Deep Learning on OpenPower using PowerAI

Workshop at NCSA, UIUC

February 28th 2018

Chekuri S. Choudary, IBM (chekuri.choudary@ibm.com)

Rodrigo Ceron, IBM (Rodrigo.ceron@ibm.com)

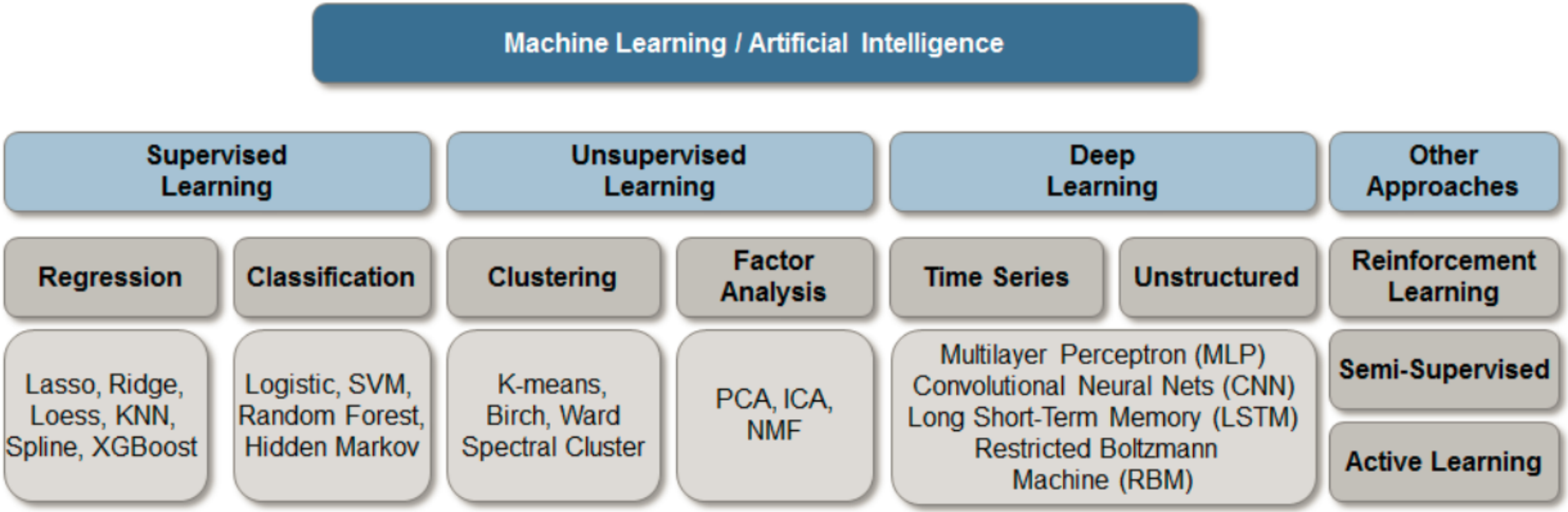
Agenda – AM Session

- 8.00 AM – 8.30 AM (slides 4-10)
  - Review of Machine Learning algorithms, IBM PowerAI, and Nimbix
  - Lab Environment Setup and Demo
  - Introduction to the Structure of a Tensorflow Program
  - Introduction to Jupyter Notebooks
- 8.30 AM – 9.30 AM (slides 11-14)
  - Introduction to Linear Regression
  - Hands-on Exercise on Linear Regression
    - Change Learning Rate, Number of Epochs, and Learning Algorithm
- 9.30 AM – 10.30 AM (slides 15-17)
  - Logistic Regression, Multinomial Logistic Regression/Softmax Regression
  - Lab – Multinomial Logistic Regression Using Tensorflow and MNIST
- 10.30 AM – 11.00 AM
  - Break
- 11.00 AM – 12.00 PM (slides 18-23)
  - Introduction to Fully Connected Neural Network
  - Lab – Fully Connected Neural Net Using Tensorflow

Agenda – PM Session

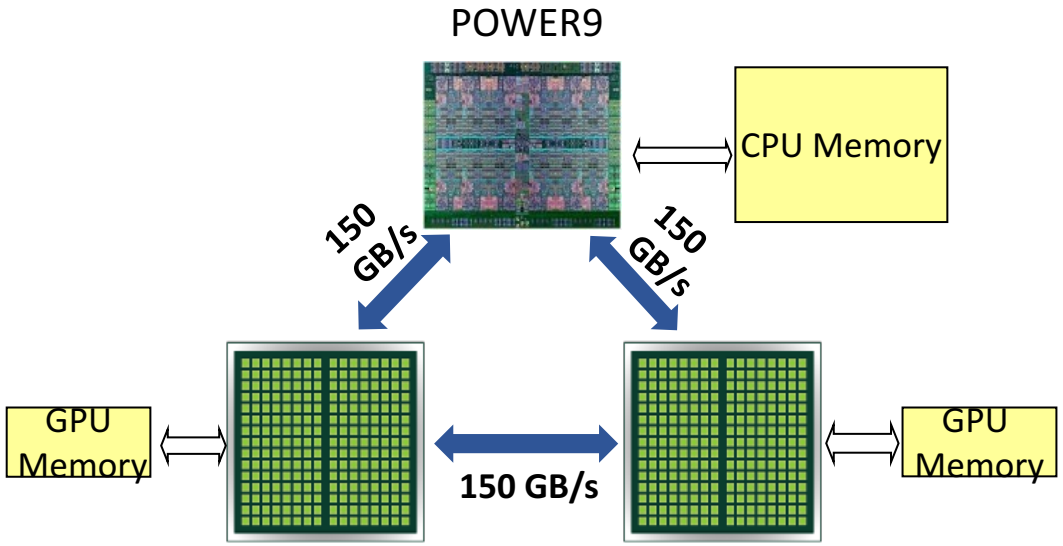
- 12.00 PM – 1.00 PM
  - Lunch
- 1.00 PM – 2.00 PM (slides 22-31)
  - Introduction to Deep Learning
  - Introduction to Convolutional Neural Networks
  - Lab – ImageNet Exercise Using Caffe
  - Lab – Transfer Learning Exercise Using Caffe
- 2.00 PM – 2.30 PM
  - Break
- 2.30 PM – 3.30 PM (slides 32 - 36)
  - Introduction to Recurrent Neural Networks
  - Lab – Exercise Using Tensorflow
- 3.30 PM – 4.00 PM
  - NCSA Review of NCSA Deep Learning Environment, How to Access etc.

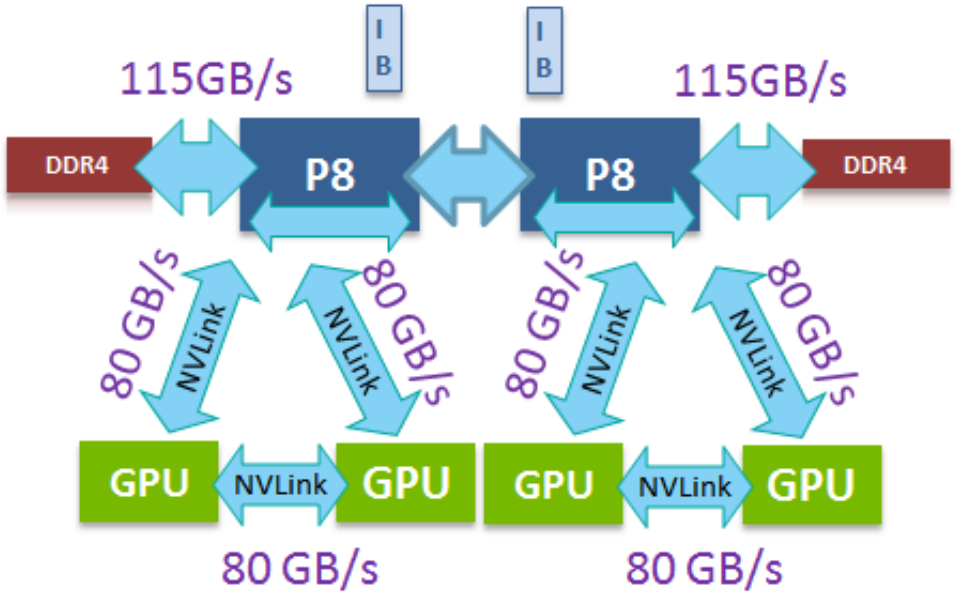
Classification of Machine Learning Techniques



Source JP Morgan QDS

POWER, NVLink and V100 Advantage





POWER9 AC922 Two cores. Four NVIDIA V100 GPU

- Shorter training times
- Facilitates distributed deep learning and large model support in IBM PowerAI

```
In [7]: # CPU mode
        net.forward()  # call once for allocation
        %timeit net.forward()

1 loop, best of 3: 7.22 s per loop
```

That's a while, even for a batch size of 50 images. Let's switch to GPU mode.

```
In [8]: # GPU mode
        caffe.set_device(0)
        caffe.set_mode_gpu()
        net.forward()  # call once for allocation
        %timeit net.forward()

10 loops, best of 3: 56.4 ms per loop
```

PowerAI Platform

Deep Learning Frameworks

- Caffe
- NVCaffe
- IBMCaffe
- Torch
- TensorFlow
- Chainer
- Theano

Supporting Libraries

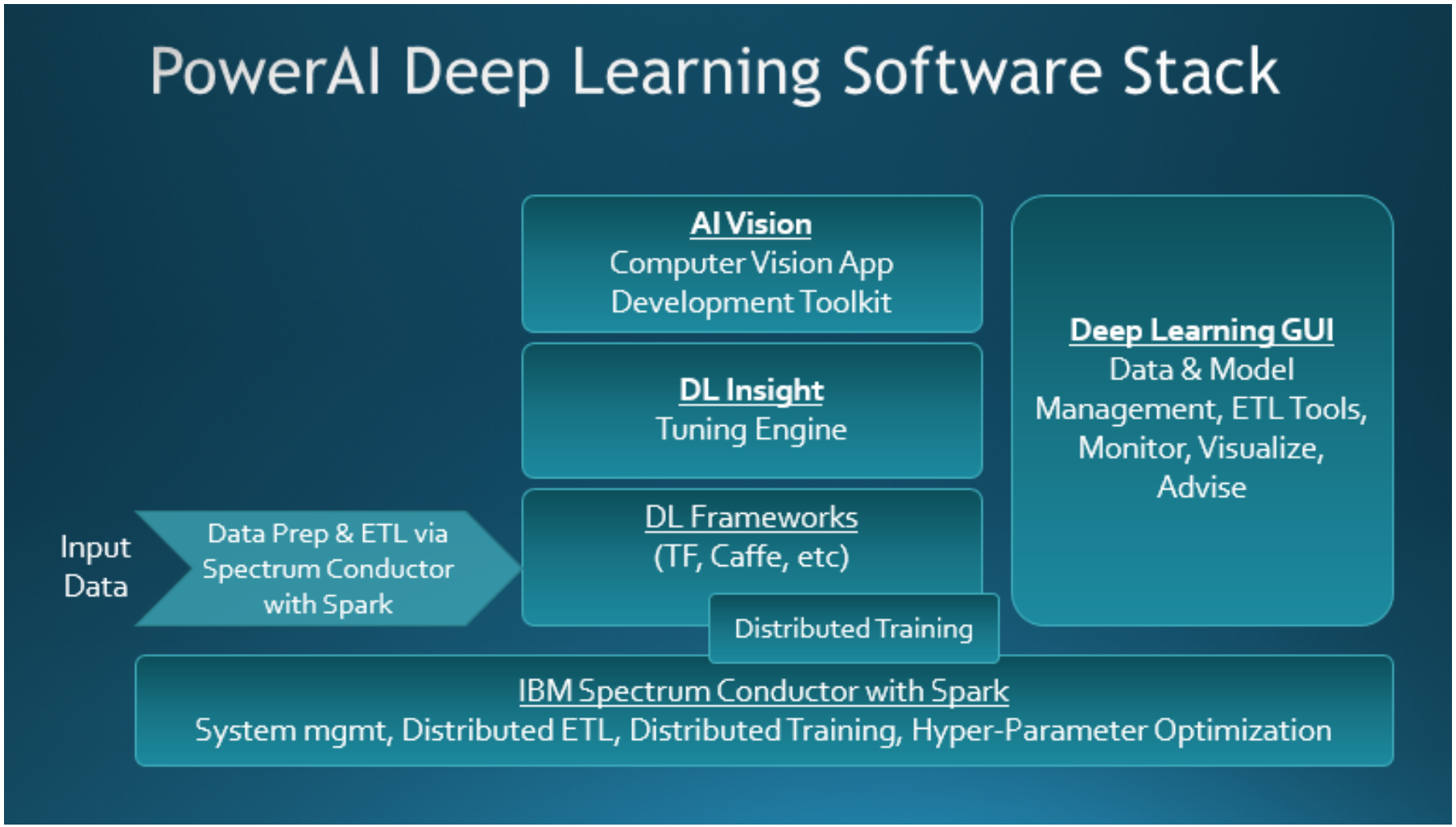
- OpenBLAS
- Bazel
- Distributed Frameworks
- NCCL
- DIGITS

