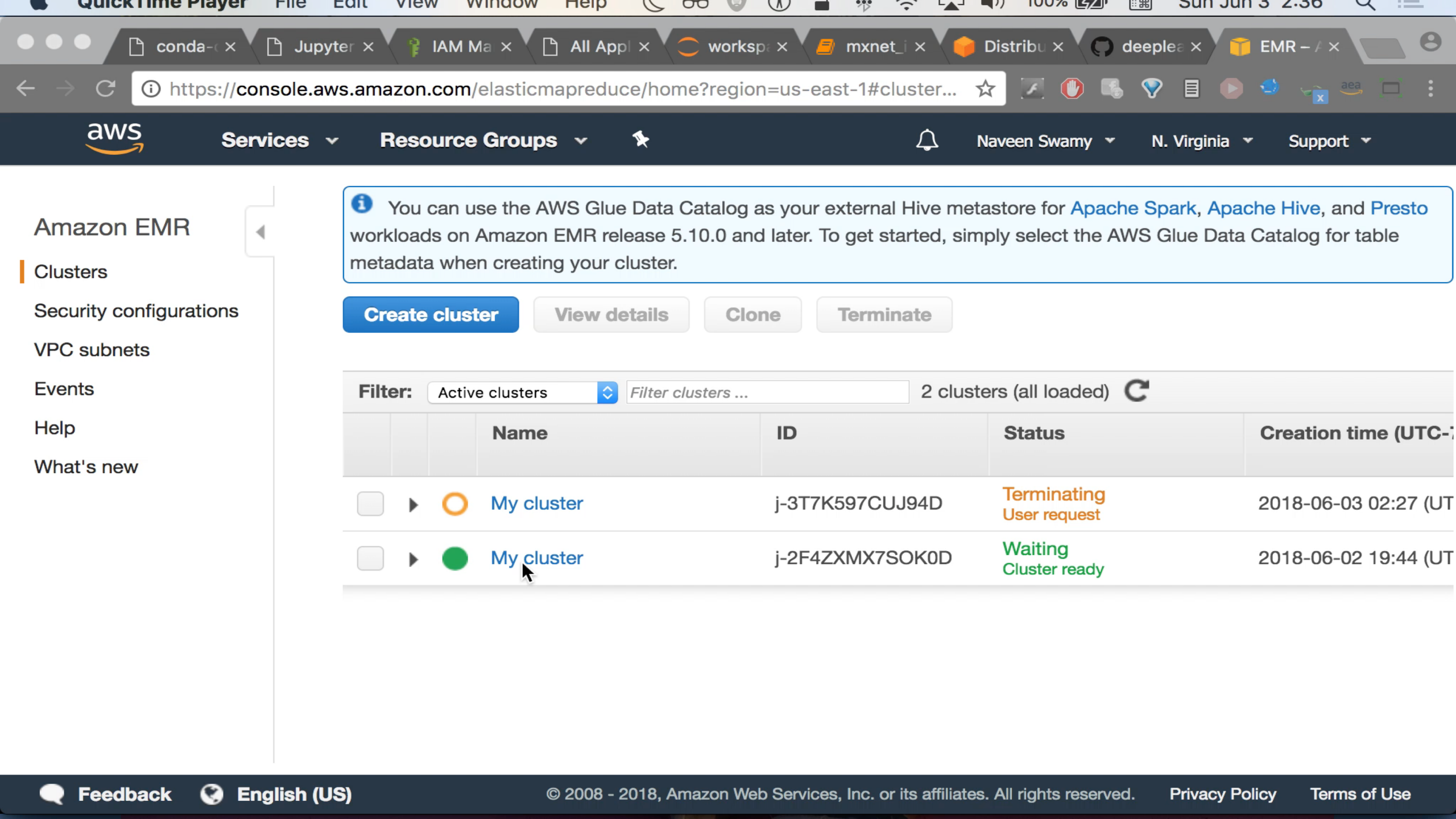
```
class MXModel(object):
    """
    This is a singleton class that just holds the loaded mxnet model in the module object
    We don't want to load the model for every inference when called from the map method
    """
    __metaclass__ = Singleton
    model_loaded = False
    mod = None
    synsets = None


    def __init__(self, sym_url, param_url, synset_url, batch_size):
        (s_fname, p_fname, synset_fname) = self.download_model_files(sym_url, param_url, synset_url)
        MXModel.synsets = self.load_synset(synset_fname)
        MXModel.mod = self.init_module(s_fname, p_fname, batch_size)
        MXModel.model_loaded = True
```

```
def predict(img_batch, args):
    """
    Run predication on batch of images in 4-D numpy array format and return the top_5 probability along with their classes
    """
    import mxnet as mx
    import numpy as np
    logger.info('predict-args:%s' %(args))

    if not MXModel.model_loaded:
        MXModel(args['sym_url'], args['param_url'], args['label_url'], args['batch'])

    MXModel.mod.forward(Batch([mx.nd.array(img_batch)]))
```


