BIGDL: DISTRIBUTED DEEP LEARNING ON APACHE SPARK

Yiheng Wang (yiheng.wang@intel.com)

Jennie Wang (jiao.wang@intel.com)

Big Data Technologies, Software and Service Group, Intel

Intel Big Data Team

A global team

- US, China, India

Strong leadership in the open source community

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

Technology and innovation oriented

- Real-time analytics, advanced analytics
- BigDL, SparseML, StatisticsOnSpark, OAP…
- Next generations of Big Data solutions with Intel customers

Agenda

❖ BigDL Introduction

- Why BigDL
- Apache Spark Basics
- Install & Running BigDL
- Model Definition in BigDL
- Prepare Data
- Use & Load Model
- Train Model
- Others

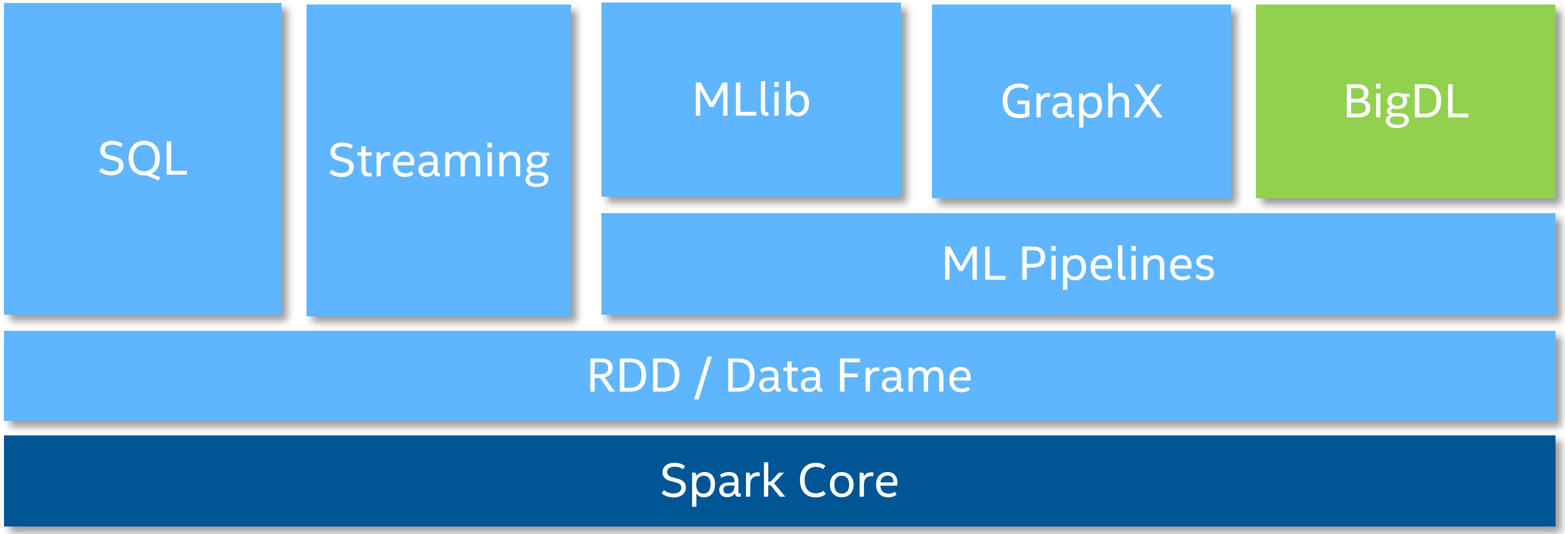
❖ Hand-on Tutorials

- Apache Spark Basics
- Deep Learning on Spark

BIGDL INTRODUCTION

What is BigDL

BigDL is a distributed deep learning library on Apache Spark*. Currently, it is implemented as a standalone library.



Why BigDL

There're a lot of deep learning frameworks. Only list a part of them

neon

Caffe

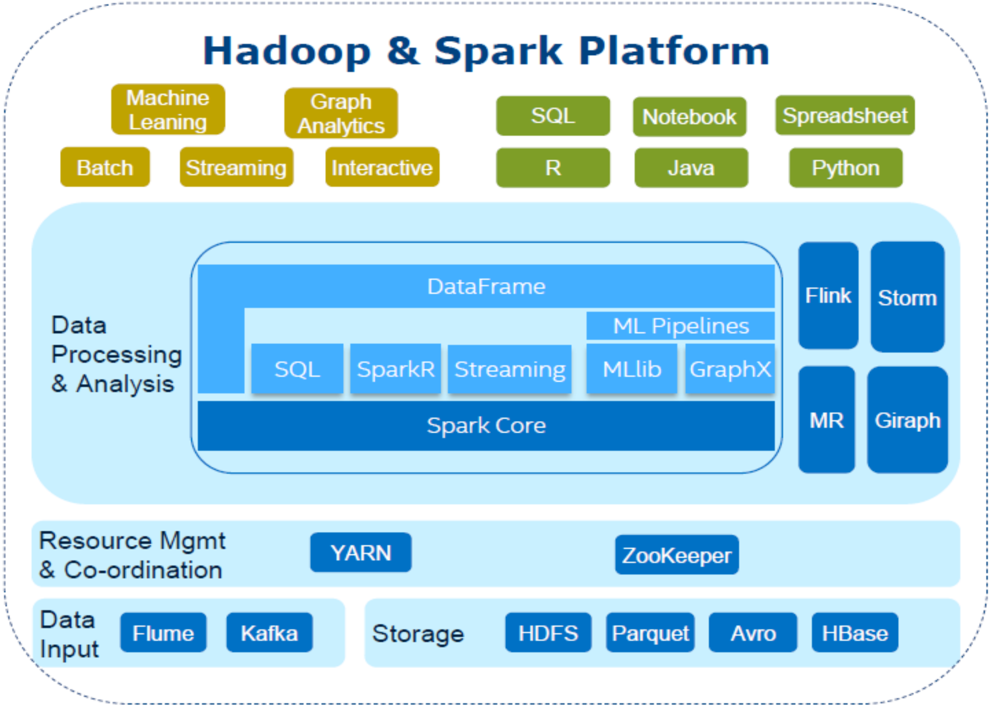
theano

mxnet

CNTK

Why BigDL

BigDL: Run deep learning on Big Data platform



Outstanding features

- Massively distributed
- Fault tolerance
- Elasticity
- Dynamic resource sharing
- …

Why BigDL

The benefits of running deep learning on Hadoop/Spark platform

- Close to your data. Easier to analyze a large amount of data on the same cluster where the data are stored
- Quickly build end-to-end deep learning solution for your big data
- Manage various big data analytics workload in one cluster, better leverage your machines

Why BigDL

Other benefits from using BigDL:

- Get the best performance of your servers
  - Powered by MKL
- Efficiently scale out
  - Hundreds of node
- Easy to deploy
  - Only need a Spark cluster, no other setup
- Rich deep learning support

Why BigDL

People use BigDL to build applications

- Large internet company
- Financial company
- Manufactory company
- Medical school

Image, Recommendation, Fraud detection, Audio, NLP

APACHE SPARK INTRODUCTION

Apache Spark*

What is Apache Spark*?

- Apache Spark* is a fast and general engine for large-scale data processing
- A powerful tool of data scientist and big data engineer

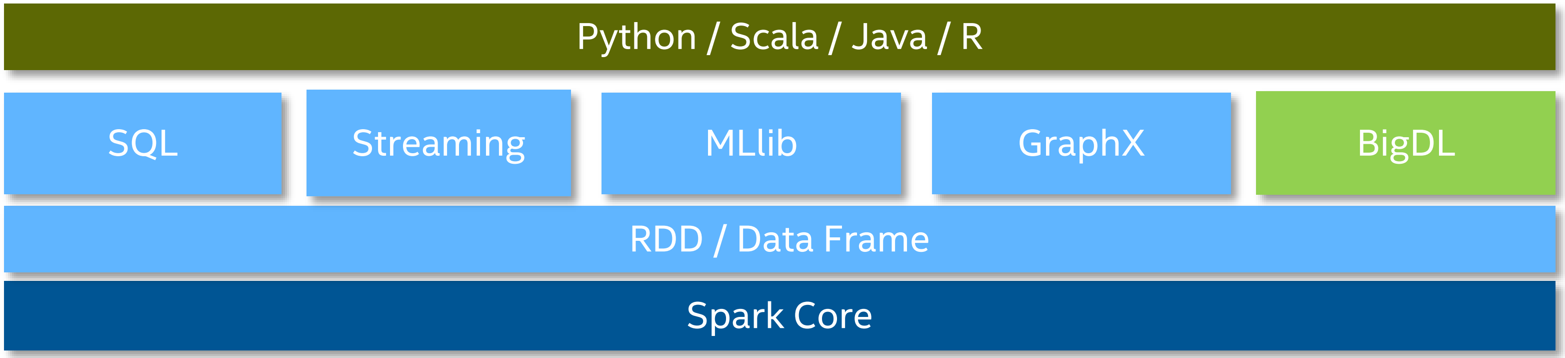
Adopted widely by industry, large community

- Large internet company, large ISP, financial company, start-ups
- Over 365K meetup numbers in 2017 (from Spark summit west 2017)

Apache Spark*

Spark Philosophy

- Unified engine/interface for complete data applications
- Streaming, SQL, ML, Graph in the same framework
- Multiple Programming APIs



Resilient Distributed Datasets (RDD)

Simple concept → RDD is a distributed list

RDD[Int] = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

Resilient Distributed Datasets (RDD)

RDD is lazy evaluated

rdd1 = rdd.map(_ + 1) ⇐ Nothing happen

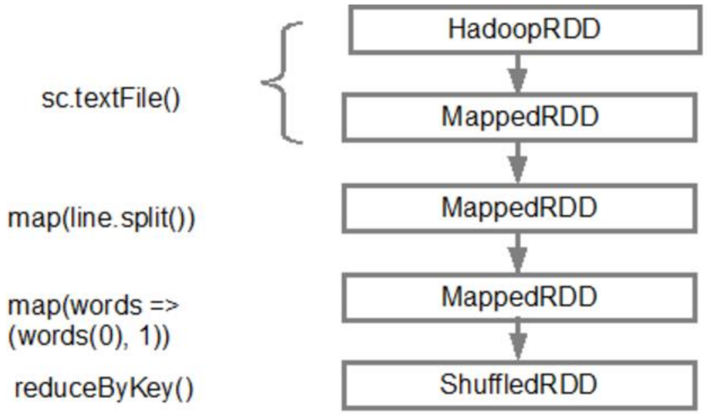
rdd2 = rdd1.zip(rdd) ⇐ Nothing happen

rdd2.count() ⇐ Bump!

