

Deep Learning Theory and Software

Basics

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

Geoffrey E Hinton



Yoshua Bengio



Yann LeCun



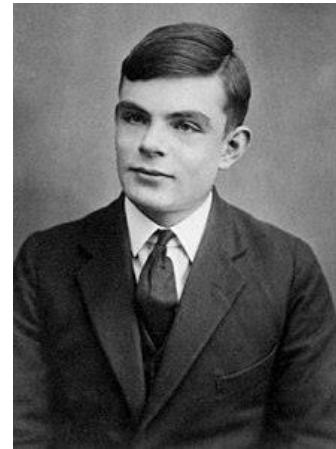
FATHERS OF THE DEEP LEARNING REVOLUTION RECEIVE ACM A.M. TURING AWARD (2019)

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.

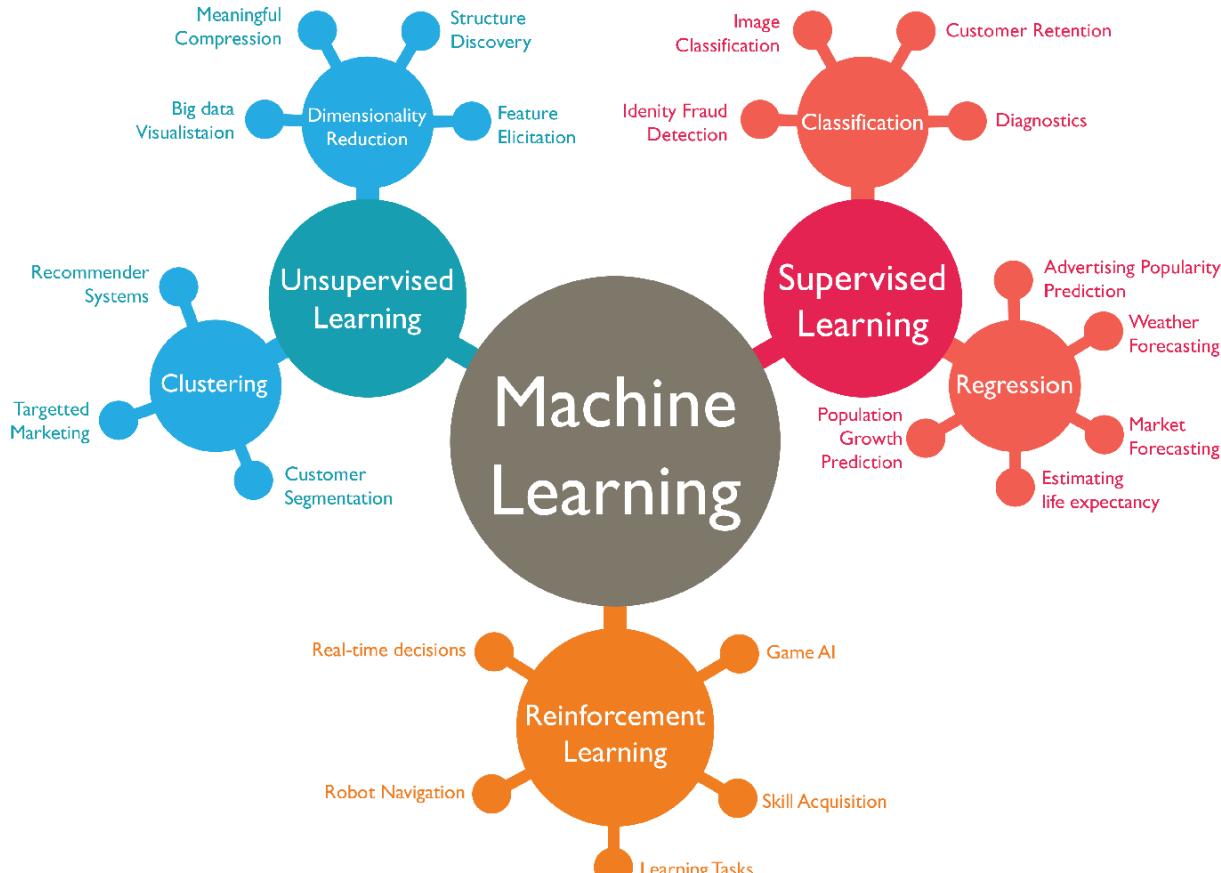
Working independently and together, Hinton, LeCun and Bengio developed conceptual foundations for the field, identified surprising phenomena through experiments, and contributed engineering advances that demonstrated the practical advantages of deep neural networks. In recent years, deep learning methods 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.

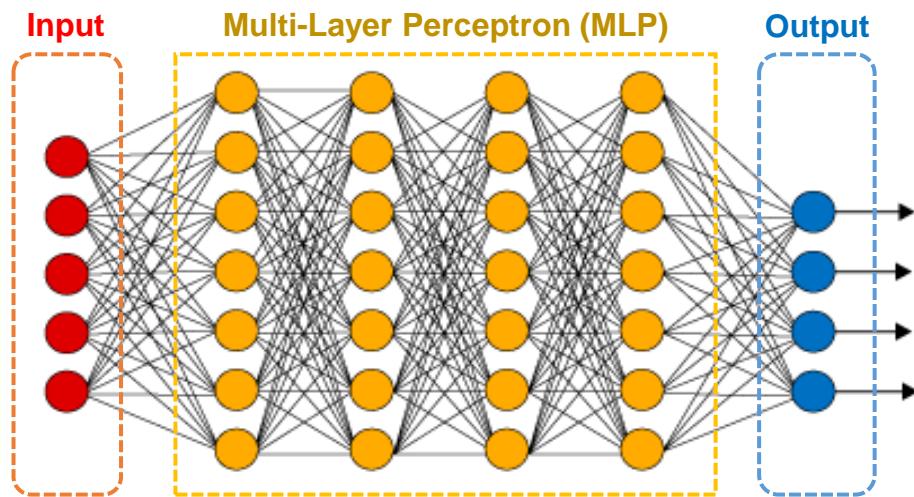
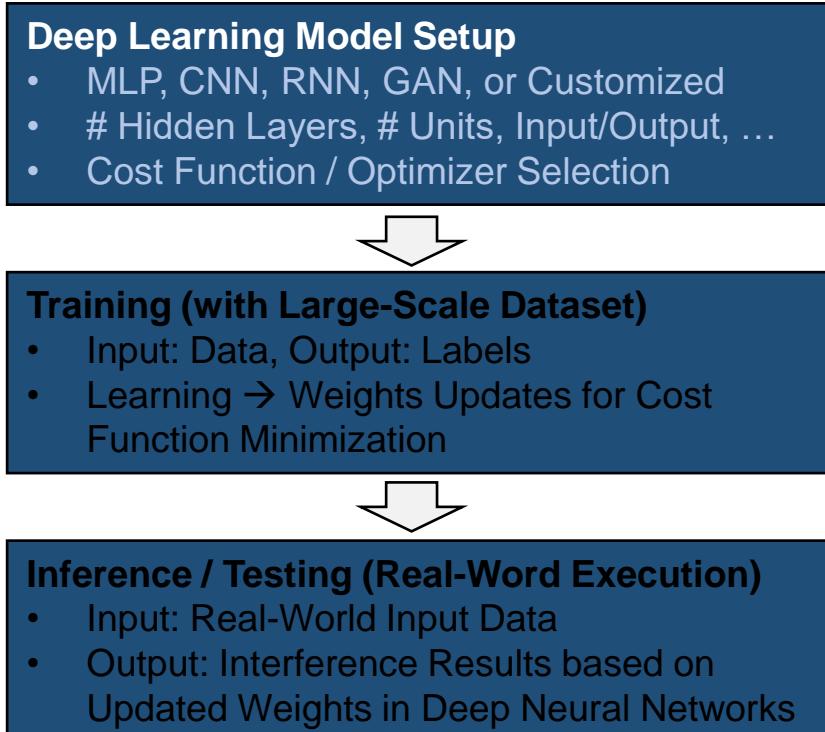


Alan Turing (1912-1954)
Father of Computer Science

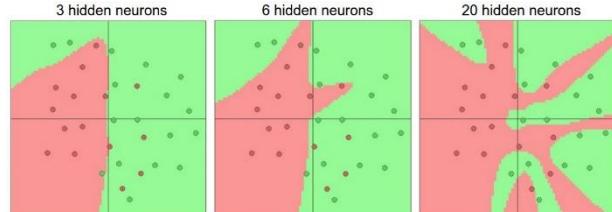
<https://amturing.acm.org/>



- How Deep Learning Works?
 - Deep Learning Computation Procedure



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



Introduction

- 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)

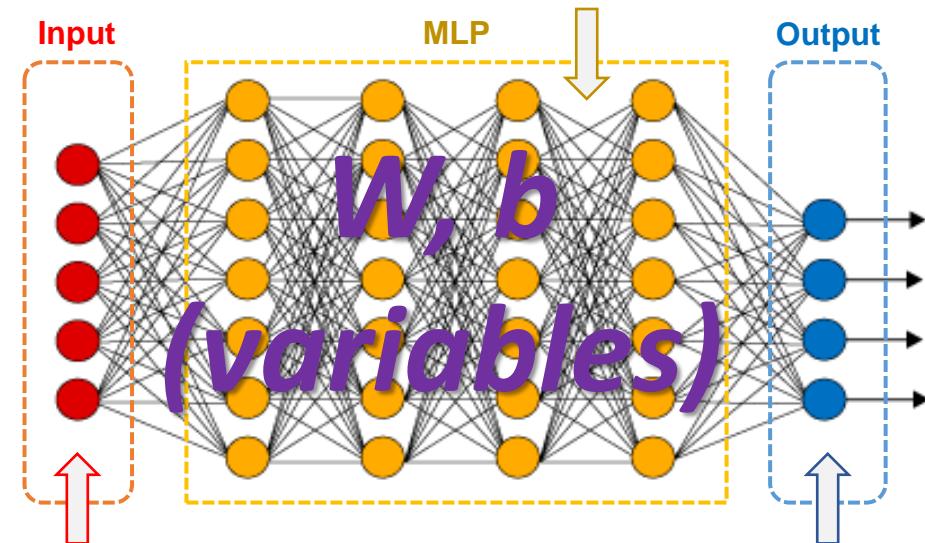
- Input: Real-World Input Data
- Output: Interference Results based on Updated Weights in Deep Neural Networks

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

Input

MLP

Output



INPUT: Data

- One-Dimension Vector

OUTPUT: Labels

- One-Hot Encoding

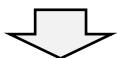
We need a lot of training data for generality
(otherwise, we will suffer from overfitting problem).

Introduction

- 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

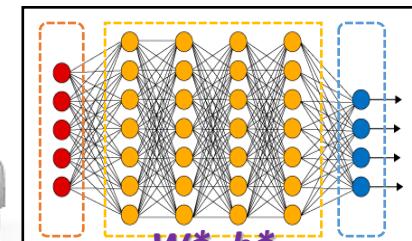


Inference / Testing (Real-World Execution)

- Input: Real-World Input Data
- Output: Inference Results based on Updated Weights in Deep Neural Networks



Trained Model



(constants)

Intelligent
Surveillance
Platforms

INPUT: Real-Time Arrivals

OUTPUT: Inference

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

Introduction

- How Deep Learning Works?

- Issue - Overfitting

Deep Learning Model Setup

- MLP, CNN, RNN, GAN, or Custom
- # Hidden Layers, # Units, Input/Output, ...
- Cost Function / Optimizer Selection

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

Training (with Large-Scale Dataset)

- Input: Data, Output: Labels
- Learning → Weights Updates for Cost Function Minimization

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



Inference / Testing (Real-World Execution)

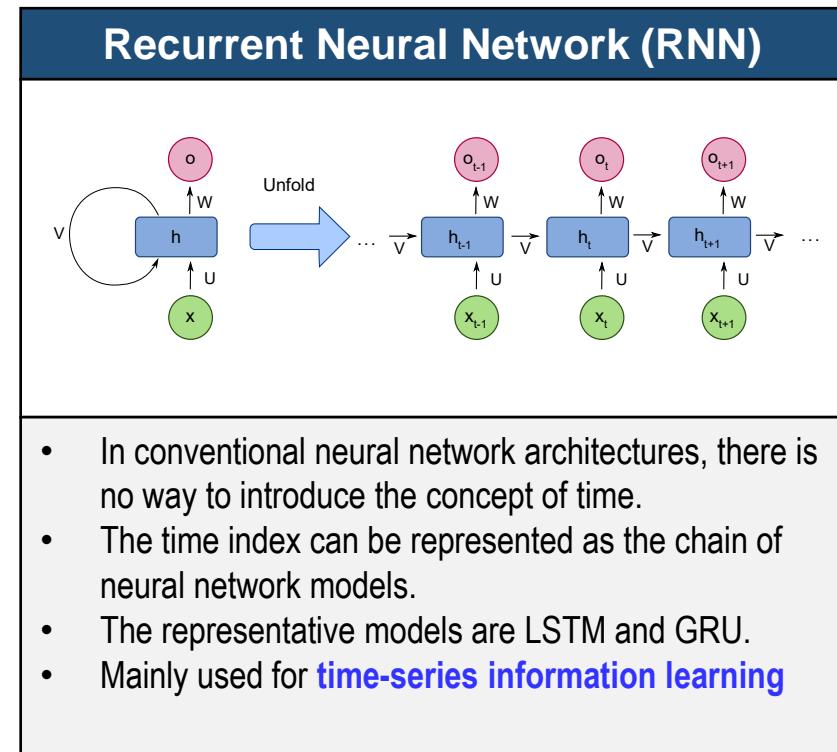
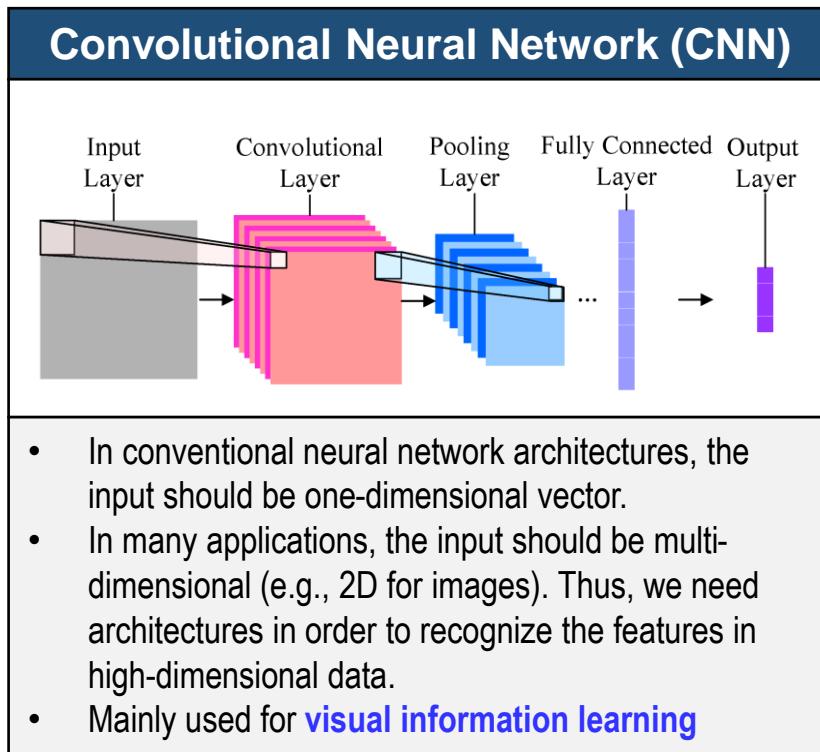
- Input: Real-World Input Data
- Output: Inference Results based on Updated Weights in Deep Neural Networks

To Combat the Overfitting

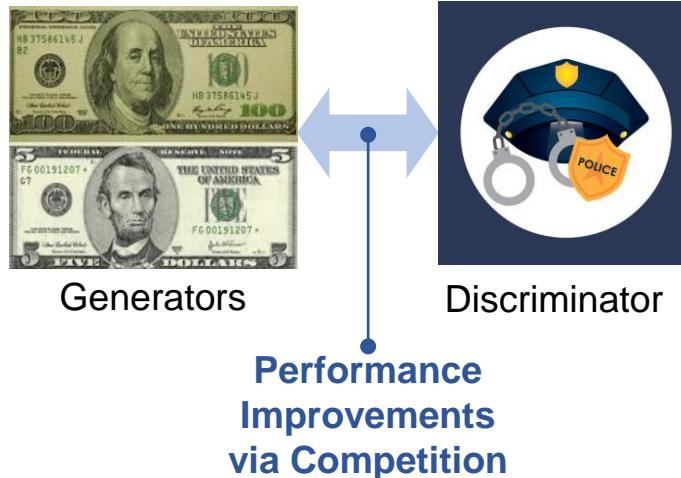
- More training data
- Autoencoding (or variational auto-encoder (VAE))
- Dropout
- Regularization

Introduction

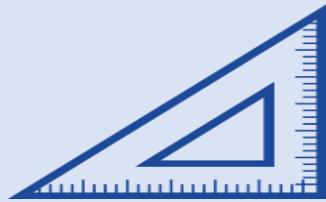
- Two Major Deep Learning Models → CNN vs. RNN



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



Linear Functions



Linear Regression

Binary Classification

Softmax Classification

Nonlinear Functions



Neural Network (NN)

Convolutional NN (CNN)

CNN for CIFAR-10

Advanced Topics



Gen. Adv. Network (GAN)

Interpolation

PCA/LDA

Overfitting



Deep Learning Basics and Software

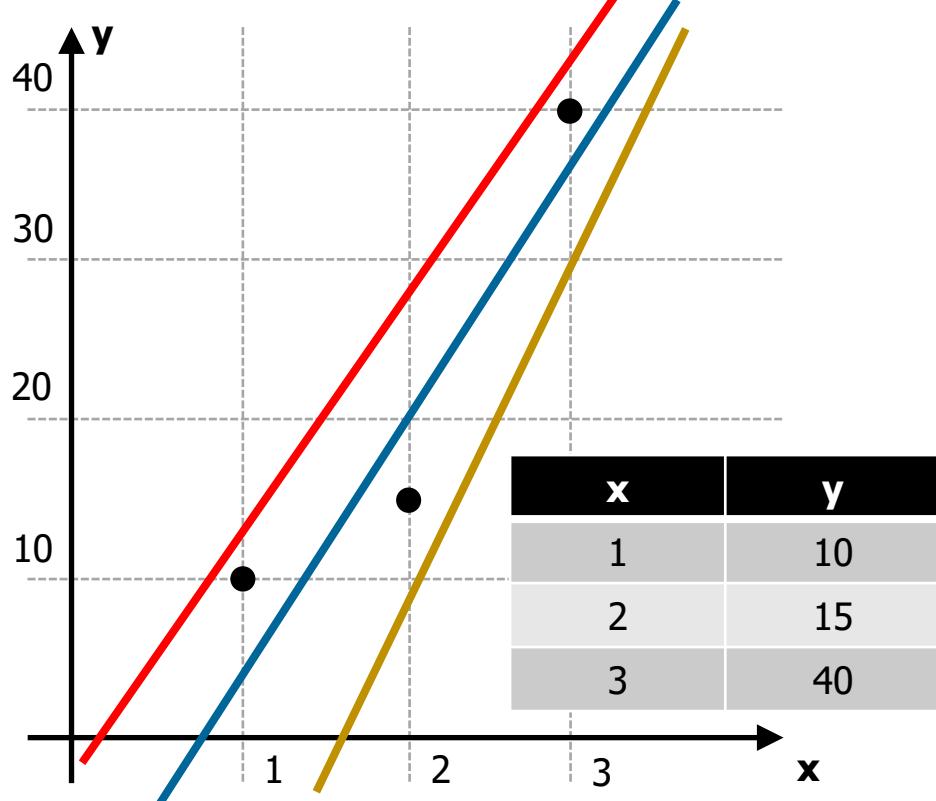
Linear Regression

Linear Regression Theory

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

Linear Regression

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



- Cost Function (or Loss Function)

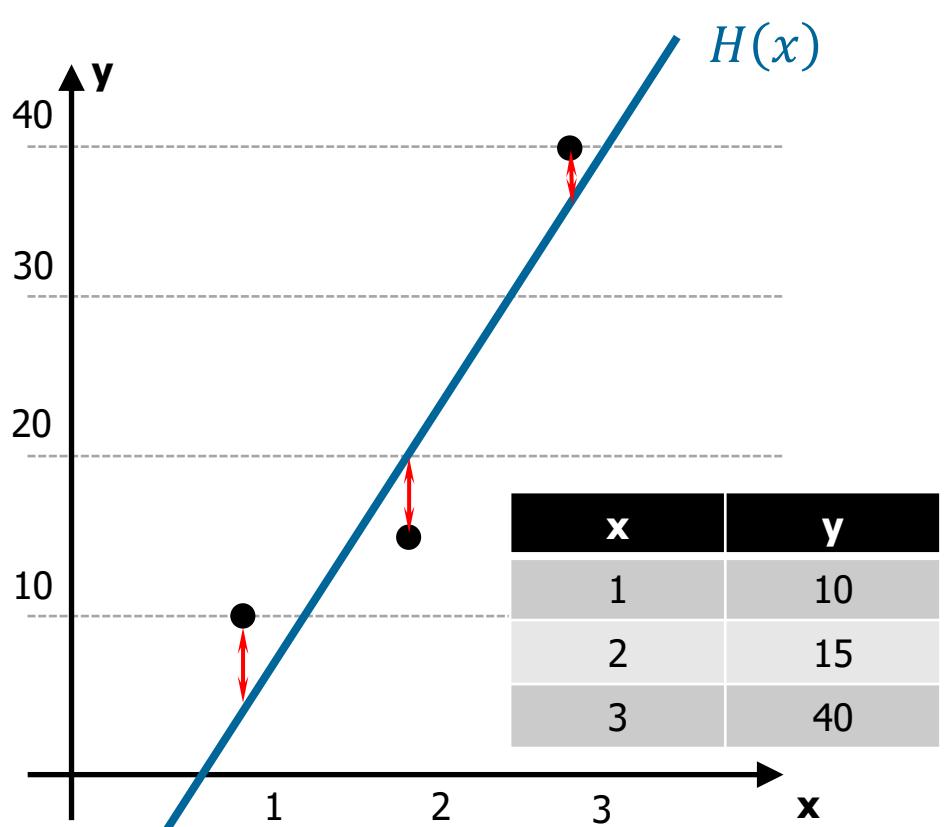
m : The number of training data

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

↓
 $H(x) = Wx + b$

Cost(W, b) =

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



- Cost Function Minimization

- Model: $H(x) = Wx + b$

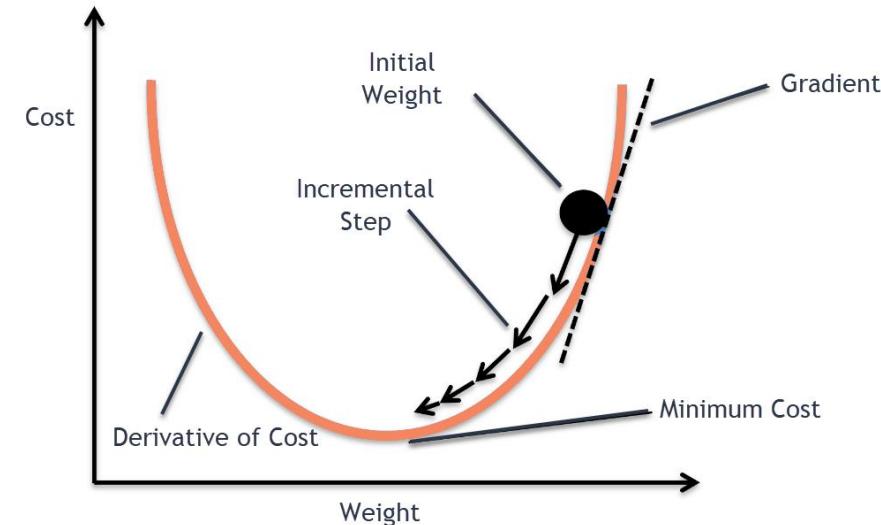
- Cost Function: $Cost(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x^i) - y^i)^2 = \frac{1}{m} \sum_{i=1}^m (Wx^i + b - y^i)^2$

- How to Minimize this Function? → **Gradient Descent Method**

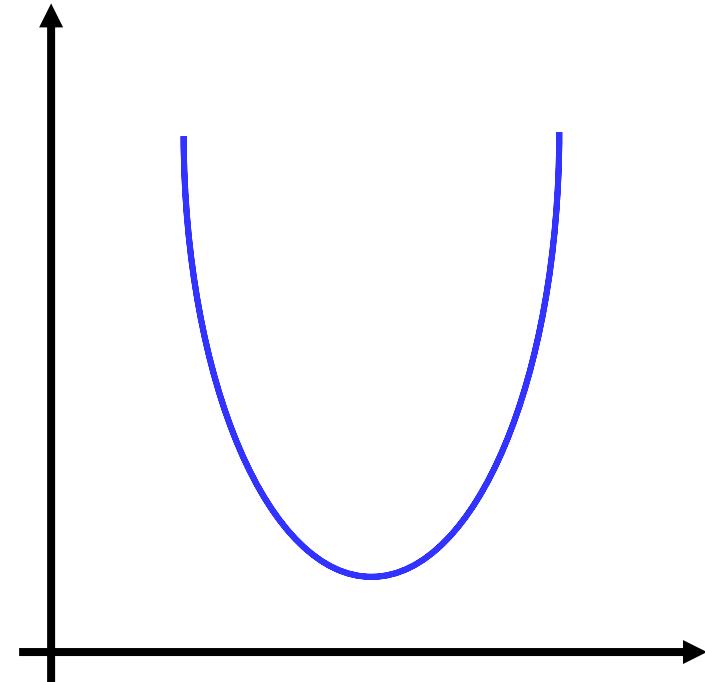
- Angle → Differentiation

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

α : Learning rate



- Learning Rates
 - Too large: Overshooting
 - Too small: taking very long time to converge
- How can we determine the learning rates?
 - Try several learning rates
 - Observe the cost function



- Multi-Variable Linear Regression

- Model:

$$H(x_1, x_2, \dots, x_n) = w_1x_1 + w_2x_2 + \dots + w_nx_n + b$$

- Cost:

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

• Multi-Variable Linear Regression

• Model:

$$H(x_1, x_2, \dots, x_n) = w_1x_1 + w_2x_2 + \dots + w_nx_n + b$$

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

$$(x_1 \quad x_2 \dots \quad x_n) \cdot \begin{pmatrix} w_1 \\ w_2 \\ \dots \\ w_n \end{pmatrix} = w_1x_1 + w_2x_2 + \dots + w_nx_n$$

