Software

BIGDL: DISTRIBUTED DEEP LEARNING ON APACHE SPARK

Yiheng Wang (yiheng.wang@intel.com)

Big Data Technologies, Software and Service Group, Intel

Introduction

Intel Big Data Technology team

- Active open source development
- Spark, Hadoop, HBase, Hive, Sentry, Storm, etc
- ~30 project committers in the team

My focusing area

- Large scale machine learning, deep learning
- Next generations of Big Data analytics solutions with Intel customers

BigDL

A distributed deep learning framework on Apache Spark

http://www.github.com/intel-analytics/BigDL

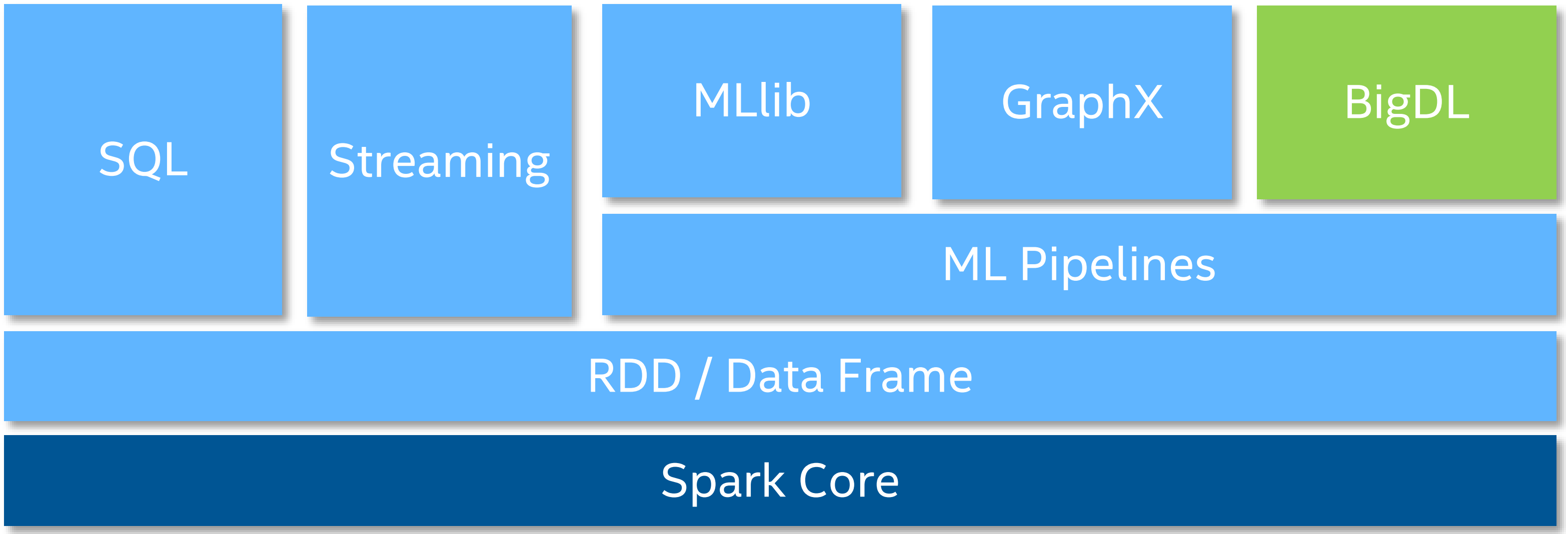
Outline

- Overview
- Install and Run BigDL
- Define models
- Train and evaluate models
- Model tuning

OVERVIEW

An overview of BigDL

Build on Apache Spark



There're a lot of deep learning solutions

neon

mxnet

Caffe

theano

CNTK

BigDL

A scalable and easy solution for deep learning on Big Data

Build an End-2-end Solution

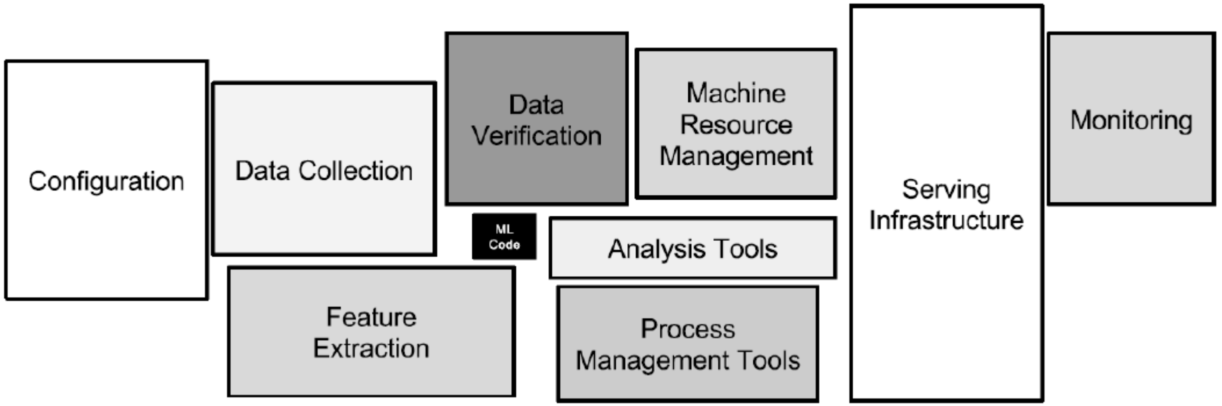


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

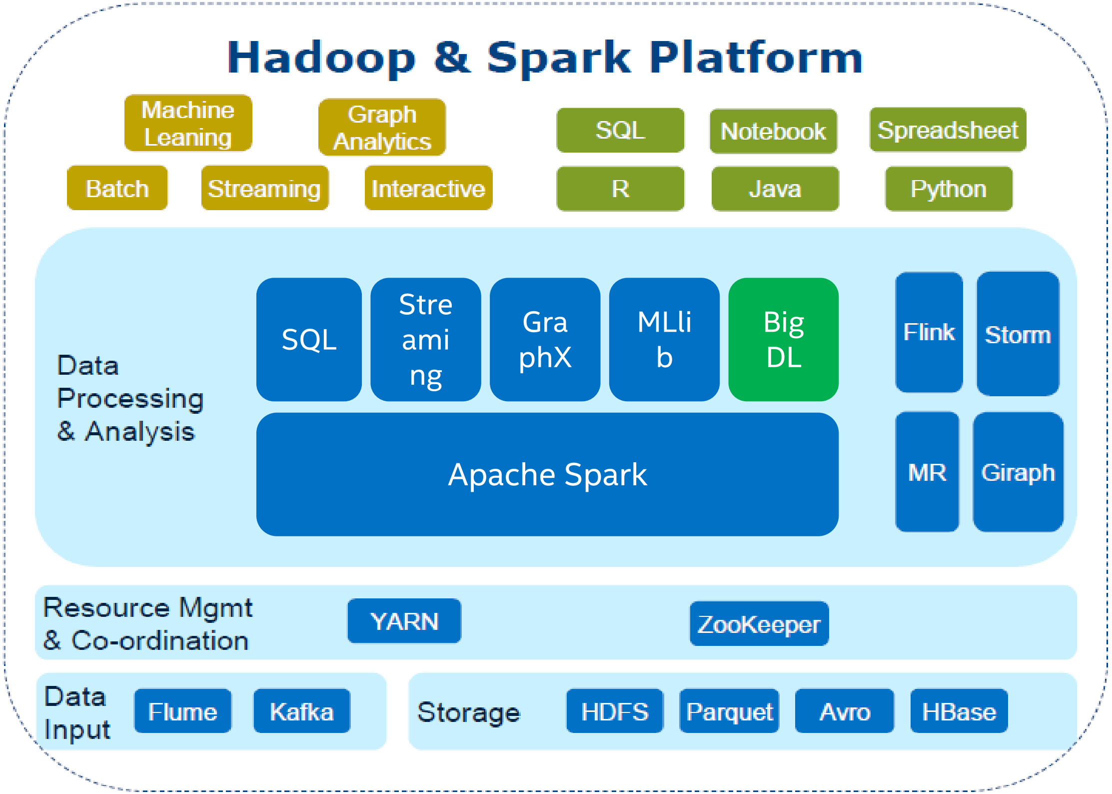
"Hidden Technical Debt in Machine Learning Systems", Google, NIPS 2015 Paper

Build an End-2-end Solution

Practical challenges:

- compatible with different data source
- performance and scalability
- stability & fault tolerant
- data management / pre-processing
- resource sharing
- programming tools / languages
- …

