Week 1
- Deep Learning Introduction

2019.05.11

Solaris

(http://solarisailab.com)

강의 자료 다운로드

- 강의에 사용되는 Slides & Datasets 다운로드

http://solarisailab.com/fc_dl_cv_10weeks_8th.html

- 강의에 사용되는 소스 코드 다운로드

https://github.com/solaris33/dl_cv_tensorflow_10weeks

- 실습을 위한 AWS 접속 방법 (Mac/Linux)

1. solaris.pem 다운로드
2. chmod 400 solaris.pem 명령어 입력
3. ssh -i "solaris.pem" ubuntu@ec2-13-125-64-90.ap-northeast-2.compute.amazonaws.com -X 명령어 입력

- Windows에서 접속은 Putty를 이용 (Appendix 참조)

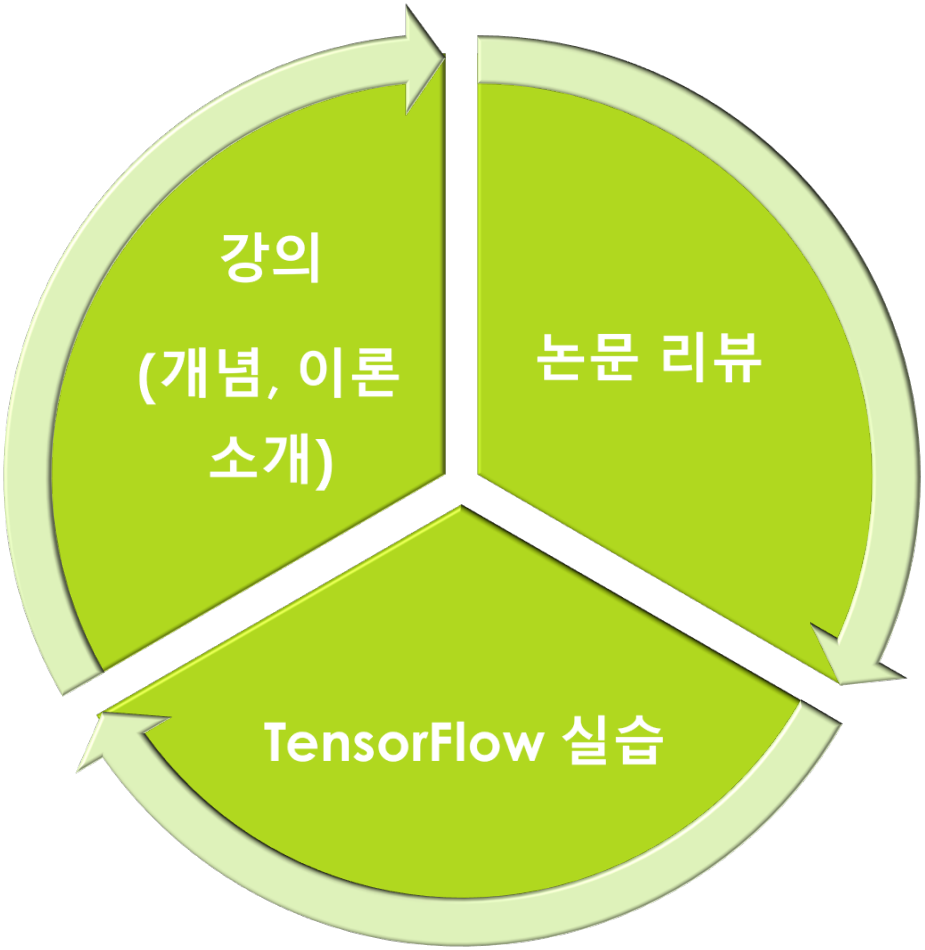
실습환경 이용 규칙

1. **mkdir <자기이름>** 명령어로 자기이름의 폴더를 생성한다.
2. **cd <자기이름>**으로 자기 이름 폴더 아래로 들어간다.
3. **git clone https://github.com/solaris33/dl_cv_tensorflow_10weeks**
   명령어로 강의에 사용되는 코드를 다운로드 받는다.
4. 코드를 실행하거나 수정해보면서 실습을 진행한다.