 X W 

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

$$\text{when } W = (w_1 \quad w_2 \dots \quad w_n)$$

Deep Learning Basics and Software

Linear Regression

Linear Regression Implementation

- Linear Regression Theory
- Linear Regression Implementation

- **TensorFlow**
- PyTorch
- Keras

- **TensorFlow**
- PyTorch
- Keras

Linear Regression Implementation (TensorFlow)

```
1 import tensorflow as tf
2 import numpy as np
3 x_data = np.array([[1,1],[2,2],[3,3]], dtype=np.float32)
4 y_data = np.array([[10],[20],[30]], dtype=np.float32)
5 W = tf.Variable(tf.random.normal([2,1]))
6 b = tf.Variable(tf.random.normal([1]))
7
8 def model_LinearRegression(x):
9     return tf.matmul(x,W)+b
10
11 def cost_LinearRegression(model_x):
12     return tf.reduce_mean(tf.square(model_x-y_data))
13
14 def train_optimization(x):
15     with tf.GradientTape() as g:
16         model = model_LinearRegression(x)
17         cost = cost_LinearRegression(model)
18         gradients = g.gradient(cost,[W,b])
19         tf.optimizers.SGD(0.01).apply_gradients(zip(gradients,[W,b]))
20
21 for step in range(2001):
22     train_optimization(x_data)
23
24 print('*'*100)
25 x_test = np.array([[4,4]],dtype=np.float32)
26 model_test = model_LinearRegression(x_test)
27 print("model for [4,4]: ", model_test.numpy())
28 print('*'*100)
```

- TensorFlow
- **PyTorch**
- Keras

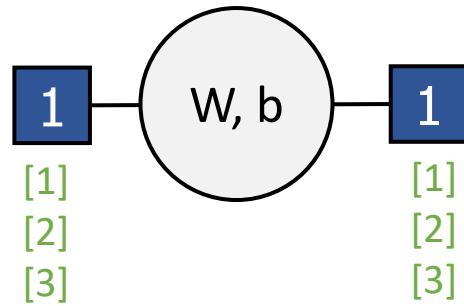
```
1 import torch
2 x_train = torch.FloatTensor([[1,1], [2,2], [3,3]])
3 y_train = torch.FloatTensor([[10], [20], [30]])
4 W = torch.randn([2,1], requires_grad=True)
5 b = torch.randn([1], requires_grad=True)
6 optimizer = torch.optim.SGD([W, b], lr=0.01)
7
8 def model_LinearRegression(x):
9     return torch.matmul(x, W) + b
10
11 for step in range(2000):
12     prediction = model_LinearRegression(x_train)
13     cost = torch.mean((prediction - y_train) ** 2)
14     optimizer.zero_grad()
15     cost.backward()
16     optimizer.step()
17
18 x_test = torch.FloatTensor([[4,4]])
19 model_test = model_LinearRegression(x_test)
20 print("Model with [4] (expectation: 40): ", model_test.detach().item())
```

```
1 import torch
2 import torch.nn as nn
3 import torch.nn.functional as F
4 x_train = torch.FloatTensor([[1,1], [2,2], [3,3]])
5 y_train = torch.FloatTensor([[10], [20], [30]])
6
7 class LinearRegression(nn.Module):
8     def __init__(self):
9         super().__init__()
10        self.linear = nn.Linear(2,1)
11    def forward(self, x):
12        return self.linear(x)
13
14 model_LinearRegression = class_LinearRegression()
15 optimizer = torch.optim.SGD(model_LinearRegression.parameters(), lr=0.01)
16
17 for step in range(2000):
18     prediction = model_LinearRegression(x_train)
19     cost = F.mse_loss(prediction, y_train)
20     optimizer.zero_grad()
21     cost.backward()
22     optimizer.step()
23
24 x_test = torch.FloatTensor([[4,4]])
25 model_test = model_LinearRegression(x_test)
26 print("Model with [4] (expectation: 40): ", model_test.detach().item())
```

- TensorFlow
- PyTorch
- Keras

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)    Model, Cost, Train
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()
```



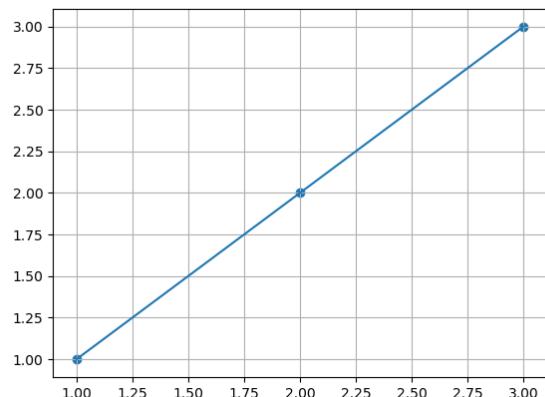
Linear Regression Implementation (Keras)



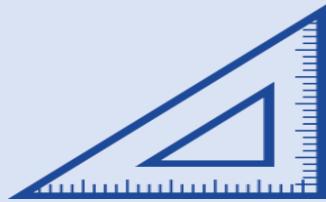
```
x - joongheon@joongheon-AB350M-Gaming-3: ~/Dropbox/codes_keras
joongheon@joongheon-AB350M-Gaming-3:~/Dropbox/codes_keras$ python keras_linearregression.py
/home/joongheon/anaconda3/lib/python3.6/site-packages/h5py/_init_.py:36: FutureWarning: Conversion of the second argument of issubdtype from `float` 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 supports 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

Advanced Topics



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



Deep Learning Basics and Software

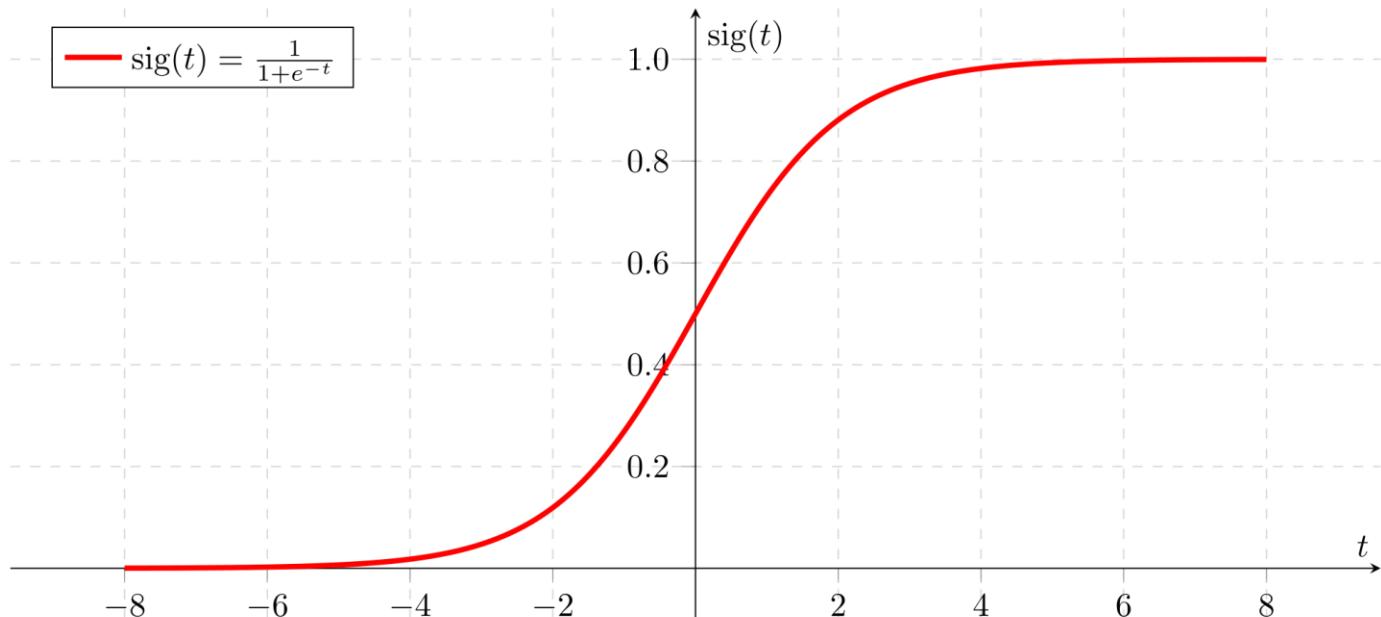
Binary Classification

Binary Classification Theory

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

- Binary Classification Examples
 - **Spam Detection:** Spam [1] or Ham [0]
 - **Facebook Feed:** Show [1] or Hide [0]
 - **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 ($\text{sig}(t)$)** based on the result of Step 1.



Linear Regression Model

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

Binary Classification Model

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

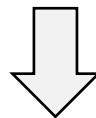
Logistic/Sigmoid Function

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

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

Linear Regression Model

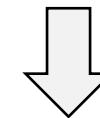
$$H(x) = Wx + b \text{ or } H(X) = W^T X$$



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

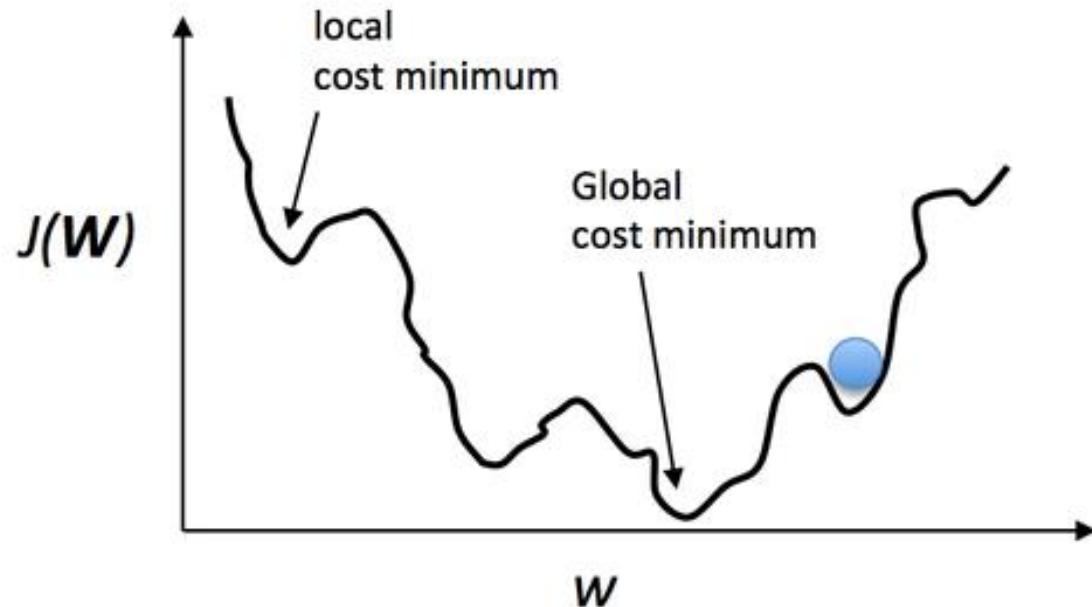
Binary Classification Model

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



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

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



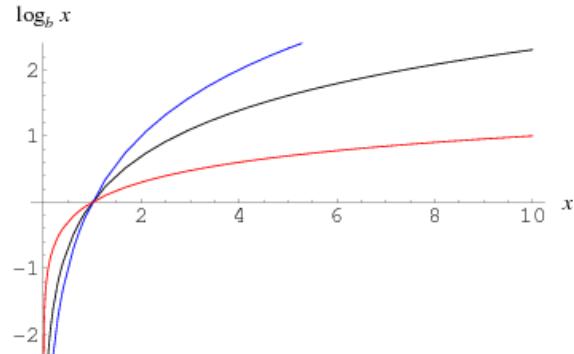
$$\text{Cost}(W) = \frac{1}{m} \sum c(H(x), y)$$

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

Understanding this Cost Function

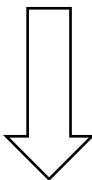
Cost	0	∞	∞	0
$H(x)$	0	0	1	1
y	0	1	0	1

Log Function



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

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



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



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



Gradient Descent Method

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

Deep Learning Basics and Software

Binary Classification

Binary Classification Implementation

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

- **TensorFlow**
- PyTorch
- Keras

Binary Classification Implementation (TensorFlow)

```
1 import tensorflow as tf
2 import numpy as np
3 x_train = np.array([[1,2], [2,3], [3,4], [4,4], [5,3], [6,2]], dtype=np.float32)
4 y_train = np.array([0, 0, 0, 1, 1, 1], dtype=np.float32)
5 W = tf.Variable(tf.random.normal([2,1]))
6 b = tf.Variable(tf.random.normal([1]))
7
8 def model_BinaryClassification(x):
9     return tf.sigmoid(tf.matmul(x,W)+b)
10
11 def cost_BinaryClassification(model_x):
12     return tf.reduce_mean((-1)*y_train*tf.math.log(model_x)+(-1)*(1.-y_train)*tf.math.log(1.-model_x))
13
14 def train_optimization(x):
15     with tf.GradientTape() as g:
16         model = model_BinaryClassification(x)
17         cost = cost_BinaryClassification(model)
18     gradients = g.gradient(cost,[W,b])
19     tf.optimizers.SGD(0.01).apply_gradients(zip(gradients,[W,b]))
20
21 for step in range(2001):
22     train_optimization(x_train)
23     if step % 100 == 0:
24         pred = model_BinaryClassification(x_train)
25         loss = cost_BinaryClassification(pred)
26         prediction = tf.cast(pred > 0.5, dtype=tf.float32)
27         accuracy = tf.reduce_mean(tf.cast(tf.equal(prediction, y_train), dtype=tf.float32))
28         print("Step: {},\t Accurarray: {},\t Loss: {}".format(step,accuracy.numpy().flatten(),loss))
29
30 print('*'*100)
31 x_test = np.array([[6,1]],dtype=np.float32)
32 model_test = model_BinaryClassification(x_test)
33 print("model for [6,1]: ", model_test.numpy())
34 print('*'*100)
```

- TensorFlow
- **PyTorch**
- Keras

```
1 import torch
2 import numpy as np
3 x_train = torch.FloatTensor([[1,2], [2,3], [3,4], [4,4], [5,3], [6,2]])
4 y_train = torch.FloatTensor([[0], [0], [0], [1], [1], [1]])
5 W = torch.randn([2,1], requires_grad=True)
6 b = torch.randn([1], requires_grad=True)
7 optimizer = torch.optim.SGD([W, b], lr=0.01)
8
9 def model_BinaryClassification(x):
10    return torch.sigmoid(torch.matmul(x, W) + b)
11
12 for step in range(2000):
13    prediction = model_BinaryClassification(x_train)
14    cost = torch.mean((-1)*y_train*torch.log(prediction)+(-1)*(1-y_train)*torch.log(1-prediction))
15    optimizer.zero_grad()
16    cost.backward()
17    optimizer.step()
18
19 x_test = torch.FloatTensor([[6,1]])
20 model_test = model_BinaryClassification(x_test)
21 print("Model with [6,1] (expectation: 1) in sigmoid: ", model_test.detach().item())
22 model_test_binary = np.round(model_test>0.5).type(torch.float32)
23 print("Model with [6,1] (expectation: 1) in np.round: ", model_test_binary.detach().item())
```

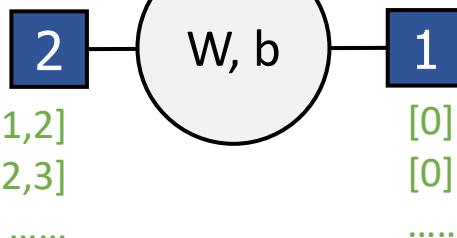
Binary Classification Implementation (PyTorch)



```
1 import torch
2 import torch.nn as nn
3 import torch.nn.functional as F
4 import numpy as np
5 x_train = torch.FloatTensor([[1,2], [2,3], [3,4], [4,4], [5,3], [6,2]])
6 y_train = torch.FloatTensor([[0], [0], [0], [1], [1], [1]])
7
8 class class_BinaryClassification(nn.Module):
9     def __init__(self):
10         super().__init__()
11         self.linear1 = nn.Linear(2,3)
12         self.linear2 = nn.Linear(3,1)
13         self.sigmoid = nn.Sigmoid()
14     def forward(self, x):
15         return self.sigmoid(self.linear2(self.sigmoid(self.linear1(x))))
16
17 model_BinaryClassification = class_BinaryClassification()
18 optimizer = torch.optim.SGD(model_BinaryClassification.parameters(), lr=0.01)
19
20 for step in range(2000):
21     prediction = model_BinaryClassification(x_train)
22     cost = F.binary_cross_entropy(prediction, y_train)
23     optimizer.zero_grad()
24     cost.backward()
25     optimizer.step()
26
27 x_test = torch.FloatTensor([[6,1]])
28 model_test = model_BinaryClassification(x_test)
29 print("Model with [6,1] (expectation: 1) in sigmoid: ", model_test.detach().item())
30 model_test_binary = np.round(model_test>0.5).type(torch.float32)
31 print("Model with [6,1] (expectation: 1) in np.round: ", model_test_binary.detach().item())
```

- TensorFlow
- PyTorch
- **Keras**

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  

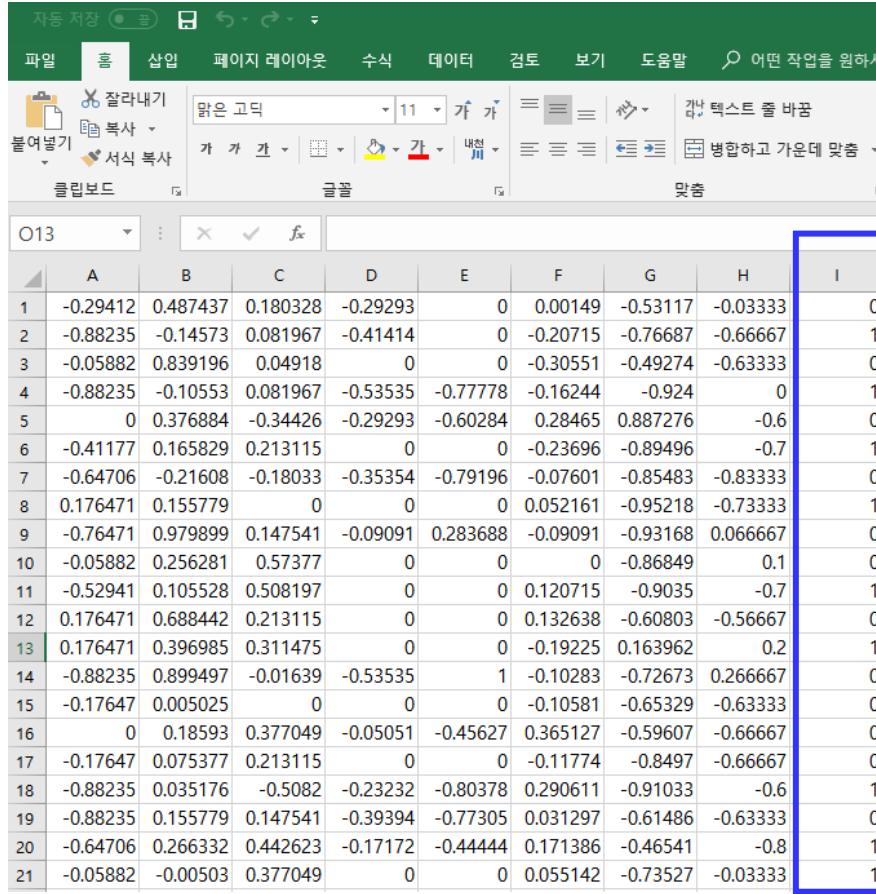
Non-trainable params: 0

-----  

[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)

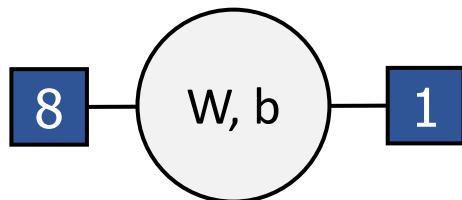
CSV file



O13	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

```
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

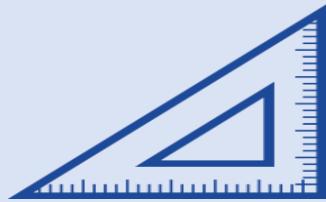


Binary Classification Implementation (Keras)

```
x - o 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

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 (Examples)

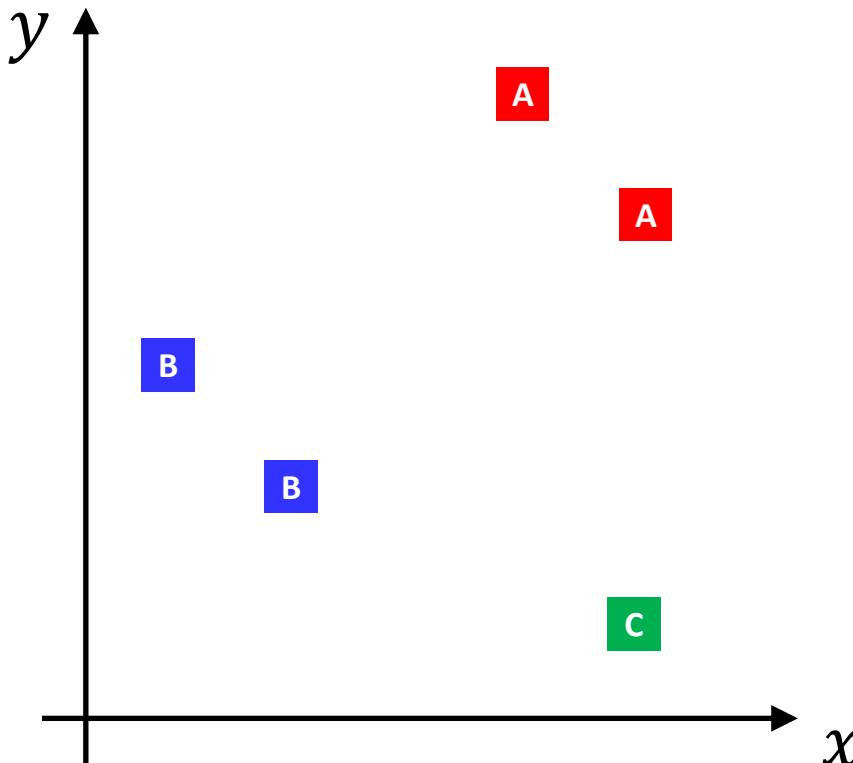
- Exam Score Prediction (Linear Regression)



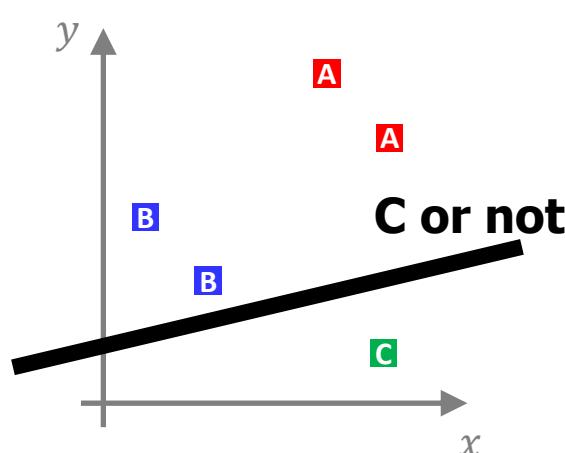
Classification (Examples)

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

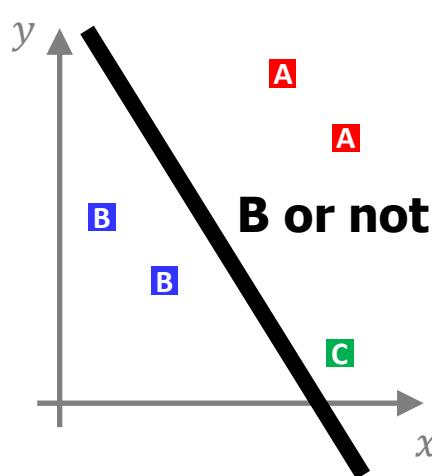




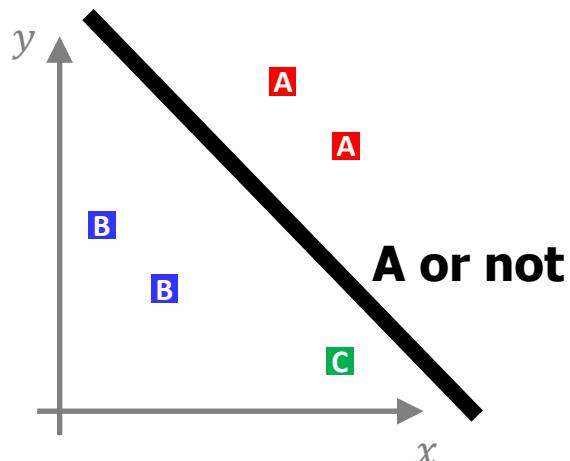
Multinomial Classification (Softmax Classification)



$$X \rightarrow W \rightarrow f(t) = \frac{1}{1+e^{-t}} \rightarrow \bar{Y}$$



$$X \rightarrow W \rightarrow f(t) = \frac{1}{1+e^{-t}} \rightarrow \bar{Y}$$

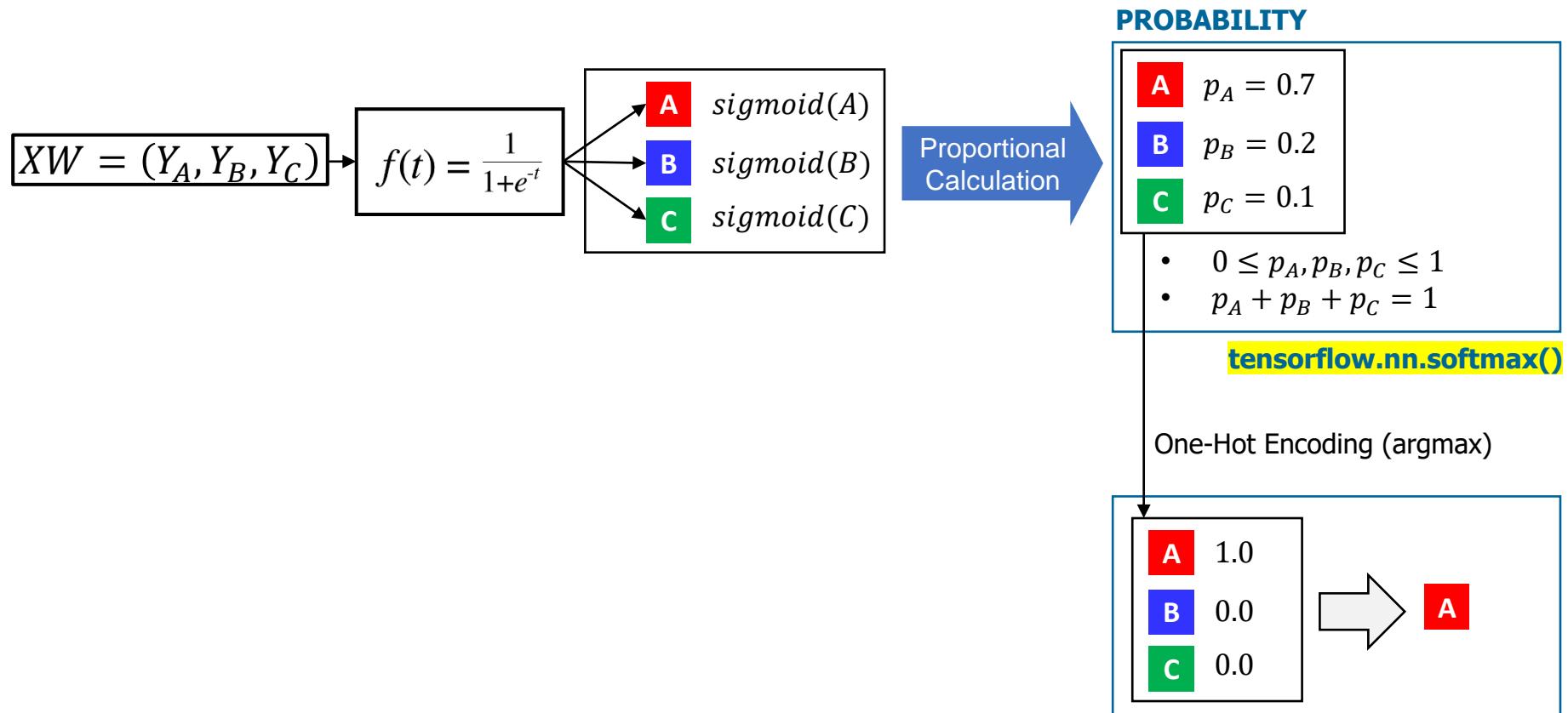


$$X \rightarrow W \rightarrow f(t) = \frac{1}{1+e^{-t}} \rightarrow \bar{Y}$$

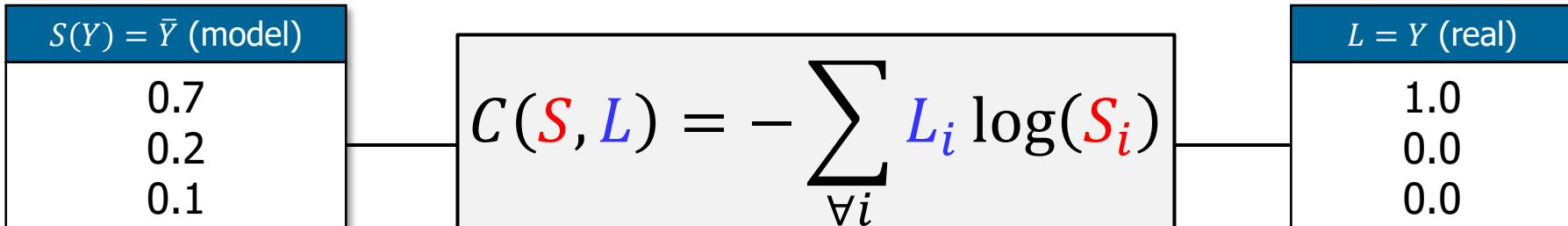
$$\begin{pmatrix} x_1 & x_2 \end{pmatrix} \cdot \begin{pmatrix} A \\ B \\ C \end{pmatrix} = \begin{pmatrix} x_1 \cdot W_{A1} + x_2 \cdot W_{A2} \\ x_1 \cdot W_{B1} + x_2 \cdot W_{B2} \\ x_1 \cdot W_{C1} + x_2 \cdot W_{C2} \end{pmatrix}$$

The matrix multiplication is shown with the input vector $(x_1 \ x_2)$ multiplied by a column vector containing weight matrices for classes A, B, and C. The resulting vector contains the weighted sum of inputs for each class.