Build an End-2-end Scalable Solution

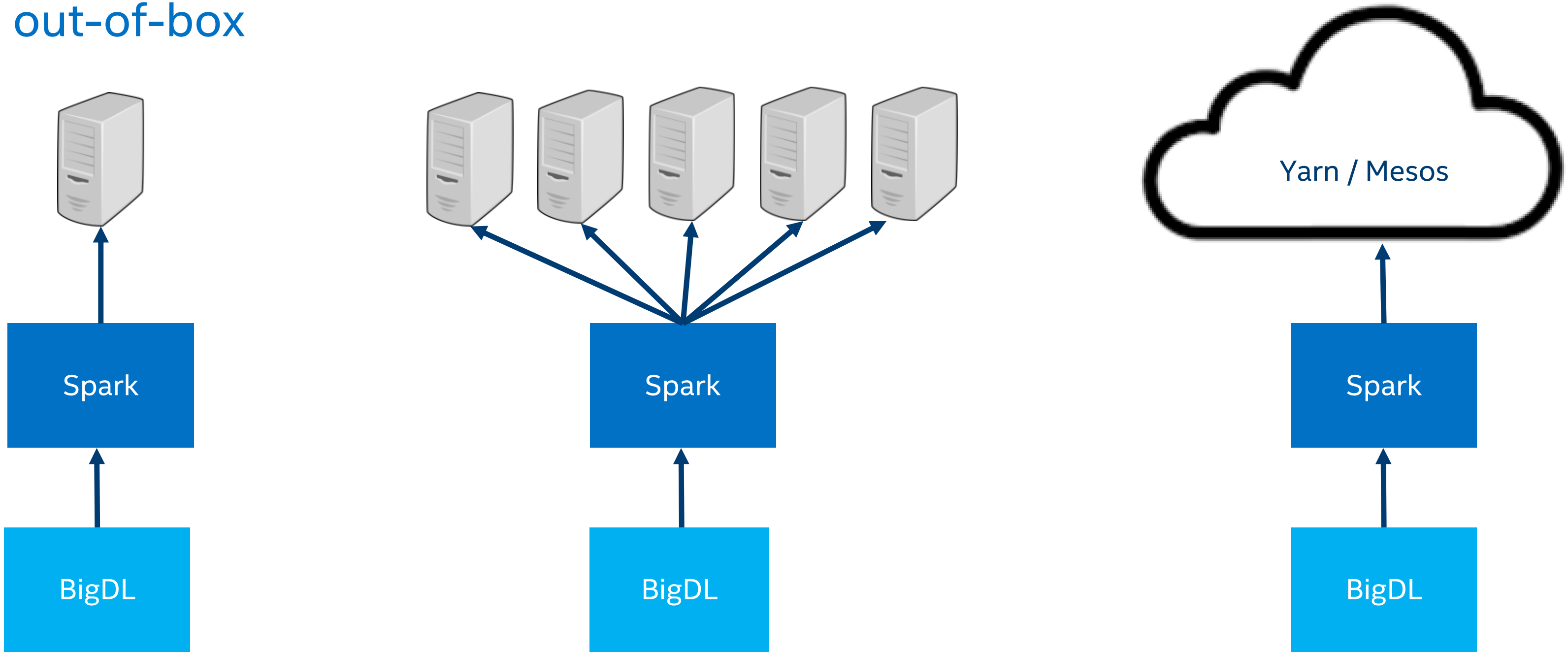


BigDL is easy to use

- A friendly API compatible with Torch
- Provide Scala and Python programming API

BigDL is easy to deploy

Real out-of-box



BigDL is easy to deploy

Public cloud blogs (See https://github.com/intel-analytics/BigDL/wiki/powered-by)

- Intel's BigDL on Databricks
- Use BigDL on AZure HDInsight
- BigDL on AliCloud E-MapReduce (in Chinese)
- Running BigDL, Deep Learning for Apache Spark, on AWS
- Running BigDL on Microsoft Data Science Virtual Machine
- Using Apache Spark with Intel BigDL on Mesosphere DC/OS by Lightbend

People use BigDL to build applications

Rich deep learning feature support

Layers

- More than 100 (Linear, Conv2D, Conv3D, Embedding, Recurrent)

Loss function

- Dozens of loss functions

Optimization algorithm

- SGD, Adagrad, Adam, Adamax, RMSProp, Adadelta

Distributed Training / Inference

Save and Load model files

- Also include torch / caffe / tensorflow

High performance from your server

- Powered by Intel Math Kernel Library
- Extremely high performance on Xeon CPUs
  - Order of magnitude faster than out of box caffe / torch / tensorflow
  - Comparable with GPU (same generation)
- Good scalability
  - Hundreds of nodes

INSTALL AND RUN BIGDL

How to install and run bigdl on your cluster

Get executable BigDL

- Download
- Maven / Sbt
- Pip install
- Build yourself

Download

- Download Page (https://github.com/intel-analytics/BigDL/wiki/Downloads)
  - Linux x64 and Mac OS
  - Windows(WIP)
- Stable release and nightly build
- Python development / Run examples

Maven / SBT

- https://github.com/intel-analytics/BigDL/wiki/Build-Page#linking
- Snapshot, release
- Java/Scala development

```
<dependencies>
  <dependency>
    <group>com.intel.analytics.bigdl</group>
    <artifactId>bigdl-SPARK_(1.5/1.6/2.0/2.1)</artifactId>
    <version>0.1.1</version>
  </dependency>
</dependencies>
```

Pip Install

See https://github.com/intel-analytics/BigDL/wiki/Install-BigDL-via-pip

- Download Spark2.x

```
wget https://d3kbcqa49mib13.cloudfront.net/spark-2.1.0-bin-hadoop2.7.tgz
```

- Extract the tar ball and set SPARK_HOME

```
tar -zxvf spark-2.1.0-bin-hadoop2.7.tgz
export SPARK_HOME=path to spark-2.1.0-bin-hadoop2.7
```

- Install BigDL 0.1.1 release via pip (we tested this on pip 9.0.1)

```
pip install --upgrade pip
pip install BigDL==0.1.1rc0    # for Python 2.7
pip3 install BigDL==0.1.1rc0  # for Python 3.n
```

- Launch with Python REPL / Jupyter

Build yourself

- Customized configuration, e.g. JDK 8, Spark version
- Develop BigDL
- No need to pre-install MKL (MKL jar will be downloaded)

```
$ git clone https://github.com/intel-analytics/BigDL.git
$ cd BigDL

$ ./make-dist.sh  # For Spark 1.5/1.6, Linux x64

$ ./make-dist.sh -P mac   # For Spark 1.5/1.6, MacOS

$ ./make-dist.sh -P spark_2.x  # For Spark 2.0/2.1, Linux x64

$ ./make-dist.sh -P mac -P spark_2.x  # For Spark 2.0/2.1, MacOS
```

Start your BigDL program

Run scala code

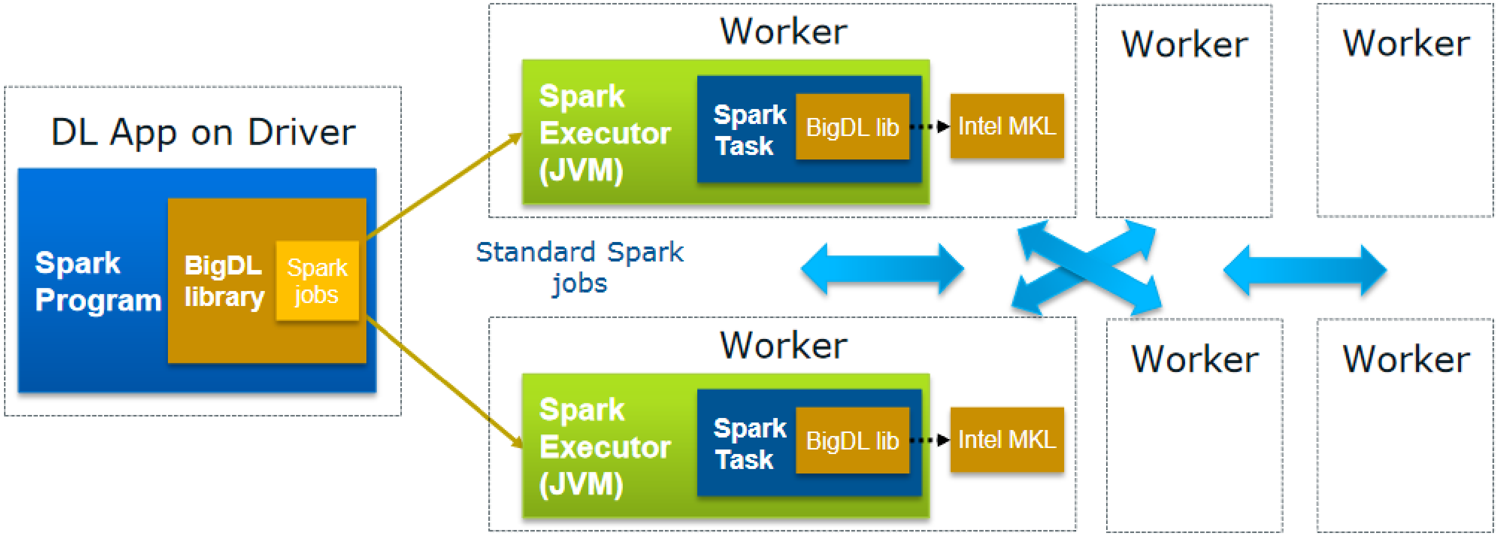
```
spark-submit \
  --master xxx
  --jars path_to_big_dl_jar
  --class main_class_full_name
  --……
  your_project_jar
  ……
```

Run python code

```
spark-submit \
  --master xxx
  --jars path_to_big_dl_jar
  --py-files path_to_big_dl_python_zip
  your_python_file
  ……
```