Outline

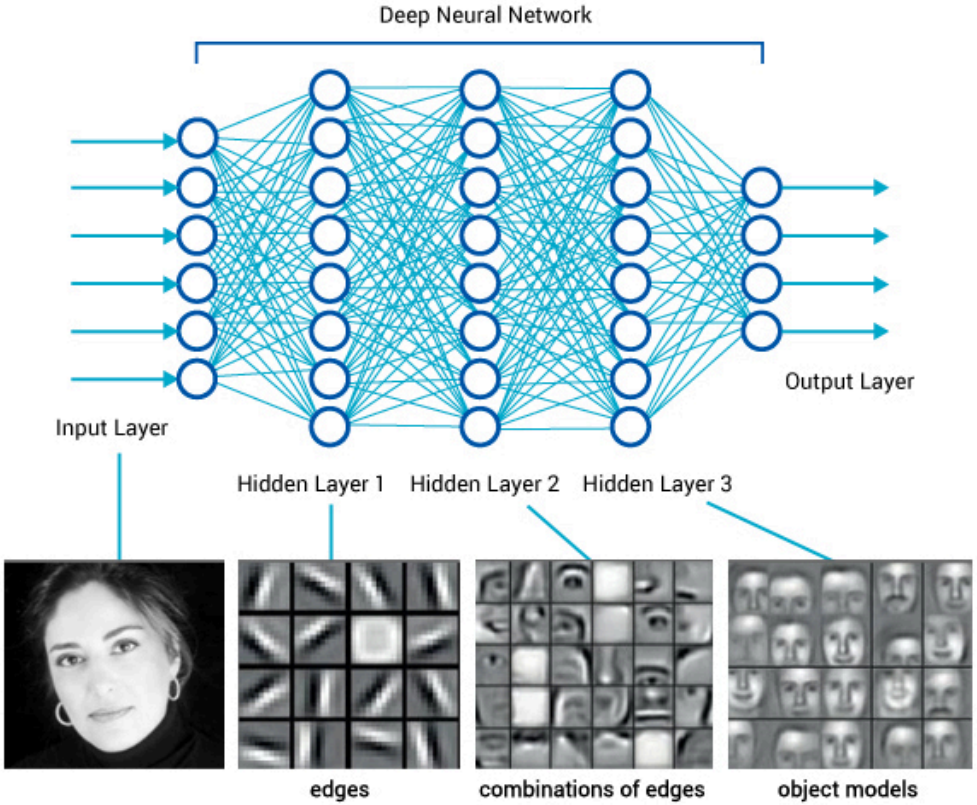
- 머신러닝 기본 이론, 간략히 살펴보는 딥러닝의 역사
- Perceptron, Artificial Neural Networks(ANNs)
- Backpropagation Algorithm 소개
- Autoencoder, Convolutional Neural Networks(CNNs), Recurrent Neural Networks(RNNs), Long-Short Term Memory(LSTM) Networks, Generative Adversarial Networks(GAN) 소개
- 효율적인 학습을 위한 테크닉들(Dropout, ReLU, Fine-Tuning-Transfer Learning-)
- 수업에서 다룰 컴퓨터비전 문제들 소개
- 논문 리뷰 – Deep Learning
- Softmax Regression 소개, Bias-Variance Trader off 소개
- TensorFlow 기초 소개
- TensorFlow 예제들 소개