Multinomial Classification (Softmax Classification)



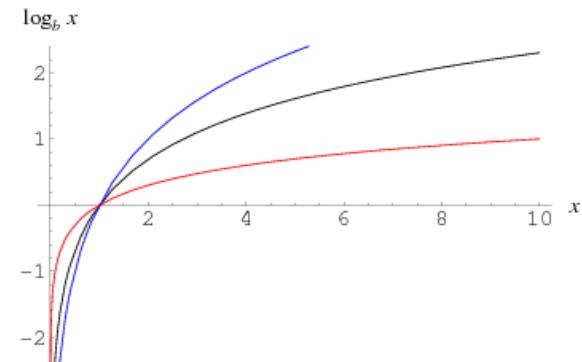
- Cost Function: **Cross-Entropy**



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





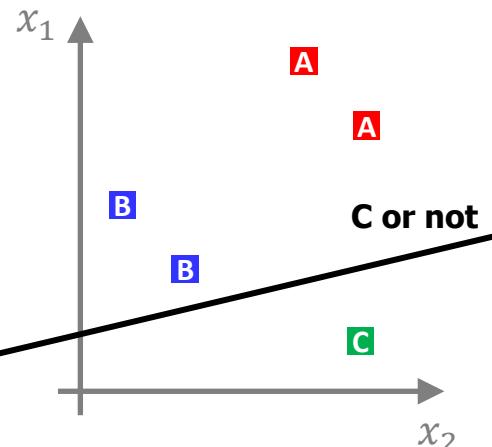
Deep Learning Basics and Software

Softmax Classification

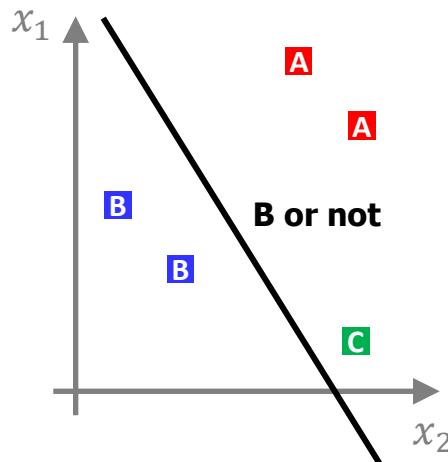
Softmax Classification Implementation

- Softmax Classification Theory
- Softmax Classification Implementation

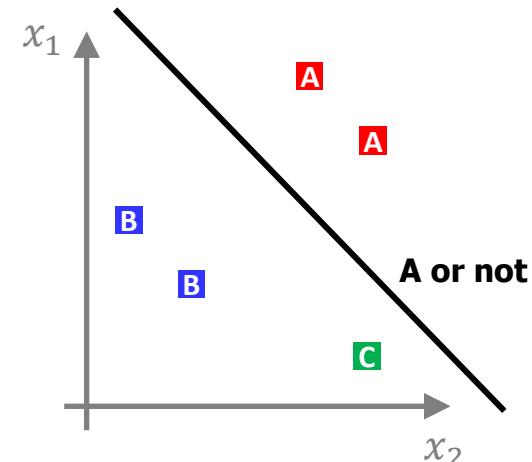
- Implementation Example



$$X \rightarrow W \rightarrow f(t) = \frac{1}{1+e^{-t}} \rightarrow \bar{Y}$$



$$X \rightarrow W \rightarrow f(t) = \frac{1}{1+e^{-t}} \rightarrow \bar{Y}$$



$$X \rightarrow W \rightarrow f(t) = \frac{1}{1+e^{-t}} \rightarrow \bar{Y}$$

- Implementation Example
 - Vector Representation

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

$$S_A \triangleq x_1 \cdot W_{A1} + x_2 \cdot W_{A2} + x_3 \cdot W_{A3} + x_4 \cdot W_{A4}$$

$$S_B \triangleq x_1 \cdot W_{B1} + x_2 \cdot W_{B2} + x_3 \cdot W_{B3} + x_4 \cdot W_{B4}$$

$$S_C \triangleq x_1 \cdot W_{C1} + x_2 \cdot W_{C2} + x_3 \cdot W_{C3} + x_4 \cdot W_{C4}$$

- **TensorFlow**
- PyTorch
- Keras

```
1 import tensorflow as tf
2 import numpy as np
3 x_train = 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]], dtype=np.float32)
4 y_train = 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]], dtype=np.float32)
5 W = tf.Variable(tf.random.normal([4, 3]))
6 b = tf.Variable(tf.random.normal([3]))
7
8 def model_SoftmaxClassificationLC(x):
9     return tf.matmul(x,W)+b
10
11 def cost_SoftmaxClassification(model_x):
12     return tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=model_x, labels=y_train))
13
14 def train_optimization(x):
15     with tf.GradientTape() as g:
16         model_LC = model_SoftmaxClassificationLC(x)
17         cost = cost_SoftmaxClassification(model_LC)
18         gradients = g.gradient(cost,[W,b])
19         tf.optimizers.SGD(0.01).apply_gradients(zip(gradients,[W,b]))
20
21 for step in range(2001):
22     train_optimization(x_train)
23     if step % 100 == 0:
24         pred = model_SoftmaxClassificationLC(x_train)
25         loss = cost_SoftmaxClassification(pred)
26         correct = tf.equal(tf.argmax(pred,1), tf.argmax(y_train,1))
27         accuracy = tf.reduce_mean(tf.cast(correct,tf.float32))
28         print("Step: {},\t Accurarray: {},\t Loss: {}".format(step,accuracy.numpy().flatten(),loss))
29
30 print('*'*100)
31 x_test = np.array([[1,8,8,8]],dtype=np.float32)
32 model_test = tf.nn.softmax(model_SoftmaxClassificationLC(x_test))
33 print("model for [1,8,8,8]: ", tf.argmax(model_test,1).numpy())
34 print('*'*100)
```

- TensorFlow
- **PyTorch**
- Keras

Softmax Classification Implementation (PyTorch)



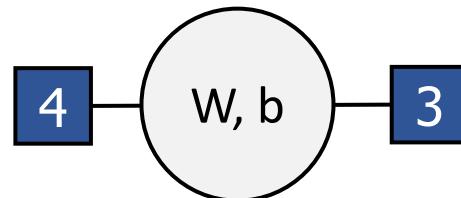
```
1 import torch
2 import torch.nn as nn
3 import torch.nn.functional as F
4 import numpy as np
5 x_train = torch.FloatTensor([[1,2,1,1], [2,1,3,2], [3,1,3,4], [4,1,5,5], [1,7,5,5], [1,2,5,6], [1,6,6,6], [1,7,7,7]])
6 y_train = torch.FloatTensor([[0,0,1], [0,0,1], [0,0,1], [0,1,0], [0,1,0], [0,1,0], [1,0,0], [1,0,0]])
7 index_y = torch.argmax(y_train,1)
8 W = torch.randn([4,3], requires_grad=True)
9 b = torch.randn([3], requires_grad=True)
10 optimizer = torch.optim.SGD([W, b], lr=0.01)
11
12 def model_SoftmaxClassification(x):
13     return F.softmax(torch.matmul(x, W) + b)
14
15 for step in range(50000):
16     prediction = model_SoftmaxClassification(x_train)
17     cost = F.cross_entropy(prediction, index_y)
18     optimizer.zero_grad()
19     cost.backward()
20     optimizer.step()
21
22 x_test = torch.FloatTensor([[1,8,8,8]])
23 model_test = model_SoftmaxClassification(x_test)
24 index = torch.argmax(model_test.detach(),1).item()
25 if index == 0:
26     print("Model with [1,8,8,8] is A.")
27 elif index == 1:
28     print("Model with [1,8,8,8] is B.")
29 elif index == 2:
30     print("Model with [1,8,8,8] is C.")
```

Softmax Classification Implementation (PyTorch)

```
1 import torch
2 import torch.nn as nn
3 import torch.nn.functional as F
4 import numpy as np
5 x_train = torch.FloatTensor([[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]])
6 y_train = torch.FloatTensor([[0,0,1], [0,0,1], [0,0,1], [0,1,0], [0,1,0], [0,1,0], [1,0,0], [1,0,0]])
7 index_y = torch.argmax(y_train,1)
8
9 class class_SoftmaxClassification(nn.Module):
10     def __init__(self):
11         super().__init__()
12         self.linear = nn.Linear(4,3)
13     def forward(self, x):
14         return F.softmax(self.linear(x))
15
16 model_SoftmaxClassification = class_SoftmaxClassification()
17 optimizer = torch.optim.SGD(model_SoftmaxClassification.parameters(), lr=0.01)
18
19 for step in range(50000):
20     prediction = model_SoftmaxClassification(x_train)
21     cost = F.cross_entropy(prediction, index_y)
22     optimizer.zero_grad()
23     cost.backward()
24     optimizer.step()
25
26 x_test = torch.FloatTensor([[1,8,8,8]])
27 model_test = model_SoftmaxClassification(x_test)
28 index = torch.argmax(model_test.detach(),1)
29 if index == 0:
30     print("Model with [1,8,8,8] is A.")
31 elif index == 1:
32     print("Model with [1,8,8,8] is B.")
33 elif index == 2:
34     print("Model with [1,8,8,8] is C.")
```