Accelerated Servers and Infrastructure for Scaling

Cluster of NVLink Servers

Spectrum Scale: High-Speed Parallel File System

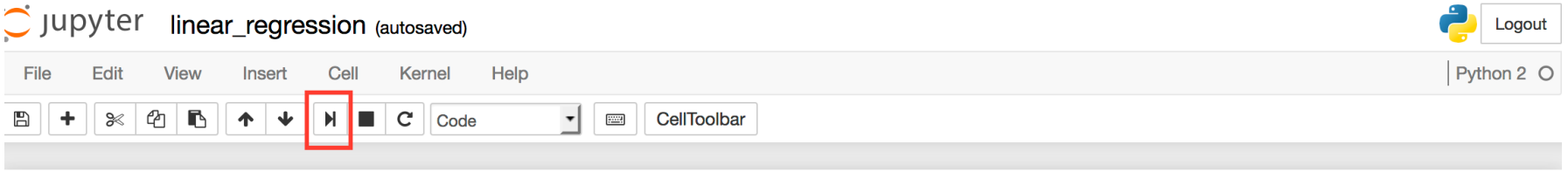
Scale to Cloud


PowerAI Deep Learning Software Stack
AI Vision
Computer Vision App Development Toolkit
DL Insight
Tuning Engine
DL Frameworks
(TF, Caffe, etc)
Distributed Training
Deep Learning GUI
Data & Model Management, ETL Tools, Monitor, Visualize, Advise
Input Data
Data Prep & ETL via Spectrum Conductor with Spark
IBM Spectrum Conductor with Spark
System mgmt, Distributed ETL, Distributed Training, Hyper-Parameter Optimization

Objectives

- Introduce the foundations of deep learning
- Give an overview of state-of-the-art deep learning technologies
- Demonstrate the benefits of leveraging IBM deep learning technology offerings (Power processors coupled with NVLink and PowerAI) for research, teaching, and course projects
- Targeted audience include software developers, faculty, research scientists, postdocs, graduate/undergraduate students across various scientific disciplines

Jupyter Notebooks



In [1]:
```
# A linear regression learning algorithm example using TensorFlow library.

# Author: Aymeric Damien
# Project: https://github.com/aymericdamien/TensorFlow-Examples/
```

In [6]:
```
import tensorflow as tf
import numpy
import matplotlib.pyplot as plt
rng = numpy.random
```

In [17]:
```
# Parameters
learning_rate = 0.01
training_epochs = 1000
display_step = 50
print "learning rate set to: ", learning_rate
print "# of epochs set to: ", training_epochs
print "display sampling set to: ", display_step
```

```
learning rate set to:  0.01
# of epochs set to:  1000
display sampling set to:  50
```

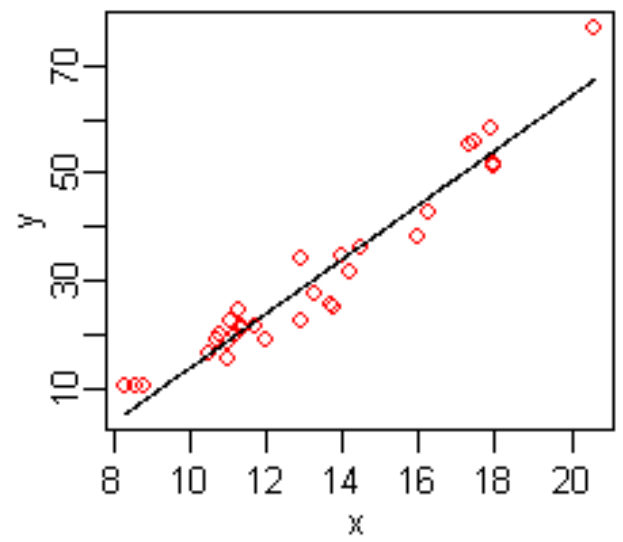
In [8]:
```
# Training Data
train_X = numpy.asarray([3.3,4.4,5.5,6.71,6.93,4.168,9.779,6.182,7.59,2.167,
                         7.042,10.791,5.313,7.997,5.654,9.27,3.1])
train_Y = numpy.asarray([1.7,2.76,2.09,3.19,1.694,1.573,3.366,2.596,2.53,1.221,
                         2.827,3.465,1.65,2.904,2.42,2.94,1.3])
n_samples = train_X.shape[0]
```

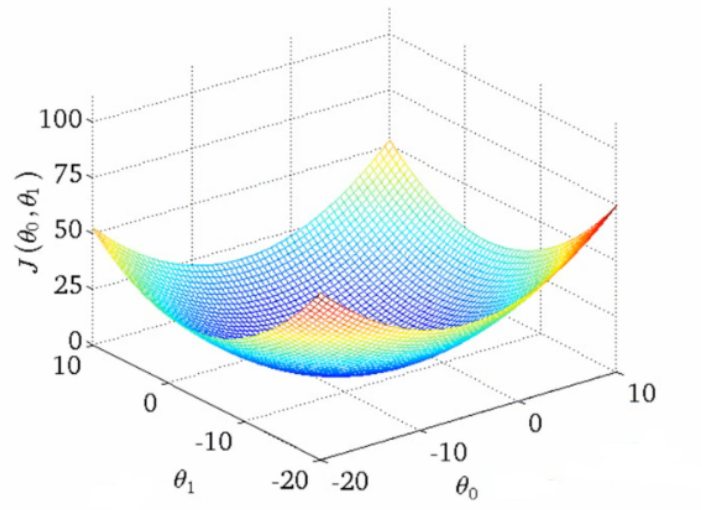
In [9]:
```
# tf Graph Input
X = tf.placeholder("float")
Y = tf.placeholder("float")

# Set model weights
W = tf.Variable(rng.randn(), name="weight")
b = tf.Variable(rng.randn(), name="bias")
```

In [11]:
```
# Construct a linear model
pred = tf.add(tf.multiply(X, W), b)
```

Lab Environment Setup and Demo

Linear Regression





Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

Parameters: θ_0, θ_1

Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

Goal: $\underset{\theta_0,\theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Gradient descent algorithm

repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x^{(i)}$$

}

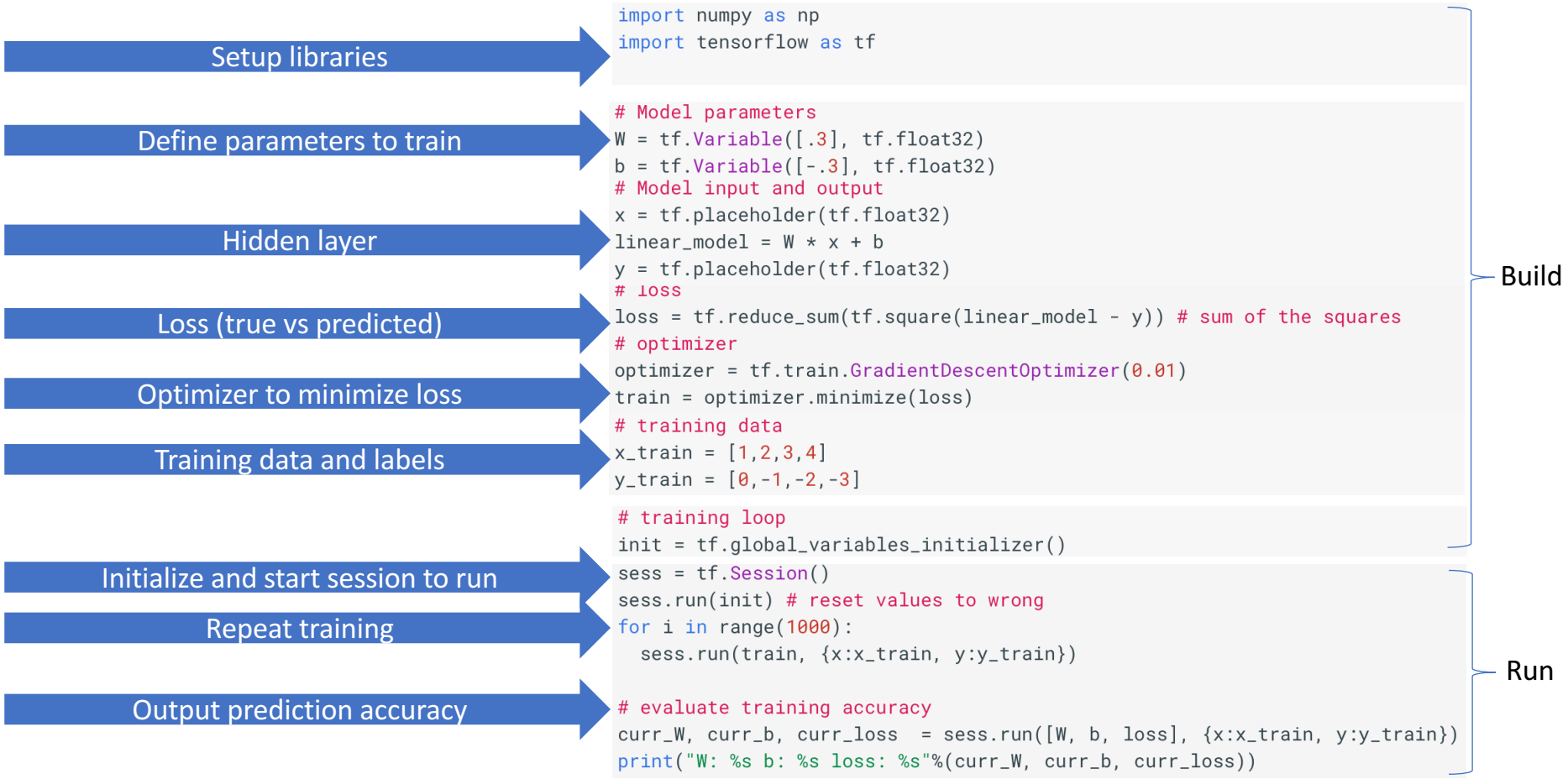
update θ_0 and θ_1 simultaneously

$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$

$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$

$h_\theta(x) = \theta_0 + \theta_1 x$

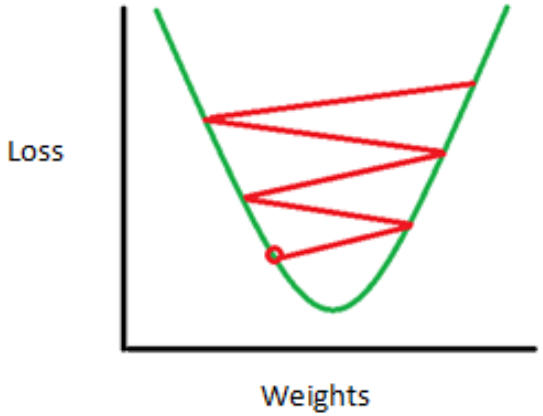
Structure of a Tensorflow Program



Ref: https://www.tensorflow.org/get_started/get_started

Linear Regression Exercise using Tensorflow

- https://github.com/aymericdamien/TensorFlow-Examples/blob/master/notebooks/2_BasicModels/linear_regression.ipynb
- Follow-up
  - Change learning rate with [0.001 0.003 0.01 0.03 0.1 0.3 1]
  - Change # of epochs with [500 1000 1500 2000 2500 3000]