강의 진행방향



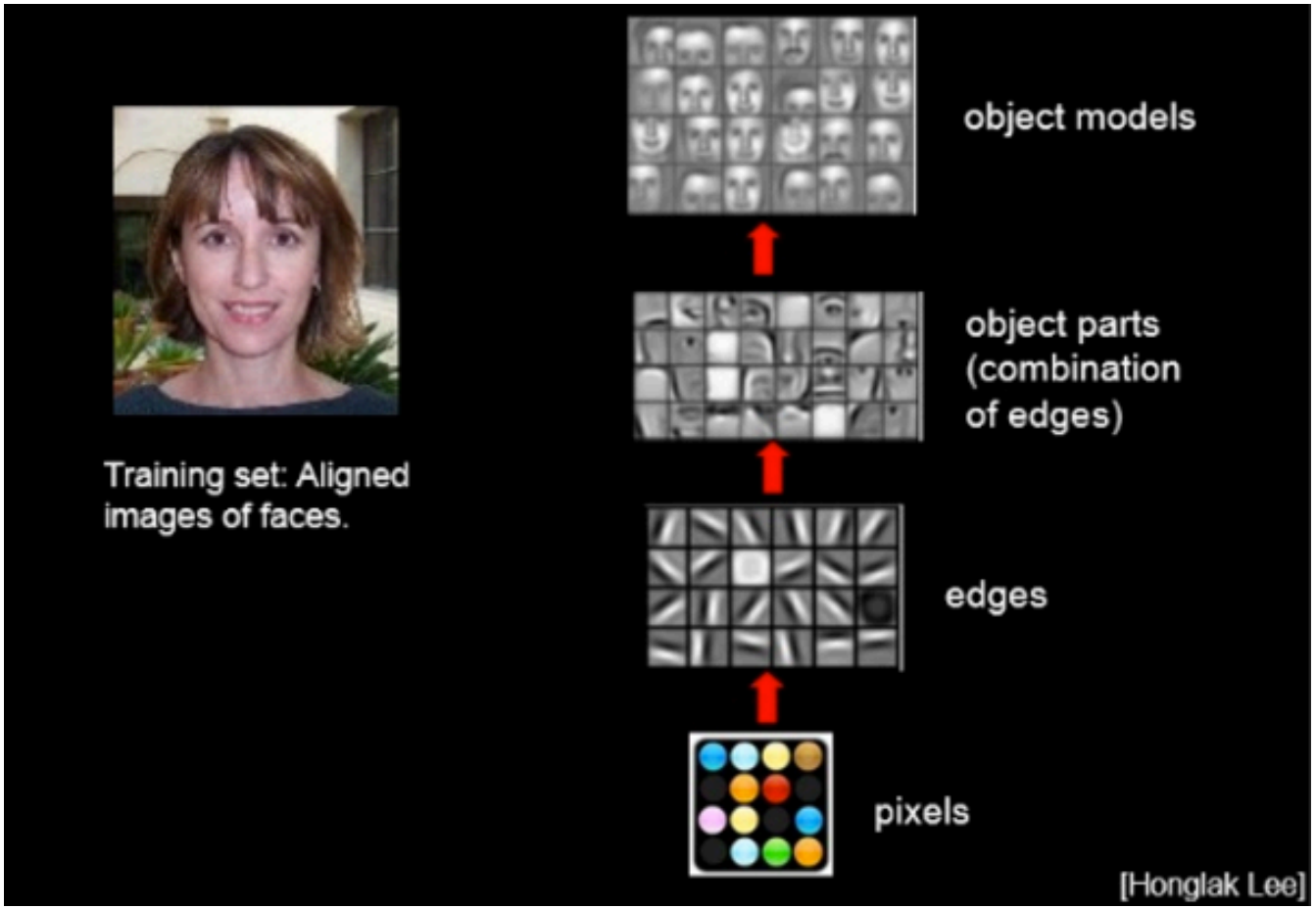
Deep Learning의 정의 1

- Deep Learning = Deep Artificial Neural Networks (깊은 인공신경망)

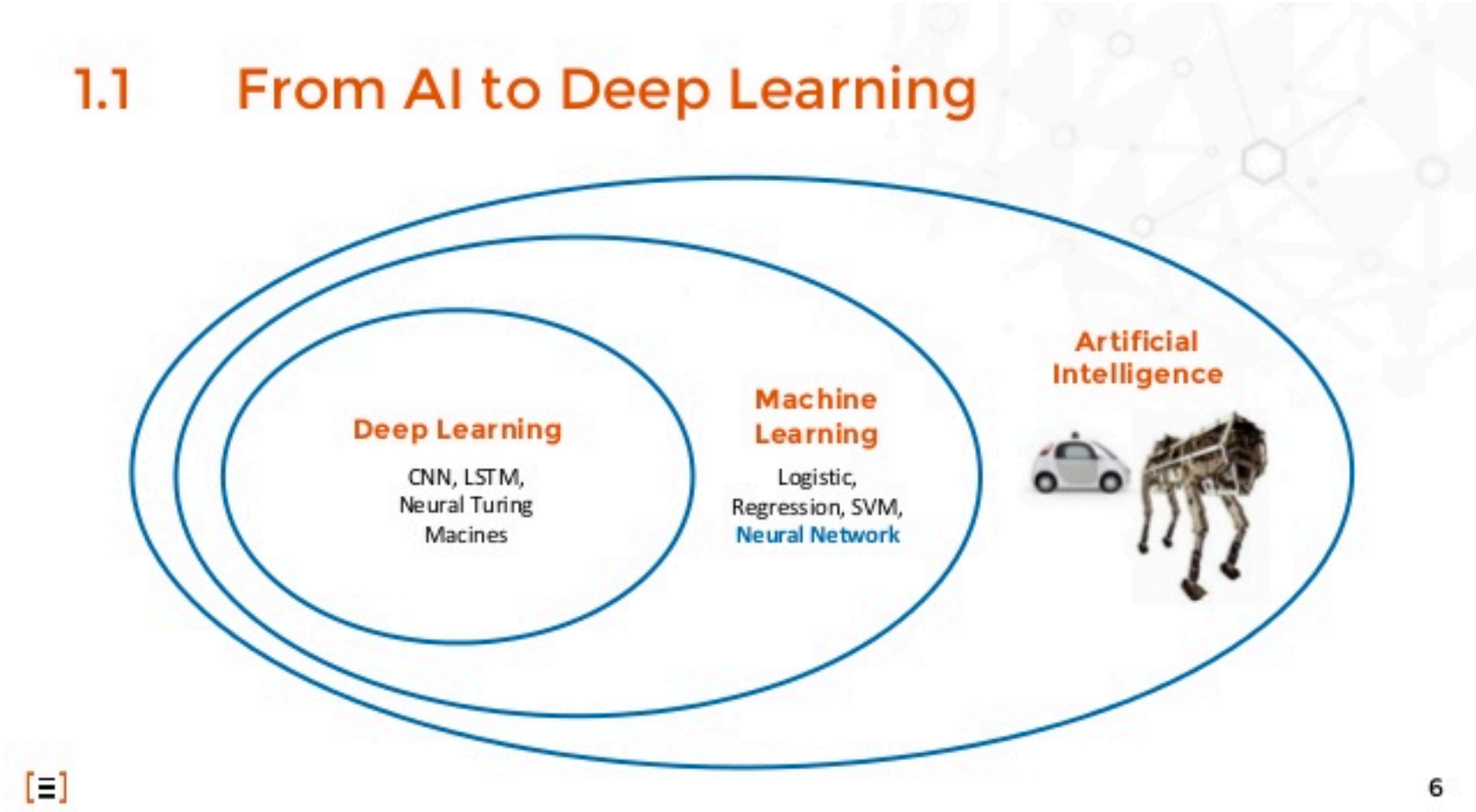


Deep Learning의 정의 2

- Deep Learning = Feature Learning (특징 학습)