- TensorFlow
- PyTorch
- **Keras**

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

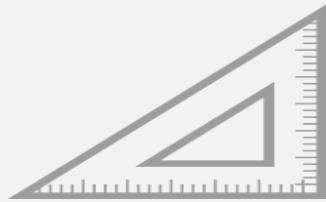
Layer (type)          Output Shape         Param #
=====
dense_1 (Dense)      (None, 3)            15
=====
Total params: 15
Trainable params: 15
Non-trainable params: 0
-----
[[1.3161632e-01 8.6831880e-01 6.4874497e-05]]
argmax: 1
```

- 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

Linear Functions



Linear Regression
Binary Classification
Softmax Classification

Nonlinear Functions



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

Advanced Topics



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

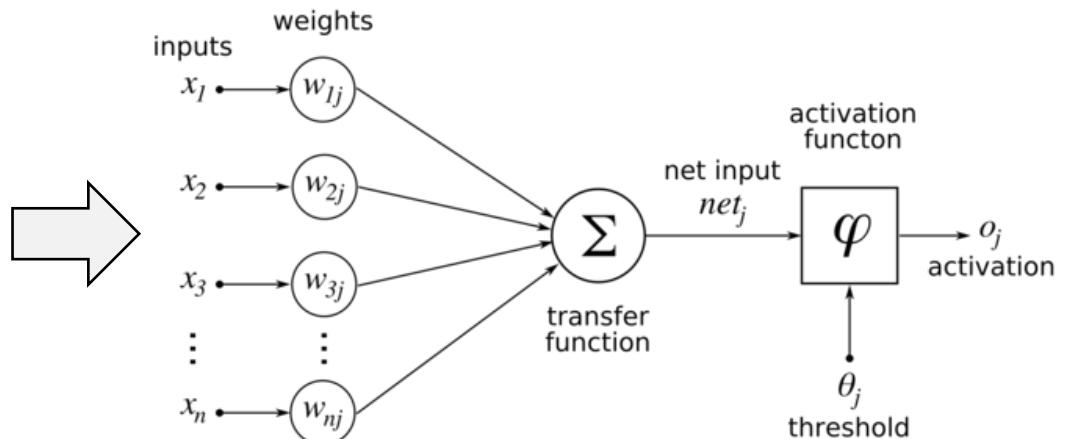
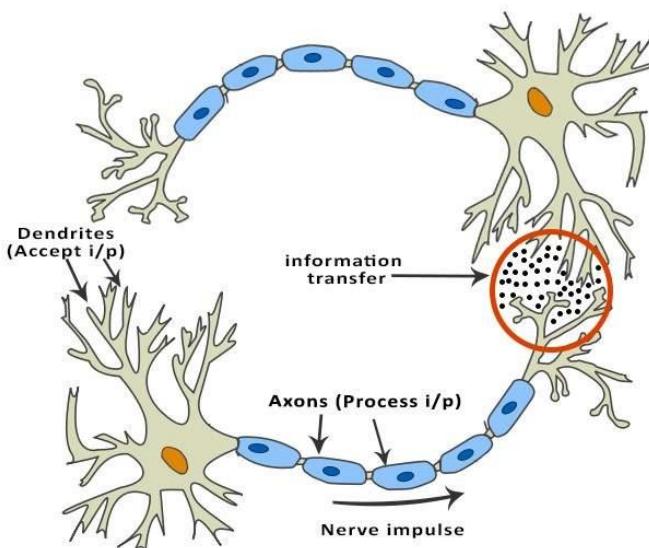
Deep Learning Basics and Software

Artificial Neural Networks (ANN)

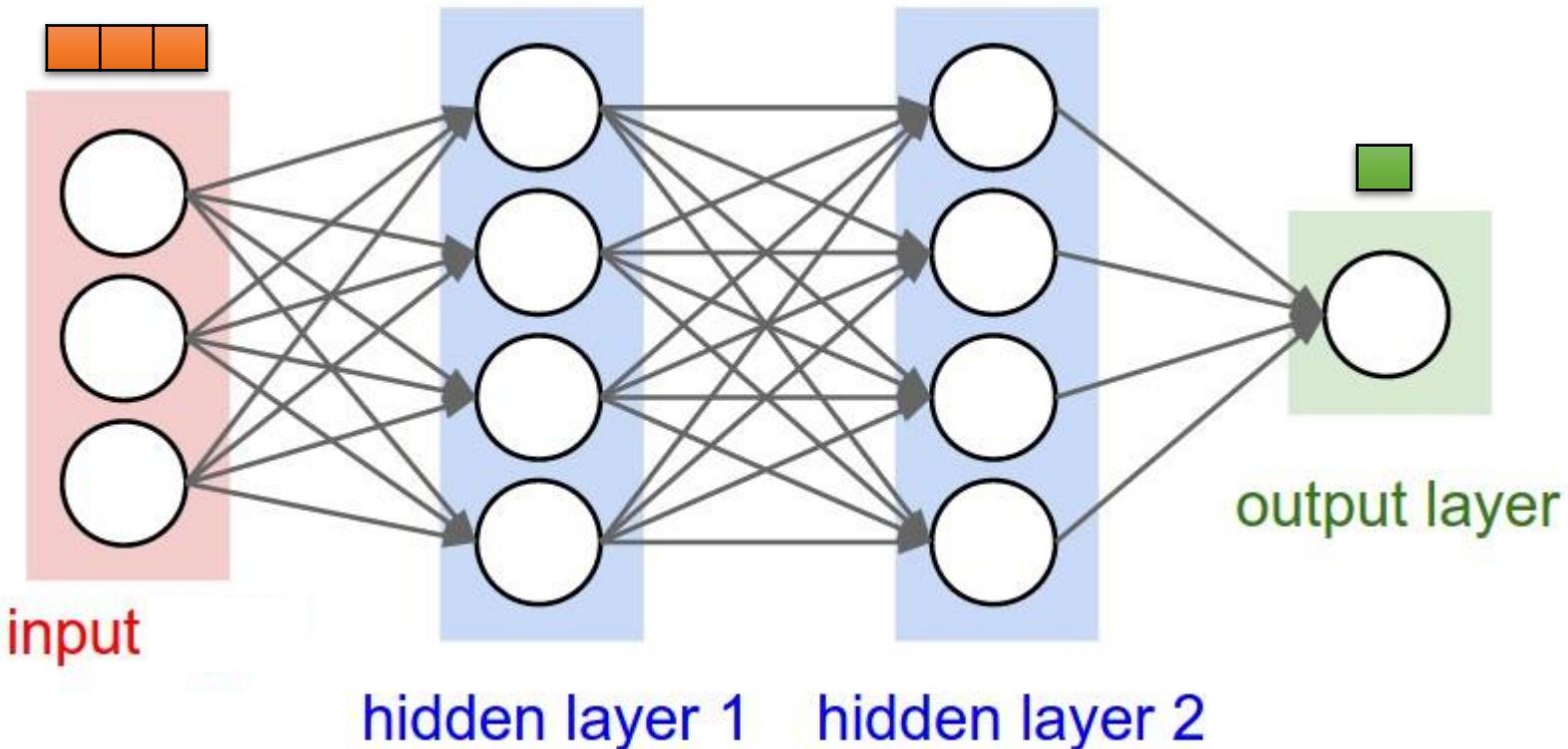
ANN Theory

- **ANN Theory**
- ANN Implementation

- Human Brain (Neuron)

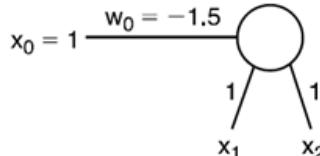


Binary Classification

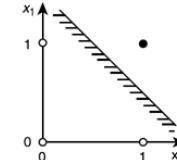
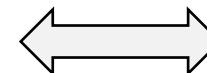


- Application to Logic Gate Design

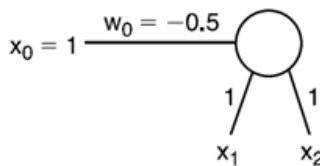
AND gate



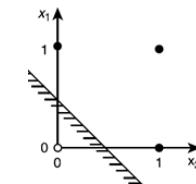
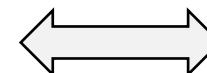
x_1	x_2	$W \cdot X$	y
0	0	-1.5	0
0	1	-0.5	0
1	0	-0.5	0
1	1	0.5	1



OR gate

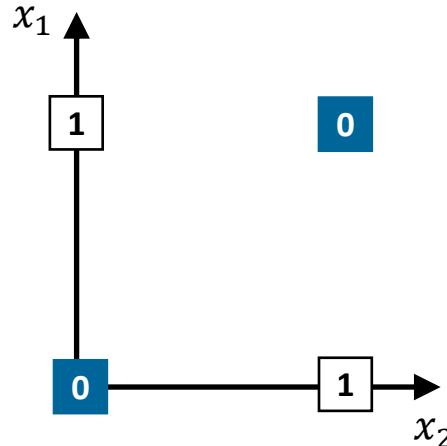


x_1	x_2	$W \cdot X$	y
0	0	-0.5	0
0	1	0.5	1
1	0	0.5	1
1	1	1.5	1



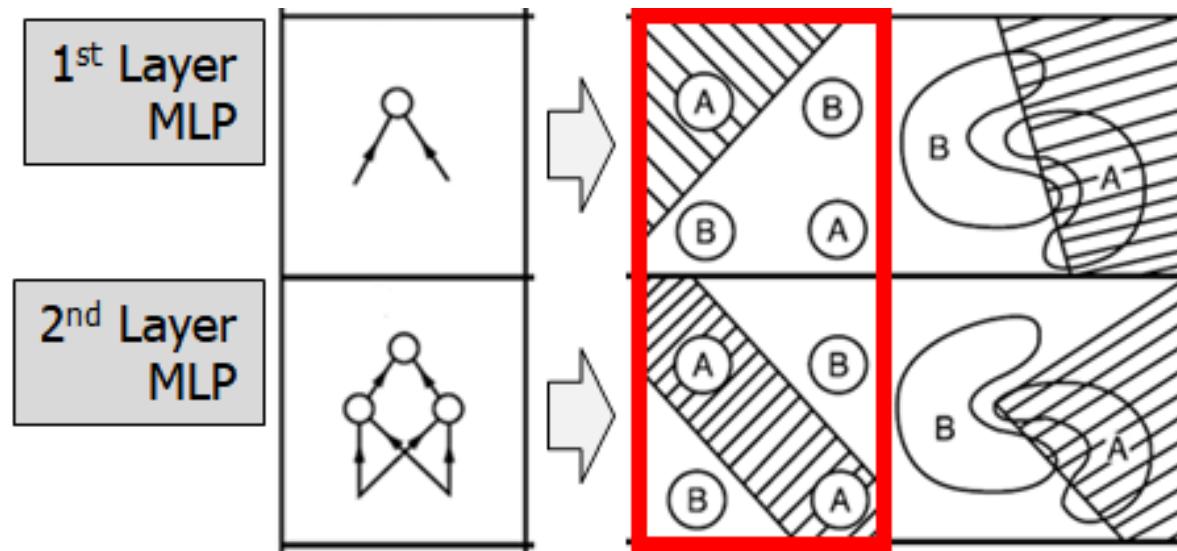
- What about XOR?

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

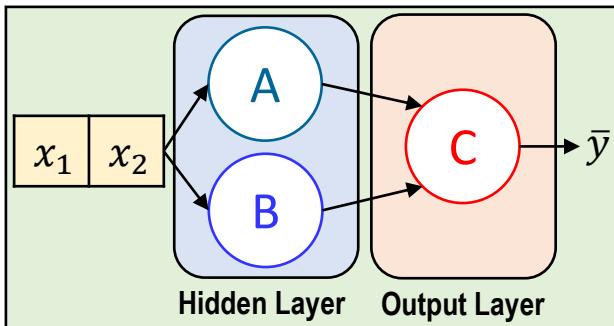
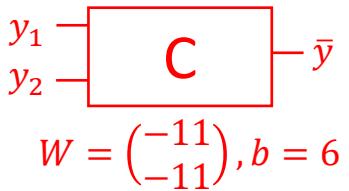
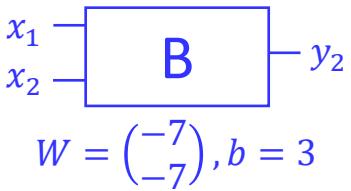
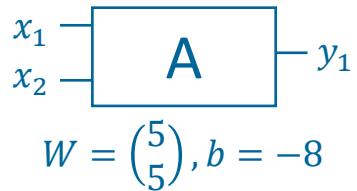


Mathematically proven by
Prof. Marvin Minsky at MIT (1969)

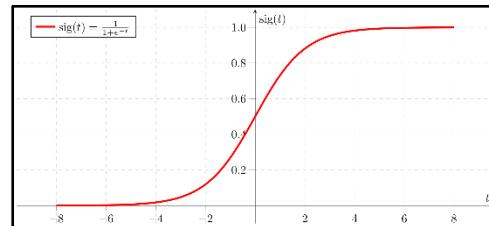
- Multilayer Perceptron (MLP)
 - Proposed by Prof. Marvin Minsky at MIT (1969)
 - Can solve XOR Problem



ANN: Solving XOR with MLP

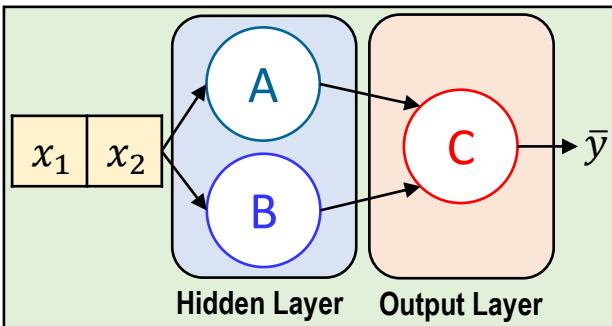
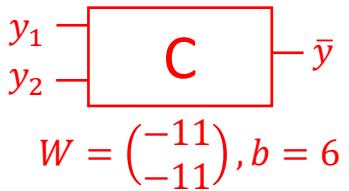
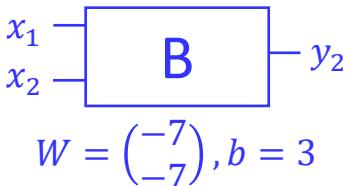
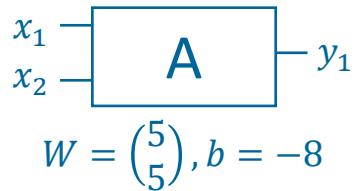


- $(x_1 \ x_2) = (0 \ 0)$
 - $(0 \ 0) \begin{pmatrix} 5 \\ 5 \end{pmatrix} + (-8) = -8$, i.e., $y_1 = \text{Sigmoid}(-8) \cong 0$
 - $(0 \ 0) \begin{pmatrix} -7 \\ 7 \end{pmatrix} + (3) = 3$, i.e., $y_2 = \text{Sigmoid}(3) \cong 1$
 - $(y_1 \ y_2) \begin{pmatrix} -11 \\ 11 \end{pmatrix} + (6) = -11 +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

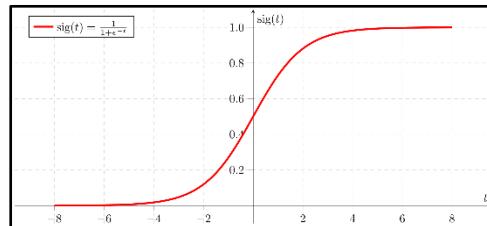


x_1	x_2	y_1	y_2	\bar{y}	XOR
0	0	0	1	0	0
0	1				1
1	0				1
1	1				0

ANN: Solving XOR with MLP

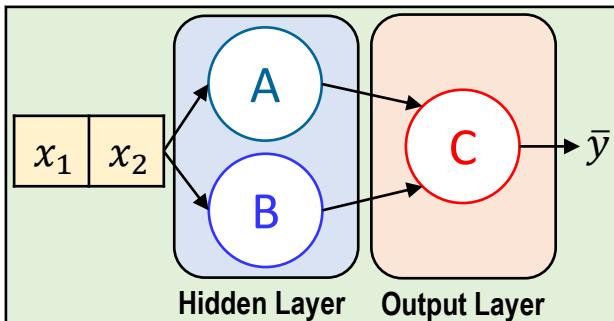
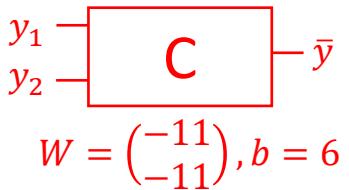
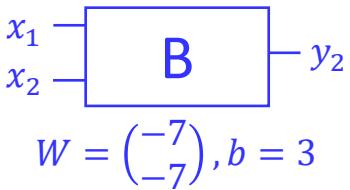
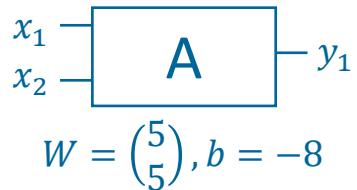


- $(x_1 \ x_2) = (0 \ 1)$
 - $(0 \ 1) \begin{pmatrix} 5 \\ 5 \end{pmatrix} + (-8) = -3$, i.e., $y_1 = Sigmoid(-3) \cong 0$
 - $(0 \ 1) \begin{pmatrix} -7 \\ -7 \end{pmatrix} + (3) = -4$, i.e., $y_2 = Sigmoid(-4) \cong 0$
 - $(y_1 \ y_2) \begin{pmatrix} -11 \\ -11 \end{pmatrix} + (6) = 6$, i.e., $\bar{y}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

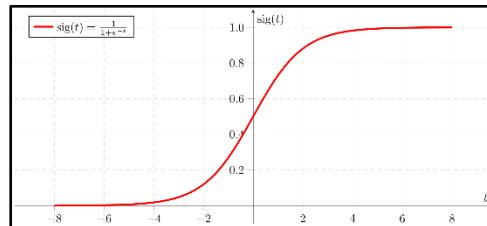


x₁	x₂	y₁	y₂	\bar{y}	XOR
0	0	0	1	0	0
0	1	0	0	1	1
1	0				1
1	1				0

ANN: Solving XOR with MLP

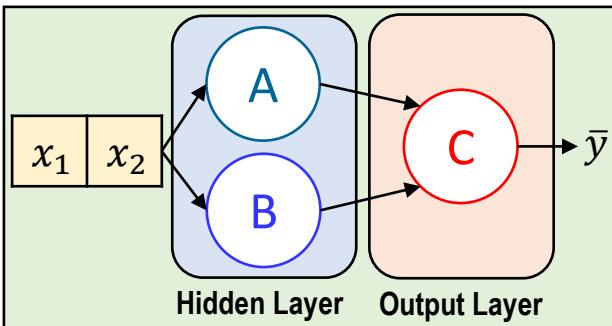
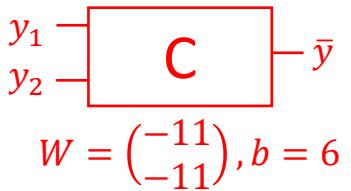
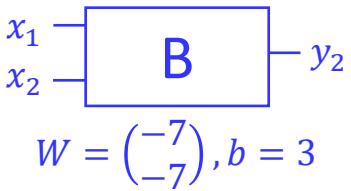
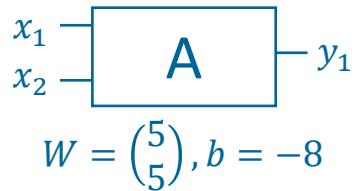


- $(x_1 \ x_2) = (1 \ 0)$
 - $(1 \ 0) \begin{pmatrix} 5 \\ 5 \end{pmatrix} + (-8) = -3$, i.e., $y_1 = \text{Sigmoid}(-3) \cong 0$
 - $(1 \ 0) \begin{pmatrix} -7 \\ -7 \end{pmatrix} + (3) = -4$, i.e., $y_2 = \text{Sigmoid}(-4) \cong 0$
 - $(y_1 \ y_2) \begin{pmatrix} -11 \\ -11 \end{pmatrix} + (6) = 6$,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

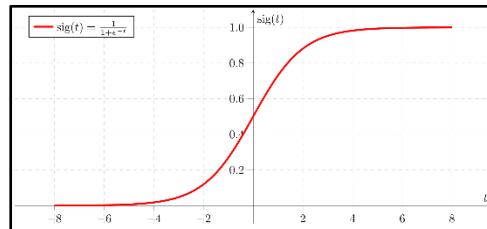


x₁	x₂	y₁	y₂	ȳ	XOR
0	0	0	1	0	0
0	1	0	0	1	1
1	0	0	0	1	1
1	1				0

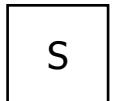
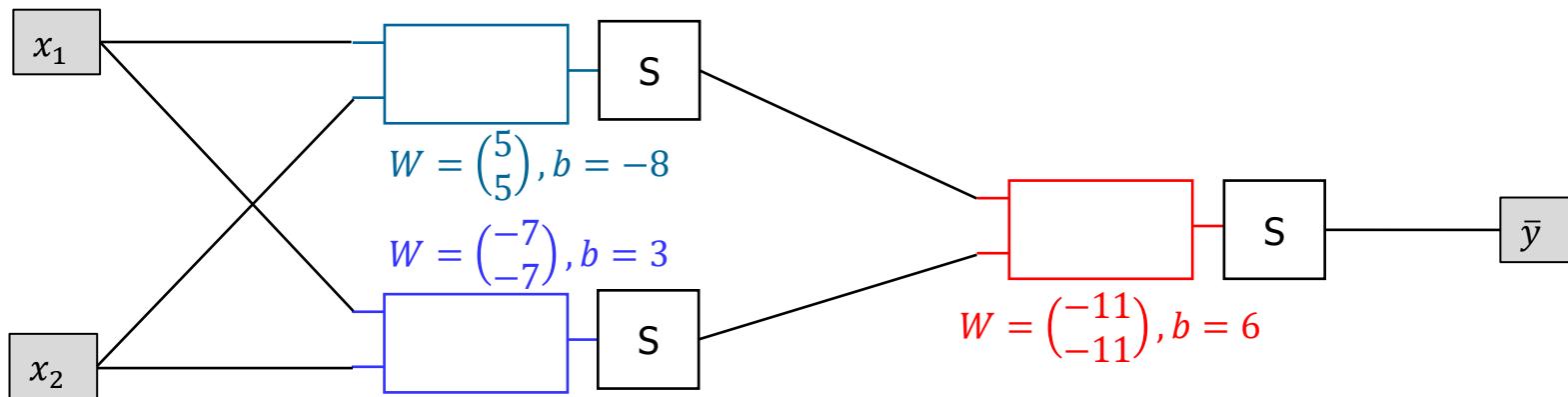
ANN: Solving XOR with MLP

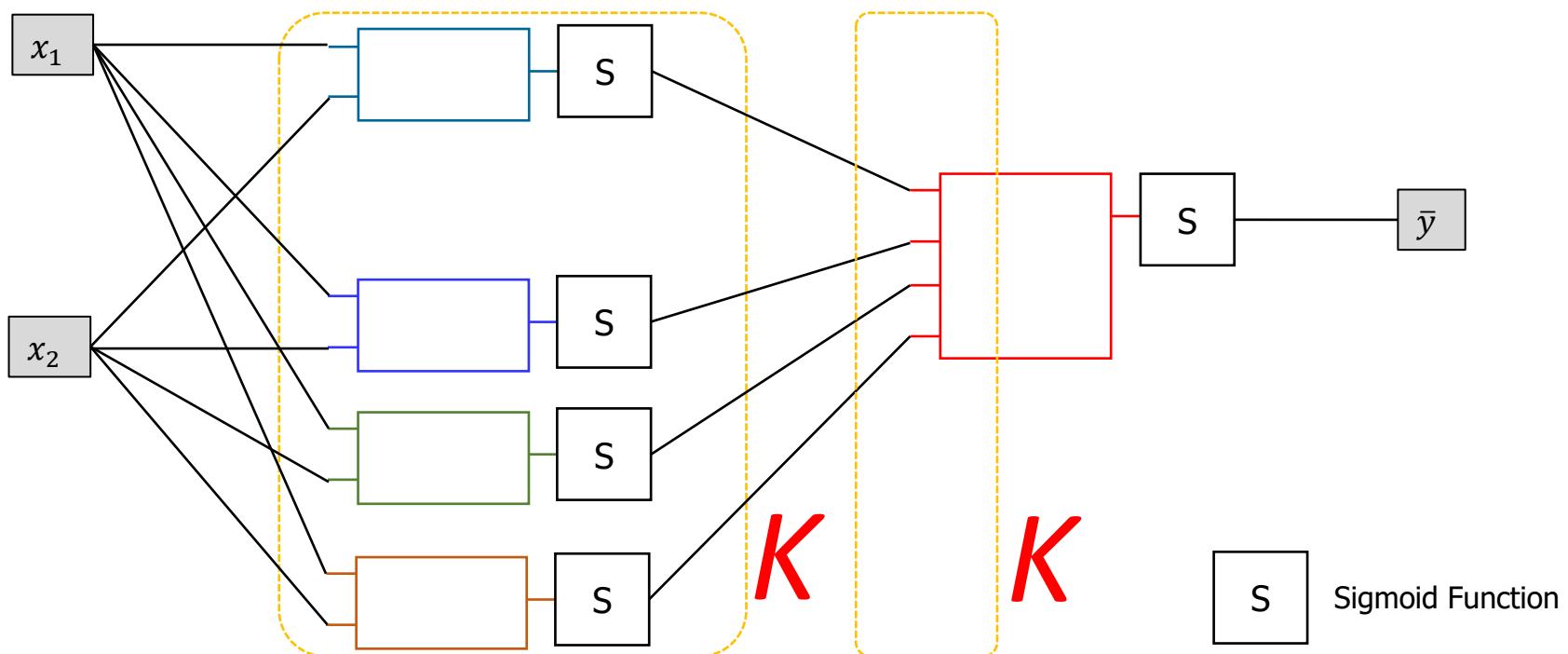


- $(x_1 \ x_2) = (1 \ 1)$
 - $(1 \ 1) \begin{pmatrix} 5 \\ 5 \end{pmatrix} + \begin{pmatrix} -8 \end{pmatrix} = 2$, i.e., $y_1 = Sigmoid(2) \cong 1$
 - $(1 \ 1) \begin{pmatrix} -7 \\ 7 \end{pmatrix} + \begin{pmatrix} 3 \end{pmatrix} = -11$, i.e., $y_2 = Sigmoid(-11) \cong 0$
 - $(y_1 \ y_2) \begin{pmatrix}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

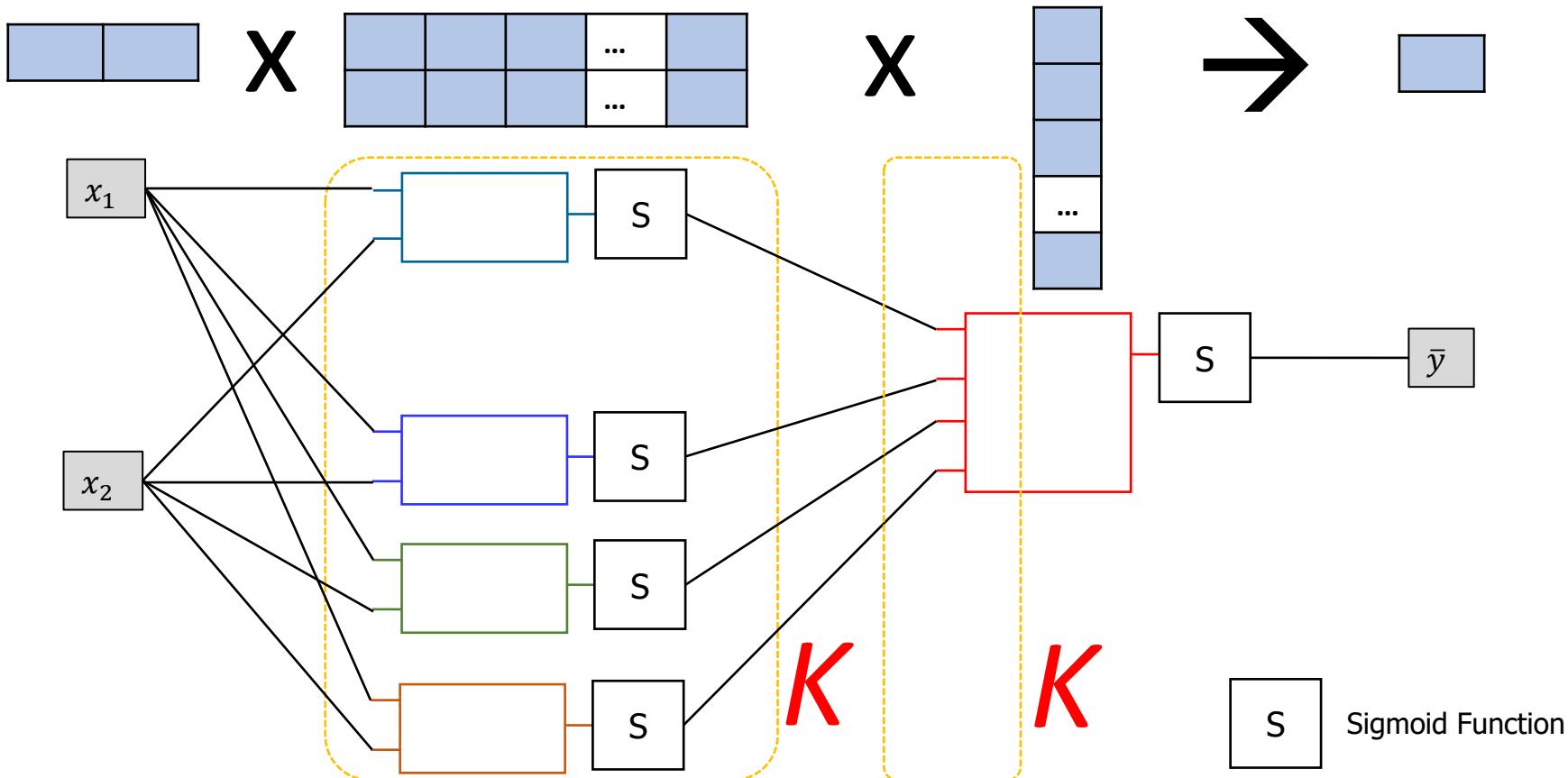


x_1	x_2	y_1	y_2	\bar{y}	XOR
0	0	0	1	0	0
0	1	0	0	1	1
1	0	0	0	1	1
1	1	1	0	0	0

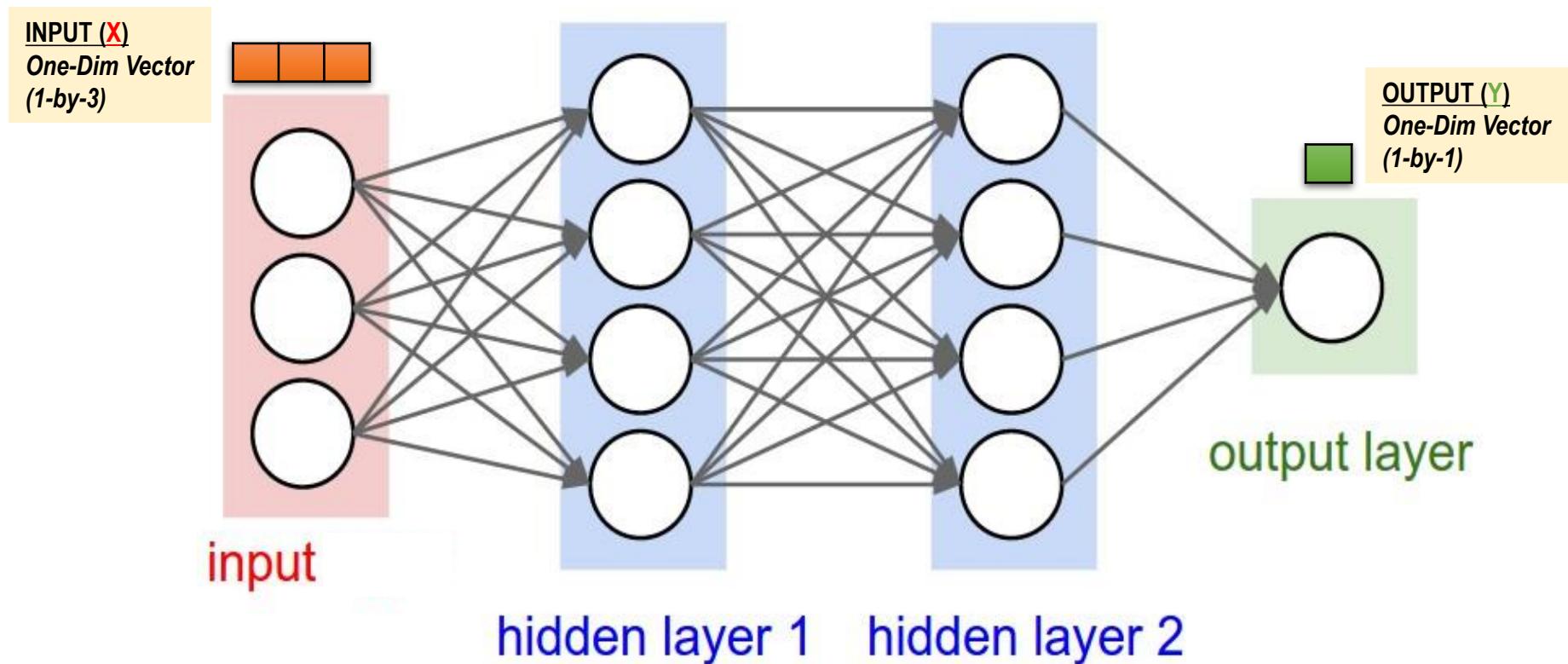
 S Sigmoid Function



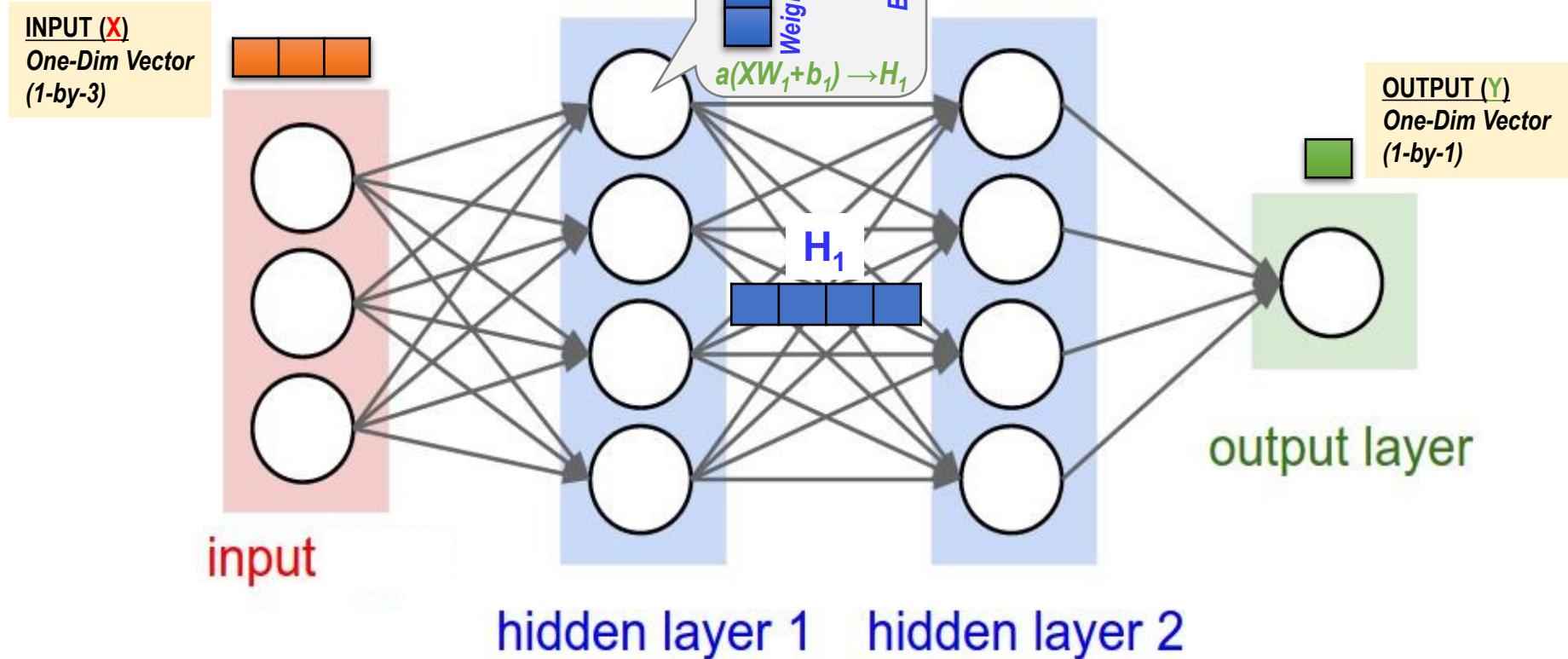
ANN: Solving XOR with MLP (Forward Propagation)



- Toy Model



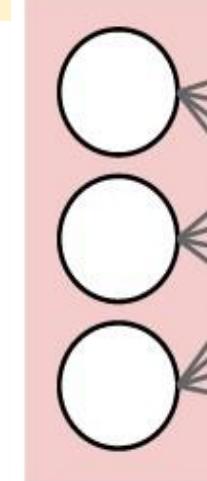
- Toy Model



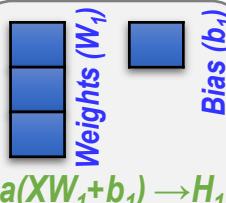
- Toy Model

INPUT (X)

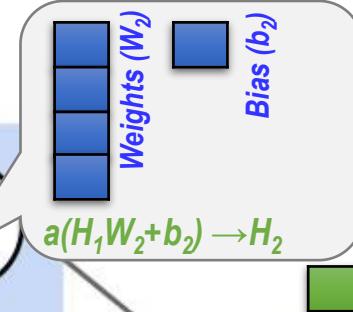
One-Dim Vector
(1-by-3)



input



$$a(XW_1 + b_1) \rightarrow H_1$$



$$a(H_1W_2 + b_2) \rightarrow H_2$$

OUTPUT (Y)
One-Dim Vector
(1-by-1)

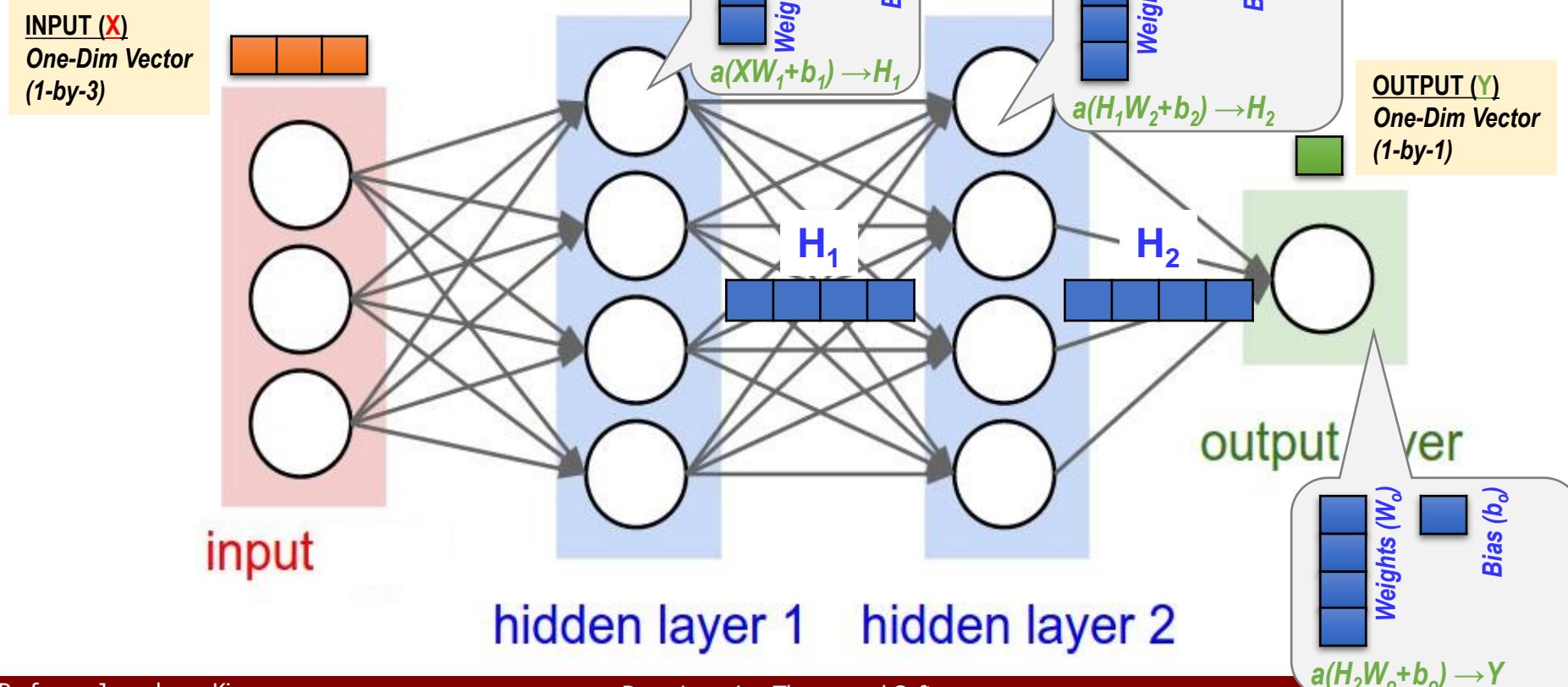


output layer

hidden layer 1 hidden layer 2

Conventional Deep Neural Network Training and Inference

- Toy Model



