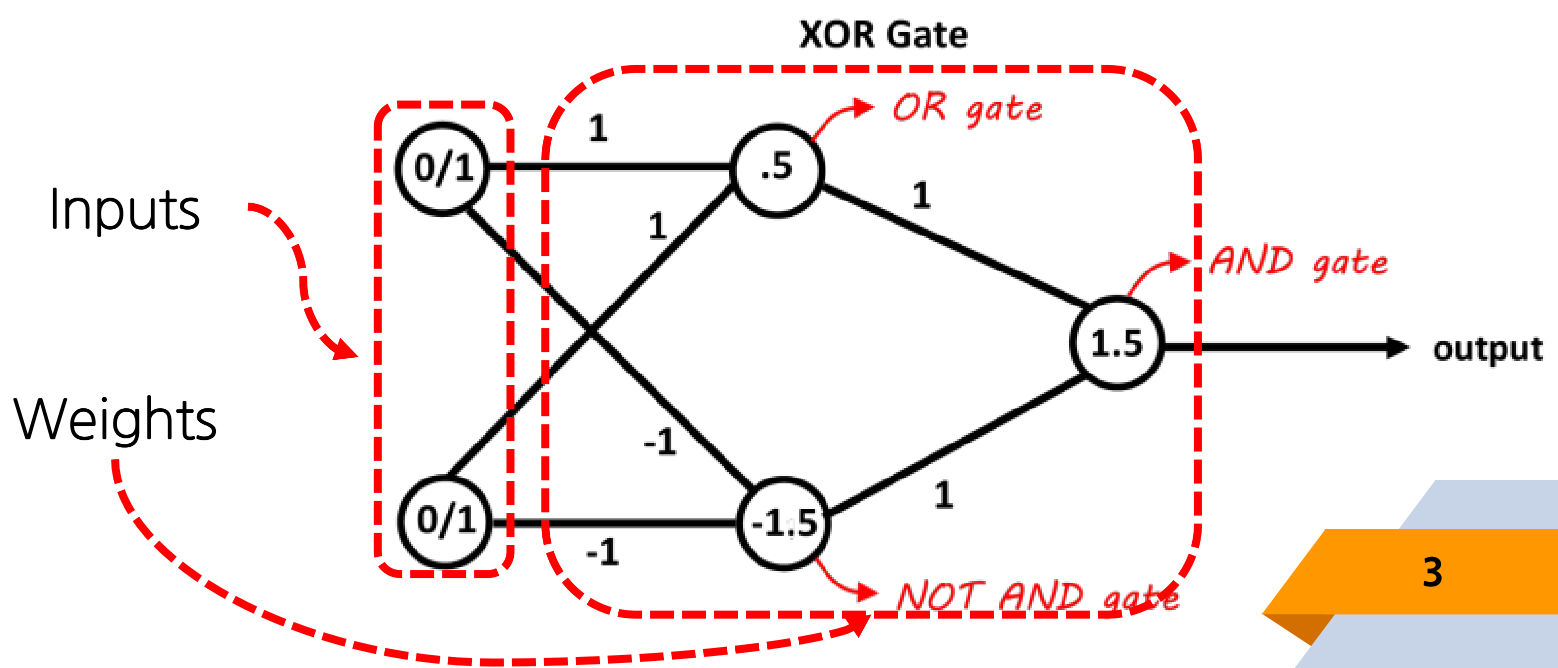
CNN 및 딥러닝 기초

Natural Language Processing & AI Lab.,
Korea University

Machine Learning Overview

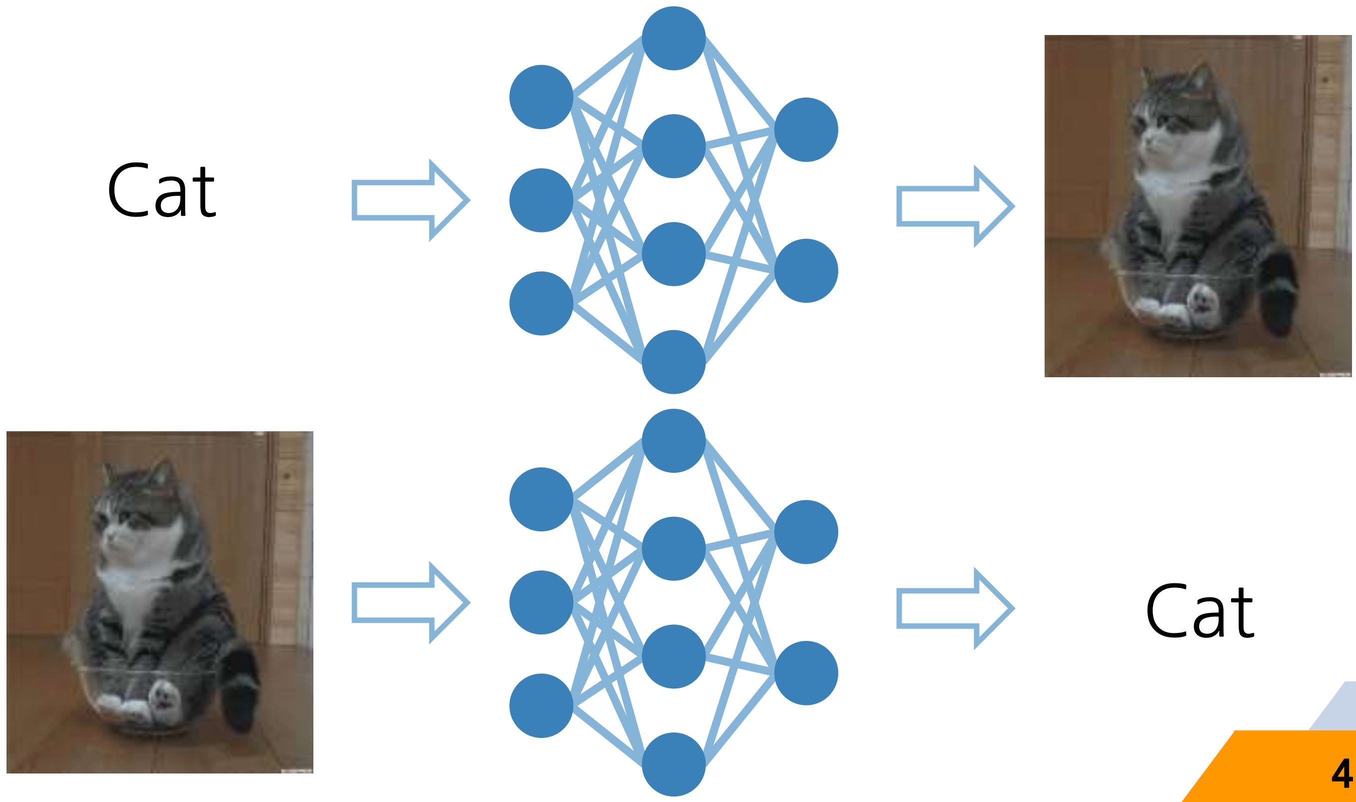
Neural Networks

- Transforms input to output
- NN is a "set of weights(parameters)"
- Need to change the weights(train) to make it do what we want