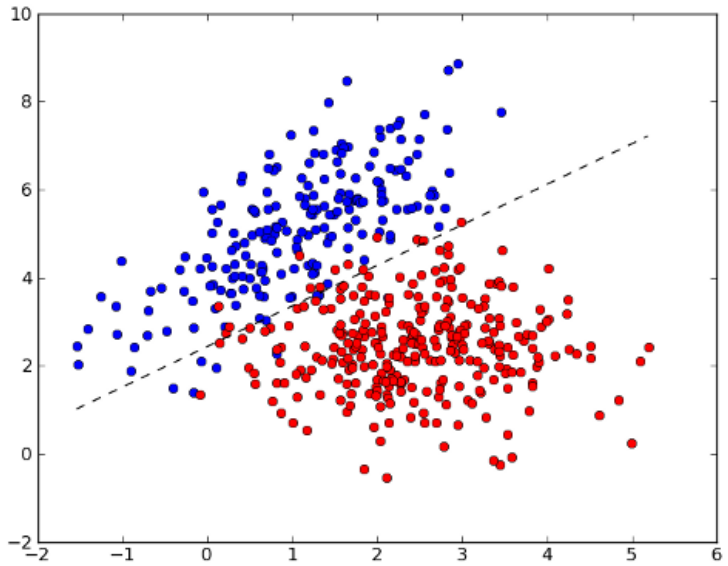


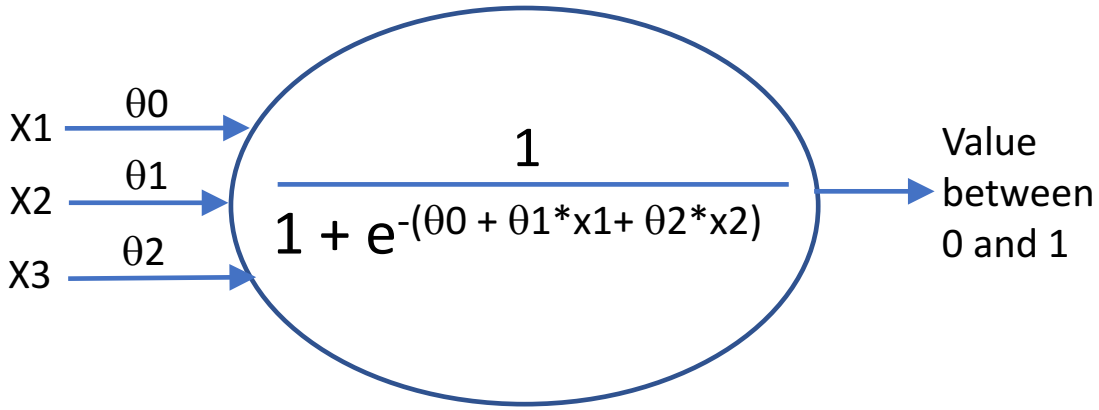
Consequence of a high learning rate where the jumps are too large and we are not able to minimize the loss.

Exploratory Data Analysis

- Trivial but critical to data science process
- Plots
  - Time-series data
  - Histograms
  - Pair-wise scatterplots
- Summary statistics
  - Mean, Median, Mode, Maximum, Minimum, Upper and lower quartiles
- Outlier analysis
- Find missing data

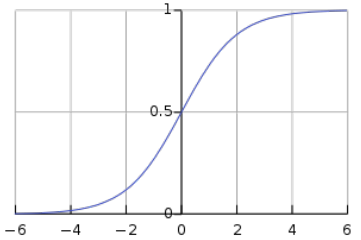
Classification (Logistic Regression)





Hypothesis:

$$\frac{1}{1 + e^{-(\theta_0 + \theta_1 * x_1 + \theta_2 * x_2)}}$$



Cost function:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1 - y^{(i)})\log(1 - h_\theta(x^{(i)}))]$$

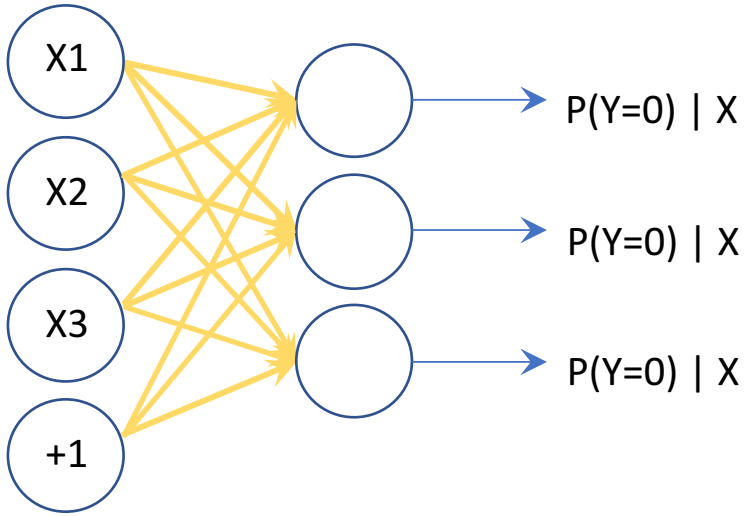
Gradient Descent Algorithm:

$$Repeat\ \{$$

$$\theta_j := \theta_j - \frac{\alpha}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

$$\}$$

Softmax Regression (Multinomial Logistic Regression)



Input Features

Softmax Layer

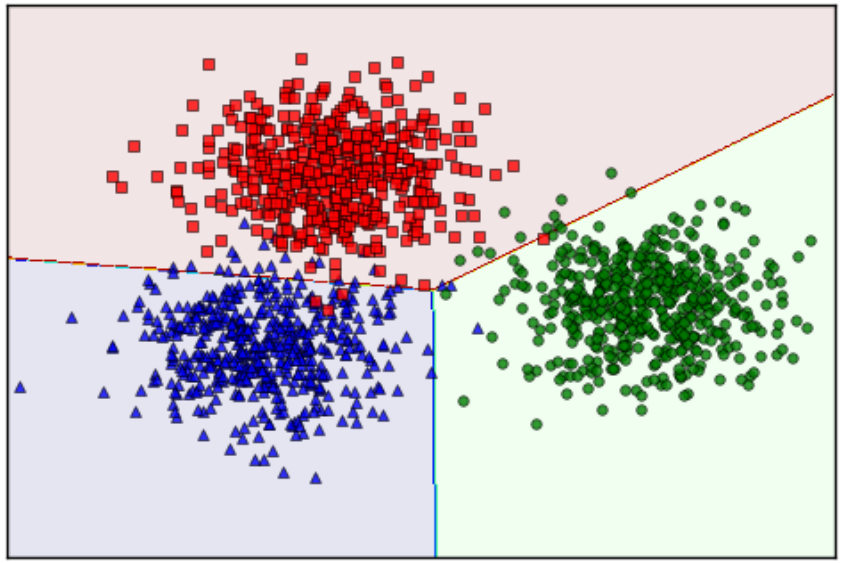
$$P(Y = i|x, W, b) = softmax_i(Wx + b) = \frac{e^{W_i x + b_i}}{\sum_j e^{W_j x + b_j}}$$

$$J(\theta) = -\sum_i y_i ln(\hat{y}_i)$$

Ex:

Computed ($\hat{y}$)	Targets (y)
[0.3, 0.3, 0.4]	[0, 0, 1]

$$J(\theta) = -\sum_i^n [0 * ln(0.3) + 0 * ln(0.3) + 1 * ln(0.4)] = -ln(0.4)$$



Multinomial Logistic Regression using Tensorflow

- https://github.com/aymericdamien/TensorFlow-Examples/blob/master/notebooks/2_BasicModels/logistic_regression.ipynb
- Stochastic Gradient Descent used in previous exercise

    for epoch in 1 : num_epochs:

        for sample in 1:num_samples

            #Run the optimizer for sample

- Mini Batch Gradient Descent used in current exercise

    Initialize batch_size

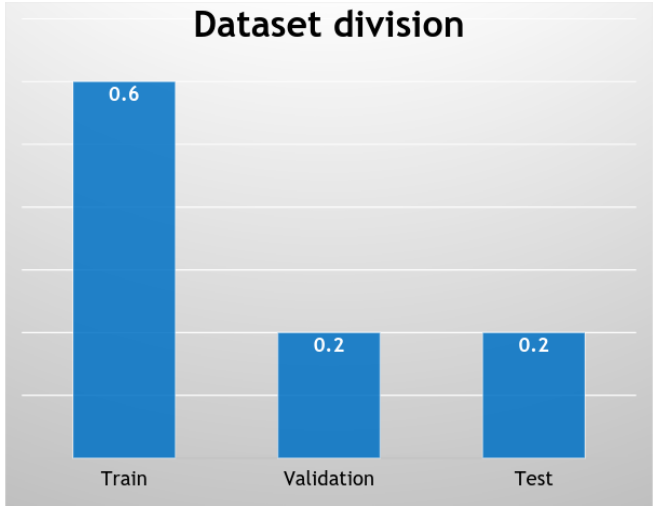
    num_batches = num_samples/batch_size

    for epoch in 1: num_epochs:

        for batch in 1:num_batches

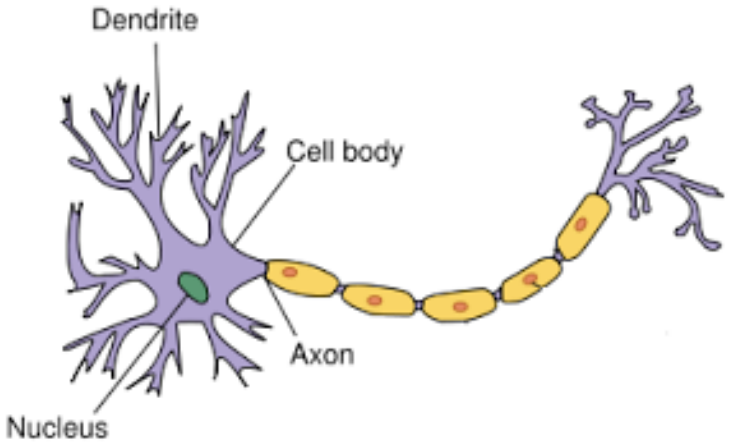
            #Run the optimizer for batch

