Smart Mobile Platform

Deep Neural Networks

Prof. Joongheon Kim
Korea University, School of Electrical Engineering
https://joongheon.github.io
joongheon@korea.ac.kr

Deep Learning Revolution is Real

Geoffrey E Hinton

Yoshua Bengio

Yann LeCun

FATHERS OF THE DEEP LEARNING REVOLUTION RECEIVE ACM A.M. TURING AWARD

Bengio, Hinton, and LeCun Ushered in Major Breakthroughs in Artificial Intelligence

ACM named Yoshua Bengio, Geoffrey Hinton, and Yann LeCun recipients of the 2018 ACM A.M. Turing Award for conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing. Bengio is Professor at the University of Montreal and Scientific Director at Mila, Quebec's Artificial Intelligence Institute; Hinton is VP and Engineering Fellow of Google, Chief Scientific Adviser of The Vector Institute, and University Professor Emeritus at the University of Toronto; and LeCun is Professor at New York University and VP and Chief AI Scientist at Facebook.

Working independently and together, Hinton, LeCun and Bengio developed conceptual foundations for the field, identified surprising phenomena through experiments, and contributed engineering advances that demonstrated the practical advantages of deep neural networks. In recent years, deep learning methods have been responsible for astonishing breakthroughs in computer vision, speech recognition, natural language processing, and robotics—among other applications.

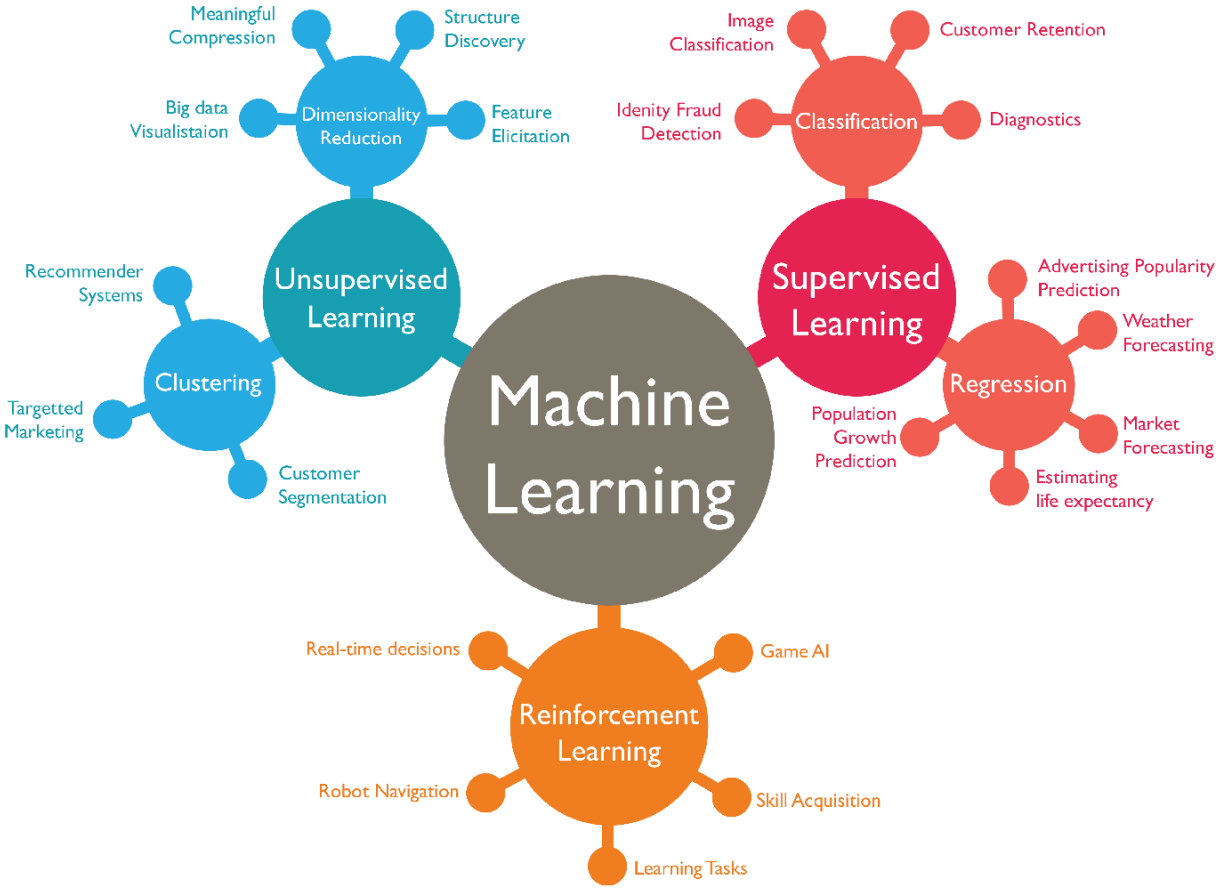
While the use of artificial neural networks as a tool to help computers recognize patterns and simulate human intelligence had been introduced in the 1980s, by the early 2000s, LeCun, Hinton and Bengio were among a small group who remained committed to this approach. Though their efforts to rekindle the AI community's interest in neural networks were initially met with skepticism, their ideas recently resulted in major technological advances, and their methodology is now the dominant paradigm in the field.

The ACM A.M. Turing Award, often referred to as the "Nobel Prize

Alan Turing (1912-1954)
Father of Computer Science

https://amturing.acm.org/

Machine Learning Overview

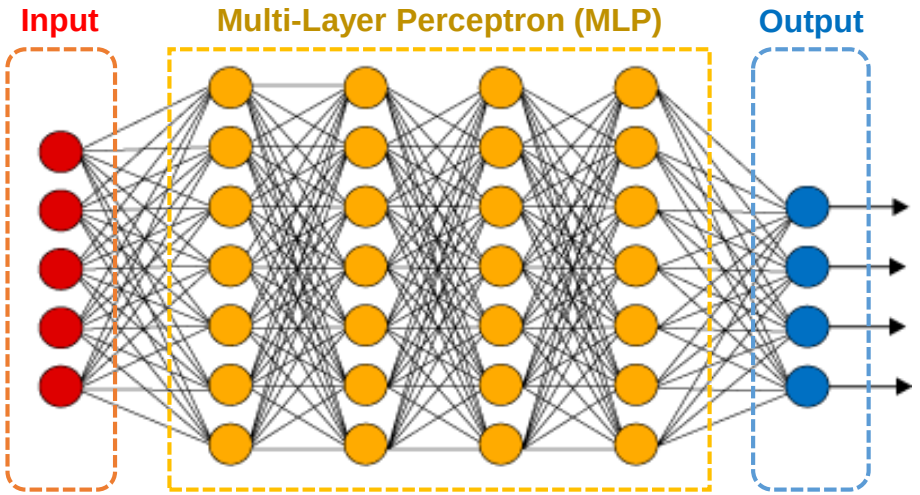


- How Deep Learning Works?
  - Deep Learning Computation Procedure

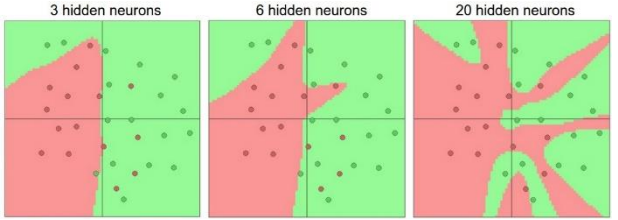
Deep Learning Model Setup
- MLP, CNN, RNN, GAN, or Customized
- # Hidden Layers, # Units, Input/Output, …
- Cost Function / Optimizer Selection

Training (with Large-Scale Dataset)
- Input: Data, Output: Labels
- Learning → Weights Updates for Cost Function Minimization

Inference / Testing (Real-Word Execution)
- Input: Real-World Input Data
- Output: Interference Results based on Updated Weights in Deep Neural Networks



Non-Linear Training (Weights Updates) for Cost Minimization: GD, SGD, Adam, etc.



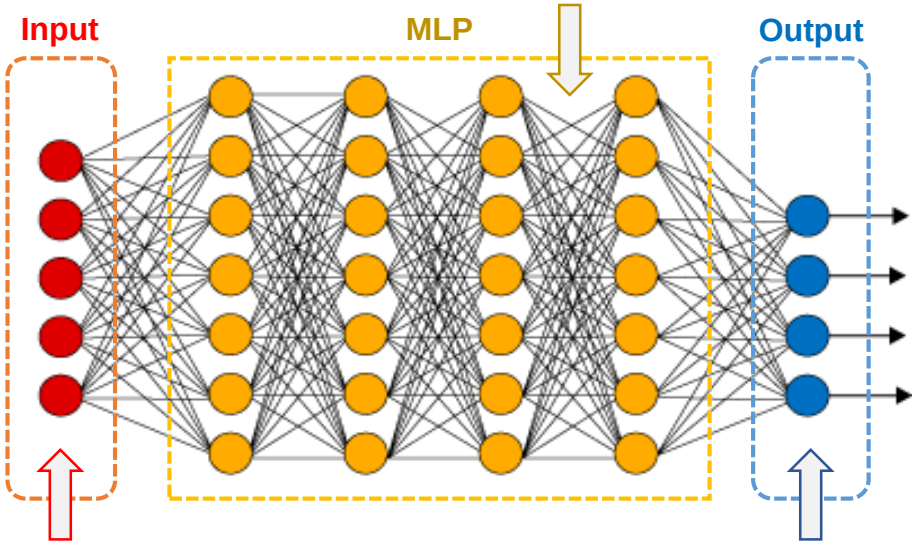
- How Deep Learning Works?
  - Deep Learning Computation Procedure

Deep Learning Model Setup
- MLP, CNN, RNN, GAN, or Customized
- # Hidden Layers, # Units, Input/Output, …
- Cost Function / Optimizer Selection

Training (with Large-Scale Dataset)
- Input: Data, Output: Labels
- Learning → Weights Updates for Cost Function Minimization

Inference / Testing (Real-Word Execution)
- Input: Real-World Input Data
- Output: Interference Results based on Updated Weights in Deep Neural Networks

All weights in units are trained/set (under cost minimization)



INPUT: Data
- One-Dimension Vector

OUTPUT: Labels
- One-Hot Encoding

We need a lot of training data for generality (otherwise, we will suffer from overfitting problem).

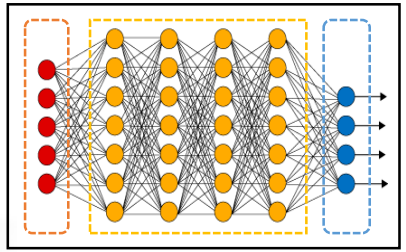
- How Deep Learning Works?
  - Deep Learning Computation Procedure

Deep Learning Model Setup
- MLP, CNN, RNN, GAN, or Customized
- # Hidden Layers, # Units, Input/Output, …
- Cost Function / Optimizer Selection

Training (with Large-Scale Dataset)
- Input: Data, Output: Labels
- Learning → Weights Updates for Cost Function Minimization

Inference / Testing (Real-Word Execution)
- Input: Real-World Input Data
- Output: Interference Results based on Updated Weights in Deep Neural Networks

Trained Model



Intelligent Surveillance Platforms

INPUT: Real-Time Arrivals

OUTPUT: Inference
- Computation Results based on (i) INPUT and (ii) trained weights in units (trained model).

- How Deep Learning Works?
  - Issue - **Overfitting**

What if we do not have enough data for training (not enough to derive Gaussian/normal distribution)?

Deep Learning Model Setup
- MLP, CNN, RNN, GAN, or Custom
- # Hidden Layers, # Units, Input/Output, …
- Cost Function / Optimizer Selection

Training (with Large-Scale Dataset)
- Input: Data, Output: Labels
- Learning → Weights Updates for Cost Function Minimization

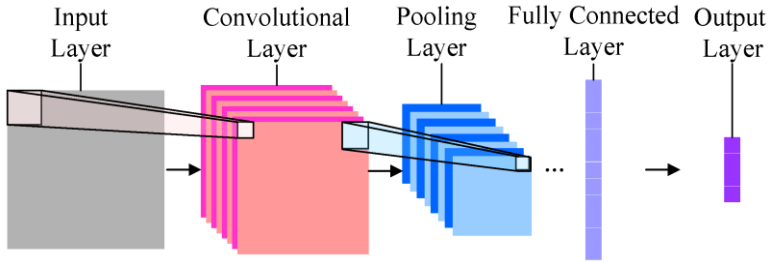
Inference / Testing (Real-Word Execution)
- Input: Real-World Input Data
- Output: Interference Results based on Updated Weights in Deep Neural Networks

Situation becomes worse when the model (with insufficient training data) accurately fits on training data.

To Combat the Overfitting
- More training data
- Autoencoding (or variational auto-encoder (VAE))
- Droupout
- Regularization

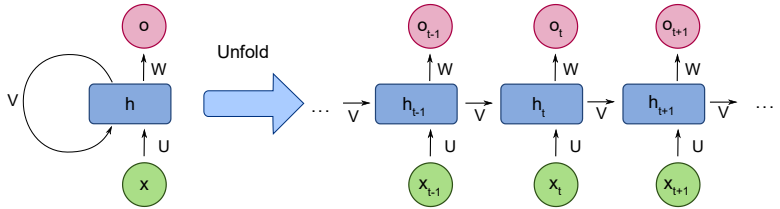
- Two Major Deep Learning Models → CNN vs. RNN

Convolutional Neural Network (CNN)



- In conventional neural network architectures, the input should be one-dimensional vector.
- In many applications, the input should be multi-dimensional (e.g., 2D for images). Thus, we need architectures in order to recognize the features in high-dimensional data.
- Mainly used for **visual information learning**

Recurrent Neural Network (RNN)

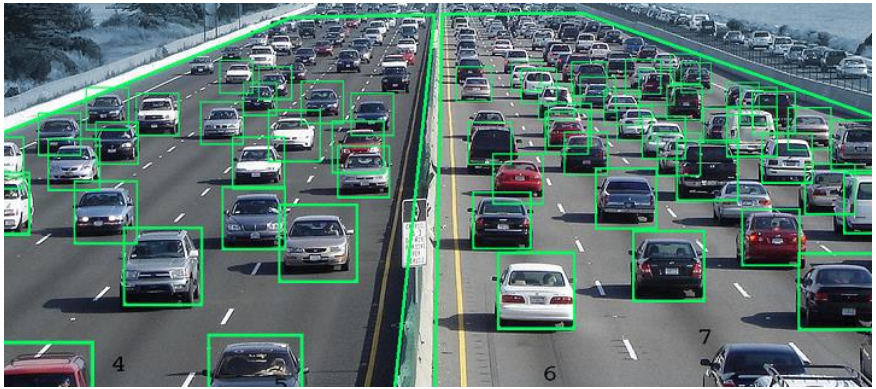


- In conventional neural network architectures, there is no way to introduce the concept of time.
- The time index can be represented as the chain of neural network models.
- The representative models are LSTM and GRU.
- Mainly used for **time-series information learning**

Deep Learning: Vision

Visual Learning

- Object Recognition
- Style Transfer
- Deblurring and Denoising
- Super-Resolution
- …



Speech/Language Learning

- Speech Recognition
- Machine Translation
- Information Retrieval
- …

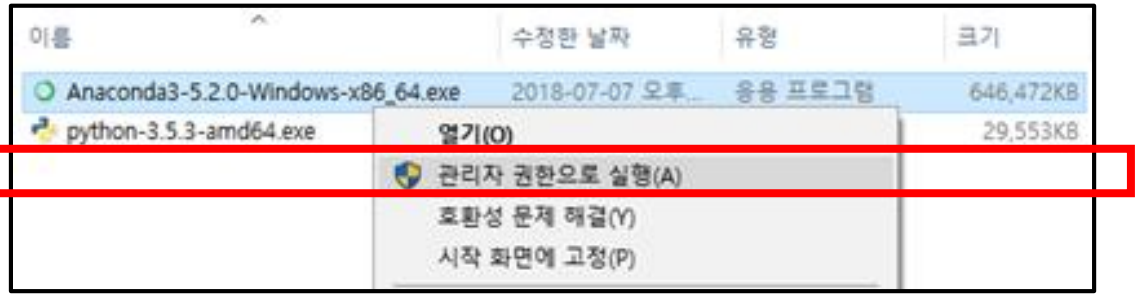
- An Emerging Direction, Generative Adversarial Network (GAN)
  - Training both of **generator** and **discriminator**; and then generates samples which are similar to the original samples.

Deep Learning Basics and Software
Software Installation and Examples

- **Python/TensorFlow/Keras Installation**
- Python/TensorFlow Examples

- Preparation
  - Python Download: **python-3.5.3-amd64.exe**
  - Anaconda Download: **Anaconda3-5.2.0-Windows-x86_64.exe**

- Installation Procedure
  - [1] Python Installation
  - [2] Conda Environment Setup
    - Execute the downloaded **Anaconda EXE as admin**
    - Execute **Anaconda Prompt** as admin
  - [3] TensorFlow Installation



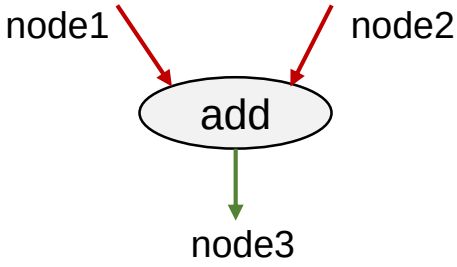
Python/TensorFlow/Keras Installation

- Installation Procedure
  - [1] Python Installation
  - [2] Conda Environment Setup
  - **[3] TensorFlow Installation**
    - **> conda create –n tensorflow python=3.5**
    - Proceed([y]/n)? y
    - **> pip install tensorflow**
    - **> pip install keras**

Deep Learning Basics and Software
Software Installation and Examples

- Python/TensorFlow/Keras Installation
- **Python/TensorFlow Examples**

- Quick Start Example



- **add** is a node which represents addition operation
  - **node1**: input tensor
  - **node2**: input tensor
  - **node3**: resultant tensor

ex_add.py

```
1  import tensorflow as tf
2  # Create nodes in computation graph
3  node1 = tf.constant(3, dtype=tf.int32)
4  node2 = tf.constant(5, dtype=tf.int32)
5  node3 = tf.add(node1, node2)
6
7  # Create session object
8  sess = tf.Session()
9  print("node1 + node2 = ", sess.run(node3))
10 # Close the session
11 sess.close()
```

- Quick Start Example

ex_add.py

```
1  import tensorflow as tf
2  # Create nodes in computation graph
3  node1 = tf.constant(3, dtype=tf.int32)
4  node2 = tf.constant(5, dtype=tf.int32)
5  node3 = tf.add(node1, node2)
6
7  # Create session object
8  sess = tf.Session()
9  print("node1 + node2 = ", sess.run(node3))
10 # Close the session
11 sess.close()
```

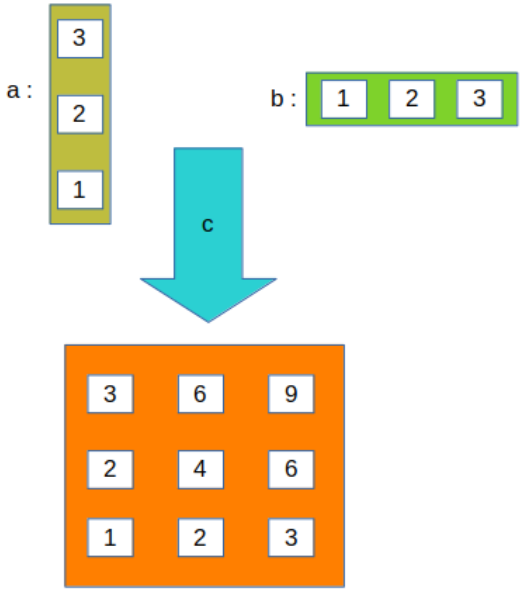
ex_add2.py

```
1  import tensorflow as tf
2  # Create nodes in computation graph
3  node1 = tf.constant(3, dtype=tf.int32)
4  node2 = tf.constant(5, dtype=tf.int32)
5  node3 = tf.add(node1, node2)
6
7  # Create session object
8  with tf.Session() as sess:
9      print("node1 + node2 = ", sess.run(node3))
```

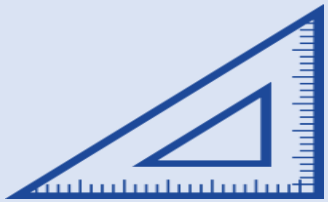
- Example: Placeholder

ex_placeholder.py

```
1  import tensorflow as tf
2
3  # Create nodes in computation graph
4  a = tf.placeholder(tf.int32, shape=(3,1))
5  b = tf.placeholder(tf.int32, shape=(1,3))
6  c = tf.matmul(a, b)
7
8  # Run computation graph
9  with tf.Session() as sess:
10     print(sess.run(c, feed_dict={a: [[3], [2], [1]], b: [[1, 2, 3]]}))
```



Linear Functions



Linear Regression
Binary Classification
Softmax Classification

Nonlinear Functions

Neural Network (NN)
Convolutional NN (CNN)
CNN for CIFAR-10
Recurrent NN (RNN)

Advanced Topics

Gen. Adv. Network (GAN)
Interpolation
PCA/LDA
Overfitting

Deep Learning Basics and Software
Linear Regression
Linear Regression Theory

- **Linear Regression Theory**
- Linear Regression Implementation

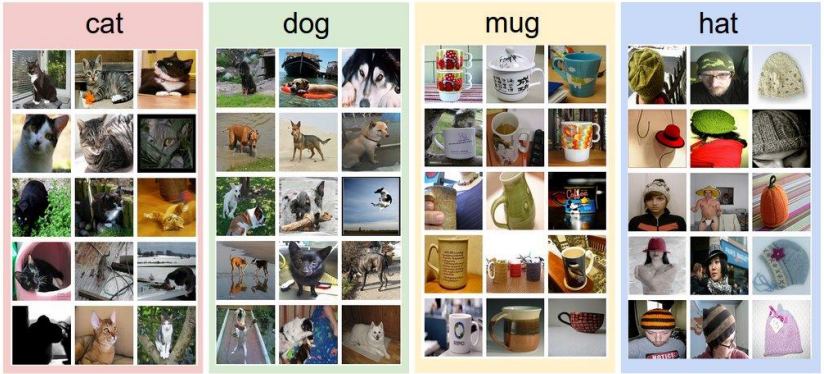
Regression and Classification

Regression (Examples)

- Exam Score Prediction (Linear Regression)

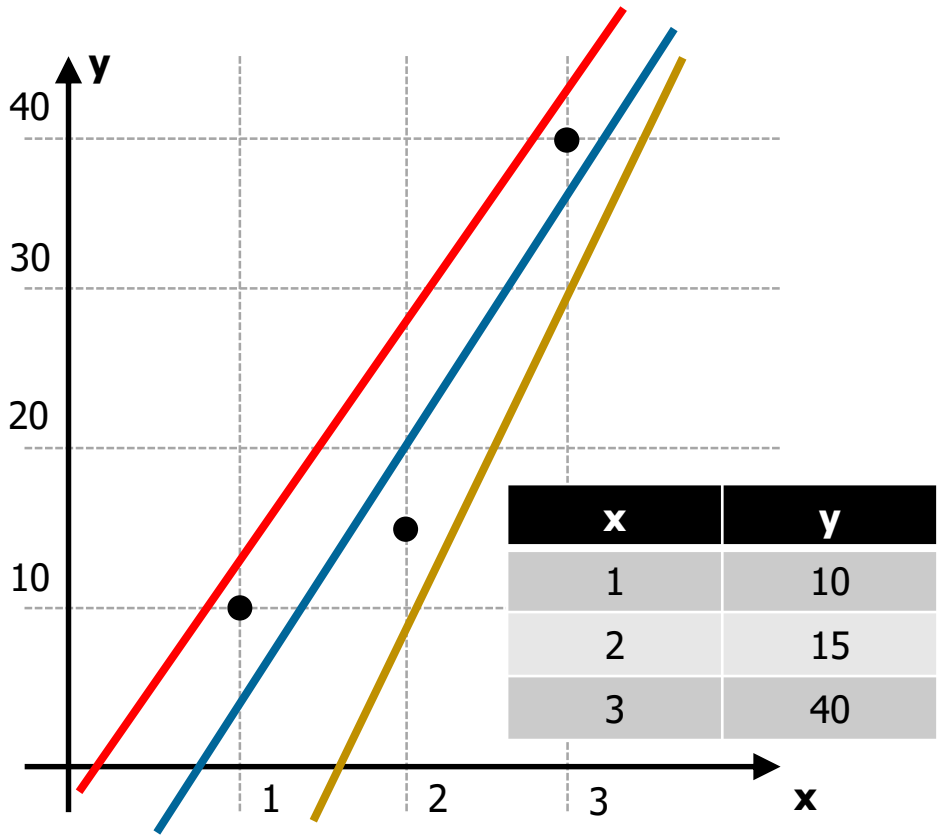
Classification (Examples)

- Pass/Fail (Binary Classification)
- Letter Grades (Multi-Level Classification)



Linear Regression

- Linear model: $H(x) = Wx + b$
- Which model is the best among the given three?



x	y
1	10
2	15
3	40

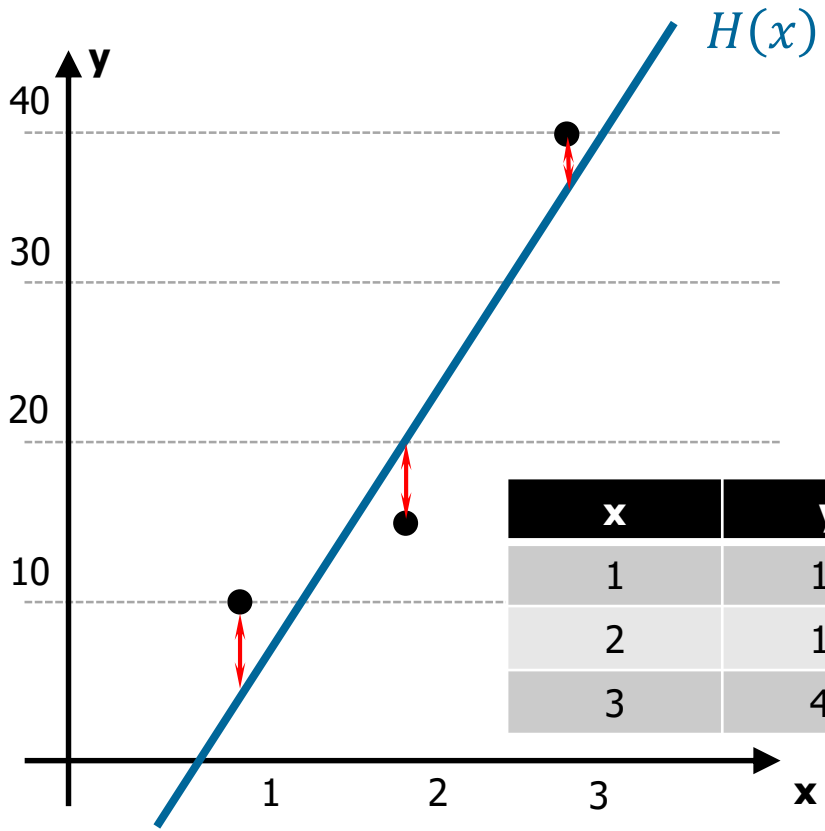
- Cost Function (or Loss Function)
  - How to fit the line to training data
  - The difference between model values and real measurements:

m: The number of training data

$$\frac{1}{m}\sum_{i=1}^{m}\left(H\left(x^i\right)-y^i\right)^2$$

$$H(x)=Wx+b$$

$$\text{Cost}(W,b)=\frac{1}{m}\sum_{i=1}^{m}\left(H\left(x^i\right)-y^i\right)^2$$

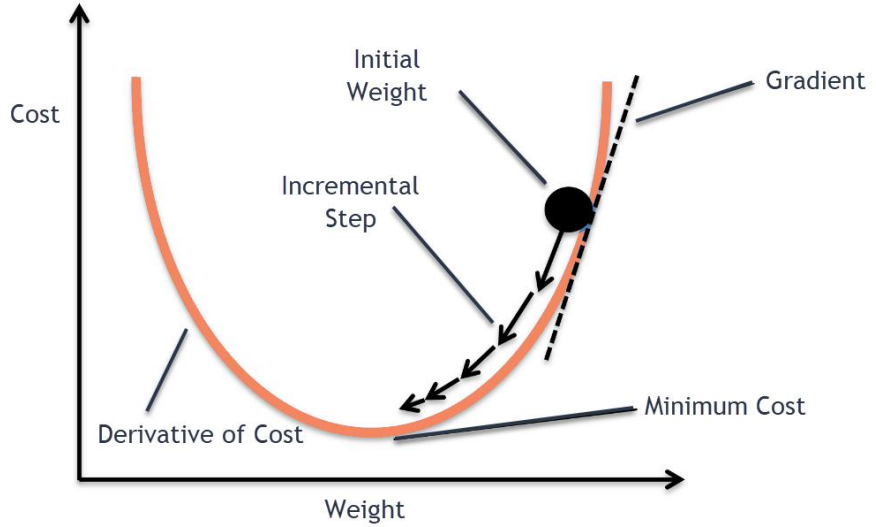


x	y
1	10
2	15
3	40

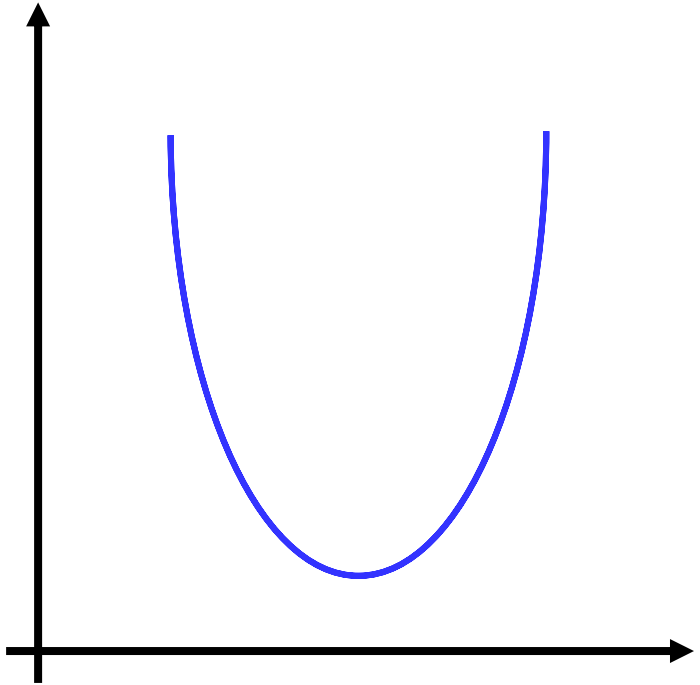
- Cost Function Minimization
  - Model: $H(x) = Wx + b$
  - Cost Function: $Cost(W, b) = \frac{1}{m}\sum_{i=1}^{m}\left(H\left(x^i\right) - y^i\right)^2 = \frac{1}{m}\sum_{i=1}^{m}\left(Wx^i + b - y^i\right)^2$
  - How to Minimize this Function? → **Gradient Descent Method**
    - Angle → Differentiation

$$W \leftarrow W - \alpha \frac{\partial}{\partial W} Cost(W)$$

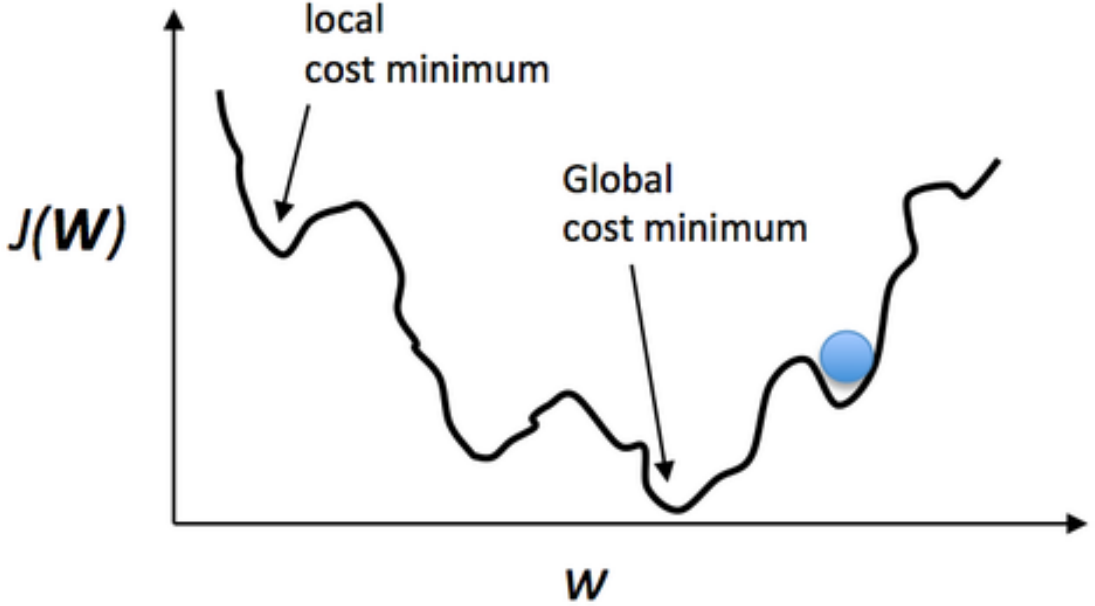
α: Learning rate



- Learning Rates
  - Too large: Overshooting
  - Too small: takes too long, stops in the middle
- How can we determine the learning rates?
  - Try several learning rates
    - Observe the cost function
    - Check it goes down in a reasonable rate



- Cost Function Minimization
  - Gradient Descent Method is only good for convex functions.