Resilient Distributed Datasets (RDD)

RDD execution is optimized

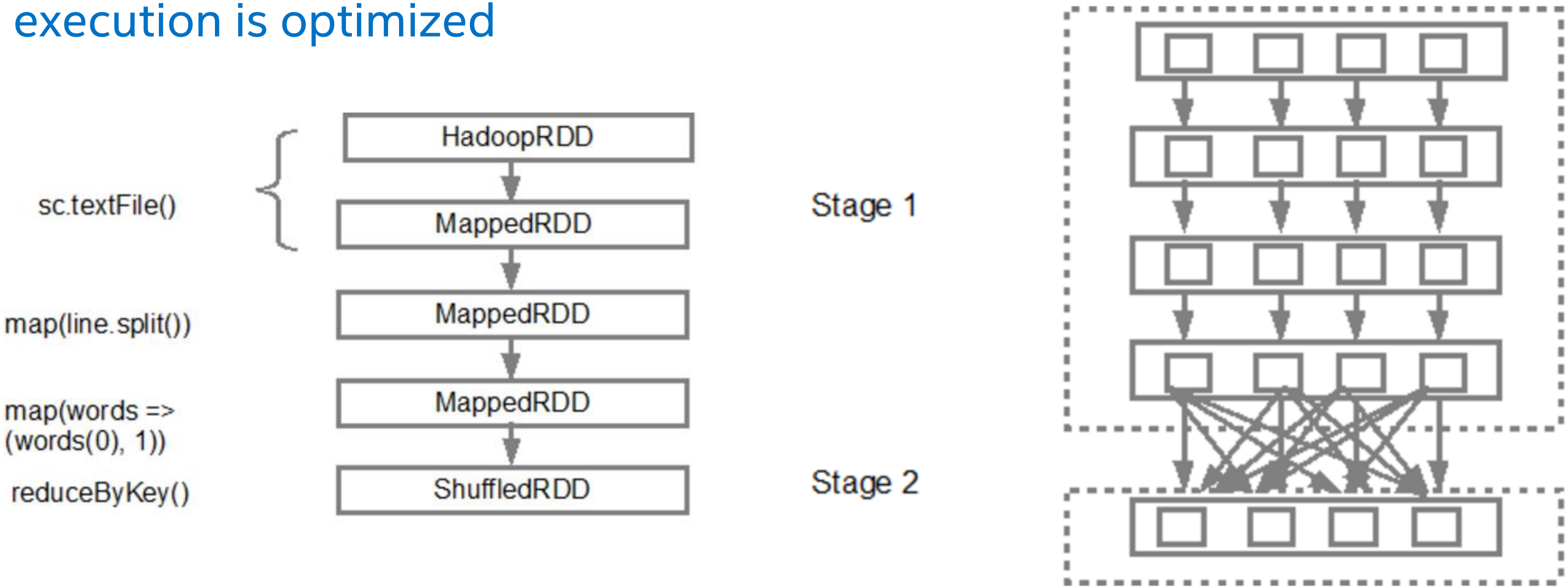
```
val input = sc.textFile("log.txt")
val splitedLines = input.map(line => line.split(" "))
                        .map(words => (words(0), 1))
                        .reduceByKey{(a,b) => a + b}
```



The example is refer to https://stackoverflow.com/questions/25836316/how-dag-works-under-the-covers-in-rdd

Resilient Distributed Datasets (RDD)

RDD execution is optimized



The example is refer to https://stackoverflow.com/questions/25836316/how-dag-works-under-the-covers-in-rdd

Resilient Distributed Datasets (RDD)

Other features of RDD

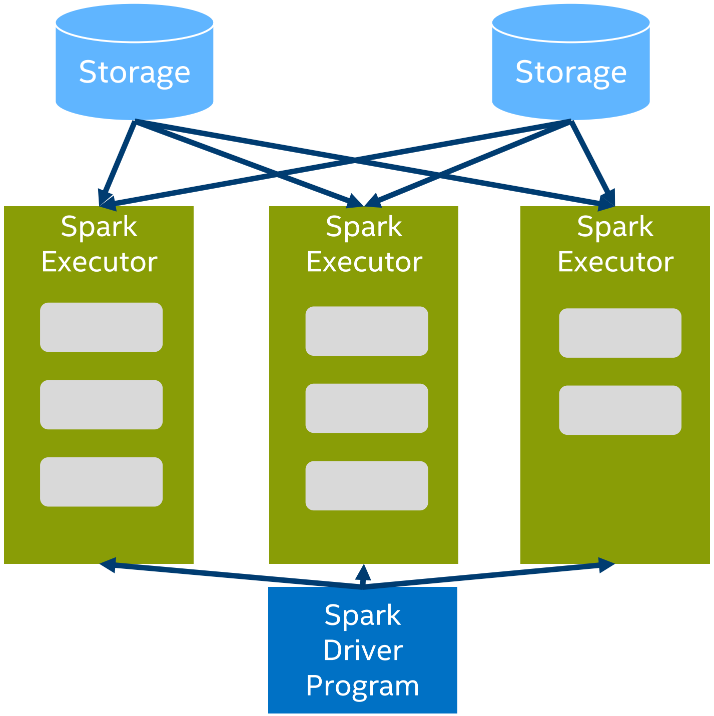
- RDD result can be cached in memory
- RDD is fault tolerant

DataFrame was brought in since Spark 1.5

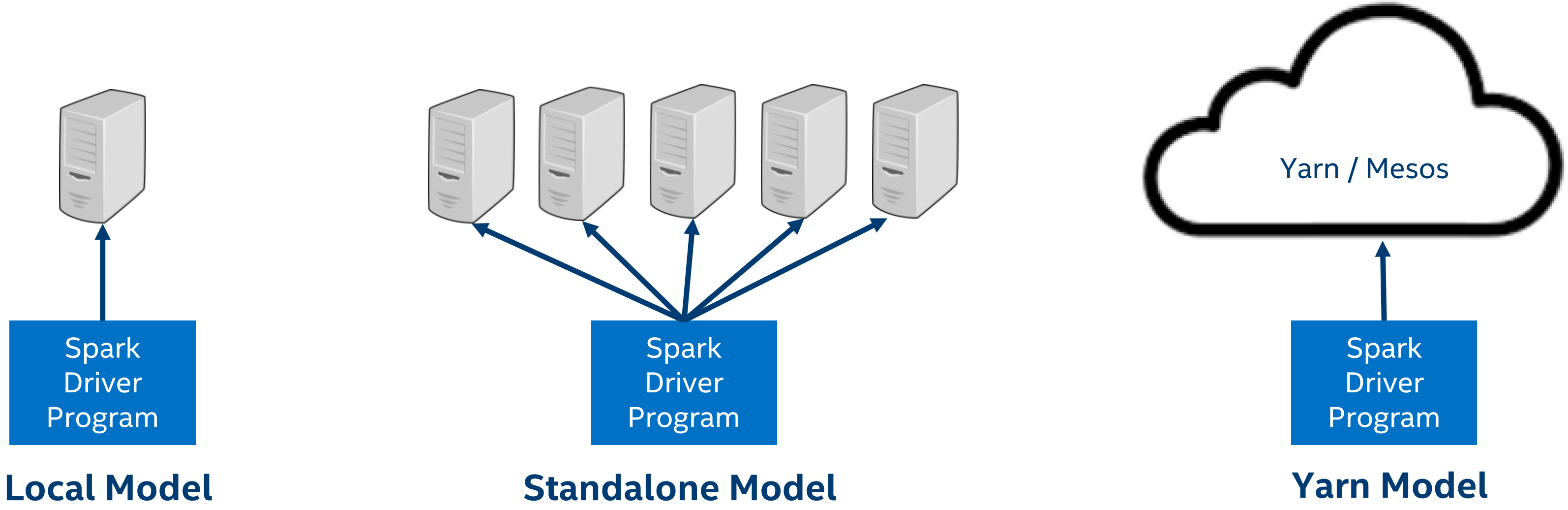
- Have schema, find some error at compile time
- Optimized SQL operations
- Reduce data space in memory

Lifecycle of a simple Spark program

1. Spark program is submitted by spark-submit
2. Spark driver/executor start
3. Spark driver run your program
4. Spark driver package RDD operation code and send them to executor to run
5. Go back to 3 until program finished



Apache Spark* Deploy Mode

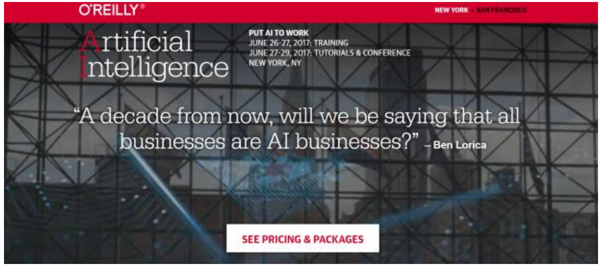


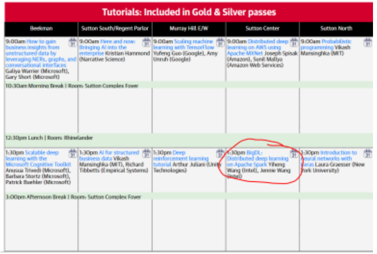
HAND-ON TUTORIAL

Checkout the instructions

https://github.com/yiheng/OReillyAIConf

- Docker
- Mac
- Linux
- Sandbox server





Materials or downloads needed in advance

- A laptop with the course materials downloaded from the GitHub repo

INSTALL AND RUN BIGDL ON SPARK

Install and Start to use BigDL

- Choice 1: Download prebuild package
  - Download Page (https://github.com/intel-analytics/BigDL/wiki/Downloads)
  - Stable release and nightly build
  - Python development, run examples

- Choice 2: Use maven/sbt to download
  - 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>
```

Install and Start to use BigDL(cont'd)

- Build it yourself
  - Customized configuration, e.g. JDK 8, Spark version
  - Develop BigDL

Example commands to build BigDL, note that this is for latest master code. In older version, say 0.1.1, the spark_2.x profile should be replaced by spark_2.0/spark_2.1 for different versions, and there was a spark_1.6 profile.