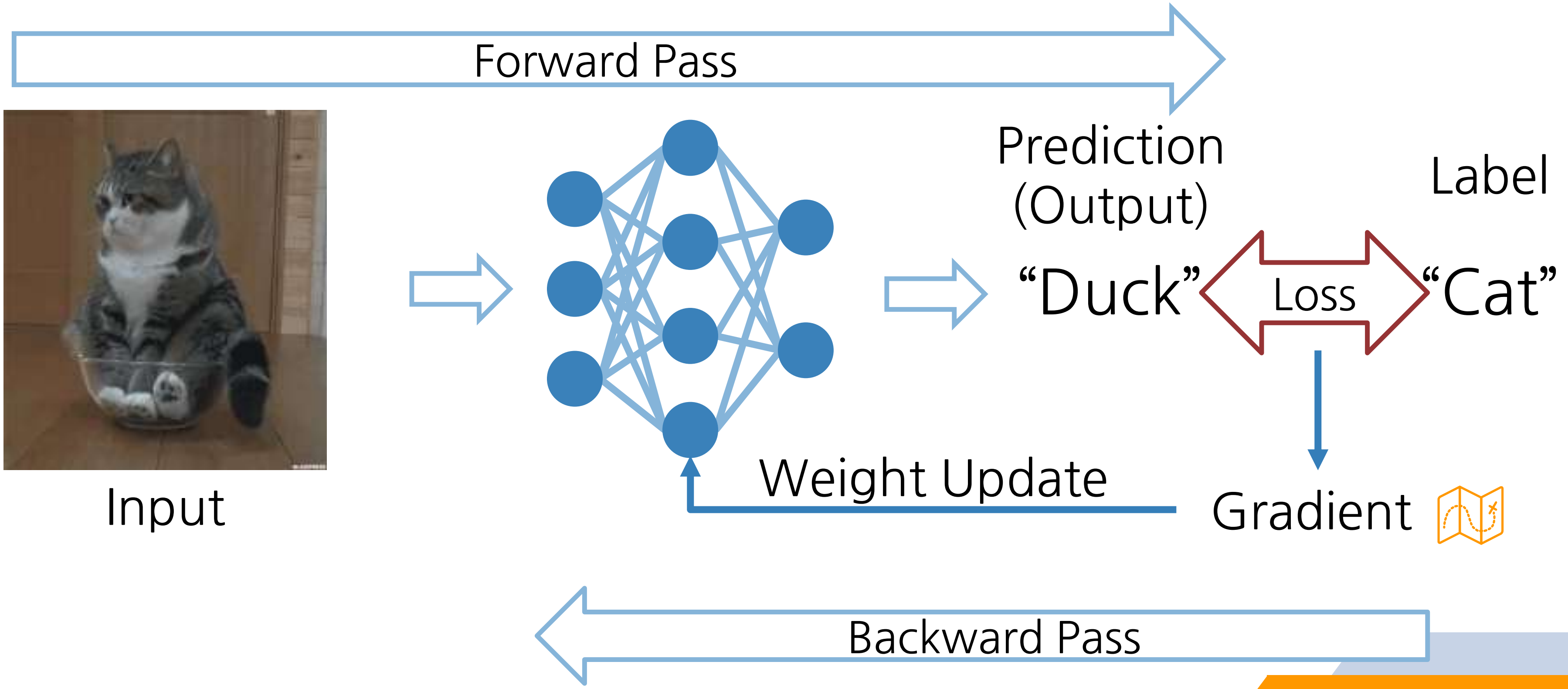
Neural Networks

- Very complex transformations (functions) can be approximated using NNs



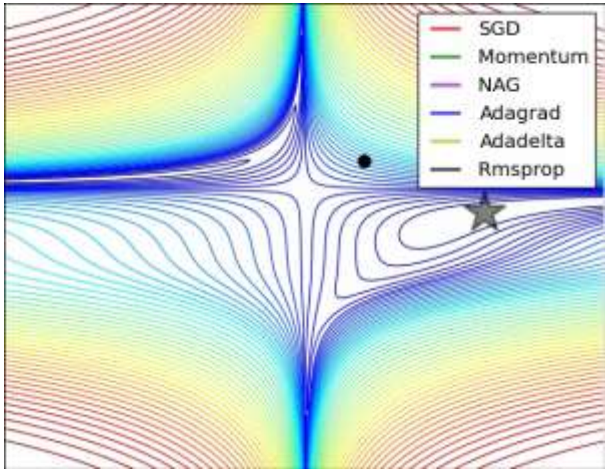
How to Train a NN

- With loss and gradient



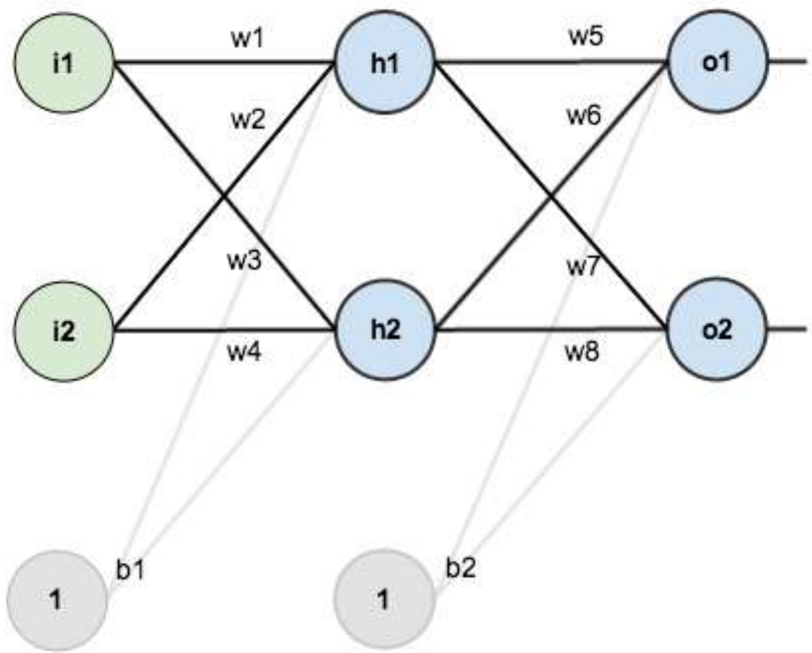
Gradient Descent Algorithms

- 함수 $F(x)$의 local minimum을 찾는 알고리즘
- 흔히, 산을 내려갈 때 현재 지점에서 가장 경사가 가파른 방향으로 이동하는 것으로 비유
- 한 점 $a = (x_1, x_2, \dots, x_n)$에서 시작하여 아래 과정을 일정 횟수 동안 혹은 $F(a)$의 변화가 threshold보다 작아질 때까지 반복
  - 각 x_k $(1 \le k \le n)$에 대하여 $F(x)$의 partial derivative $\frac{dF}{dx_k}$ (gradient) 계산
  - $x_k \coloneqq x_k - \gamma \frac{dF}{dx_k}$ (γ: learning rate)