- Multi-Variable Linear Regression
  - Model:

$$H(x_1, x_2, \dots, x_n) = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$$

  - Cost:

$$Cost(W, b) = \frac{1}{m} \sum_{i=1}^{m} \left(H\left(x_1^i, x_2^i, \dots, x_n^i\right) - y^i \right)^2$$

- Multi-Variable Linear Regression
  - Model:

$$H(x_1, x_2, \dots, x_n) = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b \Rightarrow H(X) = XW + b$$

$$\begin{pmatrix} x_1 & x_2 \dots & x_n \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ \dots \\ w_n \end{pmatrix} = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

X W

$$\Rightarrow H(X) = XW^T + b$$

when $W = \begin{pmatrix} w_1 & w_2 \dots & w_n \end{pmatrix}$

Deep Learning Basics and Software
Linear Regression
Linear Regression Implementation

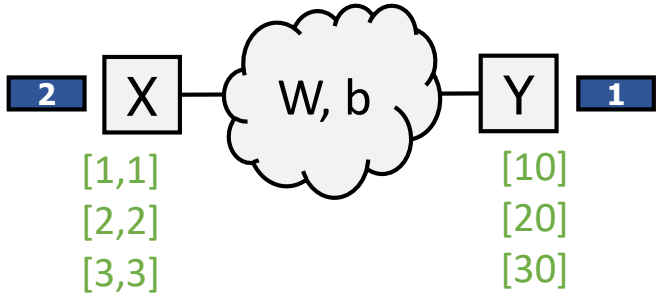
- Linear Regression Theory
- **Linear Regression Implementation**

Linear Regression Implementation (TensorFlow)

- TensorFlow
  - **Linear Regression**
- Keras
  - Linear Regression

Linear Regression Implementation (TensorFlow)

```
1  import tensorflow as tf
2
3  x_data = [[1,1], [2,2], [3,3]]
4  y_data = [[10], [20], [30]]
5  X = tf.placeholder(tf.float32, shape=[None, 2])
6  Y = tf.placeholder(tf.float32, shape=[None, 1])
7
8  W=tf.Variable(tf.random_normal([2,1]))
9  b=tf.Variable(tf.random_normal([1]))
10
11 model = tf.matmul(X,W)+b
12 cost = tf.reduce_mean(tf.square(model - Y))
13 train = tf.train.GradientDescentOptimizer(0.01).minimize(cost)
14
15 with tf.Session() as sess:
16     sess.run(tf.global_variables_initializer())
17     # Training
18     for step in range(2001):
19         c, W_, b_, _ = sess.run([cost, W, b, train], feed_dict={X: x_data, Y: y_data})
20         print(step, c, W_, b_)
21     # Testing
22     print(sess.run(model, feed_dict={X: [[4,4]]}))
```



Model, Cost, Train

Linear Regression Implementation (TensorFlow)

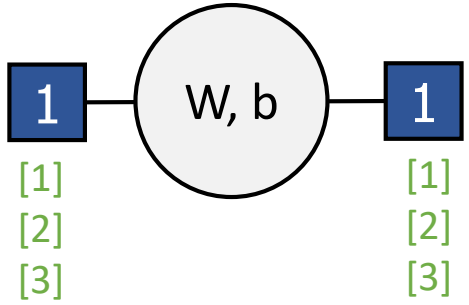
```
 [4.667964 ]] [0.014943]
1991 3.1940912e-05 [[5.325548 ]
 [4.6679726]] [0.01490401]
1992 3.1772186e-05 [[5.3255568]
 [4.667981 ]] [0.0148651]
1993 3.1603915e-05 [[5.3255653]
 [4.6679897]] [0.01482627]
1994 3.143936e-05 [[5.325574 ]
 [4.6679983]] [0.01478756]
1995 3.1277286e-05 [[5.3255825]
 [4.668007 ]] [0.01474891]
1996 3.1110336e-05 [[5.325591 ]
 [4.6680155]] [0.01471035]
1997 3.095236e-05 [[5.325599 ]
 [4.6680236]] [0.01467189]
1998 3.079518e-05 [[5.325608]
 [4.668032]] [0.01463356]
1999 3.062387e-05 [[5.325616 ]
 [4.6680403]] [0.01459529]
2000 3.0467529e-05 [[5.325624 ]
 [4.6680484]] [0.01455714]
[[39.989246]]
```

Linear Regression Implementation (Keras)

- TensorFlow
  - Linear Regression
- Keras
  - **Linear Regression**

Linear Regression Implementation (Keras)

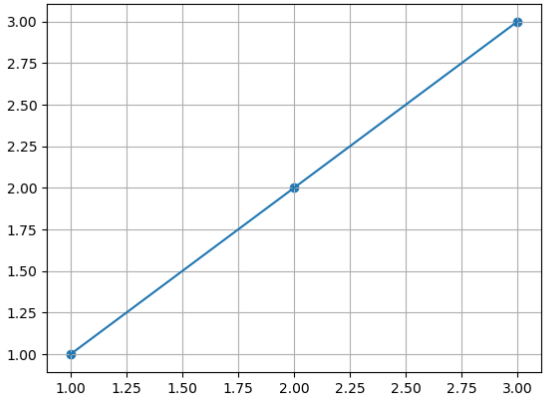
```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from keras.models import Sequential
4  from keras.layers import Dense
5
6  # Data
7  x_data = np.array([[1], [2], [3]])
8  y_data = np.array([[1], [2], [3]])
9  # Model, Cost, Train
10 model = Sequential()
11 model.add(Dense(1, input_dim=1))
12 model.compile(loss='mse', optimizer='adam')
13 model.fit(x_data, y_data, epochs=1000, verbose=0)
14 model.summary()
15 # Inference
16 print(model.get_weights())
17 print(model.predict(np.array([4])))
18 # Plot
19 plt.scatter(x_data, y_data)
20 plt.plot(x_data, y_data)
21 plt.grid(True)
22 plt.show()
```



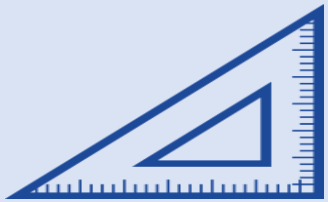
Model, Cost, Train

Linear Regression Implementation (Keras)

```
joongheon@joongheon-AB350M-Gaming-3: ~/Dropbox/codes_keras
joongheon@joongheon-AB350M-Gaming-3:~/Dropbox/codes_keras$ python keras_linearregression.py
/home/joongheon/anaconda3/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning: Con
version of the second argument of issubdtype from `float` to `np.floating` is deprecated. In
future, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
2019-06-29 16:39:04.566966: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU sup
ports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1)                 2
=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
[array([[0.999888]], dtype=float32), array([0.00024829], dtype=float32)]
[[3.9998002]]
joongheon@joongheon-AB350M-Gaming-3:~/Dropbox/codes_keras$
```



Linear Functions



Linear Regression
Binary Classification
Softmax Classification

Nonlinear Functions

Neural Network (NN)
Convolutional NN (CNN)
CNN for CIFAR-10
Recurrent NN (RNN)

Advanced Topics

Gen. Adv. Network (GAN)
Interpolation
PCA/LDA
Overfitting

Deep Learning Basics and Software
Binary Classification
Binary Classification Theory

- **Binary Classification Theory**
- Binary Classification Implementation

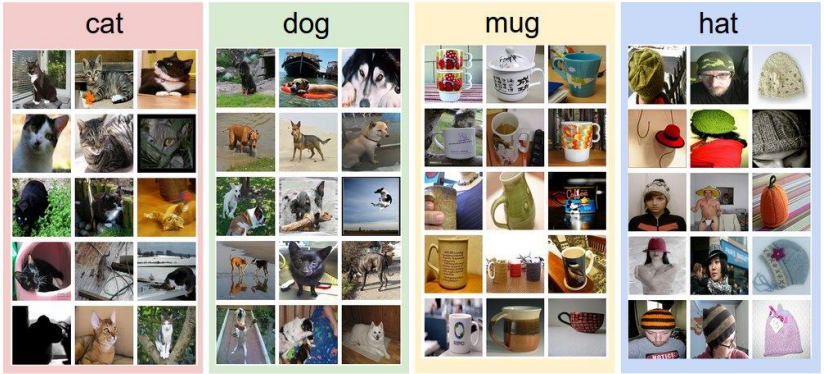
Regression and Classification

Regression (Examples)

- Exam Score Prediction (Linear Regression)

Classification (Examples)

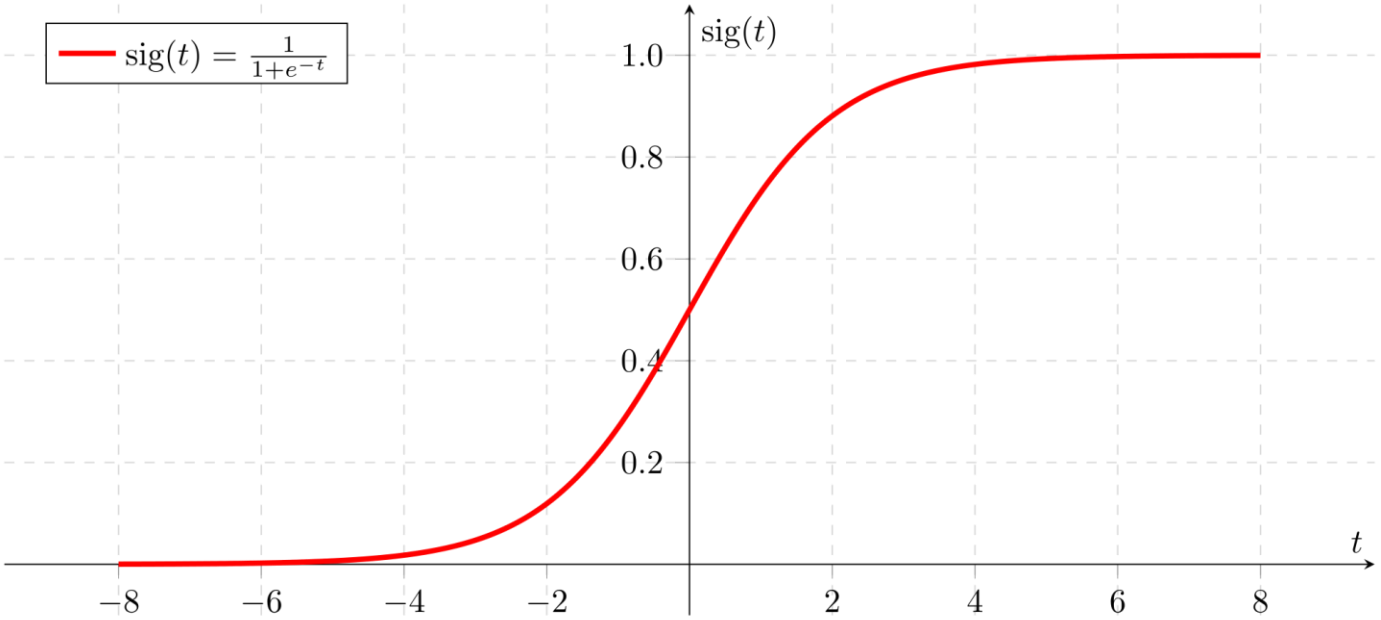
- Pass/Fail (Binary Classification)
- Letter Grades (Multi-Level Classification)



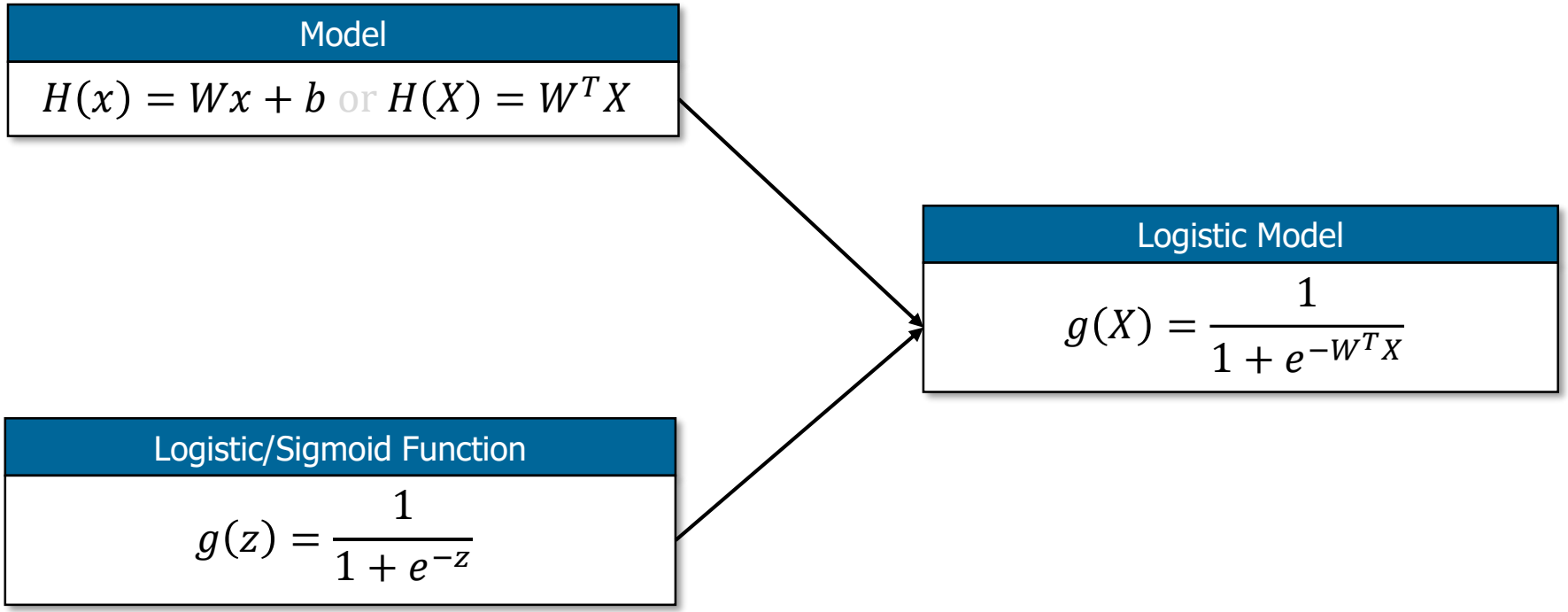
Binary Classification

- Binary Classification Examples
  - **Spam Detection:** Spam [1] or Ham [0]
  - **Facebook Feed:** Show [1] or Hide [0]
    - Facebook learns with your like-articles; and shows your favors.
  - **Credit Card Fraudulent Transaction Detection:** Fraud [1] or Legitimate [0]
  - **Tumor Image Detection in Radiology:** Malignant [1] or Benign [0]

- Binary Classification Basic Idea
  - Step 1) Linear regression with $H(x) = Wx + b$
  - Step 2) **Logistic/sigmoid function ($sig(t)$)** based on the result of Step 1.



Binary Classification



Binary Classification

$$\text{Cost}(W,b) = \frac{1}{m}\sum_{i=1}^{m}\left(H\left(x^i\right) - y^i\right)^2$$

Hypothesis

$$H(x) = Wx + b \text{ or } H(X) = W^T X$$

Gradient Descent Method can be used because $Cost(W,b)$ is convex (local minimum is global minimum).

Logistic Hypothesis

$$g(z) = \frac{1}{1 + e^{-W^T X}}$$

Gradient Descent Method can not be used because $Cost(W,b)$ is non-convex. **New Cost Function is required.**

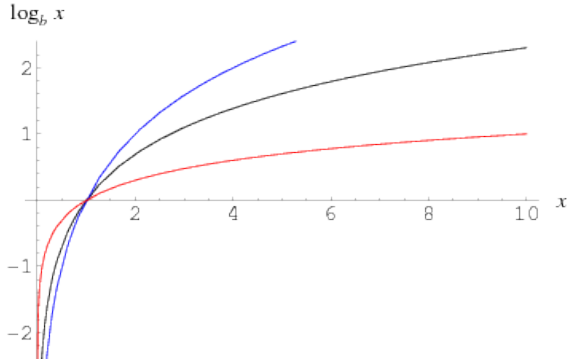
$$\text{Cost}(W) = \frac{1}{m}\sum c(H(x), y)$$

$$c(H(x), y) = \begin{cases} -\log\big(H(x)\big), & y = 1 \\ -\log\big(1 - H(x)\big), & y = 0 \end{cases}$$

Understanding this Cost Function

Cost	0	∞	∞	0
$H(x)$	0	0	1	1
y	0	1	0	1

Log Function



Binary Classification

$$\text{Cost}(W)=\frac{1}{m}\sum c(H(x),y)$$

$$c(H(x),y)=\begin{cases}-\log\left(H(x)\right), & y=1\\ -\log\left(1-H(x)\right), & y=0\end{cases}$$

$$c(H(x),y)=-y\log\left(H(x)\right)-(1-y)\log\left(1-H(x)\right)$$

$$Cost(W)=-\frac{1}{m}\sum y\log\left(H(x)\right)+(1-y)\log\left(1-H(x)\right)$$

Gradient Descent Method

$$W\leftarrow W-\alpha\frac{\partial}{\partial W}Cost(W)$$

Deep Learning Basics and Software
Binary Classification
Binary Classification Implementation

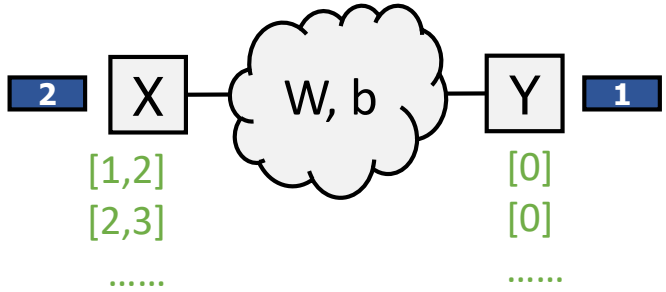
- Binary Classification Theory
- **Binary Classification Implementation**

Binary Classification Implementation (TensorFlow)

- TensorFlow
  - **Binary Classification**
  - Binary Classification (using CSV File)
- Keras
  - Binary Classification
  - Binary Classification (using CSV File)

Binary Classification Implementation (TensorFlow)

```
1  import tensorflow as tf
2
3  x_data = [[1,2], [2,3], [3,1], [4,3], [5,3], [6,2]]
4  y_data = [[0], [0], [0], [1], [1], [1]]
5  X = tf.placeholder(tf.float32, shape=[None, 2])
6  Y = tf.placeholder(tf.float32, shape=[None, 1])
7  W = tf.Variable(tf.random_normal([2,1]))
8  b = tf.Variable(tf.random_normal([1]))
9
10 model = tf.sigmoid(tf.add(tf.matmul(X,W),b))
11 cost = tf.reduce_mean((-1)*Y*tf.log(model) + (-1)*(1-Y)*tf.log(1-model))
12 train = tf.train.GradientDescentOptimizer(0.01).minimize(cost)
13
14 prediction = tf.cast(model > 0.5, dtype=tf.float32)
15 accuracy = tf.reduce_mean(tf.cast(tf.equal(prediction, Y), dtype=tf.float32))
16
17 with tf.Session() as sess:
18     sess.run(tf.global_variables_initializer())
19     # Training
20     for step in range(10001):
21         cost_val, train_val = sess.run([cost, train], feed_dict={X: x_data, Y: y_data})
22         print(step, cost_val)
23     # Testing
24     h, c, a = sess.run([model, prediction, accuracy], feed_dict={X: x_data, Y: y_data})
25     print("\nModel: ", h,"\nCorrect: ", c, "\nAccuracy: ", a)
```



Model, Cost, Train

Binary Classification Implementation (TensorFlow)

```
1  import tensorflow as tf
2
3  x_data = [[1,2], [2,3], [3,1],
4  y_data = [[0], [0], [0], [1],
5  X = tf.placeholder(tf.float32,
6  Y = tf.placeholder(tf.float32,
7  W = tf.Variable(tf.random_norma
8  b = tf.Variable(tf.random_norma
9
10 model = tf.sigmoid(tf.add(tf.ma
11 cost = tf.reduce_mean((-1)*Y*tf
12 train = tf.train.GradientDescer
13
14 prediction = tf.cast(model > 0.5, dtype=tf.float32)
15 accuracy = tf.reduce_mean(tf.cast(tf.equal(prediction, Y), dtype=tf.float32))
16
17 with tf.Session() as sess:
18     sess.run(tf.global_variables_initializer())
19     # Training
20     for step in range(10001):
21         cost_val, train_val = sess.run([cost, train], feed_dict={X: x_data, Y: y_data})
22         print(step, cost_val)
23     # Testing
24     h, c, a = sess.run([model, prediction, accuracy], feed_dict={X: x_data, Y: y_data})
25     print("\nModel: ", h,"\nCorrect: ", c, "\nAccuracy: ", a)
```

prediction

- Checking whether the first argument of **tf.cast** is true or not.
  - If true (model > 0.5), the result of **tf.cast** is **1.0**. Otherwise, the result of **tf.cast** is **0.0**.

accuracy

- **tf.equal()**
  - It returns **True** when **prediction == Y**. Otherwise, it returns **False**.
  - Accuracy: The average of T/F (= 1/0) values.

Binary Classification Implementation (TensorFlow)

```python
import tensorflow as tf

x_data = [[1,2], [2,3], [3,1], [4,3], [5,3], [6,2]]
y_data = [[0], [0], [0], [1], [1], [1]]
X = tf.placeholder(tf.float32, shape=[None, 2])
Y = tf.placeholder(tf.float32, shape=[None, 1])
W = tf.Variable(tf.random_normal([2,1]))
b = tf.Variable(tf.random_normal([1]))

model = tf.sigmoid(tf.add(tf.matmul(X,W),b))
cost = tf.reduce_mean((-1)*Y*tf.log(model) + (-1)*(1-Y)*tf.log(1-model))
train = tf.train.GradientDescentOptimizer(0.01).minimize(cost)

prediction = tf.cast(model > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(prediction, Y), dtype=tf.float32))

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    # Training
    for step in range(10001):
        cost_val, train_val = sess.run([cost, train], feed_dict={X: x_data, Y: y_data})
        print(step, cost_val)
    # Testing
    h, c, a = sess.run([model, prediction, accuracy], feed_dict={X: x_data, Y: y_data})
    print("\nModel: ", h,"\nCorrect: ", c, "\nAccuracy: ", a)
```

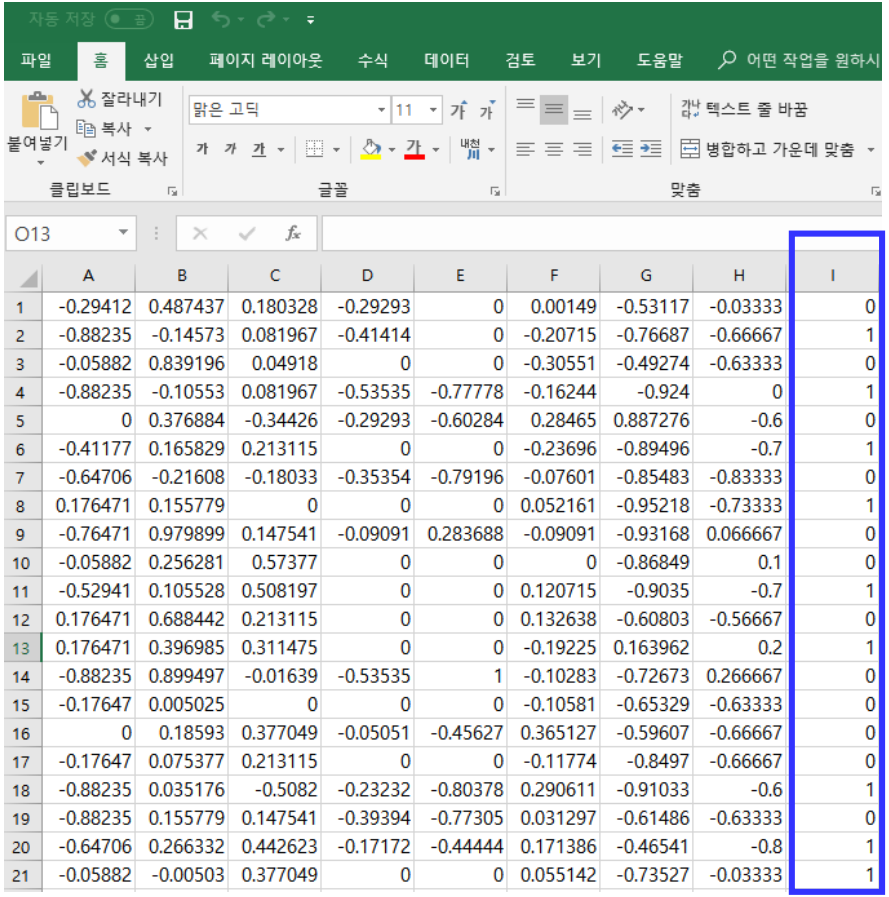
```
9985 0.1493395
9986 0.14932828
9987 0.14931704
9988 0.14930585
9989 0.14929464
9990 0.1492834
9991 0.14927219
9992 0.14926098
9993 0.14924978
9994 0.14923854
9995 0.14922734
9996 0.14921615
9997 0.14920495
9998 0.14919376
9999 0.1491826
10000 0.1491714

Model:  [[0.0306041 ]
 [0.15866211]
 [0.30421484]
 [0.7816807 ]
 [0.93976283]
 [0.9802319 ]]
Correct:  [[0.]
 [0.]
 [0.]
 [1.]
 [1.]
 [1.]]
Accuracy:  1.0
```

Binary Classification Implementation (TensorFlow)

- TensorFlow
  - Binary Classification
  - **Binary Classification (using CSV File)**
- Keras
  - Binary Classification
  - Binary Classification (using CSV File)

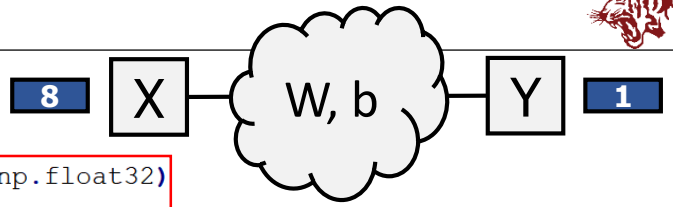
Binary Classification Implementation (TensorFlow)

CSV file



	A	B	C	D	E	F	G	H	I
1	-0.29412	0.487437	0.180328	-0.29293	0	0.00149	-0.53117	-0.03333	0
2	-0.88235	-0.14573	0.081967	-0.41414	0	-0.20715	-0.76687	-0.66667	1
3	-0.05882	0.839196	0.04918	0	0	-0.30551	-0.49274	-0.63333	0
4	-0.88235	-0.10553	0.081967	-0.53535	-0.77778	-0.16244	-0.924	0	1
5	0	0.376884	-0.34426	-0.29293	-0.60284	0.28465	0.887276	-0.6	0
6	-0.41177	0.165829	0.213115	0	0	-0.23696	-0.89496	-0.7	1
7	-0.64706	-0.21608	-0.18033	-0.35354	-0.79196	-0.07601	-0.85483	-0.83333	0
8	0.176471	0.155779	0	0	0	0.052161	-0.95218	-0.73333	1
9	-0.76471	0.979899	0.147541	-0.09091	0.283688	-0.09091	-0.93168	0.066667	0
10	-0.05882	0.256281	0.57377	0	0	0	-0.86849	0.1	0
11	-0.52941	0.105528	0.508197	0	0	0.120715	-0.9035	-0.7	1
12	0.176471	0.688442	0.213115	0	0	0.132638	-0.60803	-0.56667	0
13	0.176471	0.396985	0.311475	0	0	-0.19225	0.163962	0.2	1
14	-0.88235	0.899497	-0.01639	-0.53535	1	-0.10283	-0.72673	0.266667	0
15	-0.17647	0.005025	0	0	0	-0.10581	-0.65329	-0.63333	0
16	0	0.18593	0.377049	-0.05051	-0.45627	0.365127	-0.59607	-0.66667	0
17	-0.17647	0.075377	0.213115	0	0	-0.11774	-0.8497	-0.66667	0
18	-0.88235	0.035176	-0.5082	-0.23232	-0.80378	0.290611	-0.91033	-0.6	1
19	-0.88235	0.155779	0.147541	-0.39394	-0.77305	0.031297	-0.61486	-0.63333	0
20	-0.64706	0.266332	0.442623	-0.17172	-0.44444	0.171386	-0.46541	-0.8	1
21	-0.05882	-0.00503	0.377049	0	0	0.055142	-0.73527	-0.03333	1

Binary Classification Implementation (TensorFlow)




```
1  import tensorflow as tf
2  import numpy as np
3
4  xy = np.loadtxt('data-diabetes.csv', delimiter=',', dtype=np.float32)
5  x_data = xy[:, 0:-1]
6  y_data = xy[:, [-1]]
7
8  X = tf.placeholder(tf.float32, shape=[None, x_data.shape[1]])
9  Y = tf.placeholder(tf.float32, shape=[None, 1])
10 W = tf.Variable(tf.random_normal([x_data.shape[1], 1]))
11 b = tf.Variable(tf.random_normal([1]))
12
13 model = tf.sigmoid(tf.matmul(X, W) + b)
14 cost = tf.reduce_mean((-1)*Y*tf.log(model) + (-1)*(1-Y)*tf.log(1-model))
15 train = tf.train.GradientDescentOptimizer(0.01).minimize(cost)
16
17 prediction = tf.cast(model > 0.5, dtype=tf.float32)
18 accuracy = tf.reduce_mean(tf.cast(tf.equal(prediction, Y), dtype=tf.float32))
19
20 with tf.Session() as sess:
21     sess.run(tf.global_variables_initializer())
22     # Training
23     for step in range(100001):
24         c, _ = sess.run([cost, train], feed_dict={X: x_data, Y: y_data})
25         print(step, c)
26     # Testing
27     h, c, a = sess.run([model, prediction, accuracy], feed_dict={X: x_data, Y: y_data})
28     print("\nHypothesis: ", h, "\nCorrect (Y): ", c, "\nAccuracy: ", a)
```

CSV data loading

x_data.shape[1] == 8

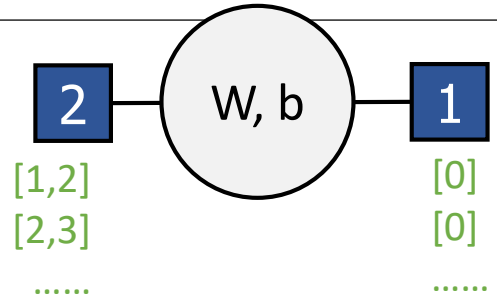
Model, Cost, Train

```
 [0.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]]
Accuracy:  0.76943344
```

Binary Classification Implementation (Keras)

- TensorFlow
  - Binary Classification
  - Binary Classification (using CSV File)
- Keras
  - **Binary Classification**
  - Binary Classification (using CSV File)

Binary Classification Implementation (Keras)




```
1  import numpy as np
2  from keras.models import Sequential
3  from keras.layers import Dense
4
5  # Data
6  x_data = np.array([[1,2], [2,3], [3,1], [4,3], [5,3], [6,2]])
7  y_data = np.array([[0], [0], [0], [1], [1], [1]])
8  # Model, Cost, Train
9  model = Sequential()
10 model.add(Dense(1, activation='sigmoid'))
11 model.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])
12 model.fit(x_data, y_data, epochs=10000, verbose=1)
13 model.summary()
14 # Inference
15 print(model.get_weights())
16 print(model.predict(x_data))
```

Model, Cost, Train

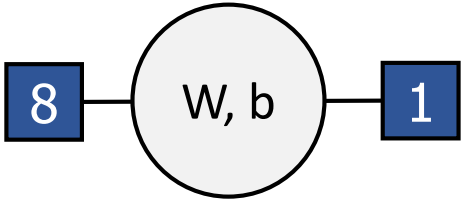
Binary Classification Implementation (Keras)

```
joongheon@joongheon-AB350M-Gaming-3: ~/Dropbox/codes_keras
6/6 [==============================] - 0s 107us/step - loss: 0.1504 - acc: 1.0000
Epoch 9992/10000
6/6 [==============================] - 0s 116us/step - loss: 0.1504 - acc: 1.0000
Epoch 9993/10000
6/6 [==============================] - 0s 96us/step - loss: 0.1504 - acc: 1.0000
Epoch 9994/10000
6/6 [==============================] - 0s 102us/step - loss: 0.1504 - acc: 1.0000
Epoch 9995/10000
6/6 [==============================] - 0s 97us/step - loss: 0.1504 - acc: 1.0000
Epoch 9996/10000
6/6 [==============================] - 0s 101us/step - loss: 0.1504 - acc: 1.0000
Epoch 9997/10000
6/6 [==============================] - 0s 102us/step - loss: 0.1504 - acc: 1.0000
Epoch 9998/10000
6/6 [==============================] - 0s 101us/step - loss: 0.1503 - acc: 1.0000
Epoch 9999/10000
6/6 [==============================] - 0s 104us/step - loss: 0.1503 - acc: 1.0000
Epoch 10000/10000
6/6 [==============================] - 0s 102us/step - loss: 0.1503 - acc: 1.0000
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1)                 3
=================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
_________________________________________________________________
[array([[1.4663278],
       [0.3097024]], dtype=float32), array([-5.5251], dtype=float32)]
[[0.03108752]
 [0.15931448]
 [0.30652532]
 [0.780626  ]
 [0.9390975 ]
 [0.98000884]]
```

Binary Classification Implementation (Keras)

- TensorFlow
  - Binary Classification
  - Binary Classification (using CSV File)
- Keras
  - Binary Classification
  - **Binary Classification (using CSV File)**

Binary Classification Implementation (Keras)



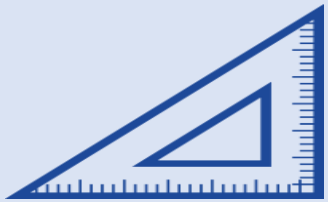

```
1  import numpy as np
2  from keras.models import Sequential
3  from keras.layers import Dense
4
5  # Data
6  xy = np.loadtxt('data-diabetes.csv', delimiter=',', dtype=np.float32)
7  x_data = xy[:, 0:-1]
8  y_data = xy[:, [-1]]
9
10 # Model, Cost, Train
11 model = Sequential()
12 model.add(Dense(1, activation='sigmoid'))
13 model.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])
14 model.fit(x_data, y_data, epochs=1000, verbose=1)
15 model.summary()
```

Model, Cost, Train

Binary Classification Implementation (Keras)

```
joongheon@joongheon-AB350M-Gaming-3: ~/Dropbox/codes_keras
759/759 [==============================] - 0s 18us/step - loss: 0.4725 - acc: 0.7694
Epoch 988/1000
759/759 [==============================] - 0s 23us/step - loss: 0.4724 - acc: 0.7694
Epoch 989/1000
759/759 [==============================] - 0s 23us/step - loss: 0.4724 - acc: 0.7694
Epoch 990/1000
759/759 [==============================] - 0s 21us/step - loss: 0.4724 - acc: 0.7694
Epoch 991/1000
759/759 [==============================] - 0s 20us/step - loss: 0.4724 - acc: 0.7708
Epoch 992/1000
759/759 [==============================] - 0s 23us/step - loss: 0.4724 - acc: 0.7694
Epoch 993/1000
759/759 [==============================] - 0s 21us/step - loss: 0.4724 - acc: 0.7708
Epoch 994/1000
759/759 [==============================] - 0s 24us/step - loss: 0.4724 - acc: 0.7708
Epoch 995/1000
759/759 [==============================] - 0s 24us/step - loss: 0.4724 - acc: 0.7708
Epoch 996/1000
759/759 [==============================] - 0s 22us/step - loss: 0.4724 - acc: 0.7708
Epoch 997/1000
759/759 [==============================] - 0s 22us/step - loss: 0.4724 - acc: 0.7694
Epoch 998/1000
759/759 [==============================] - 0s 21us/step - loss: 0.4724 - acc: 0.7694
Epoch 999/1000
759/759 [==============================] - 0s 21us/step - loss: 0.4724 - acc: 0.7708
Epoch 1000/1000
759/759 [==============================] - 0s 24us/step - loss: 0.4724 - acc: 0.7708
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1)                 9
=================================================================
Total params: 9
Trainable params: 9
Non-trainable params: 0
_________________________________________________________________
```

Linear Functions



Linear Regression
Binary Classification
Softmax Classification

Nonlinear Functions

Neural Network (NN)
Convolutional NN (CNN)
CNN for CIFAR-10
Recurrent NN (RNN)

Advanced Topics

Gen. Adv. Network (GAN)
Interpolation
PCA/LDA
Overfitting

Deep Learning Basics and Software
Softmax Classification
Softmax Classification Theory