AI, Machine Learning, Deep Learning



머신 러닝의 정의

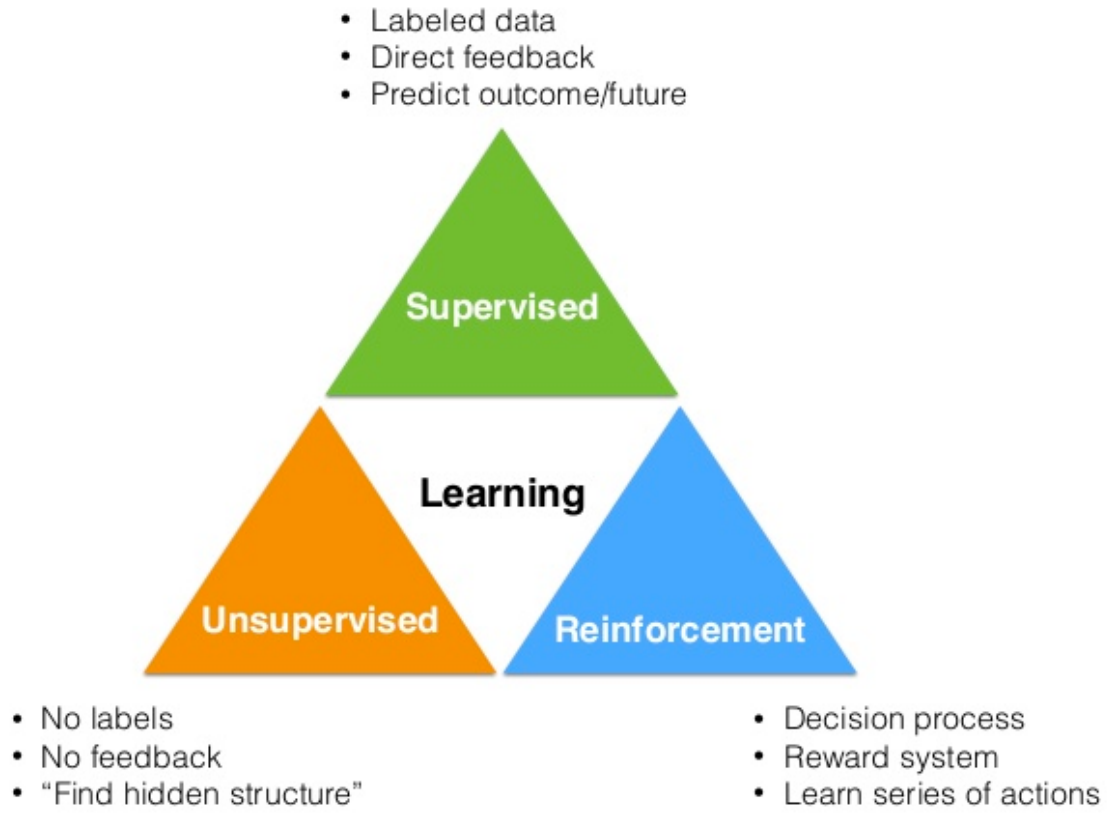
Machine Learning definition

- Arthur Samuel (1959). Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed.
- Tom Mitchell (1998) Well-posed Learning Problem: A computer program is said to *learn* from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

Andrew Ng

머신러닝(Machine Learning) 알고리즘의 세가지 분류

- 지도 학습(Supervised Learning), 비지도학습(Unsupervised Learning), 강화학습(Reinforcement Learning)