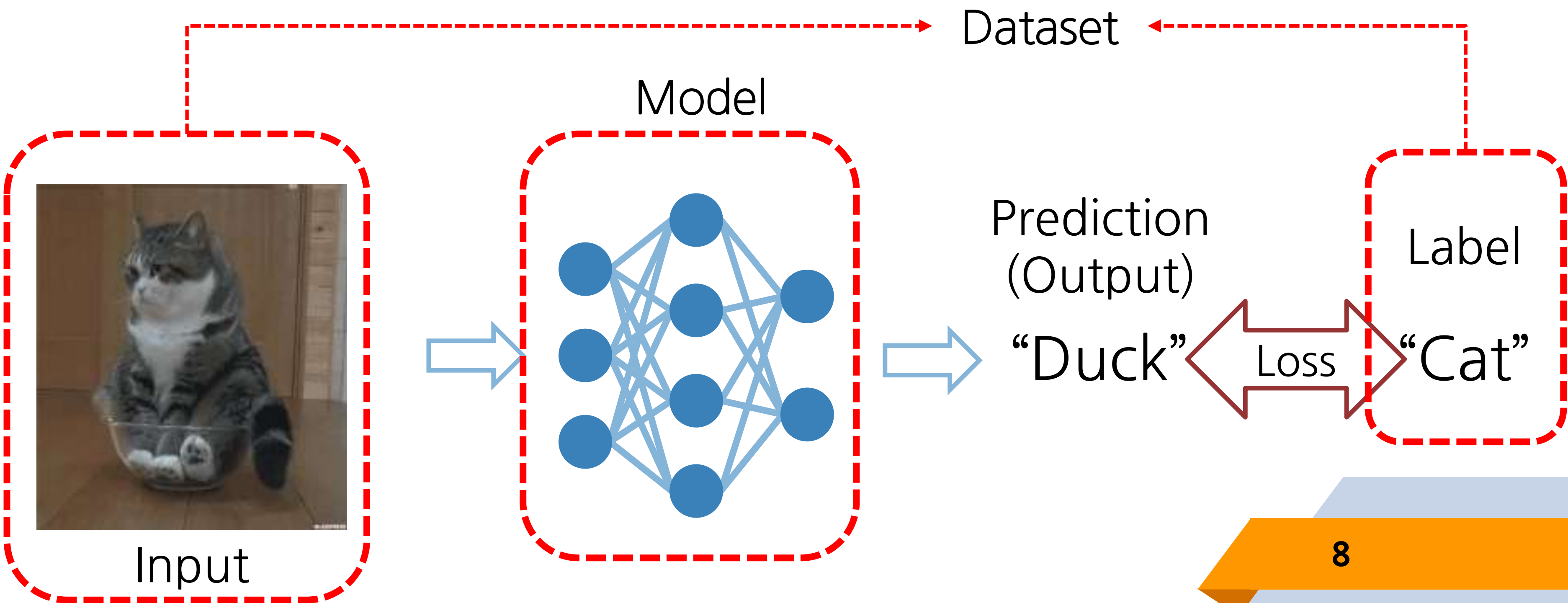
Backpropagation Step-by-Step

- https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/



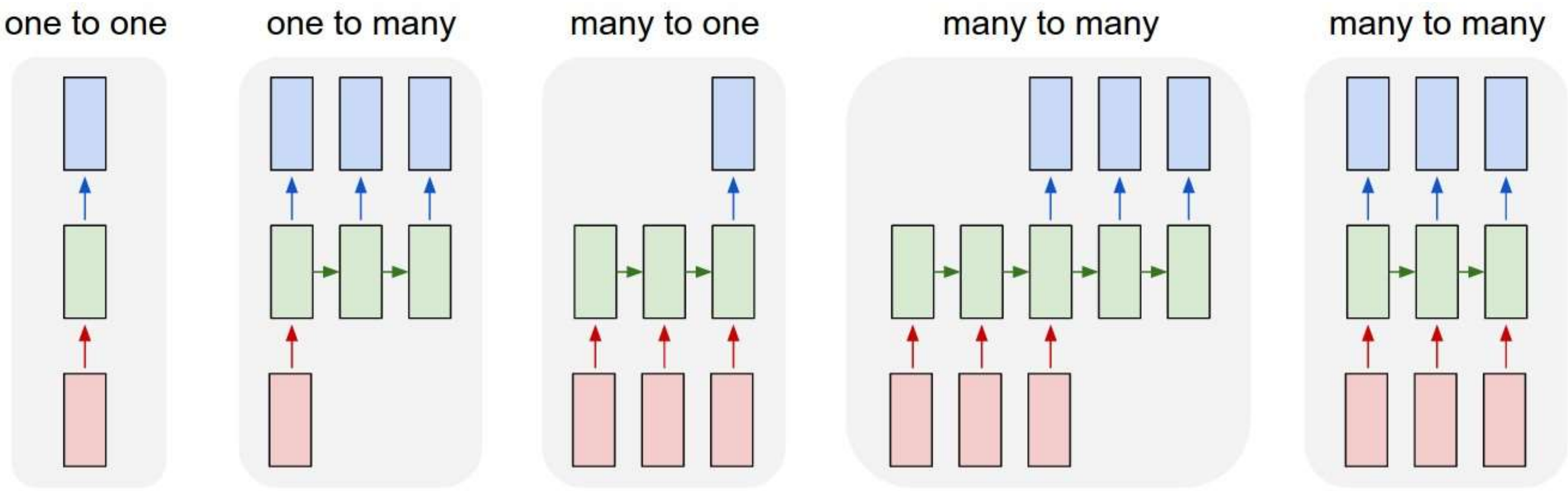
Key Components of NN

- Input and label: these forms a "dataset"
- Neural network model
- …



Types of NN Model

- Almost all neural network models can be classified into one of the followings



Things to Consider

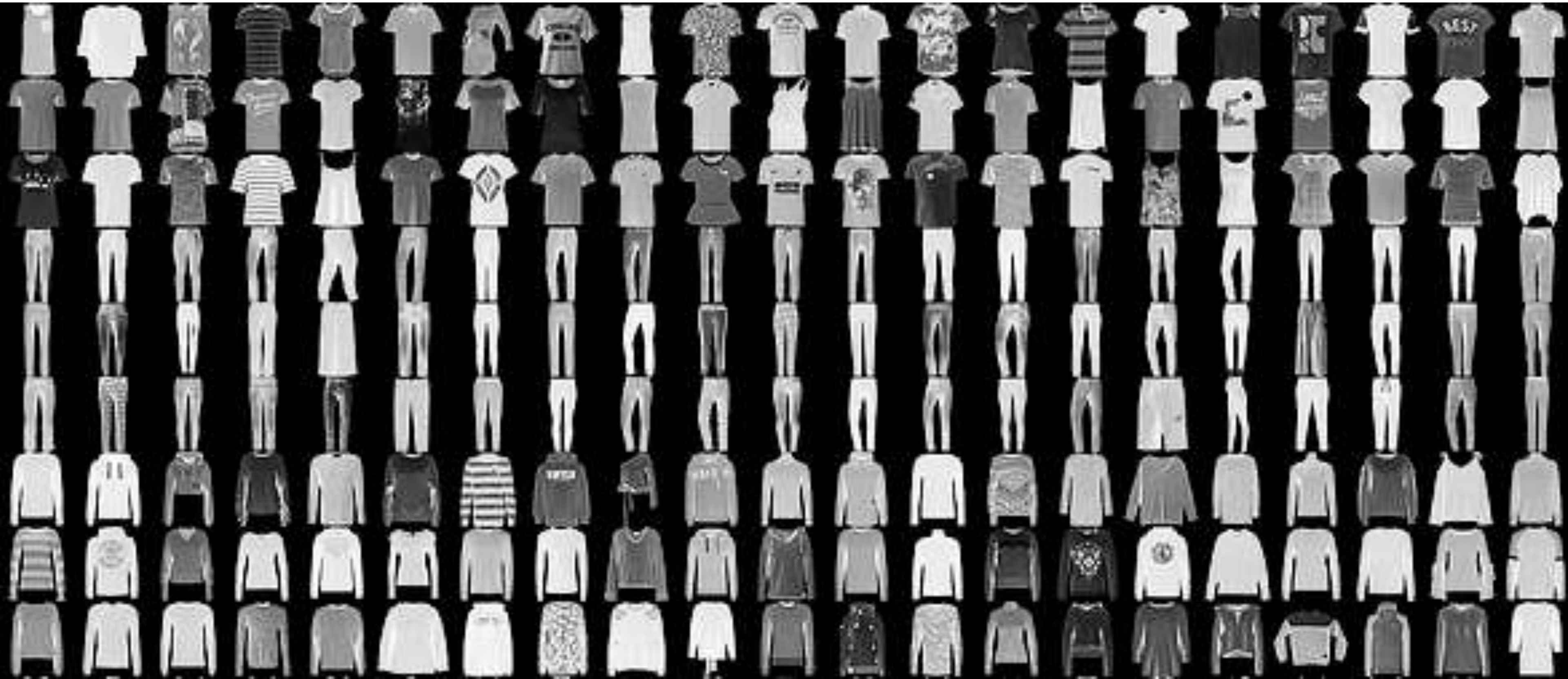
- How to acquire the needed dataset - All machine learning methods require data
  - Unsupervised ≠ no data (or label) needed
  - Unsupervised = label can be generated w/o human labor (much cheaper than supervised data)
  - Unsupervised training usually yields much lower performance, or is not possible
- What model to use
  - Many public models available
  - Usually less important than dataset - garbage in, garbage out!

The Dataset

Fashion-MNIST

Fashion-MNIST

- Like the famous MNIST, but with fashion images



Fashion-MNIST

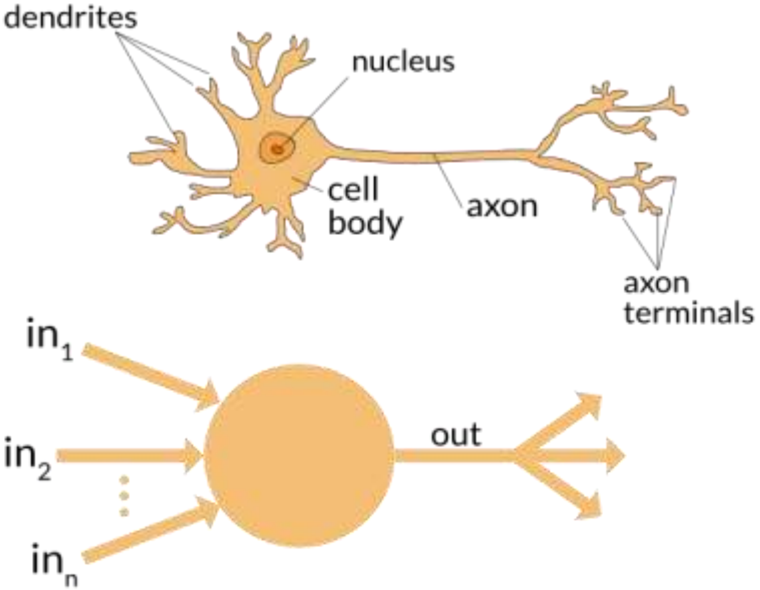
- Same number of samples (60,000 train, 10,000 test) and classes (10), same resolution grayscale images
- Task: categorize each image into one of 10 classes - T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot
- https://github.com/zalandoresearch/fashion-mnist

Perceptrons

The most basic form of ANN

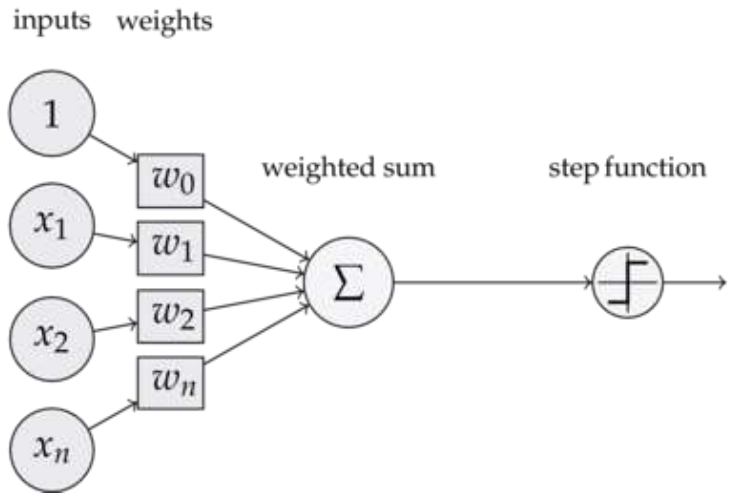
Neurons and Perceptrons

- Artificial Neural Networks(ANN) were inspired by the central nervous system of humans
- ANN's are built upon simple signal processing elements that are connected together into a large mesh