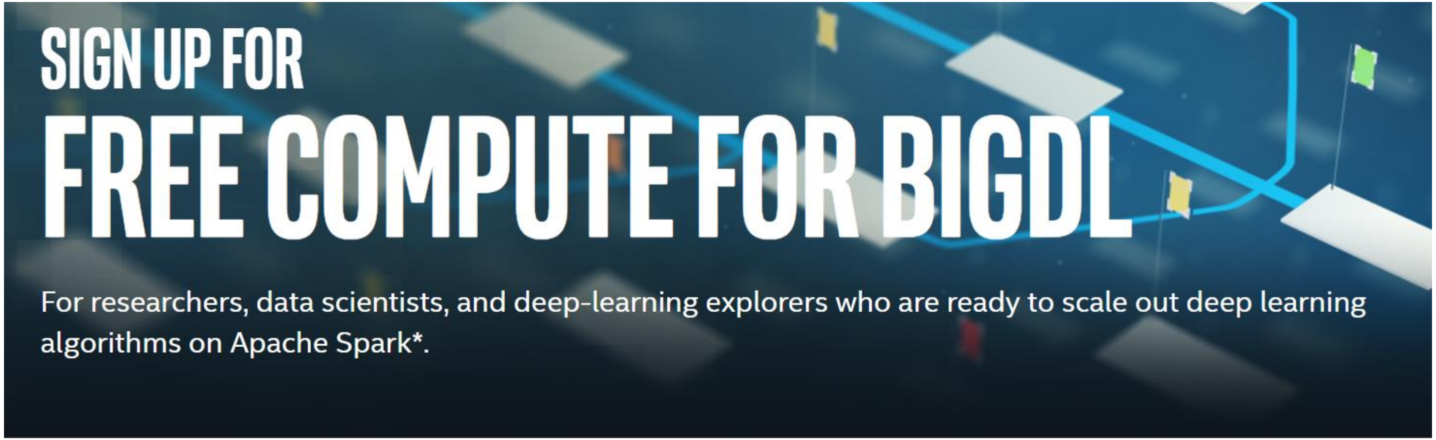
In BigDL 0.1.0 and 0.1.1, you need to run **source bigdl.sh** before you run the spark-submit command

How BigDL run on Apache Spark*



Sign up for free compute for BigDL

https://software.intel.com/en-us/ai/frameworks/bigdl/remote-access

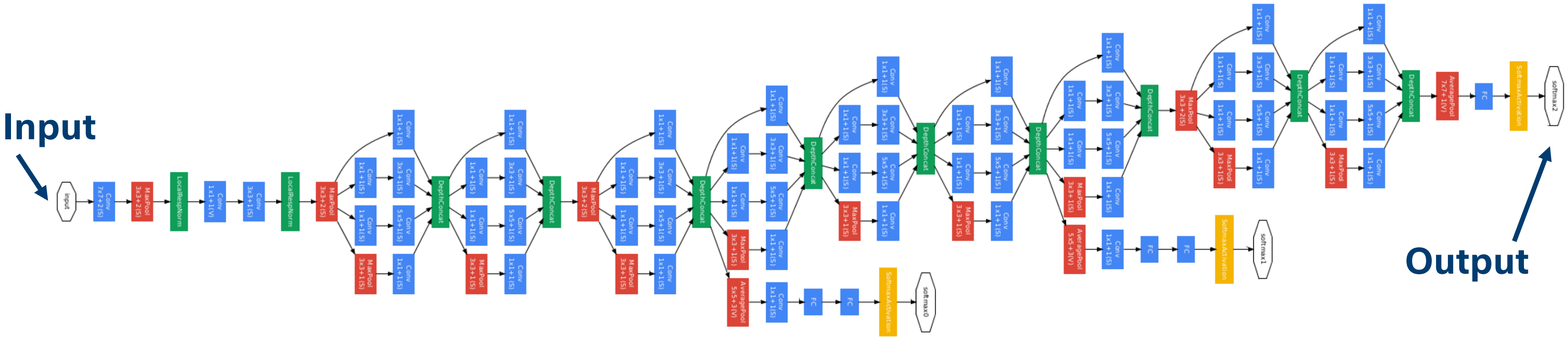


Preregister for Free Compute for BigDL, sponsored by Intel, and provide feedback to help make BigDL better for new users. You don't need to share your code. Preference goes to those who share their BigDL story.

DEFINE MODELS

How to define model in BigDL

What is a model



BigDL provides two kind of model definition style

- Sequential API
  - In sequential API, user add layer into some containers to build the model
- Functional API
  - In functional API, the model is described as a graph

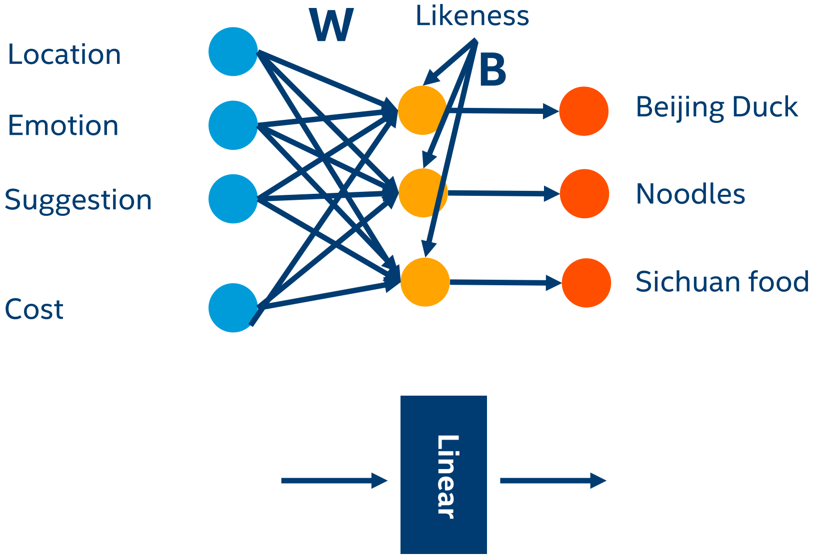
Define a model (Linear)

Scala

```
val model = Sequential()

model.add(Linear(4, 3))
```

Python

```
model = Sequential()

model.add(Linear(4, 3))
```



Simple Linear classification

$$Y = X * W + B$$

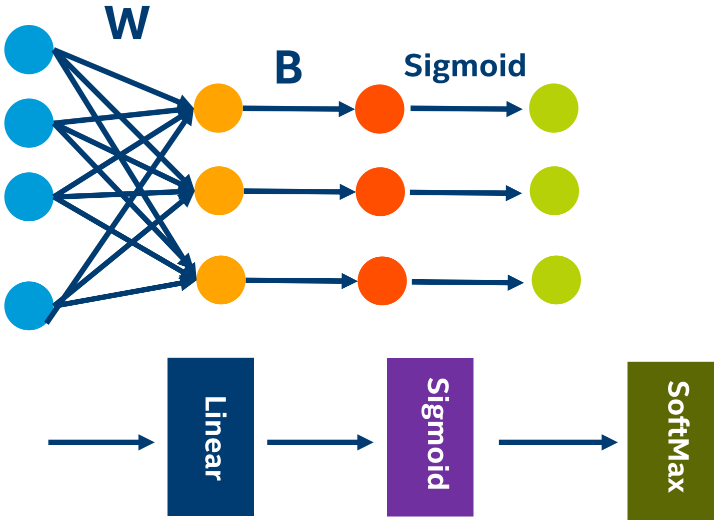
Add activation functions

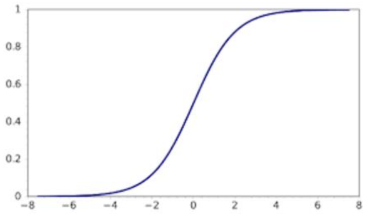
Scala

```
val model = Sequential()

model.add(Linear(4, 3))

model.add(Sigmoid())
```

Python

```
model = Sequential()

model.add(Linear(4, 3))

model.add(Sigmoid())
```

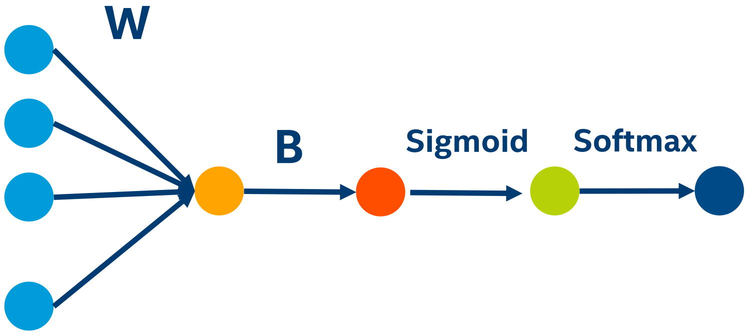




Logistic Regression

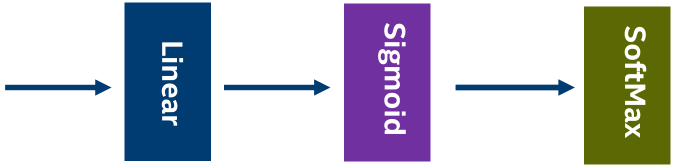
Scala

```
val model = Sequential()
model.add(Linear(4, 1))
model.add(Sigmoid())
Model.add(Softmax())
```