```
$ 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 Program with BigDL

Run scala code

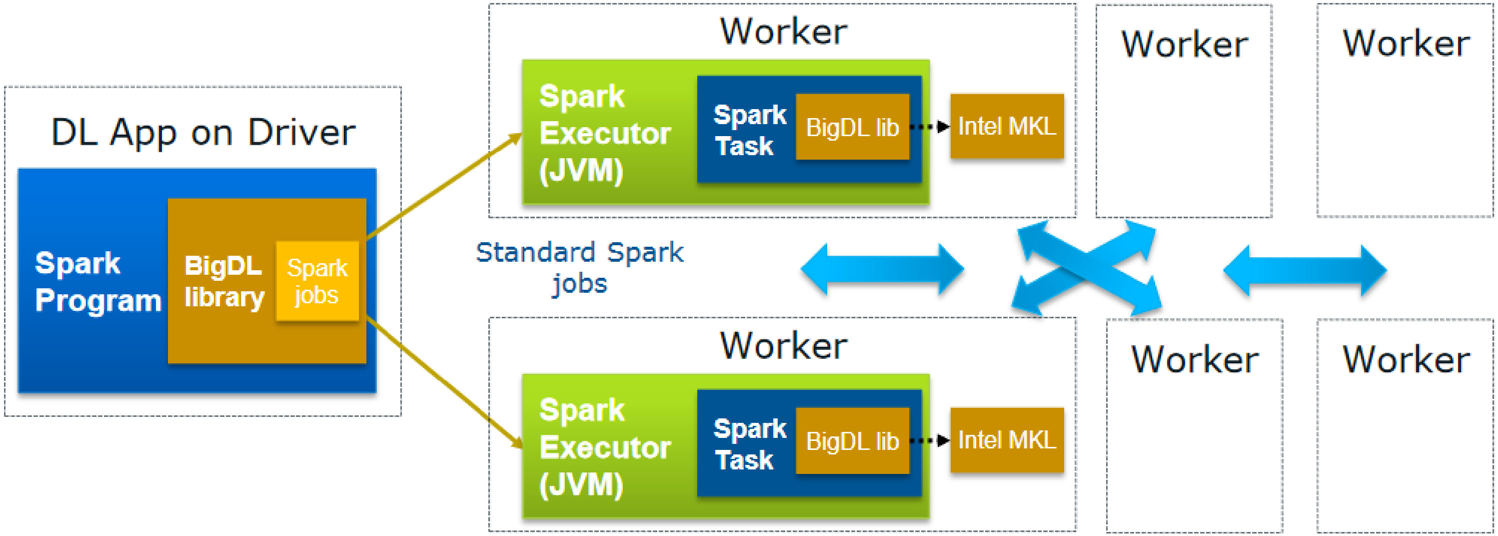
```
spark-submit \
  --master xxx
  --jars path_to_big_dl_jar
  --class main_class_full_name
  --……
  your_project_jar
  ……
```

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*



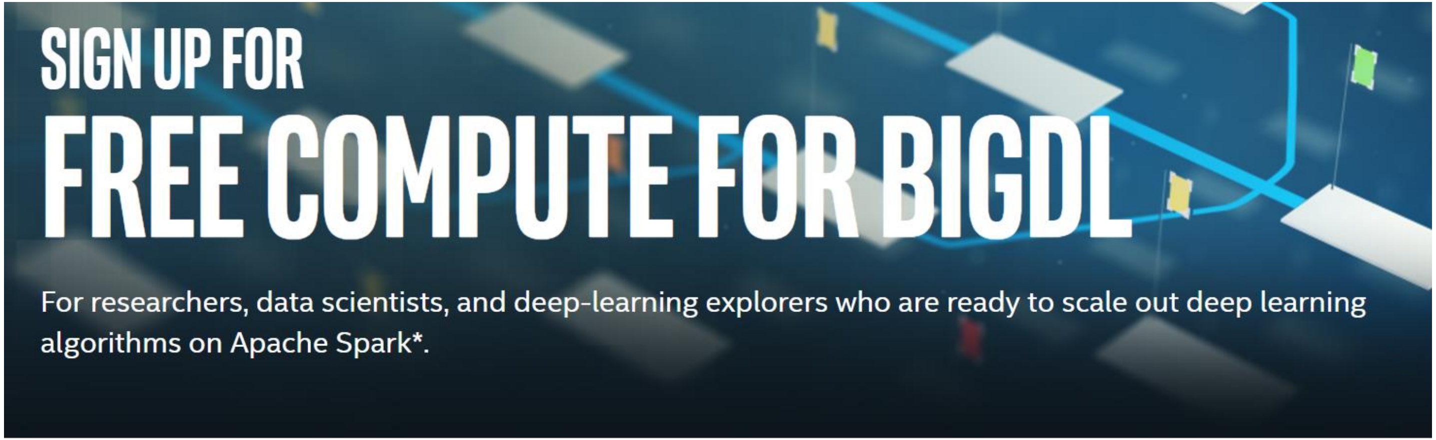
BigDL on public cloud service

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

Sign up for free compute for BigDL

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



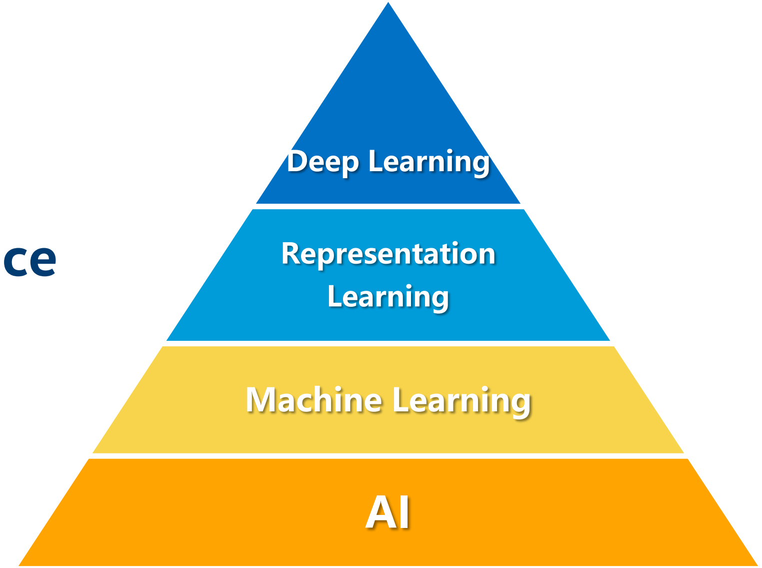
Preregister for Free Compute for BigDL, sponsored by Intel, and provide feedback to help make BigDL better for new users. You don't need to share your code. Preference goes to those who share their BigDL story.

NEURAL NETWORK MODEL IN BIGDL

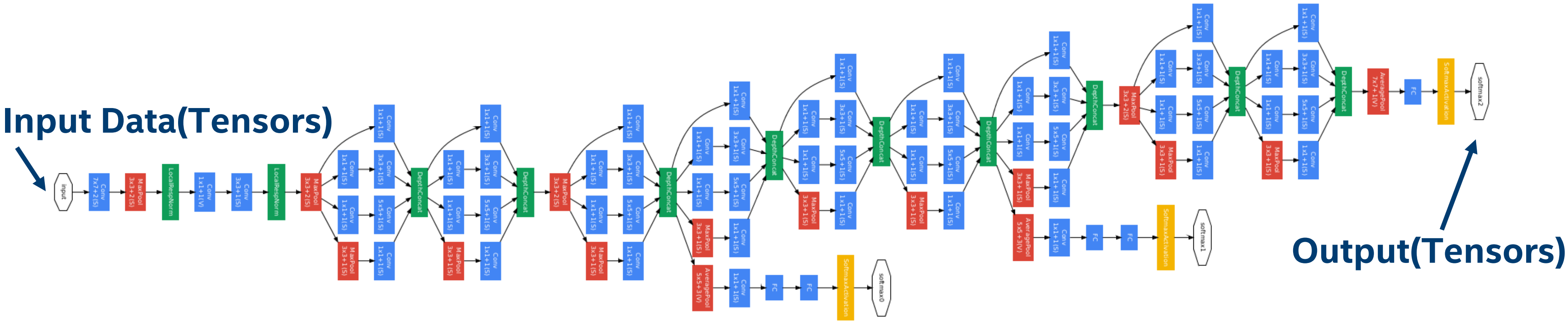
Deep Learning

What is deep learning?

A Way To Artificial Intelligence



What is Neural Network



Function composition: $y = f(g(h(i(j(k(l(m(x))))))))$

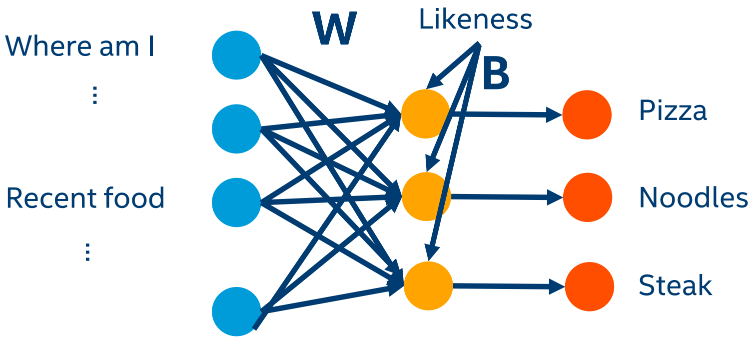
Define a model (Linear Classifier)

Scala

```
val model = Sequential()

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

Python

```
model = Sequential()

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



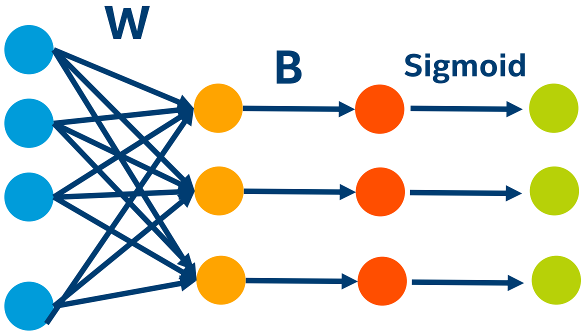
Simple Linear classification

$$Y = X * W + B$$

Add activation functions

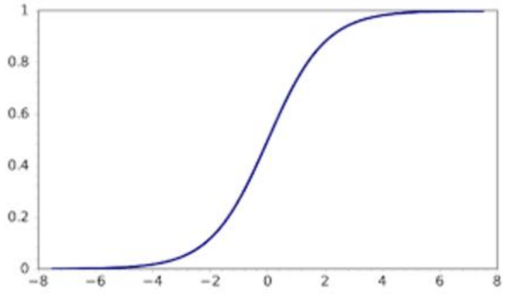
Scala

```
val model = Sequential()

model.add(Linear(4, 3))

model.add(Sigmoid())
```



Python

```
model = Sequential()

model.add(Linear(4, 3))

model.add(Sigmoid())
```



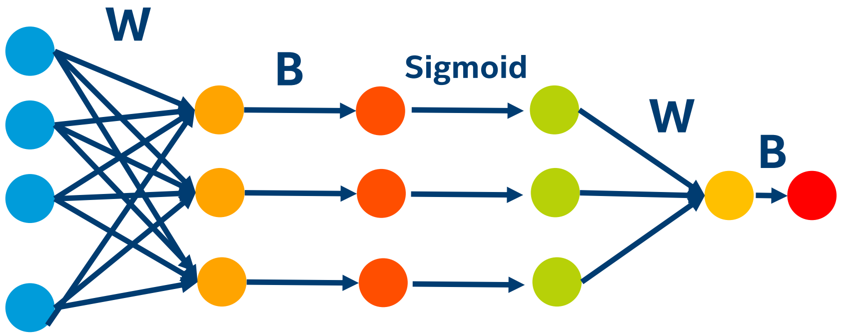
Multiple Layers

Scala

```
val model = Sequential()

model.add(Linear(4, 3))

model.add(Sigmoid())

model.add(Linear(3, 1))
```

Python

```
model = Sequential()

model.add(Linear(4, 3))

model.add(Sigmoid())

model.add(Linear(3, 1))
```