- **Softmax Classification Theory**
- Softmax Classification Implementation

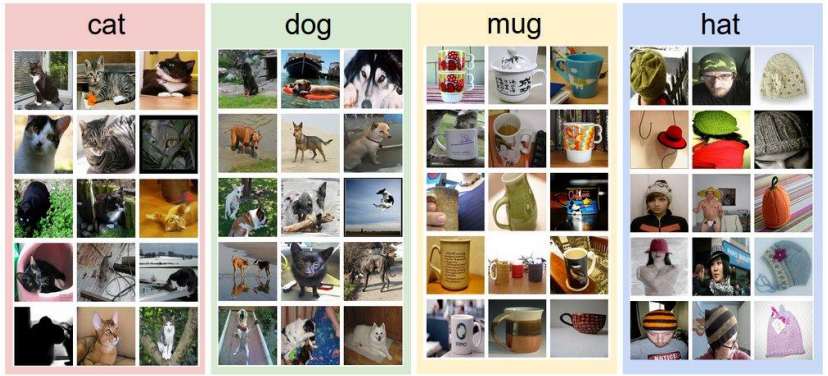
Regression and Classification

Regression (Examples)

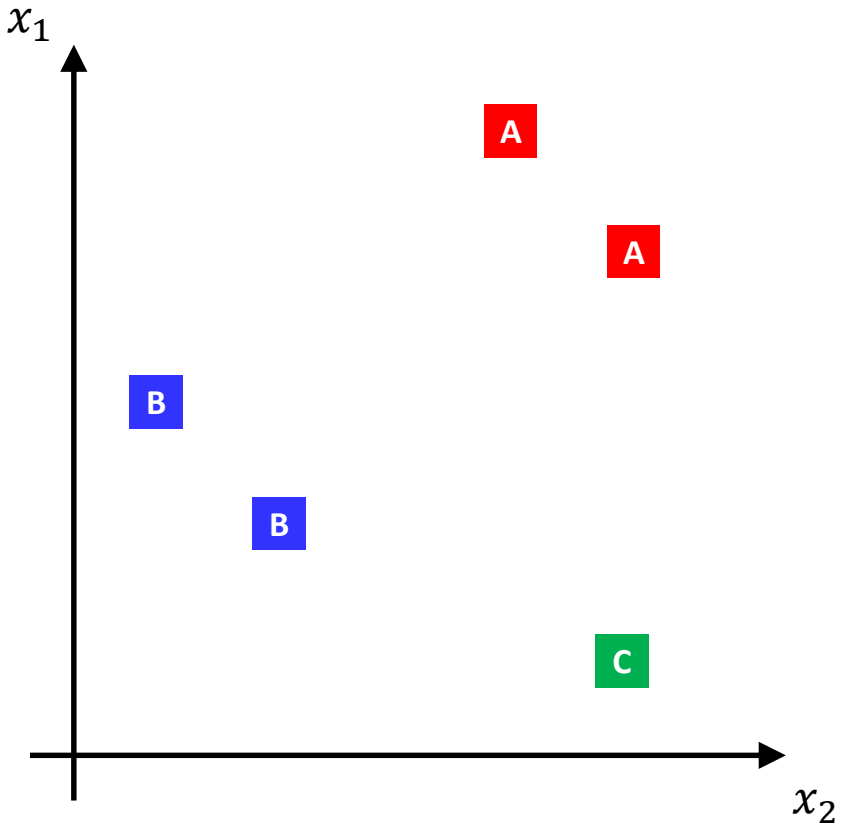
- Exam Score Prediction (Linear Regression)

Classification (Examples)

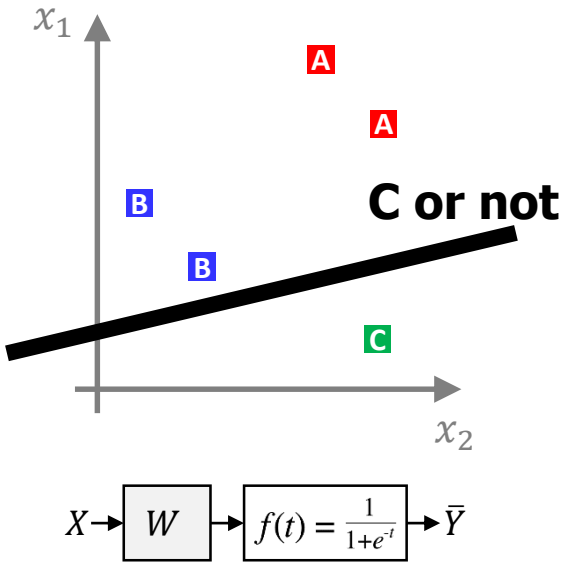
- Pass/Fail (Binary Classification)
- Letter Grades (Multi-Level Classification)

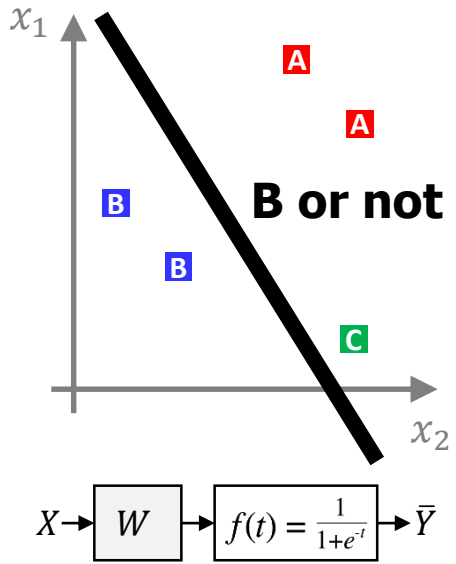


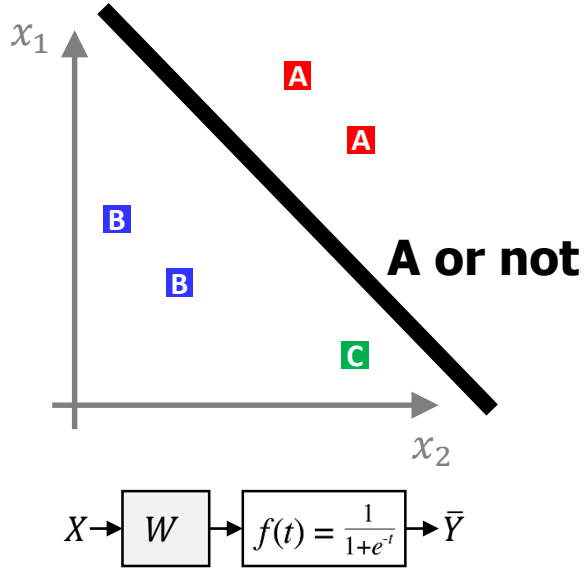
Multinomial Classification (Softmax Classification)



Multinomial Classification (Softmax Classification)

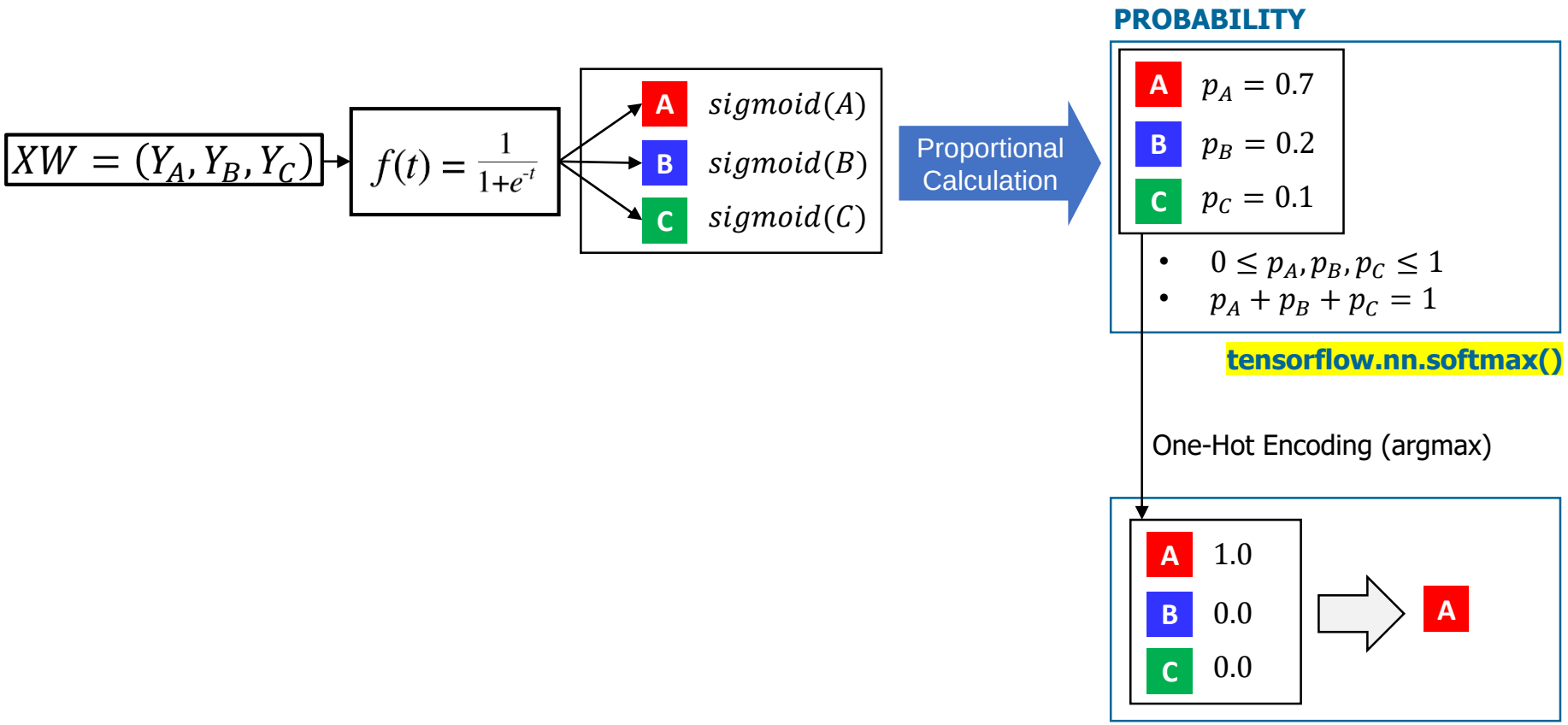






$$\begin{pmatrix} x_1 & x_2 \end{pmatrix} \cdot \begin{pmatrix} \overset{A}{W_{A1}} & \overset{B}{W_{B1}} & \overset{C}{W_{C1}} \\ W_{A2} & W_{B2} & W_{C2} \end{pmatrix} = \begin{pmatrix} \overset{A}{x_1 \cdot W_{A1} + x_2 \cdot W_{A2}} & \overset{B}{x_1 \cdot W_{B1} + x_2 \cdot W_{B2}} & \overset{C}{x_1 \cdot W_{C1} + x_2 \cdot W_{C2}} \end{pmatrix}$$

Multinomial Classification (Softmax Classification)



- Cost Function: **Cross-Entropy**

$S(Y) = \bar{Y}$ (estimated)
0.7
0.2
0.1

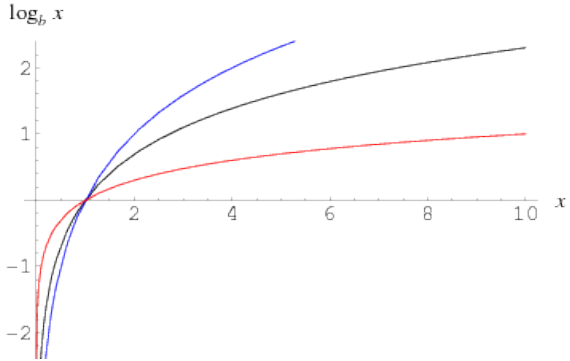
$$C(S, L) = -\sum_{\forall i} L_i \log(S_i)$$

$L = Y$ (real)
1.0
0.0
0.0

Understanding this Cost Function

L	S	Cost
[1,0,0]	[1,0,0]	$-1 \cdot \log 1 - 0 \cdot \log 0 - 0 \cdot \log 0 = 0$
	[0,1,0]	$-1 \cdot \log 0 - 0 \cdot \log 1 - 0 \cdot \log 0 = \infty$
	[0,0,1]	$-1 \cdot \log 0 - 0 \cdot \log 0 - 0 \cdot \log 1 = \infty$

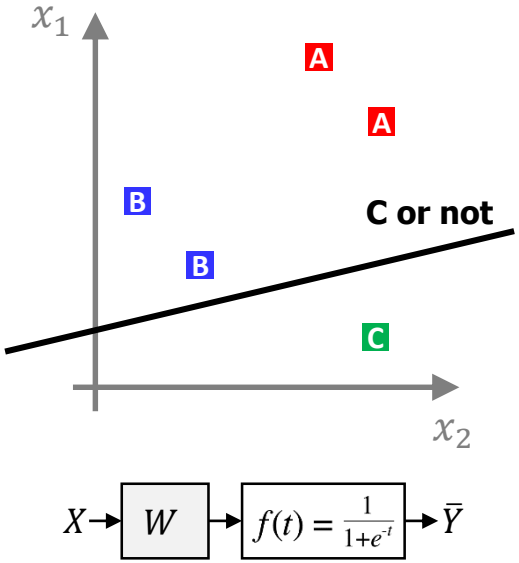
Log Function

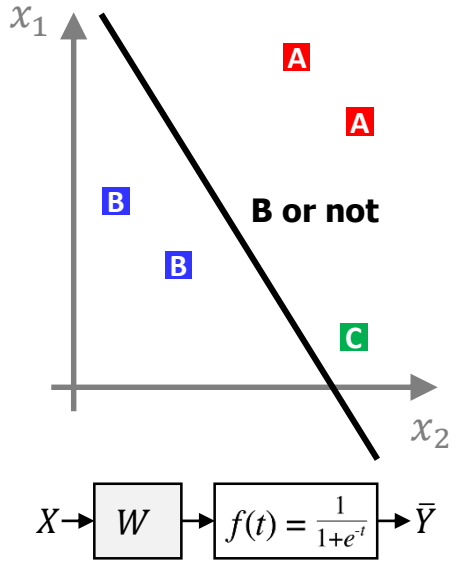


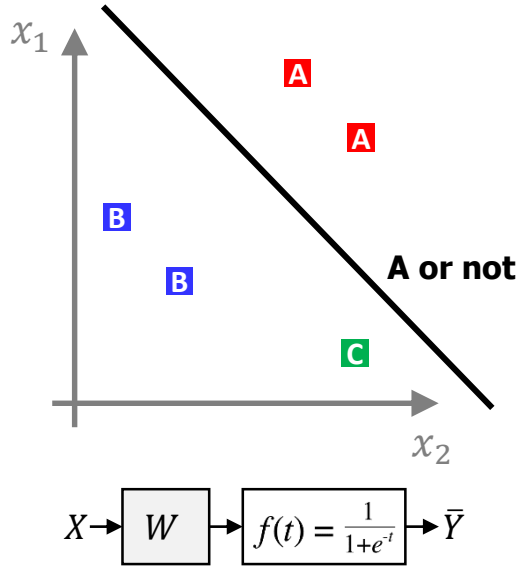
Deep Learning Basics and Software
Softmax Classification
Softmax Classification Implementation

- Softmax Classification Theory
- **Softmax Classification Implementation**

- Implementation Example







- Implementation Example
  - Vector Representation

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 \end{pmatrix} \cdot \begin{pmatrix} W_{A1} & W_{B1} & W_{C1} \\ W_{A2} & W_{B2} & W_{C2} \\ W_{A3} & W_{B3} & W_{C3} \\ W_{A4} & W_{B4} & W_{C4} \end{pmatrix} = \begin{pmatrix} S_A & S_B & S_C \end{pmatrix}$$

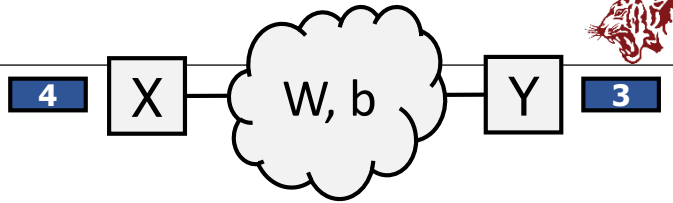
A B C

$$S_A \triangleq x_1 \cdot W_{A1} + x_2 \cdot W_{A2} + x_3 \cdot W_{A3} + x_4 \cdot W_{A4}$$
$$S_B \triangleq x_1 \cdot W_{B1} + x_2 \cdot W_{B2} + x_3 \cdot W_{B3} + x_4 \cdot W_{B4}$$
$$S_C \triangleq x_1 \cdot W_{C1} + x_2 \cdot W_{C2} + x_3 \cdot W_{C3} + x_4 \cdot W_{C4}$$

Softmax Classification Implementation (TensorFlow)

- TensorFlow
  - **Softmax Classification (with Given Cost Function)**
- Keras
  - Softmax Classification

Softmax Classification Implementation (TensorFlow)




```python
import tensorflow as tf

x_data = [[1,2,1,1], [2,1,3,2], [3,1,3,4], [4,1,5,5], [1,7,5,5], [1,2,5,6], [1,6,6,6], [1,7,7,7]] # vectors
y_data = [[0,0,1], [0,0,1], [0,0,1], [0,1,0], [0,1,0], [0,1,0], [1,0,0], [1,0,0]] # one hot encoding
X = tf.placeholder(tf.float32, shape=[None, 4])
Y = tf.placeholder(tf.float32, shape=[None, 3])
W = tf.Variable(tf.random_normal([4, 3]))
b = tf.Variable(tf.random_normal([3]))

model_LC = tf.add(tf.matmul(X,W),b)
model = tf.nn.softmax(model_LC)
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=model_LC, labels=Y))
train = tf.train.GradientDescentOptimizer(0.1).minimize(cost)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    # Training
    for step in range(2001):
        c, _ = sess.run([cost, train], feed_dict={X: x_data, Y: y_data})
        print(step, c)
    # Testing
    test1 = sess.run(model, feed_dict={X: [[1,11,7,9]]})
    print(test1, sess.run(tf.argmax(test1, 1)))
```

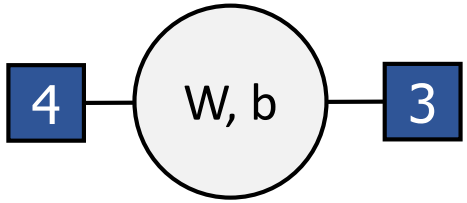
Model, Cost, Train

```
1988 0.16142774
1989 0.16136909
1990 0.16131032
1991 0.16125184
1992 0.16119315
1993 0.16113463
1994 0.16107623
1995 0.16101775
1996 0.16095944
1997 0.1609011
1998 0.16084275
1999 0.16078432
2000 0.16072604
[[7.2217123e-03 9.9276876e-01 9.6337890e-06]] [1]
```

Softmax Classification Implementation (Keras)

- TensorFlow
  - Softmax Classification (with Given Cost Function)
- Keras
  - **Softmax Classification**

Softmax Classification Implementation (Keras)




```
1  import numpy as np
2  from keras.models import Sequential
3  from keras.layers import Dense
4
5  # Data
6  x_data = np.array([[1,2,1,1], [2,1,3,2], [3,1,3,4], [4,1,5,5], [1,7,5,5], [1,2,5,6], [1,6,6,6], [1,7,7,7]])
7  y_data = np.array([[0,0,1], [0,0,1], [0,0,1], [0,1,0], [0,1,0], [0,1,0], [1,0,0], [1,0,0]])
8
9  # Model, Cost, Train
10 model = Sequential()
11 model.add(Dense(3, activation='softmax'))
12 model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
13 model.fit(x_data, y_data, epochs=10000, verbose=1)
14 model.summary()
15 # Inference
16 y_predict = model.predict(np.array([[1,11,7,9]]))
17 print(y_predict)
18 print("argmax: ", np.argmax(y predict))
```

Model, Cost, Train

Softmax Classification Implementation (Keras)

```
joongheon@joongheon-AB350M-Gaming-3: ~/Dropbox/codes_keras
8/8 [==============================] - 0s 96us/step - loss: 0.2581 - acc: 1.0000
Epoch 9989/10000
8/8 [==============================] - 0s 89us/step - loss: 0.2581 - acc: 1.0000
Epoch 9990/10000
8/8 [==============================] - 0s 87us/step - loss: 0.2581 - acc: 1.0000
Epoch 9991/10000
8/8 [==============================] - 0s 76us/step - loss: 0.2580 - acc: 1.0000
Epoch 9992/10000
8/8 [==============================] - 0s 80us/step - loss: 0.2580 - acc: 1.0000
Epoch 9993/10000
8/8 [==============================] - 0s 69us/step - loss: 0.2580 - acc: 1.0000
Epoch 9994/10000
8/8 [==============================] - 0s 69us/step - loss: 0.2580 - acc: 1.0000
Epoch 9995/10000
8/8 [==============================] - 0s 68us/step - loss: 0.2580 - acc: 1.0000
Epoch 9996/10000
8/8 [==============================] - 0s 74us/step - loss: 0.2580 - acc: 1.0000
Epoch 9997/10000
8/8 [==============================] - 0s 69us/step - loss: 0.2580 - acc: 1.0000
Epoch 9998/10000
8/8 [==============================] - 0s 70us/step - loss: 0.2579 - acc: 1.0000
Epoch 9999/10000
8/8 [==============================] - 0s 66us/step - loss: 0.2579 - acc: 1.0000
Epoch 10000/10000
8/8 [==============================] - 0s 66us/step - loss: 0.2579 - acc: 1.0000
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 3)                 15
=================================================================
Total params: 15
Trainable params: 15
Non-trainable params: 0
_________________________________________________________________
[[1.3161632e-01 8.6831880e-01 6.4874497e-05]]
argmax:  1
```

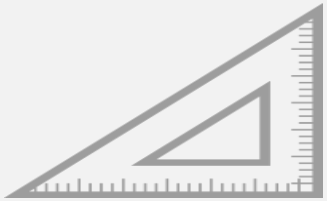
- Keras Tips
  - Parameter Setting Table

	[activation] setting in [Dense] keras.layers in [add] function	[loss] setting in [compile] function
Linear Regression		mse
Binary Classification	sigmoid	binary_crossentropy
Softmax Classification	softmax	categorical_crossentropy

  - Optimizer
    - **sgd** // stochastic gradient descent optimizer
    - **adam** // adam optimizer

Course Outline

Linear Functions



Linear Regression
Binary Classification
Softmax Classification

Nonlinear Functions

Neural Network (NN)
Convolutional NN (CNN)
CNN for CIFAR-10
Recurrent NN (RNN)

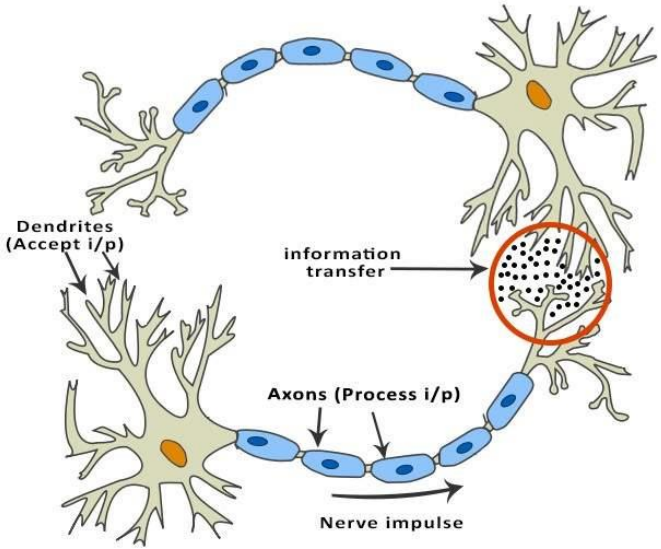
Advanced Topics

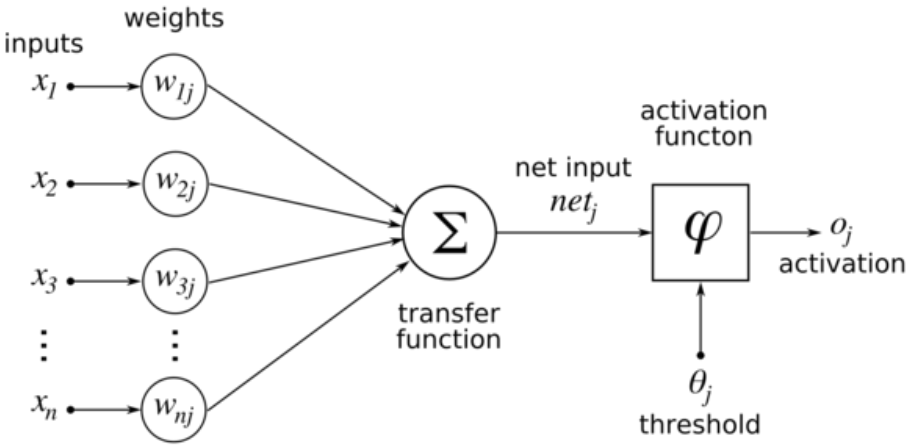
Gen. Adv. Network (GAN)
Interpolation
PCA/LDA
Overfitting

Deep Learning Basics and Software
Artificial Neural Networks (ANN)
ANN Theory

- **ANN Theory**
- ANN Implementation

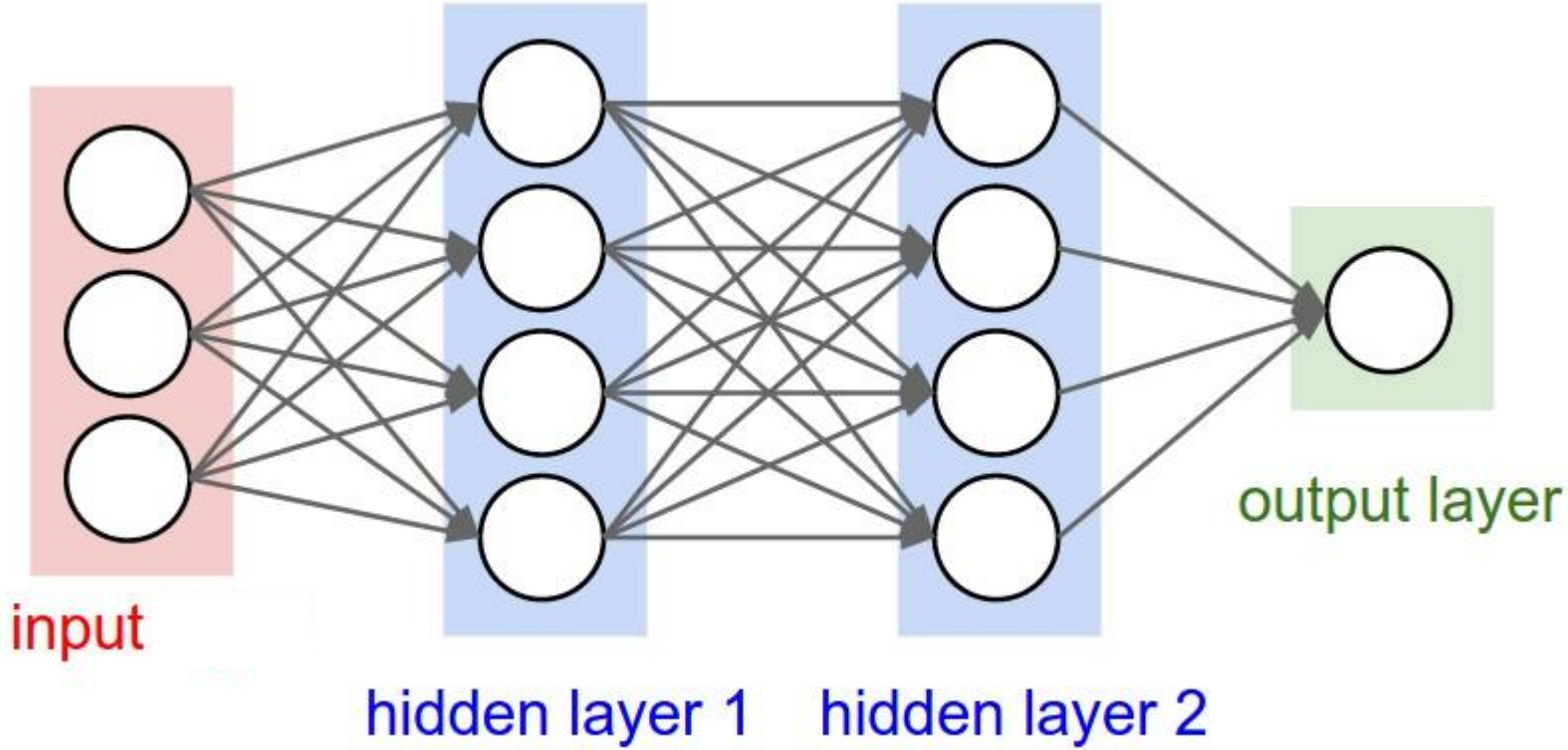
- Human Brain (Neuron)





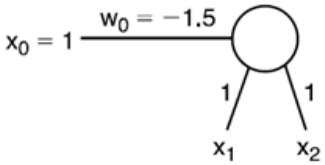
Binary Classification

Artificial Neural Networks (ANN): Multilayer Perceptron

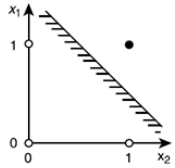


- Application to Logic Gate Design

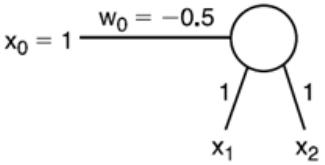
AND gate



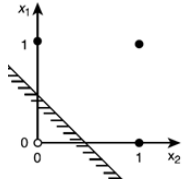
x_1	x_2	$W \cdot X$	y
0	0	−1.5	0
0	1	−0.5	0
1	0	−0.5	0
1	1	0.5	1



OR gate



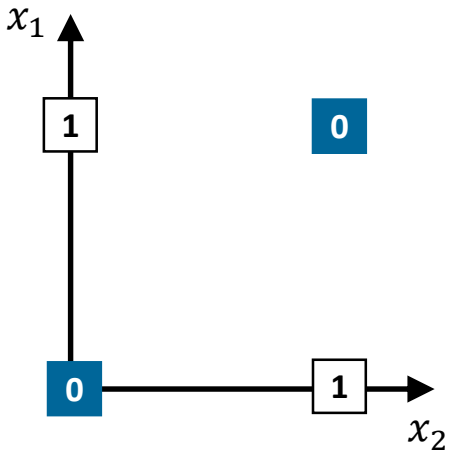
x_1	x_2	$W \cdot X$	y
0	0	−0.5	0
0	1	0.5	1
1	0	0.5	1
1	1	1.5	1



- **What about XOR?**

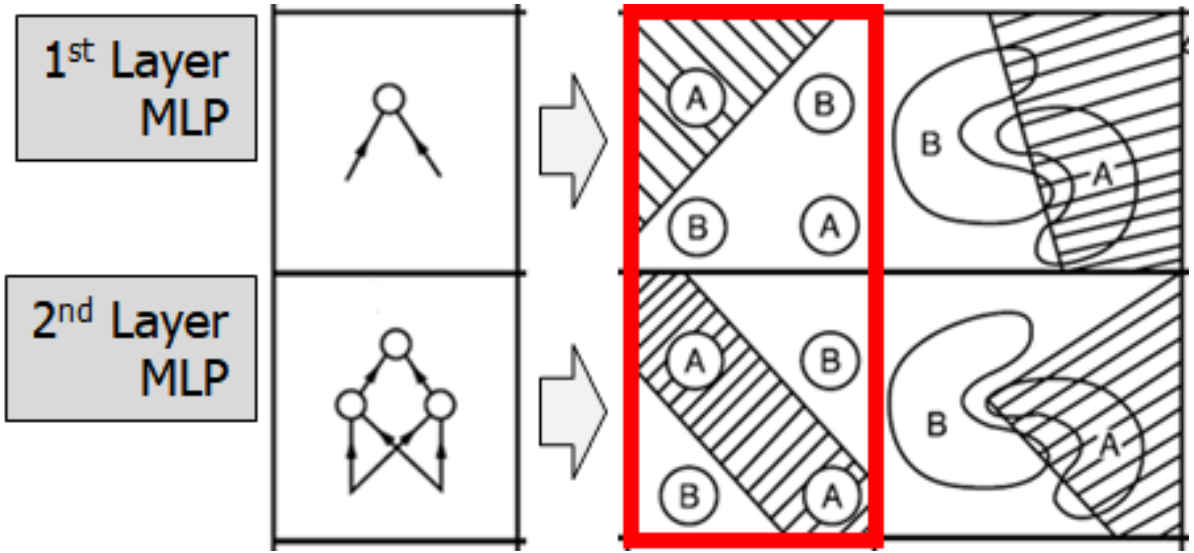
Artificial Neural Networks (ANN): Multilayer Perceptron

x_1	x_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0



Mathematically proven by
Prof. Marvin Minsky at MIT (1969)

- Multilayer Perceptron (MLP)
  - Proposed by Prof. Marvin Minsky at MIT (1969)
  - Can solve XOR Problem

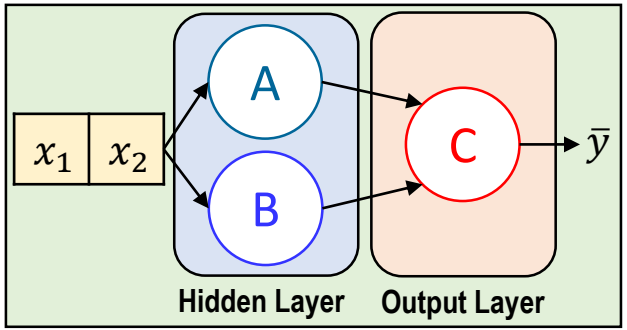


ANN: Solving XOR with MLP

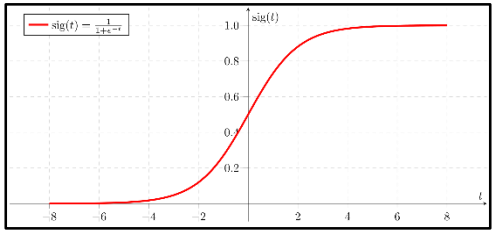
$W = \begin{pmatrix} 5 \\ 5 \end{pmatrix}, b = -8$

$W = \begin{pmatrix} -7 \\ -7 \end{pmatrix}, b = 3$

$W = \begin{pmatrix} -11 \\ -11 \end{pmatrix}, b = 6$

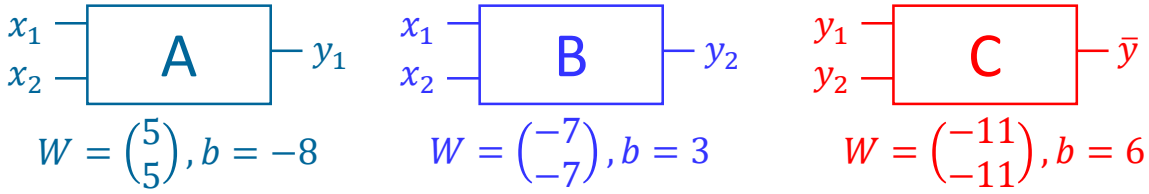


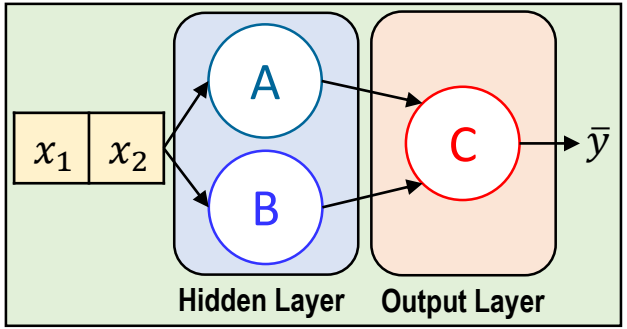
- $(x_1\ x_2) = (0\ 0)$
  - $(0\ 0)\begin{pmatrix} 5 \\ 5 \end{pmatrix} + (-8) = -8$, i.e., $y_1 = Sigmoid(-8) \cong 0$
  - $(0\ 0)\begin{pmatrix} -7 \\ -7 \end{pmatrix} + (3) = 3$, i.e., $y_2 = Sigmoid(3) \cong 1$
  - $(y_1\ y_2)\begin{pmatrix} -11 \\ -11 \end{pmatrix} + (6) = -11 + 6 = -5$, i.e., $\bar{y} = Sigmoid(-5) \cong 0$