Python

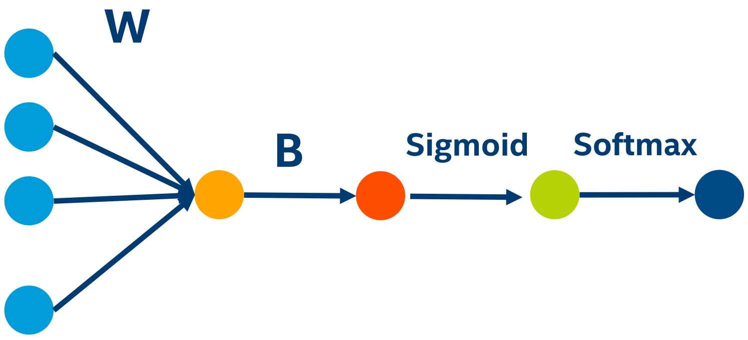
```
model = Sequential()
model.add(Linear(4, 1))
model.add(Sigmoid())
Model.add(Softmax())
```



Another way to define Logistic Regression

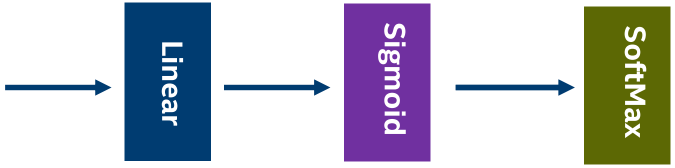
Scala

```
val linear = Linear(4, 1).inputs()

val sigmoid = Sigmoid().inputs(linear)

val softmax = Softmax().inputs(sigmoid)

val model = Graph(Seq[linear], Seq[softmax])
```



Python

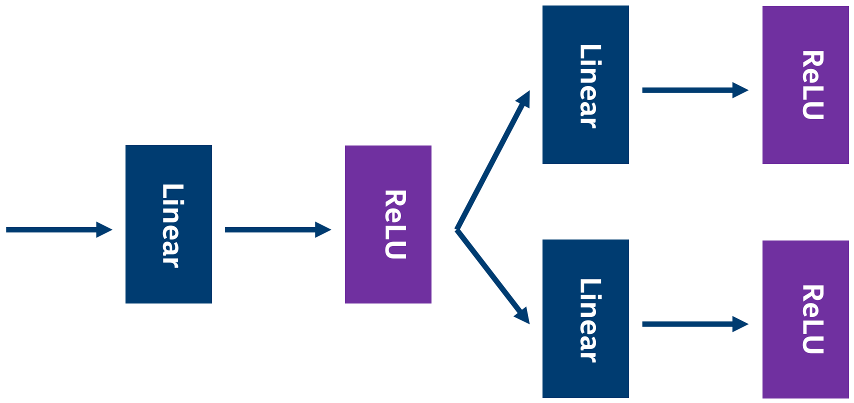
```
linear = Linear(4, 1)()

sigmoid = Sigmoid()(linear)

softmax = Softmax()(sigmoid)

model = Model([linear], [softmax])
```



Define a model with branches

Sequential

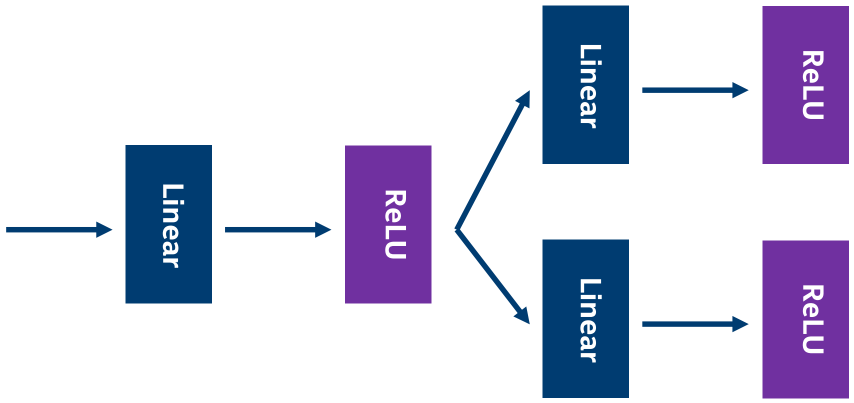
```
branch1 = Sequential().add(Linear(…)).add(ReLU())

branch2 = Sequential().add(Linear(…)).add(ReLU())

branches =
ConcatTable().add(branch1).add(branch2)


val model = Sequential()

model.add(Linear(…))

model.add(ReLU())

model.add(branches)
```



Define a model with branches

Functional

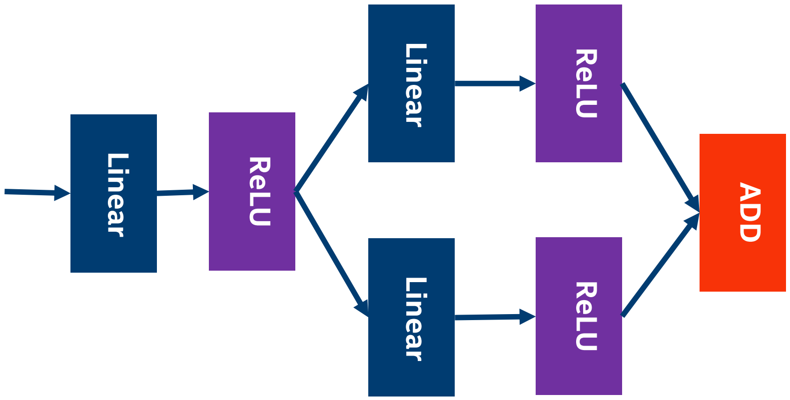
```
linear1 = Linear(...)()

relu1 = ReLU()(linear1)

linear2 = Linear(...)(relu1)

relu2 = ReLU()(linear2)

linear3 = Linear(...)(relu1)

relu3 = ReLU()(linear3)

model = Model(Seq[linear1], Seq[relu2, relu3])
```



Define a model with merged branch

Sequential

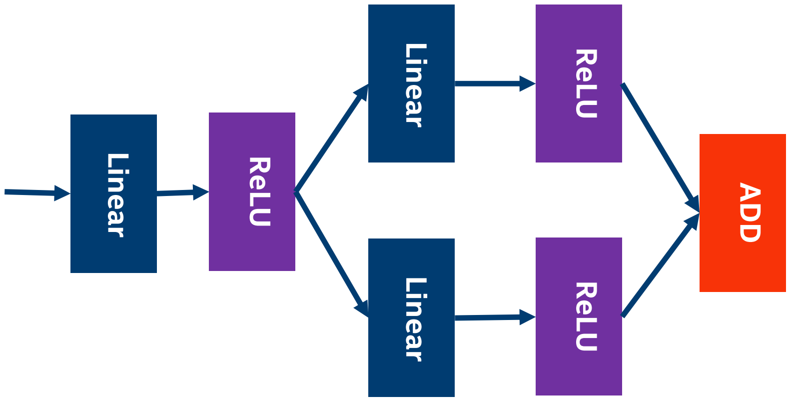
```
branch1 = Sequential().add(Linear(…)).add(ReLU())

branch2 = Sequential().add(Linear(…)).add(ReLU())

branches =
ConcatTable().add(branch1).add(branch2)


val model = Sequential()

model.add(Linear(…))

model.add(ReLU())

model.add(branches)

model.add(CAddTable())
```



Define a model with merged branch

Functional

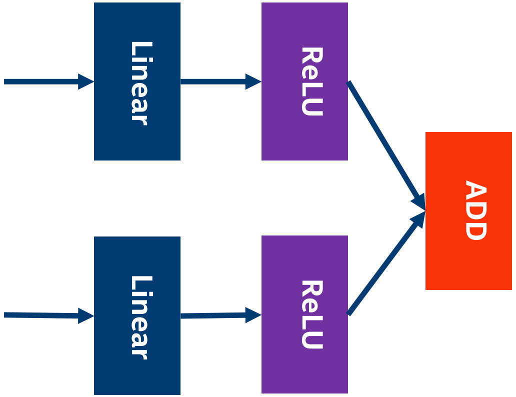
```
linear1 = Linear(...)()

relu1 = ReLU()(linear1)

linear2 = Linear(...)(relu1)

relu2 = ReLU()(linear2)

linear3 = Linear(...)(relu1)

relu3 = ReLU()(linear3)

add = CAddTable()(relu2, relu3)

model = Model(Seq[linear1], Seq[add])
```



Define a model with multiple inputs

Sequential

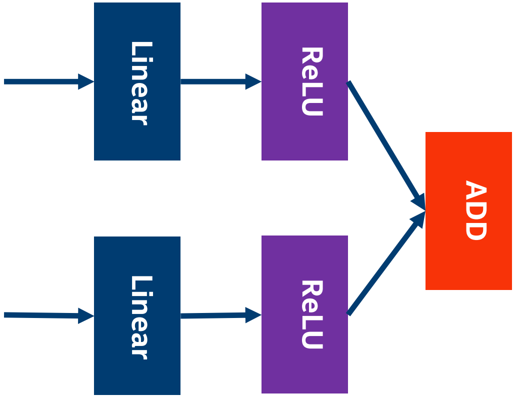
```
model = Sequential()

branches = ParallelTable()

branch1 = Sequential().add(Linear(…)).add(ReLU())

branch2 = Sequential().add(Linear(…)).add(ReLU())

branches.add(branch1).add(branch2)

model.add(branches).add(CAddTable)
```