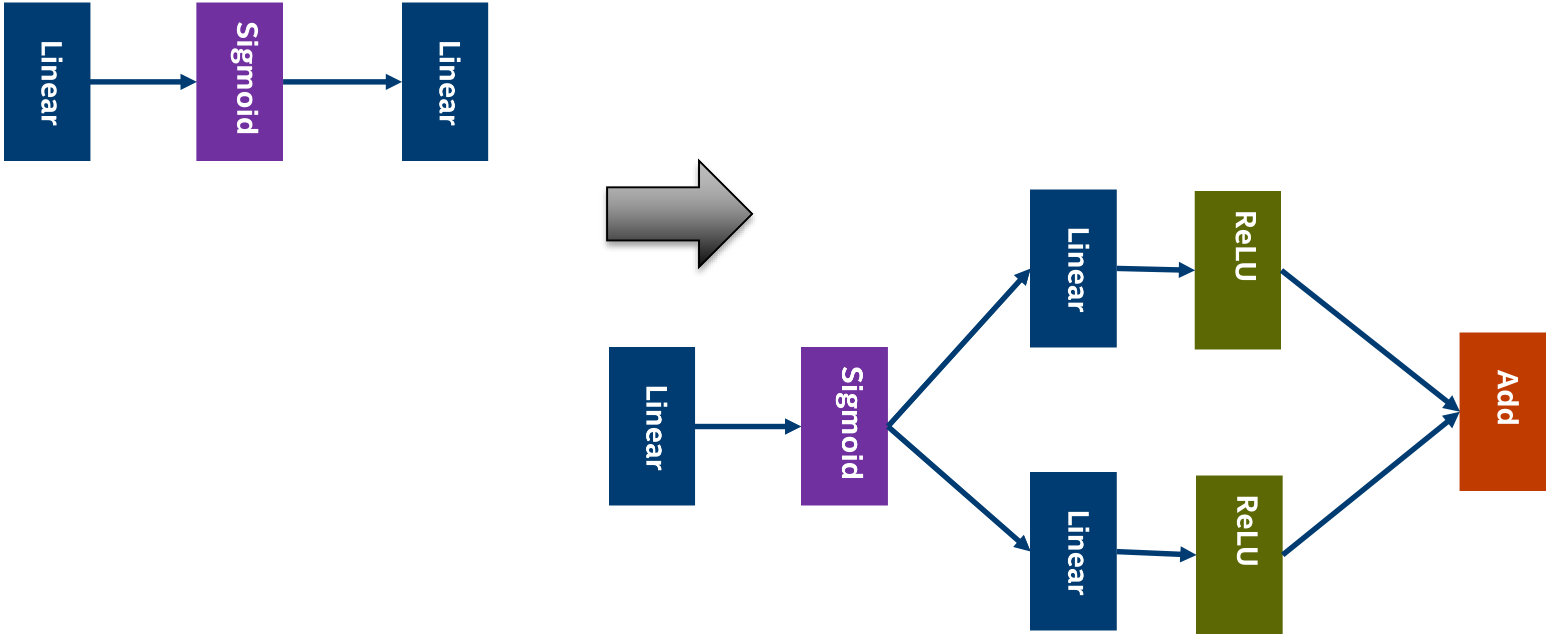


You need non-linear activation function to build multi-layer model

model.add(Linear(4, 3)).add(Linear(3, 1)) ==
model.add(Linear(4, 1))

without non-linear activation fucntion

Define more complex neural network

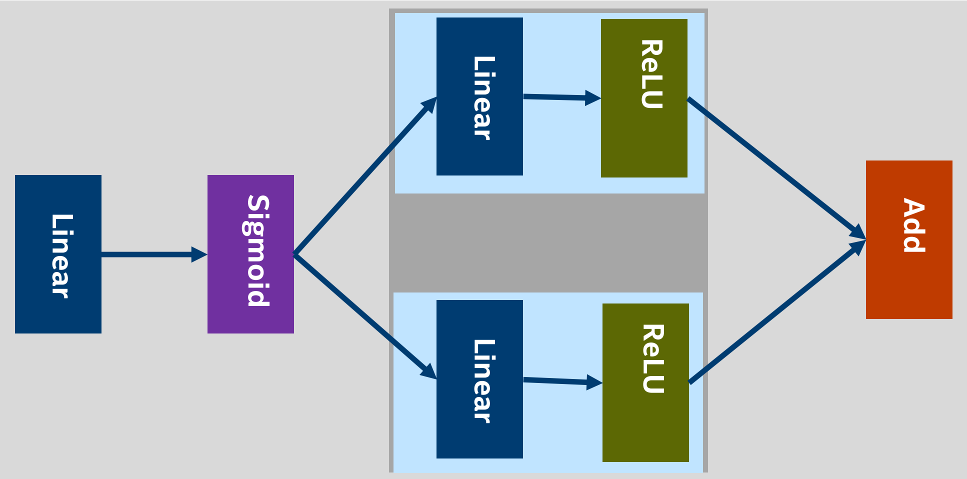


Sequential Style Model Definition

Python

```
model = Sequential()

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

branch1 = Sequential()

branch1.add(Linear(3, 2)).add(ReLU())

branch2 = Sequential()

branch2.add(Linear).add(ReLU)

branches = Sequential()

branches.add(branch1).add(branch2)

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

Layers named with Table deal with multiple input/output

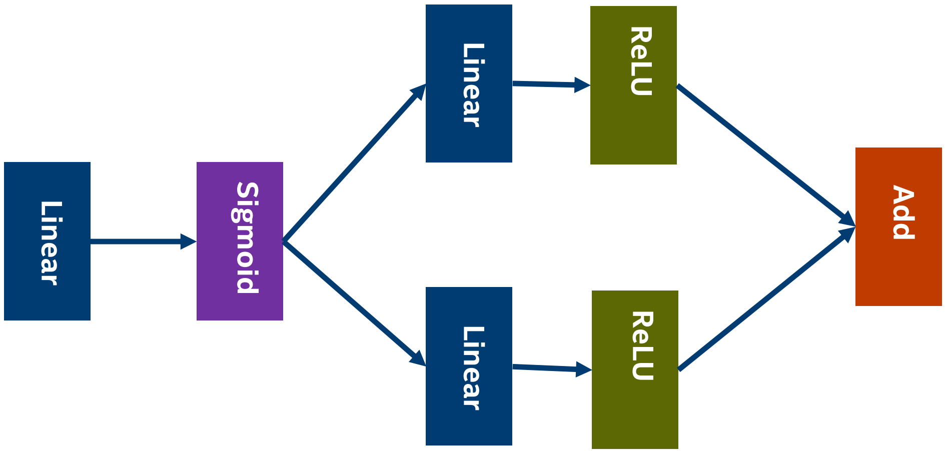


Functional Style Model Definition

Python

```
linear1 = Linear(4, 3)()

sigmoid = Sigmoid()linear1)

linear2 = Linear(3, 2)(sigmoid)

relu1 = ReLU()(linear2)

linear3 = Linear(3, 2)(sigmoid)

Relu2 = ReLU()(linear3)

add = CAddTable(relu1, relu2)