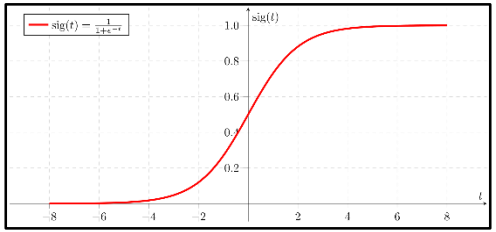
x_1	x_2	y_1	y_2	$\bar{y}$	XOR
0	0	**0**	**1**	**0**	**0**
0	1				**1**
1	0				**1**
1	1				**0**

ANN: Solving XOR with MLP



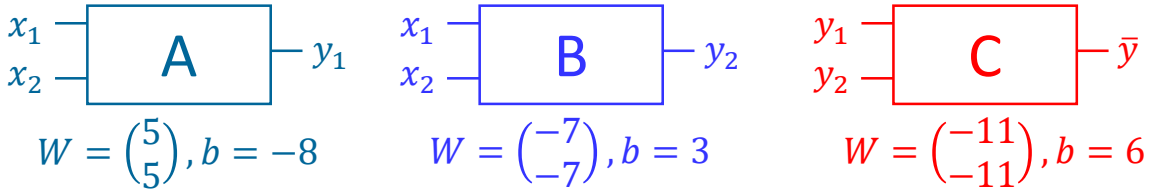


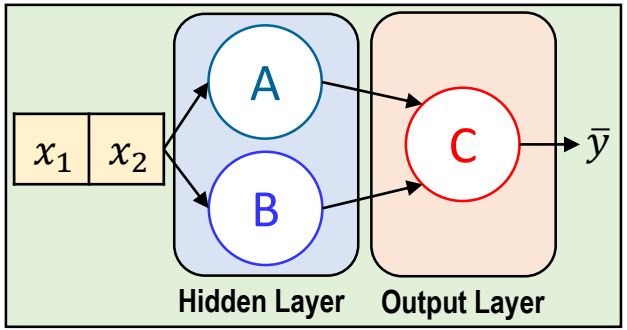
- $(x_1\ x_2) = (0\ 1)$
  - $(0\ 1)\begin{pmatrix} 5 \\ 5 \end{pmatrix} + (-8) = -3$, i.e., $y_1 = Sigmoid(-3) \cong 0$
  - $(0\ 1)\begin{pmatrix} -7 \\ -7 \end{pmatrix} + (3) = -4$, i.e., $y_2 = Sigmoid(-4) \cong 0$
  - $(y_1\ y_2)\begin{pmatrix} -11 \\ -11 \end{pmatrix} + (6) = 6$, i.e., $\bar{y} = Sigmoid(6) \cong 1$



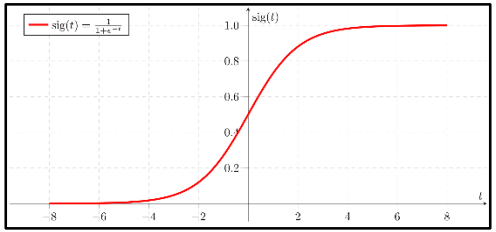
x_1	x_2	y_1	y_2	$\bar{y}$	XOR
0	0	**0**	**1**	**0**	**0**
0	1	**0**	**0**	**1**	**1**
1	0				**1**
1	1				**0**

ANN: Solving XOR with MLP





- $(x_1\ x_2) = (1\ 0)$
  - $(1\ 0)\begin{pmatrix} 5 \\ 5 \end{pmatrix} + (-8) = -3$, i.e., $y_1 = Sigmoid(-3) \cong 0$
  - $(1\ 0)\begin{pmatrix} -7 \\ -7 \end{pmatrix} + (3) = -4$, i.e., $y_2 = Sigmoid(-4) \cong 0$
  - $(y_1\ y_2)\begin{pmatrix} -11 \\ -11 \end{pmatrix} + (6) = 6$, i.e., $\bar{y} = Sigmoid(6) \cong 1$



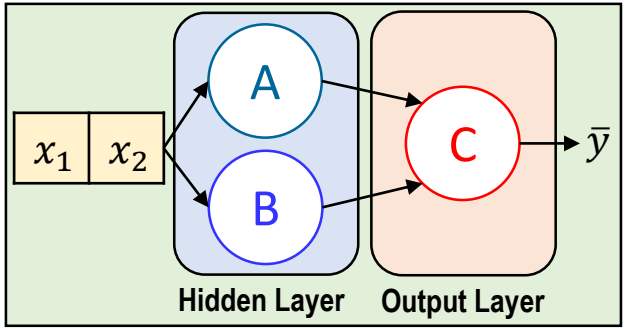
x_1	x_2	y_1	y_2	$\bar{y}$	XOR
0	0	**0**	**1**	**0**	**0**
0	1	**0**	**0**	**1**	**1**
1	0	**0**	**0**	**1**	**1**
1	1				**0**

ANN: Solving XOR with MLP

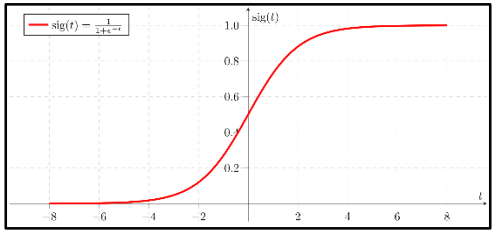
$W = \begin{pmatrix} 5 \\ 5 \end{pmatrix}, b = -8$

$W = \begin{pmatrix} -7 \\ -7 \end{pmatrix}, b = 3$

$W = \begin{pmatrix} -11 \\ -11 \end{pmatrix}, b = 6$

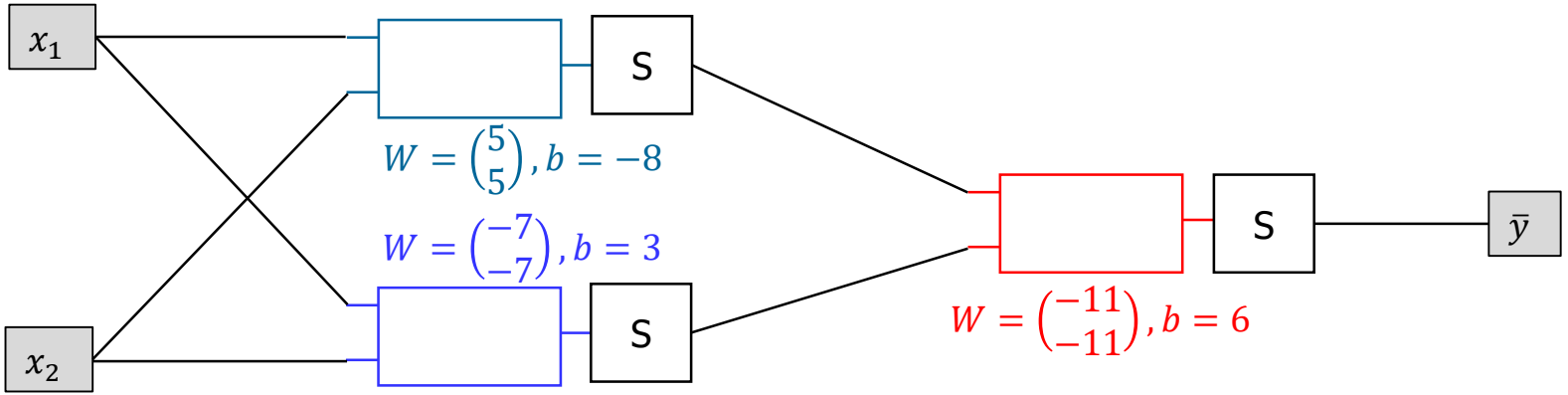


- $(x_1\ x_2) = (1\ 1)$
  - $(1\ 1)\begin{pmatrix} 5 \\ 5 \end{pmatrix} + (-8) = 2$, i.e., $y_1 = Sigmoid(2) \cong 1$
  - $(1\ 1)\begin{pmatrix} -7 \\ -7 \end{pmatrix} + (3) = -11$, i.e., $y_2 = Sigmoid(-11) \cong 0$
  - $(y_1\ y_2)\begin{pmatrix} -11 \\ -11 \end{pmatrix} + (6) = -5$, i.e., $\bar{y} = Sigmoid(-5) \cong 0$



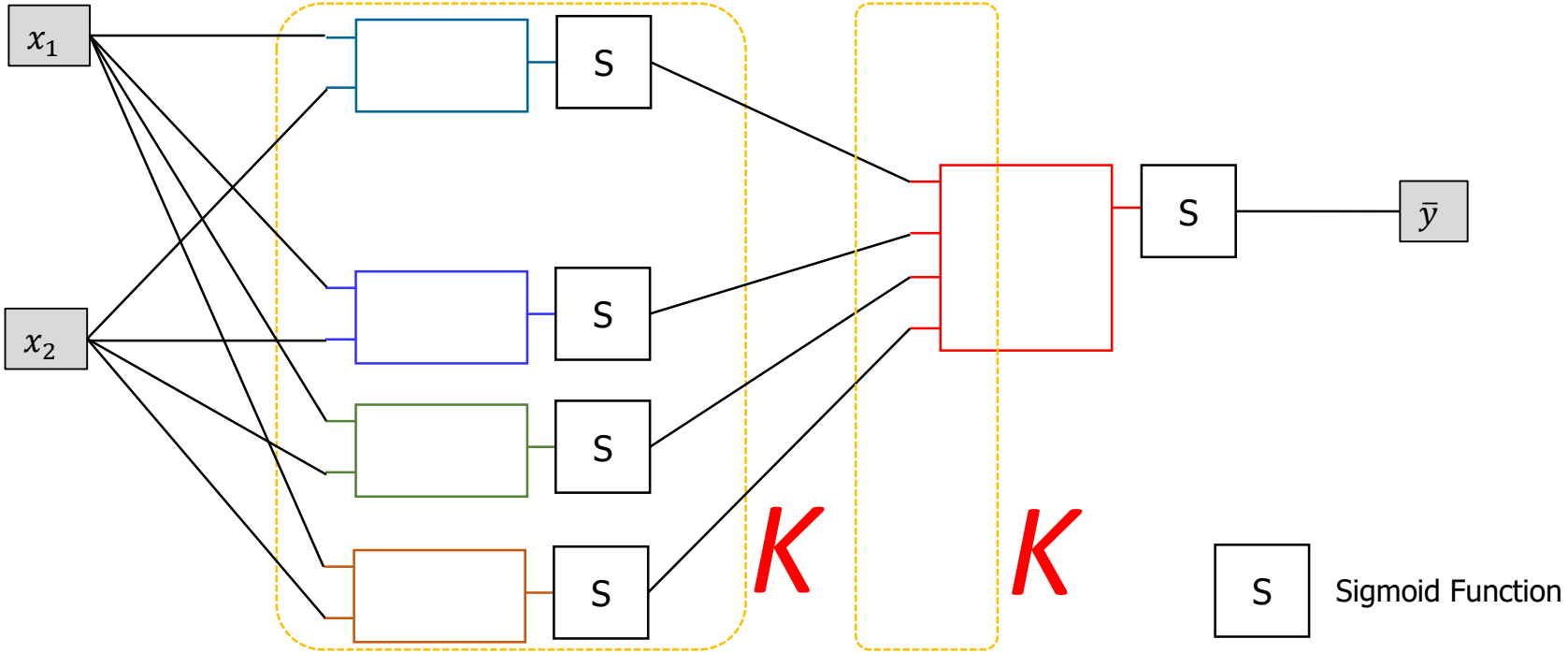
x_1	x_2	y_1	y_2	$\bar{y}$	XOR
0	0	0	1	0	0
0	1	0	0	1	1
1	0	0	0	1	1
1	1	1	0	0	0

ANN: Solving XOR with MLP (Forward Propagation)

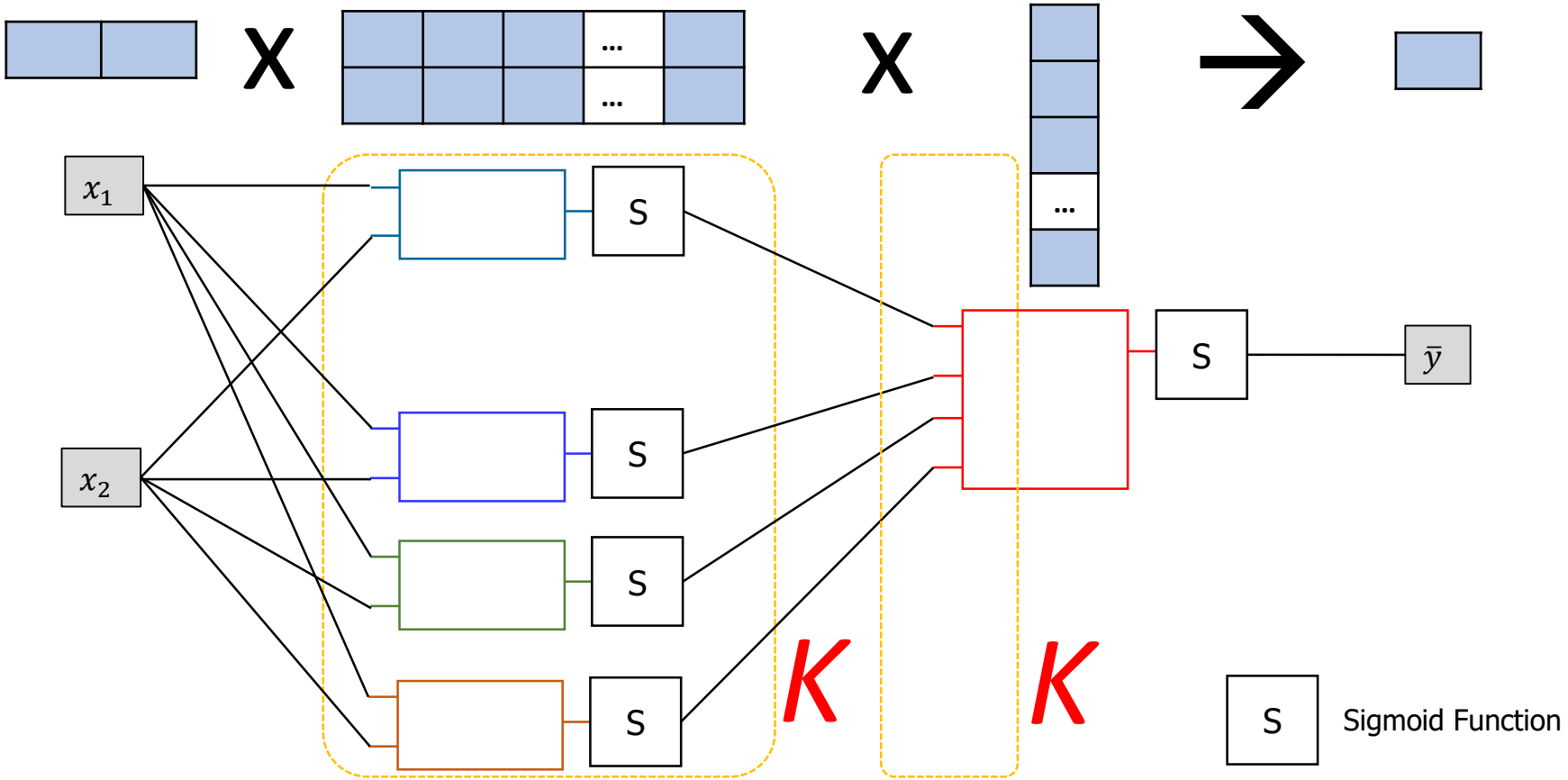


S Sigmoid Function

ANN: Solving XOR with MLP (Forward Propagation)



ANN: Solving XOR with MLP (Forward Propagation)



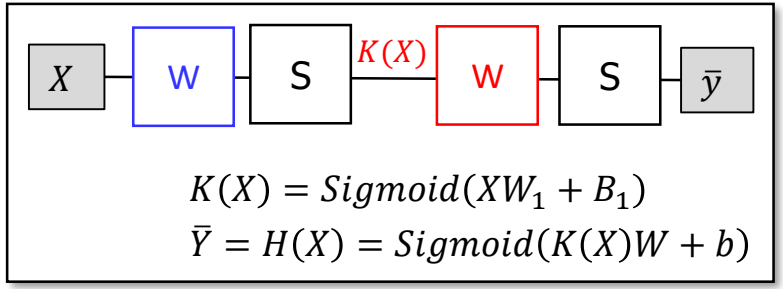
TensorFlow for ANN (XOR with Binary Classification)

```
0 1.0826586
2000 0.6931472
4000 0.6931472
6000 0.6931472
8000 0.6931472
10000 0.6931472
12000 0.6931472
14000 0.6931472
16000 0.6931472
18000 0.6931472
20000 0.6931472
[[0.5]
 [0.5]
 [0.5]
 [0.5]] [[0.]
 [0.]
 [0.]
 [0.]] 0.5
```

```python
import tensorflow as tf

x_data = [[0,0], [0,1], [1,0], [1,1]]
y_data = [[0], [1], [1], [0]]
X = tf.placeholder(tf.float32, shape=[None, 2])
Y = tf.placeholder(tf.float32, shape=[None, 1])
W = tf.Variable(tf.random_normal([2,1]))
b = tf.Variable(tf.random_normal([1]))
model = tf.sigmoid(tf.matmul(X,W)+b)
cost = tf.reduce_mean((-1)*Y*tf.log(model) + (-1)*(1-Y)*tf.log(1-model))
train = tf.train.GradientDescentOptimizer(0.1).minimize(cost)

prediction = tf.cast(model > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(prediction, Y), dtype=tf.float32))

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    # Training
    for step in range(20001):
        c, _ = sess.run([cost, train], feed_dict={X: x_data, Y: y_data})
        if step % 2000 == 0:
            print(step, c)
    # Testing
    m, p, a = sess.run([model, prediction, accuracy], feed_dict={X: x_data, Y: y_data})
    print(m,p,a)
```

TensorFlow for ANN (XOR with ANN)

```python
import tensorflow as tf

x_data = [[0,0], [0,1], [1,0], [1,1]]
y_data = [[0], [1], [1], [0]]
X = tf.placeholder(tf.float32, shape=[None, 2])
Y = tf.placeholder(tf.float32, shape=[None, 1])
W_h = tf.Variable(tf.random_normal([2,3]))
b_h = tf.Variable(tf.random_normal([3]))
H1 = tf.sigmoid(tf.matmul(X,W_h)+b_h)
W_o = tf.Variable(tf.random_normal([3,1]))
b_o = tf.Variable(tf.random_normal([1]))
model = tf.sigmoid(tf.matmul(H1,W_o)+b_o)
cost = tf.reduce_mean((-1)*Y*tf.log(model) + (-1)*(1-Y)*tf.log(1-model))
train = tf.train.GradientDescentOptimizer(0.1).minimize(cost)

prediction = tf.cast(model > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(prediction, Y), dtype=tf.float32))

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    # Training
    for step in range(20001):
        c, _ = sess.run([cost, train], feed_dict={X: x_data, Y: y_data})
        if step % 2000 == 0:
            print(step, c)
    # Testing
    m, p, a = sess.run([model, prediction, accuracy], feed_dict={X: x_data, Y: y_data})
    print(m,p,a)
```




```
0 0.86801255
2000 0.27334553
4000 0.046823796
6000 0.02246793
8000 0.0143708335
10000 0.010443565
12000 0.008153362
14000 0.0066633224
16000 0.0056207716
18000 0.0048525333
20000 0.004264111
[[0.00480547]
 [0.99502313]
 [0.9966924 ]
 [0.00392833]] [[0.]
 [1.]
 [1.]
 [0.]] 1.0
```

- **Wide** ANN for XOR

```
W1 = tf.Variable(tf.random_normal([2, 10]))
b1 = tf.Variable(tf.random_normal([10]))
H1 = tf.sigmoid(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([10, 1]))
b2 = tf.Variable(tf.random_normal([1]))
model = tf.sigmoid(tf.matmul(H1, W2) + b2)
```

[2,10], [10,1]

Hypothesis:
[[0.00358802]
 [0.99366933]
 [0.99204296]
 [0.0095663]]
Correct:
[[0.]
 [1.]
 [1.]
 [0.]]
Accuracy: 1.0

[2,2], [2,1]

Hypothesis:
[[0.01338218]
[0.98166394]
[0.98809403]
[0.01135799]]
Correct:
[[0.]
[1.]
[1.]
[0.]]
Accuracy: 1.0

- **Deep** ANN for XOR

```
W1 = tf.Variable(tf.random_normal([2, 10]))
b1 = tf.Variable(tf.random_normal([10]))
H1 = tf.sigmoid(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([10, 10]))
b2 = tf.Variable(tf.random_normal([10]))
H2 = tf.sigmoid(tf.matmul(H1, W2) + b2)

W3 = tf.Variable(tf.random_normal([10, 10]))
b3 = tf.Variable(tf.random_normal([10]))
H3 = tf.sigmoid(tf.matmul(H2, W3) + b3)

W4 = tf.Variable(tf.random_normal([10, 1]))
b4 = tf.Variable(tf.random_normal([1]))
model = tf.sigmoid(tf.matmul(H3, W4) + b4)
```

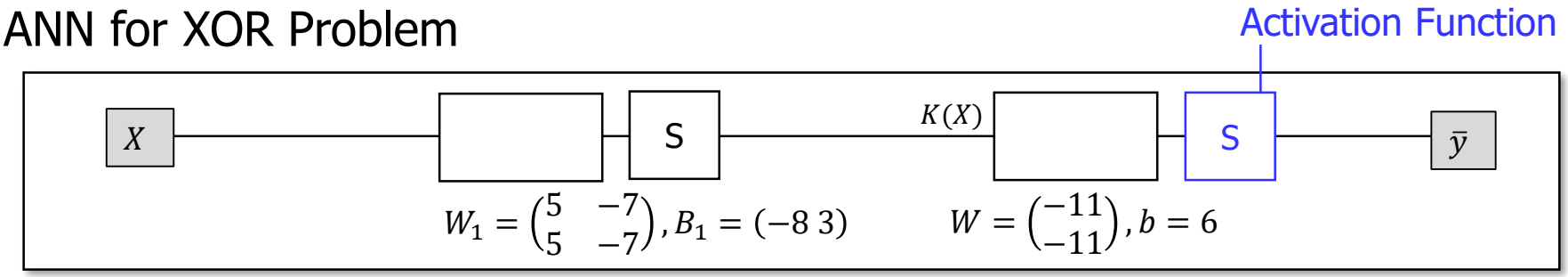
4 layers

Hypothesis:
[[7.80e-04]
 [9.99e-01]
 [9.98e-01]
 [1.55e-03]]
Correct:
[[0.]
 [1.]
 [1.]
 [0.]]
Accuracy: 1.0

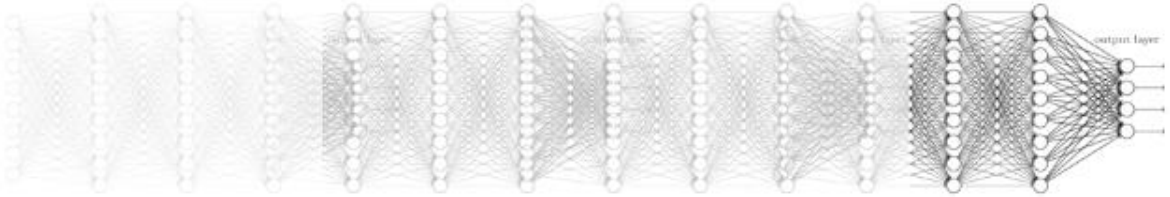
2 layers

Hypothesis:
[[0.01338218]
 [0.98166394]
 [0.98809403]
 [0.01135799]]
Correct:
[[0.]
 [1.]
 [1.]
 [0.]]
Accuracy: 1.0

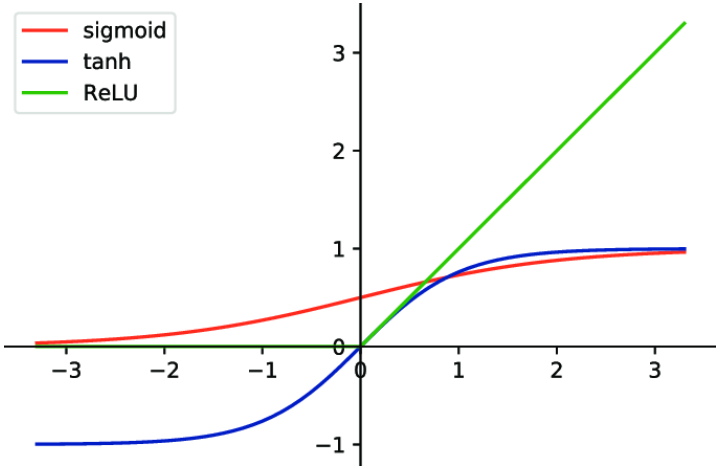
- ANN for XOR Problem

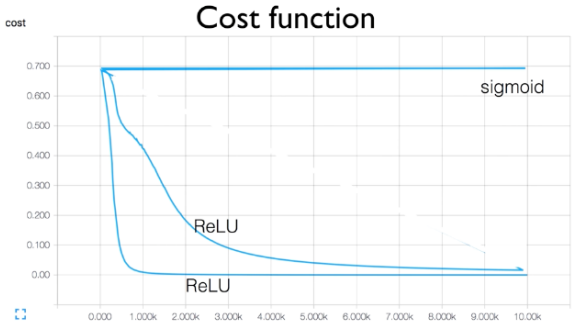


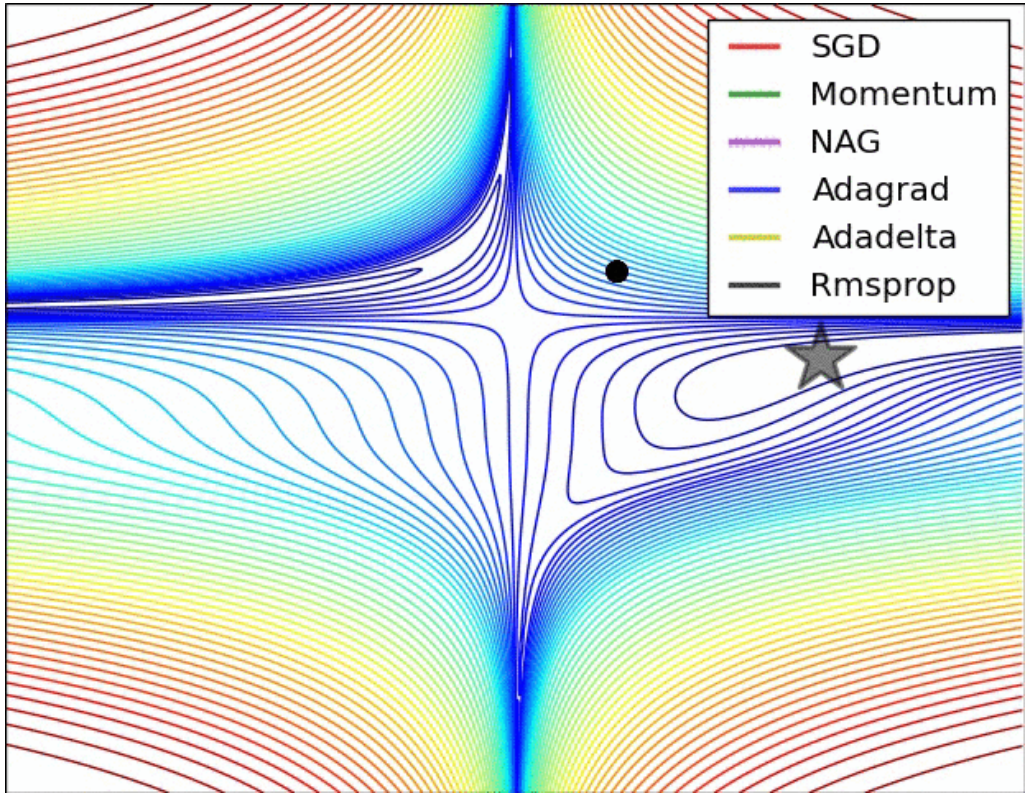
- Observation)
  - There exists cases when the accuracy is low even if the # layers is high. Why?
  - Answer)
    - The result of one ANN is the result of sigmoid function (**between 0 and 1**).
    - The numerous multiplication of this result converges to near zero.
      → **Gradient Vanishing Problem**



ANN: ReLU (Rectified Linear Unit)







- ReLU

```
# input place holders
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

# weights & bias for NN layers
W1 = tf.Variable(tf.random_normal([784, 256]))
b1 = tf.Variable(tf.random_normal([256]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([256, 256]))
b2 = tf.Variable(tf.random_normal([256]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)

W3 = tf.Variable(tf.random_normal([256, 10]))
b3 = tf.Variable(tf.random_normal([10]))

model_LC = tf.matmul(L2, W3) + b3
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=model_LC, labels=Y))
train = tf.train.AdamOptimizer(0.01).minimize(cost)
```

- Deep Learning Revolution is Real

Our **labeled datasets** were **too small**.

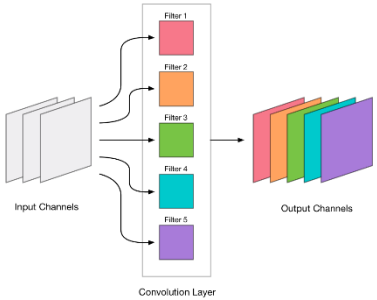
IMAGENET
14.2 million images

Big-Data

Our **computers** were millions of times **too slow**.

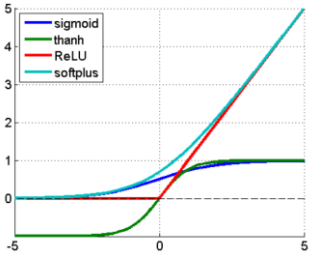
GPU

We only consider **one-dimensional vector** as an input.