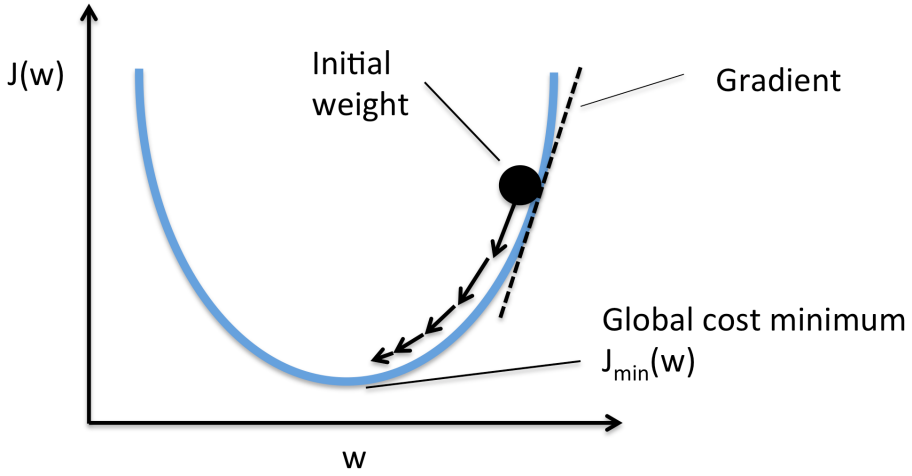


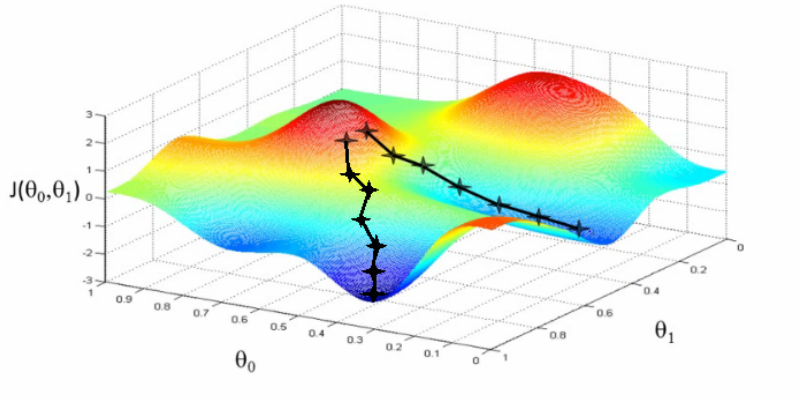
머신러닝(Machine Learning)의 기본 프로세스

1. 학습하고자하는 가설(hypothesis) θ을 수학적 표현식으로 나타낸다.
2. 가설의 성능을 측정할 수 있는 비용함수(cost function) $J(\theta)$을 정의한다.
3. cost function $J(\theta)$ 을 최소화할 수 있는 학습 알고리즘을 설계한다.

비용 함수(cost function)을 최소화하는 알고리즘 – Gradient Descent

- Gradient Descent (경사 하강법) Algorithm의 동작 과정





Gradient Descent 알고리즘의 수학적 표현

Gradient descent algorithm

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

(simultaneously update $j = 0$ and $j = 1$)