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

Support in 0.2.0

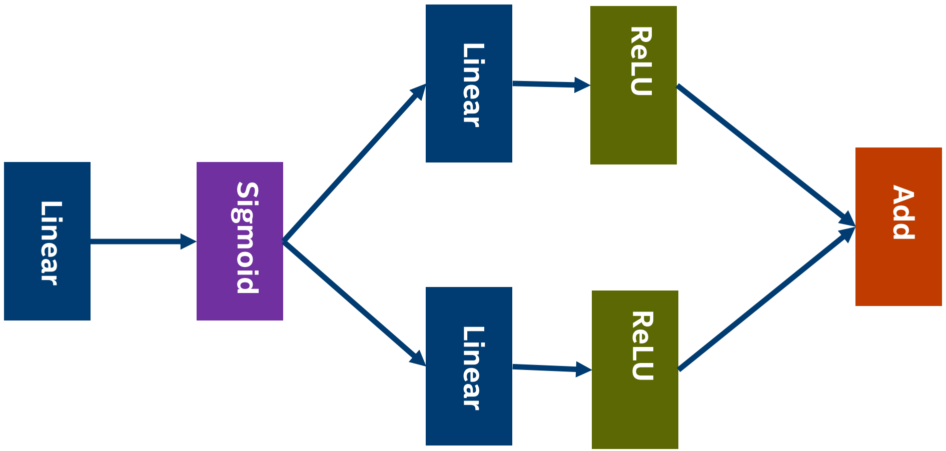


Functional Style Model Definition

Scala

```scala
val linear1 = Linear(4, 3).inputs()

val sigmoid = Sigmoid().inputs(linear1)

val linear2 = Linear(3, 2).inputs(sigmoid)

val relu1 = ReLU().inputs(linear2)

val linear3 = Linear(3, 2).inputs(sigmoid)

val Relu2 = ReLU().inputs(linear3)

val add = CAddTable().inputs(relu1, relu2)

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

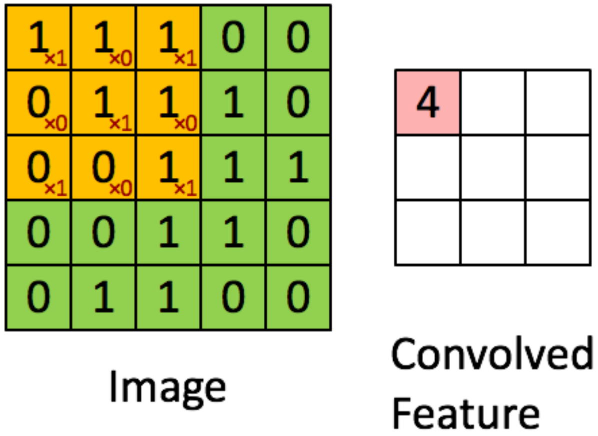
Support in 0.2.0

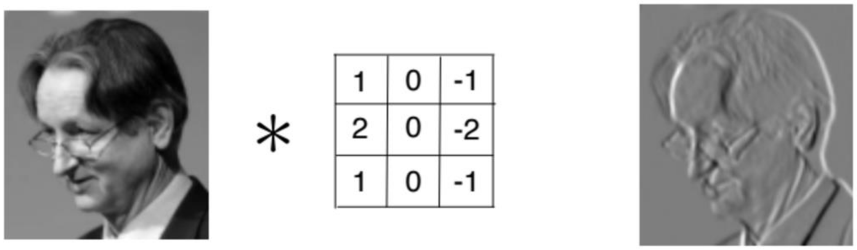


Convolution neural networks

Convolution Layers

- Widely used in image related models (not limited)





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

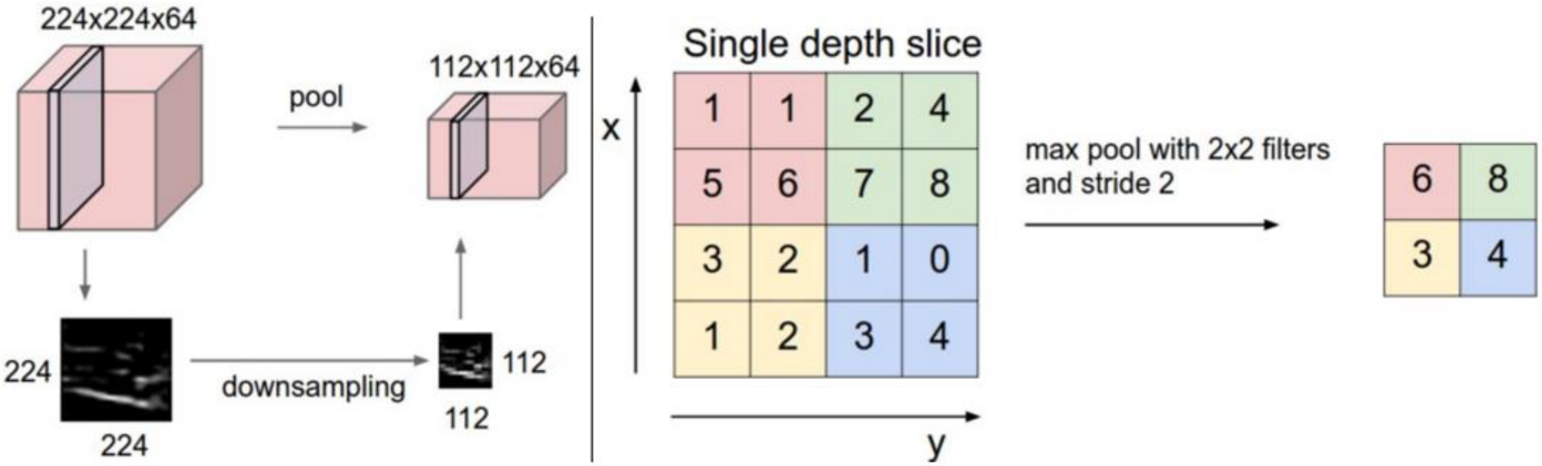
Convolution neural networks

Convolution in BigDL

- SpatialConvolution
- SpatialConvolutionMap
- SpatialDilatedConvolution
- SpatialFullConvolution
- SpatialShareConvolution
- VolumetricConvolution

Convolution neural networks

Pooling



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

Convolution neural networks

Pooling in BigDL

- SpatialAveragePooling
- SpatialMaxPooling
- VolumetricMaxPooling

Convolution neural networks

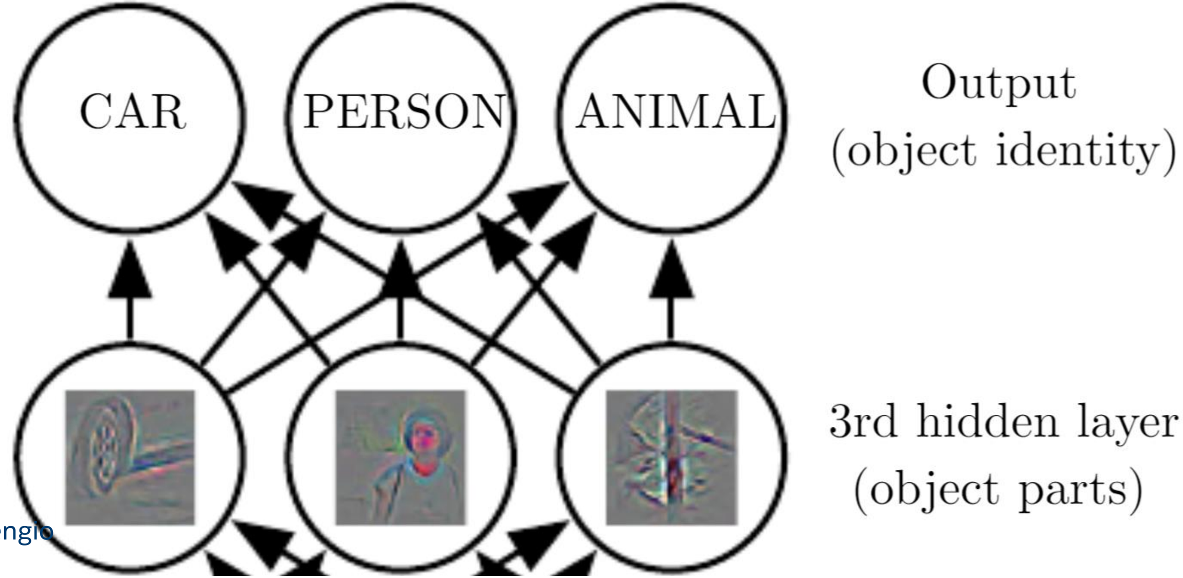
Common CNN models

- LeNet
- AlexNet
- Inception
- ResNet
- SSD
- FasterRCNN

Why so many layers

Object part -> identification

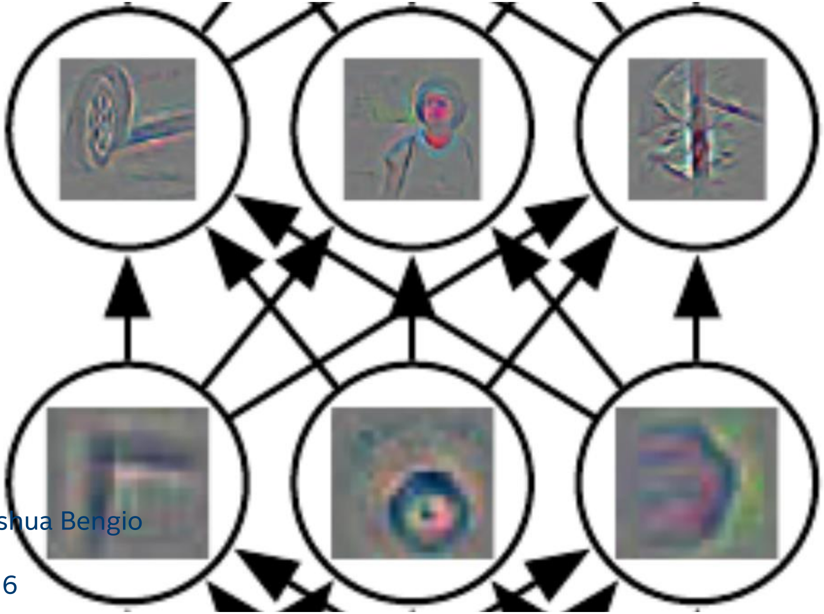
???Hard



Deep Learning, Ian Goodfellow and Yoshua Bengio and Aaron Courville, MIT Press, http://www.deeplearningbook.org, 2016

Why so many layers

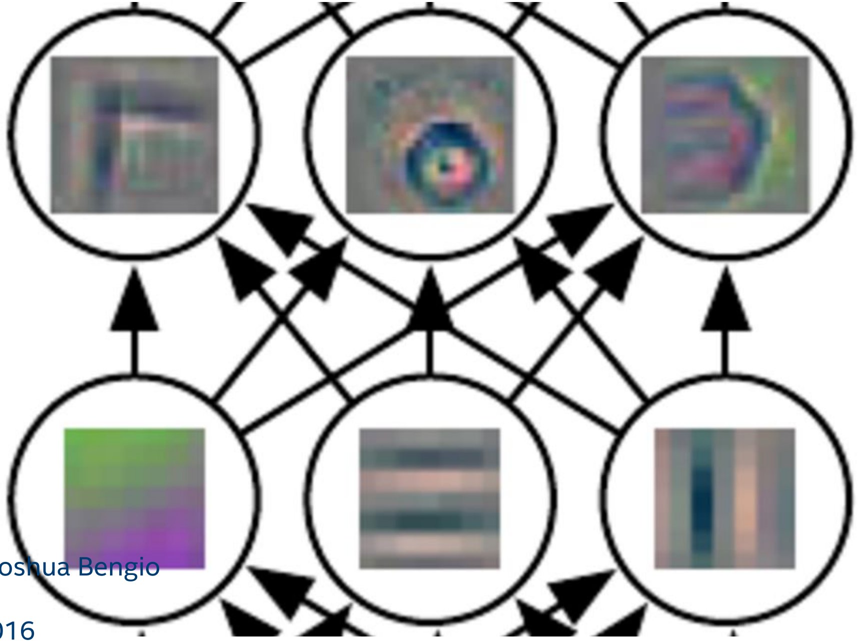
Corners and contours -> object part



Deep Learning, Ian Goodfellow and Yoshua Bengio and Aaron Courville, MIT Press, http://www.deeplearningbook.org, 2016

Why so many layers

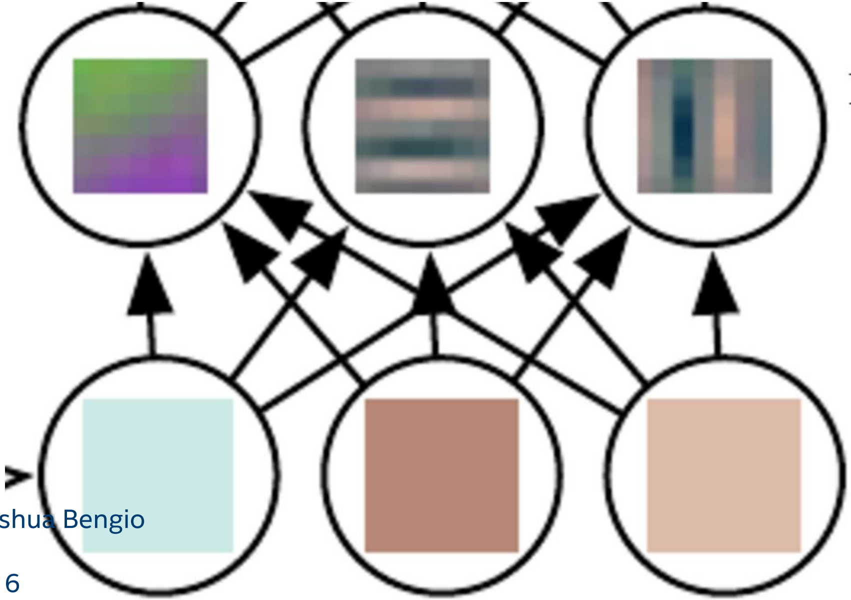
Edges -> corners and contours



Deep Learning, Ian Goodfellow and Yoshua Bengio
and Aaron Courville, MIT Press,
http://www.deeplearningbook.org, 2016

Why so many layers

Pixels-> Edges



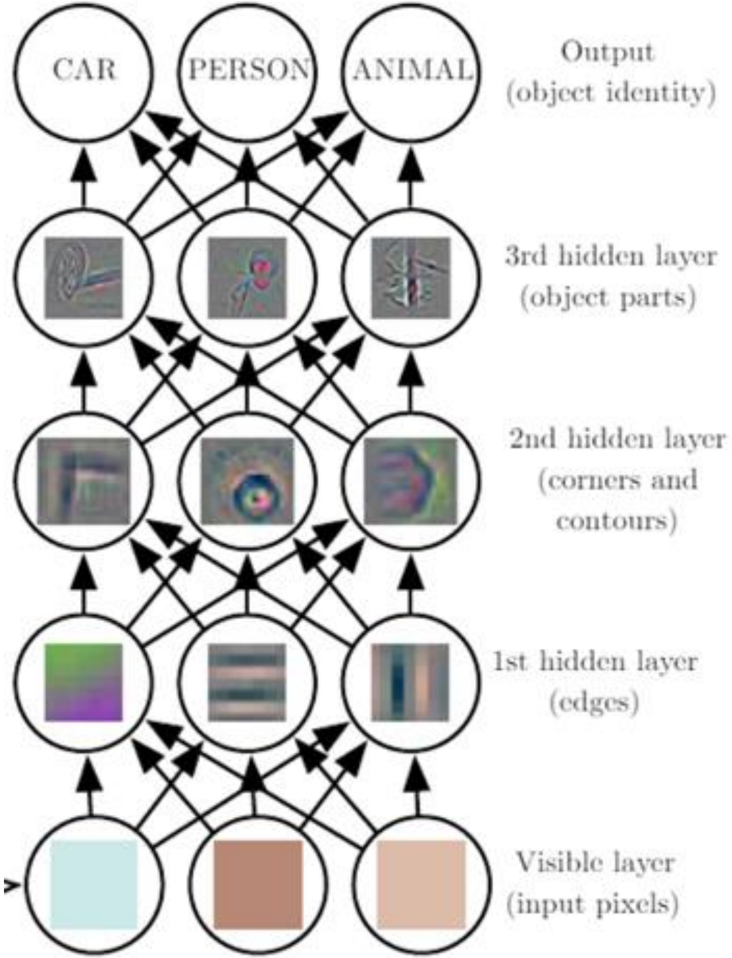
Deep Learning, Ian Goodfellow and Yoshua Bengio
and Aaron Courville, MIT Press,
http://www.deeplearningbook.org, 2016

Why so many layers

- Multi-level representation
  - Decompose complex object into simpler objects
  - Each layer represent different level of concept

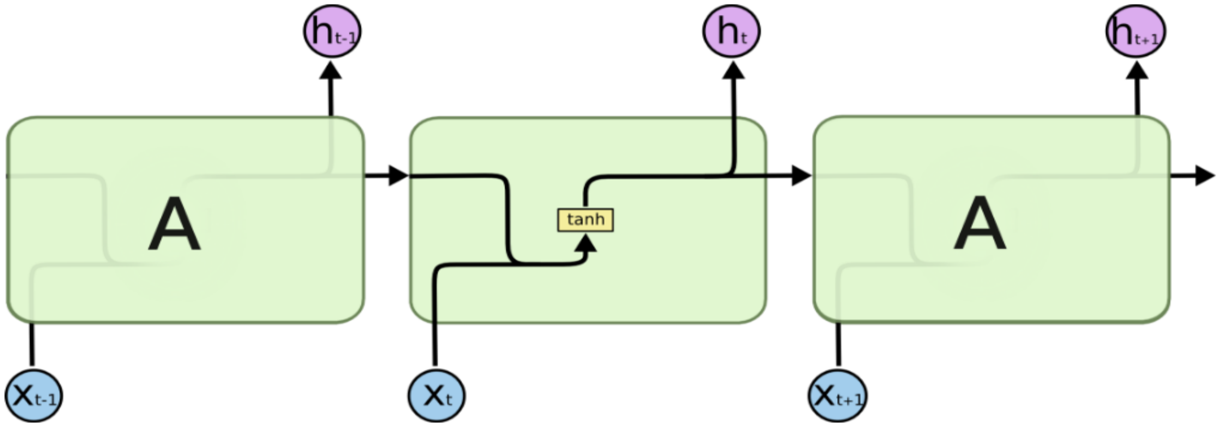
It comes with a price

Deep Learning, Ian Goodfellow and Yoshua Bengio and Aaron Courville, MIT Press, http://www.deeplearningbook.org, 2016



Recurrent Models in BigDL

RNN



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

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

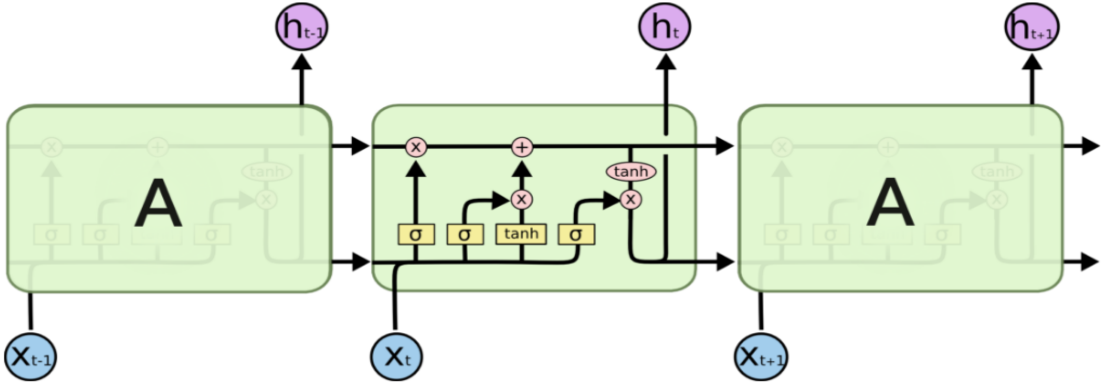
Recurrent Models in BigDL

RNN

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

Recurrent Models in BigDL

LSTM



The repeating module in an LSTM contains four interacting layers.

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

Recurrent Models in BigDL

LSTM

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

GRU

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

After define the model

- Pump data into your model
- Train Model
- Save Model
- Load / use Model

PREPARE DATA

Data preprocess

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

1. They need to be convert to tensors
2. 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

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

Data preprocess