Define a model with multiple inputs

Functional

```
linear1 = Linear(...)()

relu1 = ReLU()(linear1)

linear2 = Linear(...)()

relu2 = ReLU()(linear2)

add = CAddTable()(relu1, relu2)

model = Model(Seq[linear1, linear2], Seq[add])
```



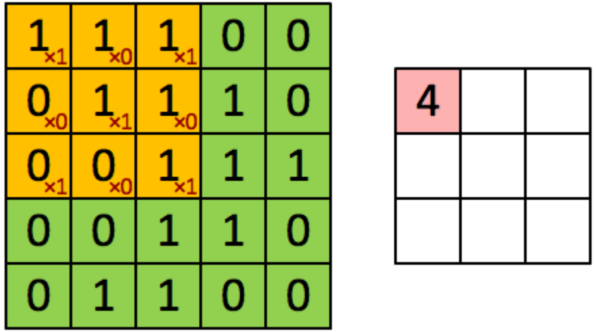
Model definition

Let's take a look at some other layers

Convolution neural networks

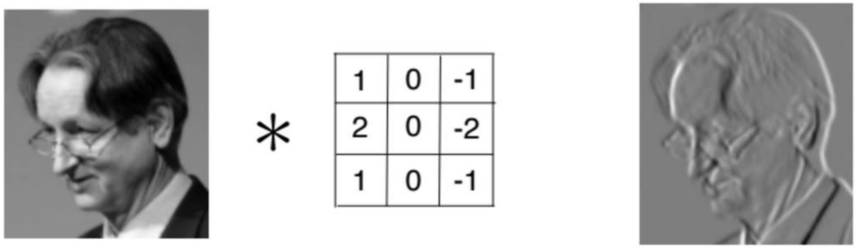
Convolution Layers

- Widely used in image related models (not limited)



Image

Convolved Feature



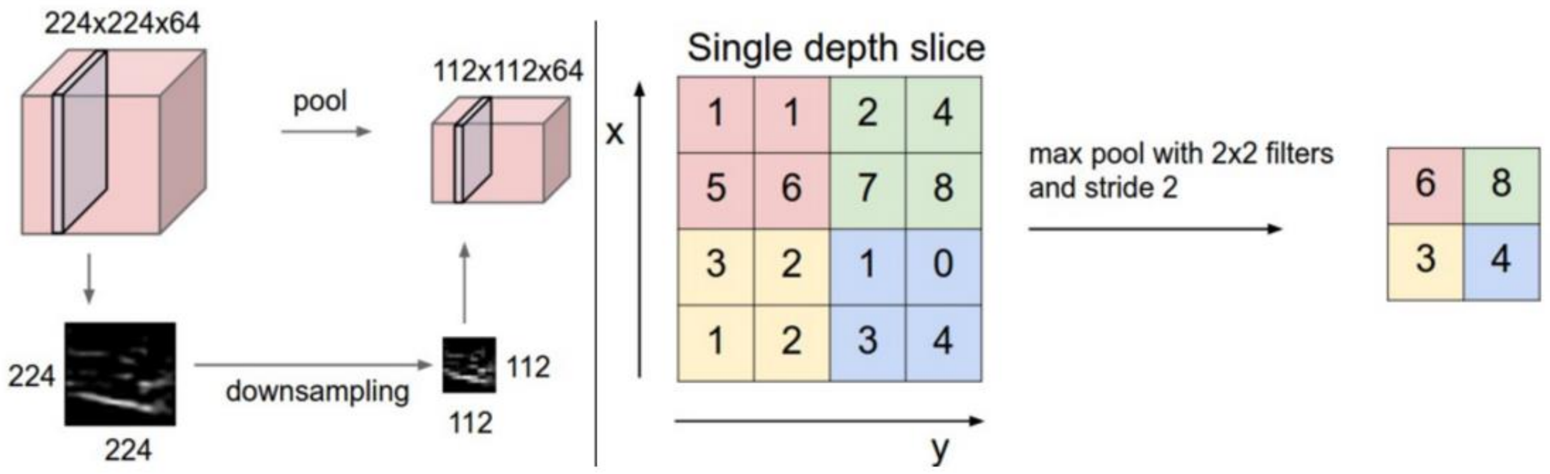
Images are from: https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/convolution.html

Convolution neural networks

```
SpatialConvolution(
    nInputPlane, nOutputPlane,
    kernelW, kernelH,
    strideW=1, strideH=1,
    padW=0, padH=0,
    nGroup=1,
    wRegularizer=null, bRegularizer=null,
    initWeight=null, initBias=null, initGradWeight=null, initGradBias=null
)
```

Convolution neural networks

Pooling

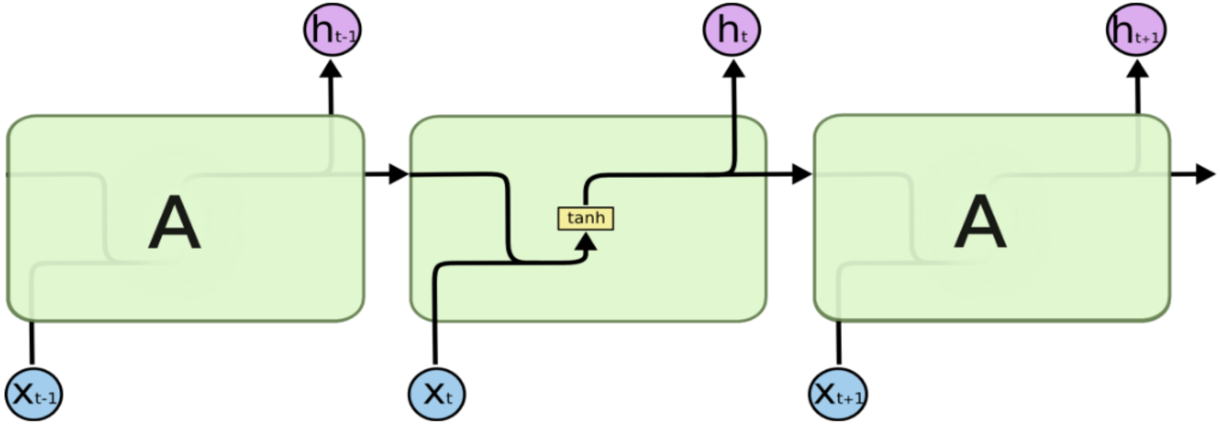


The image is from: https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/pooling_layer.html

Convolution neural networks

```
SpatialMaxPooling(
    kW, kH,
    dW=1, dH=1,
    padW=0, padH=0,
    ceilMode=false
)
```

RNN



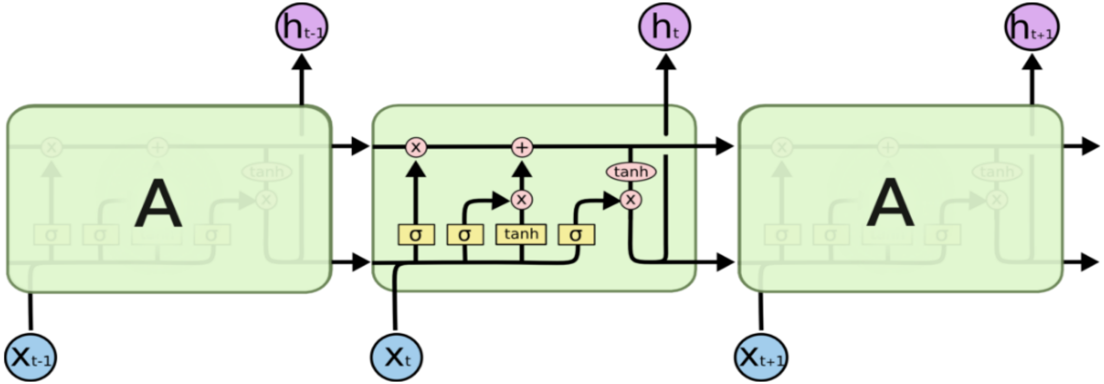
The repeating module in a standard RNN contains a single layer.