}

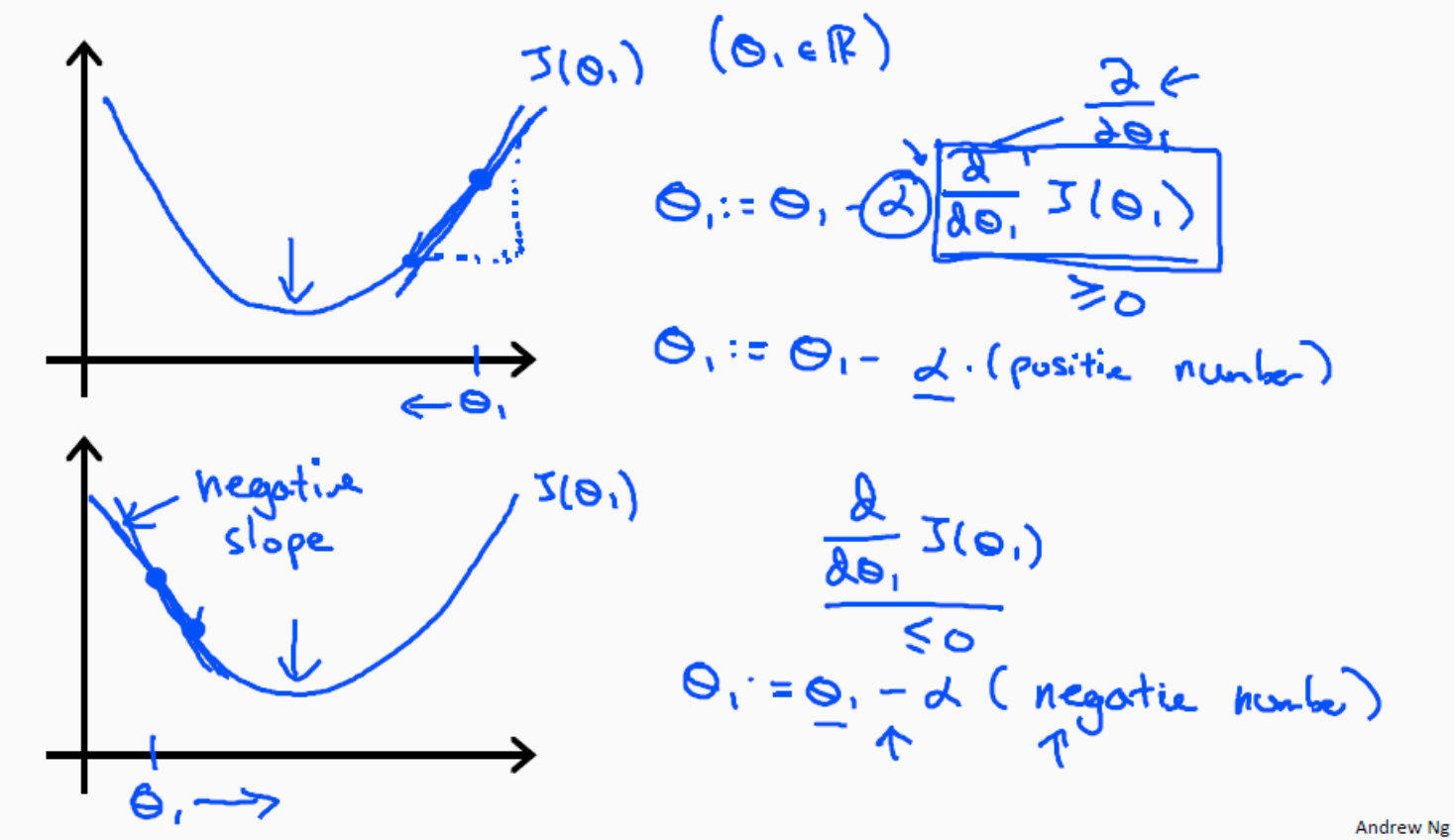
learning rate

derivative

$\min_{\theta_1} J(\theta_1)$ $\theta_1 \in \mathbb{R}$.

Andrew Ng

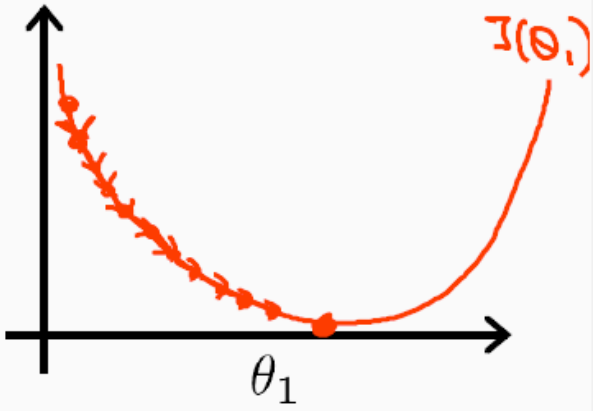
Gradient Descent 알고리즘 동작과정의 직관적 이해



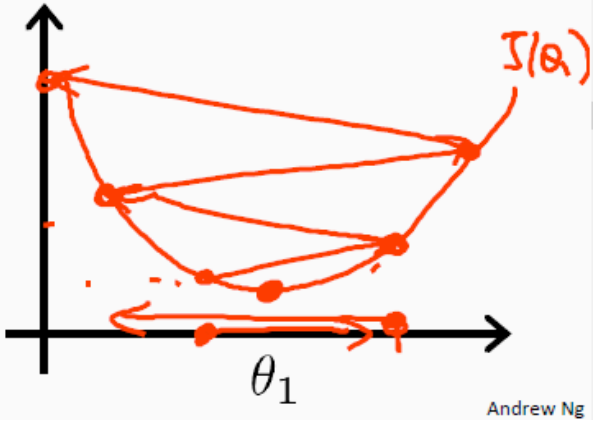
Learning Rate(α)의 영향

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

If α is too small, gradient descent can be slow.

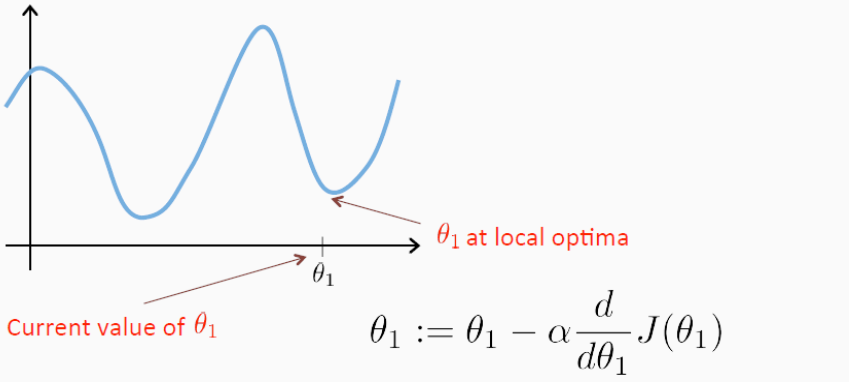


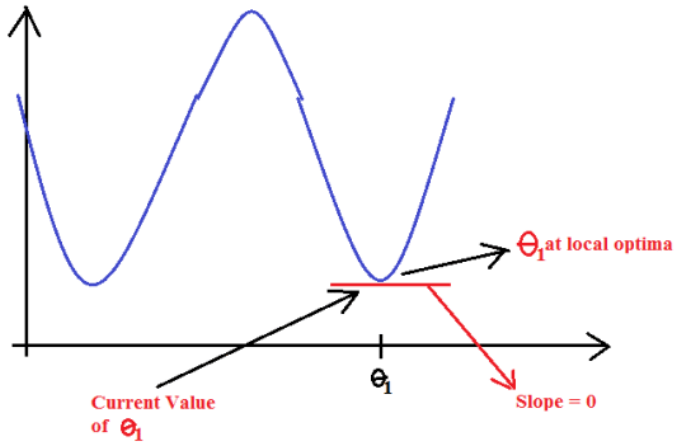
If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.