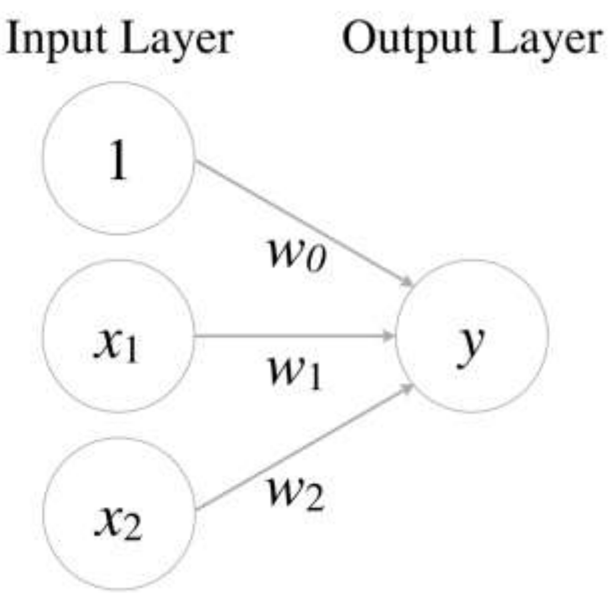


Perceptrons

- Just like a biological neuron has dendrites to receive signals, a cell body to process them, and an axon to send signals out to other neurons, the artificial neuron has a number of input channels, a processing stage, and one output that can fan out to multiple other artificial neurons

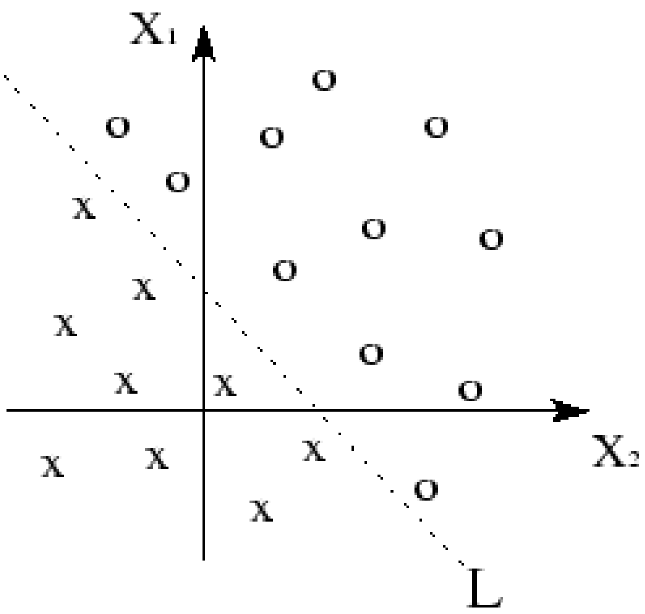


Perceptrons



- $y = x_1 \times w_1 + x_2 \times w_2 + w_0$
- $output = \begin{cases} O & if\ y > threshold \\ X & otherwise \end{cases}$
- If $threshold = 0$, the decision boundary is
- $x_1 \times w_1 + x_2 \times w_2 + w_0 = 0$
- $x_1 = -\frac{w_2}{w_1} x_2 - \frac{w_0}{w_1}$

Linearly Separable



- $y = x_1 \times w_1 + x_2 \times w_2 + w_0$
- $output = \begin{cases} O & if\ y > threshold \\ X & otherwise \end{cases}$
- If $threshold = 0$, the decision boundary is
- $x_1 \times w_1 + x_2 \times w_2 + w_0 = 0$
- $x_1 = -\frac{w_2}{w_1} x_2 - \frac{w_0}{w_1}$

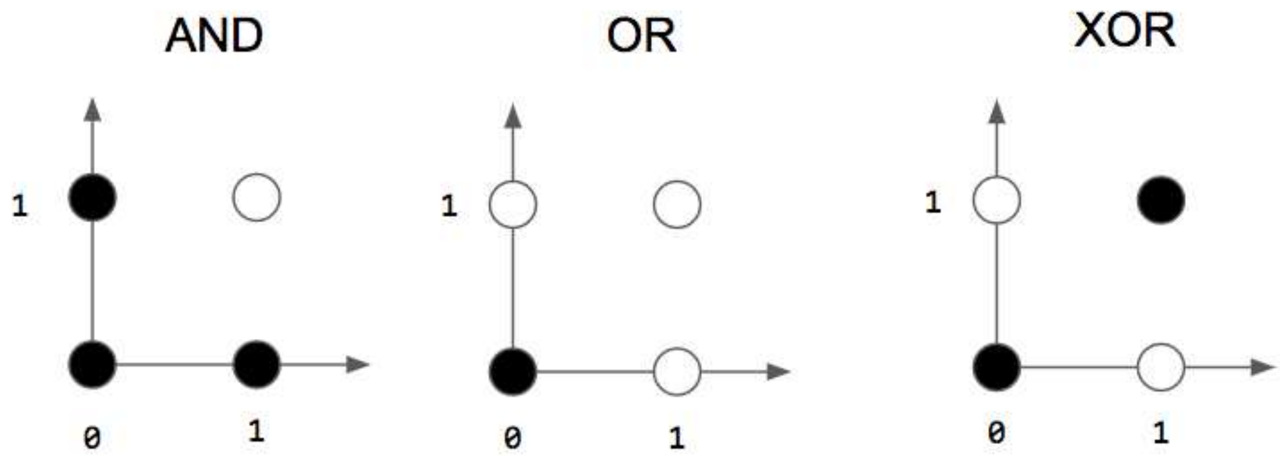
Logic Gates with a Perceptron

- Can we implement basic logic gates using perceptrons?

X1	X2	Y(AND)	Y(OR)	Y(XOR)
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

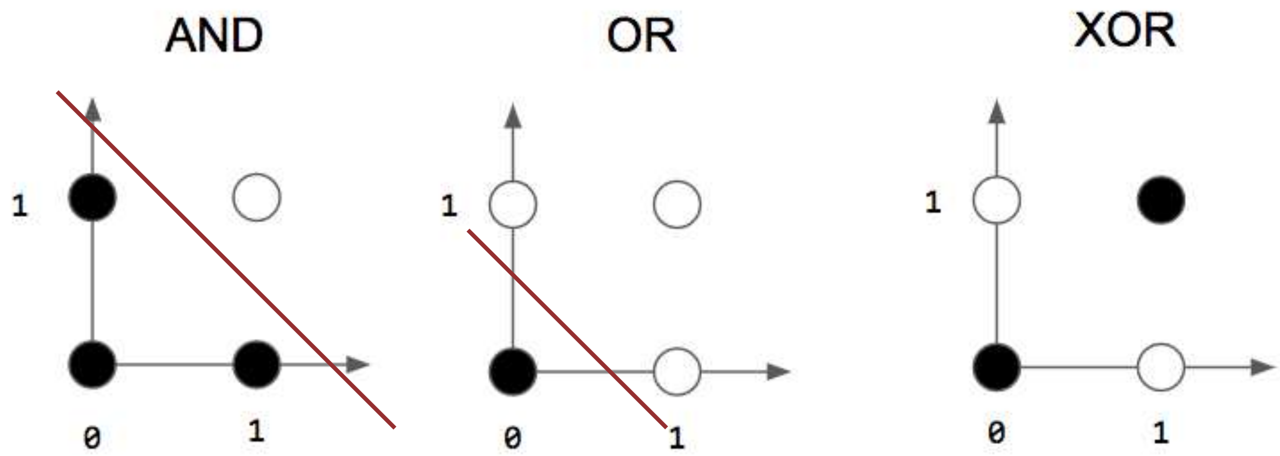
Logic Gates with a Perceptron

- Can we implement basic logic gates using perceptrons?



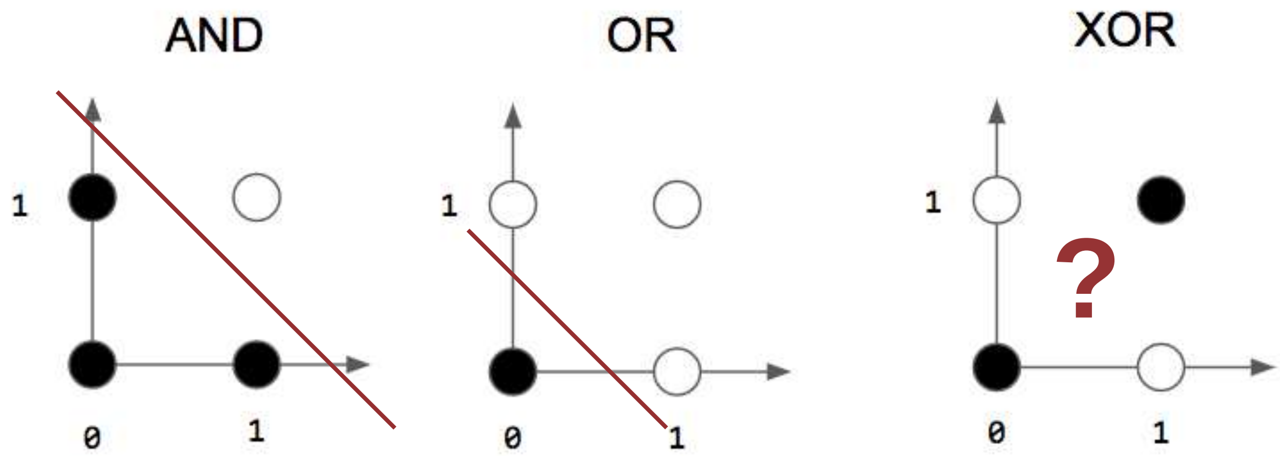
Logic Gates with a Perceptron

- Can we implement basic logic gates using perceptrons?