```
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
```

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

Tensor

Data are converted to tensors

- 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)
  )
)
```

Tensor

Tensor:

- 1D vector: a word vector, feature vector
- 2D matrix: a gray image, a sentence
- 3D: a RGB image, a batch of sentence
- 4D: a 3D image, a batch of image
- 5D: a batch of 3D image

Tensor in neural network

Layer inputs/outputs are tensor or sequence of tensors

Scala(type of Activity):

- Table: T(Tensor1, Tensor3,Tensor3…)
- Tensor

Python:

- [ndarray1, ndarray2, ndarray3…]
- ndarray

Tensor in neural network

Python

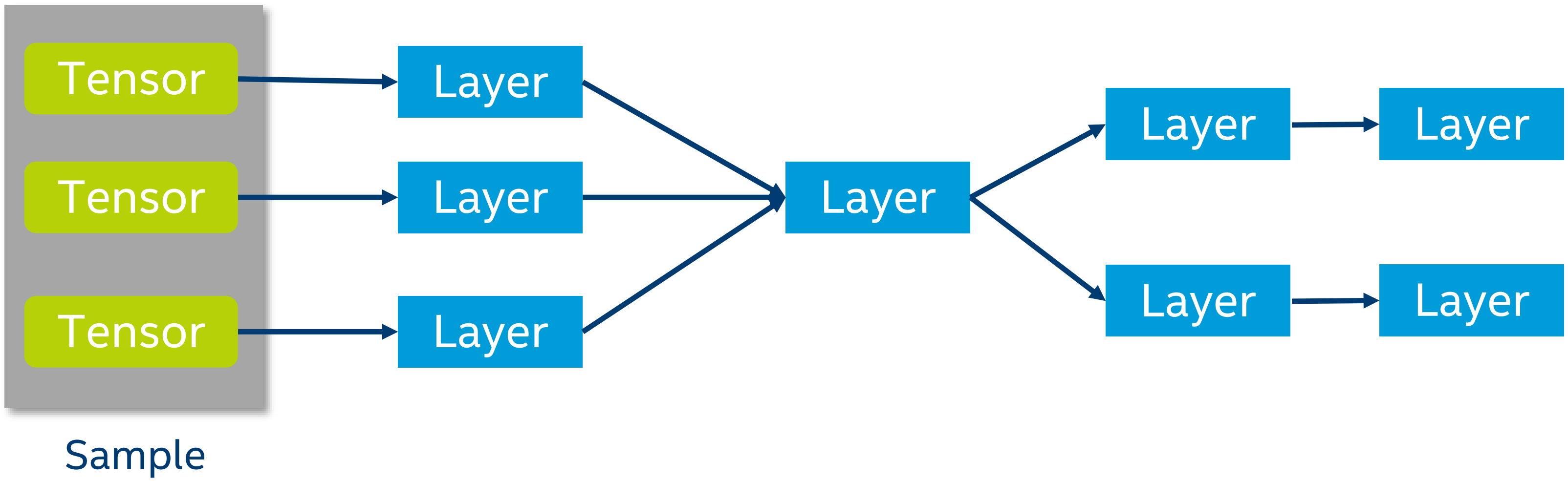
```python
from bigdl.nn.layer import CMinTable
import numpy as np

layer = CMinTable()
layer.forward([
  np.array([1.0, 5.0, 2.0]),
  np.array([3.0, 4.0, -1.0]),
  np.array([5.0, 7.0, -5.0])
])

layer.backward([
  np.array([1.0, 5.0, 2.0]),
  np.array([3.0, 4.0, -1.0]),
  np.array([5.0, 7.0, -5.0])
], np.array([0.1, 0.2, 0.3]))
```

Scala

```scala
import com.intel.analytics.bigdl.nn._
import com.intel.analytics.bigdl.utils.T
import com.intel.analytics.bigdl.tensor.Tensor

val layer = CMinTable[Float]()
layer.forward(T(
  Tensor[Float](T(1.0f, 5.0f, 2.0f)),
  Tensor[Float](T(3.0f, 4.0f, -1.0f)),
  Tensor[Float](T(5.0f, 7.0f, -5.0f))
))
layer.backward(T(
  Tensor[Float](T(1.0f, 5.0f, 2.0f)),
  Tensor[Float](T(3.0f, 4.0f, -1.0f)),
  Tensor[Float](T(5.0f, 7.0f, -5.0f))
), Tensor[Float](T(0.1f, 0.2f, 0.3f)))
```

Sample

Sample is a sequence of tensors



USE MODEL

Load model

From BigDL saved model (Java object file, is being refactor)

```
model = Module.load(path)
```

From Caffe model

```
model = Module.loadCaffe(path, path)  // load weights to BigDL model
model = Module.loadCaffeDynamic(path, path)  // load graph and weights
```

From TensorFlow model

```
model = Module.loadTF(path, inputs, outputs)
```

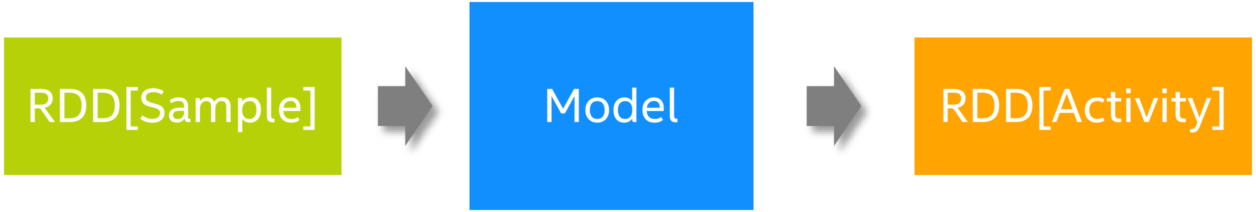
From Torch model

```
model = Module.loadTorch(path)
```

Use model

val input : RDD[Sample] = …

model.predict(input) // result is also a RDD, each record is one tensor or multiple tensors