Andrew Ng

Local Optima Problem





Stochastic Gradient Descent(Mini-batch Gradient Descent) Algorithm

Mini-batch gradient descent

→ Batch gradient descent: Use all m examples in each iteration

→ Stochastic gradient descent: Use 1 example in each iteration

Mini-batch gradient descent: Use b examples in each iteration

b = mini-batch size. $b = 10$. 2-100

Get $b = 10$ examples $(x^{(i)}, y^{(i)}), \ldots (x^{(i+9)}, y^{(i+9)})$

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) \cdot x_j^{(k)} .$$

$i := i + 10$

Andrew Ng

Stochastic Gradient Descent(Mini-batch Gradient Descent) 예제

Mini-batch gradient descent

→ b examples

→ 1 example

Say $b = 10, m = 1000.$

Repeat {

Vectorization

for $i = 1, 11, 21, 31, \ldots, 991$ {

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$$

(for every $j = 0, \ldots, n$)

}

}

M = 300, 600, 000

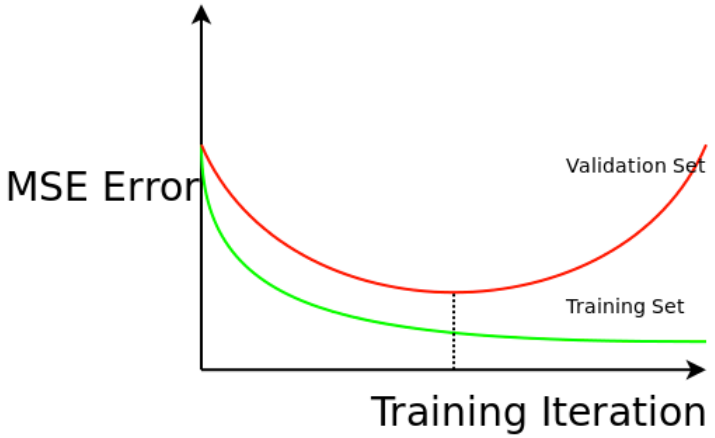
b = 10

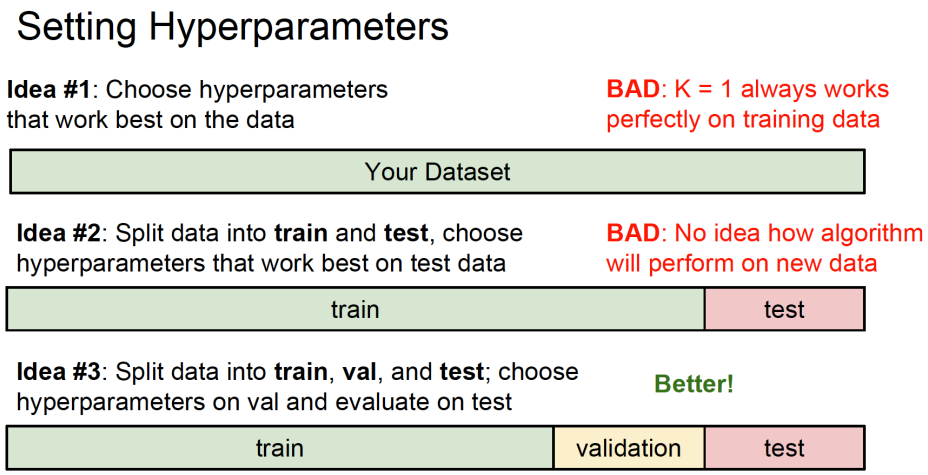
Training, Validation, Test Set

1. **Train data** : 머신러닝 알고리즘의 학습에 사용되는 데이터

2. **Validation data** : 검증용 데이터. Training error와 validation error를 이용해서 어느 시점에 멈추어야 Overfitting을 방지할수 있을지를 결정한다.

3. **Test data** : 머신러닝 알고리즘의 일반화 능력(generalization)을 측정하기 위한 데이터. 트레이닝 과정에서 한번도 보지 못한 데이터이다.