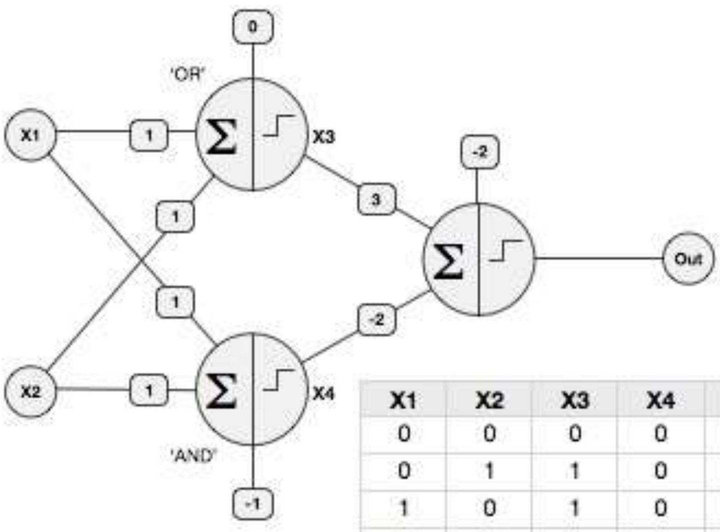
XOR Problem

- It is impossible to implement XOR gate with a perceptron



XOR Problem

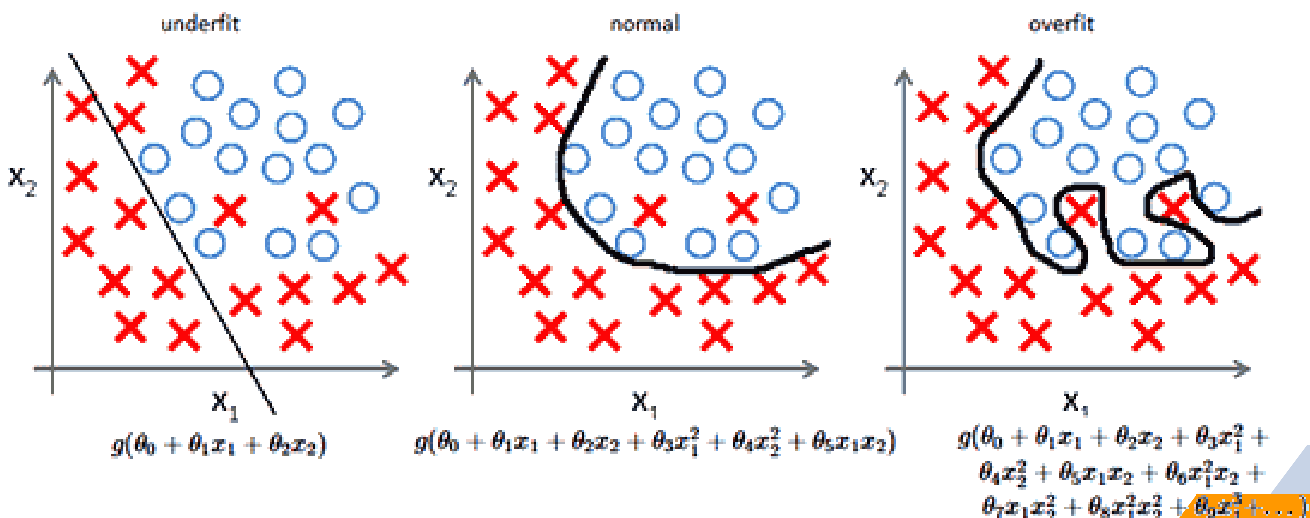
- It is possible with two layers of perceptrons
- Using another perceptron that aggregates the outputs of AND, OR gates (also implemented with perceptrons)
- Multi-layer Perceptron(MLP)



X1	X2	X3	X4	Out
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

Adding Layers to Network

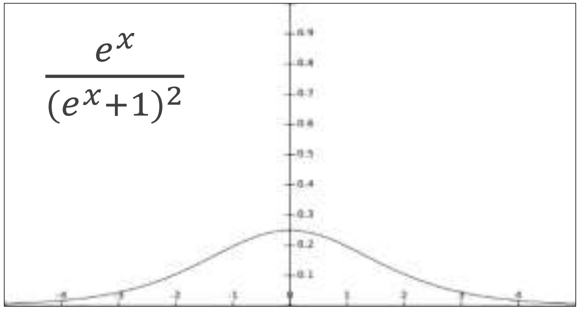
- Adding more layers to neural networks has an effect of making the decision boundary more complex
- Mis-selecting number of layers can lead to under/over fitting

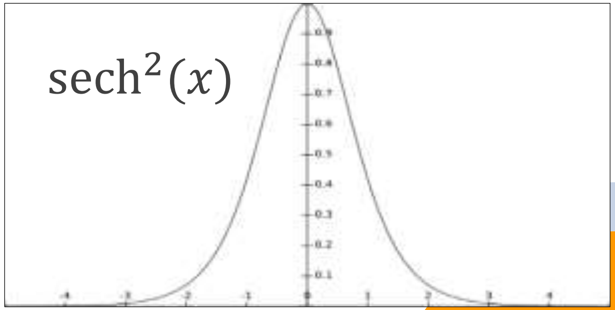


Vanishing Gradient Problem

Problem

- 주로 sigmoid function과 hyperbolic tangent function을 activation function으로 사용
  - Sigmoid function: $f(x) = \frac{1}{1+e^{-x}}, \frac{df}{dx} = \frac{e^x}{(e^x+1)^2} \Rightarrow 0 < \frac{df}{dx} \leq 0.25$
  - Hyperbolic tangent function: $f(x) = \tanh(x), \frac{df}{dx} = \mathrm{sech}^2(x) \Rightarrow 0 < \frac{df}{dx} \leq 1$
- Chain Rule을 사용하여 gradient를 계산하는 과정에서 0과 1 사이의 수를 반복하여 곱하게 됨
- Deep structure의 경우 저층 layer의 gradient가 너무 작아져서 parameter update가 매우 느려짐



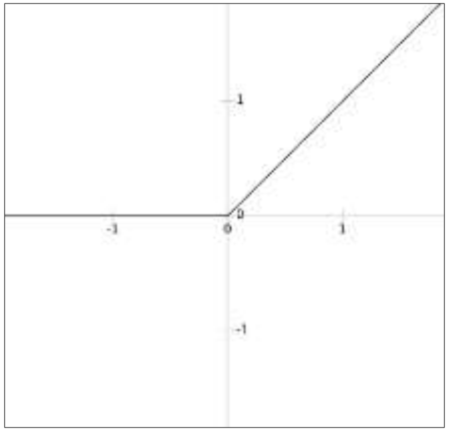


Solution

- NN의 각 layer를 독립적으로 Restricted Boltzmann Machine, Denoising Auto Encoder와 같은 unsupervised method를 사용하여 pre-training한 후, 전체 NN을 supervised method를 사용하여 훈련 (e.g. Deep Belief Network (Hinton et al., 2006))
- Vanishing/exploding gradient effect를 최소화할 수 있는 값으로 각 layer를 초기화 (e.g. Xavier initialization (Glorot et al., 2010))
- 더 빠른 하드웨어를 사용하여 NN을 많은 epoch동안 훈련
- $f'(x)$가 1인 구간이 있는 activation function 사용

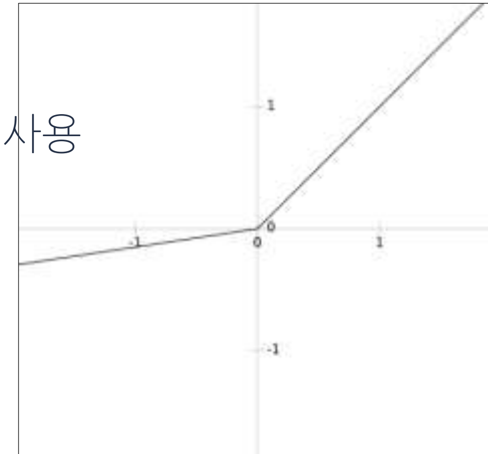
Rectified Linear Units

- Rectified Linear Unit(ReLU) (Nair et al., 2010)
  - $f(x) = \begin{cases} x & if\ x \geq 0 \\ 0 & if\ x < 0 \end{cases}$ 혹은 $f(x) = \max(0, x)$
  - $x \geq 0$인 경우 $f'(x) = 1$이므로 deep NN의 저층 layer의 gradient를 계산할 때 gradient가 소실되거나 증폭되는 문제가 완화됨
- ReLU의 단점
  - $x < 0$인 경우 x의 값과 상관 없이 $f(x) = 0$이므로 x의 값이 무시됨
  - $x < 0$인 경우 $f'(x) = 0$이므로 전체 gradient가 모두 0이 됨