QuickTime Player File Edit View Window Help
Sun Jun 3 2:36
conda- Jupyter IAM Ma All Appl worksp mxnet_i Distribu deeplea EMR –
https://console.aws.amazon.com/elasticmapreduce/home?region=us-east-1#cluster...
aws
Services
Resource Groups
Naveen Swamy
N. Virginia
Support
Amazon EMR
Clusters
Security configurations
VPC subnets
Events
Help
What's new
You can use the AWS Glue Data Catalog as your external Hive metastore for Apache Spark, Apache Hive, and Presto workloads on Amazon EMR release 5.10.0 and later. To get started, simply select the AWS Glue Data Catalog for table metadata when creating your cluster.
Create cluster
View details
Clone
Terminate
Filter:
Active clusters
Filter clusters ...
2 clusters (all loaded)
Name
ID
Status
Creation time (UTC-
My cluster
j-3T7K597CUJ94D
Terminating
User request
2018-06-03 02:27 (UT
My cluster
j-2F4ZXMX7SOK0D
Waiting
Cluster ready
2018-06-02 19:44 (UT
Feedback
English (US)
© 2008 - 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

Summary

- Overview of Deep Learning
  - How Deep Learning works and Why Deep Learning is a big deal.
  - Phases of Deep Learning
  - Types of Learning
- Apache MXNet – Efficient deep learning library
  - NDArray/Symbol/Module
- Apache MXNet and Spark for distributed Inference.

What's Next ?

- Released v1.3.0 with many new features and improvements –

  Addition of Clojure support, User friendly APIs in Scala, Improved RNN support in Gluon, …

- Working on Java APIs for Inference.

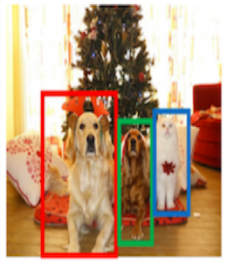
- MXNet community is fast evolving, join hands to democratize AI.

Resources/References

- https://github.com/apache/incubator-mxnet
- Blog- Distributed Inference using MXNet and Spark
- Distributed Inference code sample on GitHub
- Apache MXNet Gluon Tutorials
- Apache MXNet – Flexible and efficient deep learning.
- The Deep Learning Book
- MXNet – Using pre-trained models
- Amazon Elastic MapReduce

Object Detection Using Convolutional Neural Networks

So far, when we've talked about making predictions based on images, we were concerned only with classification. We asked questions like is this digit a "0", "1", ..., or "9?" or, does this picture depict a "cat" or a "dog"? Object detection is a more challenging task. Here our goal is not only to say *what* is in the image but also to recognize *where* it is in the image. As an example, consider the following image, which depicts two dogs and a cat together with their locations.



So object defers from image classification in a few ways. First, while a classifier outputs a single category per image, an object detector must be able to recognize multiple objects in a single image. Technically, this task is called *multiple object detection*, but most research in the area addresses the multiple object setting, so we'll abuse terminology just a little. Second, while classifiers need only to output probabilities over classes, object detectors must output both probabilities of class membership and also the coordinates that identify the location of the objects.

On this chapter we'll demonstrate the single shot multiple box object detector (SSD), a popular model for object detection that was first described in this paper, and is straightforward to implement in MXNet Gluon.

SSD: Single Shot MultiBox Detector

Apache MXNet (Incubating)

A flexible and efficient library for deep learning.

Install Learn More

A 60-minute Gluon Crash Course

Check out our quick overview of how to use Gluon, the imperative interface of MXNet.

Learn More

MXNet 1.2.0 Released

We're excited to announce the release of MXNet 1.2.0! Check out the release notes for latest updates.

Thank You

wamy@amazon.com