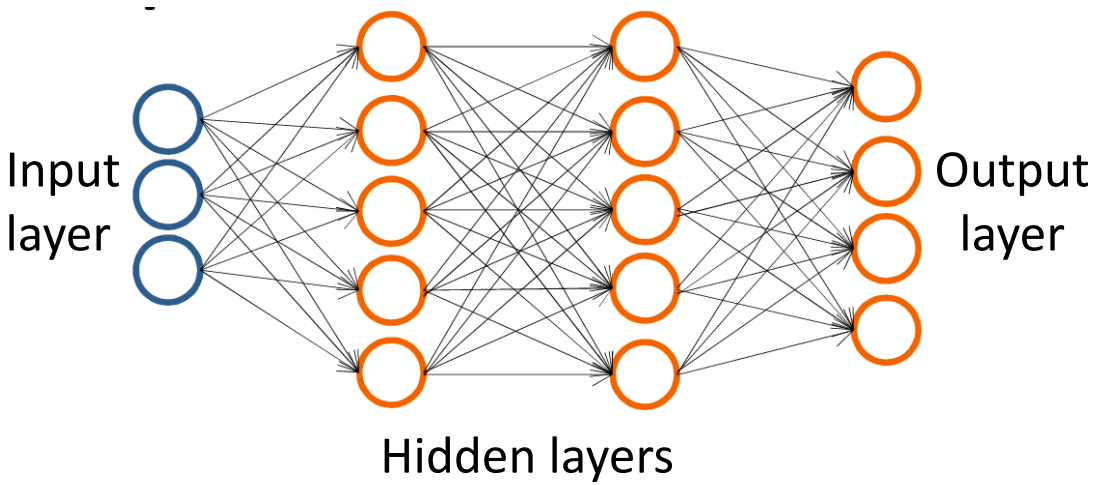
- Follow-up
- Change batch size in [25 50 100 200 400 800]
- Try with batch size 55000, i.e, batch gradient descent
- Try with batch size = 1, i.e, stochastic gradient descent
- Change number of epochs



- Training set:
    - bang on this data all you want
- Test set:
    - rarely (weekly), evaluate progress
- Validation set:
    - periodically during training, check

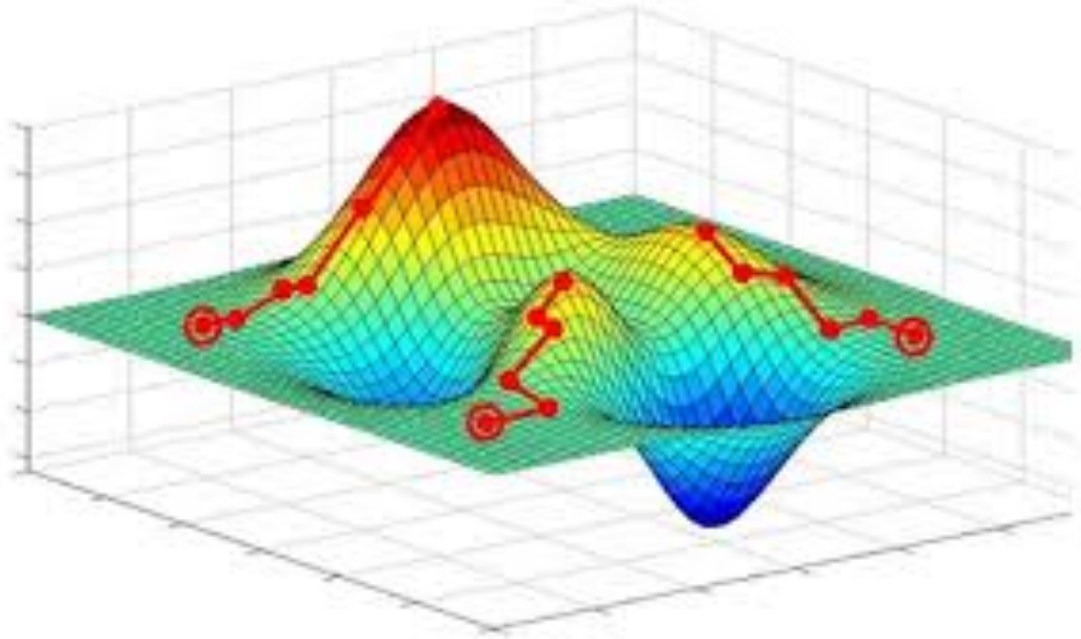
Artificial Neural Networks





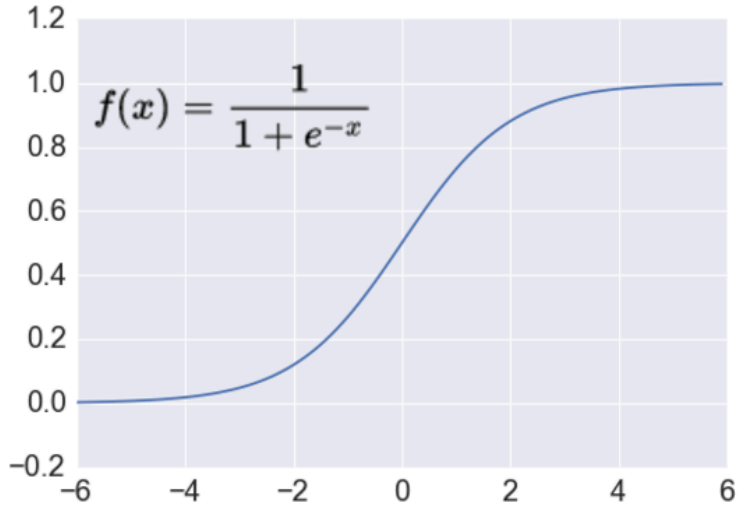
Cost Function:

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[y_k^{(i)}\log((h_\Theta(x^{(i)}))_k) + (1-y_k^{(i)})\log(1-(h_\Theta(x^{(i)}))_k)\right].$$

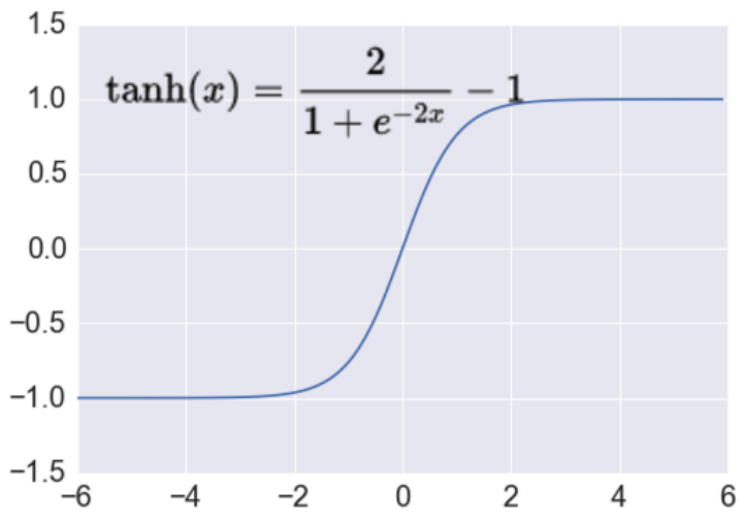


Activation Functions

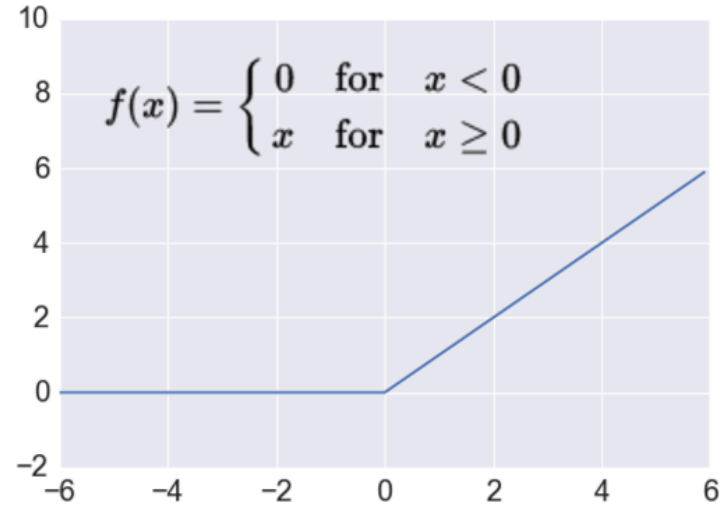
Sigmoid



TanH



ReLU

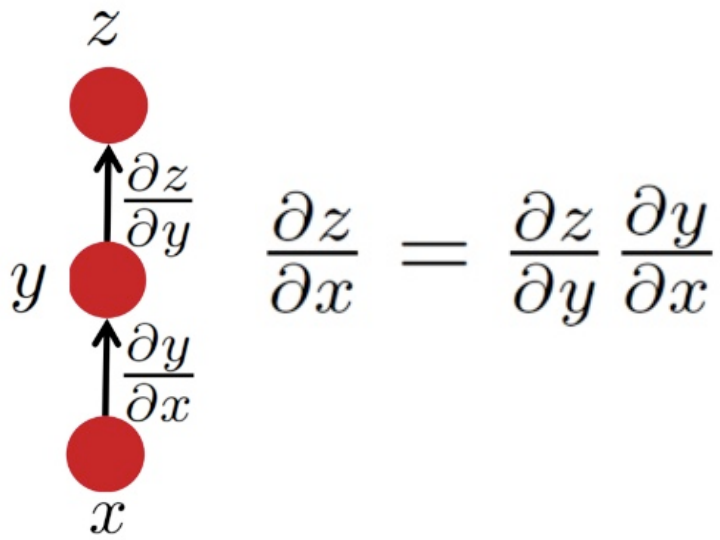


Forward Propagation and Back Propagation

Gradient Descent Iteration:

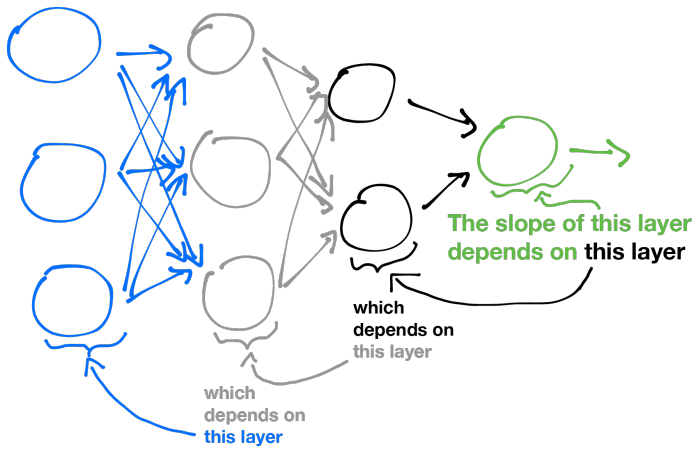
1. Forward Propagation (Calculate the cost using cost function)
2. Backward Propagation (Calculate partial derivatives of cost function w.r.t each parameter)
    - Option 1: Numerical Derivatives
        - Change the weight a little, calculate the change in cost function
        - Computationally intractable
    - **Option 2: Analytical Derivatives**
        - **Use Chain rule and compute analytical derivatives**
        - **Reasonable turnaround times, 1000s of times cheaper**

Chain Rule:

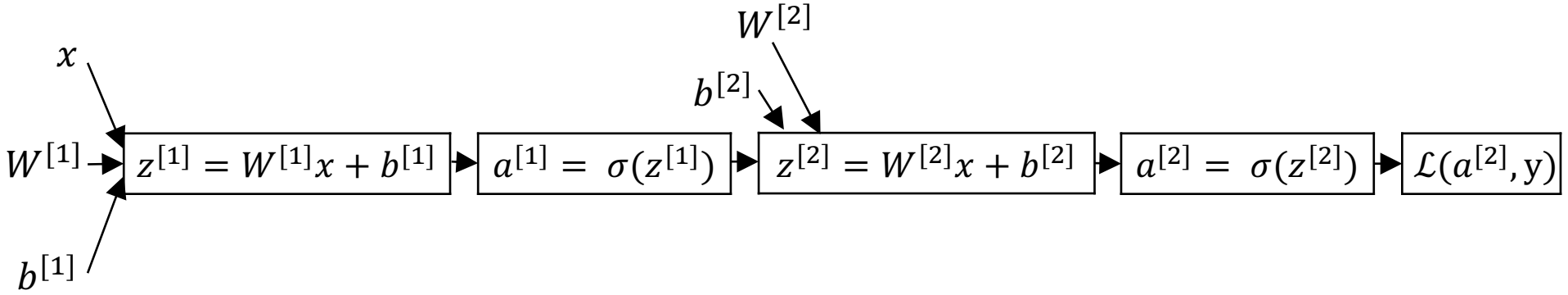


$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

Back Propagation:



Neural Network Gradients



$$dZ^{[2]} = A^{[2]} - Y$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]^T}$$

$$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True)$$

$$dZ^{[1]} = W^{[2]^T} dZ^{[2]} * g^{[1]'}(Z^{[1]})$$

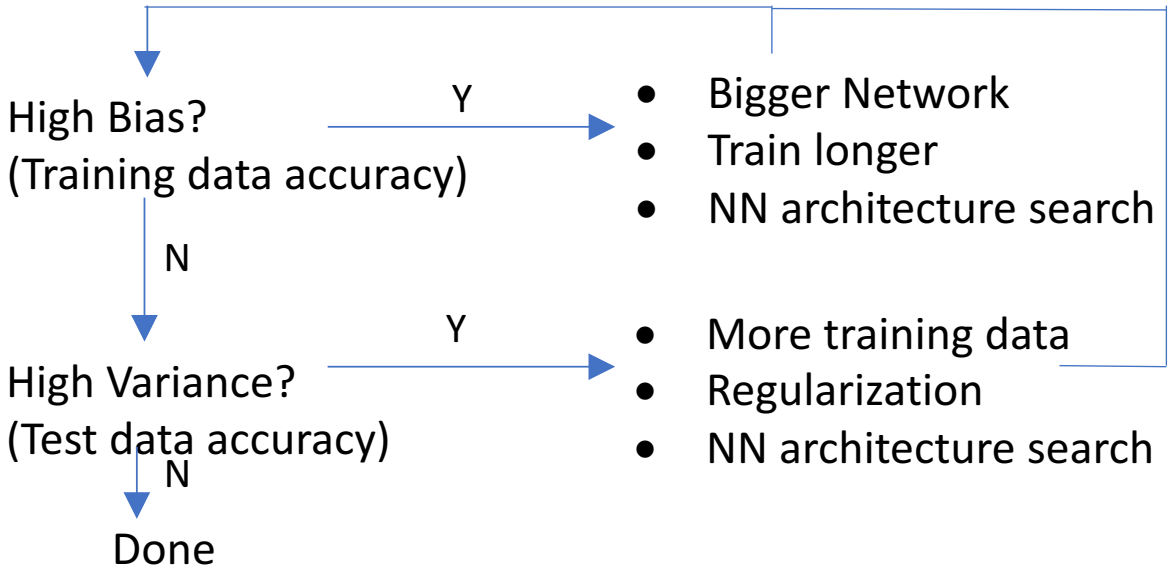
$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$

$$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True)$$

Deep Neural Networks

- Local Minima Unlikely. Saddle points and plateaus more common
  - Training longer generates better results
  - Momentum, Adams and RMSProp algorithms
- Vanishing Gradients and Exploding Gradients
  - Gradient Clipping, Tanh and ReLU, Initialization, GRUs and LSTMs etc.
- Allows addressing Bias and Variance Separately

	High Bias (Underfitting)	High Variance (Overfitting)	High Bias & High Variance	Low Bias & Low Variance
Training Error	15%	5%	15%	5%
Testing Error	15%	15%	30%	5%



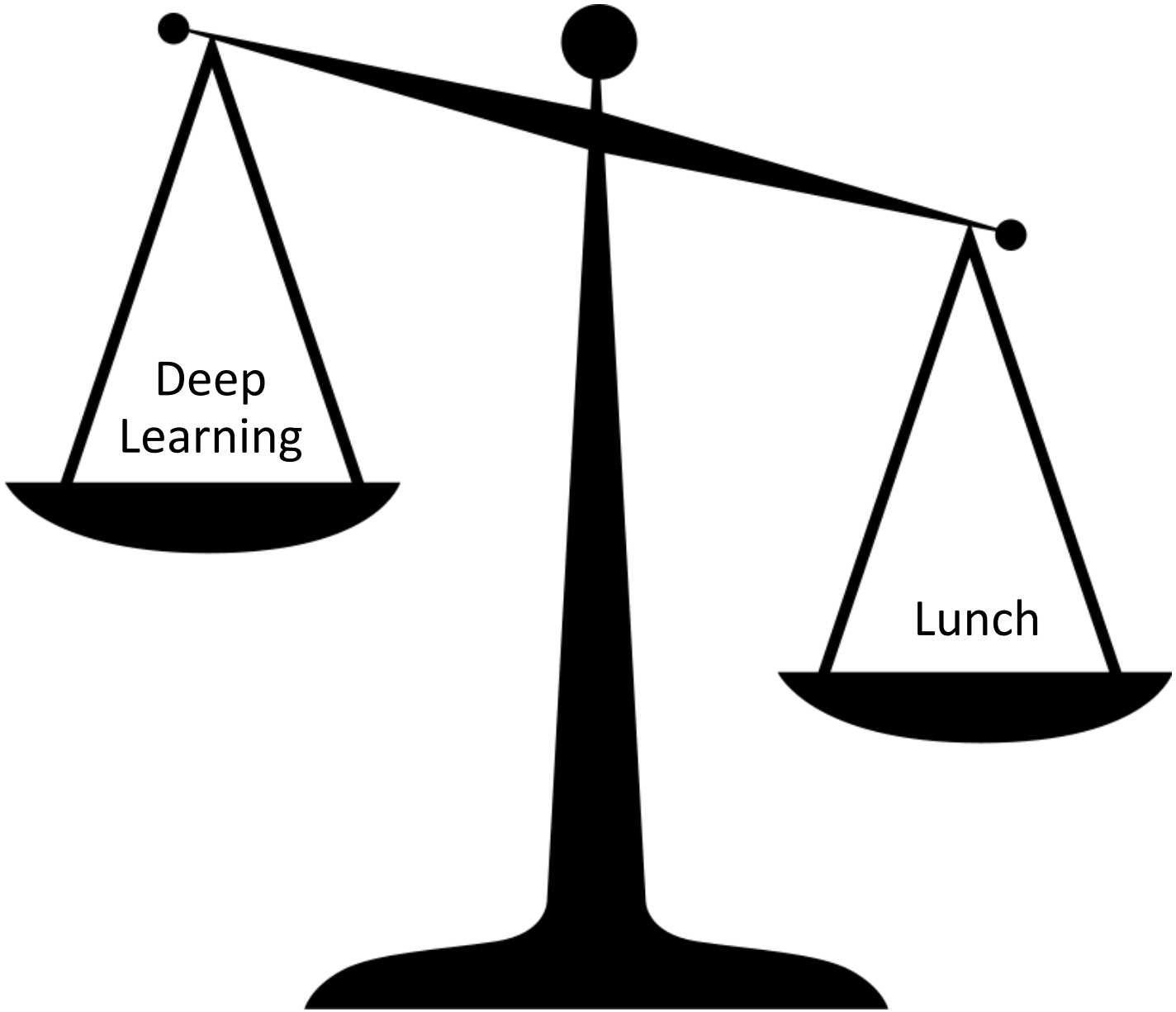
Andrew Ng

Fully Connected Neural Network using Tensorflow

- https://github.com/aymericdamien/TensorFlow-Examples/blob/master/notebooks/3_NeuralNetworks/neural_network_raw.ipynb
- For the first 10 images in mnist.test.images
  - Display Image
  - Compute the prediction, then print the image and prediction by the model
- Change the number of neurons in hidden layers 1 and 2
- Change the activation function
- Change number of layers
- Use GPU
- Change learning algorithm with
  - [GradientDescentOptimizer MomentumOptimizer RMSPropOptimizer AdamOptimizer]

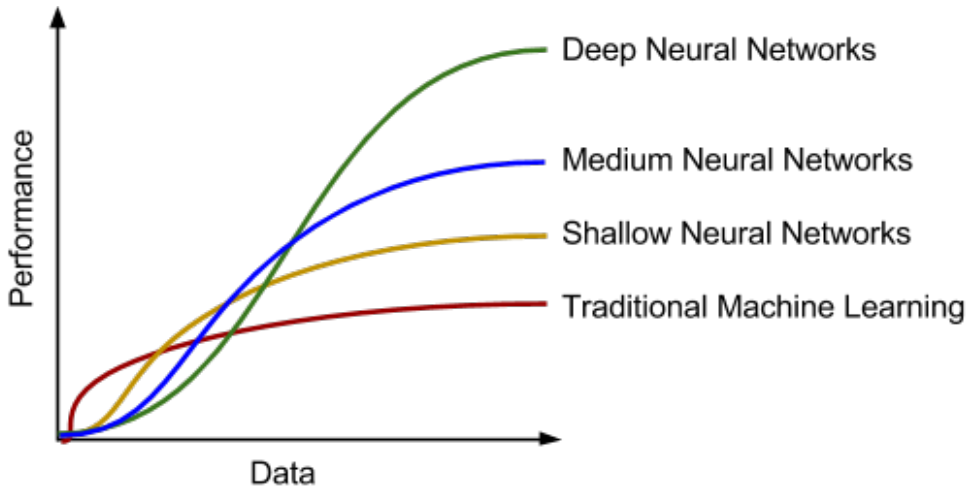
Solution

- for i in range(10):
  - plt.imshow(np.reshape (mnist.test.images[i], [28, 28]), cmap='gray')
  - plt.show()
  - y = tf.nn.softmax(neural_net (np.transpose (mnist.test.images[i]).reshape(1,784)))
  - print('NN predicted', sess.run(tf.argmax (y,1)))

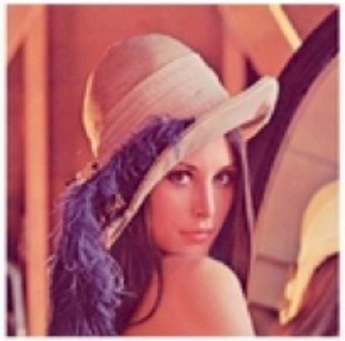

Deep
Learning
Lunch

Deep Learning

- Neural networks with lot more hidden layers (tens or hundreds)
- Types of Deep Neural Networks
  - Convolutional Neural Networks
  - Recurrent Neural Networks
  - Autoencoders
  - Restricted Boltzmann Machines
  - Generative Adversarial Networks
- Why now?
  - Data explosion
  - GPUs and other SIMD architectures
  - Some advancements in neural networks
- Nobody knows why it works but it works
- End-to-end Learning
  - No need for feature engineering