```
1 #TensorFlow for ANN (XOR with ANN)
2 import tensorflow as tf
3 import numpy as np
4
5 x_train = np.array([[0,0], [0,1], [1,0], [1,1]], dtype=np.float32)
6 y_train = np.array([[0], [1], [1], [0]], dtype=np.float32)
7 W_h = tf.Variable(tf.random.normal([2,3]))
8 b_h = tf.Variable(tf.random.normal([3]))
9 W_o = tf.Variable(tf.random.normal([3,1]))
10 b_o = tf.Variable(tf.random.normal([1]))
11 train_step = 10001
12
13 def model_ANN_XOR(x):
14     H1 = tf.sigmoid(tf.matmul(x,W_h)+b_h)
15     return tf.sigmoid(tf.matmul(H1,W_o)+b_o)
16
17 def cost_ANN_XOR(model_x):
18     return tf.reduce_mean((-1)*y_train*tf.math.log(model_x)+(-1)*(1.-y_train)*tf.math.log(1.-model_x))
19
20 def train_optimization(x):
21     with tf.GradientTape() as g:
22         model = model_ANN_XOR(x)
23         cost = cost_ANN_XOR(model)
24     gradients = g.gradient(cost,[W_h,W_o,b_h,b_o])
25     tf.optimizers.SGD(0.1).apply_gradients(zip(gradients,[W_h,W_o,b_h,b_o]))
26
27 for step in range(train_step):
28     train_optimization(x_train)
29
30 print('*'*150)
31 x_test = np.array([[1,0],[0,1],[1,1],[0,0]], dtype = np.float32)
32 y_test = np.array([[1],[1],[0],[0]], dtype = np.float32)
33 model_test = model_ANN_XOR(x_test)
34 test_prediction = tf.cast(model_test > 0.5, dtype=tf.float32)
35 test_accuracy = tf.reduce_mean(tf.cast(tf.equal(test_prediction, y_test), dtype=tf.float32))
36 print("Test_Accuracy: ", test_accuracy.numpy())
37 print('*'*150)
```

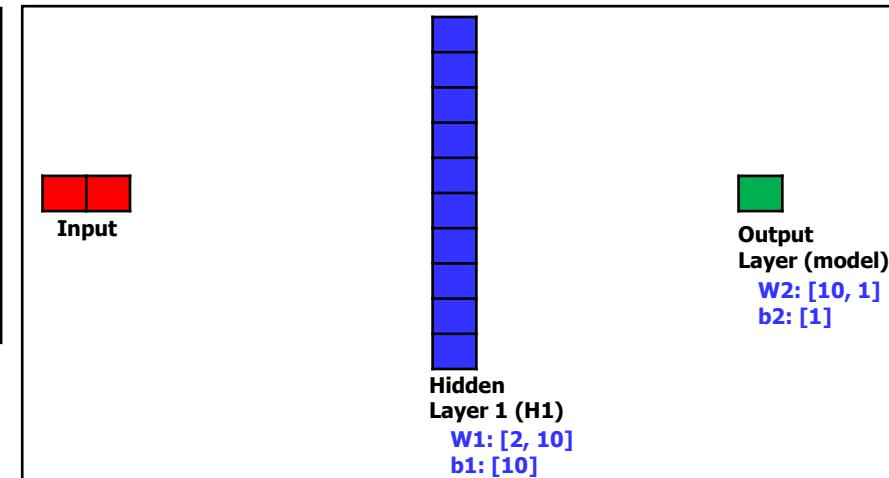
```
1 import torch
2 import numpy as np
3 x_train = torch.FloatTensor([[0,0], [0,1], [1,0], [1,1]])
4 y_train = torch.FloatTensor([[0], [1], [1], [0]])
5 W_h = torch.randn([2,3], requires_grad=True)
6 b_h = torch.randn([3], requires_grad=True)
7 W_o = torch.randn([3,1], requires_grad=True)
8 b_o = torch.randn([1], requires_grad=True)
9 optimizer = torch.optim.SGD([W_h, b_h, W_o, b_o], lr=0.01)
10
11 def model_ANN(x):
12     HL1 = torch.sigmoid(torch.matmul(x, W_h) + b_h)
13     Out = torch.sigmoid(torch.matmul(HL1, W_o) + b_o)
14     return Out
15
16 for step in range(200000):
17     prediction = model_ANN(x_train)
18     cost = torch.mean((-1)*y_train*torch.log(prediction)+(-1)*(1-y_train)*torch.log(1-prediction))
19     optimizer.zero_grad()
20     cost.backward()
21     optimizer.step()
22
23 model_test = model_ANN(x_train)
24 print(model test.detach().numpy())
```

- **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)
    
```



- Deep ANN for XOR

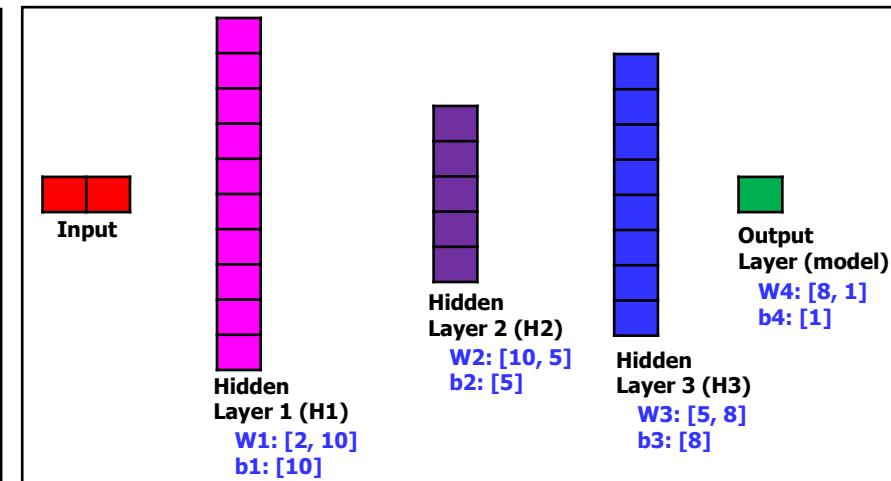
```

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

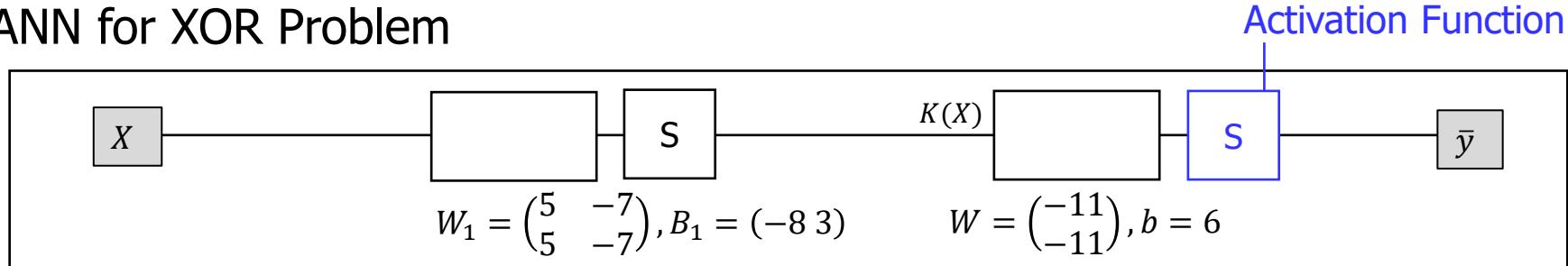
W2 = tf.Variable(tf.random_normal([10, 5]))
b2 = tf.Variable(tf.random_normal([5]))
H2 = tf.sigmoid(tf.matmul(H1, W2) + b2)

W3 = tf.Variable(tf.random_normal([5, 8]))
b3 = tf.Variable(tf.random_normal([8]))
H3 = tf.sigmoid(tf.matmul(H2, W3) + b3)

W4 = tf.Variable(tf.random_normal([8, 1]))
b4 = tf.Variable(tf.random_normal([1]))
model = tf.sigmoid(tf.matmul(H3, W4) + b4)
  
```



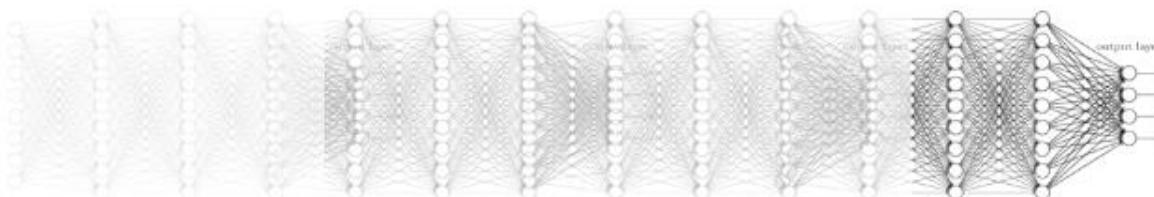
• ANN for XOR Problem



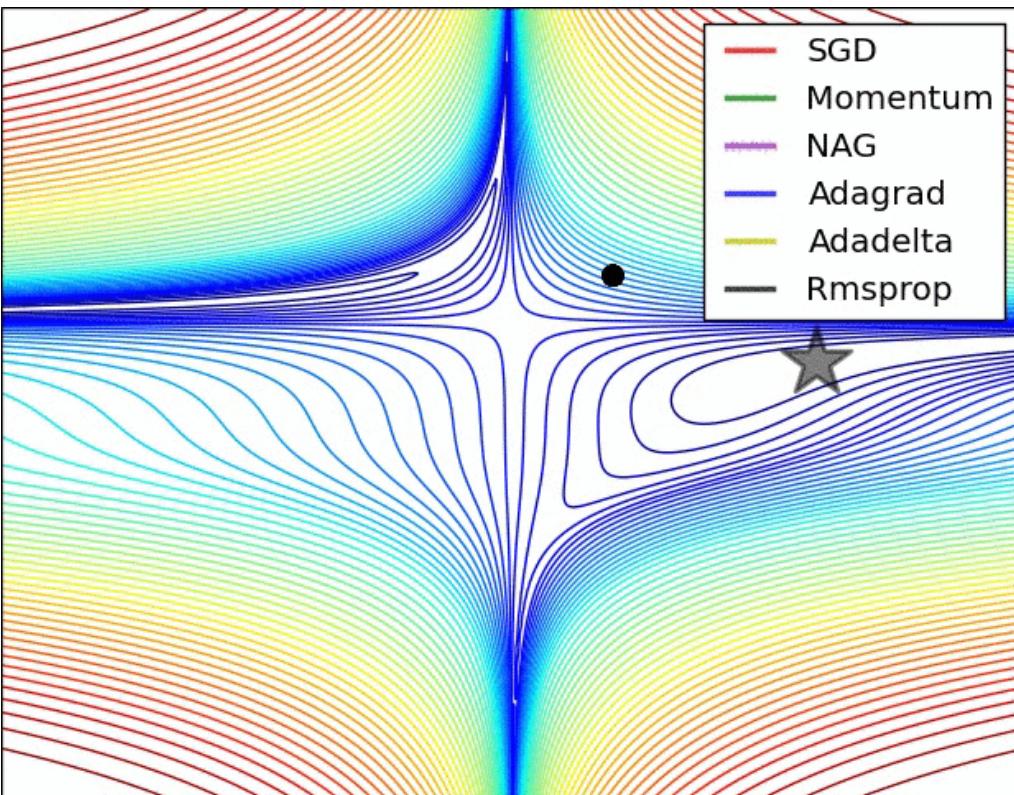
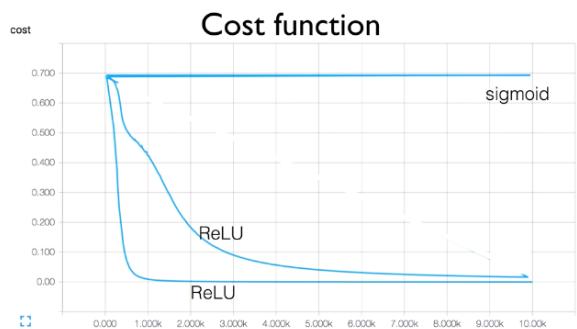
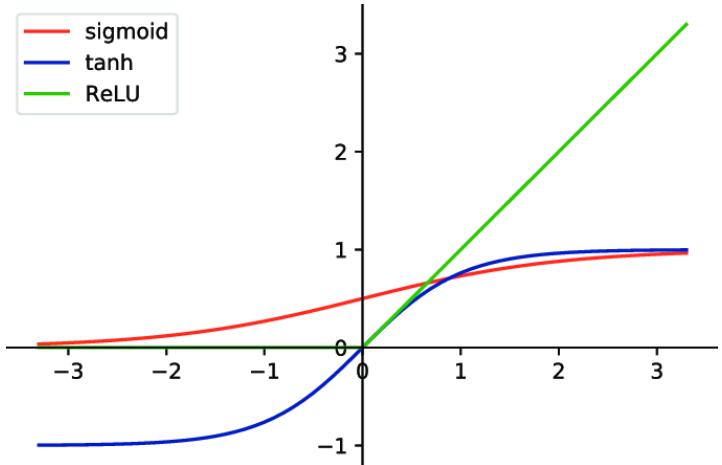
• Observation)

- 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)



- Deep Learning Revolution is Real

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

IMAGENET
14.2 million images



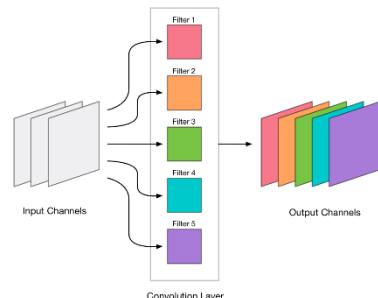
Big-Data

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



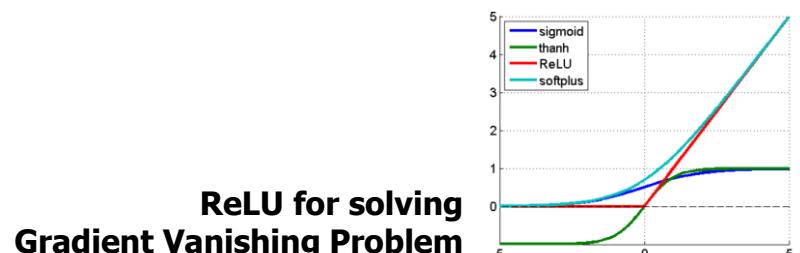
GPU

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).



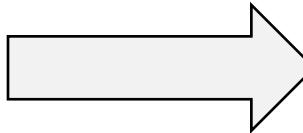
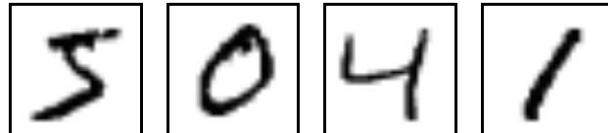
Deep Learning Theory and Software

Artificial Neural Networks (ANN)

ANN Implementation

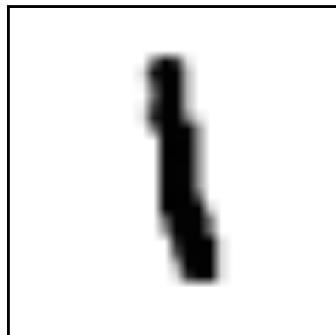
- ANN Theory
- **ANN Implementation**

- MNIST Data Set
 - Hand-written images and their labels
 - For training: 55,000 (or 60,000)
 - For testing: 10,000



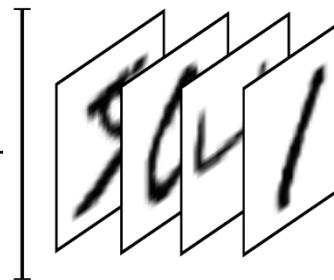
[0,0,0,0,0,1,0,0,0,0]	→ '5'
[1,0,0,0,0,0,0,0,0,0]	→ '0'
[0,0,0,0,1,0,0,0,0,0]	→ '4'
[0,1,0,0,0,0,0,0,0,0]	→ '1'

- Image: 28-by-28 (pixels)



$$\approx \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

784



60000

mnist.train.xs

- **TensorFlow**
- PyTorch
- Keras

```
1 import tensorflow as tf
2 import numpy as np
3 import torch.nn as nn
4 from tensorflow.keras.datasets import mnist
5
6 batch_size = 128
7 nH1 = 256
8 nH2 = 256
9 nH3 = 256
10
11 (x_train, y_train), (x_test, y_test) = mnist.load_data()
12 x_train, x_test = x_train.astype('float32'), x_test.astype('float32')
13 x_train, x_test = x_train.reshape([-1, 784]), x_test.reshape([-1, 784])
14 x_train, x_test = x_train / 255., x_test / 255.
15 y_train, y_test = tf.one_hot(y_train, depth=10), tf.one_hot(y_test, depth=10)
16 train_data = tf.data.Dataset.from_tensor_slices((x_train, y_train))
17 train_data = train_data.repeat().shuffle(5000).batch(batch_size)
18
19 class ANN(nn.Module):
20     def __init__(self):
21         self.W_1 = tf.Variable(tf.random.normal(shape=[784, nH1]))
22         self.W_2 = tf.Variable(tf.random.normal(shape=[nH1, nH2]))
23         self.W_3 = tf.Variable(tf.random.normal(shape=[nH2, nH3]))
24         self.W_Out = tf.Variable(tf.random.normal(shape=[nH3, 10]))
25         self.b_1 = tf.Variable(tf.random.normal(shape=[nH1]))
26         self.b_2 = tf.Variable(tf.random.normal(shape=[nH2]))
27         self.b_3 = tf.Variable(tf.random.normal(shape=[nH3]))
28         self.b_Out = tf.Variable(tf.random.normal(shape=[10]))
29
30     def __call__(self, x):
31         # pytorch에서는 __forward__ tensor에서는 __call__ 사용
32         H1_Out = tf.nn.relu(tf.matmul(x, self.W_1) + self.b_1)
33         H2_Out = tf.nn.relu(tf.matmul(H1_Out, self.W_2) + self.b_2)
34         H3_Out = tf.nn.relu(tf.matmul(H2_Out, self.W_3) + self.b_3)
35         Out = tf.matmul(H2_Out, self.W_Out) + self.b_Out
36
37         return Out
```

```
38 ANN_model = ANN()
39 optimizer = tf.optimizers.Adam(0.01)
40
41 def cost_ANN_mnist(x,y):
42     y = tf.cast(y, tf.int64)
43     loss = tf.nn.softmax_cross_entropy_with_logits(logits=x,labels=y)
44     return tf.reduce_mean(loss)
45
46 def accuracy(x,y):
47     correct = tf.equal(tf.argmax(x,1), tf.argmax(y,1))
48     accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
49     return accuracy
50
51 def train_optimization(x,y):
52     with tf.GradientTape() as g:
53         pred = ANN_model(x)
54         cost = cost_ANN_mnist(pred,y)
55         gradients = g.gradient(cost, vars(ANN_model).values(), unconnected_gradients=tf.UnconnectedGradients.ZERO)
56         #기존의 gradients와 다르게 뒤에 unconnected_gradients를 정의함으로써 연결되지 않는 그래디언트로 인한 오류를 없앤다.
57         optimizer.apply_gradients(zip(gradients, vars(ANN_model).values()))
58
59 for step, (batch_x, batch_y) in enumerate(train_data.take(300), 1):
60     train_optimization(batch_x, batch_y)
61     if step % 10 == 0:
62         pred = ANN_model(batch_x)
63         loss = cost_ANN_mnist(pred,batch_y)
64         acc = accuracy(pred, batch_y)
65         print("Step: {},\t Accurarray: {},\t Loss: {}".format(step,acc.numpy().flatten(),loss.numpy().flatten()))
66     print('*'*30)
67 print('Model test')
68 print("Test_Accurarray: ",accuracy(ANN_model(x_test),y_test).numpy().flatten())
69 print('*'*30)
```

- TensorFlow
- **PyTorch**
- Keras

```
1 import torch
2 import torch.nn as nn
3 import torch.nn.functional as F
4 from torch.utils.data import DataLoader
5 from torchvision import datasets, transforms
6 import numpy as np
7 import pandas as pd
8 from sklearn.metrics import confusion_matrix
9 from torch.autograd import Variable
10
11 train_data = datasets.MNIST(root='./Data', train=True, download=True, transform=transforms.ToTensor())
12 test_data = datasets.MNIST(root='./Data', train=False, download=True, transform=transforms.ToTensor())
13 train_loader = DataLoader(train_data, batch_size=128, shuffle=True)
14 test_loader = DataLoader(test_data, batch_size=128, shuffle=False)
15
16 class ANNModel(nn.Module):
17     def __init__(self, input_dim, hidden_dim, output_dim):
18         super(ANNModel, self).__init__()
19         self.net = nn.Sequential(nn.Linear(input_dim, hidden_dim),
20                                nn.ReLU(),
21                                nn.Linear(hidden_dim, hidden_dim),
22                                nn.ReLU(),
23                                nn.Linear(hidden_dim, hidden_dim),
24                                nn.ReLU(),
25                                nn.Linear(hidden_dim, output_dim))
26
27     def forward(self, x):
28         return self.net(x)
```

```
30 model = ANNModel(input_dim=784, hidden_dim=256, output_dim=10)
31 cost = nn.CrossEntropyLoss()
32 optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
33 step = 0
34 loss_list = []
35 accuracy_list = []
36
37 for epoch in range(5):
38     for i, (images, labels) in enumerate(train_loader):
39         train = Variable(images.view(-1, 784))
40         labels = Variable(labels)
41         optimizer.zero_grad()
42         outputs = model(train)
43         loss = cost(outputs, labels)
44         loss.backward()
45         optimizer.step()
46         step += 1
47         if step % 50 == 0:
48             correct = 0
49             total = 0
50             for images, labels in test_loader:
51                 test = Variable(images.view(-1, 784))
52                 outputs = model(test)
53                 predicted = torch.max(outputs.data, 1)[1]
54                 total += len(labels)
55                 correct += (predicted == labels).sum()
56             accuracy = correct.item() / total
57             loss_list.append(loss.data.item())
58             accuracy_list.append(accuracy)
59             if step % 100 == 0:
60                 print("Step: {},\t Accurarray: {},\t Loss: {}".format(step, accuracy, loss))
```

- TensorFlow
- PyTorch
- 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)