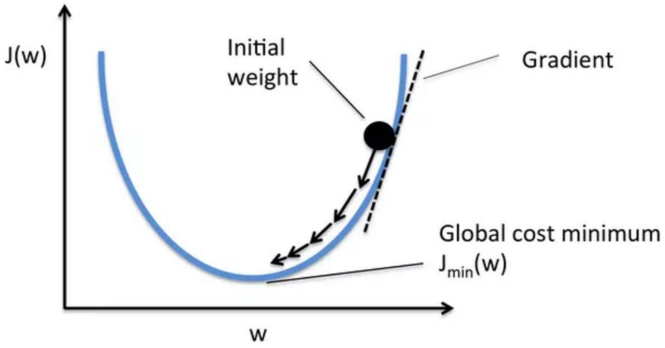


TRAIN MODEL

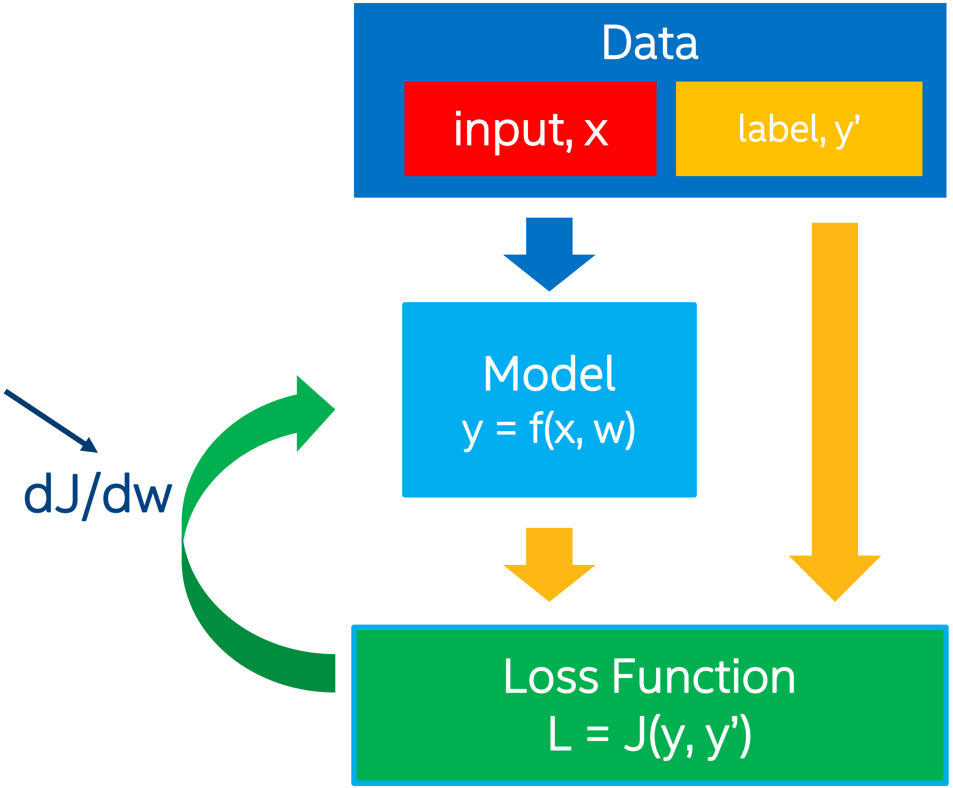
Train your model

- **Model (covered)**
- **Data (covered)**
- **Loss function**
- **Batch size**

Train your model



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



Supervised learning

Forward/Backward

An example

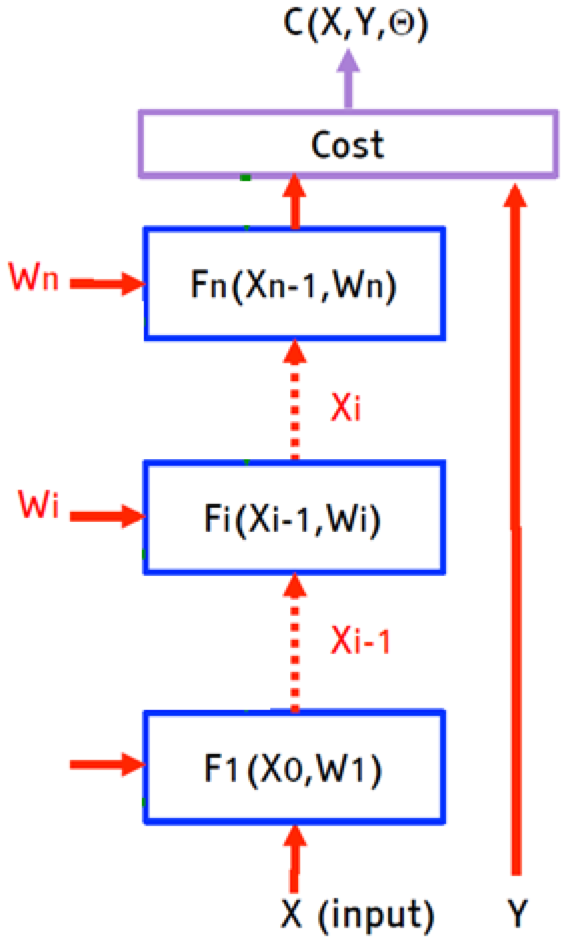
```scala
val layer = CMinTable[Float]()
layer.forward(T(
  Tensor[Float](T(1.0f, 5.0f, 2.0f)),
  Tensor[Float](T(3.0f, 4.0f, -1.0f)),
  Tensor[Float](T(5.0f, 7.0f, -5.0f))
))
layer.backward(T(
  Tensor[Float](T(1.0f, 5.0f, 2.0f)),
  Tensor[Float](T(3.0f, 4.0f, -1.0f)),
  Tensor[Float](T(5.0f, 7.0f, -5.0f))
), Tensor[Float](T(0.1f, 0.2f, 0.3f)))
```

```
1.0
4.0
-5.0
[com.intel.analytics.bigdl.tensor.DenseTensor of size 3]
```

```
{
  2: 0.0
     0.2
     0.0
     [com.intel.analytics.bigdl.tensor.DenseTensor of size 3]
  1: 0.1
     0.0
     0.0
     [com.intel.analytics.bigdl.tensor.DenseTensor of size 3]
  3: 0.0
     0.0
     0.3
  [com.intel.analytics.bigdl.tensor.DenseTensor of size 3]
}
```

Let's look closer to the model

Forward to get the output



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

Let's look closer to the model

Backpropagation to get the gradients

- Backprop for the activities

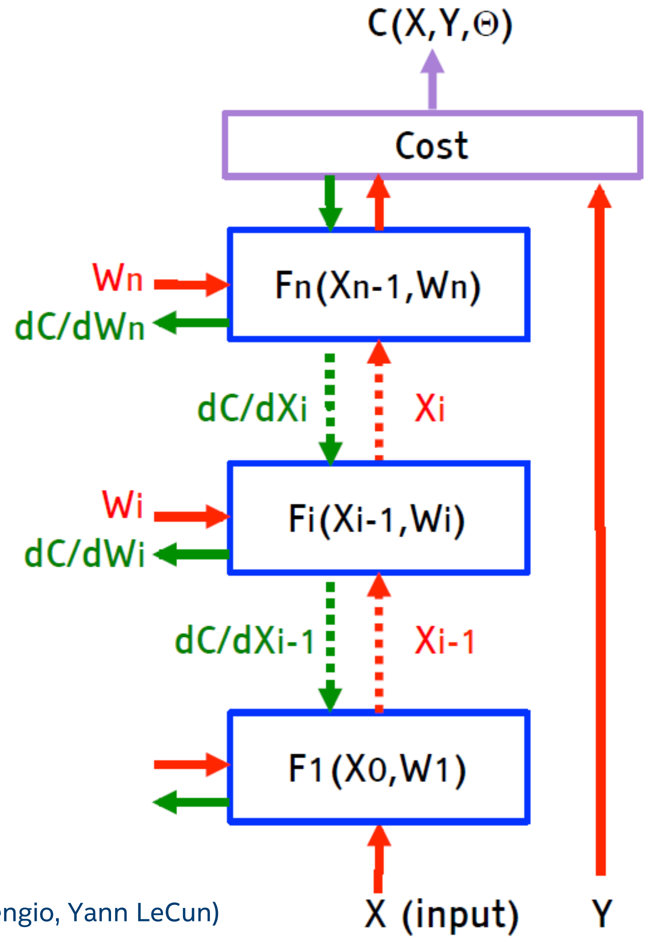
  dC / dXi-1 = dC / dXi * dXi / dXi-1

  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)

Train your model

A very simple example to train Linear on dummy data : y = 0.1 * x[1] + 0.3 * x[2]

```
model = Linear(2, 1)
samples = [
  Sample.from_ndarray(np.array([5, 5]), np.array([2.0])),
  Sample.from_ndarray(np.array([-5, -5]), np.array([-2.0])),
  Sample.from_ndarray(np.array([-2, 5]), np.array([1.3])),
  Sample.from_ndarray(np.array([-5, 2]), np.array([0.1])),
  Sample.from_ndarray(np.array([5, -2]), np.array([-0.1])),
  Sample.from_ndarray(np.array([2, -5]), np.array([-1.3]))
]
train_data = sc.parallelize(samples, 1)
```

Train your model

```
init_engine()
optimizer = Optimizer(model, train_data, MSECriterion(), MaxIteration(100), 4)
optimizer.optimize()
model.get_weights()[0]
```

```
array([[ 0.11578175,  0.28315681]], dtype=float32)
```

Loss functions

BigDL support 23 loss function:

- AbsCriterion
- BCECriterion
- ClassNLLCriterion
- CrossEntropyCriterion
- DiceCoefficientCriterion
- MSECriterion
- ……

Optimization Algorithms

The default optimization algorithm is SGD, BigDL also support

- Adadelta
- Adagrad
- Adam
- Adamax
- RMSprop

Optimization Algorithms

Change optimization algorithms

Python ：

```
# Python need to define in the constructor
optimizer = Optimizer(model, train_data, MSECriterion(), MaxIteration(100), 4, optim_method = Adam())
```

Scala:

```
// The define is SGD
optimizer.setOptimMethod(new Adam())
```

Something about the optimization

- Choose the hyper-parameter of the optimization algorithm carefully
- Hyper-parameter need to adjust when batch size change
- Async SGD/ Parallel SGD often can't get the final performance as well as sync SGD on dense data, but do well on sparse data
- Avoid to use Map/Reduce in distributed model training, use parameter server like technology
- BigDL use BlockManager as Parameter server, a P2P all-reduce algorithm to sync the parameter
- Preprocess data and sync parameter in parallel

When to end the training

Python

```
# Python need to define in the constructor
optimizer = Optimizer(model, train_data, MSECriterion(), MaxIteration(100), 4)
```

Scala

```
// The define endWhen in scala is 100 iterations
optimizer.setEndWhen(Trigger.maxEpoch(10))  // Change to 10 epoch
```

Validate model in training

Model may perform well on training data, but perform poor on a separated dataset, a.k.a, overfitting

Monitor this in training

```
optimizer.setValidation(trigger, testData, validationMethod, batchSize)
```

```
optimizer.set_validation(batch_size, val_rdd, trigger, validationMethod)
```

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

Initialize your model correctly

Set initialization method

Scala

```
layer.setInitMethod(weightInitMethod = Xavier)
```

Python

```
layer.set_init_method(Xavier())
```

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

BigDL support use tensorboard to visualize training process

```
pip install tensorboard==1.0.0a4
```