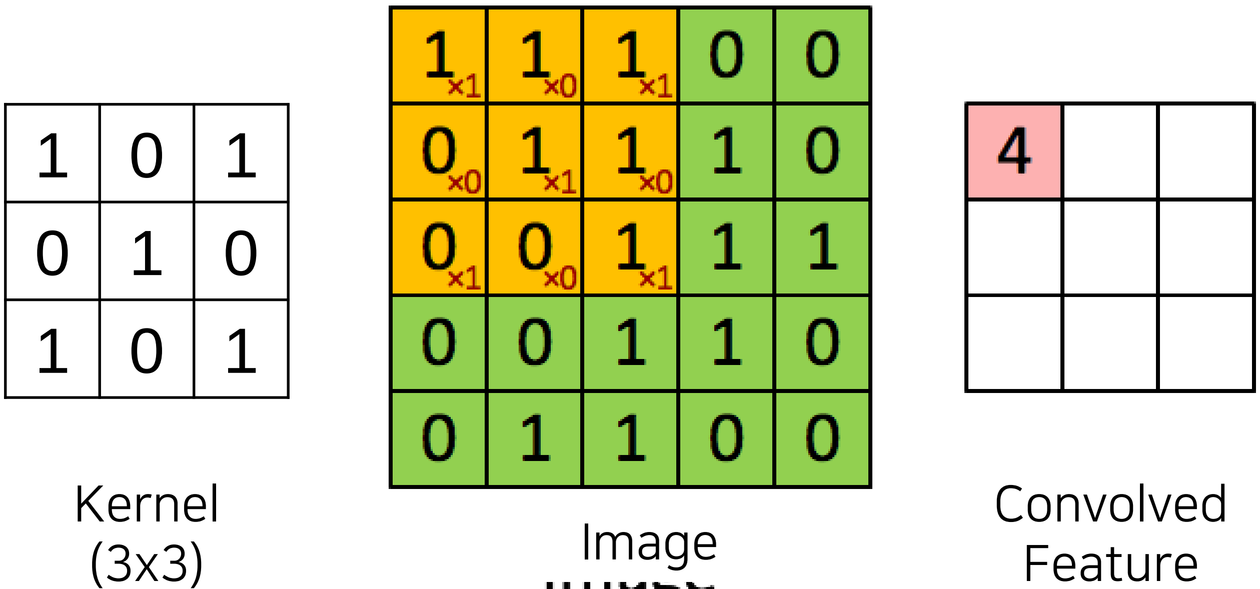
ReLU의 변형

- Leaky ReLU (Maas et al., 2013)
  - $f(x) = \begin{cases} x & if\ x \geq 0 \\ 0.01x & if\ x < 0 \end{cases}$ 혹은 $f(x) = \max(0, x) + 0.01\min(0, x)$
  - ReLU에서 $f(x) = 0$인 부분에서 문제가 발생하므로 $x < 0$인 부분에 작은 경사를 둠
- Parametric ReLU (He at al., 2015)
  - $f(x_i) = \begin{cases} x_i & if\ x \geq 0 \\ a_i x_i & if\ x < 0 \end{cases}$ 혹은 $f(x_i) = \max(0, x_i) + a_i\min(0, x_i)$
  - $x < 0$인 부분의 경사를 trainable parameter로 설정하여 training data에 맞게 activation function의 형태가 변하도록 함
  - Channel-wise: layer의 각 node별로 독립적인 a_i값 사용
  - Channel-shared: 동일한 layer 내의 모든 node가 공통의 a_i값 사용



Convolutional Neural Network Model

Convolution



- Mainly used for image processing
- Filter matrix value multiplied by the pixel value of the image.

Convolution

0	0	0	0	0
0	0	-1	0	0
0	-1	5	-1	0
0	0	-1	0	0
0	0	0	0	0

Sharpen

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

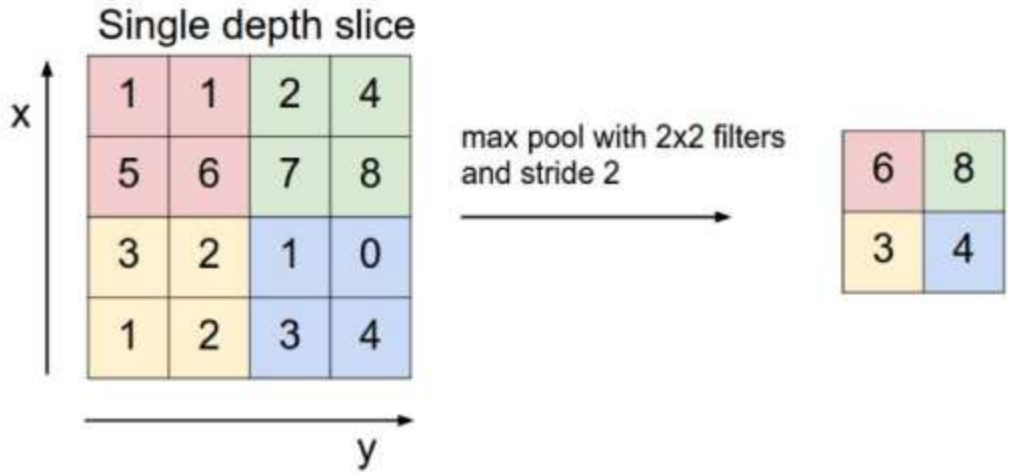
Blur

	0	1	0	
	1	-4	1	
	0	1	0	

Edge detect

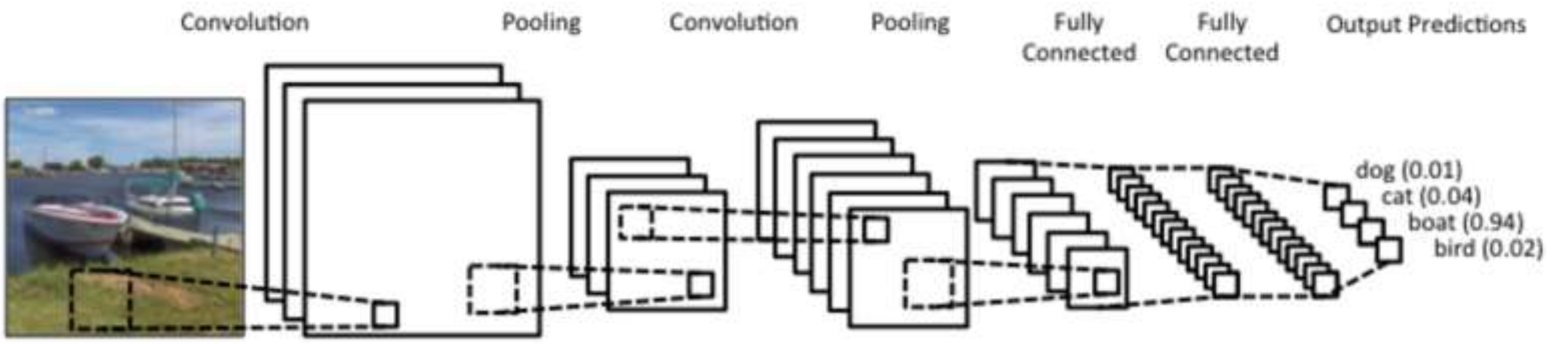
- Depending on the value of the filter, the image can have various effects.