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN

```
model.add(
  Recurrent[Float]()
    .add(
      RnnCell[Float](inputSize, outptuSize, Tanh[Float]())
    )
)
```

LSTM



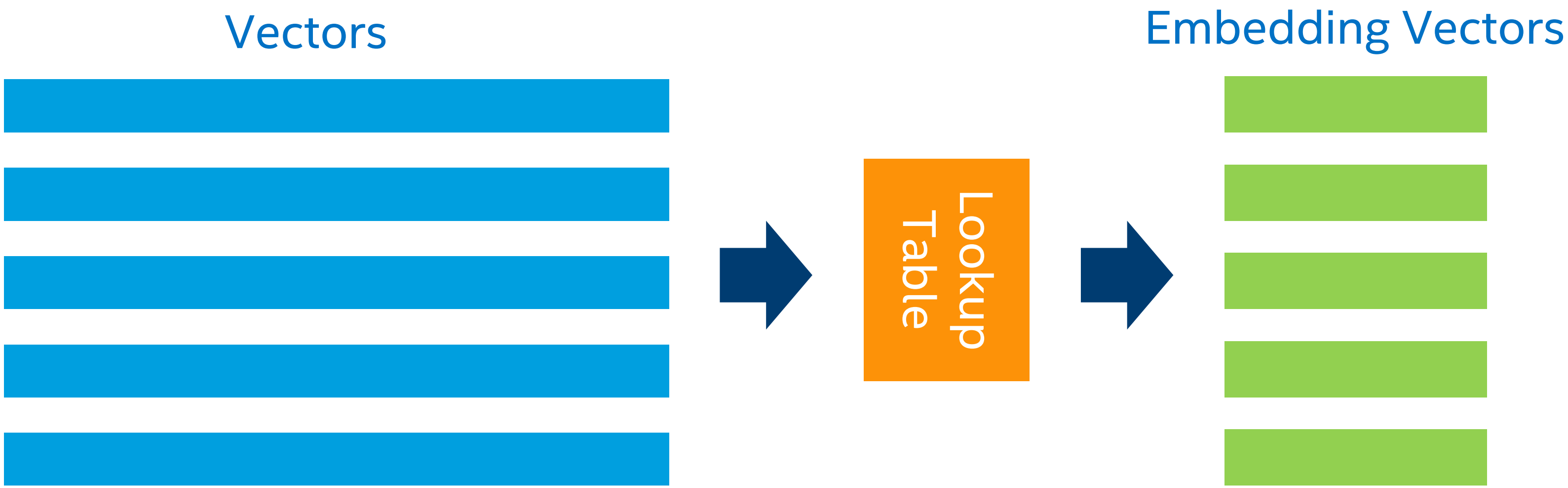
The repeating module in an LSTM contains four interacting layers.

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM

```
model.add(
    Recurrent[Float]()
        .add(
            LSTM[Float](inputSize, outptuSize)
        )
)
```

Embedding Layer



Embedding Layer

```
LookupTable(
    nIndex: Int, nOutput: Int,
    paddingValue: Double = 0,
    maxNorm: Double = Double.MaxValue,
    normType: Double = 2.0,
    shouldScaleGradByFreq: Boolean = false,
    wRegularizer: Regularizer[T] = null
)
```

TAKE A BREAK

TRAIN AND EVALUATE MODEL

How to train a model and how to evaluate it

We will take a look at

- How to prepare your data
- Define a training process
- Predict with your model

Data preprocess

The raw data(image, audio, text) can not be used with model directly

- They need to be convert to tensors
- Preprocessing is often necessary
  - Normalization
  - Embedding
  - Scale
  - Crop
  - Augmentation

Data preprocess

In Python, thanks to the rich data analytics libraries, you can do it easily

- Numpy, Pandas...

In Scala, BigDL provide several utilities to do preprocessing

```
trait Transformer[A, B] extends Serializable {
  def apply(prev: Iterator[A]): Iterator[B]
}
```

Data preprocess in Scala

```scala
class PathToImage extends Transformer[Path, Image]
class ImageToArray extends Transformer[Image, Array]
class Normalizor extends Transformer[Array, Array]
class Cropper extends Transformer[Array, Array]

PathToImage -> ImageToArray -> Normalizor -> Cropper
```

```scala
val rddA : RDD[A] = ...
val tran : Transformer[A, B] = ...
val rddB : RDD[B] = rdd.mapPartitions(tran(_))
```

Tensor

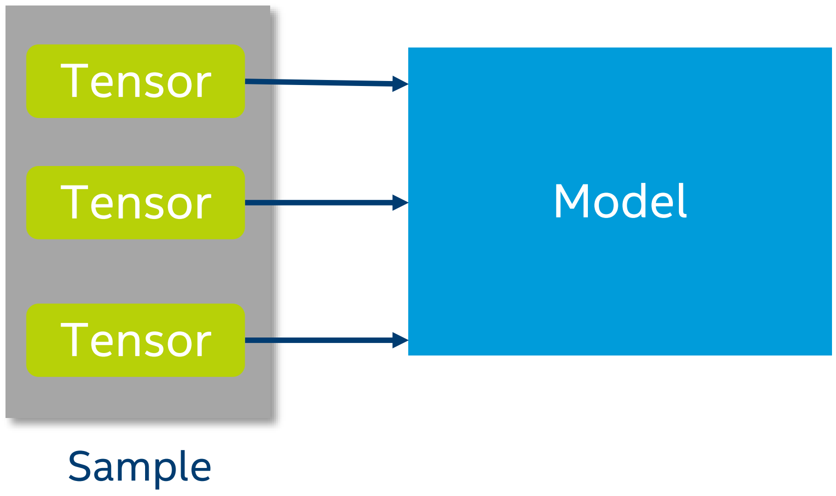
Numpy NDarray for Python

```
np.array(
  [
    [1.0, 1.0, 1.0, 1.0]
    [3.0, 3.0, 3.0, 3.0]
  ]
)
```

Tensor for Scala

```
Tensor[Float](
 T(
     T(1.0f, 1.0f, 1.0f, 1.0f),
     T(3.0f, 3.0f, 3.0f, 3.0f)
  )
)
```

Sample



In distributed training or inference

- RDD[Sample]

Let’s prepare some data

MNIST Dataset

- http://yann.lecun.com/exdb/mnist/

THE MNIST DATABASE

of handwritten digits

Yann LeCun, Courant Institute, NYU
Corinna Cortes, Google Labs, New York
Christopher J.C. Burges, Microsoft Research, Redmond

Sandbox enviroments

https://github.com/yiheng/OReillyAIConf#sandbox-environment

Take a look at MNIST data

```
%pylab inline
from bigdl.dataset import mnist

mnist_path = "datasets/mnist"
(train_images, train_labels) = mnist.read_data_sets(mnist_path, "train")
(test_images, test_labels) = mnist.read_data_sets(mnist_path, "test")

print train_images.shape
print train_labels.shape
print test_images.shape
print test_labels.shape

imshow(np.column_stack(train_images[0:10].reshape(10, 28,28)),cmap='gray'); axis('off')
print "groud true labels: "
print train_labels[0:10]
```

Convert MNIST to RDD (code part 1)

```
from bigdl.util.common import Sample
from bigdl.dataset import mnist

def get_mnist(sc, mnist_path):
    # target is start from 0,
    (train_images, train_labels) = mnist.read_data_sets(mnist_path, "train")
    (test_images, test_labels) = mnist.read_data_sets(mnist_path, "test")
    training_mean = np.mean(train_images)
    training_std = np.std(train_images)
    rdd_train_images = sc.parallelize(train_images)
    rdd_train_labels = sc.parallelize(train_labels)
    rdd_test_images = sc.parallelize(test_images)
    rdd_test_labels = sc.parallelize(test_labels)
```

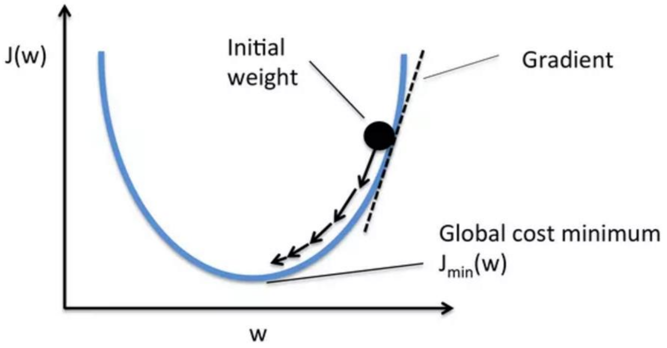
Convert MNIST to RDD (code part 2)

```
    rdd_train_sample = rdd_train_images.zip(rdd_train_labels).map(lambda (features,
label):
            Sample.from_ndarray((features - training_mean)/training_std, label + 1))
  rdd_test_sample = rdd_test_images.zip(rdd_test_labels).map(lambda (features, label):
            Sample.from_ndarray((features - training_mean)/training_std, label + 1))
  return (rdd_train_sample, rdd_test_sample)

(train_data, test_data) = get_mnist(sc, mnist_path)
print train_data.count()
print test_data.count()
```