Edge Detection in Image Processing



Original

Sobel (Th = 0.05)

Sobel (Th = 0.15)

Canny (Th = 0.05)

Canny (Th = 0.15)

Roberts (Th = 0.05)

Roberts (Th = 0.15)

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

$$|G| = \sqrt{Gx^2 + Gy^2}$$

End-to-End Deep Learning

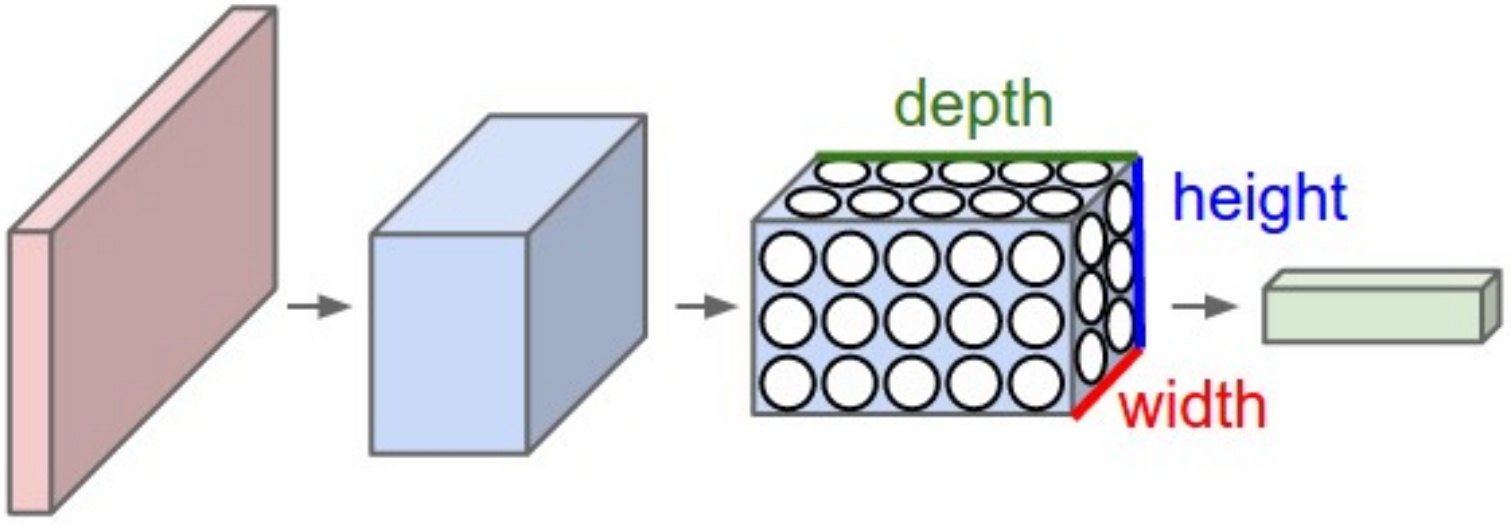
- Neural Networks workflow
  - Feature extraction
  - Filtering for redundant features, correlating features etc.
- Deep Neural Networks
  - No need for feature engineering

Content Frames

Non-content Frames

Sample Classification problem that can be easily accomplished using deep learning with high accuracy. It was a non-trivial problem a decade ago.

Convolutional Neural Networks

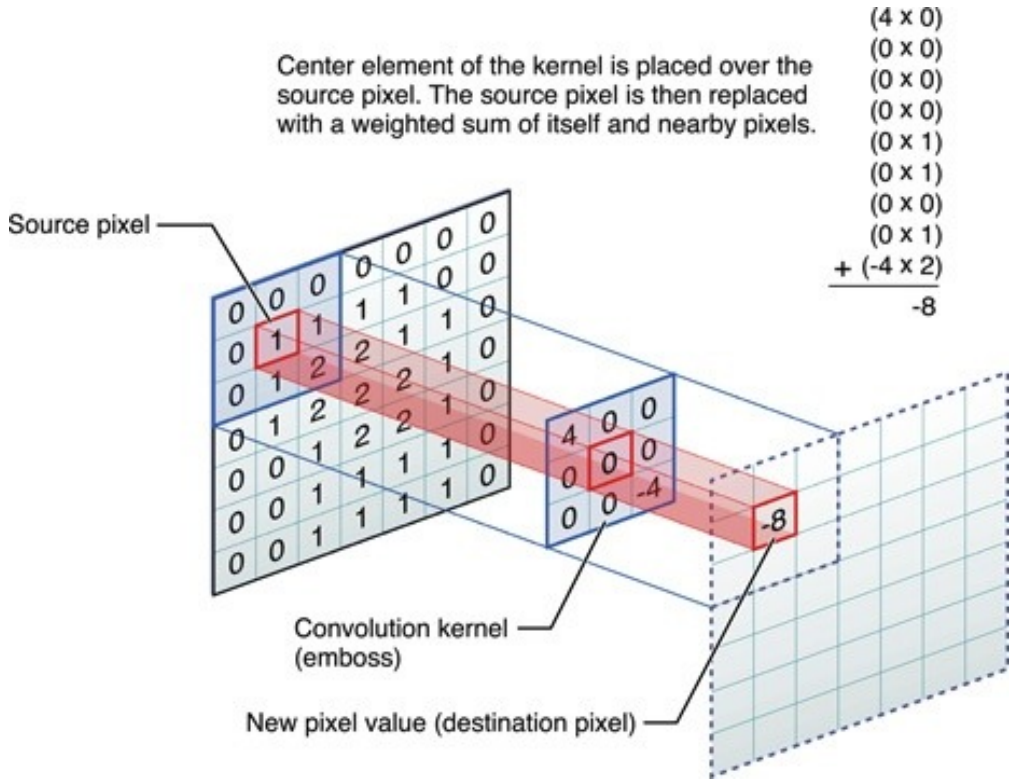


Types of Layers:

- Convolution Layer
- ReLU Layer
- Pooling Layer
- Dropout Layer
- Softmax Layer

http://cs231n.github.io/convolutional-networks/

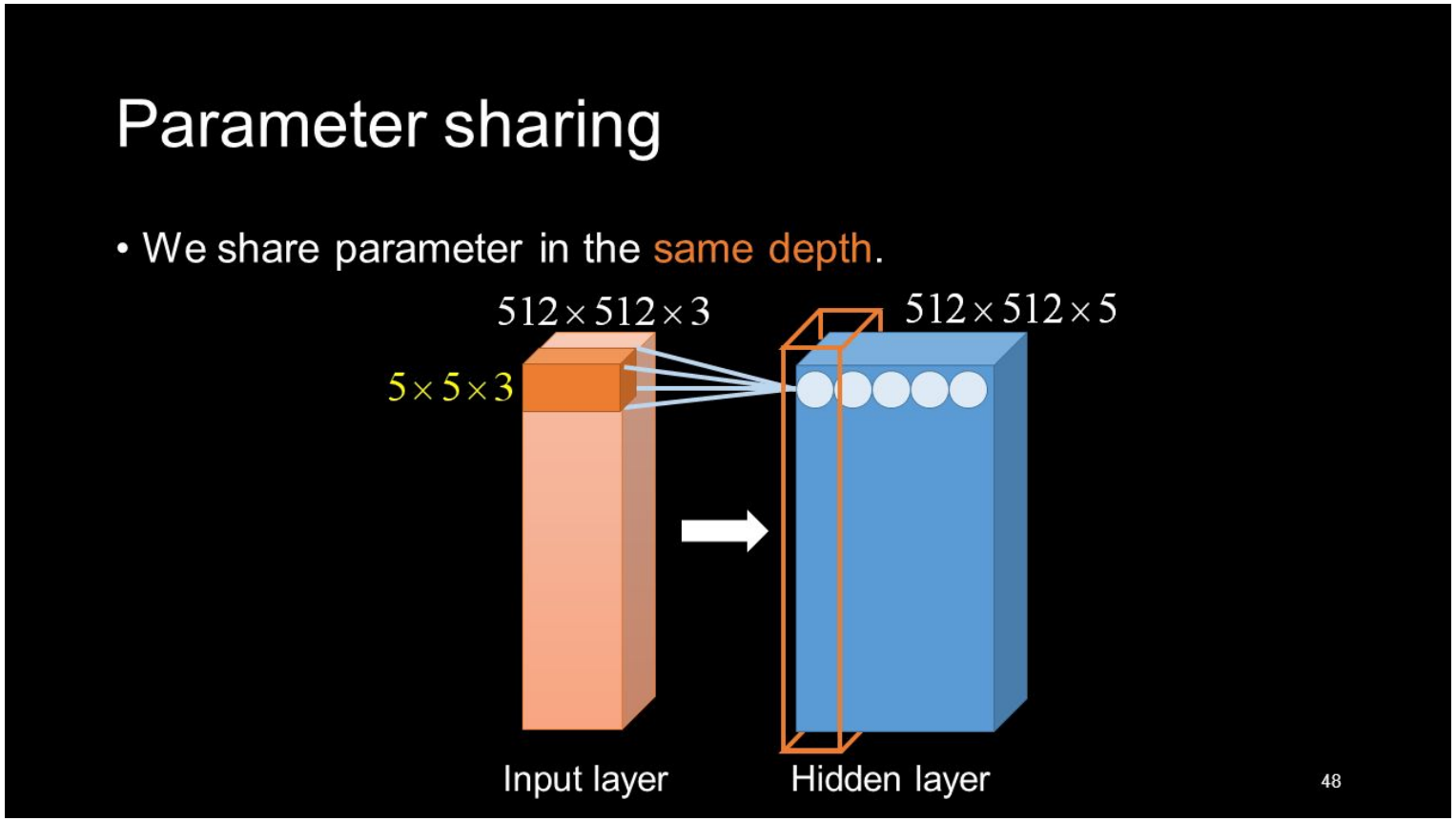
Convolution Operator in Convolutional Neural Networks



https://community.arm.com/graphics/b/blog/posts/when-parallelism-gets-tricky-accelerating-floyd-steinberg-on-the-mali-gpu

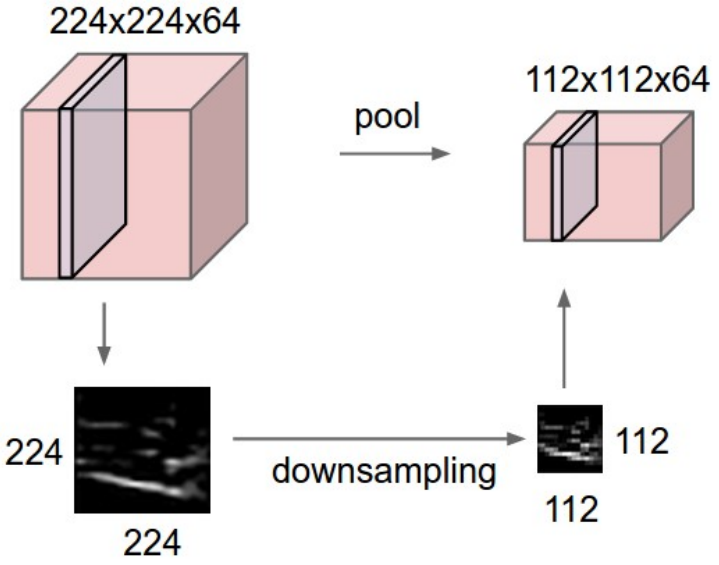
- CNN Hyper Parameters
  - Number of Filters (# of feature maps, depth of output layer)
  - Filter dimensions
  - Zero padding
  - Stride