Pooling



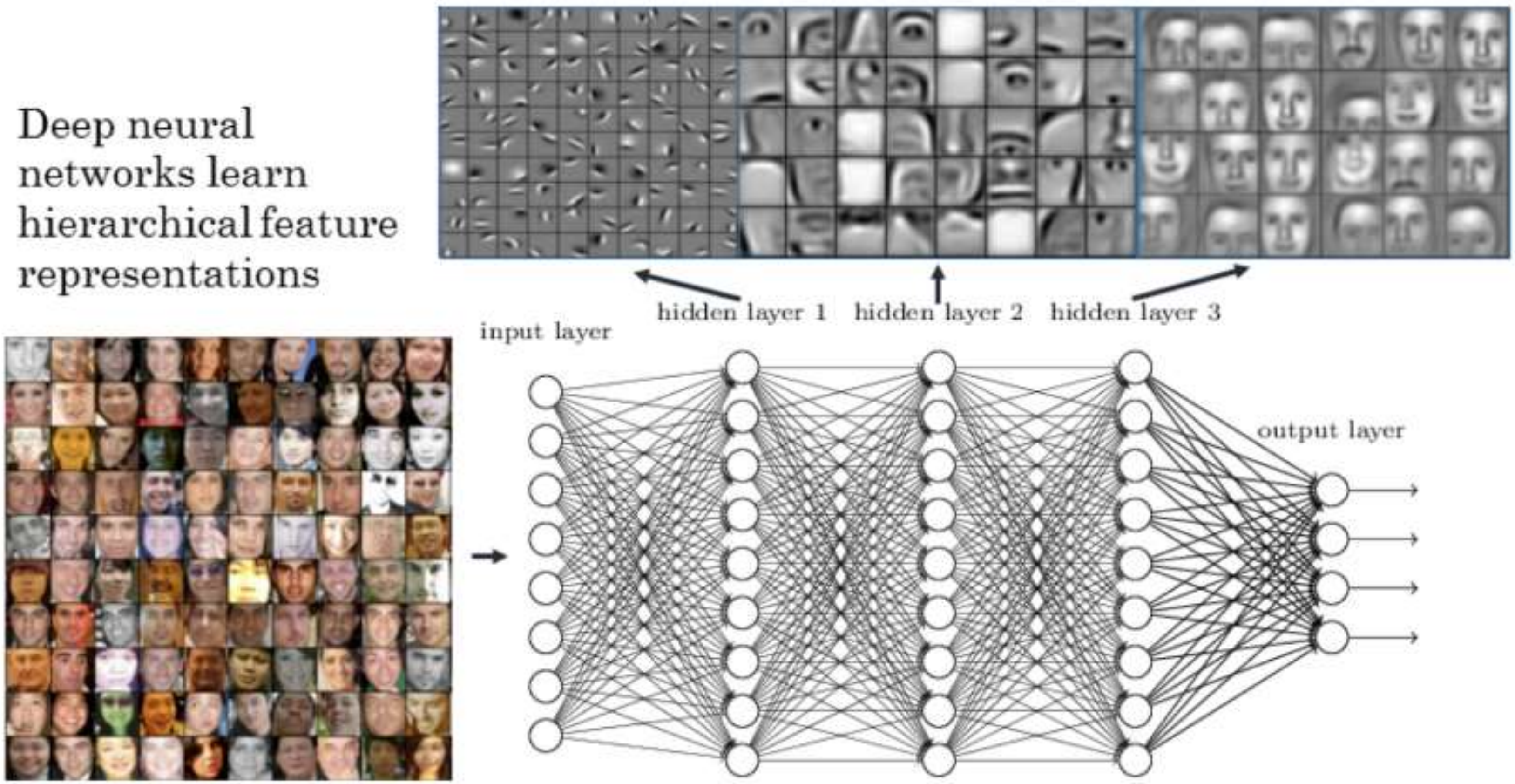
- Reduce the size of the feature
- The most commonly used max pooling preserves only the maximum value of the values in the filter.

Convolutional Neural Network(CNN)



- Extract features through convolution and pooling
- Generally hundreds to thousands of filters (kernels) are used
- It has mainly been applied to image processing, but it is also applied to NLP task recently.

Convolutional Neural Network(CNN)

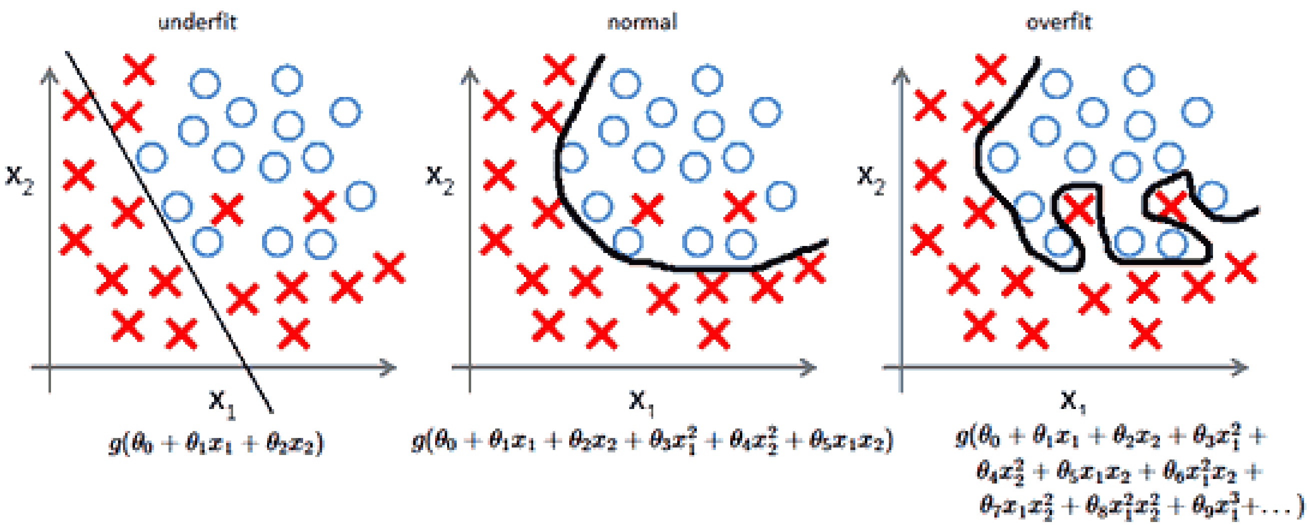


More Techniques

Early Stopping, Dropout,
Training with Momentum

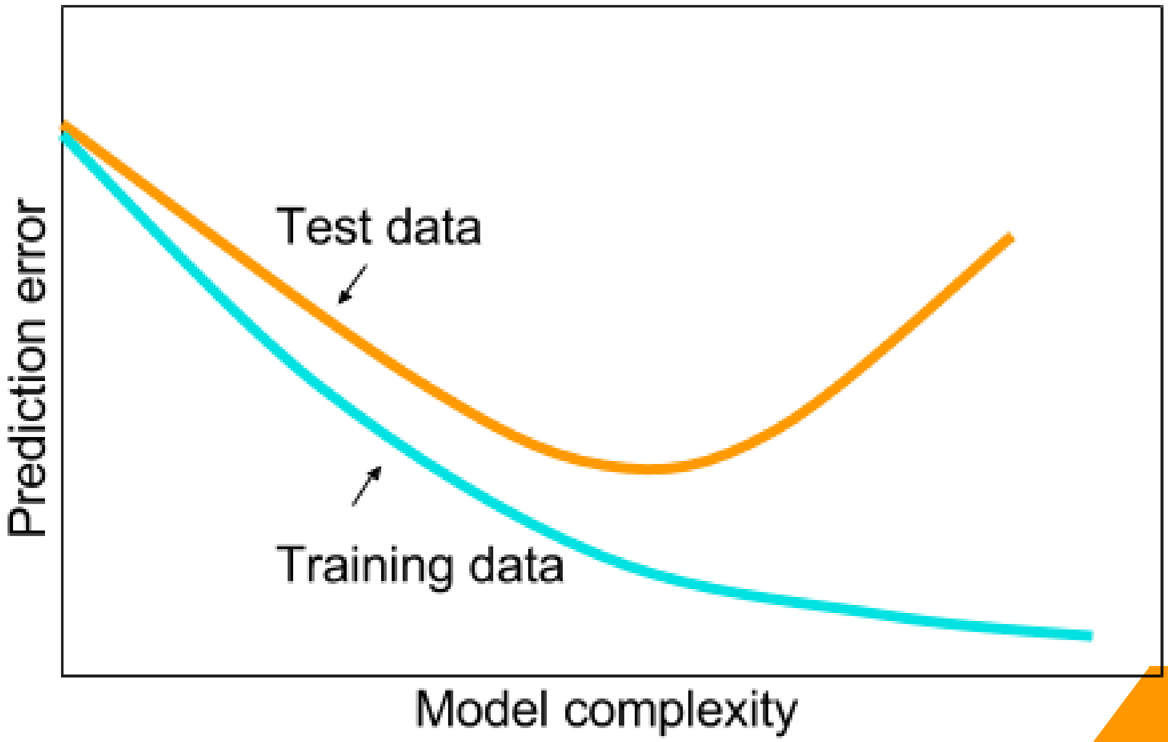
Under/Over Fitting and Model Complexity

- Underfitting - model is too simple for the dataset
- Overfitting - model is too complex for the dataset



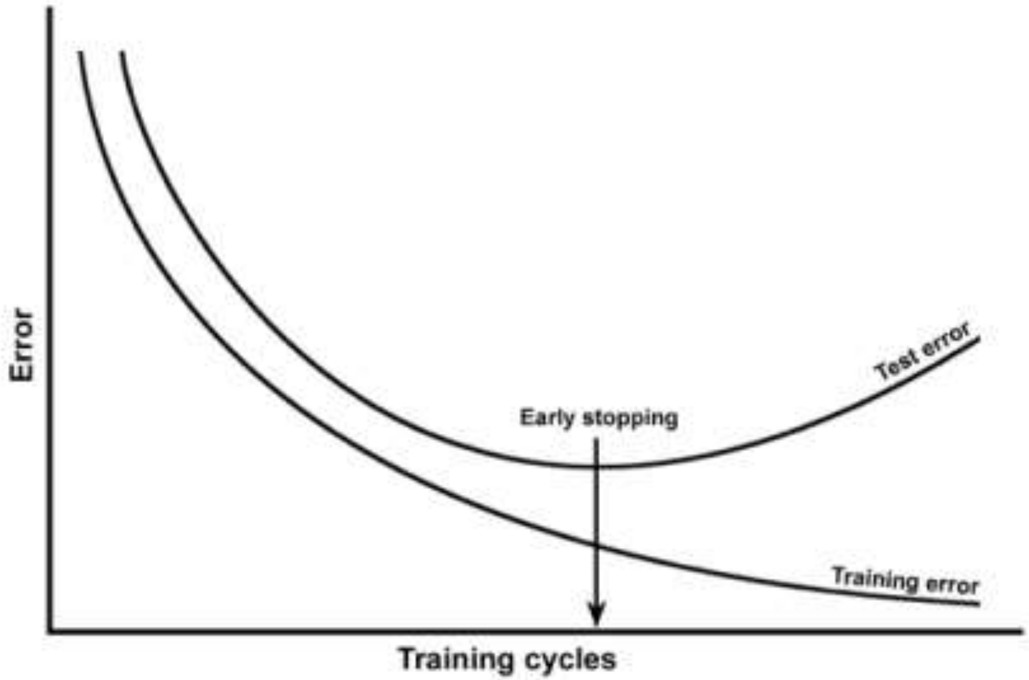
Under/Over Fitting and Model Complexity

- Underfitting - model is too simple for the dataset
- Overfitting - model is too complex for the dataset