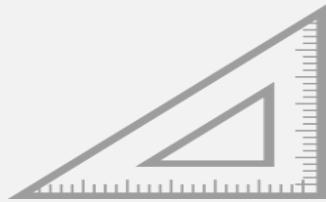


```
(60000, 28, 28) (60000,) (10000, 28, 28) (10000,)

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

Advanced Topics



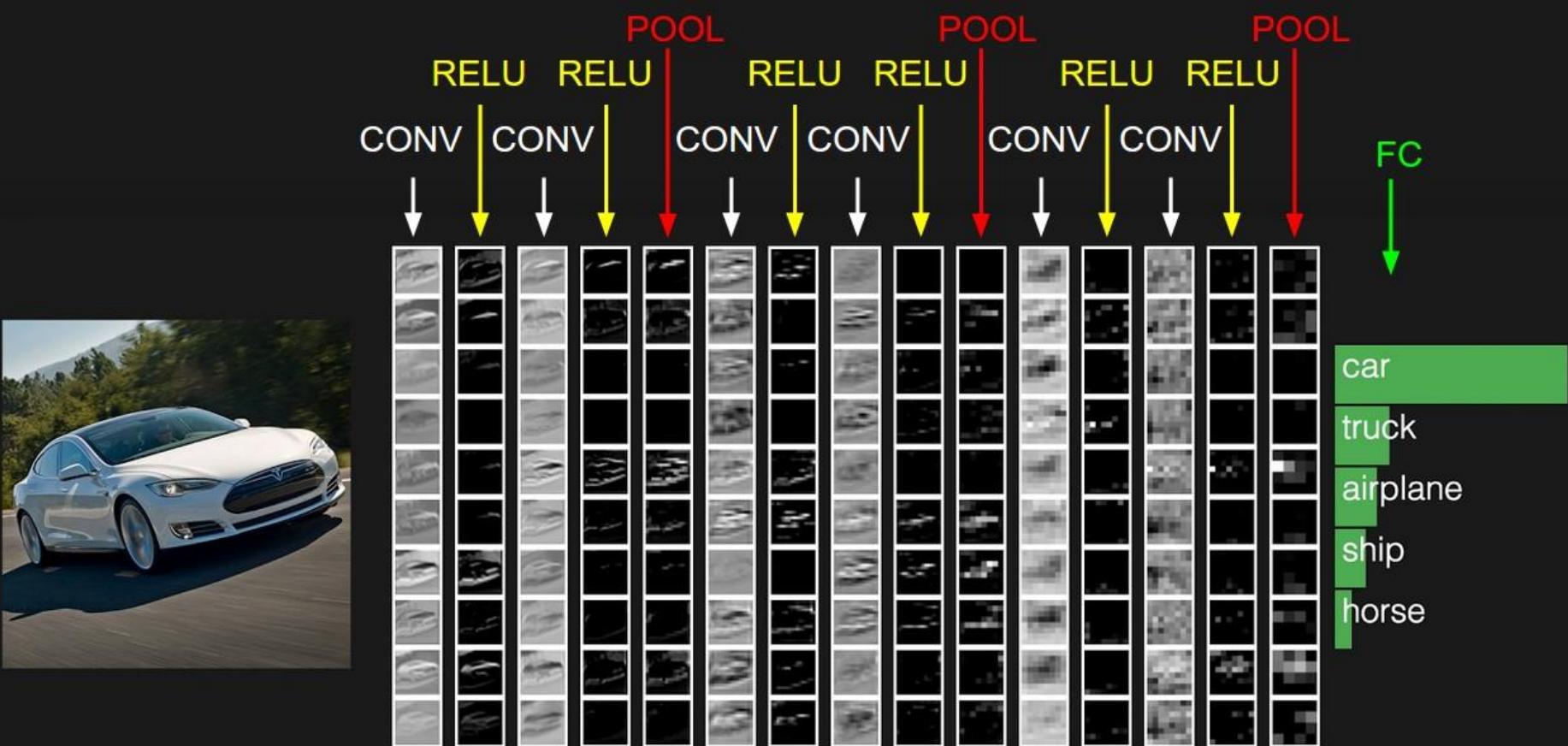
Gen. Adv. Network (GAN)
Interpolation
PCA/LDA
Overfitting

Deep Learning Theory and Software

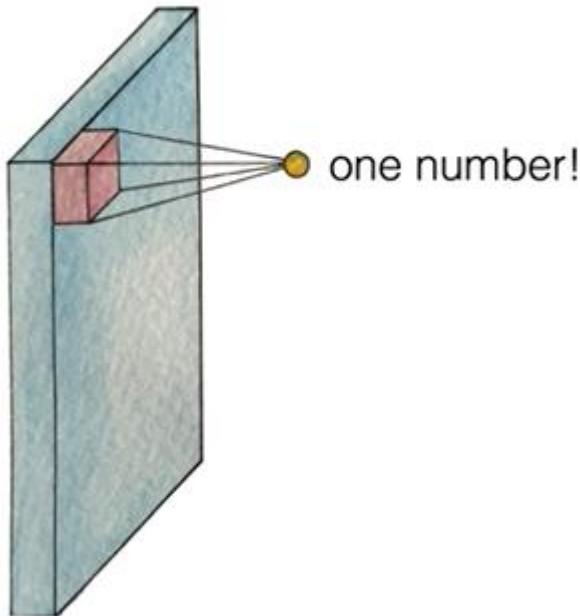
Convolutional Neural Networks (CNN)

CNN Theory

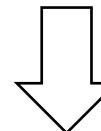
- **CNN Theory**
- CNN Implementation



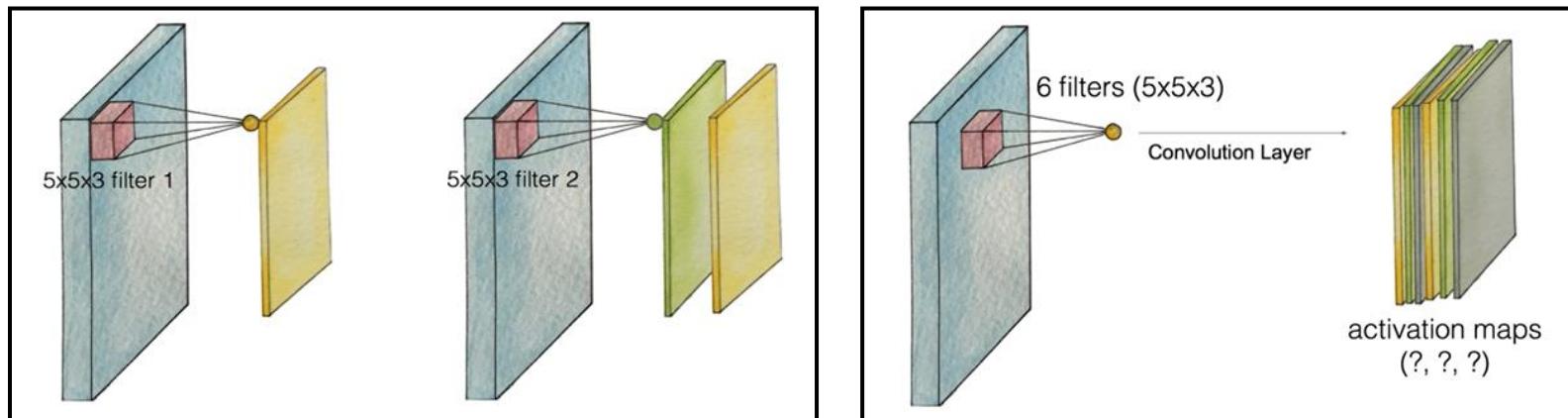
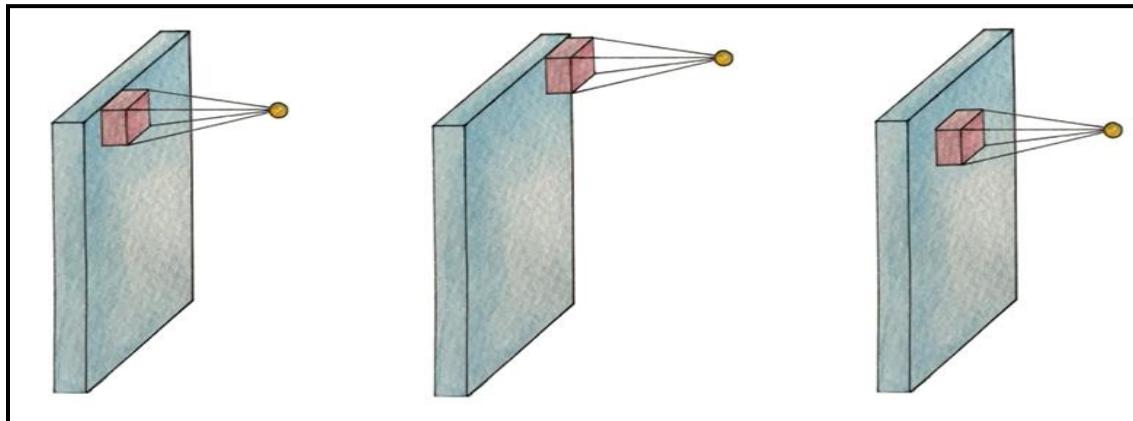
- Extracting **one point** with a filter

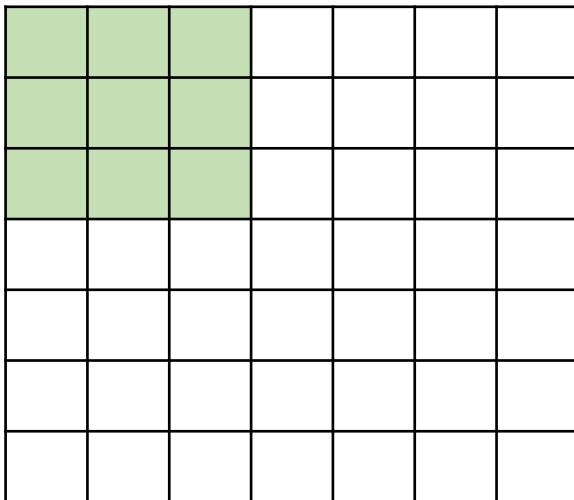


$$Wx + b$$



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





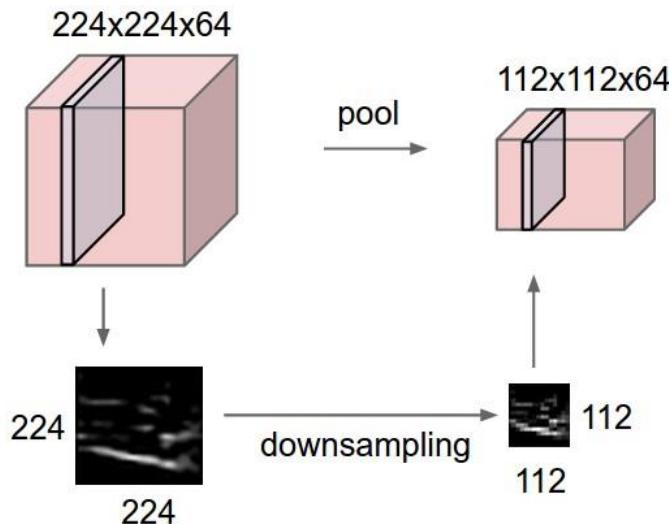
- Image Size: 7-by-7
- Filter Size: 3-by-3
- Strides (step-size of filter moving)
 - 1: $5 \times 5 = 25$
 - 2: $3 \times 3 = 9$
 - ...
 - → Large strides loose information (value degrades).

- Padding

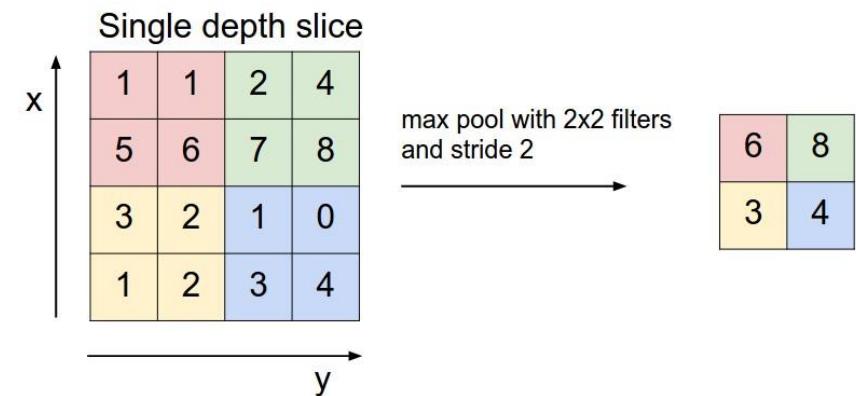
- Marking for the image areas
- Preventing the case where the sizes of return images becomes smaller.

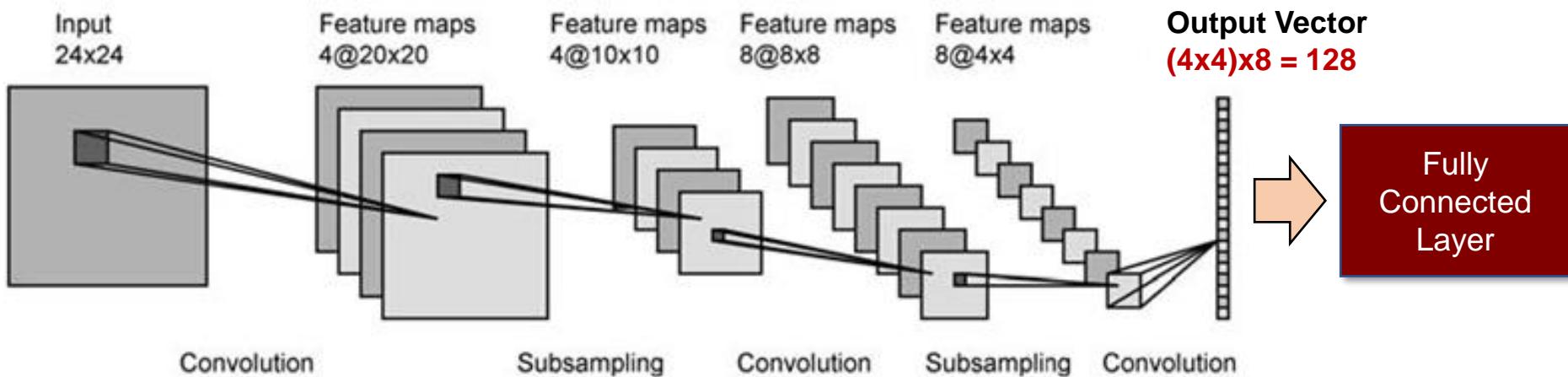
0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

- Pooling Layer (Sampling)

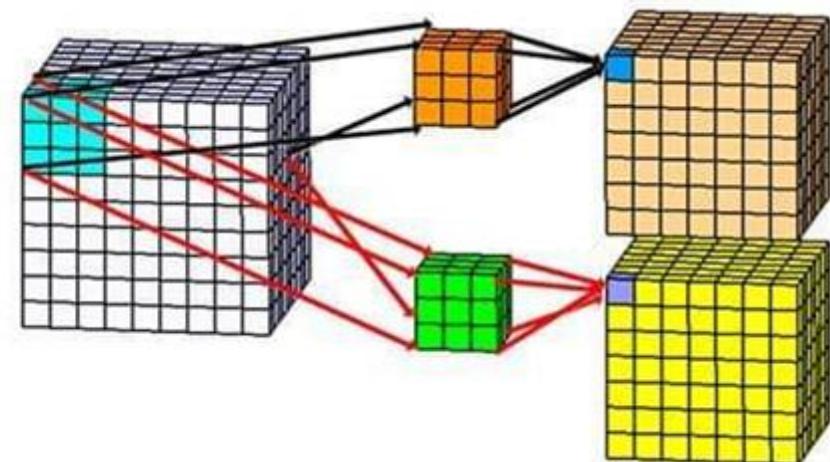
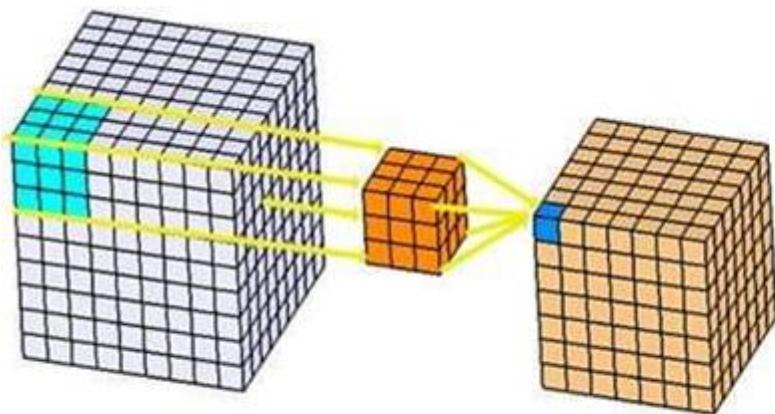


- Max Pooling Concept





- 3D Convolution



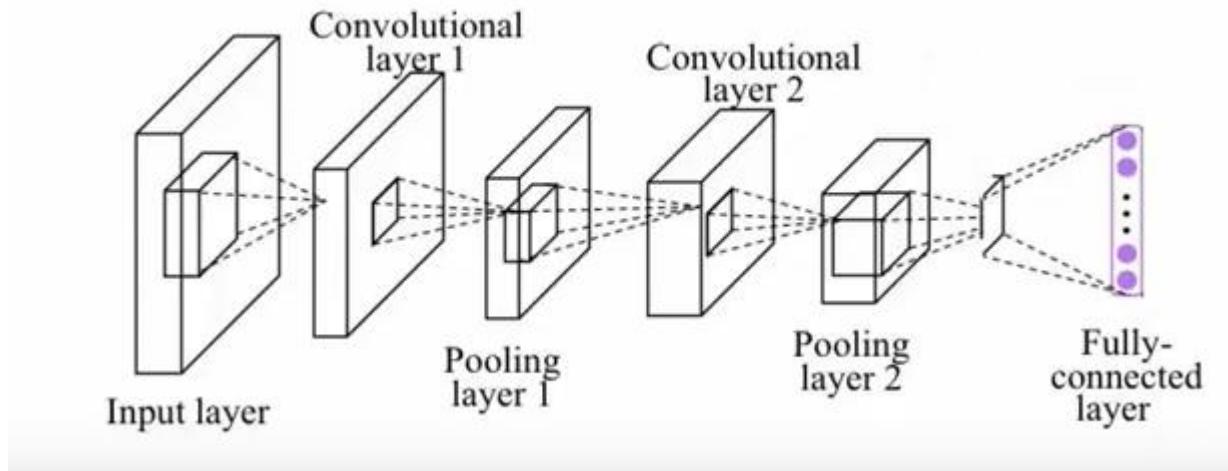
Deep Learning Theory and Software

Convolutional Neural Networks (CNN)

CNN Implementation

- CNN Theory
- CNN Implementation

- CNN for MNIST Image Classification



- **TensorFlow**
- Keras

```
1 #CNN Implementation (TensorFlow)
2 import tensorflow as tf
3 import numpy as np
4 import time
5 from tensorflow.keras.datasets import mnist
6 from tensorflow.keras import Model, layers
7
8 batch_size = 128
9 conv1_filters = 32
10 conv2_filters = 64
11 fc1_units = 1024
12
13 (x_train, y_train), (x_test, y_test) = mnist.load_data()
14 x_train, x_test = np.array(x_train, np.float32), np.array(x_test, np.float32)
15 x_train, x_test = x_train / 255., x_test / 255.
16 train_data = tf.data.Dataset.from_tensor_slices((x_train, y_train))
17 train_data = train_data.repeat().shuffle(5000).batch(batch_size)
```

```
19 class CNN(Model):
20     def __init__(self):
21         super(CNN, self).__init__()
22         self.conv1 = layers.Conv2D(32, kernel_size=3, activation=tf.nn.relu) # (필터 사이즈 3-by-3으로 32개)
23         self.maxpool1 = layers.MaxPool2D(2, strides=2)
24         self.conv2 = layers.Conv2D(64, kernel_size=3, activation=tf.nn.relu) # (필터 사이즈 3-by-3으로 64개)
25         self.maxpool2 = layers.MaxPool2D(2, strides=2)
26         self.flatten = layers.Flatten()
27         self.fcl = layers.Dense(1024)
28         self.dropout = layers.Dropout(rate=0.5)
29         self.out = layers.Dense(10)
30
31     def call(self, x, is_training=False):
32         x = tf.reshape(x, [-1, 28, 28, 1])
33         x = self.conv1(x)
34         x = self.maxpool1(x)
35         x = self.conv2(x)
36         x = self.maxpool2(x)
37         x = self.flatten(x)
38         x = self.fcl(x)
39         x = self.dropout(x, training=is_training)
40         x = self.out(x)
41         return x
42
43 CNN_model = CNN()
44 optimizer = tf.optimizers.Adam(0.01)
```

```
47 def cross_entropy_loss(x,y):
48     y = tf.cast(y, tf.int64)
49     loss = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y, logits=x) # loss에 softmax 가 포함
50     return tf.reduce_mean(loss)
51
52 def accuracy(x,y):
53     correct = tf.equal(tf.argmax(x, 1), tf.cast(y, tf.int64))
54     return tf.reduce_mean(tf.cast(correct, tf.float32), axis=-1)
55
56
57 def train_step(x, y):
58     with tf.GradientTape() as g:
59         y_pred = CNN_model.call(x)
60         loss = cross_entropy_loss(y_pred,y)
61         trainable_variables = CNN_model.trainable_variables
62         gradients = g.gradient(loss, trainable_variables)
63         optimizer.apply_gradients(zip(gradients,trainable_variables))
64
65 for step, (batch_x, batch_y) in enumerate(train_data.take(300), 1):
66     train_step(batch_x, batch_y)
67     if step % 10 == 0:
68         pred = CNN_model.call(batch_x)
69         loss = cross_entropy_loss(pred,batch_y)
70         acc = accuracy(pred, batch_y)
71         print("Step: {},\t Accurarray: {},\t Loss: {}".format(step,acc.numpy().flatten(),loss.numpy().flatten()))
72     print('*'*30)
73     print('Model test')
74     print("Test_Accurarray: ",accuracy(CNN_model(x_test),y_test).numpy().flatten())
75     print('*'*30)
```

- TensorFlow
- **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()
```

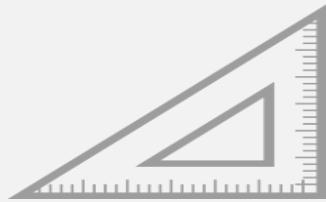
CNN Implementation (Keras)



```
(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 (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_2 (Conv2D)	(None, 14, 14, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(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

Advanced Topics



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

Deep Learning Theory and Software

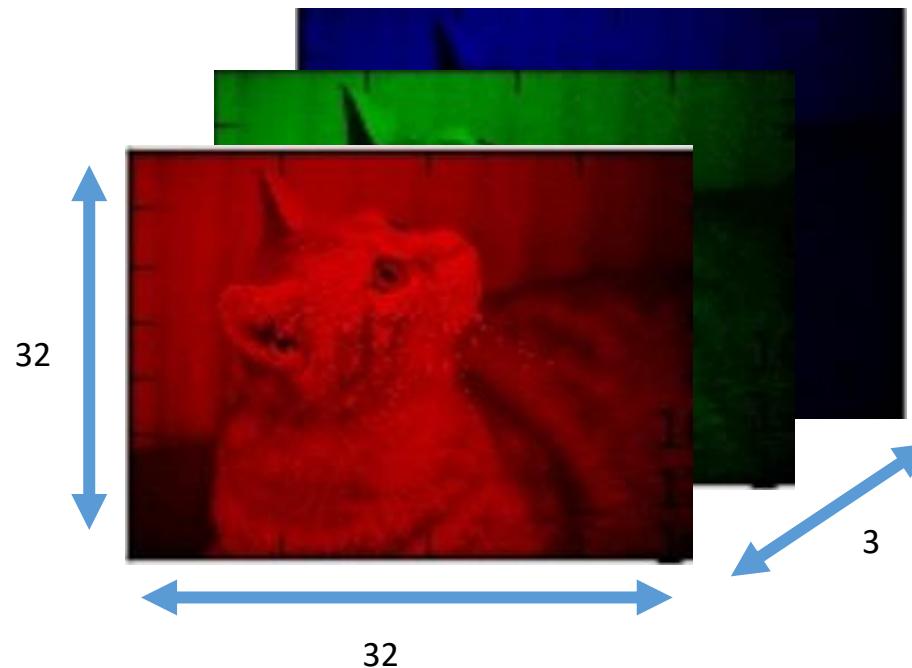
CNN for CIFAR-10

CIFAR-10 Dataset

- **CIFAR-10 Dataset**

	MNIST	CIFAR-10,100	IMAGENET																				
	<pre> 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 </pre>	<table border="1"> <tr> <td>airplane</td> <td></td> </tr> <tr> <td>automobile</td> <td></td> </tr> <tr> <td>bird</td> <td></td> </tr> <tr> <td>cat</td> <td></td> </tr> <tr> <td>deer</td> <td></td> </tr> <tr> <td>dog</td> <td></td> </tr> <tr> <td>frog</td> <td></td> </tr> <tr> <td>horse</td> <td></td> </tr> <tr> <td>ship</td> <td></td> </tr> <tr> <td>truck</td> <td></td> </tr> </table>	airplane		automobile		bird		cat		deer		dog		frog		horse		ship		truck		
airplane																							
automobile																							
bird																							
cat																							
deer																							
dog																							
frog																							
horse																							
ship																							
truck																							
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	55,000	50,000	200,000																				

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



```
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

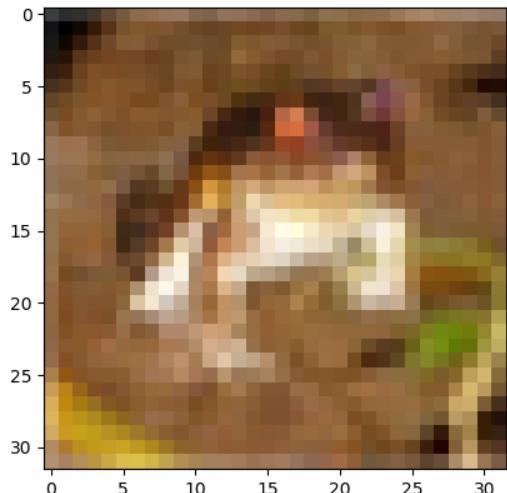
- Train

- $(50000, 32, 32, 3) \rightarrow$ 3 channel RGB (32-by-32), 50000 images
 - $(50000, 1) \rightarrow$ 50000 labels (no one-hot encoding)

- Test

- $(10000, 32, 32, 3) \rightarrow$ 3 channel RGB (32-by-32), 10000 images
 - $(10000, 1) \rightarrow$ 10000 labels (no one-hot encoding)

```
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()
```

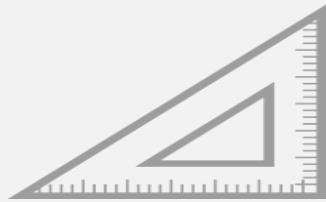


Implementation for CIFAR-10 (Keras)



```
1 from keras.utils import np_utils
2 from keras.datasets import cifar10
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) = cifar10.load_data()
7 print(train_images.shape, train_labels.shape, test_images.shape, test_labels.shape)
8
9 train_images = train_images.reshape(train_images.shape[0], 32,32,3).astype('float32')/255.0
10 test_images = test_images.reshape(test_images.shape[0], 32,32,3).astype('float32')/255.0
11
12 train_labels = np_utils.to_categorical(train_labels) # One-Hot Encoding
13 test_labels = np_utils.to_categorical(test_labels) # One-Hot Encoding
14 # Model
15 model = Sequential()
16 model.add(Conv2D(32, (3,3), padding='same', strides=(1,1), activation='relu', input_shape=(32,32,3)))
17 model.add(pooling.MaxPooling2D(pool_size=(2,2)))
18 model.add(Conv2D(64, (3,3), padding='same', strides=(1,1), activation='relu'))
19 model.add(pooling.MaxPooling2D(pool_size=(2,2)))
20 model.add(Flatten())
21 model.add(Dense(10, activation='softmax')) # units=10, activation='softmax'
22 model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
23 # Training
24 model.fit(train_images, train_labels, epochs=5, batch_size=32, verbose=1)
25 # Testing
26 _, accuracy = model.evaluate(test_images, test_labels)
27 print('Accuracy: ', accuracy)
28 model.summary()
```

Linear Functions



Linear Regression
Binary Classification
Softmax Classification

Nonlinear Functions



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

Advanced Topics



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

Deep Learning Theory and Software

Generative Adversarial Networks (GAN)

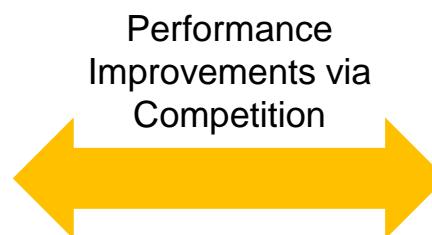
GAN Theory

- **GAN Theory**
- GAN Implementation

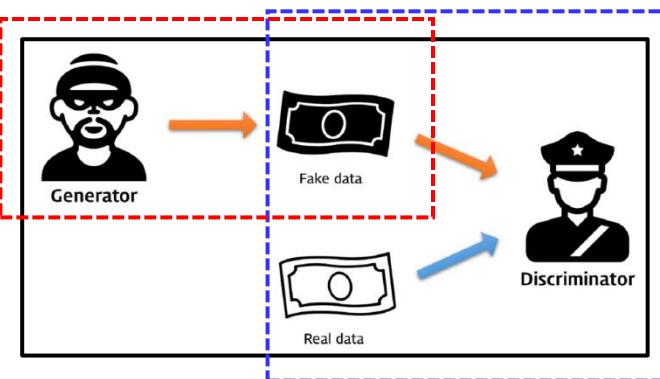
- GAN: Generative Adversarial Network
- Training both of **generator** and **discriminator**; and then generates samples which are similar to the original samples



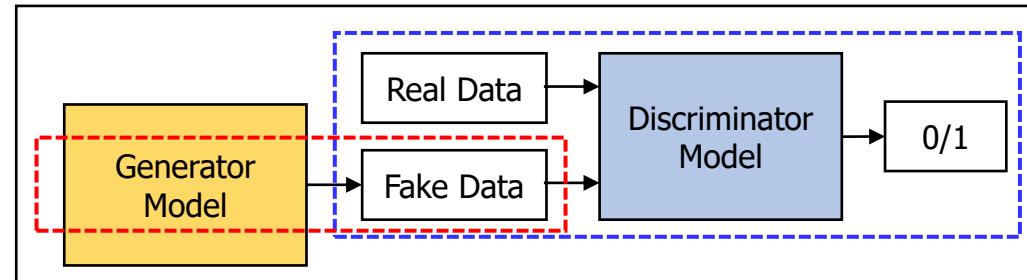
Generators



Discriminator



GAN architecture



Discriminator Model

- The discriminative model learns **how to classify** input to its class (fake → fake class, real → real class).
- Binary classifier.

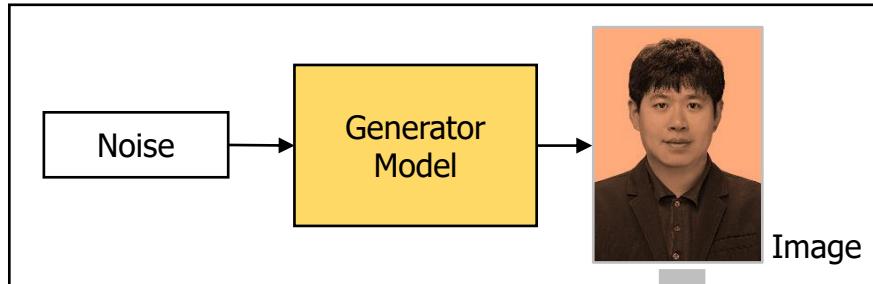
Supervised Learning

Generator Model

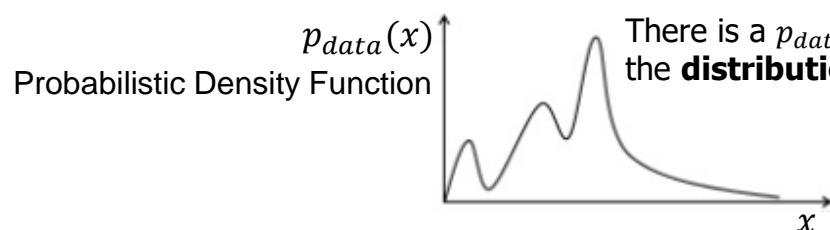
- The generative model learns **the distribution of training data**.

Unsupervised Learning

- Generative Model

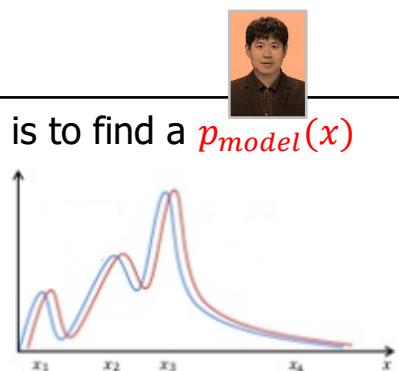


The generator model learns **the distribution of training data**.