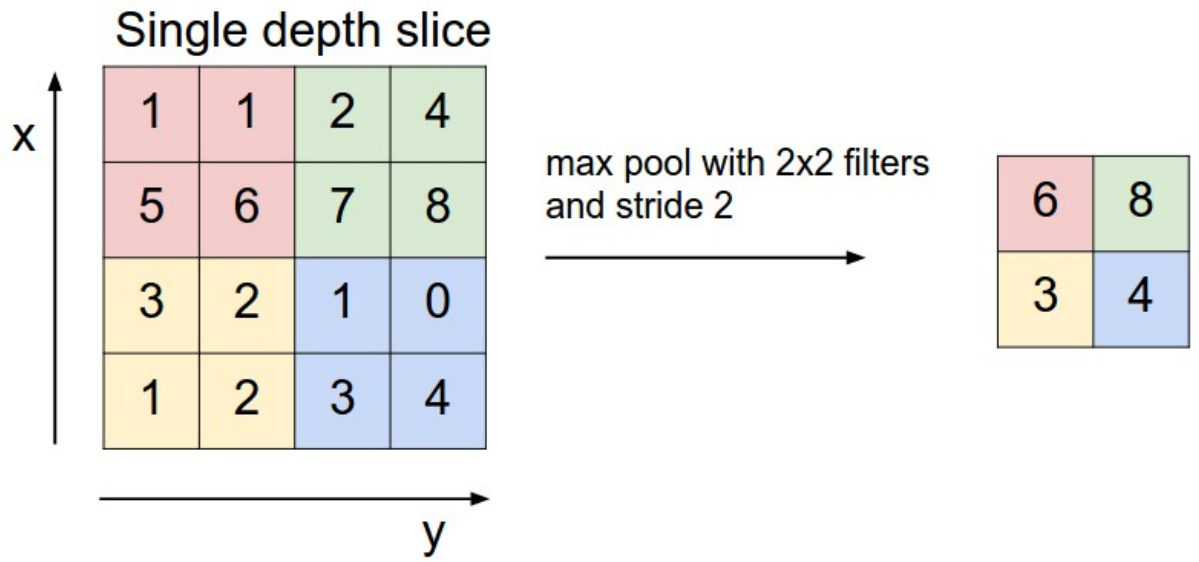
Parameter Sharing



- The kernel weights are shared across the image
- Reduces number of parameters
- Improves generalization capability of the model

Pooling in CNN

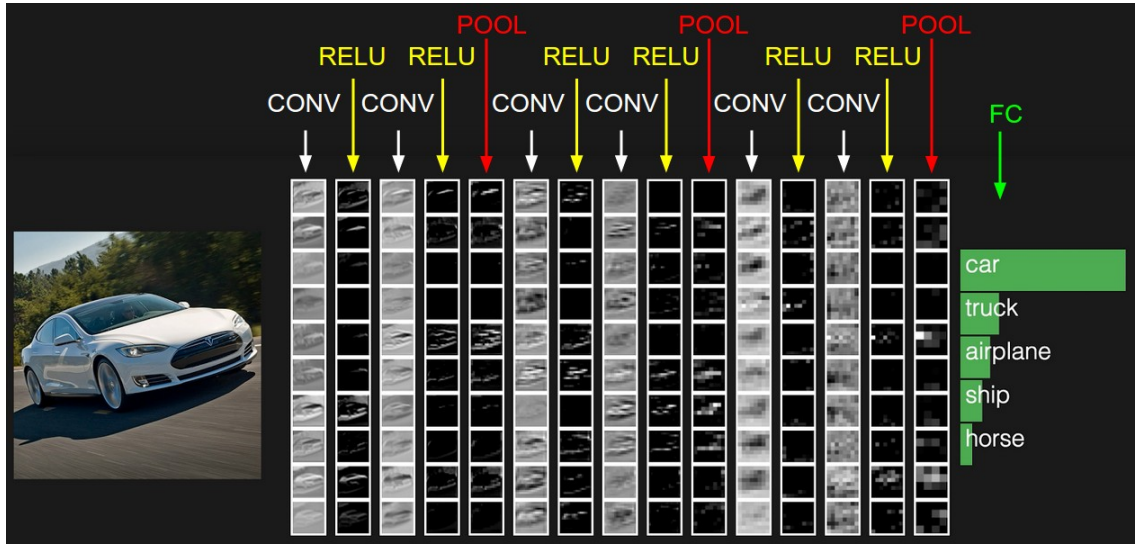




Convolutional Neural Networks

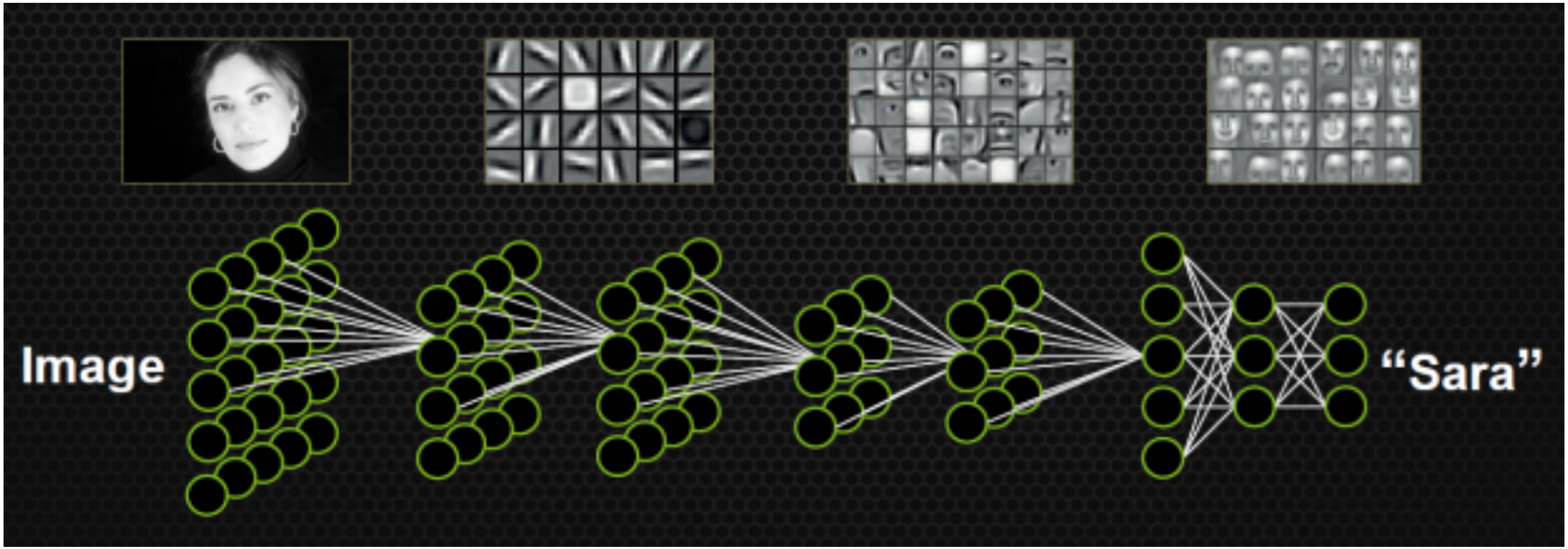
INPUT -> [[CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC

* Indicates repitition
POOL? – Optional pooling layer
M >= 0
N >= 0
K >= 0



https://community.arm.com/graphics/b/blog/posts/when-parallelism-gets-tricky-accelerating-floyd-steinberg-on-the-mali-gpu

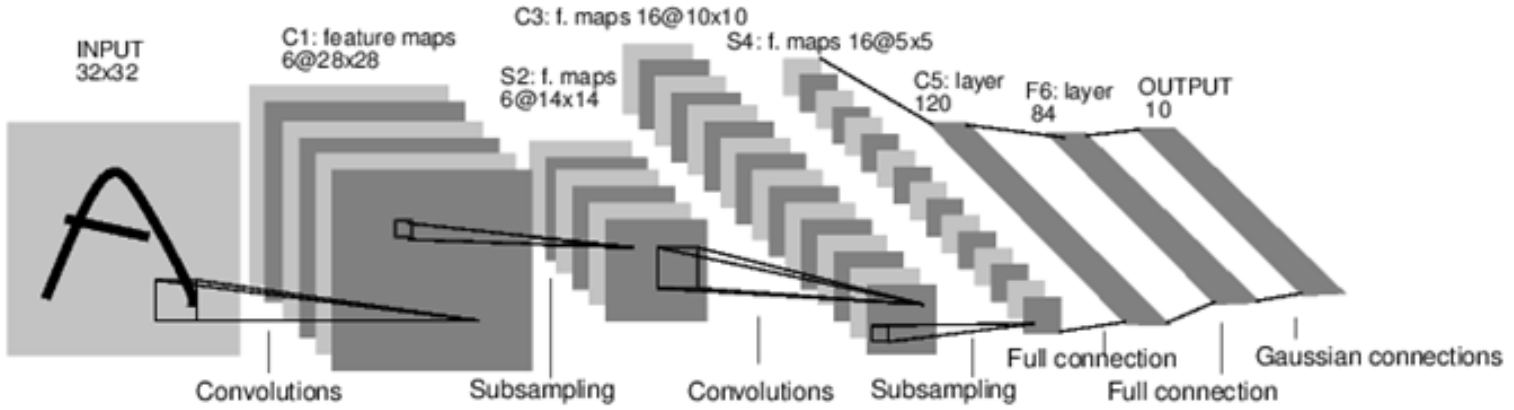
Features learned in CNN



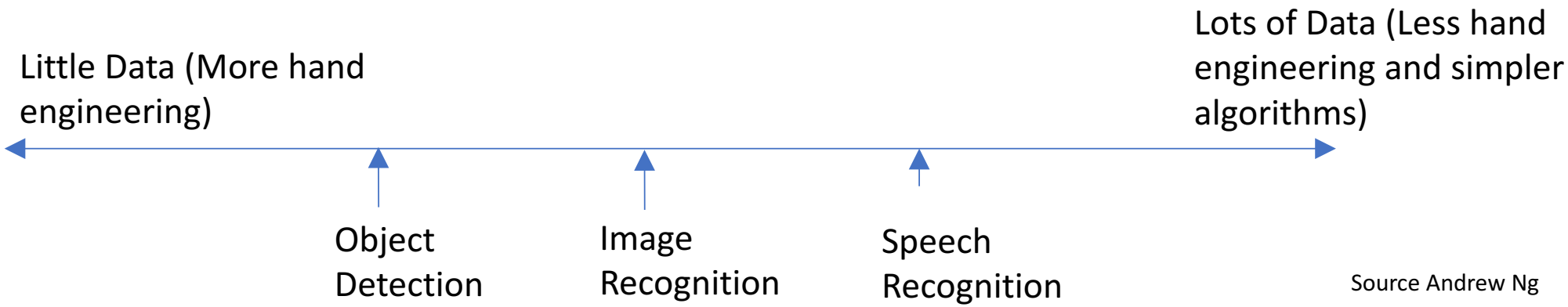
https://devblogs.nvidia.com/parallelforall/accelerate-machine-learning-cudnn-deep-neural-network-library/

Some Popular CNNs

- LeNet (1990)
- AlexNet (2012)
- ZF net (2013)
- GoogLeNet (2014)
- VGGNet (2014)
- ResNet (2015)



A Full Convolutional Neural Network (LeNet)