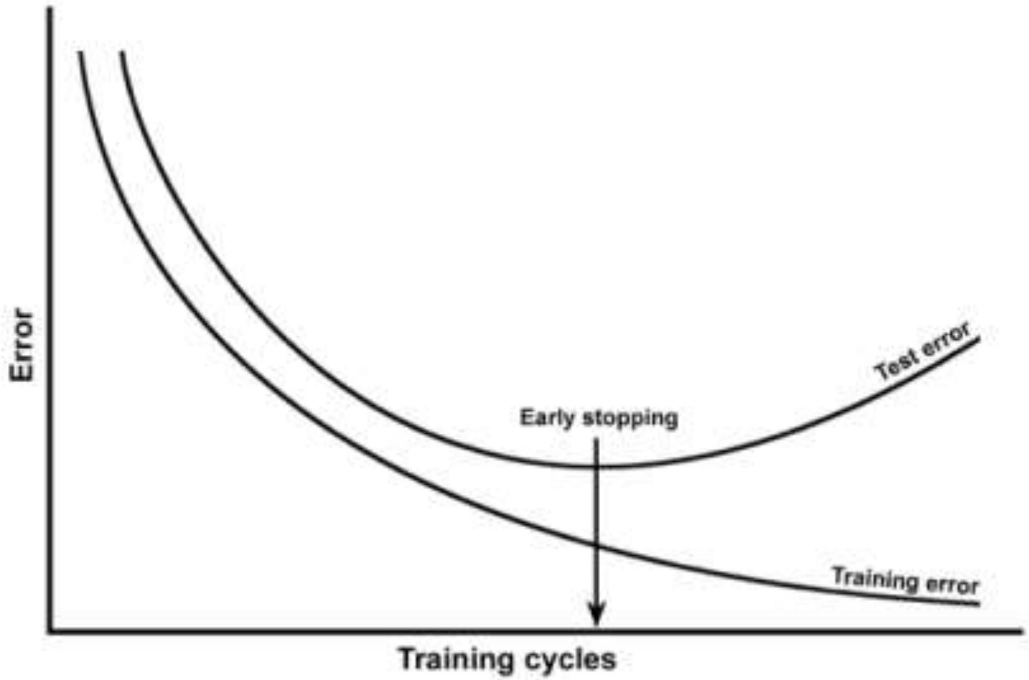
Under/Over Fitting and Training Iterations

- Underfitting - model can improve from more training iterations
- Overfitting - model is "memorizing" the dataset, instead of generalizing to it



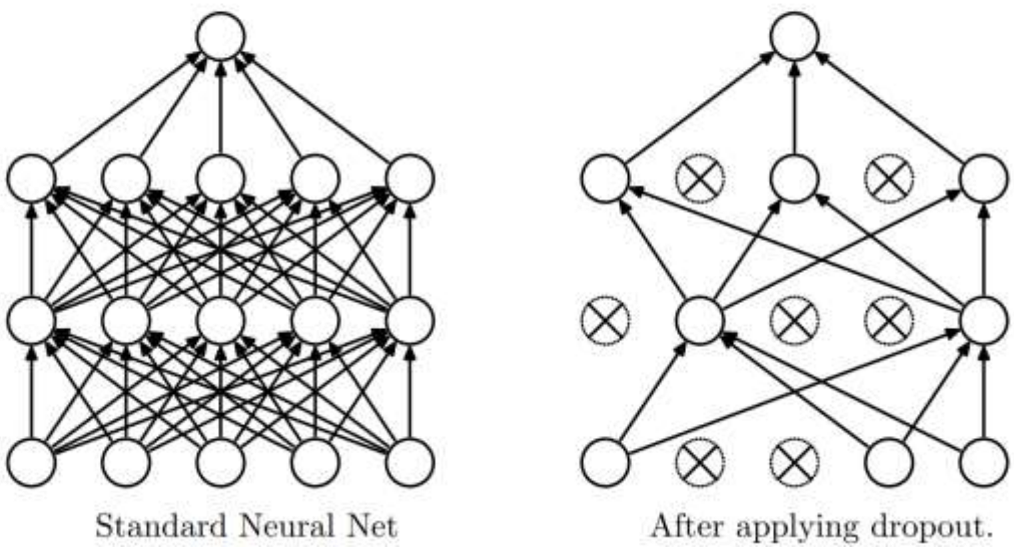
Early Stopping

- Monitor the test/validation error, and stop training when test/validation error starts to increase



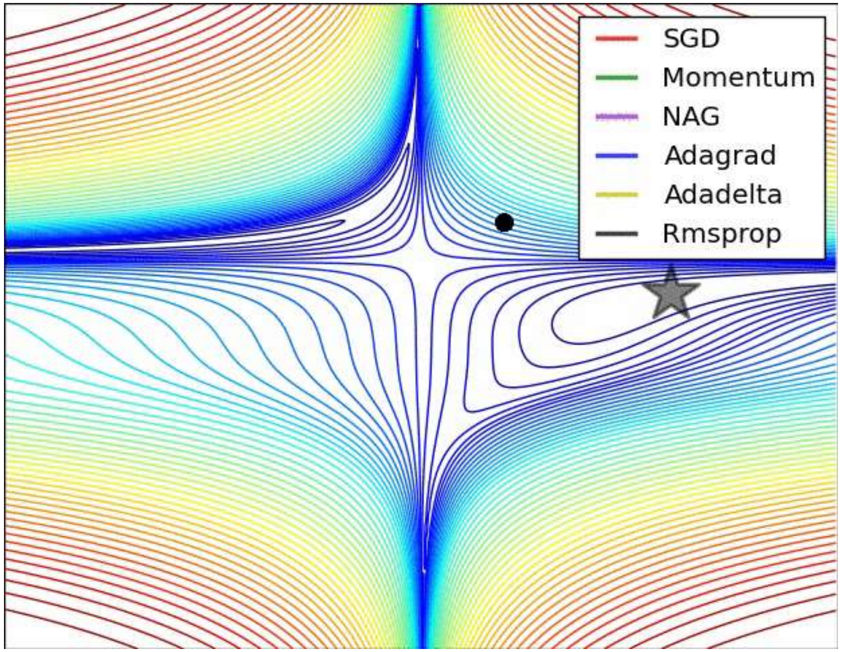
Dropout

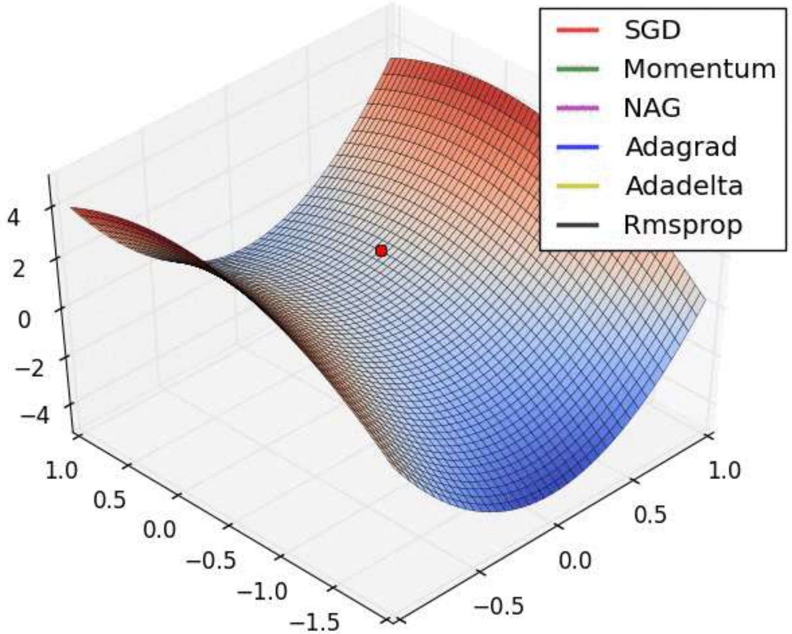
- A Simple Way to Prevent Neural Networks from Overfitting
- Randomly dropout (i.e. set to zero) nodes of a network
  - Reduces the model complexity
  - Ensemble of possible models



Training with Momentum

- Vanilla gradient descent is slow and can settle in saddle points
- Solution - gradient descent with momentum





THANKS!

Any questions?