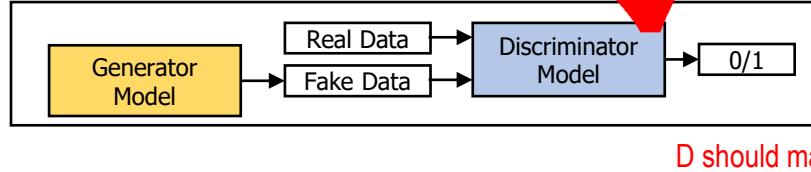


There is a $p_{data}(x)$ that represents
the distribution of actual images (training data).

The goal of the generative model is to find a $p_{model}(x)$
that approximates $p_{data}(x)$ well.



GAN architecture



Objective of D

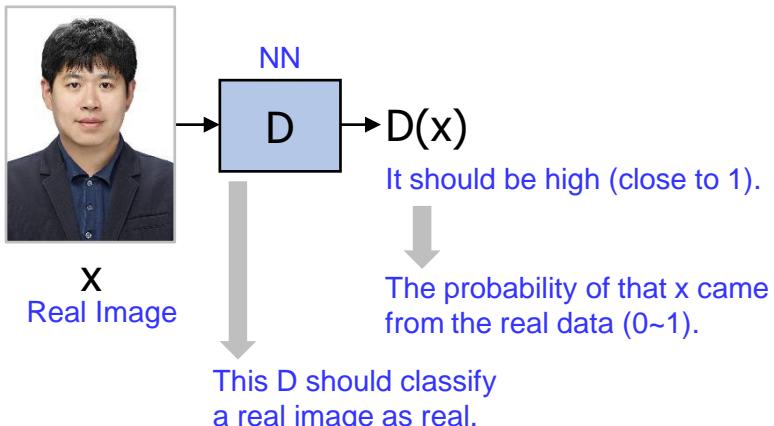
$$\min_G \max_D V(D, G) = E_{x \sim p_{\text{data}}(x)}[\log D(x)] + E_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

D should maximize $V(D, G)$

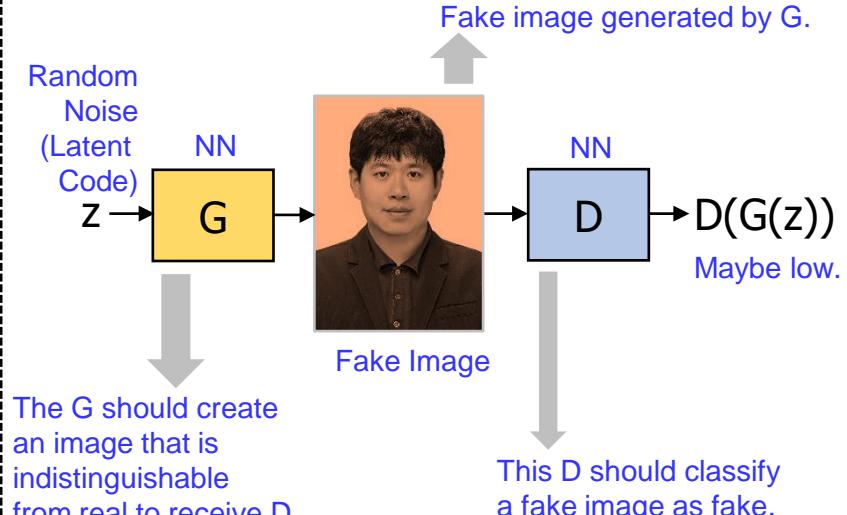
Maximize when $D(x)=1$

Maximize when $D(G(z))=0$

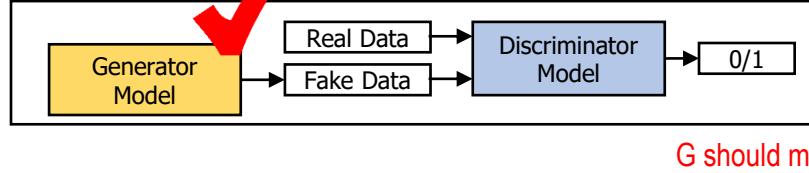
Training with REAL



Training with FAKE



GAN architecture



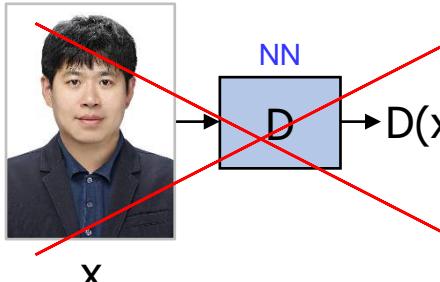
Objective of G

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

Red annotations explain the objective:

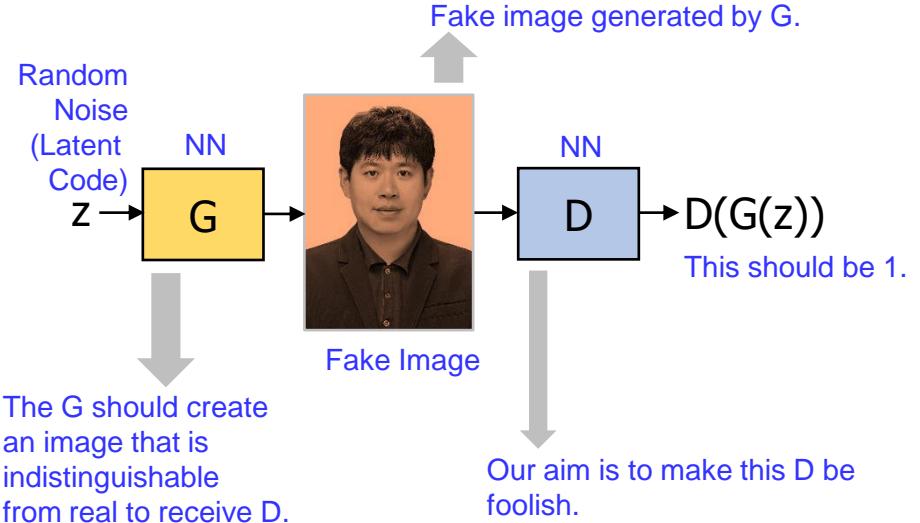
- A red checkmark is placed above the Generator Model box.
- The text "G should minimize V(D,G)" is written below the equation.
- A red arrow points from the text "G is independent to this part" to the term $E_{z \sim p_z(z)}[\log(1 - D(G(z)))]$.
- A red arrow points from the text "Sample latent code x from Gaussian distribution" to the term $E_{x \sim p_{data}(x)}[\log D(x)]$.
- A red arrow points from the text "Minimum when $D(G(z))=1$ " to the term $E_{z \sim p_z(z)}[\log(1 - D(G(z)))]$.

Training with REAL

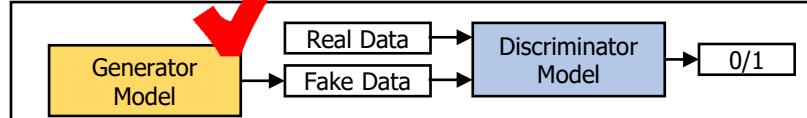


G has no inputs.

Training with FAKE



GAN architecture



G should minimize $V(D, G)$

Objective of G

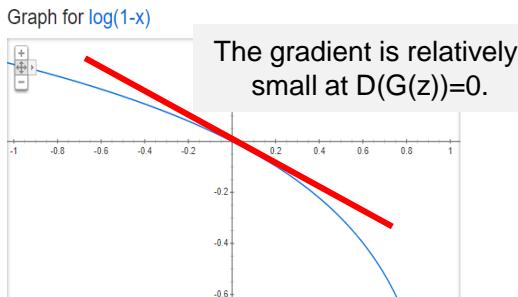
$$\min_G \max_D V(D, G) = E_{x \sim p_{data}(x)}[\log D(x)] + E_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

G is independent to this part

Sample latent code x from Gaussian distribution

Minimum when $D(G(z))=1$

- At the beginning of training, the D can clearly classify the generated image as fake because the quality of the image is very low.
- This means $D(G(z))$ is almost zero at early stage of training.



$$\min_G E_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

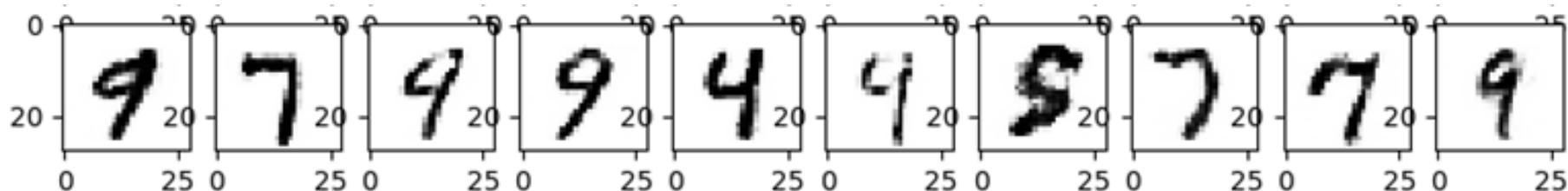
$$\max_G E_{z \sim p_z(z)}[\log(D(G(z)))]$$

Deep Learning Theory and Software

Generative Adversarial Networks (GAN)

GAN Implementation

- GAN Theory
- **GAN Implementation**



```
1 # MNIST data
2 from tensorflow.examples.tutorials.mnist import input_data
3 mnist = input_data.read_data_sets("data_MNIST", one_hot=True)
4
5 import matplotlib.pyplot as plt
6 import numpy as np
7 import tensorflow as tf
8
9 # Training Params
10 num_steps = 100000
11 batch_size = 128
12
13 # Network Params
14 dim_image = 784 # 28*28 pixels
15 nHL_G = 256
16 nHL_D = 256
17 dim_noise = 100 # Noise data points
18
19 # A custom initialization (Xavier Glorot init)
20 def glorot_init(shape):
21     return tf.random_normal(shape=shape, stddev=1. / tf.sqrt(shape[0] / 2.))
```

```

23   W = {
24     'HL_G' : tf.Variable(glorot_init([dim_noise, nHL_G])),
25     'OL_G' : tf.Variable(glorot_init([nHL_G, dim_image])),
26     'HL_D' : tf.Variable(glorot_init([dim_image, nHL_D])),
27     'OL_D' : tf.Variable(glorot_init([nHL_D, 1])),
28   }
29   b = {
30     'HL_G' : tf.Variable(tf.zeros([nHL_G])),
31     'OL_G' : tf.Variable(tf.zeros([dim_image])),
32     'HL_D' : tf.Variable(tf.zeros([nHL_D])),
33     'OL_D' : tf.Variable(tf.zeros([1])),
34   }
35
36   # Neural Network: Generator
37   def nn_G(x):
38     HL = tf.nn.relu(tf.add(tf.matmul(x, W['HL_G']), b['HL_G']))
39     OL = tf.nn.sigmoid(tf.add(tf.matmul(HL, W['OL_G']), b['OL_G']))
40     return OL
41
42   # Neural Network: Discriminator
43   def nn_D(x):
44     HL = tf.nn.relu(tf.add(tf.matmul(x, W['HL_D']), b['HL_D']))
45     OL = tf.nn.sigmoid(tf.add(tf.matmul(HL, W['OL_D']), b['OL_D']))
46     return OL
47
48   # Network Inputs
49   IN_G = tf.placeholder(tf.float32, shape=[None, dim_noise])
50   IN_D = tf.placeholder(tf.float32, shape=[None, dim_image])

```

```

52 # Build Generator Neural Network
53 sample_G = nn_G(IN_G) --> G(z)
54                                         D(x)           D(G(z))
55 # Build Discriminator Neural Network (one from noise input, one from generated samples)
56 D_real = nn_D(IN_D)
57 D_fake = nn_D(sample_G)
58 vars_G = [W['HL_G'], W['OL_G'], b['HL_G'], b['OL_G']]
59 vars_D = [W['HL_D'], W['OL_D'], b['HL_D'], b['OL_D']]
60
61 # Cost, Train
62 cost_G = -tf.reduce_mean(tf.log(D_fake))
63 cost_D = -tf.reduce_mean(tf.log(D_real) + tf.log(1. - D_fake))
64 train_G = tf.train.AdamOptimizer(0.0002).minimize(cost_G, var_list=vars_G)
65 train_D = tf.train.AdamOptimizer(0.0002).minimize(cost_D, var_list=vars_D)

```

Objective of G $\max_G E_{z \sim p_z(z)}[\log(D(G(z)))]$

Objective of D

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

D should maximize $V(D, G)$

Sample x from real data distribution

Sample latent code z from Gaussian distribution

Maximize when $D(x)=1$

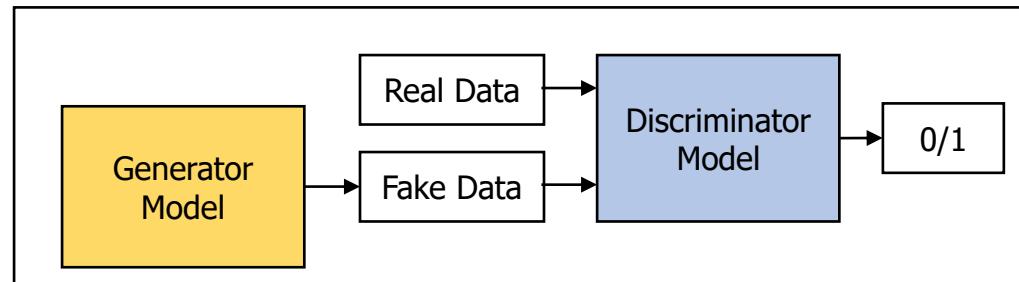
Maximize when $D(G(z))=0$

```

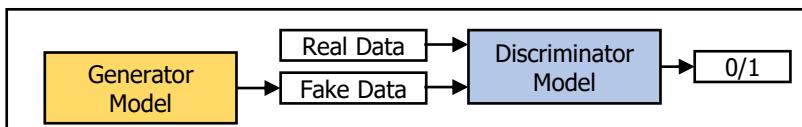
52 # Build Generator Neural Network
53 sample_G = nn_G(IN_G)
54
55 # Build Discriminator Neural Network (one from noise input, one from generated samples)
56 D_real = nn_D(IN_D)
57 D_fake = nn_D(sample_G)
58 vars_G = [W['HL_G'], W['OL_G'], b['HL_G'], b['OL_G']]
59 vars_D = [W['HL_D'], W['OL_D'], b['HL_D'], b['OL_D']]
60
61 # Cost, Train
62 cost_G = -tf.reduce_mean(tf.log(D_fake))
63 cost_D = -tf.reduce_mean(tf.log(D_real) + tf.log(1. - D_fake))
64 train_G = tf.train.AdamOptimizer(0.0002).minimize(cost_G, var_list=vars_G)
65 train_D = tf.train.AdamOptimizer(0.0002).minimize(cost_D, var_list=vars_D)

```

GAN architecture



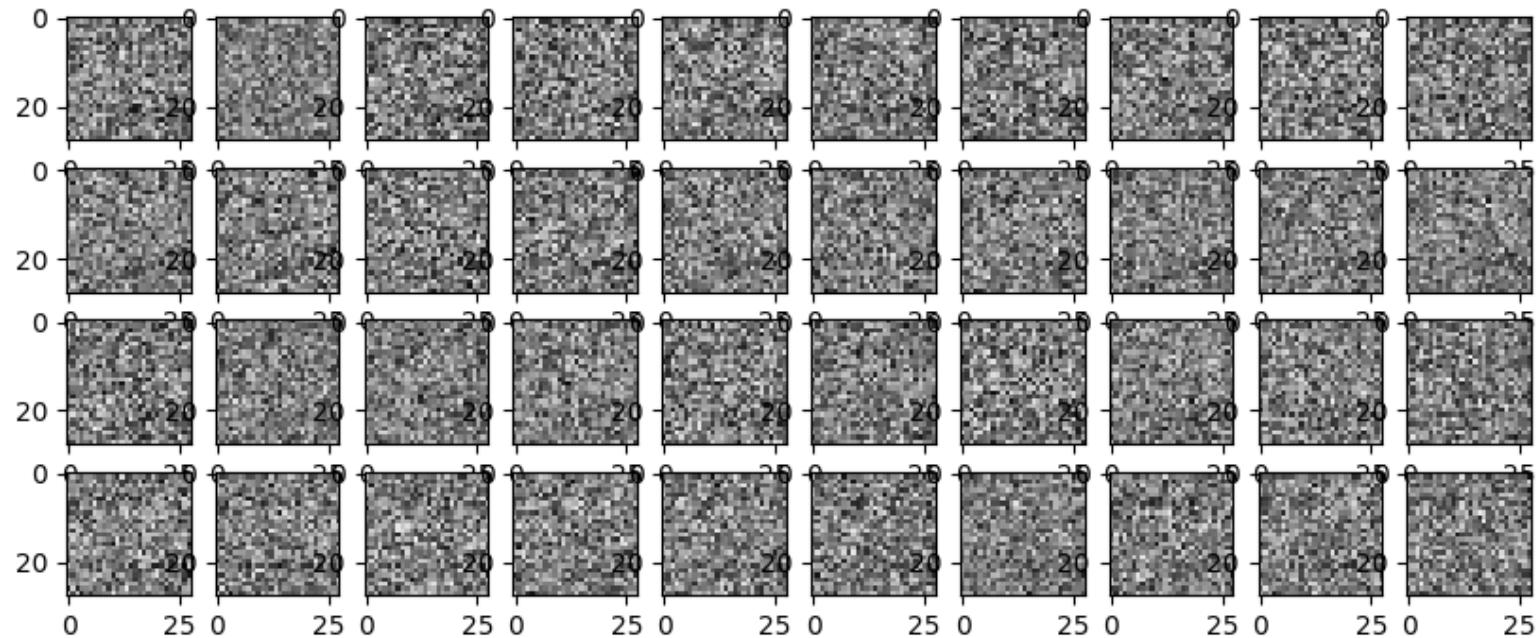
GAN architecture



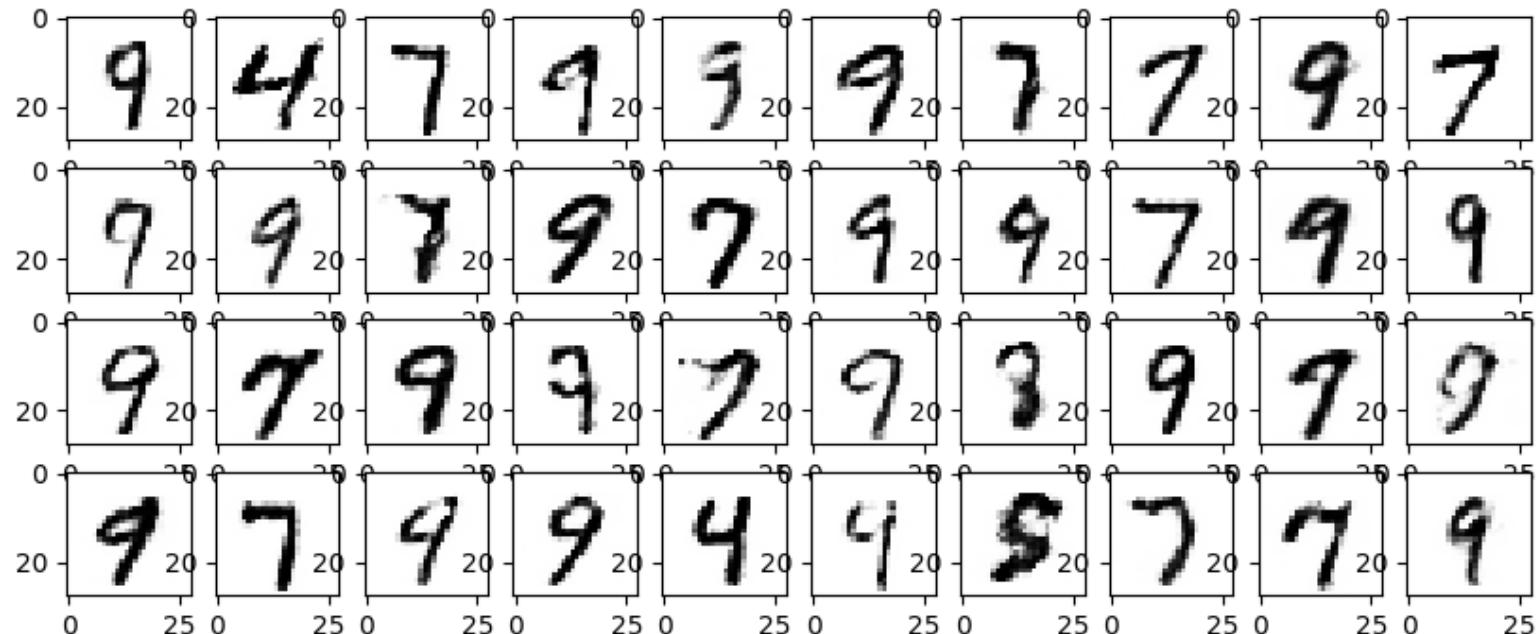
```

67 # Session
68 with tf.Session() as sess:
69     sess.run(tf.global_variables_initializer())
70     for i in range(1, num_steps+1):
71         # Get the next batch of MNIST data
72         batch_images, _ = mnist.train.next_batch(batch_size)
73         # Generate noise to feed to the generator G
74         z = np.random.uniform(-1., 1., size=[batch_size, dim_noise])
75         # Train
76         sess.run([train_G, train_D], feed_dict = {IN_D: batch_images, IN_G: z})
77         f, a = plt.subplots(4, 10, figsize=(10, 4))
78         for i in range(10):
79             z = np.random.uniform(-1., 1., size=[4, dim_noise])
80             g = sess.run([sample_G], feed_dict={IN_G: z})
81             g = np.reshape(g, newshape=(4, 28, 28, 1))
82             # Reverse colors for better display
83             g = -1 * (g - 1)
84             for j in range(4):
85                 # Generate image from noise. Extend to 3 channels for matplotlib figure.
86                 img = np.reshape(np.repeat(g[j][:, :, np.newaxis], 3, axis=2), newshape=(28, 28, 3))
87                 a[j][i].imshow(img)
88         f.show()
89         plt.draw()
90         plt.waitforbuttonpress()
  
```

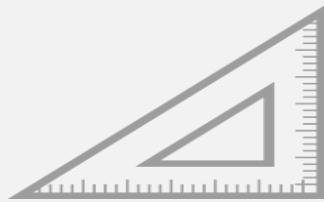
num_steps: 1



num_steps: 100000



Linear Functions



Linear Regression
Binary Classification
Softmax Classification

Nonlinear Functions



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

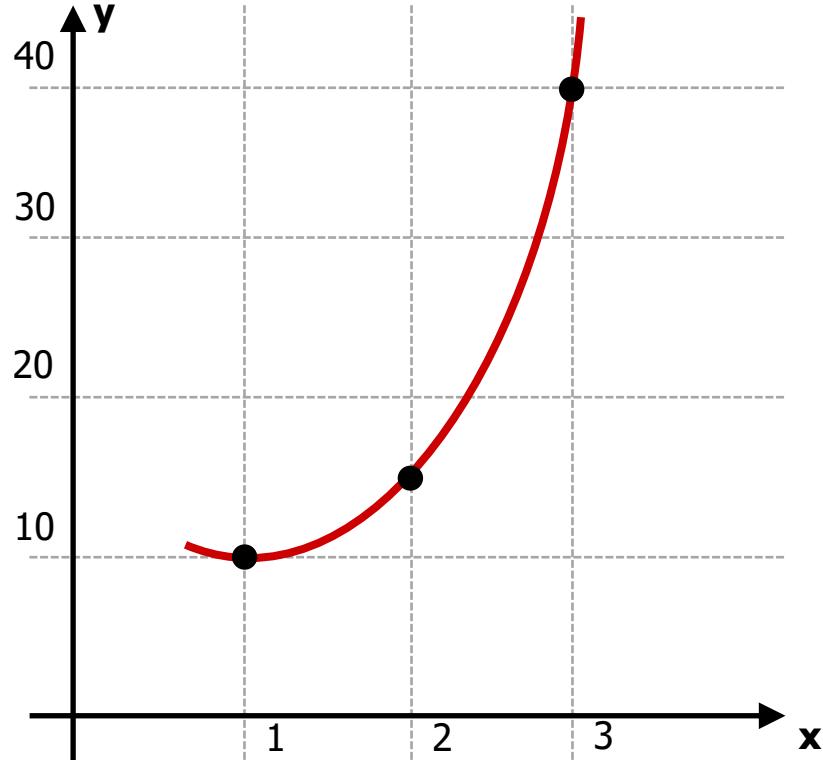
Advanced Topics



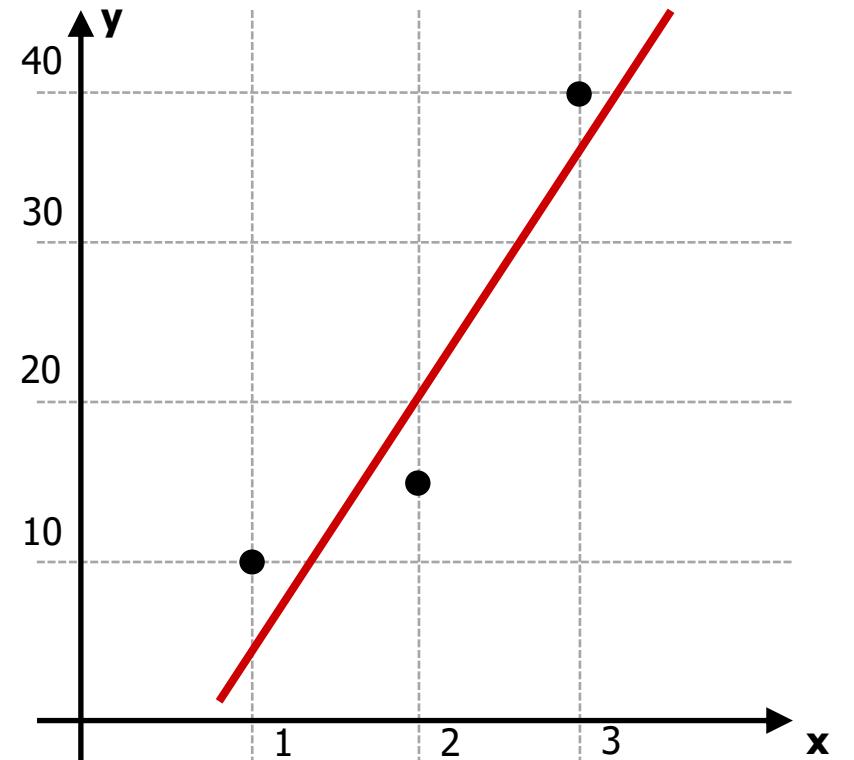
Gen. Adv. Network (GAN)
Interpolation
PCA/LDA
Overfitting

Interpolation vs. Linear Regression

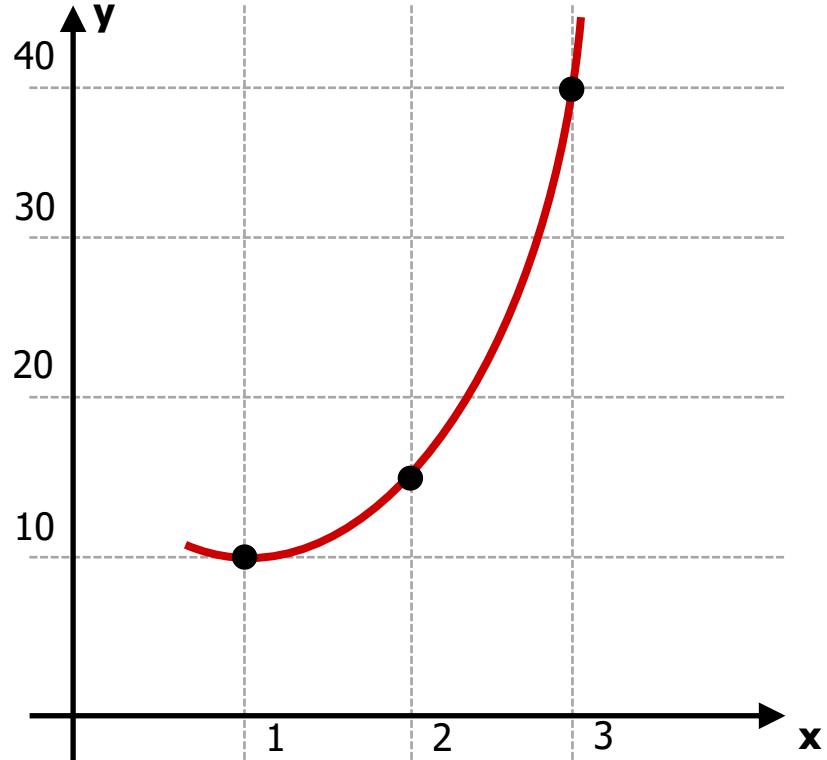
Interpolation



Linear Regression



Interpolation



Interpolation with Polynomials

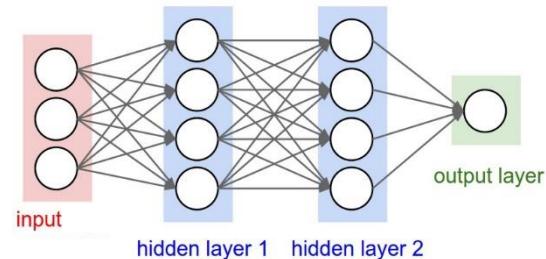
$$y = a_2 x^2 + a_1 x^1 + a_0$$

where three points are given.