Source Andrew Ng

AlexNet Exercise with IBM-Caffe

- /opt/DL/caffe-ibm/examples/00-classification.ipynb
- Linux Commands:
  - cp /opt/DL/caffe-ibm/examples/00-classification.ipynb $PWD
  - source /opt/DL/caffe-ibm/caffe-activate
  - pip install scikit-image (3-5 minutes)
  - sudo pip install pyyaml
- Watch for ipython errors in command line interface. For Linux commands in the notebook, use sudo for permissions issues

Transfer Learning

Transfer Learning Scenarios

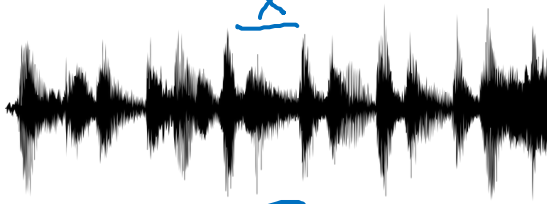
- Fixed Feature Extractor
- Fine Tuning the ConvNet
  - Train entire or part of the network
- Caffe-Zoo

Transfer Learning Exercise Using Caffe

- Linux Commands
  - cp /opt/DL/caffe-ibm/examples/02-fine-tuning.ipynb $PWD
  - sudo chmod 777 /opt/DL/caffe-ibm/models/finetune_flickr_style
- Watch for ipython errors in command line interface. For Linux commands in the notebook, use sudo for permissions issues

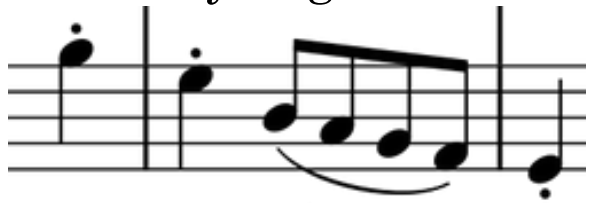
Applications of RNNs

Speech recognition



"The quick brown fox jumped over the lazy dog."

Music generation



Sentiment classification

"There is nothing to like in this movie."

DNA sequence analysis

AGCCCCTGTGAGGAACTAG

AGCCCCTGTGAGGAACTAG

Machine translation

Voulez-vous chanter avec moi?

Do you want to sing with me?

Video activity recognition

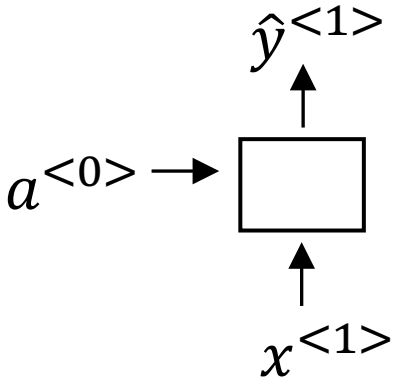
Running

Name entity recognition

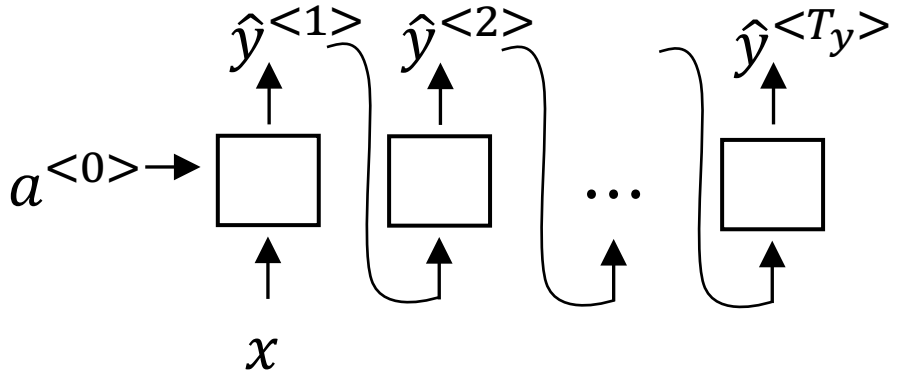
Yesterday, Harry Potter met Hermione Granger.

Yesterday, Harry Potter met Hermione Granger.

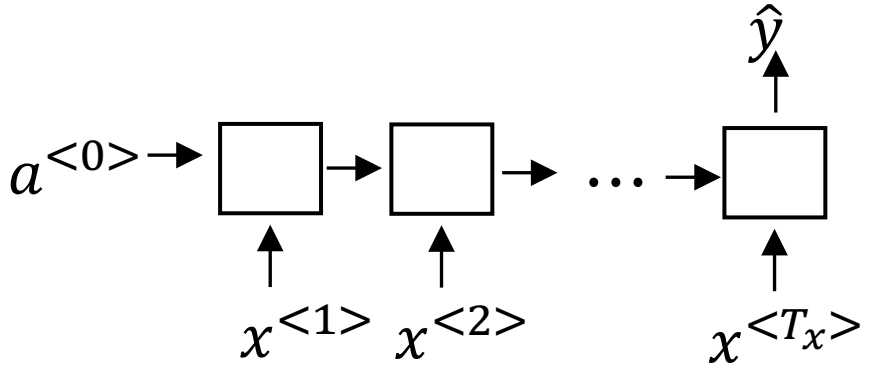
Summary of RNN types



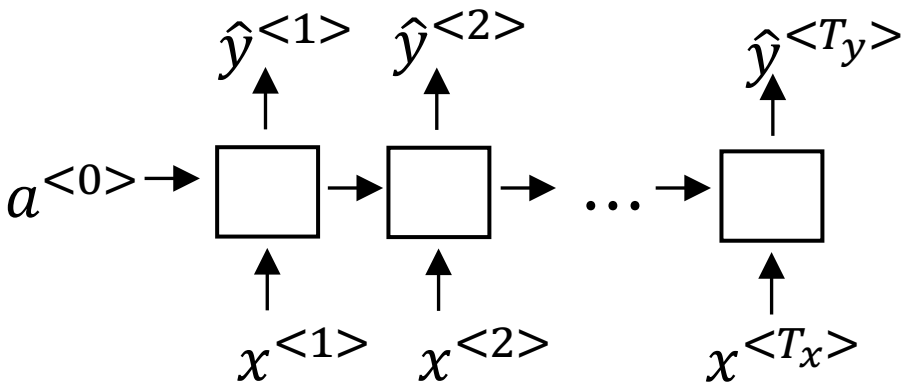
One to one



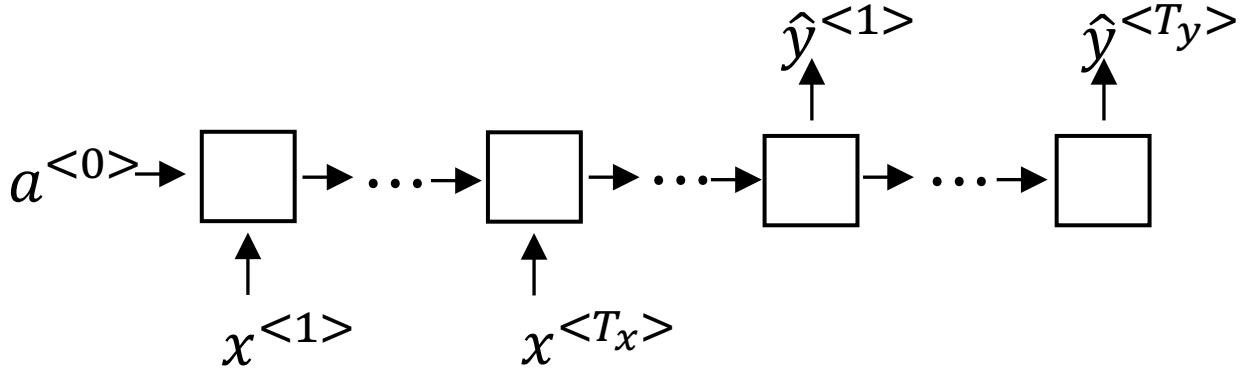
One to many



Many to one

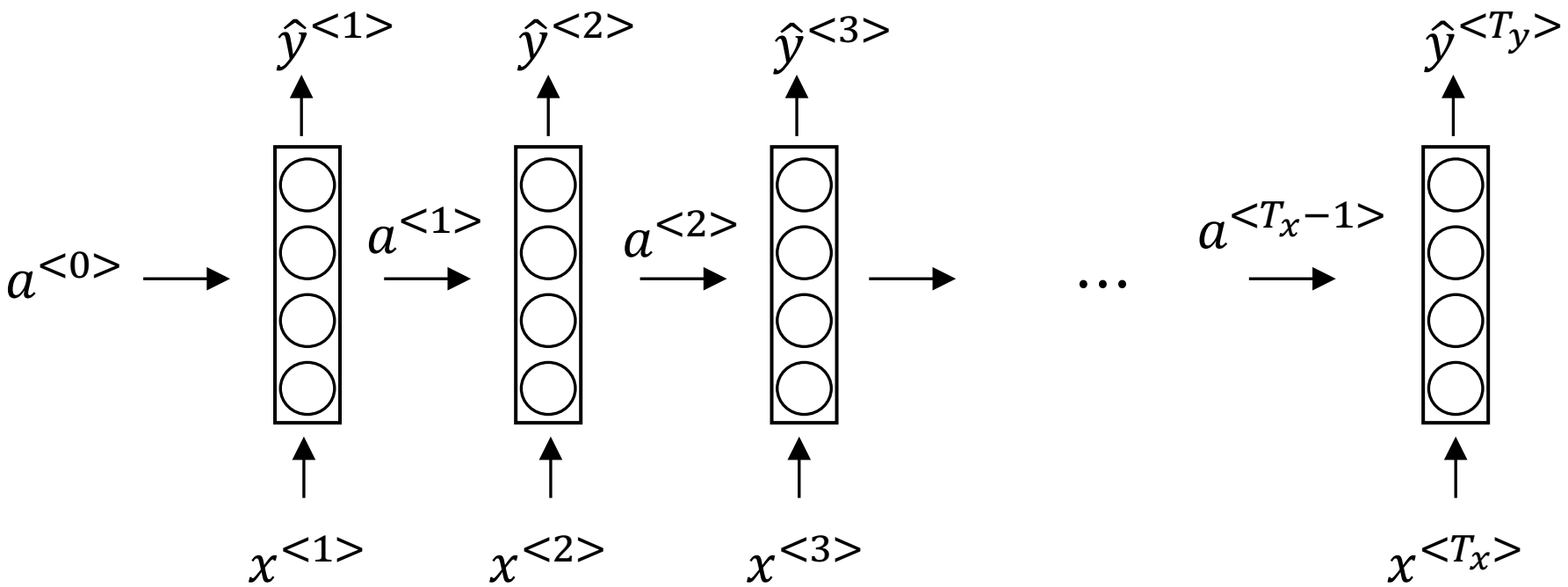


Many to many



Many to many

RNN Forward Propagation



RNN Parameters

$$a^{<t>} = g(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

$$\hat{y}^{<t>} = g(W_{ya}a^{<t>} + b_y)$$

Recurrent Neural Networks

https://github.com/aymericdamien/TensorFlow-Examples/blob/master/notebooks/3_NeuralNetworks/recurrent_network.ipynb

https://github.com/aymericdamien/TensorFlow-Examples/blob/master/notebooks/3_NeuralNetworks/dynamic_rnn.ipynb

https://github.com/aymericdamien/TensorFlow-Examples/blob/master/notebooks/3_NeuralNetworks/bidirectional_rnn.ipynb

Thank You