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()
```

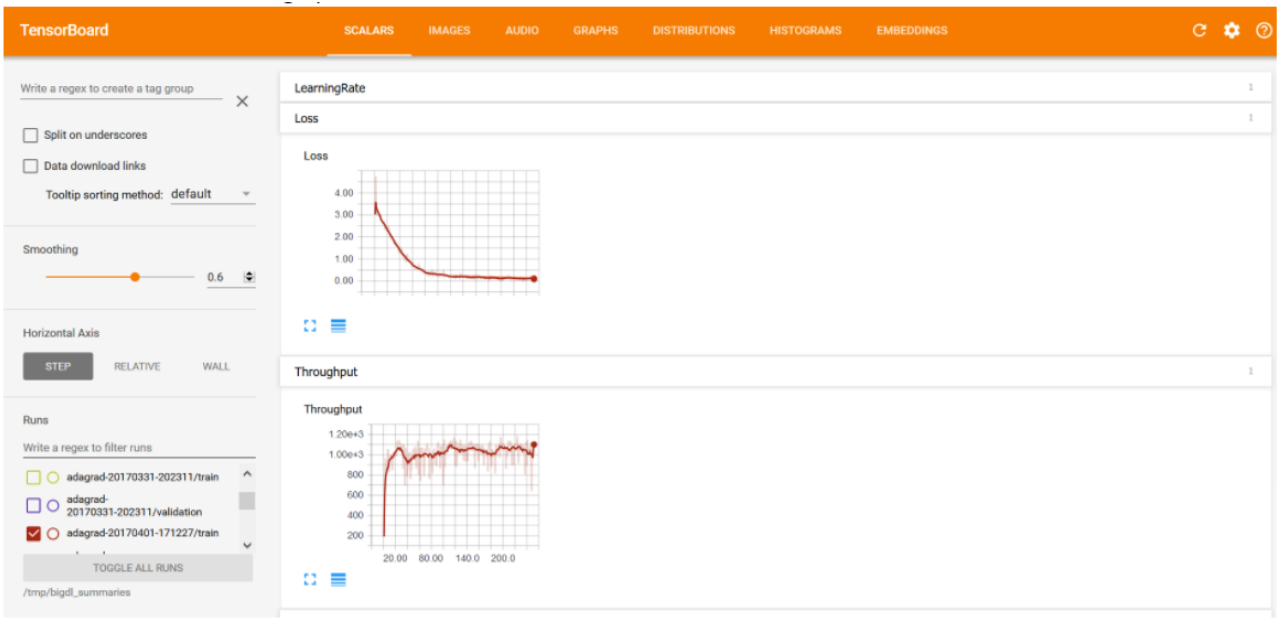
Visualize your training process

Turn on persist training summary, Python

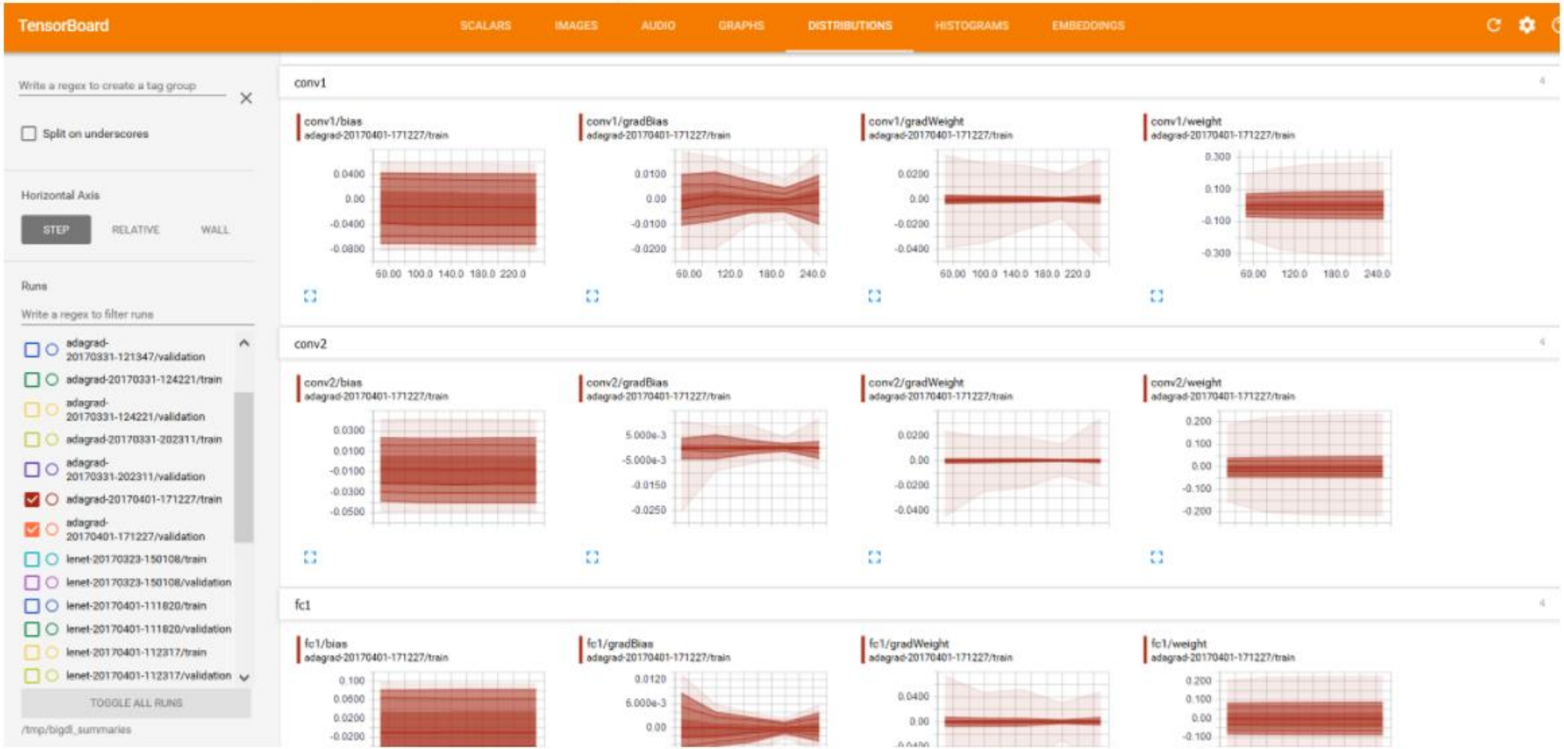
```
optimizer = Optimizer(...)
...
log_dir = 'mylogdir'
app_name = 'myapp'
train_summary = TrainSummary(log_dir=log_dir, app_name=app_name)
train_summary.set_summary_trigger('Parameters', SeveralIteration(20))
val_summary = ValidationSummary(log_dir=log_dir, app_name=app_name)
optimizer.set_train_summary(train_summary)
optimizer.set_val_summary(val_summary)
...
trainedModel = optimizer.optimize()
```

Visualize your training process

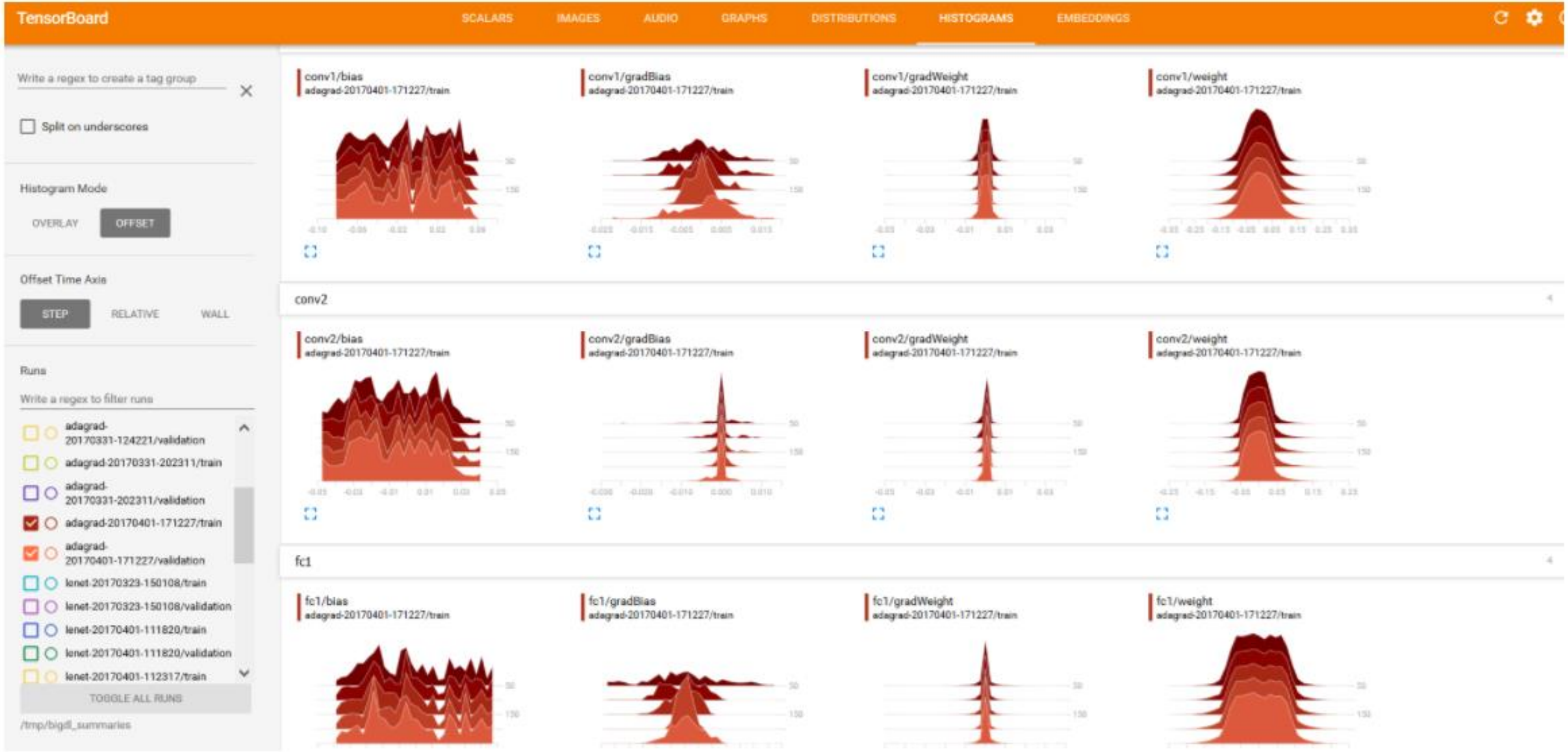
tensorboard --logdir=/tmp/bigdl_summaries



Visualize your training process



Visualize your training process



Integrate with ML Pipeline

BigDL can be easily to integrate to ML Pipeline

- DLEstimator
- DLTransfomer

https://github.com/intel-analytics/BigDL/tree/master/spark/dl/src/main/scala/com/intel/analytics/bigdl/example/MLPipeline

https://github.com/intel-analytics/BigDL/blob/master/spark/dl/src/main/scala/com/intel/analytics/bigdl/example/imageclassification/ImagePredictor.scala

START FROM HERE

Documentation and examples

Wiki

https://github.com/intel-analytics/BigDL/wiki

Step-by-step python notebook tutorial

https://github.com/intel-analytics/BigDL-Tutorials

Documentation and examples

Examples

https://github.com/intel-analytics/BigDL/wiki/Examples

More examples

https://github.com/intel-analytics/analytics-zoo/

Need help?

Send email to

bigdl-user-group@googlegroups.com

Open tickets on

https://github.com/intel-analytics/BigDL/issues

HAND-ON TUTORIALS

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.

intel®

Software