Convolution Layers for Multi-Dimensional Inputs

We used the **wrong type of non-linearity (activation function).**



ReLU for solving Gradient Vanishing Problem

Deep Learning Theory and Software
Artificial Neural Networks (ANN)
ANN Implementation

- ANN Theory
- **ANN Implementation**

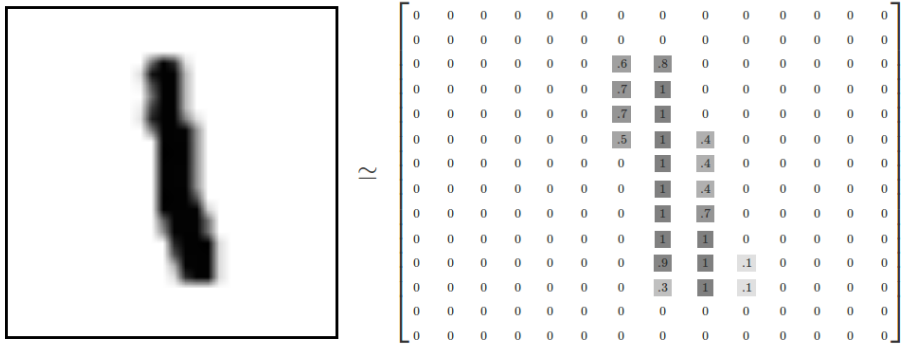
ANN Implementation (TensorFlow)

- TensorFlow
  - **MLP for MNIST**
  - MLP for MNIST (tensorflow.layers.dense)
- Keras
  - MLP for MNIST

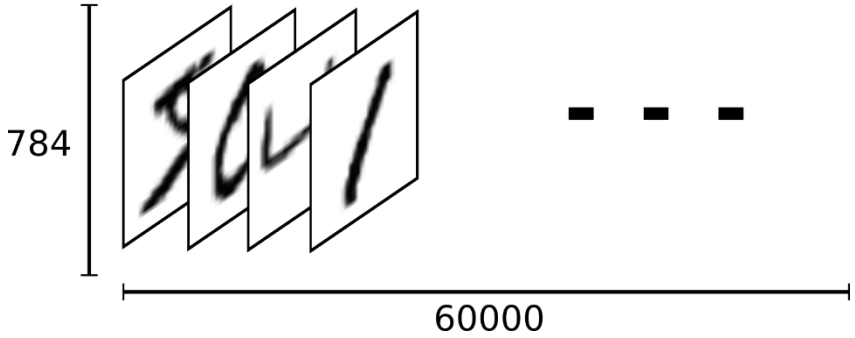
- MNIST Data Set
  - Hand written images and their labels
    - For training: 55,000
    - For testing: 5,000

[0,0,0,0,0,1,0,0,0,0] → '5'
[1,0,0,0,0,0,0,0,0,0] → '0'
[0,0,0,0,1,0,0,0,0,0] → '4'
[0,1,0,0,0,0,0,0,0,0] → '1'

  - Image: 28-by-28 (pixels)



mnist.train.xs



ANN Implementation (TensorFlow)

```
1  # MNIST
2  from tensorflow.examples.tutorials.mnist import input_data
3  mnist = input_data.read_data_sets("data_MNIST", one_hot = True)
4
5  # Setting
6  import tensorflow as tf
7  import time
8  num_steps = 5000
9  batch_size =128
10 nH1 = 256
11 nH2 = 256
12 nH3 = 256
13
14 X = tf.placeholder("float", [None, 784])
15 Y = tf.placeholder("float", [None, 10])
16
```

ANN Implementation (TensorFlow)

```
17  def mlp_LC(img):
18      HL1 = tf.nn.relu(tf.add(tf.matmul(img, W['HL1']), b['HL1']))
19      HL2 = tf.nn.relu(tf.add(tf.matmul(HL1, W['HL2']), b['HL2']))
20      HL3 = tf.nn.relu(tf.add(tf.matmul(HL2, W['HL3']), b['HL3']))
21      Out = tf.matmul(HL3, W['Out']) + b['Out']
22      return Out
23  W = {
24      'HL1' : tf.Variable(tf.random_normal([784, nH1])),
25      'HL2' : tf.Variable(tf.random_normal([nH1, nH2])),
26      'HL3' : tf.Variable(tf.random_normal([nH2, nH3])),
27      'Out' : tf.Variable(tf.random_normal([nH3, 10]))
28  }
29  b = {
30      'HL1' : tf.Variable(tf.random_normal([nH1])),
31      'HL2' : tf.Variable(tf.random_normal([nH2])),
32      'HL3' : tf.Variable(tf.random_normal([nH3])),
33      'Out' : tf.Variable(tf.random_normal([10]))
34  }
```

ANN Implementation (TensorFlow)

```
36  # Model, Cost, Train
37  model_LC = mlp_LC(X)
38  model = tf.nn.softmax(model_LC)
39  cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=model_LC, labels=Y))
40  train = tf.train.AdamOptimizer(0.01).minimize(cost)
41
42  # Accuracy Computation
43  accuracy = tf.reduce_mean(tf.cast(tf.equal(tf.argmax(model, 1), tf.argmax(Y,1)), tf.float32))
44
45  # Session
46  with tf.Session() as sess:
47      sess.run(tf.global_variables_initializer())
48      # Train
49      t1 = time.time()
50      for step in range(1, num_steps+1):
51          train_images, train_labels = mnist.train.next_batch(batch_size)
52          sess.run(train, feed_dict={X: train_images, Y: train_labels})
53          if step % 500 == 0:
54              print(step)
55      t2 = time.time()
56      print("Training Time (seconds): ", t2-t1)
57      print("Testing Accuracy: ", sess.run(accuracy, feed_dict={X: mnist.test.images, Y: mnist.test.labels}))
```

```
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
Training Time (seconds):  14.891910552978516
Testing Accuracy:  0.9581
```

- Line 48:
  - **batch_train_images:** matrix for **mnist.train.next_batch**: **(batch_size)-by-(784)** → ***image data***
  - **batch_train_labels:** matrix for **mnist.train.next_batch**: **(batch_size)-by-(10)** → ***image labels***
- Line 54:
  - **Testing** with **mnist.test.images** and **mnist.test.labels**

ANN Implementation (TensorFlow)

- TensorFlow
  - MLP for MNIST
  - **MLP for MNIST (tensorflow.layers.dense)**
- Keras
  - MLP for MNIST

ANN Implementation (TensorFlow)

```
17  def mlp_LC(img):
18      HL1 = tf.nn.relu(tf.add(tf.matmul(img, W['HL1']), b['HL1']))
19      HL2 = tf.nn.relu(tf.add(tf.matmul(HL1, W['HL2']), b['HL2']))
20      HL3 = tf.nn.relu(tf.add(tf.matmul(HL2, W['HL3']), b['HL3']))
21      Out = tf.matmul(HL3, W['Out']) + b['Out']
22      return Out
23  W = {
24      'HL1' : tf.Variable(tf.random_normal([784, nH1])),
25      'HL2' : tf.Variable(tf.random_normal([nH1, nH2])),
26      'HL3' : tf.Variable(tf.random_normal([nH2, nH3])),
27      'Out' : tf.Variable(tf.random_normal([nH3, 10]))
28  }
29  b = {
30      'HL1' : tf.Variable(tf.random_normal([nH1])),
31      'HL2' : tf.Variable(tf.random_normal([nH2])),
32      'HL3' : tf.Variable(tf.random_normal([nH3])),
33      'Out' : tf.Variable(tf.random_normal([10]))
34  }
```

tensorflow.layers.dense

```
17  def mlp_LC(img):
18      HL1 = tf.layers.dense(inputs=img, units=nH1, activation=tf.nn.relu)
19      HL2 = tf.layers.dense(inputs=HL1, units=nH2, activation=tf.nn.relu)
20      HL3 = tf.layers.dense(inputs=HL2, units=nH3, activation=tf.nn.relu)
21      Out = tf.layers.dense(inputs=HL3, units=10, activation=None)
22      return Out
```

ANN Implementation (TensorFlow)

- TensorFlow
  - MLP for MNIST
  - MLP for MNIST (tensorflow.layers.dense)
- Keras
  - **MLP for MNIST**

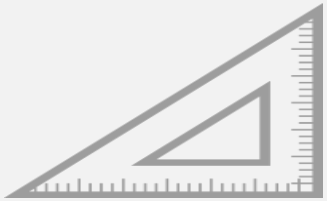
ANN Implementation (Keras)

```
1  from keras.utils import np_utils
2  from keras.datasets import mnist
3  from keras.models import Sequential
4  from keras.layers import Dense
5  # MNIST data
6  (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
7  print(train_images.shape, train_labels.shape, test_images.shape, test_labels.shape)
8  train_images = train_images.reshape(train_images.shape[0], 784).astype('float32')/255.0
9  test_images = test_images.reshape(test_images.shape[0], 784).astype('float32')/255.0
10 train_labels = np_utils.to_categorical(train_labels) # One-Hot Encoding
11 test_labels = np_utils.to_categorical(test_labels) # One-Hot Encoding
12 # Model
13 model = Sequential()
14 model.add(Dense(256, activation='relu')) # units=256, activation='relu'
15 model.add(Dense(256, activation='relu')) # units=256, activation='relu'
16 model.add(Dense(256, activation='relu')) # units=256, activation='relu'
17 model.add(Dense(10, activation='softmax')) # units=10, activation='softmax'
18 model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
19 # Training
20 model.fit(train_images, train_labels, epochs=5, batch_size=32, verbose=1)
21 # Testing
22 _, accuracy = model.evaluate(test_images, test_labels)
23 print('Accuracy: ', accuracy)
24 model.summary()
```

ANN Implementation (Keras)

```
(60000, 28, 28) (60000,) (10000, 28, 28) (10000,)
Epoch 1/5
2019-08-04 21:38:20.386123: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU
supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
60000/60000 [==============================] - 5s 87us/step - loss: 0.6131 - acc: 0.8331
Epoch 2/5
60000/60000 [==============================] - 5s 85us/step - loss: 0.2544 - acc: 0.9269
Epoch 3/5
60000/60000 [==============================] - 5s 85us/step - loss: 0.1971 - acc: 0.9426
Epoch 4/5
60000/60000 [==============================] - 5s 85us/step - loss: 0.1610 - acc: 0.9529
Epoch 5/5
60000/60000 [==============================] - 5s 85us/step - loss: 0.1356 - acc: 0.9608
10000/10000 [==============================] - 0s 32us/step
Accuracy:  0.9602

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 256)               200960
_________________________________________________________________
dense_2 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_3 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_4 (Dense)              (None, 10)                2570
=================================================================
Total params: 335,114
Trainable params: 335,114
Non-trainable params: 0
_________________________________________________________________
```

Linear Functions



Linear Regression
Binary Classification
Softmax Classification

Nonlinear Functions

Neural Network (NN)
Convolutional NN (CNN)
CNN for CIFAR-10
Recurrent NN (RNN)

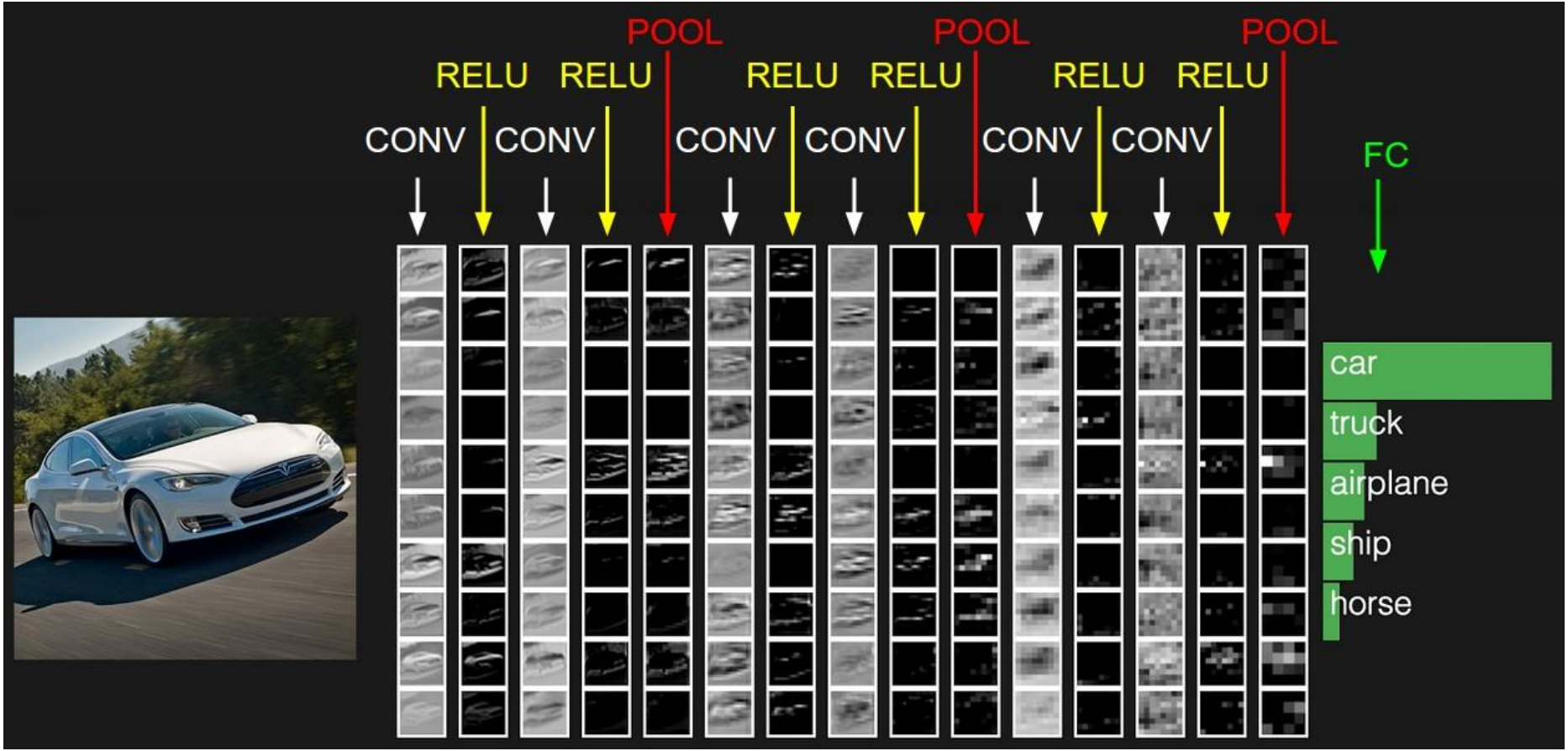
Advanced Topics

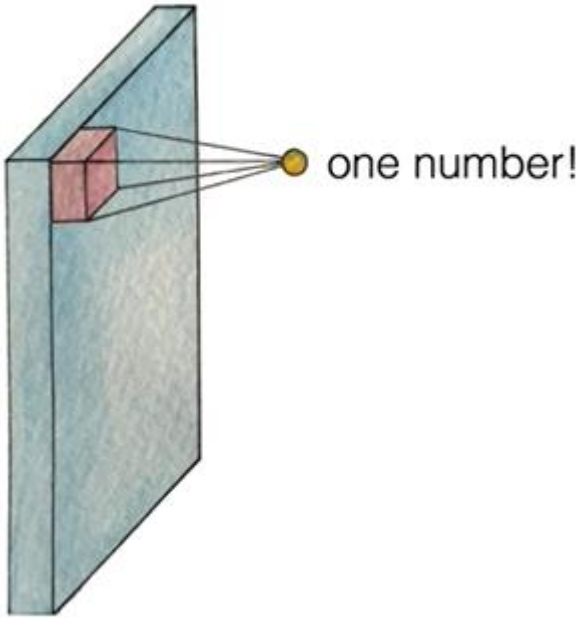
Gen. Adv. Network (GAN)
Interpolation
PCA/LDA
Overfitting

Deep Learning Theory and Software
Convolutional Neural Networks (CNN)
CNN Theory

- **CNN Theory**
- CNN Implementation

CNN: Convolution



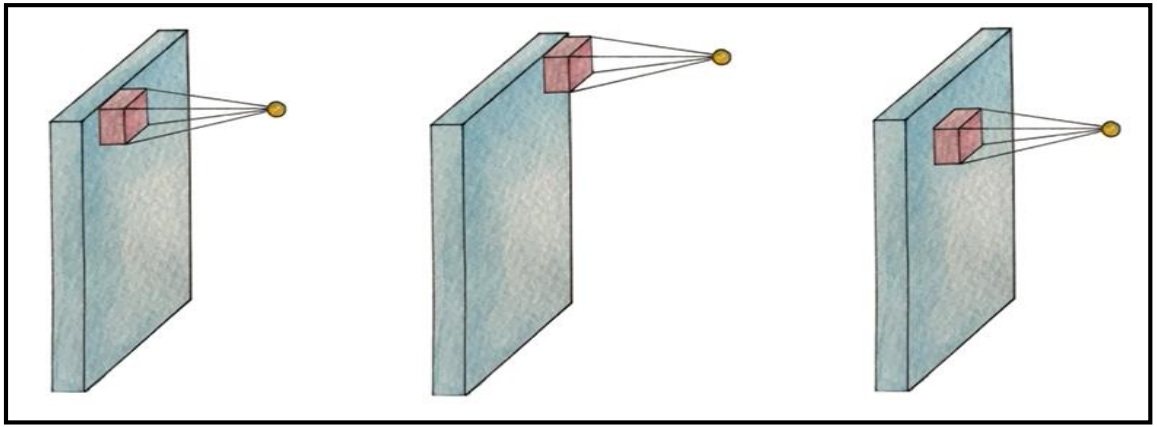


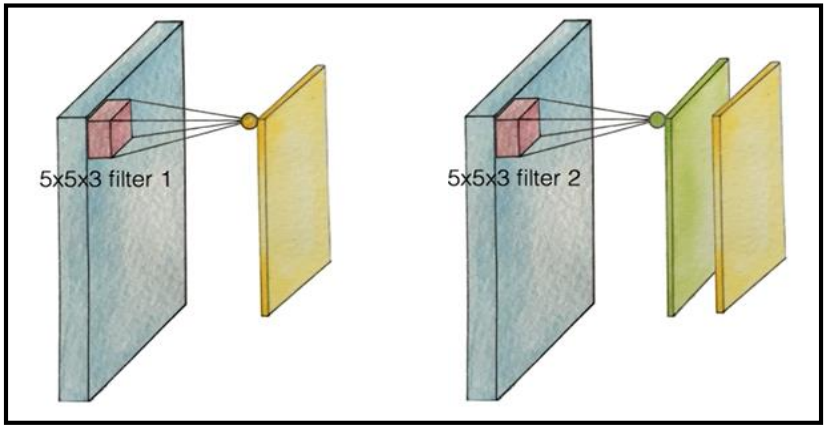
- Extracting **one point** with a filter

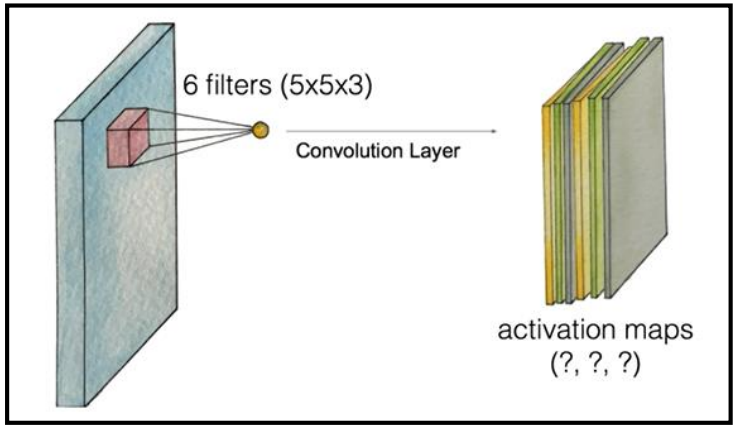
$$Wx + b$$

$$Y = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$$

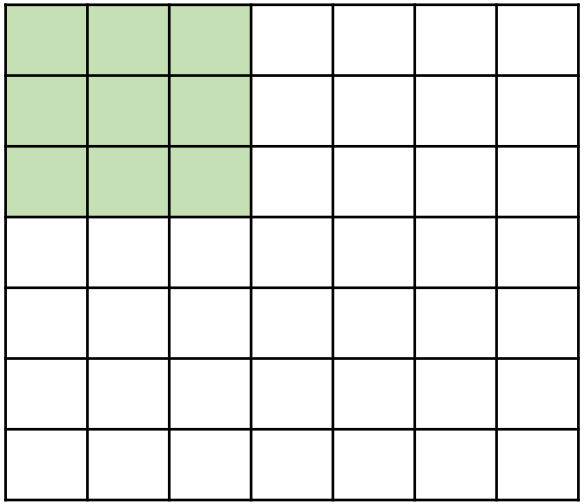
CNN: Convolution







CNN: Convolution



- Image Size: 7-by-7
- Filter Size: 3-by-3
- Strides (step-size of filter moving)
  - 1: 5*5=25
  - 2: 3*3=9
  - …
  - → Large strides loose information (value degrades).

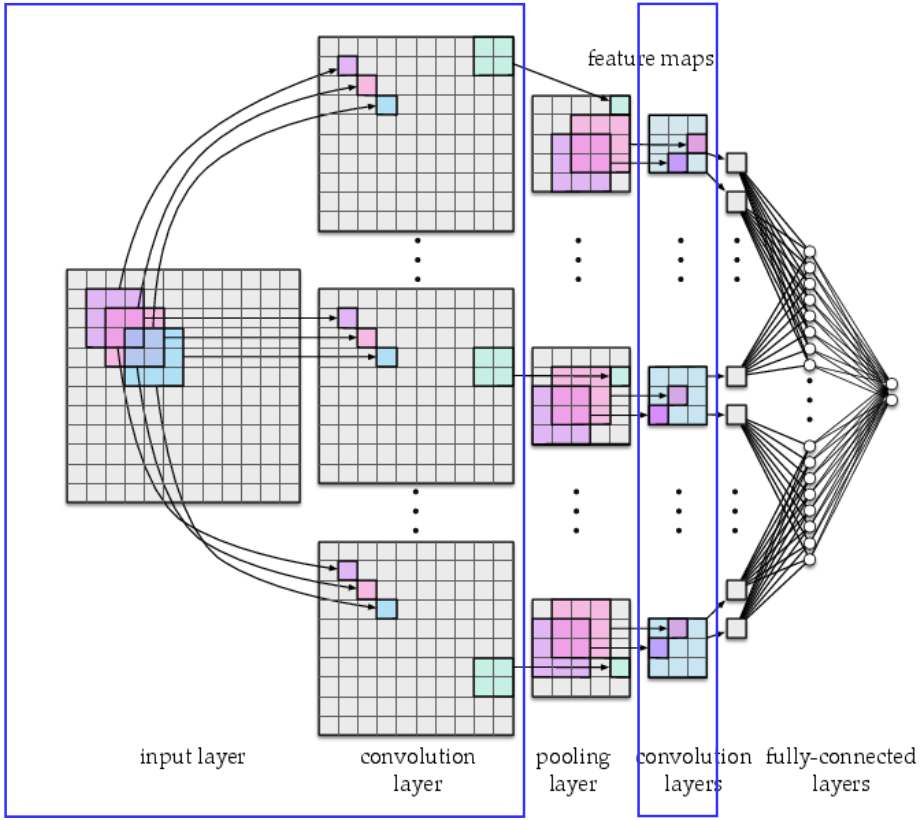
CNN: Convolution

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

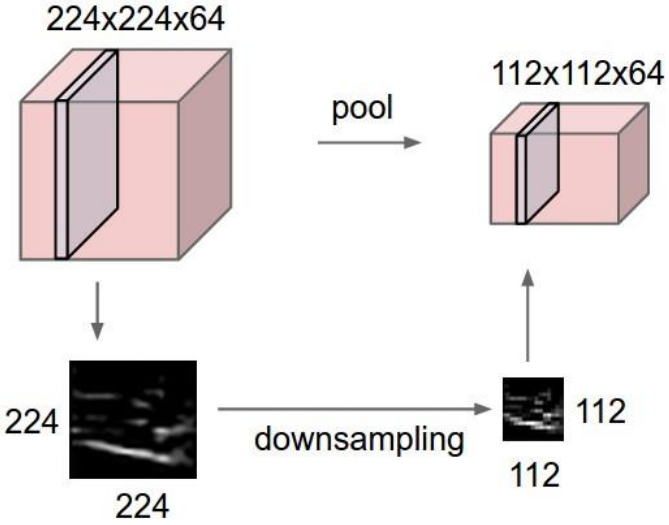
- Padding
  - Marking for the image areas
  - Preventing the case where the sizes of return images becomes smaller.

CNN: Convolution

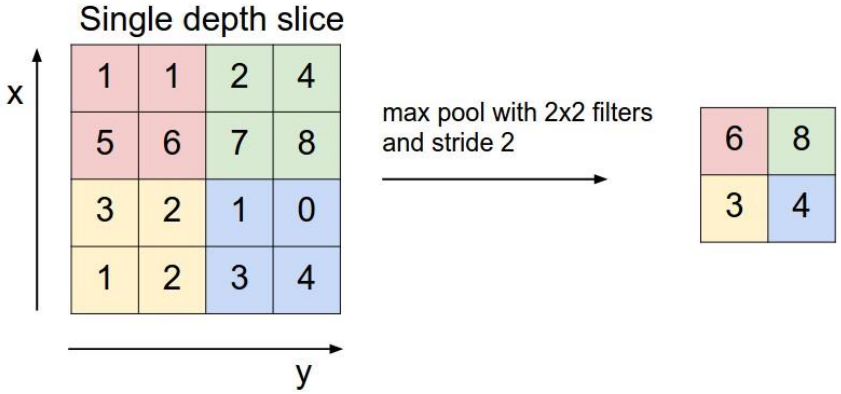
- From one image, multiple filters can be used for swiping the entire image.



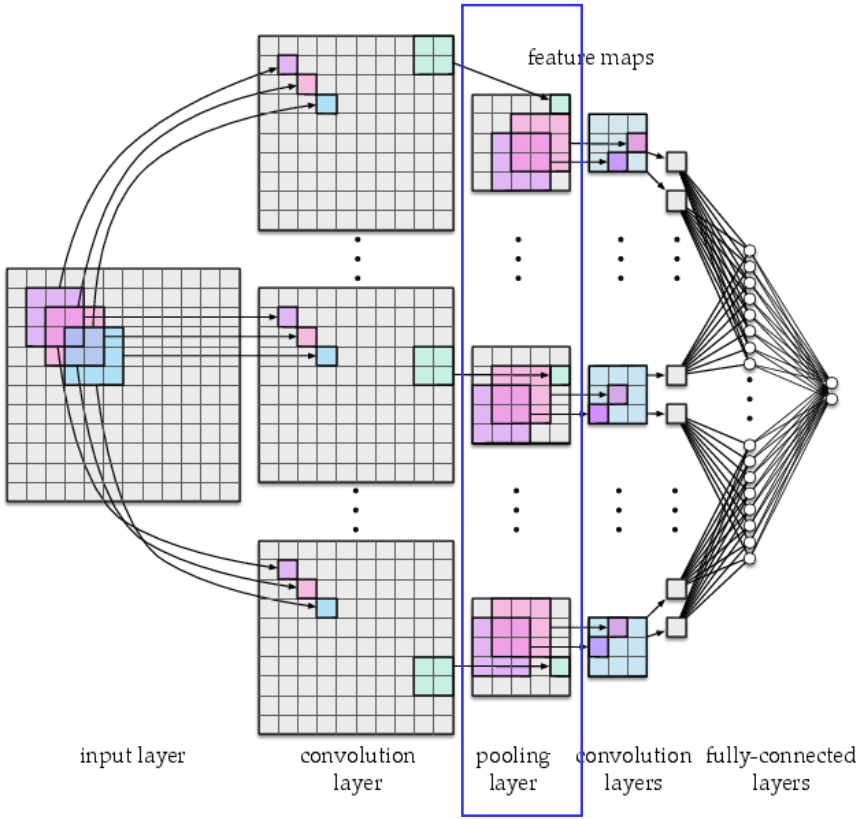
- Pooling Layer (Sampling)



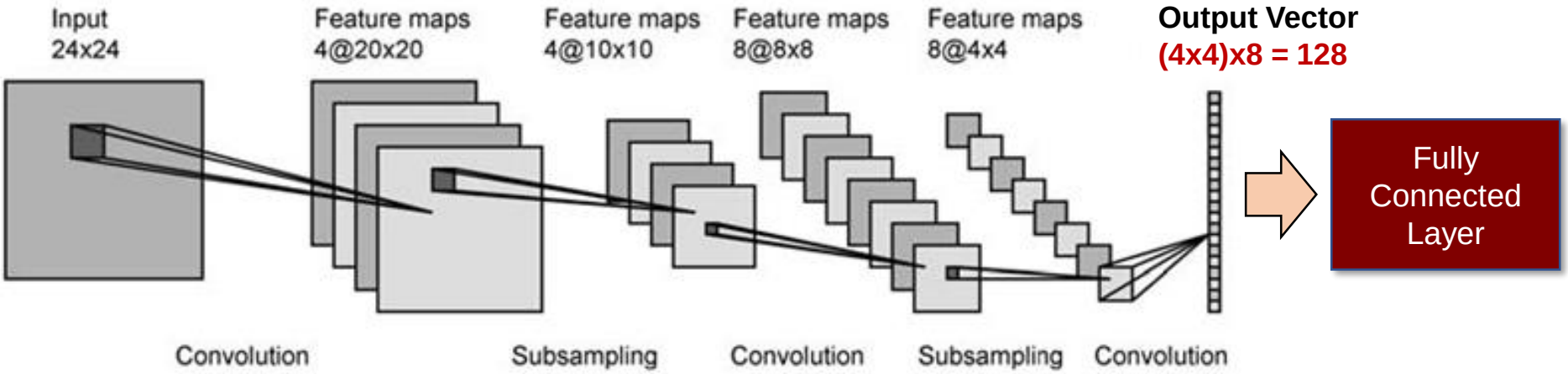
- Max Pooling Concept



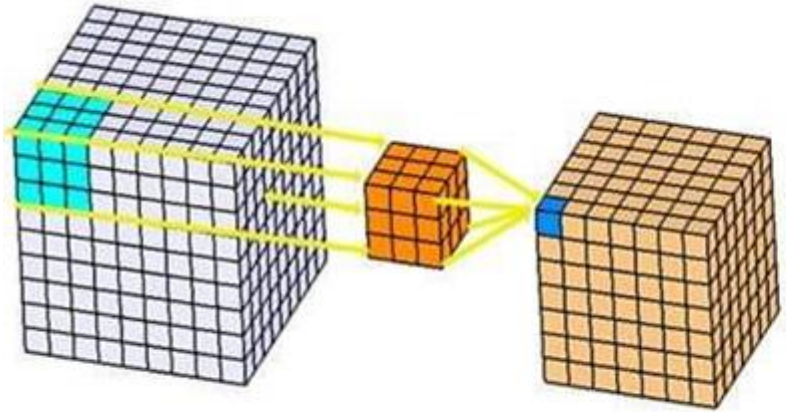
CNN: Max Pooling

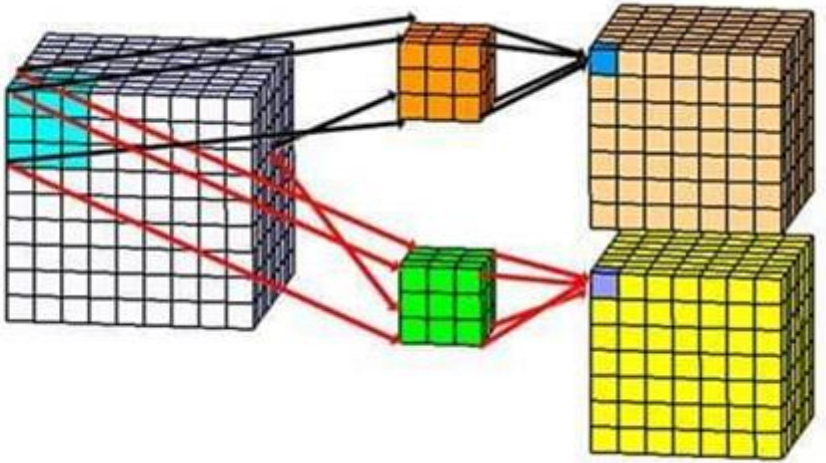


CNN: Example


Input
24x24
Feature maps
4@20x20
Feature maps
4@10x10
Feature maps
8@8x8
Feature maps
8@4x4
Output Vector
(4x4)x8 = 128
Fully Connected Layer
Convolution
Subsampling
Convolution
Subsampling
Convolution

- 3D Convolution

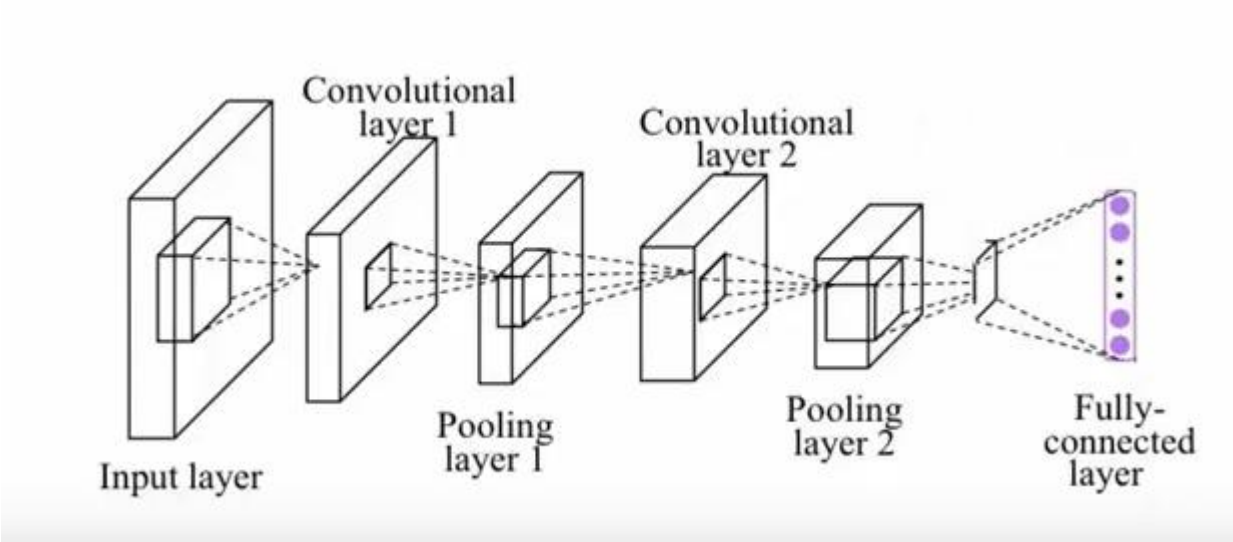




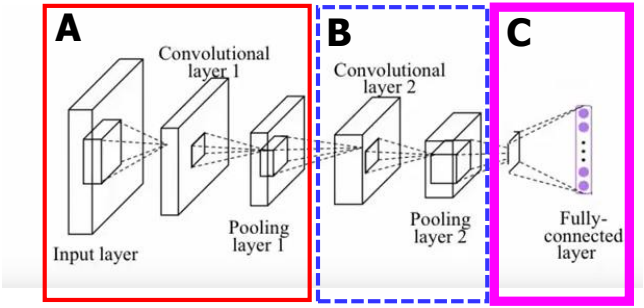
Deep Learning Theory and Software
Convolutional Neural Networks (CNN)
CNN Implementation

- CNN Theory
- **CNN Implementation**

- CNN for MNIST Image Classification



CNN Implementation (TensorFlow)



A

```
# Convolution Layer 1
W1 = tf.Variable(tf.random_normal([3,3,1,32], stddev=0.01))
CL1 = tf.nn.conv2d(X_img, W1, strides=[1,1,1,1], padding='SAME')
CL1 = tf.nn.relu(CL1)

# Pooling Layer 1
PL1 = tf.nn.max_pool(CL1, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')
```

B

```
# Convolution Layer 2
W2 = tf.Variable(tf.random_normal([3,3,32,64], stddev=0.01))
CL2 = tf.nn.conv2d(PL1, W2, strides=[1,1,1,1], padding='SAME')
CL2 = tf.nn.relu(CL2)

# Pooling Layer 2
PL2 = tf.nn.max_pool(CL2, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')
```

C

```
# Fully Connected (FC) Layer
L_flat = tf.reshape(PL2, [-1,7*7*64])
W3 = tf.Variable(tf.random_normal([7*7*64,10], stddev=0.01))
b3 = tf.Variable(tf.random_normal([10]))
```

CNN Implementation (TensorFlow)

- TensorFlow
  - **CNN for MNIST**
  - CNN for MNIST (with tensorflow.layers)
- Keras
  - CNN for MNIST

CNN Implementation (TensorFlow)

```
1  from tensorflow.examples.tutorials.mnist import input_data
2  mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
3
4  import tensorflow as tf
5  import time
6
7  training_epochs = 15
8  batch_size = 100
9
10 X = tf.placeholder(tf.float32, [None, 784])
11 Y = tf.placeholder(tf.float32, [None, 10])
12 X_img = tf.reshape(X, [-1, 28, 28, 1])
13
14 # Convolution Layer 1
15 W1 = tf.Variable(tf.random_normal([3,3,1,32], stddev=0.01))
16 CL1 = tf.nn.conv2d(X_img, W1, strides=[1,1,1,1], padding='SAME')
17 CL1 = tf.nn.relu(CL1)
18 # Pooling Layer 1
19 PL1 = tf.nn.max_pool(CL1, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')
20 # Convolution Layer 2
21 W2 = tf.Variable(tf.random_normal([3,3,32,64], stddev=0.01))
22 CL2 = tf.nn.conv2d(PL1, W2, strides=[1,1,1,1], padding='SAME')
23 CL2 = tf.nn.relu(CL2)
24 # Pooling Layer 2
25 PL2 = tf.nn.max_pool(CL2, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')
26 # Fully Connected (FC) Layer
27 L_flat = tf.reshape(PL2, [-1,7*7*64])
28 W3 = tf.Variable(tf.random_normal([7*7*64,10], stddev=0.01))
29 b3 = tf.Variable(tf.random_normal([10]))
```

CNN Implementation (TensorFlow)

```
31  # Model, Cost, Train
32  model_LC = tf.matmul(L_flat, W3) + b3
33  model = tf.nn.softmax(model_LC)
34  cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=model_LC, labels=Y))
35  train = tf.train.AdamOptimizer(0.01).minimize(cost)
36
37  # Accuracy
38  accuracy = tf.reduce_mean(tf.cast(tf.equal(tf.argmax(model, 1), tf.argmax(Y, 1)), tf.float32))
39
40  # Session
41  with tf.Session() as sess:
42      sess.run(tf.global_variables_initializer())
43      # Training
44      t1 = time.time()
45      for epoch in range(training_epochs):
46          total_batch = int(mnist.train.num_examples / batch_size)
47          for i in range(total_batch):
48              train_images, train_labels = mnist.train.next_batch(batch_size)
49              c, _ = sess.run([cost, train], feed_dict={X: train_images, Y: train_labels})
50              if i % 10 == 0:
51                  print('epoch:', epoch, ', batch number:', i)
52      t2 = time.time()
53      # Testing
54      print('Training Time (Seconds): ', t2-t1)
55      print('Accuracy: ', sess.run(accuracy, feed_dict={X: mnist.test.images, Y: mnist.test.labels}))
```

```
epoch: 14 , batch number: 520
epoch: 14 , batch number: 530
epoch: 14 , batch number: 540
Training Time (Seconds):  275.65979075431824
Accuracy:  0.9858
```

CNN Implementation (TensorFlow)

- TensorFlow
  - CNN for MNIST
  - **CNN for MNIST (with tensorflow.layers)**
- Keras
  - CNN for MNIST