→ Unique coefficients (a_0, a_1, a_2) can be calculated.



Is this related to
Neural Network Training?

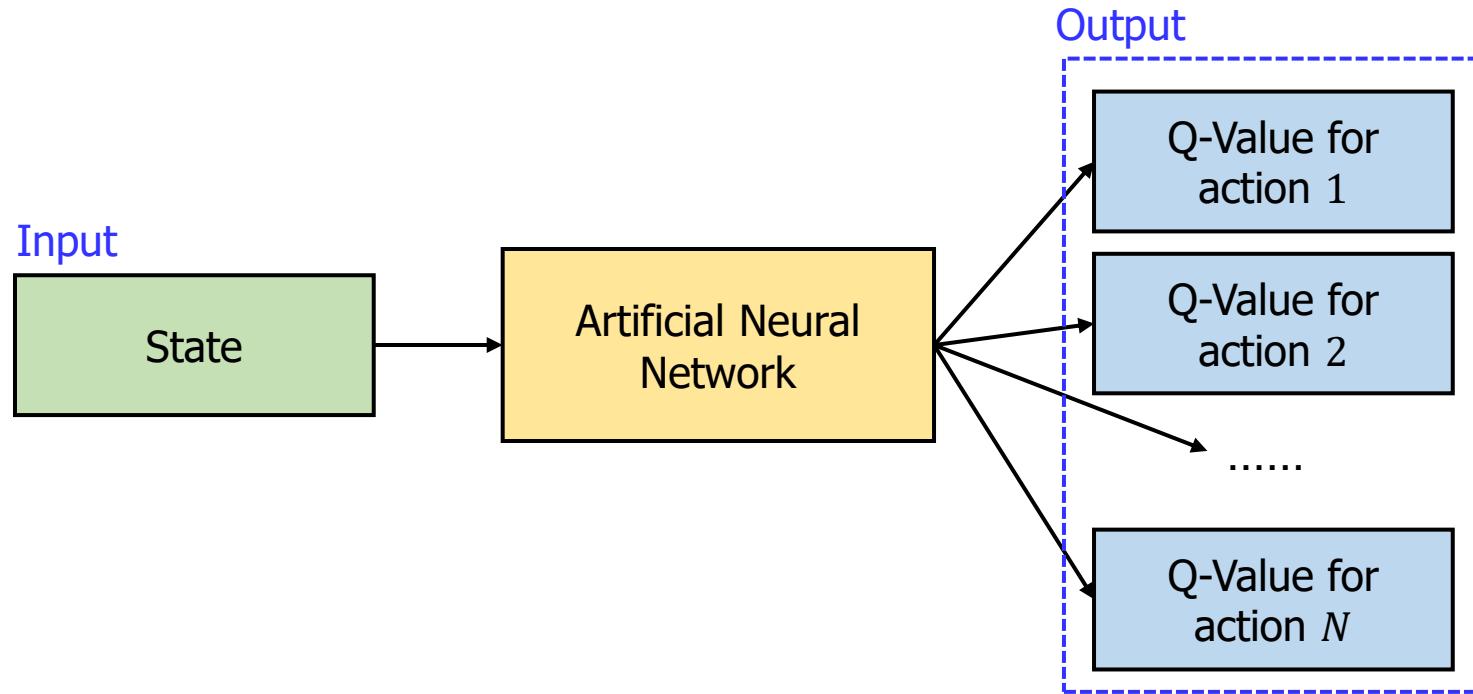


$$Y = a(a(a(X \cdot W_1 + b_1) \cdot W_2 + b_2) \cdot W_o + b_o)$$

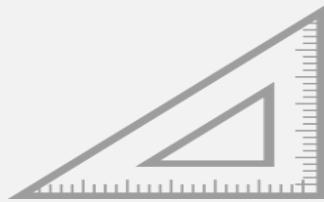
where training data/labels (X : data, Y : labels) are given.

- 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

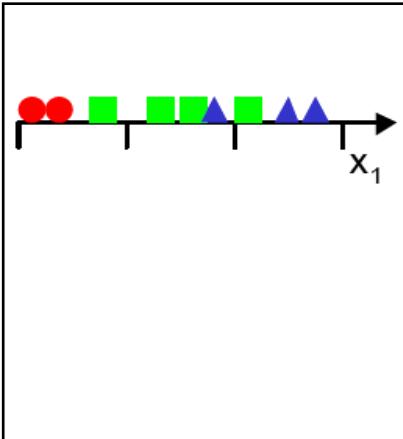
Advanced Topics



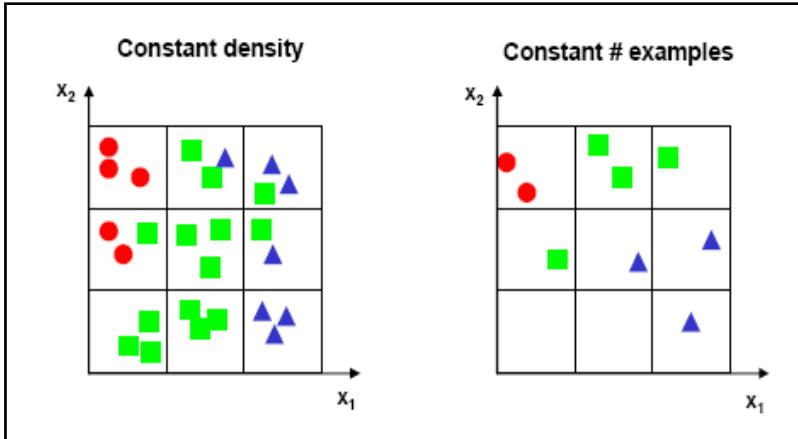
Gen. Adv. Network (GAN)
Interpolation
PCA/LDA
Overfitting

Curse of Dimensionality

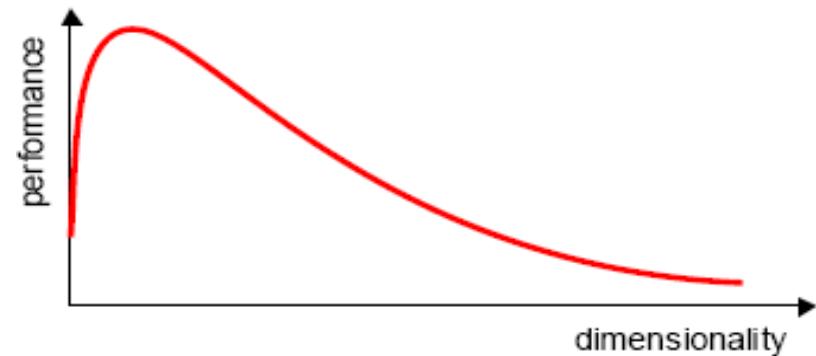
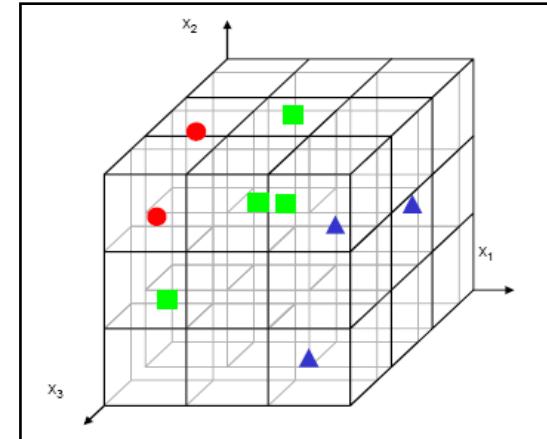
1D



2D



3D



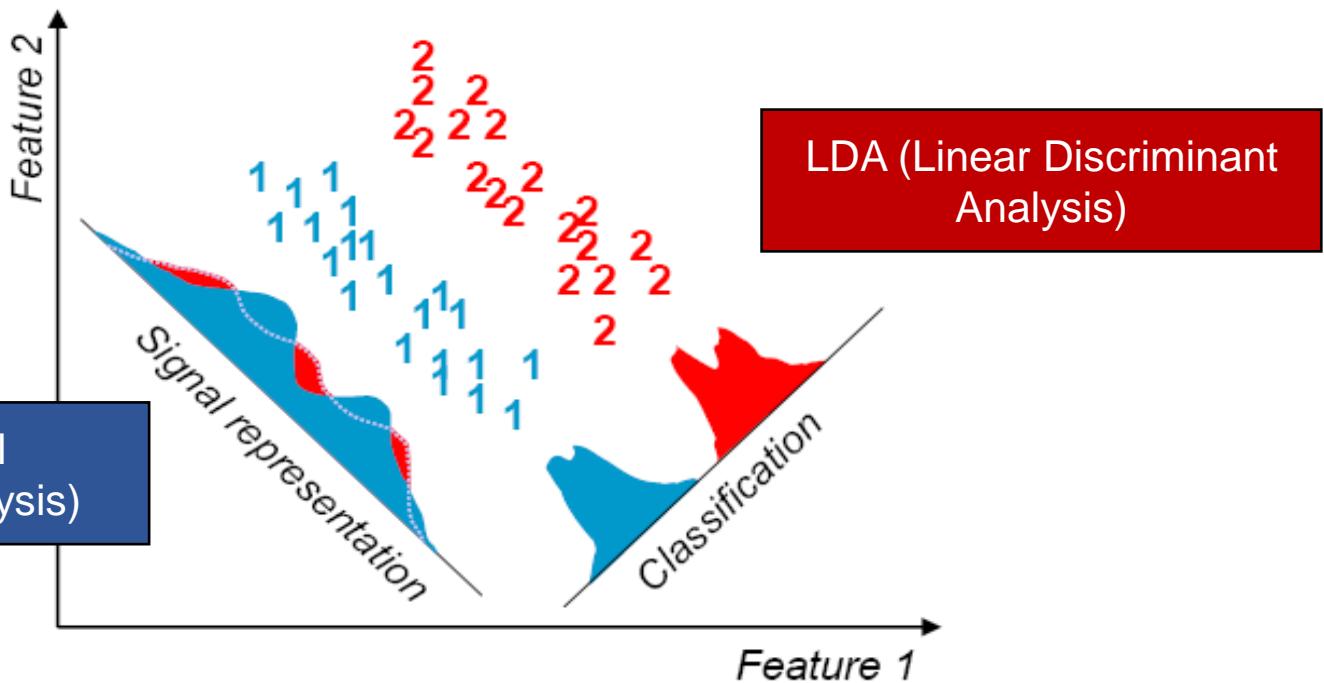
Feature Selection

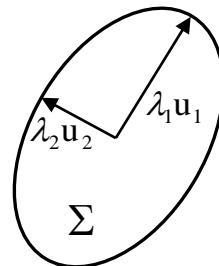
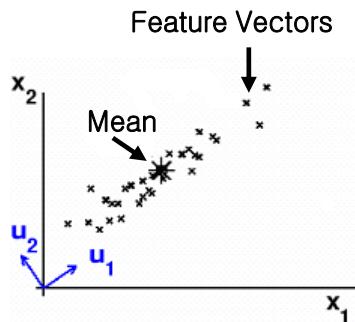
$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \xrightarrow{\text{feature selection}} \begin{bmatrix} x_{i_1} \\ x_{i_2} \\ \vdots \\ x_{i_M} \end{bmatrix}$$

Feature Extraction

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

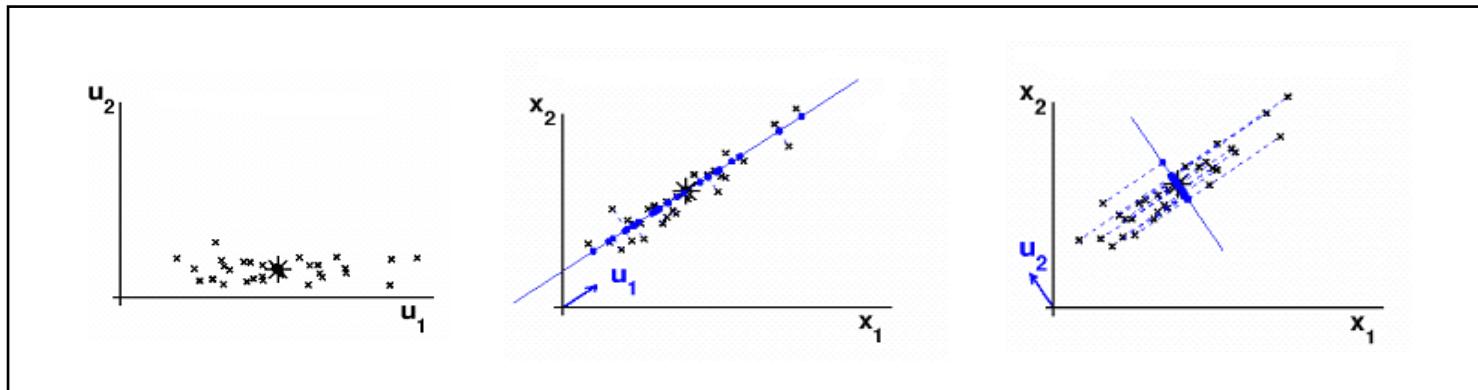
$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \xrightarrow{\text{linear feature extraction}} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M1}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





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

$$\lambda_1 > \lambda_2$$





35세 남자



40세 남자



닮 . 은 . 연 . 예 . 인

72%
이병현
남 41세

얼핏보면 연예인으로
착각하겠어요~

1위

작는, 순간 빠진다! 푸딩 얼굴인식

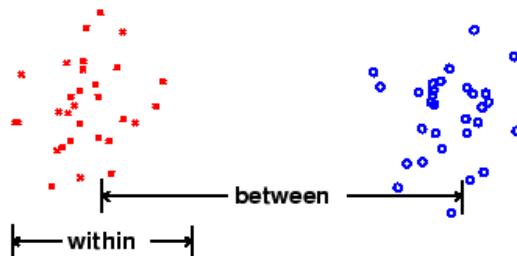
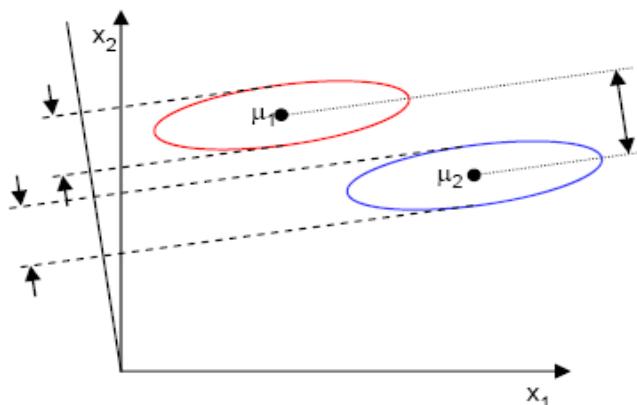
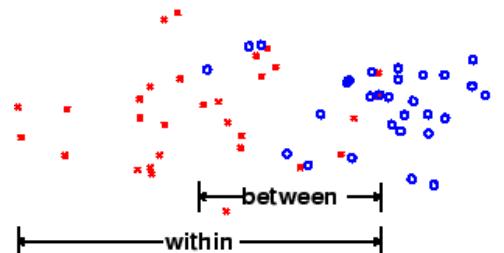
닮 . 은 . 연 . 예 . 인

46%
엄앵란
여 75세

그다지 닮진 않았지만
굳이 한명을 뽑자면...

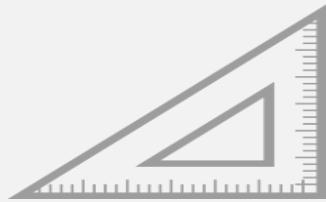
1위

작는, 순간 빠진다! 푸딩 얼굴인식

Good class separation**Bad class separation****Fisher's linear discriminant function**

$$J(\mathbf{w}) = \frac{|\tilde{\boldsymbol{\mu}}_1 - \tilde{\boldsymbol{\mu}}_2|^2}{\tilde{\mathbf{S}}_1^2 + \tilde{\mathbf{S}}_2^2} \rightarrow \begin{array}{l} \text{Maximize!} \\ \text{Minimize!} \end{array}$$

Linear Functions



Linear Regression
Binary Classification
Softmax Classification

Nonlinear Functions



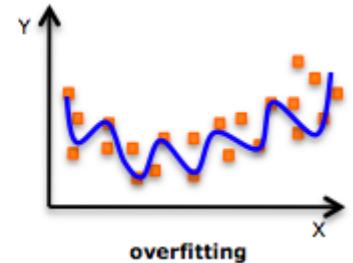
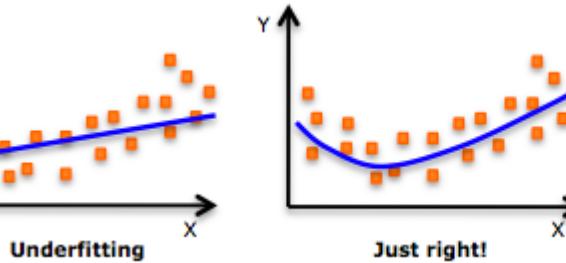
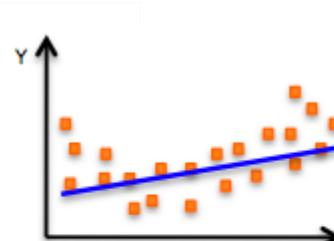
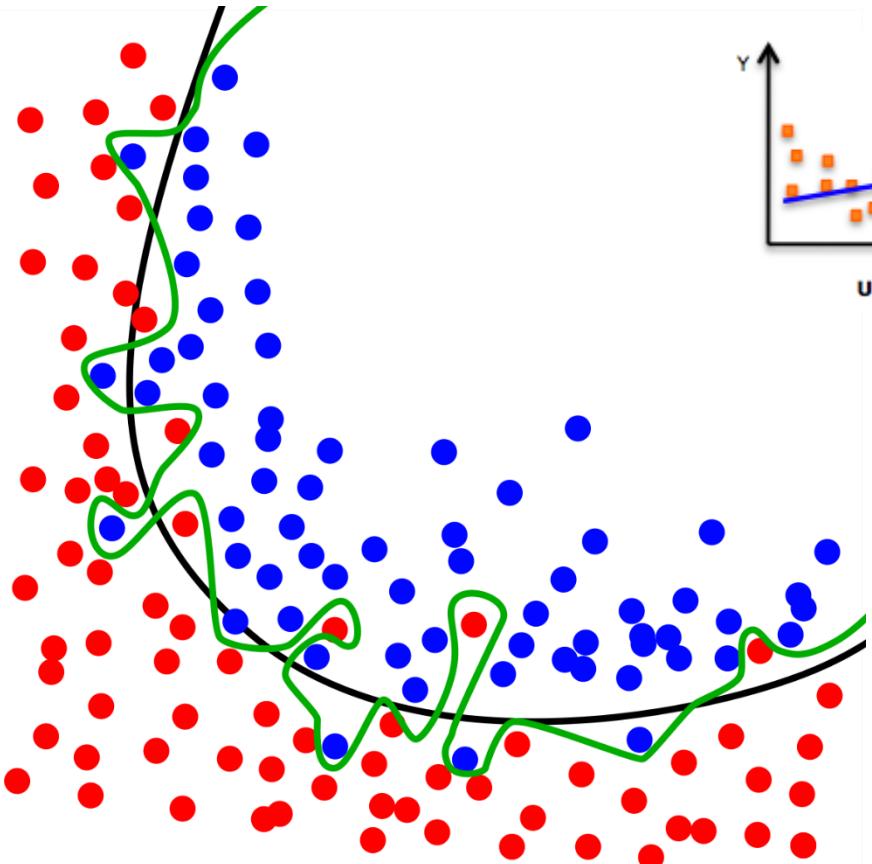
Neural Network (NN)
Convolutional NN (CNN)
CNN for CIFAR-10

Advanced Topics

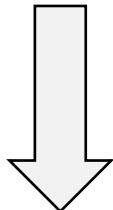


Gen. Adv. Network (GAN)
Interpolation
PCA/LDA
Overfitting

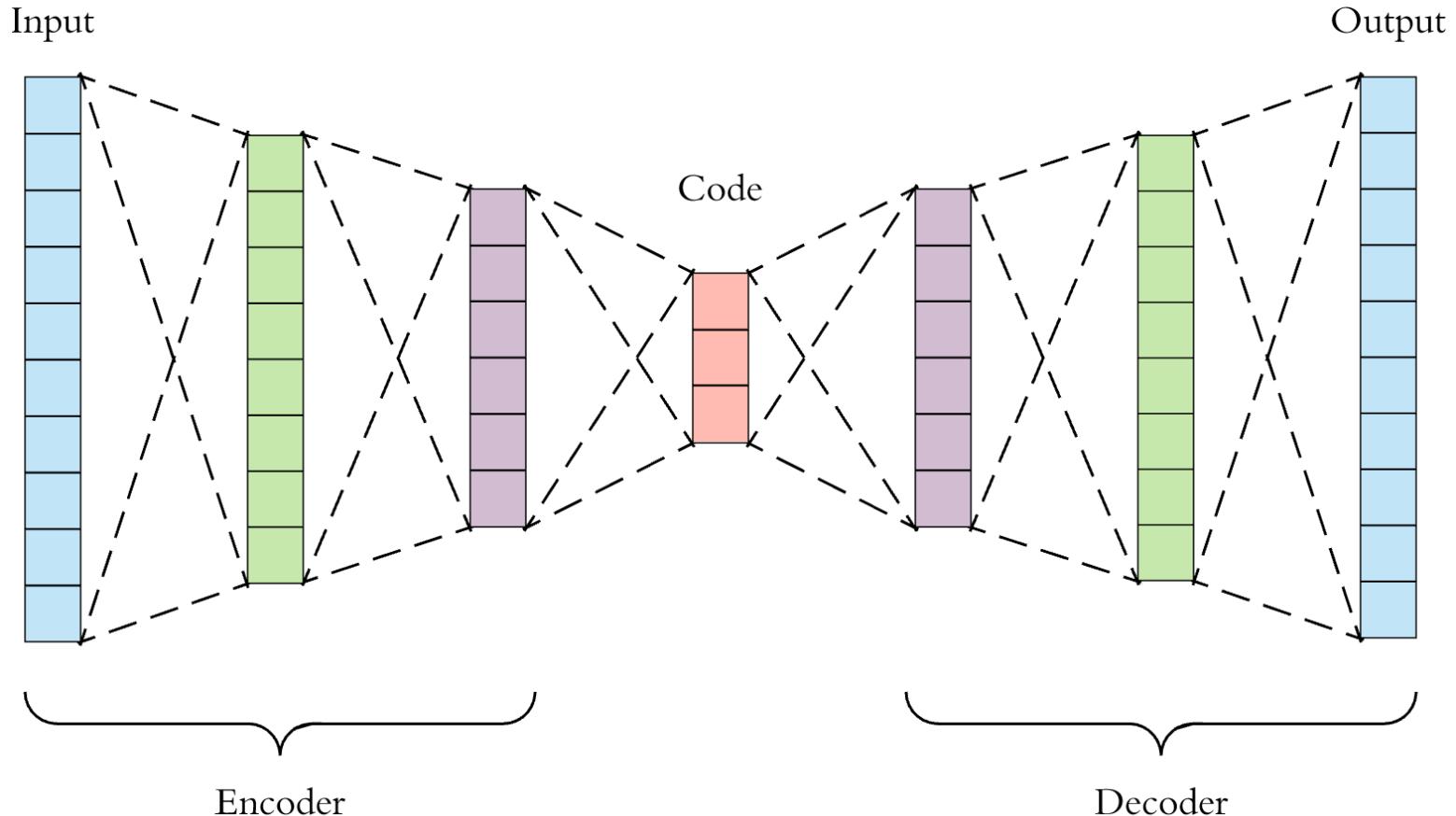
Overfitting



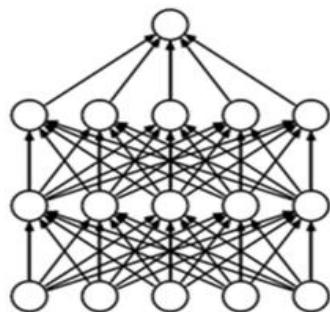
- How to overcome?
 - More training data
 - Reduce the number of features → autoencoding, dropout



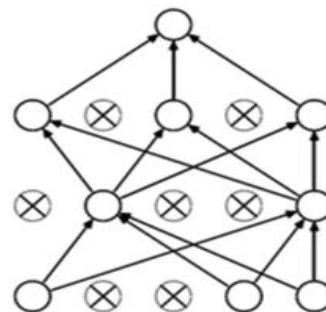
- **Autoencoding**
- **Dropout**
- **Regularization**



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

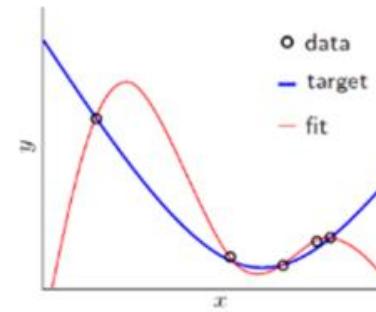


(a) Standard Neural Net

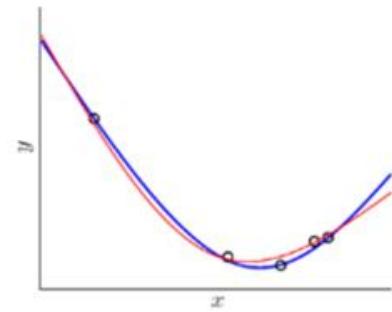


(b) After applying dropout.

- Regularization



(a) without regularization



(b) with regularization