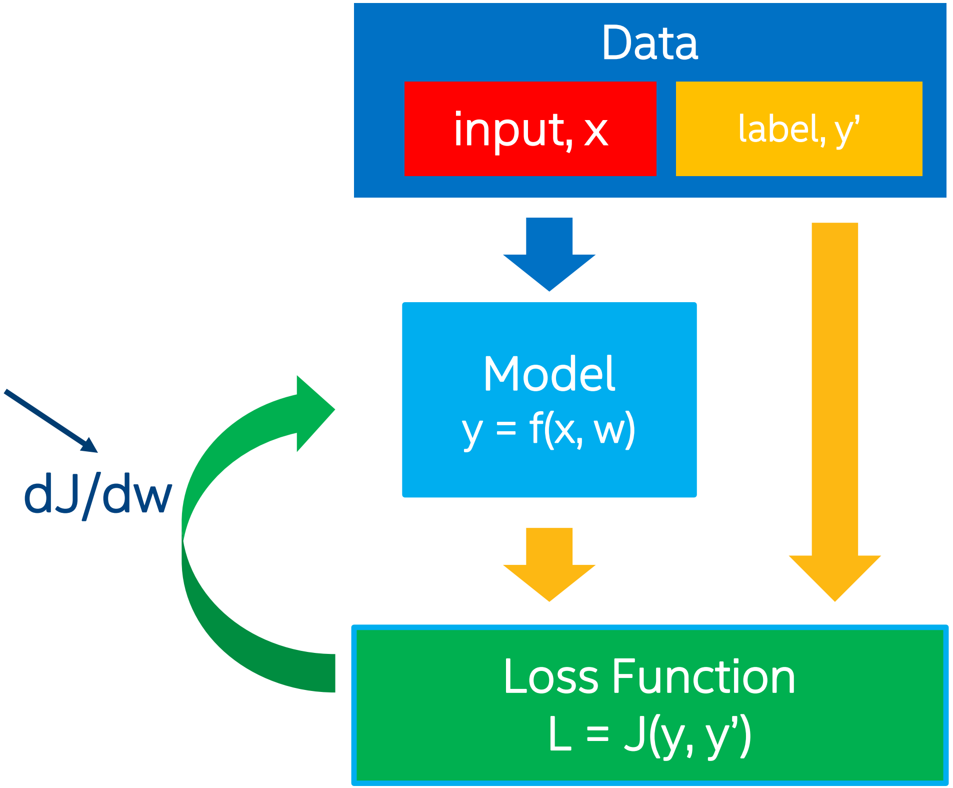
Define a training process

Take a look at the theory first

How to train the model



https://www.quora.com/Whats-the-difference-between-gradient-descent-and-stochastic-gradient-descent

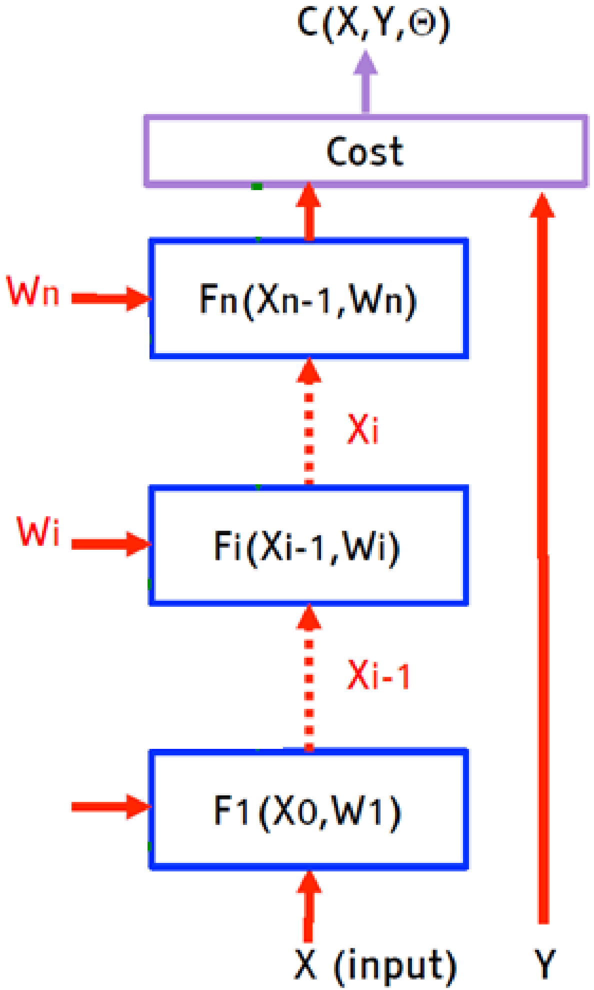


Supervised learning

Forward and backward

Model run a forward to get the output

- It's what actually inference do



NIPS2015 DL-Tutorial (Geoff Hinton, Yoshua Bengio, Yann LeCun)

Forward and backward

Backpropagation to calculate the gradients, maybe different graph path compare to forward

- Backprop for the activities

  dC / dXi-1 = dC / dXi * dXi / dXi-1

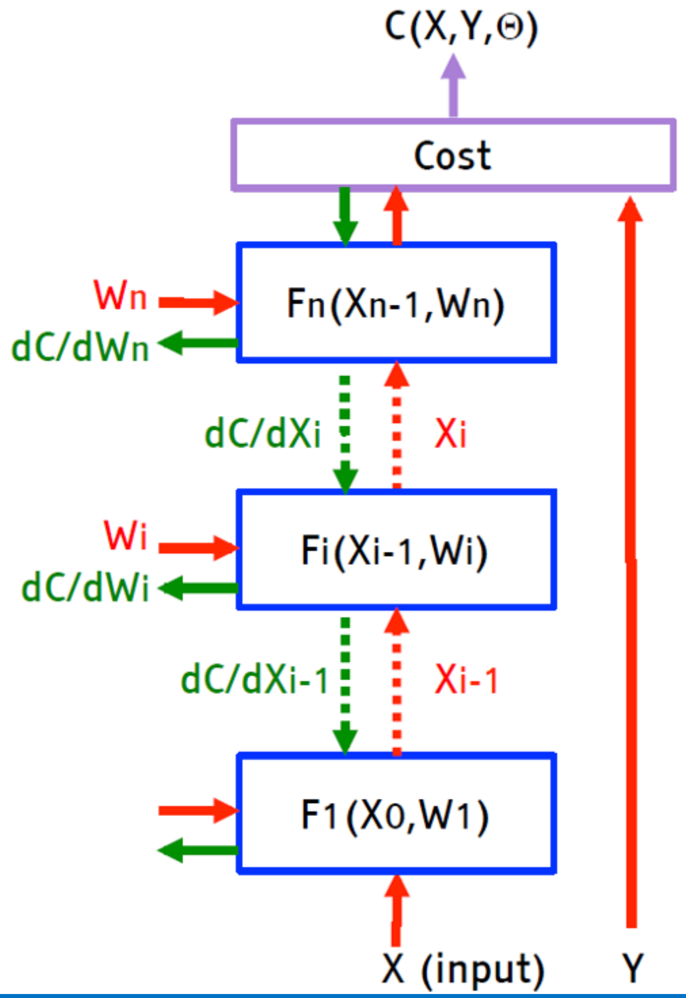
  dC / dXi-1 = dC / dXi * dFi(Xi-1, Wi) / dXi-1

- Backprop for the weights

  dC / dWi = dC/dXi * dXi / dXi-1

  dC / dWi = dC/dXi * dFi(Xi-1, Wi) / dWi

NIPS2015 DL-Tutorial (Geoff Hinton, Yoshua Bengio, Yann LeCun)



Now come back to the code

- Optimizer
  - Define a training process
- Optim Method
  - SGD, Adam…
- Triggers
  - when to validate
  - when to stop training
  - when to checkpoint model
- Batch size
  - Global batch size, should be dividable by the core number
- Validation method
  - Evaluate metric on validation data

Train a logistic regression model

```
%%time
from bigdl.nn.layer import *
from bigdl.nn.criterion import *
from bigdl.optim.optimizer import *
from bigdl.util.common import *

def logistic_regression(n_input, n_classes):
    # Initialize a sequential container
    model = Sequential()

    model.add(Reshape([28*28]))
    model.add(Linear(n_input, n_classes))
    model.add(LogSoftMax())

    return model
```

Train a logistic regression model

```
model = logistic_regression(784, 10)

init_engine()
optimizer = Optimizer(
    model=model,
    training_rdd=train_data,
    criterion=ClassNLLCriterion(),
    optim_method="SGD",
    state={"learningRate": 0.2},
    end_trigger=MaxEpoch(15),
    batch_size=2048)

# Start to train
trained_model = optimizer.optimize()
print "Optimization Done."
```

Inference

```
def map_predict_label(l):
    return np.array(l).argmax()
def map_groundtruth_label(l):
    return l[0] - 1

# Prediction
predictions = trained_model.predict(test_data)
imshow(np.column_stack([np.array(s.features).reshape(28,28) for s in
test_data.take(8)]),cmap='gray'); axis('off')
print 'Ground Truth labels:'
print ', '.join(str(map_groundtruth_label(s.label)) for s in test_data.take(8))
print 'Predicted labels:'
print ', '.join(str(map_predict_label(s)) for s in predictions.take(8))
```

Train a CNN model

```
%%time
def build_model(class_num):
    model = Sequential()
    model.add(Reshape([1, 28, 28]))
    model.add(SpatialConvolution(1, 6, 5, 5).set_name('conv1'))
    model.add(Tanh())
    model.add(SpatialMaxPooling(2, 2, 2, 2).set_name('pool1'))
    model.add(Tanh())
    model.add(SpatialConvolution(6, 12, 5, 5).set_name('conv2'))
    model.add(SpatialMaxPooling(2, 2, 2, 2).set_name('pool2'))
    model.add(Reshape([12 * 4 * 4]))
    model.add(Linear(12 * 4 * 4, 100).set_name('fc1'))
    model.add(Tanh())
    model.add(Linear(100, class_num).set_name('score'))
    model.add(LogSoftMax())
    return model
```

Train a CNN model

```
lenet_model = build_model(10)

import datetime as dt

optimizer = Optimizer(
    model=lenet_model,
    training_rdd=train_data,
    criterion=ClassNLLCriterion(),
    optim_method="SGD",
    state={"learningRate": 0.4, "learningRateDecay": 0.0002},
    end_trigger=MaxEpoch(5),
    batch_size=2048)
```