CNN Implementation (TensorFlow)

```
# Convolution Layer 1
W1 = tf.Variable(tf.random_normal([3,3,1,32], stddev=0.01))
CL1 = tf.nn.conv2d(X_img, W1, strides=[1,1,1,1], padding='SAME')
CL1 = tf.nn.relu(CL1)
# Pooling Layer 1
PL1 = tf.nn.max_pool(CL1, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')
# Convolution Layer 2
W2 = tf.Variable(tf.random_normal([3,3,32,64], stddev=0.01))
CL2 = tf.nn.conv2d(PL1, W2, strides=[1,1,1,1], padding='SAME')
CL2 = tf.nn.relu(CL2)
# Pooling Layer 2
PL2 = tf.nn.max_pool(CL2, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')
# Fully Connected (FC) Layer
L_flat = tf.reshape(PL2, [-1,7*7*64])
W3 = tf.Variable(tf.random_normal([7*7*64,10], stddev=0.01))
b3 = tf.Variable(tf.random_normal([10]))
```

tensorflow.layers.conv2d
tensorflow.layers.max_pooling2d

```
# Convolution Layer 1
CL1 = tf.layers.conv2d(inputs=X_img, filters=32, kernel_size=[3,3], padding='SAME', strides=1, activation=tf.nn.relu)
# Pooling Layer 1
PL1 = tf.layers.max_pooling2d(inputs=CL1, pool_size=[2,2], padding='SAME', strides=2)
# Convolution Layer 2
CL2 = tf.layers.conv2d(inputs=PL1, filters=64, kernel_size=[3,3], padding='SAME', strides=1, activation=tf.nn.relu)
# Pooling Layer 1
PL2 = tf.layers.max_pooling2d(inputs=CL2, pool_size=[2,2], padding='SAME', strides=2)
# Fully Connected (FC) Layer
L_flat = tf.reshape(PL2, [-1,7*7*64])
W3 = tf.Variable(tf.random_normal([7*7*64,10], stddev=0.01))
b3 = tf.Variable(tf.random_normal([10]))
```

CNN Implementation (Keras)

- TensorFlow
  - CNN for MNIST
  - CNN for MNIST (with tensorflow.layers)
- Keras
  - **CNN for MNIST**

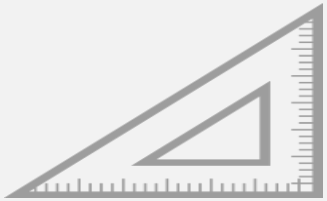
CNN Implementation (Keras)

```
1  from keras.utils import np_utils
2  from keras.datasets import mnist
3  from keras.models import Sequential
4  from keras.layers import Conv2D, pooling, Flatten, Dense
5  # MNIST data
6  (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
7  print(train_images.shape, train_labels.shape, test_images.shape, test_labels.shape)
8  train_images = train_images.reshape(train_images.shape[0], 28,28,1).astype('float32')/255.0
9  test_images = test_images.reshape(test_images.shape[0], 28,28,1).astype('float32')/255.0
10 train_labels = np_utils.to_categorical(train_labels) # One-Hot Encoding
11 test_labels = np_utils.to_categorical(test_labels) # One-Hot Encoding
12 # Model
13 model = Sequential()
14 model.add(Conv2D(32, (3,3), padding='same', strides=(1,1), activation='relu', input_shape=(28,28,1)))
15 print(model.output_shape)
16 model.add(pooling.MaxPooling2D(pool_size=(2,2)))
17 print(model.output_shape)
18 model.add(Conv2D(64, (3,3), padding='same', strides=(1,1), activation='relu'))
19 print(model.output_shape)
20 model.add(pooling.MaxPooling2D(pool_size=(2,2)))
21 print(model.output_shape)
22 model.add(Flatten())
23 model.add(Dense(10, activation='softmax')) # units=10, activation='softmax'
24 model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
25 # Training
26 model.fit(train_images, train_labels, epochs=5, batch_size=32, verbose=1)
27 # Testing
28 _, accuracy = model.evaluate(test_images, test_labels)
29 print('Accuracy: ', accuracy)
30 model.summary()
```

CNN Implementation (Keras)

```
(60000, 28, 28) (60000,) (10000, 28, 28) (10000,)
(None, 28, 28, 32)
(None, 14, 14, 32)
(None, 14, 14, 64)
(None, 7, 7, 64)
Epoch 1/5
2019-08-04 22:30:43.603681: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU
supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
60000/60000 [==============================] - 22s 359us/step - loss: 0.5194 - acc: 0.8502
Epoch 2/5
60000/60000 [==============================] - 21s 357us/step - loss: 0.1657 - acc: 0.9506
Epoch 3/5
60000/60000 [==============================] - 21s 358us/step - loss: 0.1138 - acc: 0.9668
Epoch 4/5
60000/60000 [==============================] - 21s 357us/step - loss: 0.0914 - acc: 0.9733
Epoch 5/5
60000/60000 [==============================] - 22s 362us/step - loss: 0.0786 - acc: 0.9762
10000/10000 [==============================] - 1s 117us/step
Accuracy:  0.9774

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                31370
=================================================================
Total params: 50,186
Trainable params: 50,186
Non-trainable params: 0
```

Linear Functions



Linear Regression
Binary Classification
Softmax Classification

Nonlinear Functions

Neural Network (NN)
Convolutional NN (CNN)
CNN for CIFAR-10
Recurrent NN (RNN)

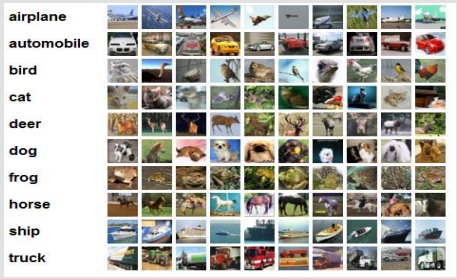
Advanced Topics

Gen. Adv. Network (GAN)
Interpolation
PCA/LDA
Overfitting

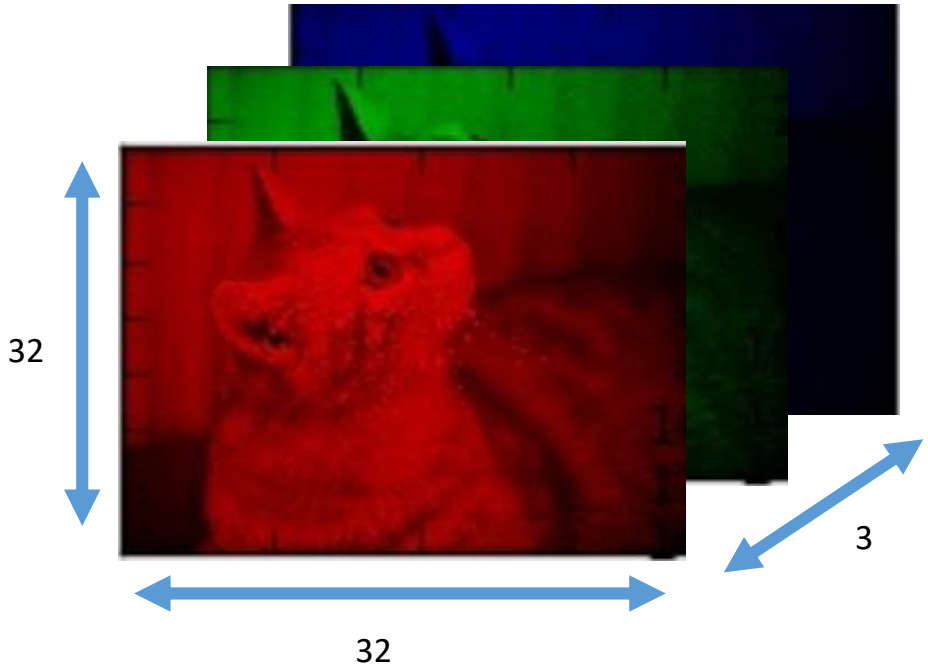
Deep Learning Theory and Software
CNN for CIFAR-10
CIFAR-10 Dataset

- **CIFAR-10 Dataset**

Image Dataset

	MNIST	CIFAR-10,100	IMAGENET
			
Num Channel	1 (Gray scale)	3 (R, G, B)	3 (R, G, B)
Num Classes	10	10 , 100	1000
Resolution	28 * 28	32 * 32	(256 * 256)
Num Training Set	50,000	50,000	200,000

- RGB 3 channel image (32-by-32-by-3 matrix)



CIFAR-10

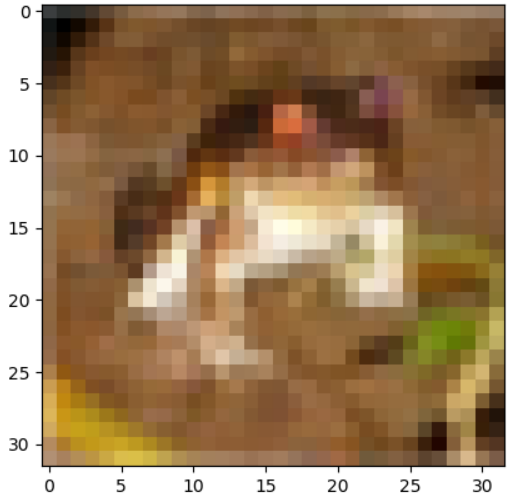
```
1  from keras.datasets import cifar10
2
3  (train_images, train_labels), (test_images, test_labels) = cifar10.load_data()
4  print('train_images', train_images.shape)
5  print('train_labels', train_labels.shape)
6  print('test_images', test_images.shape)
7  print('test_labels', test_labels.shape)
```

```
train_images (50000, 32, 32, 3)
train_labels (50000, 1)
test_images (10000, 32, 32, 3)
test_labels (10000, 1)
```

- CIFAR-10
  - Train
    - (50000, 32, 32, 3) → 3 channel RGB (32-by-32), 50000 images
    - (50000, 1) → 50000 labels (no one-hot encoding)
  - Test
    - (10000, 32, 32, 3) → 3 channel RGB (32-by-32), 10000 images
    - (10000, 1) → 10000 labels (no one-hot encoding)

CIFAR-10

```
1  from keras.datasets import cifar10
2  from matplotlib import pyplot as plt
3
4  (train_images, train_labels), (test_images, test_labels) = cifar10.load_data()
5
6  plt.imshow(train_images[0])
7  plt.show()
```



Implementation

- CIFAR-10 Classification Implementation
  - **TensorFlow**
  - Keras

Implementation for CIFAR-10 (TensorFlow)

```
1  from keras.utils import np_utils
2  from keras.datasets import cifar10
3  import tensorflow as tf
4  import time
5
6  (train_images, train_labels), (test_images, test_labels) = cifar10.load_data()
7  train_labels = np_utils.to_categorical(train_labels) # One-Hot Encoding
8  test_labels = np_utils.to_categorical(test_labels) # One-Hot Encoding
9
10 training_epochs = 15
11 batch_size = 100
12
13 X = tf.placeholder(tf.float32, [None, 32,32,3])
14 Y = tf.placeholder(tf.float32, [None, 10])
15 X_img = tf.reshape(X, [-1, 32, 32, 3])
16
17 # Convolution Layer 1
18 W1 = tf.Variable(tf.random_normal([3,3,3,32], stddev=0.01))
19 CL1 = tf.nn.conv2d(X_img, W1, strides=[1,1,1,1], padding='SAME')
20 CL1 = tf.nn.relu(CL1)
21 # Pooling Layer 1
22 PL1 = tf.nn.max_pool(CL1, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')
23 # Convolution Layer 2
24 W2 = tf.Variable(tf.random_normal([3,3,32,64], stddev=0.01))
25 CL2 = tf.nn.conv2d(PL1, W2, strides=[1,1,1,1], padding='SAME')
26 CL2 = tf.nn.relu(CL2)
27 # Pooling Layer 2
28 PL2 = tf.nn.max_pool(CL2, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')
29 # Fully Connected (FC) Layer
30 L_flat = tf.reshape(PL2, [-1,8*8*64])
31 W3 = tf.Variable(tf.random_normal([8*8*64,10], stddev=0.01))
32 b3 = tf.Variable(tf.random_normal([10]))
```

Implementation for CIFAR-10 (TensorFlow)

```
34  # Model, Cost, Train
35  model_LC = tf.matmul(L_flat, W3) + b3
36  model = tf.nn.softmax(model_LC)
37  cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=model_LC, labels=Y))
38  train = tf.train.AdamOptimizer(0.01).minimize(cost)
39
40  # Accuracy
41  accuracy = tf.reduce_mean(tf.cast(tf.equal(tf.argmax(model, 1), tf.argmax(Y, 1)), tf.float32))
42
43  # Session
44  with tf.Session() as sess:
45      sess.run(tf.global_variables_initializer())
46      # Training
47      t1 = time.time()
48      for epoch in range(training_epochs):
49          total_batch = int(train_images.shape[0] / batch_size)
50          for i in range(total_batch):
51              batch_train_images = train_images[i*batch_size:(i+1)*batch_size]
52              batch_train_labels = train_labels[i*batch_size:(i+1)*batch_size]
53              c, _ = sess.run([cost, train], feed_dict={X: batch_train_images, Y: batch_train_labels})
54              if i % 10 == 0:
55                  print('epoch:', epoch, ', batch number:', i)
56      t2 = time.time()
57      # Testing
58      print('Training Time (Seconds): ', t2-t1)
59      total_batch = int(test_images.shape[0] / batch_size)
60      for i in range(total_batch):
61          batch_test_images = test_images[i*batch_size:(i+1)*batch_size]
62          batch_test_labels = test_labels[i*batch_size:(i+1)*batch_size]
63      print('Accuracy: ', sess.run(accuracy, feed_dict={X: batch_test_images, Y: batch_test_labels}))
```

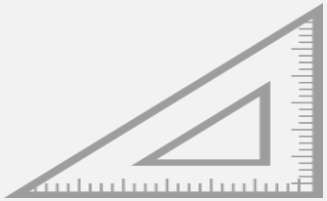
Implementation

- CIFAR-10 Classification Implementation
  - TensorFlow
  - **Keras**

Implementation for CIFAR-10 (Keras)

```
from keras.utils import np_utils
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Conv2D, pooling, Flatten, Dense
# MNIST data
(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()
print(train_images.shape, train_labels.shape, test_images.shape, test_labels.shape)

train_images = train_images.reshape(train_images.shape[0], 32,32,3).astype('float32')/255.0
test_images = test_images.reshape(test_images.shape[0], 32,32,3).astype('float32')/255.0

train_labels = np_utils.to_categorical(train_labels) # One-Hot Encoding
test_labels = np_utils.to_categorical(test_labels) # One-Hot Encoding
# Model
model = Sequential()
model.add(Conv2D(32, (3,3), padding='same', strides=(1,1), activation='relu', input_shape=(32,32,3)))
model.add(pooling.MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(64, (3,3), padding='same', strides=(1,1), activation='relu'))
model.add(pooling.MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(10, activation='softmax')) # units=10, activation='softmax'
model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
# Training
model.fit(train_images, train_labels, epochs=5, batch_size=32, verbose=1)
# Testing
_, accuracy = model.evaluate(test_images, test_labels)
print('Accuracy: ', accuracy)
model.summary()
```

Linear Functions



Linear Regression
Binary Classification
Softmax Classification

Nonlinear Functions

Neural Network (NN)
Convolutional NN (CNN)
CNN for CIFAR-10
Recurrent NN (RNN)

Advanced Topics

Gen. Adv. Network (GAN)
Interpolation
PCA/LDA
Overfitting

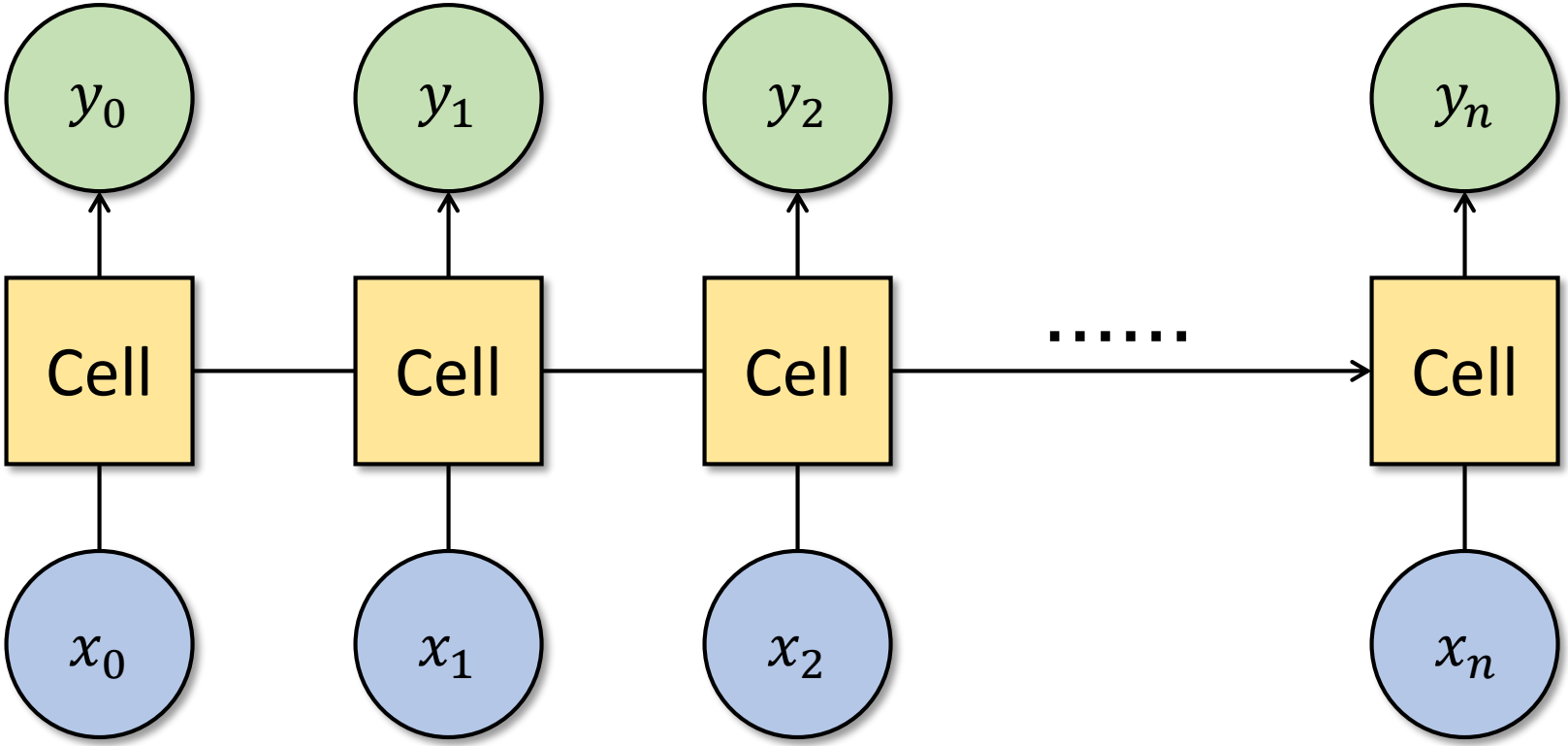
Deep Learning Theory and Software
Recurrent Neural Networks (CNN)
RNN Theory

- **RNN Theory**
- RNN Implementation

Recurrent Neural Networks (RNN): Introduction

- Sequence Data
  - We don’t understand one word only.
    We understand based on the previous words + this word (time series).
  - ANN/CNN cannot do this.

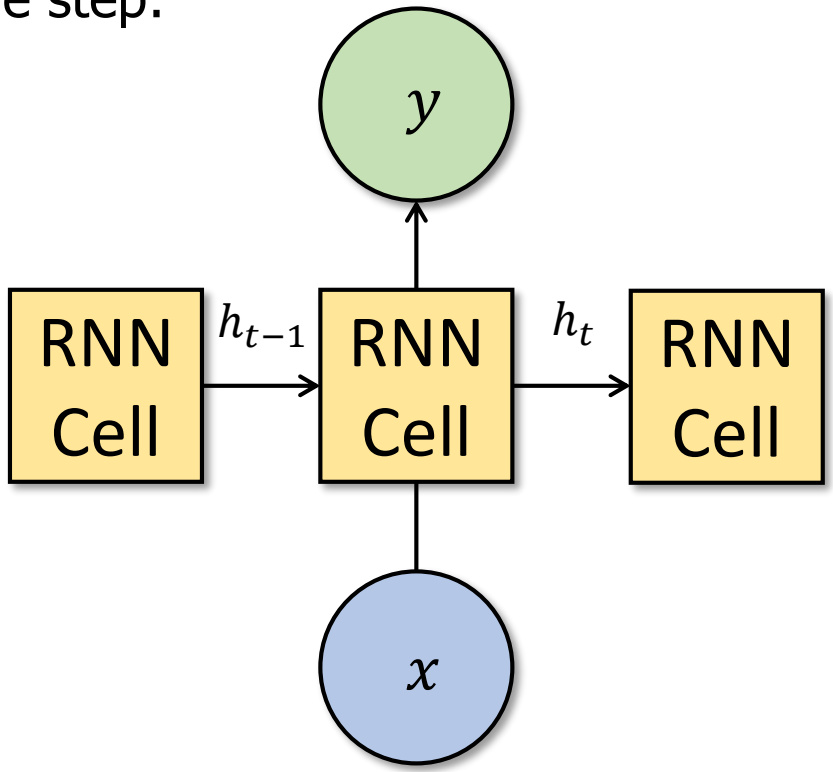
Recurrent Neural Networks (RNN): Introduction



- We can process a sequence of vector $\vec{x}$ by applying a recurrence formula at every time step:

$$h_t = f_W(h_{t-1}, x_t)$$

- h_t: New state
- $f_W(\cdot)$: Some function with parameters W
  - Equivalent to all RNN functions
- h_{t-1}: Old state
- x_t: Input vector at some time step



- The state consists of a single hidden vector $\vec{h}$

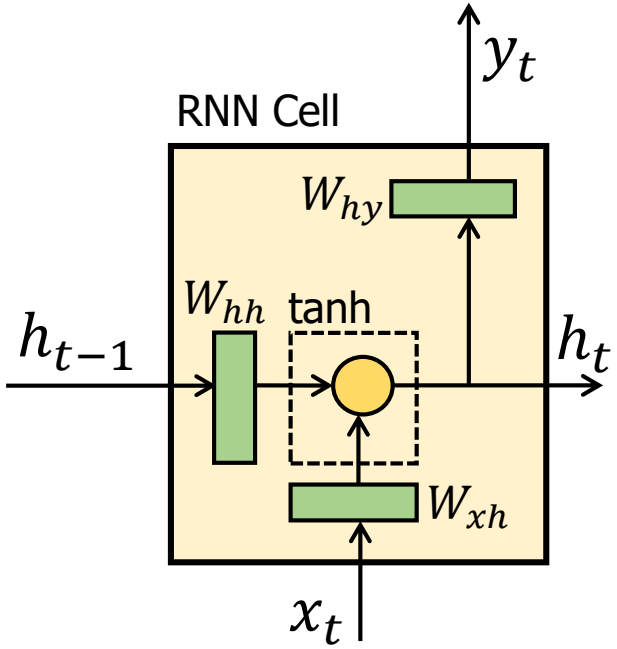
$$h_t = f_W(h_{t-1}, x_t)$$

Input to next step RNN

$$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Output

$$y_t = W_{hy}h_t$$

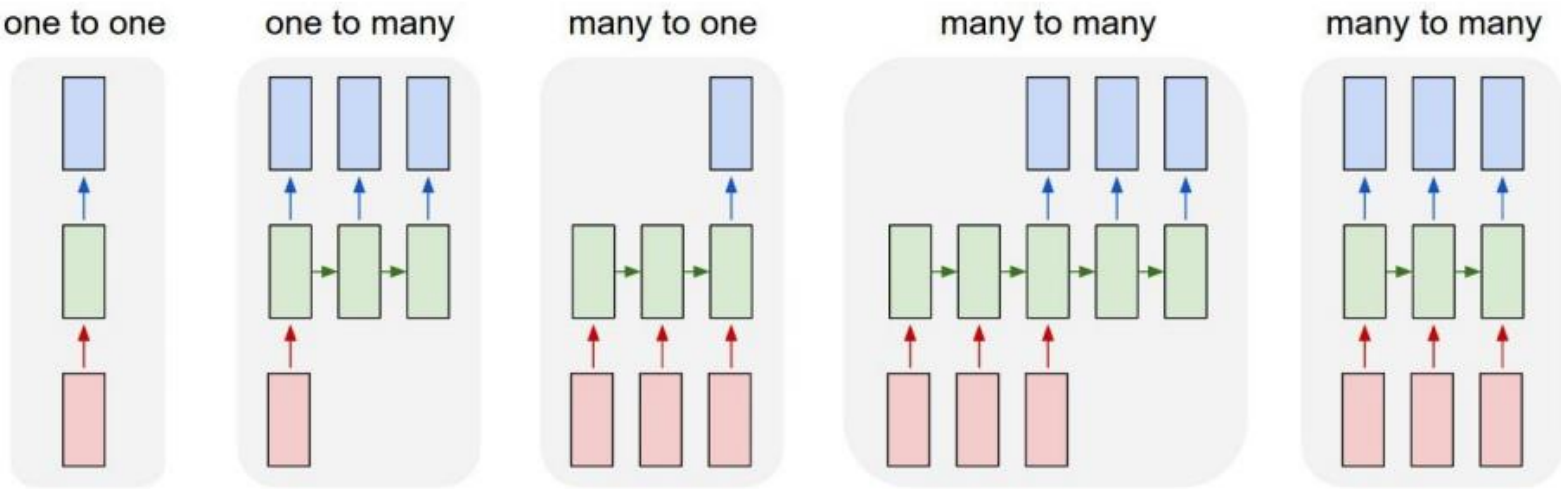


RNN: Applications

- RNN applications
  - Language Modeling
  - Speech Recognition
  - Machine Translation
  - Question Answering (QA) Systems
  - Conversation Modeling
  - Image/Video Captioning
  - Image/Music/Dance Generation

RNN: Applications

- RNN offers a lot of flexibility.



- RNN offers a lot of flexibility.

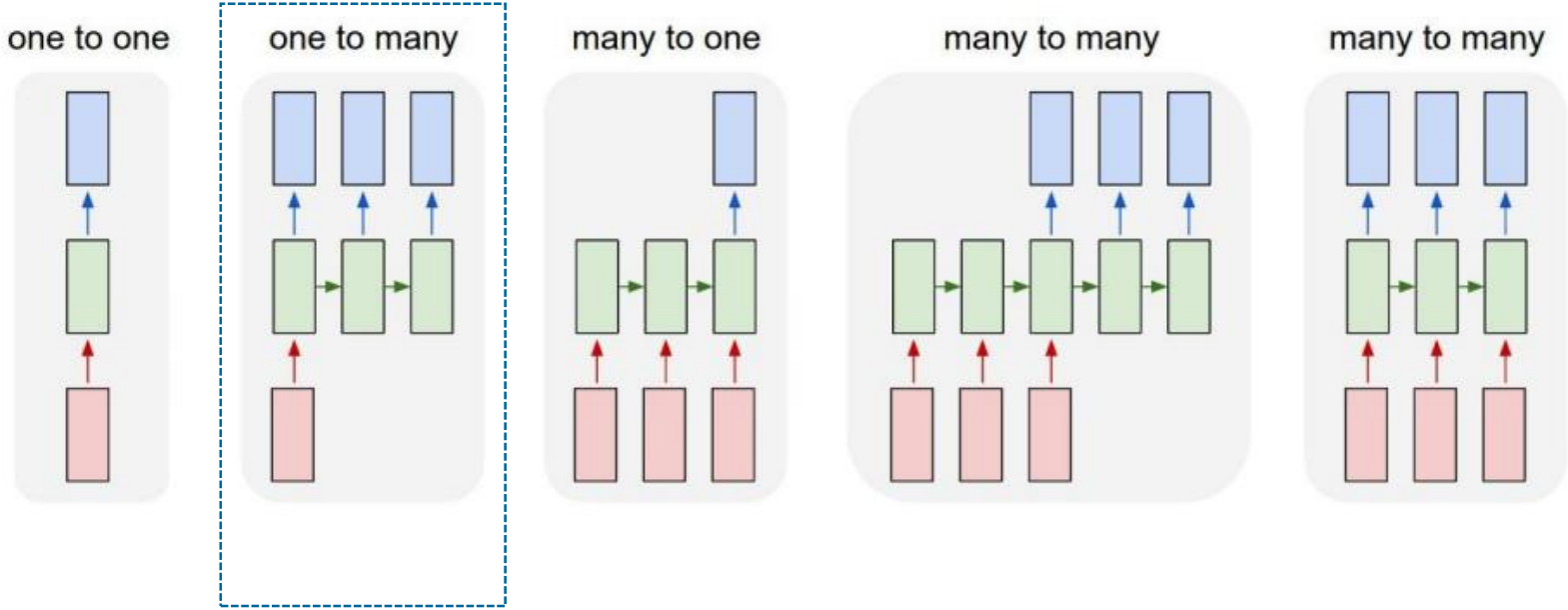
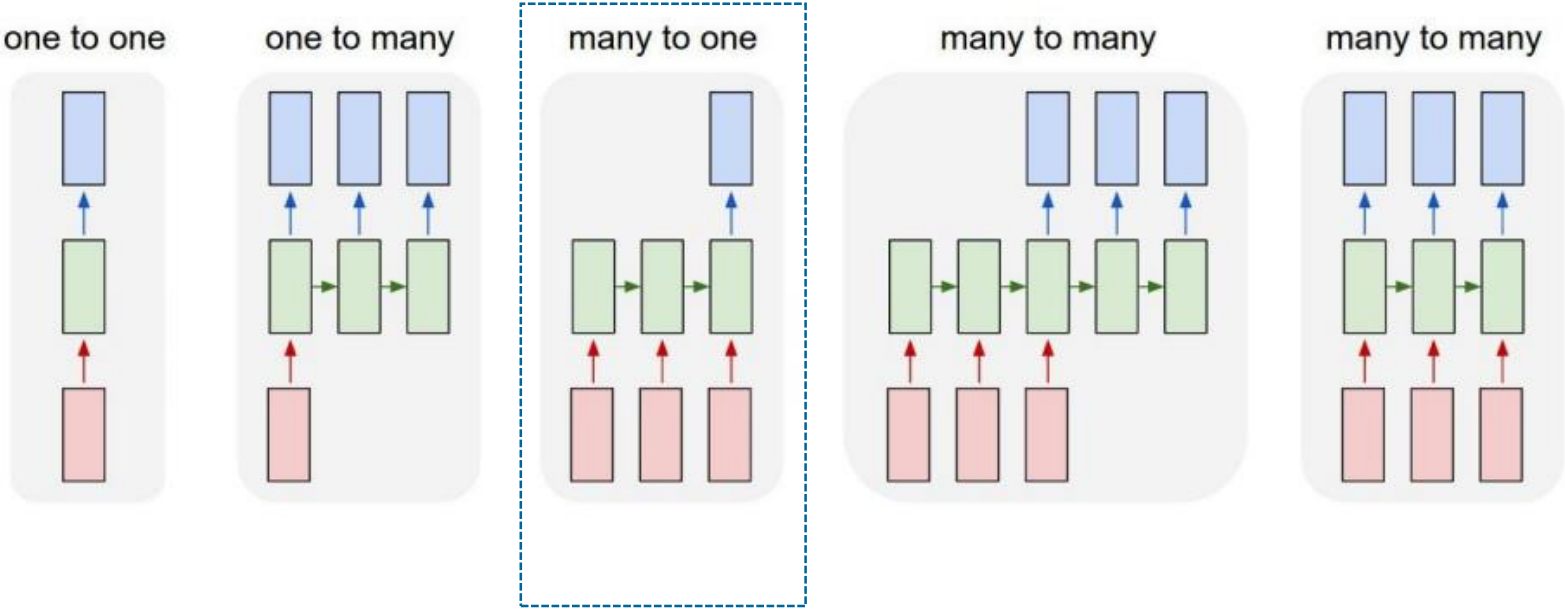


Image Captioning

- Deriving a sequence of words from one image

- RNN offers a lot of flexibility.

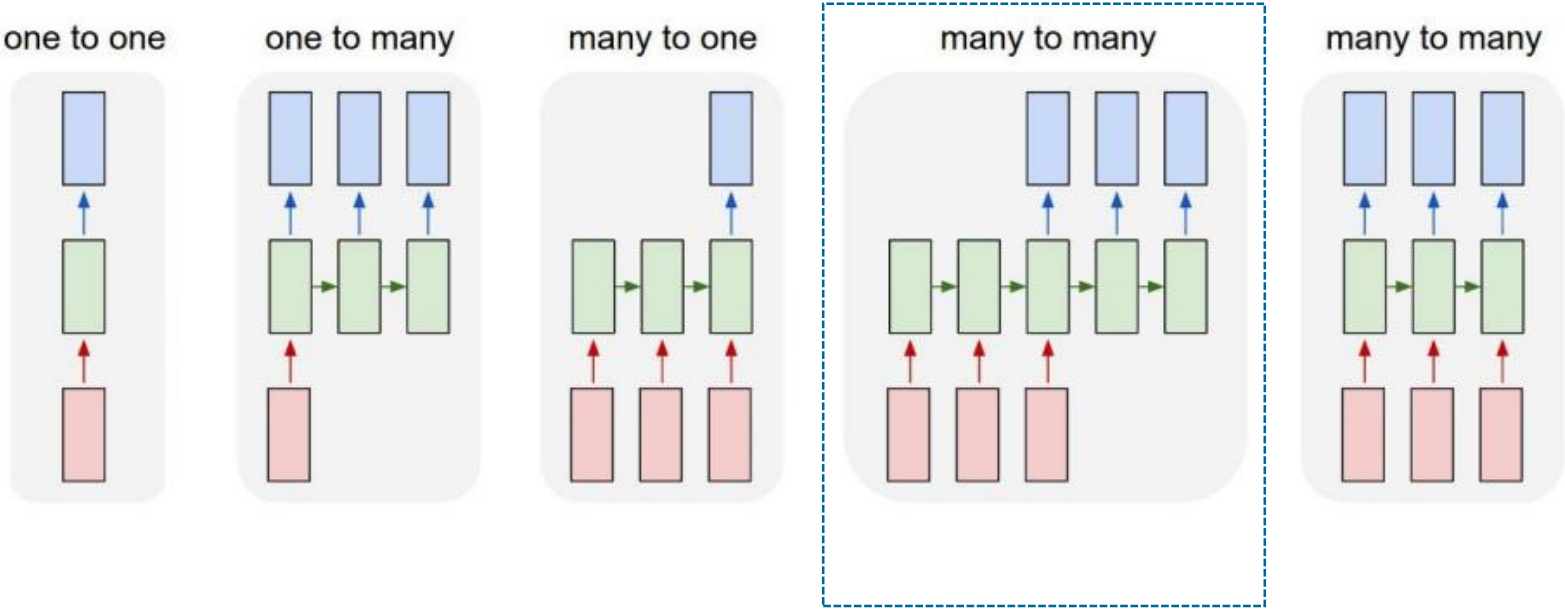


Sentiment Classification

- Deriving a sentiment from several sentences
- Affective computing

RNN: Applications

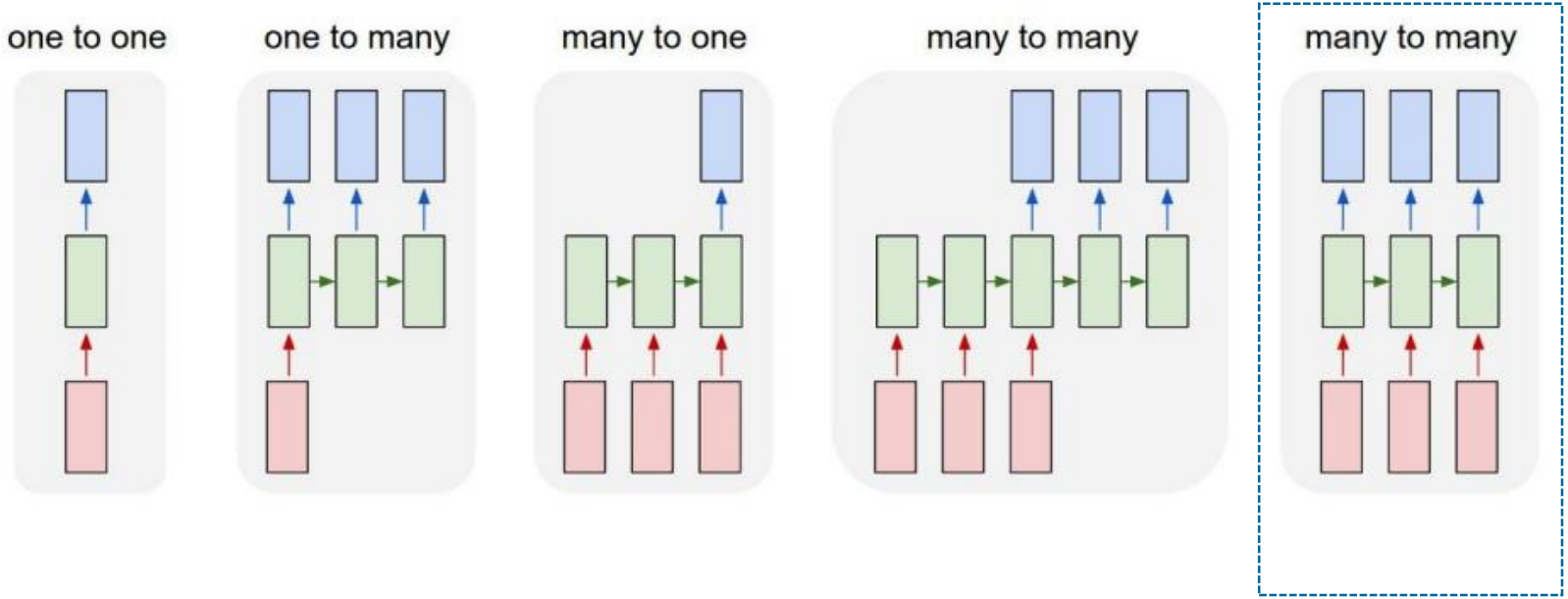
- RNN offers a lot of flexibility.



Machine Translation

- Deriving a sequence of words from the other sequence of words

- RNN offers a lot of flexibility.



Video Classification on frame level

- Deriving video descriptions from individual frames

Deep Learning Theory and Software
Recurrent Neural Networks (CNN)
RNN Implementation

- RNN Theory
- **RNN Implementation**

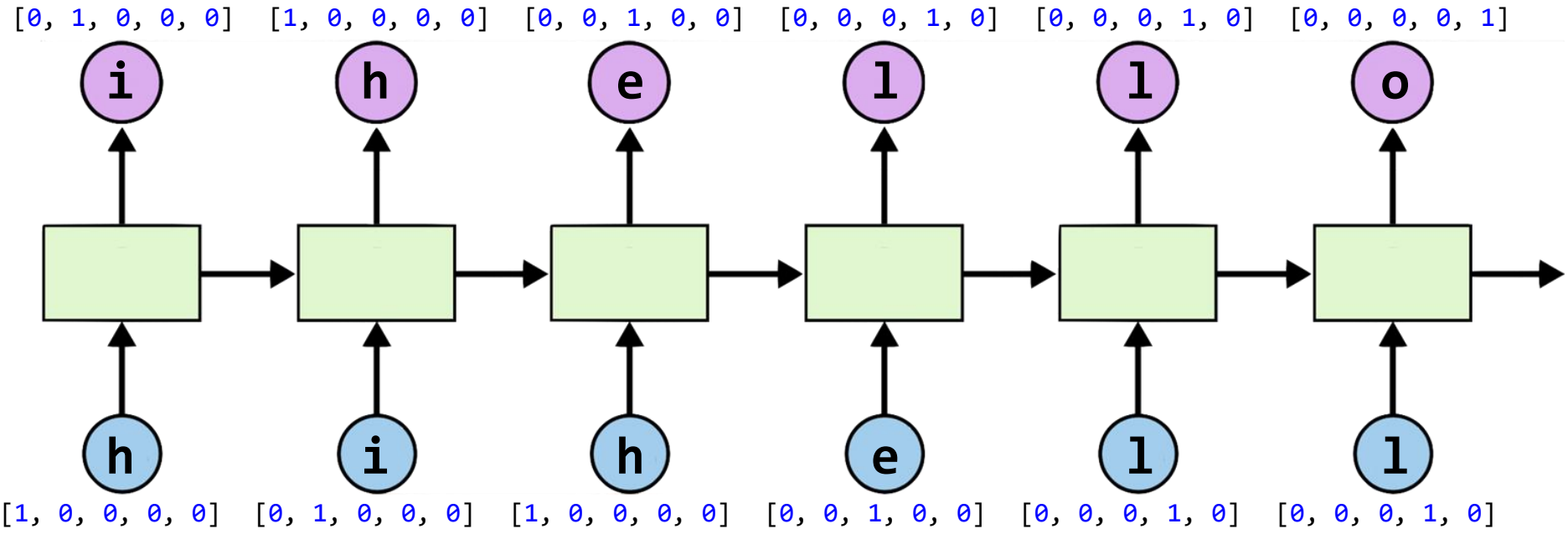
RNN Implementation (TensorFlow)

- TensorFlow
  - **RNN Basic**
- Keras
  - RNN Learning (LSTM)

RNN Implementation (TensorFlow)

- Teach RNN 'hihello' (example)

```
[1, 0, 0, 0, 0],   # h 0
[0, 1, 0, 0, 0],   # i 1
[0, 0, 1, 0, 0],   # e 2
[0, 0, 0, 1, 0],   # l 3
[0, 0, 0, 1, 0],   # l 3
[0, 0, 0, 0, 1],   # o 4
```

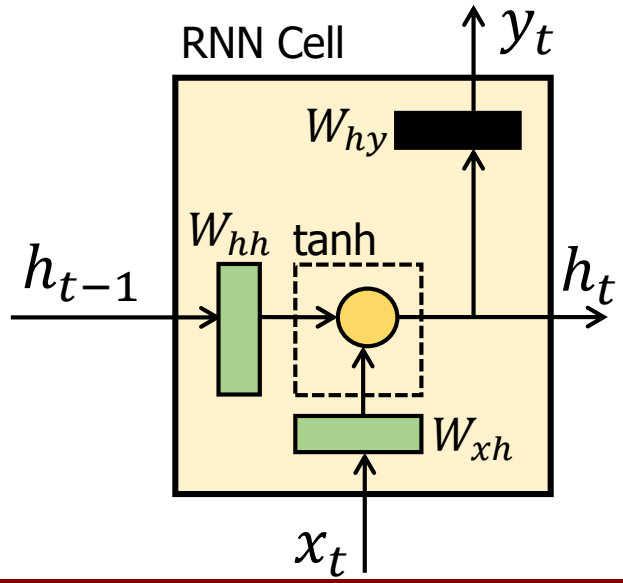


RNN Implementation (TensorFlow)

```
1  import tensorflow as tf
2  import numpy as np
3  sample = " My name is Joongheon Kim."
4  idx2char = list(set(sample))  # index -> char
5  print(idx2char)
6  char2idx = {c: i for i, c in enumerate(idx2char)}  # char -> idex
7  print(char2idx)
8  # hyper parameters
9  dic_size = len(char2idx)  # RNN input size (one hot size)
10 print(dic_size)
11 hidden_size = len(char2idx)  # RNN output size
12 num_classes = len(char2idx)  # final output size (RNN or softmax, etc.)
13 batch_size = 1  # one sample data, one batch
14 sequence_length = len(sample) - 1  # number of RNN rollings (unit #)
15 print(sequence_length)
16 sample_idx = [char2idx[c] for c in sample]  # char to index
17 print(sample_idx)
18 x_data = [sample_idx[:-1]]  # X data sample (0 ~ n-1)
19 y_data = [sample_idx[1:]]  # Y label sample (1 ~ n)
20 X = tf.placeholder(tf.int32, [None, sequence_length])  # X data
21 Y = tf.placeholder(tf.int32, [None, sequence_length])  # Y label
22 x_one_hot = tf.one_hot(X, num_classes)  # one hot: 1 -> 0 1 0 0 0 0 0 0 ... 0
```

RNN Implementation (TensorFlow)

```
23  # cell and RNN
24  cell = tf.contrib.rnn.BasicRNNCell(num_units=hidden_size)
25  outputs, _states = tf.nn.dynamic_rnn(cell, x_one_hot, dtype=tf.float32)
26  # FC layer
27  X_for_fc = tf.reshape(outputs, [-1, hidden_size])
28  outputs = tf.contrib.layers.fully_connected(X_for_fc, num_classes, activation_fn=None)
29  # reshape out for sequence_loss
30  outputs = tf.reshape(outputs, [batch_size, sequence_length, num_classes])
31  sequence_loss = tf.contrib.seq2seq.sequence_loss(logits=outputs, targets=Y, weights=tf.ones([batch_size, sequence_length]))
32  cost = tf.reduce_mean(sequence_loss)
33  train = tf.train.AdamOptimizer(0.1).minimize(cost)
34  prediction = tf.argmax(outputs, axis=2)
```



RNN Implementation (TensorFlow)

```
36  with tf.Session() as sess:
37      sess.run(tf.global_variables_initializer())
38      for i in range(20):
39          l, _ = sess.run([cost, train], feed_dict={X: x_data, Y: y_data})
40          result = sess.run(prediction, feed_dict={X: x_data})
41          # print char using dic
42          result_str = [idx2char[c] for c in np.squeeze(result)]
43          print(i, ", loss:", l, ", Prediction:", ''.join(result_str))
```

RNN Implementation (TensorFlow)

```
['e', 'n', 'm', 'J', ' ', 'y', 'h', 's', 'i', 'g', 'K', 'M', '.', 'o', 'a']
{'e': 0, 'n': 1, 'm': 2, 'J': 3, ' ': 4, 'y': 5, 'h': 6, 's': 7, 'i': 8, 'g': 9, 'K': 10, 'M': 11, '.': 12, 'o': 13, 'a':
14}
15
25
[4, 11, 5, 4, 1, 14, 2, 0, 4, 8, 7, 4, 3, 13, 13, 1, 9, 6, 0, 13, 1, 4, 10, 8, 2, 12]
WARNING:tensorflow:From C:/Users/CAU/Desktop/untitled4.py:31: BasicRNNCell.__init__ (from
tensorflow.python.ops.rnn_cell_impl) is deprecated and will be removed in a future version.
Instructions for updating:
This class is equivalent as tf.keras.layers.SimpleRNNCell, and will be replaced by that in Tensorflow 2.0.
0 , loss: 2.7530553 , Prediction: iyaaaeeansssoonnieanaaeme
1 , loss: 2.2681878 , Prediction:  omnome ssoJoohgie n  Mo.
2 , loss: 1.9158436 , Prediction:  omngme n oJoongheonnheme
3 , loss: 1.4575784 , Prediction:  ooname mn JoongheonnKeme
4 , loss: 1.1863785 , Prediction:  yoname e  JoongheongKem.
5 , loss: 0.8737224 , Prediction: Jy name e  JoongheongKem.
6 , loss: 0.6875408 , Prediction: Jy name es Joongheon Kim.
7 , loss: 0.56999004 , Prediction: My name es Joonaheon Kim.
8 , loss: 0.38090125 , Prediction: My name is Joongheon Kim.
9 , loss: 0.26949963 , Prediction: My name is Joongheon Kim.
10 , loss: 0.17376524 , Prediction: My name is Joongheon Kim.
11 , loss: 0.12433112 , Prediction: My name is Joongheon Kim.
12 , loss: 0.09018694 , Prediction: My name is Joongheon Kim.
13 , loss: 0.06243824 , Prediction: My name is Joongheon Kim.
14 , loss: 0.044248596 , Prediction: My name is Joongheon Kim.
15 , loss: 0.0328589 , Prediction: My name is Joongheon Kim.
16 , loss: 0.024894495 , Prediction: My name is Joongheon Kim.
17 , loss: 0.018896589 , Prediction: My name is Joongheon Kim.
18 , loss: 0.014688267 , Prediction: My name is Joongheon Kim.
19 , loss: 0.012020792 , Prediction: My name is Joongheon Kim.
```

RNN Implementation (Keras)

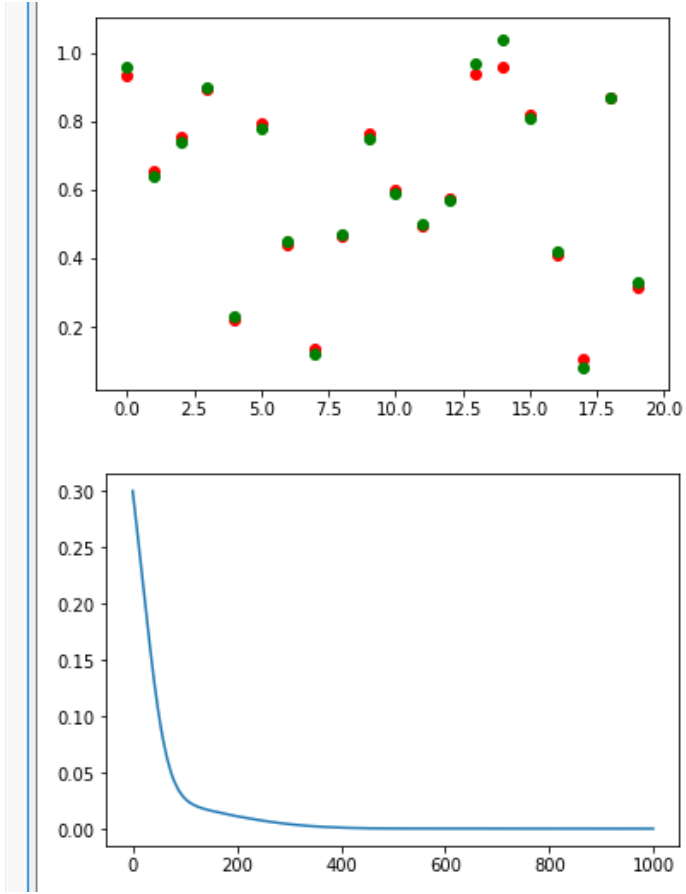
- TensorFlow
  - RNN Basic
- Keras
  - **RNN Learning (LSTM)**

RNN Implementation (Keras)

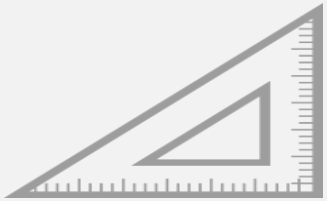
```
1  from keras.models import Sequential
2  from keras.layers import LSTM
3  from sklearn.model_selection import train_test_split # pip install -U scikit-learn
4  import numpy as np
5  import matplotlib.pyplot as plt
6
7  x_data = [[[(i+j)/100] for i in range (5)] for j in range(100)]
8  y_data = [(i+5)/100 for i in range (100)]
9  x_data = np.array(x_data, dtype=float)
10 y_data = np.array(y_data, dtype=float)
11 x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.2)
12 model=Sequential()
13 model.add(LSTM(1, input_dim=1, input_length=5, return_sequences = False))
14 model.compile(loss='mse', optimizer='adam')
15 model.summary()
16 history = model.fit(x_train, y_train, epochs=1000, verbose=0)
17 y_predict = model.predict(x_test)
18 plt.scatter(range(20), y_predict, c='r')
19 plt.scatter(range(20), y_test, c='g')
20 plt.show()
21 plt.plot(history.history['loss'])
22 plt.show()
```

x_data: 100-by-5 samples
y_data: 100-by-1 samples

RNN Implementation (Keras)



Linear Functions



Linear Regression
Binary Classification
Softmax Classification

Nonlinear Functions

Neural Network (NN)
Convolutional NN (CNN)
CNN for CIFAR-10
Recurrent NN (RNN)

Advanced Topics

Gen. Adv. Network (GAN)
Interpolation
PCA/LDA
Overfitting

Deep Learning Theory and Software
Generative Adversarial Networks (GAN)
GAN Theory

- **GAN Theory**
- GAN Implementation

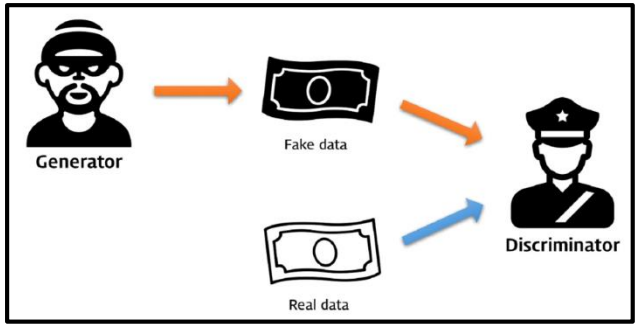
- GAN: Generative Adversarial Network
- Training both of **generator** and **discriminator**; and then generates samples which are similar to the original samples

Performance Improvements via Competition

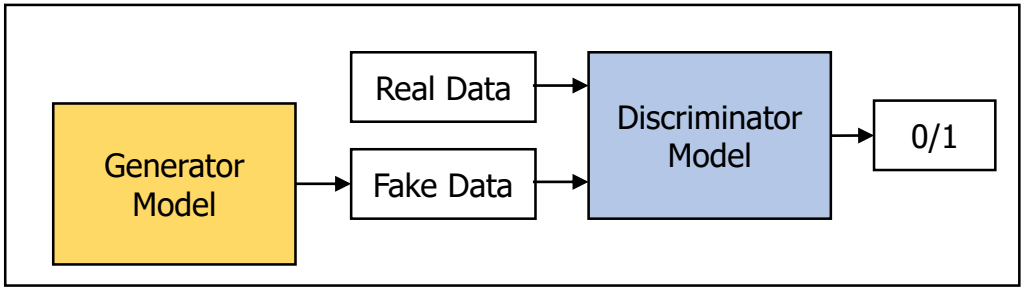
Generators

Discriminator

Introduction to GAN



GAN architecture

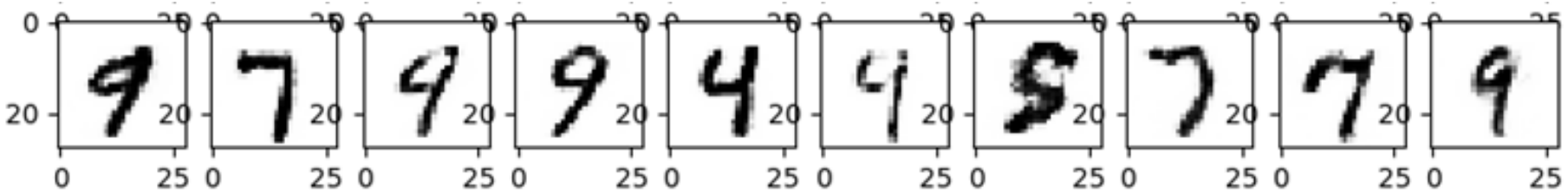


$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[log\ D(x)] + E_{z\sim p_z(z)}[log(1 - D(G(z))]$$

Deep Learning Theory and Software
Generative Adversarial Networks (GAN)
GAN Implementation

- GAN Theory
- **GAN Implementation**

TensorFlow for GAN



TensorFlow for GAN

```
# MNIST data
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("data_MNIST", one_hot=True)

import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf

# Training Params
num_steps = 100000
batch_size = 128

# Network Params
dim_image = 784 # 28*28 pixels
nHL_thief = 256
nHL_police = 256
dim_noise = 100 # Noise data points

# A custom initialization (see Xavier Glorot init)
def glorot_init(shape):
    return tf.random_normal(shape=shape, stddev=1. / tf.sqrt(shape[0] / 2.))
```

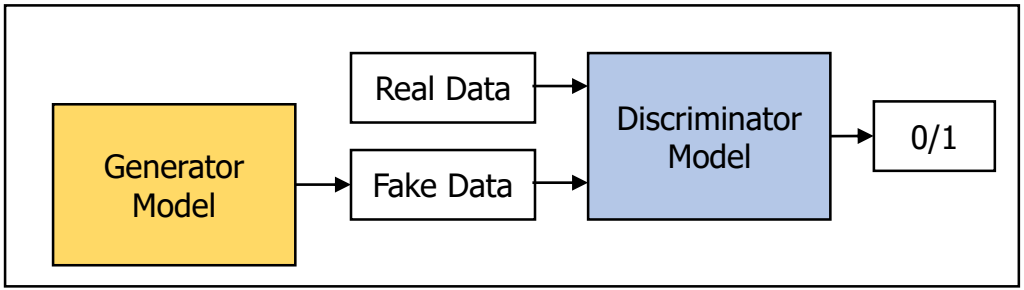
TensorFlow for GAN

```
23  W = {
24      'HL_thief' : tf.Variable(glorot_init([dim_noise, nHL_thief])),
25      'OL_thief' : tf.Variable(glorot_init([nHL_thief, dim_image])),
26      'HL_police': tf.Variable(glorot_init([dim_image, nHL_police])),
27      'OL_police': tf.Variable(glorot_init([nHL_police, 1])),
28  }
29  b = {
30      'HL_thief' : tf.Variable(tf.zeros([nHL_thief])),
31      'OL_thief' : tf.Variable(tf.zeros([dim_image])),
32      'HL_police': tf.Variable(tf.zeros([nHL_police])),
33      'OL_police': tf.Variable(tf.zeros([1])),
34  }
35
36  # Neural Network: Thief
37  def nn_thief(x):
38      HL = tf.nn.relu(tf.add(tf.matmul(x, W['HL_thief']), b['HL_thief']))
39      OL = tf.nn.sigmoid(tf.add(tf.matmul(HL, W['OL_thief']), b['OL_thief']))
40      return OL
41
42  # Neural Network: Police
43  def nn_police(x):
44      HL = tf.nn.relu(tf.add(tf.matmul(x, W['HL_police']), b['HL_police']))
45      OL = tf.nn.sigmoid(tf.add(tf.matmul(HL, W['OL_police']), b['OL_police']))
46      return OL
47
48  # Network Inputs
49  IN_THIEF  = tf.placeholder(tf.float32, shape=[None, dim_noise])
50  IN_POLICE = tf.placeholder(tf.float32, shape=[None, dim_image])
```

TensorFlow for GAN

```
52  # Build Thief/Generator Neural Network
53  sample_thief = nn_thief(IN_THIEF)
54
55  # Build Police/Discriminator Neural Network (one from noise input, one from generated samples)
56  police_data_real = nn_police(IN_POLICE)
57  police_data_fake = nn_police(sample_thief)
58  vars_thief  = [W['HL_thief'],  W['OL_thief'],  b['HL_thief'],  b['OL_thief'] ]
59  vars_police = [W['HL_police'], W['OL_police'], b['HL_police'], b['OL_police']]
60
61  # Cost, Train
62  cost_thief   = -tf.reduce_mean(tf.log(police_data_fake))
63  cost_police  = -tf.reduce_mean(tf.log(police_data_real) + tf.log(1. - police_data_fake))
64  train_thief  = tf.train.AdamOptimizer(0.0002).minimize(cost_thief,  var_list=vars_thief)
65  train_police = tf.train.AdamOptimizer(0.0002).minimize(cost_police, var_list=vars_police)
```

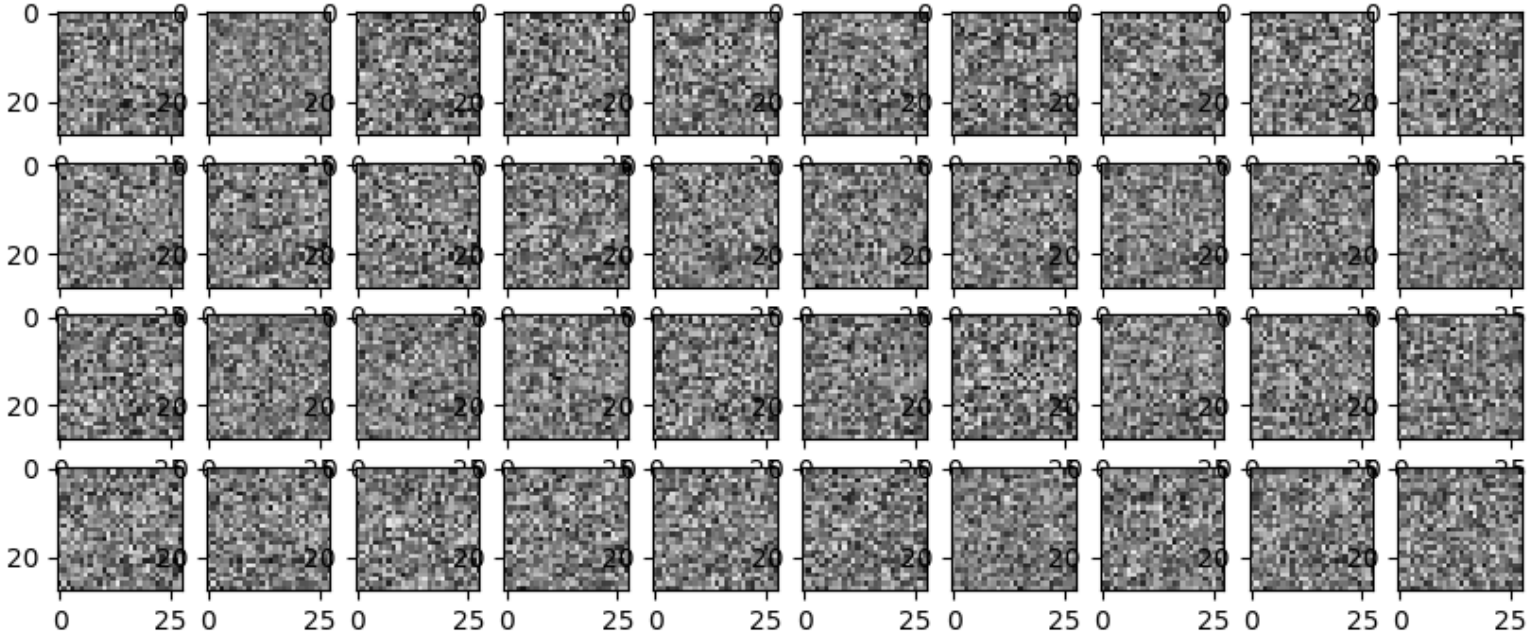
GAN architecture



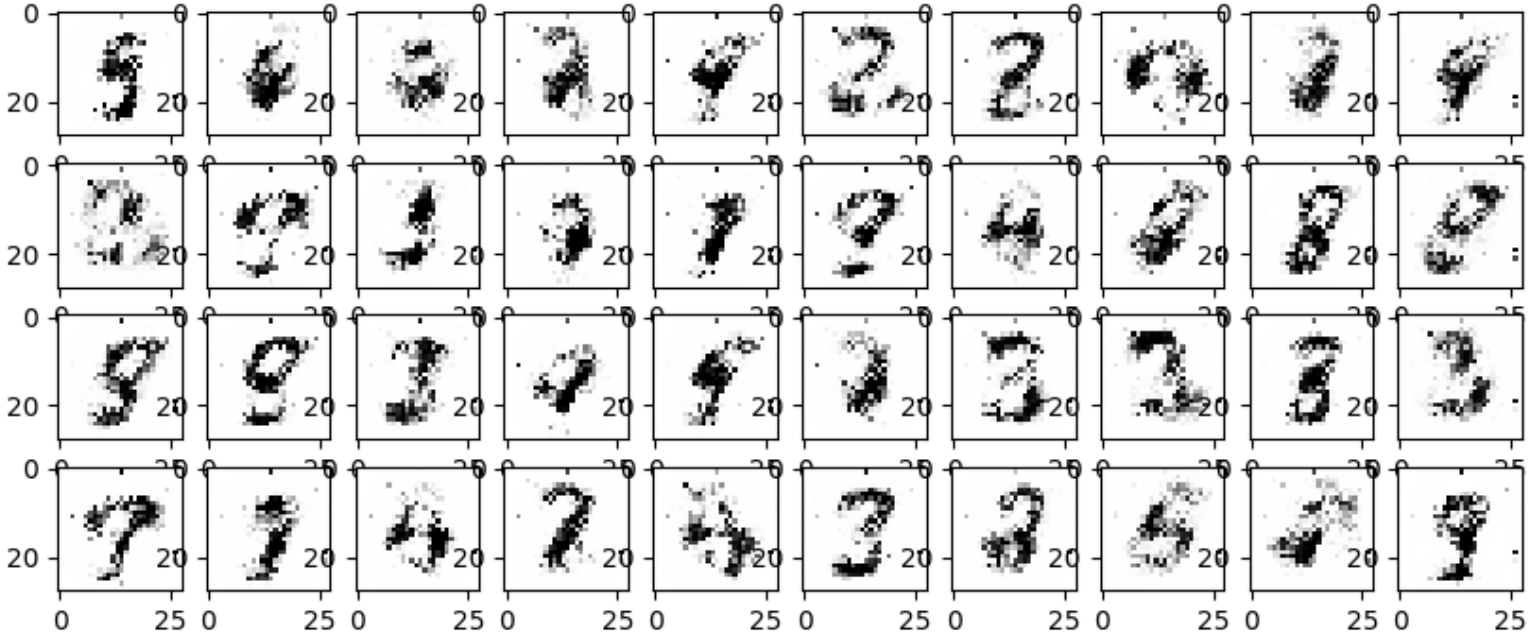
TensorFlow for GAN

```
# Session
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(1, num_steps+1):
        # Prepare Data
        # Get the next batch of MNIST data (only images are needed, not labels)
        batch_images, _ = mnist.train.next_batch(batch_size)
        # Generate noise to feed to the generator/thief
        z = np.random.uniform(-1., 1., size=[batch_size, dim_noise])
        # Train
        sess.run([train_thief, train_police, cost_thief, cost_police], feed_dict = {IN_POLICE: batch_images, IN_THIEF: z})
        # Generate images from noise, using the generator network.
        f, a = plt.subplots(4, 10, figsize=(10, 4))
        for i in range(10):
            # Noise input
            z = np.random.uniform(-1., 1., size=[4, dim_noise])
            g = sess.run([sample_thief], feed_dict={IN_THIEF: z})
            g = np.reshape(g, newshape=(4, 28, 28, 1))
            # Reverse colors for better display
            g = -1 * (g - 1)
            for j in range(4):
                # Generate image from noise. Extend to 3 channels for matplot figure.
                img = np.reshape(np.repeat(g[j][:, :, np.newaxis], 3, axis=2), newshape=(28, 28, 3))
                a[j][i].imshow(img)
        print(i)
    f.show()
    plt.draw()
    plt.waitforbuttonpress()
```

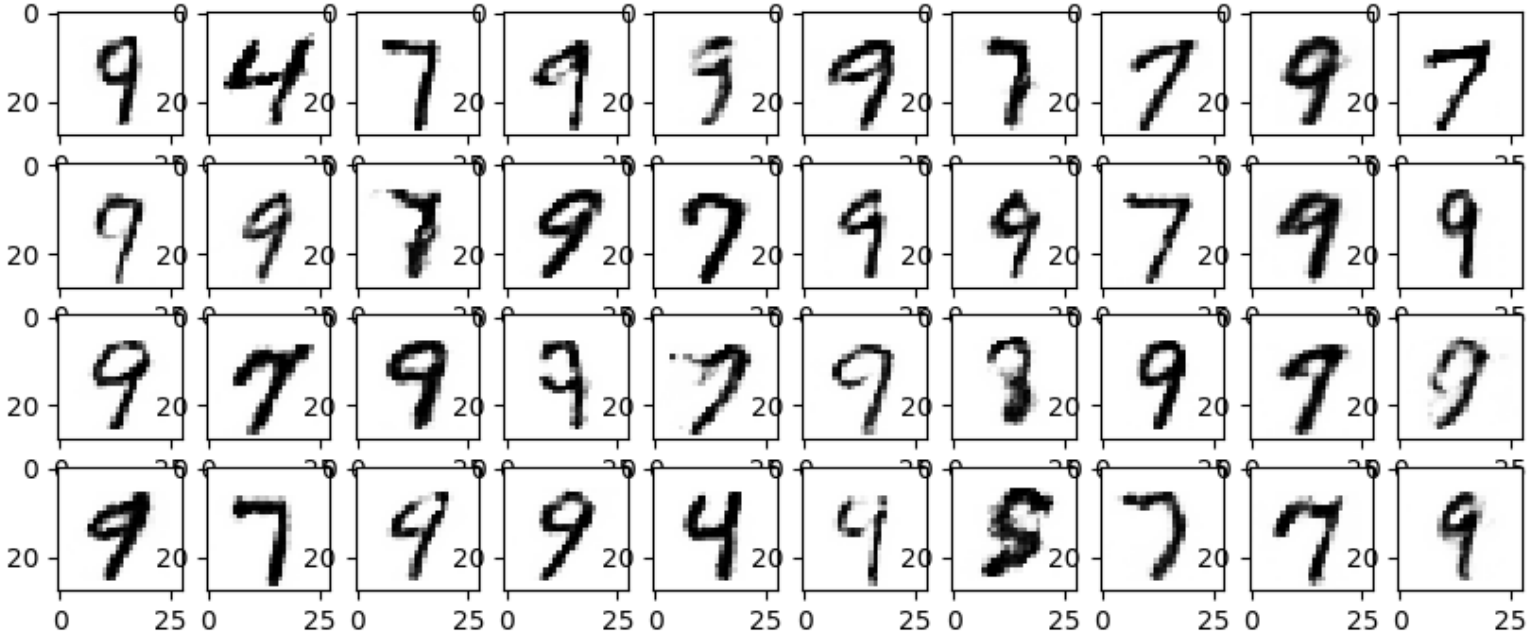
num_steps: 1



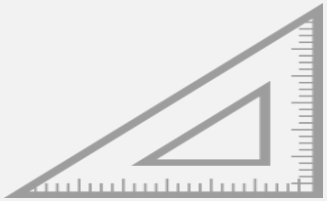
num_steps: 10000



num_steps: 100000



Linear Functions



Linear Regression
Binary Classification
Softmax Classification

Nonlinear Functions

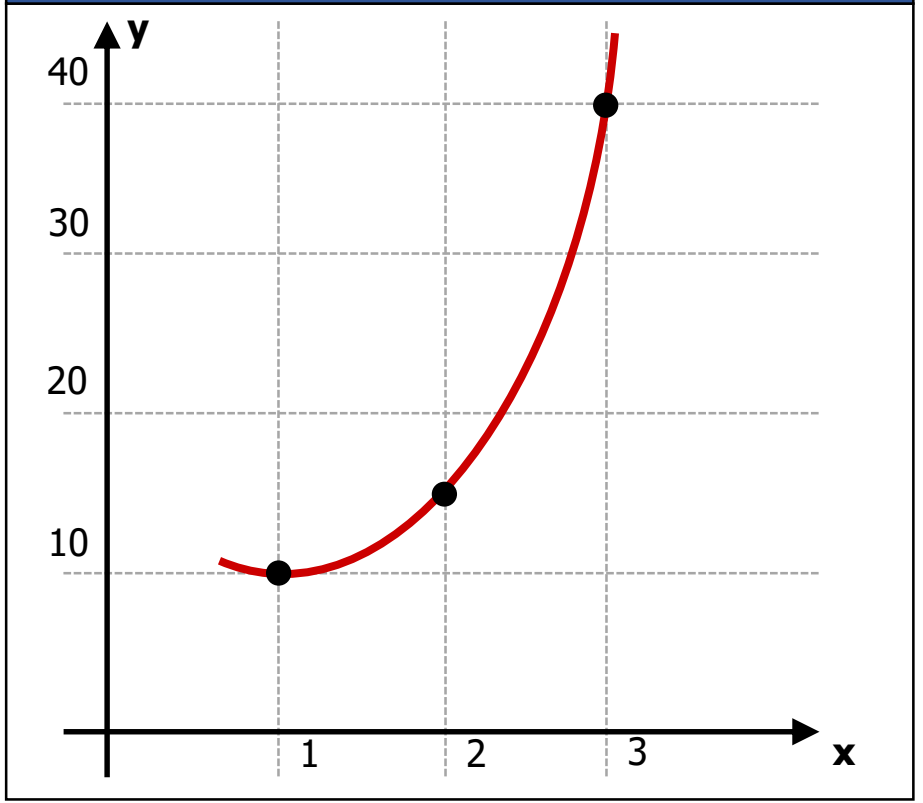
Neural Network (NN)
Convolutional NN (CNN)
CNN for CIFAR-10
Recurrent NN (RNN)