Train a CNN model

```
optimizer.set_validation(
    batch_size=2048,
    val_rdd=test_data,
    trigger=EveryEpoch(),
    val_method=["Top1Accuracy"]
)

app_name='lenet-'+dt.datetime.now().strftime("%Y%m%d-%H%M%S")
train_summary = TrainSummary(log_dir='/tmp/bigdl_summaries',
                                     app_name=app_name)
train_summary.set_summary_trigger("Parameters", SeveralIteration(50))
val_summary = ValidationSummary(log_dir='/tmp/bigdl_summaries',
                                        app_name=app_name)
optimizer.set_train_summary(train_summary)
optimizer.set_val_summary(val_summary)
print "saving logs to ",app_name
```

Train a CNN model

```
# Boot training process
trained_model = optimizer.optimize()
print "Optimization Done."
```

Visualize your training

```
loss = np.array(train_summary.read_scalar("Loss"))
top1 = np.array(val_summary.read_scalar("Top1Accuracy"))

plt.figure(figsize = (12,12))
plt.subplot(2,1,1)
plt.plot(loss[:,0],loss[:,1],label='loss')
plt.xlim(0,loss.shape[0]+10)
plt.grid(True)
plt.title("loss")
plt.subplot(2,1,2)
plt.plot(top1[:,0],top1[:,1],label='top1')
plt.xlim(0,loss.shape[0]+10)
plt.title("top1 accuracy")
plt.grid(True)
```

MODEL TUNING

Tuning your optimization

- Choose the hyper-parameter of the optimization algorithm carefully
- Hyper-parameter need to adjust when batch size change
- Set log level of com.intel.analytics.bigdl.optim.DistriOptimizer to debug to see fine details of your training process
- Use physical core number of your server, which means if hyper-thread is turned on, use half of the v-core number
- Initialize your model correctly
- Use some regularization
- Visualize your training process

Initialize your model correctly

Model parameter is initialized randomly. You can change how to init them

- Uniform distribution
- Normal distribution
- Constant
- Xavier
- Bilinear

Bad initialization may cause model can't train

Regularization

Regularization is important to improve model quality

Set it in optimization algorithm

Python: val sgd = new SGD(..., weightDecay = 0.001, ...)

Scala: sgd = SGD(..., weight_decay = 0.001, ...)

Set it layer wise

```
Linear(inputN, outputN,
        wRegularizer = L2Regularizer(0.1),
        bRegularizer = L2Regularizer(0.1))
```

The challenge to train deep model

Gradient vanishing / exploding

- ReLU
- Initialize model correctly (Xavier/pre-trained model)
- Batchnormalization

Overfitting

- More data (data augumentation)
- Regularization
- Dropout

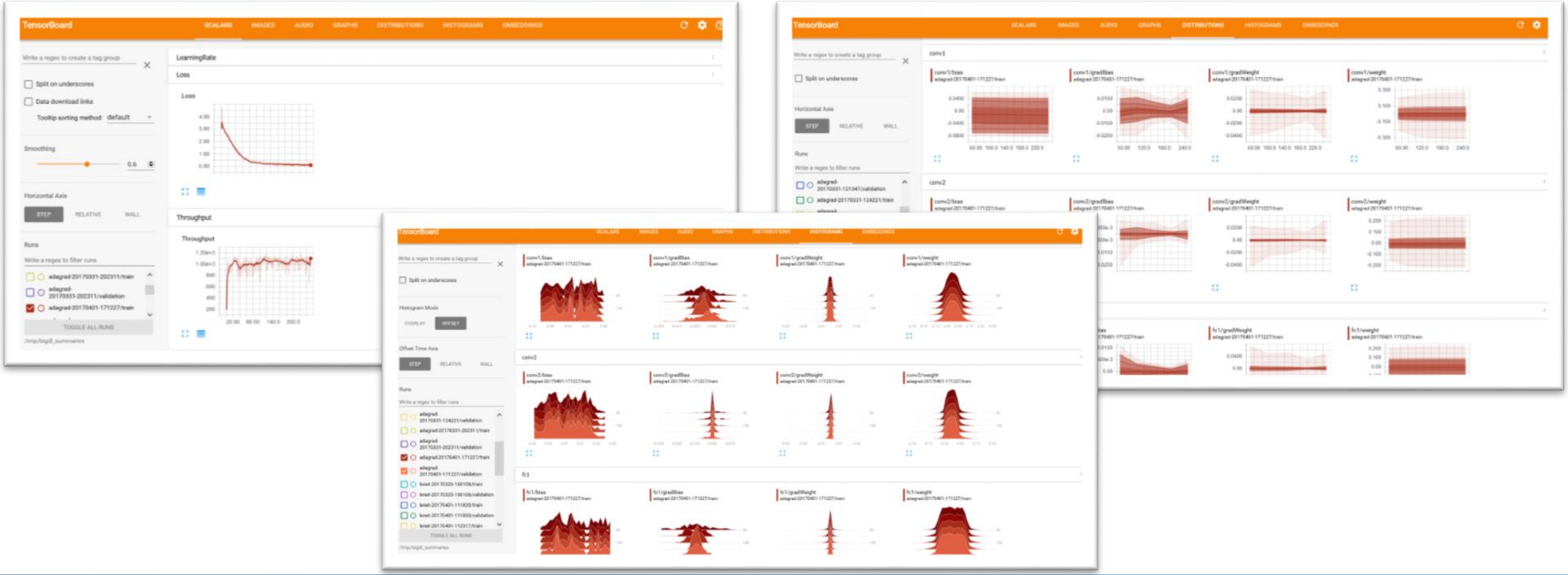
Visualize your training process

Turn on persist training summary, Scala

```
val optimizer = Optimizer(...)
...
val logdir = "mylogdir"
val appName = "myapp"
val trainSummary = TrainSummary(logdir, appName)
trainummary.setSummaryTrigger("Parameters", Trigger.severalIteration(20))
val validationSummary = ValidationSummary(logdir, appName)
optimizer.setTrainSummary(trainSummary)
optimizer.setValidationSummary(validationSummary)
...
val trained_model = optimizer.optimize()
```

```
pip install tensorboard==1.0.0a4
tensorboard --logdir=/tmp/bigdl_summaries
```

Visualize your training process



Software

Legal Disclaimer

Risk Factors

The above statements and any others in this document that refer to plans and expectations for the first quarter, the year and the future are forward-looking statements that involve a number of risks and uncertainties. Words such as “anticipates,” “expects,” “intends,” “plans,” “believes,” “seeks,” “estimates,” “may,” “will,” “should” and their variations identify forward-looking statements. Statements that refer to or are based on projections, uncertain events or assumptions also identify forward-looking statements. Many factors could affect Intel’s actual results, and variances from Intel’s current expectations regarding such factors could cause actual results to differ materially from those expressed in these forward-looking statements. Intel presently considers the following to be the important factors that could cause actual results to differ materially from the company’s expectations. Demand could be different from Intel's expectations due to factors including changes in business and economic conditions; customer acceptance of Intel’s and competitors’ products; supply constraints and other disruptions affecting customers; changes in customer order patterns including order cancellations; and changes in the level of inventory at customers. Uncertainty in global economic and financial conditions poses a risk that consumers and businesses may defer purchases in response to negative financial events, which could negatively affect product demand and other related matters. Intel operates in intensely competitive industries that are characterized by a high percentage of costs that are fixed or difficult to reduce in the short term and product demand that is highly variable and difficult to forecast. Revenue and the gross margin percentage are affected by the timing of Intel product introductions and the demand for and market acceptance of Intel's products; actions taken by Intel's competitors, including product offerings and introductions, marketing programs and pricing pressures and Intel’s response to such actions; and Intel’s ability to respond quickly to technological developments and to incorporate new features into its products. The gross margin percentage could vary significantly from expectations based on capacity utilization; variations in inventory valuation, including variations related to the timing of qualifying products for sale; changes in revenue levels; segment product mix; the timing and execution of the manufacturing ramp and associated costs; start-up costs; excess or obsolete inventory; changes in unit costs; defects or disruptions in the supply of materials or resources; product manufacturing quality/yields; and impairments of long-lived assets, including manufacturing, assembly/test and intangible assets. Intel's results could be affected by adverse economic, social, political and physical/infrastructure conditions in countries where Intel, its customers or its suppliers operate, including military conflict and other security risks, natural disasters, infrastructure disruptions, health concerns and fluctuations in currency exchange rates. Expenses, particularly certain marketing and compensation expenses, as well as restructuring and asset impairment charges, vary depending on the level of demand for Intel's products and the level of revenue and profits. Intel’s results could be affected by the timing of closing of acquisitions and divestitures. Intel's results could be affected by adverse effects associated with product defects and errata (deviations from published specifications), and by litigation or regulatory matters involving intellectual property, stockholder, consumer, antitrust, disclosure and other issues, such as the litigation and regulatory matters described in Intel's SEC reports. An unfavorable ruling could include monetary damages or an injunction prohibiting Intel from manufacturing or selling one or more products, precluding particular business practices, impacting Intel’s ability to design its products, or requiring other remedies such as compulsory licensing of intellectual property. A detailed discussion of these and other factors that could affect Intel’s results is included in Intel’s SEC filings, including the company’s most recent reports on Form 10-Q, Form 10-K and earnings release.