Advanced Topics

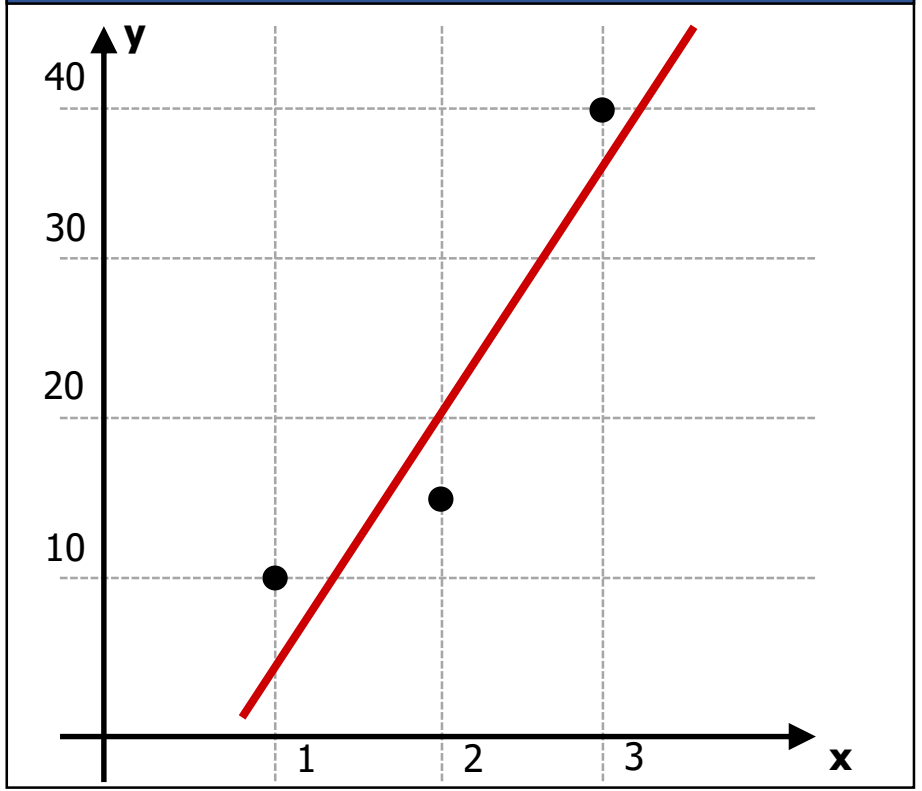
Gen. Adv. Network (GAN)
Interpolation
PCA/LDA
Overfitting

Interpolation vs. Linear Regression

Interpolation

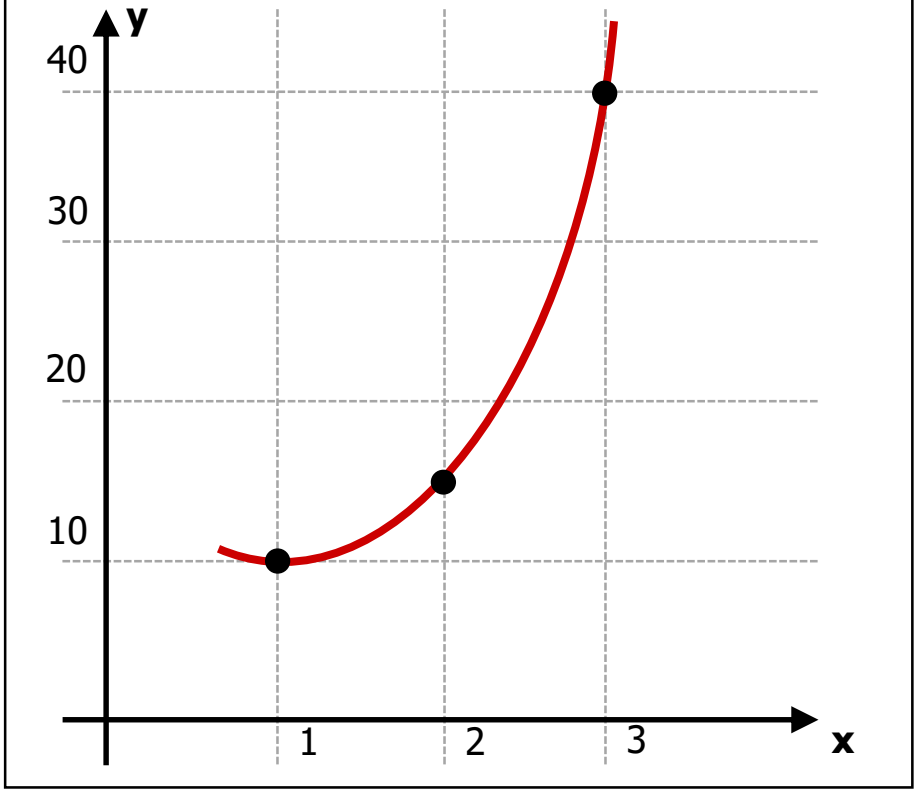


Linear Regression



Interpolation vs. Linear Regression

Interpolation

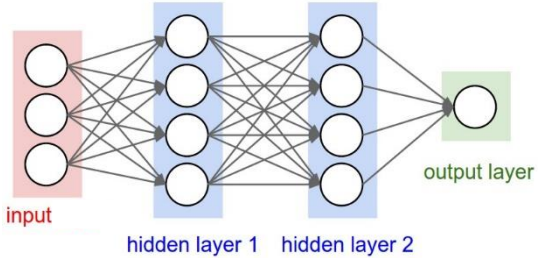


Interpolation with Polynomials

$$y = a_2 x^2 + a_1 x^1 + a_0$$

where three points are given.
→ Unique coefficients (a_0, a_1, a_2) can be calculated.

Is this related to
Neural Network Training?



$$Y = a(a(a(X \cdot W_1 + b_1) \cdot W_2 + b_2) \cdot W_o + b_o)$$

where training data/labels (X: data, Y: labels) are given.

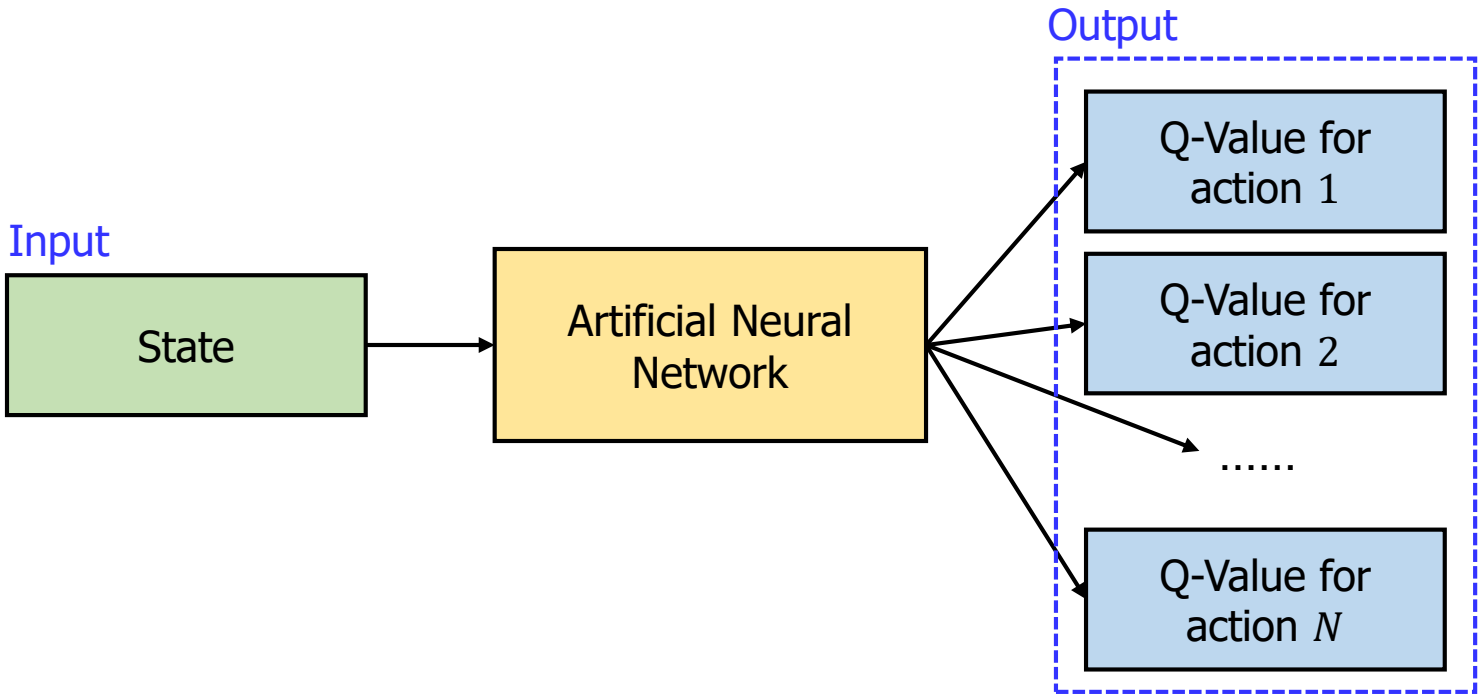
→ Find $W_1, b_1, W_2, b_2, W_o, b_o$

→ This is the mathematical meaning of neural network training.

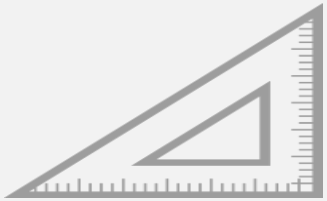
→ **Function Approximation**

→ The most well-known function approximation with neural network: **Deep Reinforcement Learning**

- It is inefficient to make the Q-table for each state-action pair.
  → ANN is used to **approximate the Q-function**.



Linear Functions



Linear Regression
Binary Classification
Softmax Classification

Nonlinear Functions

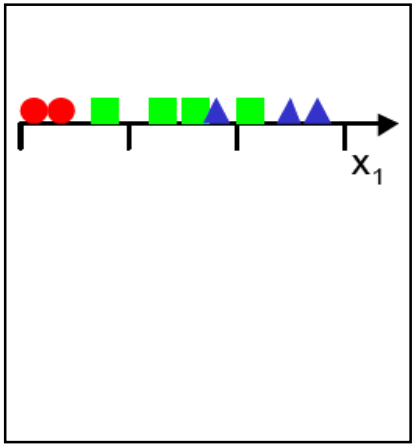
Neural Network (NN)
Convolutional NN (CNN)
CNN for CIFAR-10
Recurrent NN (RNN)

Advanced Topics

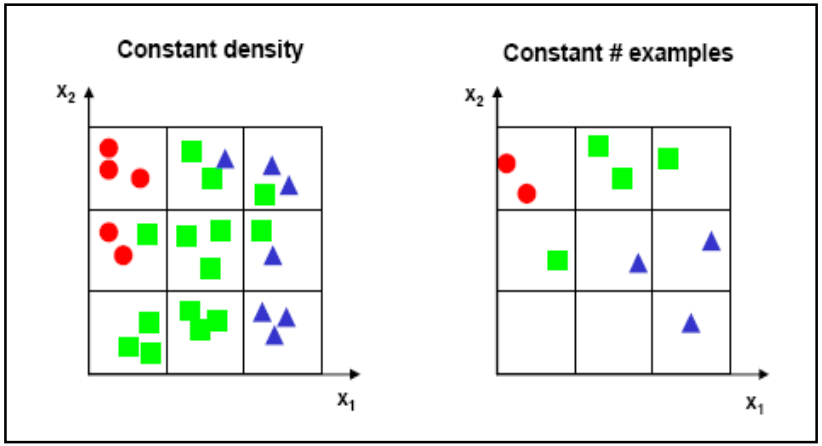
Gen. Adv. Network (GAN)
Interpolation
PCA/LDA
Overfitting

Curse of Dimensionality

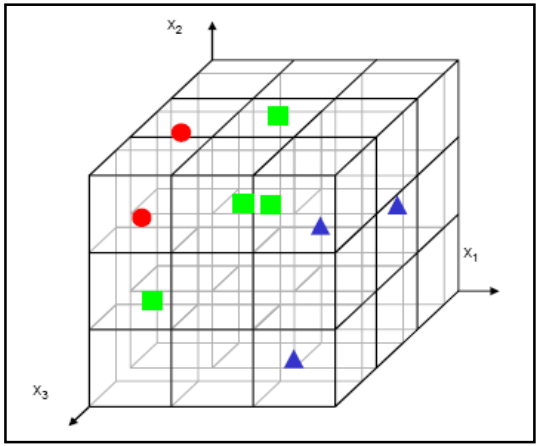
1D

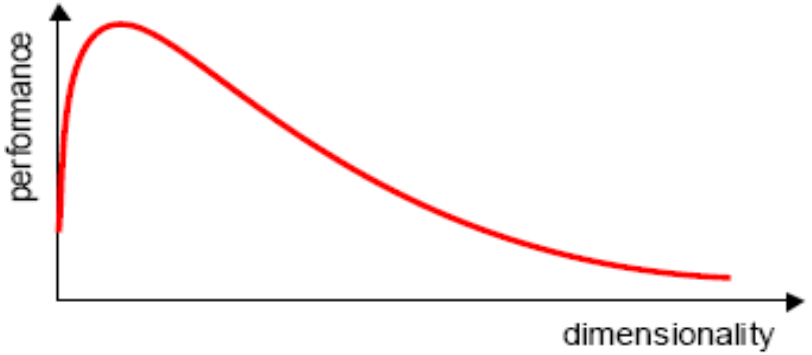


2D



3D





Dimensionally Reduction

Feature Selection

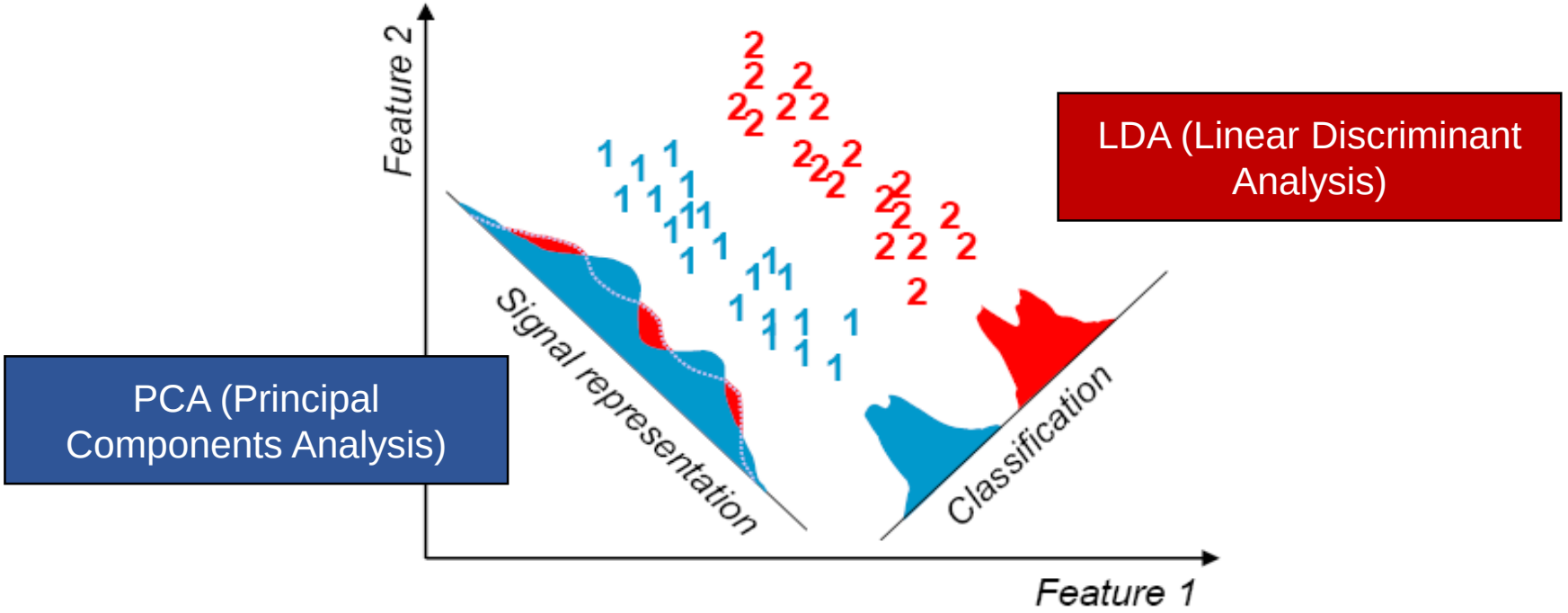
$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \xrightarrow{\text{feature selection}} \begin{bmatrix} x_{i_1} \\ x_{i_2} \\ \\ x_{i_M} \end{bmatrix}$$

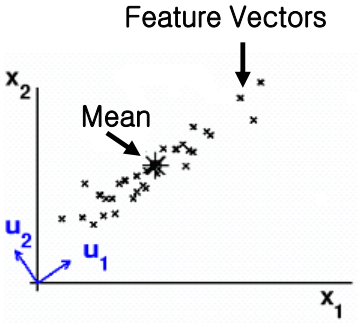
Feature Extraction

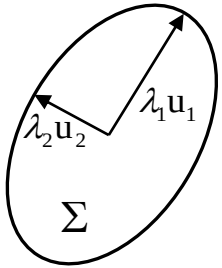
$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \xrightarrow{\text{feature extraction}} \begin{bmatrix} y_1 \\ y_2 \\ \\ y_M \end{bmatrix} = f\left(\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}\right)$$

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \xrightarrow{\text{linear feature extraction}} \begin{bmatrix} y_1 \\ y_2 \\ \\ y_M \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M1} & w_{M2} & & w_{MN} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

Dimensionally Reduction: PCA and LDA

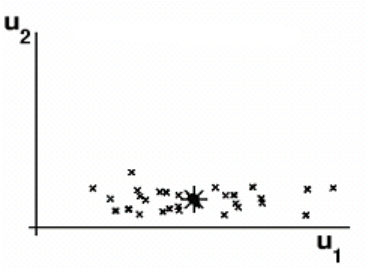


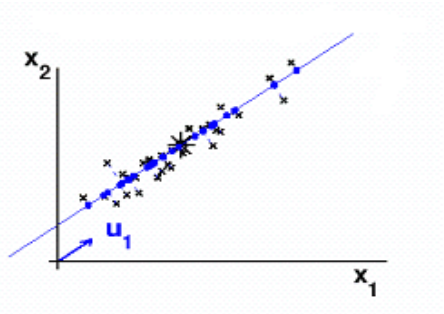


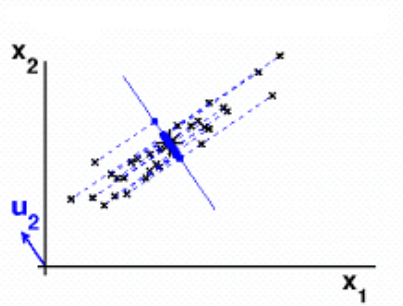


$$\Leftrightarrow \Sigma = [u_1 u_2] \times \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \times [u_1 u_2]^T$$

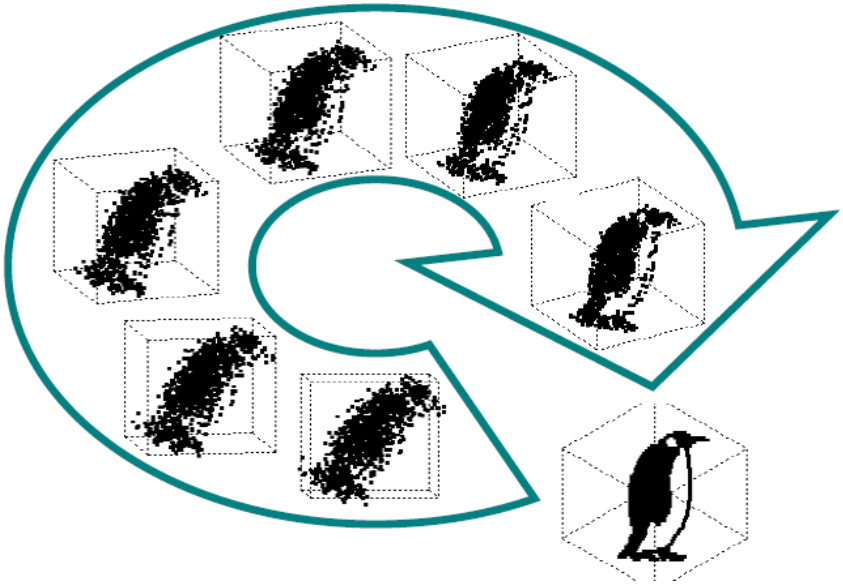
$$\lambda_1 > \lambda_2$$

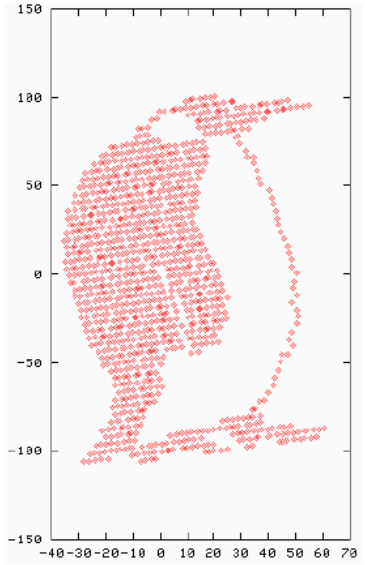




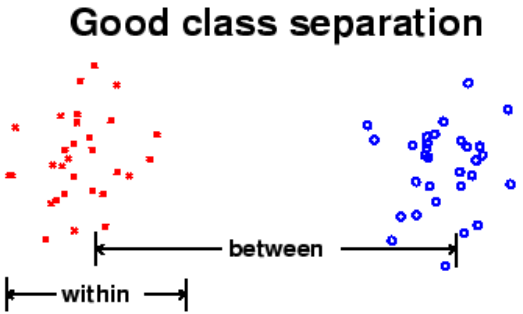


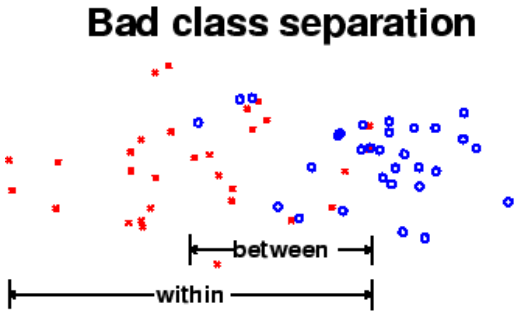
PCA Example

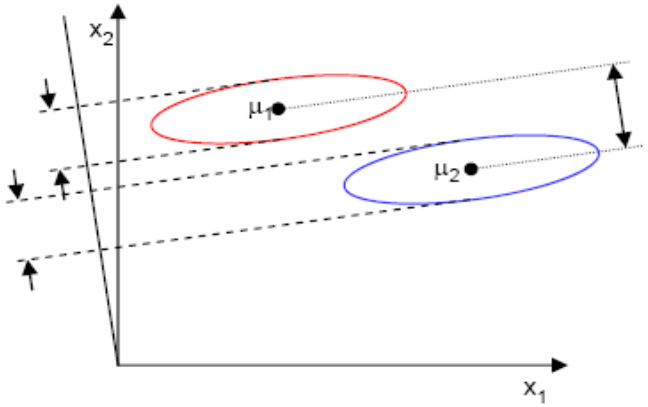




PCA Example: Face Recognition







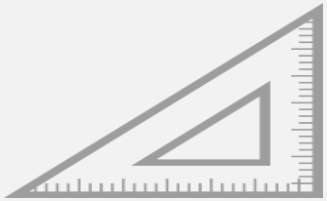
Fisher's linear discriminant function

$$J(\mathbf{w}) = \frac{\left|\tilde{\boldsymbol{\mu}}_1 - \tilde{\boldsymbol{\mu}}_2\right|^2}{\tilde{\mathbf{S}}_1^2 + \tilde{\mathbf{S}}_2^2}$$

→ Maximize!

→ Minimize!

Linear Functions



Linear Regression
Binary Classification
Softmax Classification

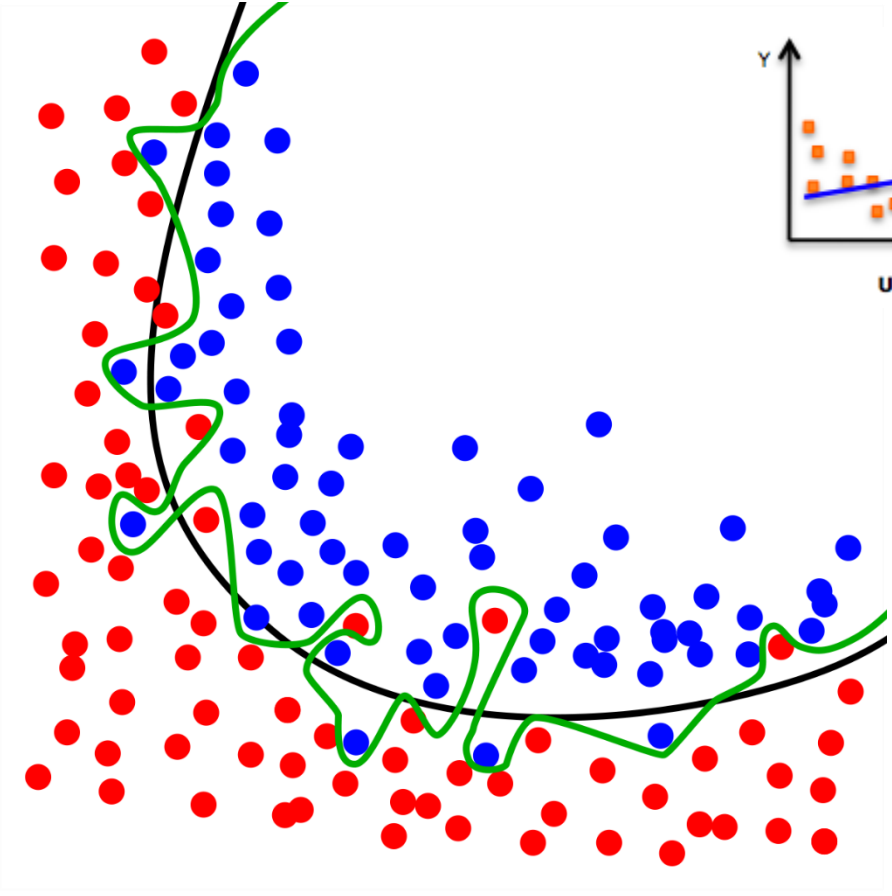
Nonlinear Functions

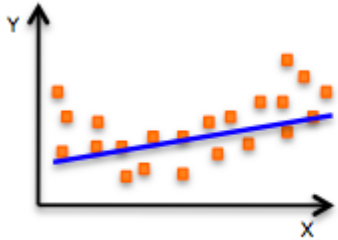
Neural Network (NN)
Convolutional NN (CNN)
CNN for CIFAR-10
Recurrent NN (RNN)

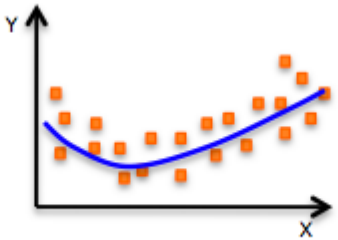
Advanced Topics

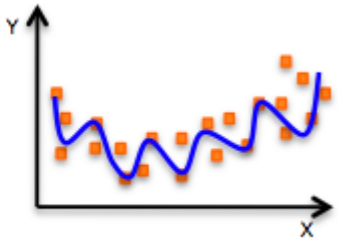
Gen. Adv. Network (GAN)
Interpolation
PCA/LDA
Overfitting

Overfitting

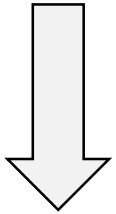






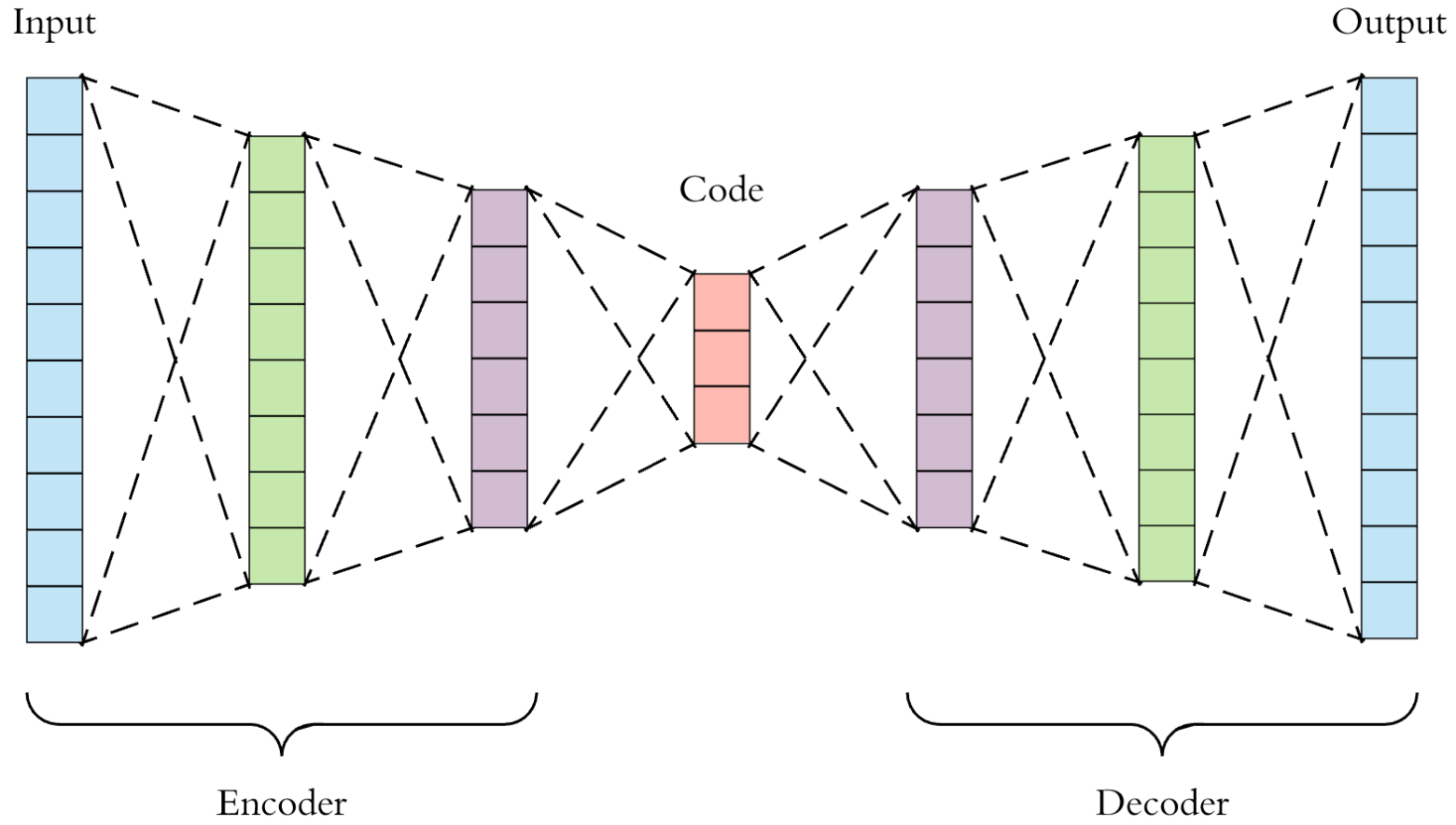


- How to overcome?
  - More training data
  - Reduce the number of features → autoencoding, dropout
  - Regularization

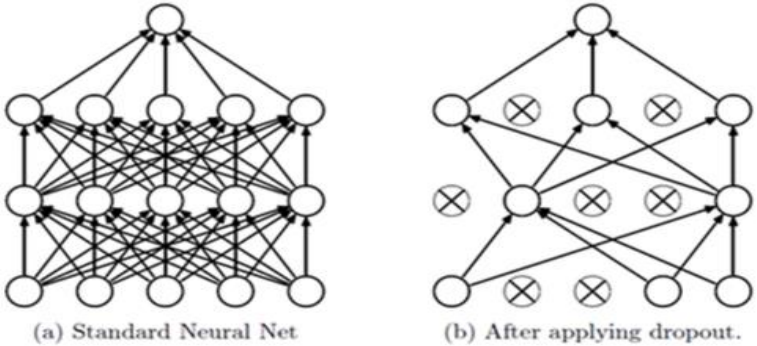


- **Autoencoding**
- **Dropout**
- **Regularization**

Autoencoding

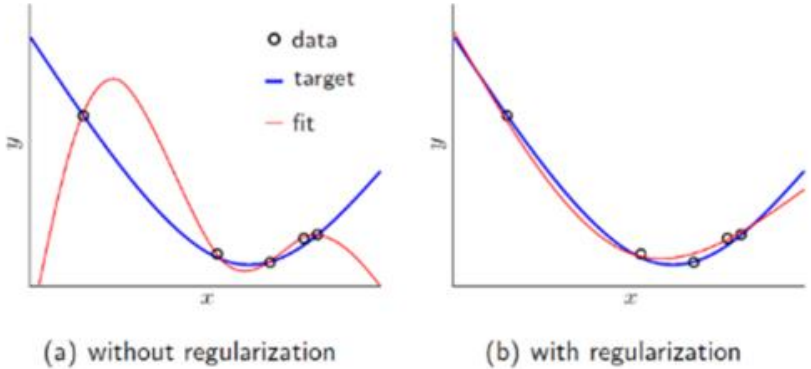


- Dropout
  - **tf.nn.dropout(layer, keep_prob=0.9)**


(a) Standard Neural Net

(b) After applying dropout.

- Regularization



(a) without regularization

(b) with regularization