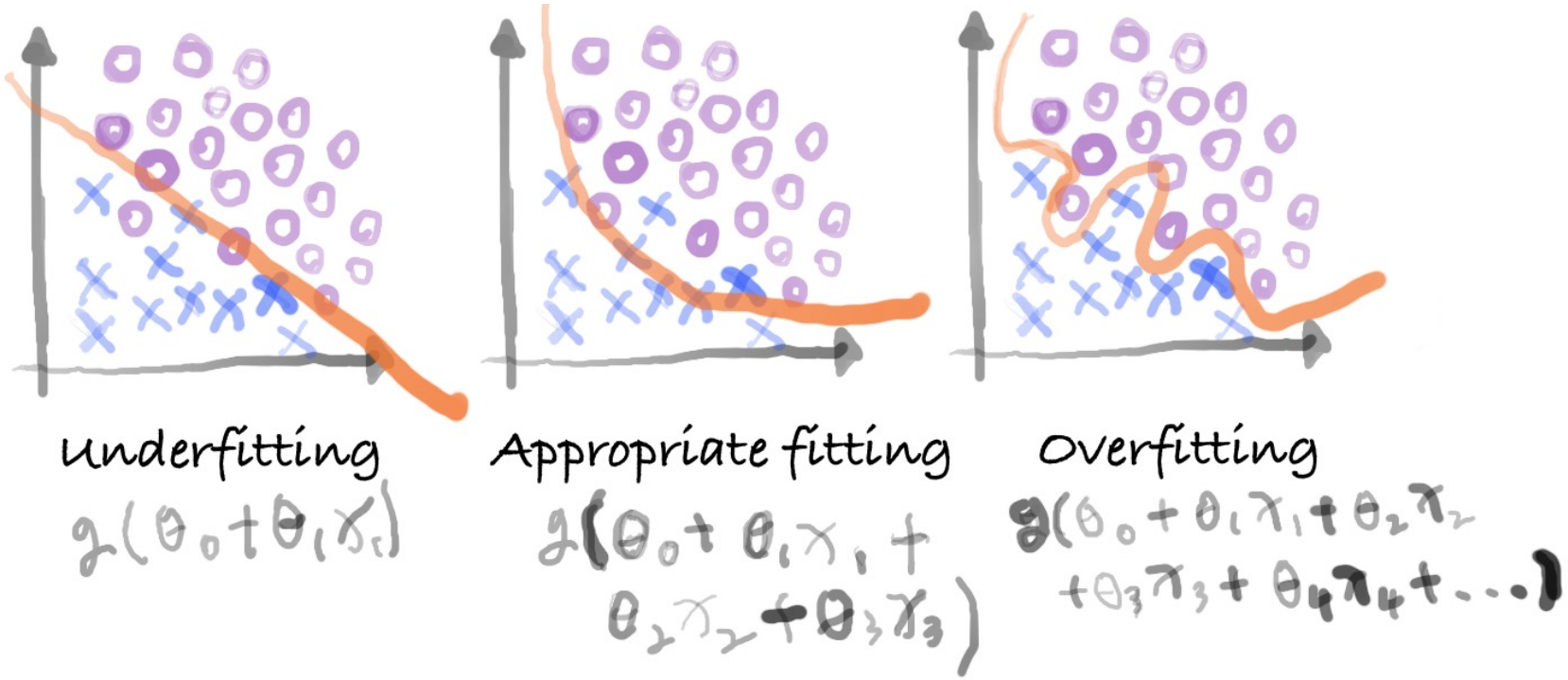




Slide credit: CS231n (https://goo.gl/QSA36t)

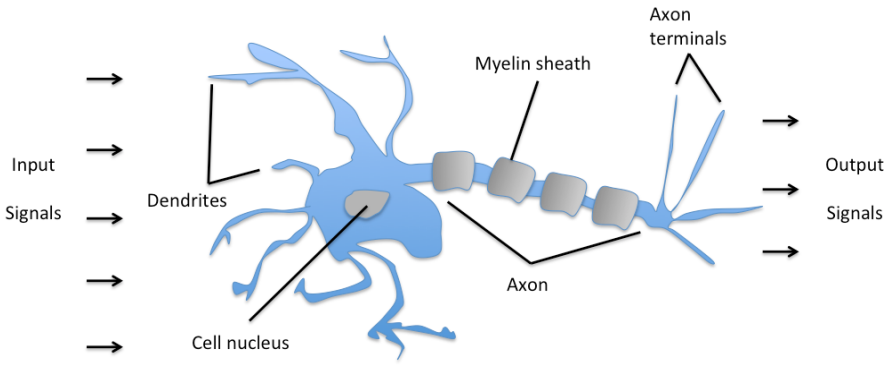
Overfitting

- **Overfitting** : 학습 결과가 Training Data에만 최적화 되어 있어서 새로운 데이터(Test Data)에 대해서는 성능이 떨어지는 상태. 학습의 일반화(Generalization)가 부족한 상태이다.



간략히 살펴보는 딥러닝의 역사 (1세대 : 1943~1986년)

- 1943년에 McCulloch, Warren S., and Walter Pitts가 인공신경망의 아이디어를 제안



Schematic of a biological neuron.

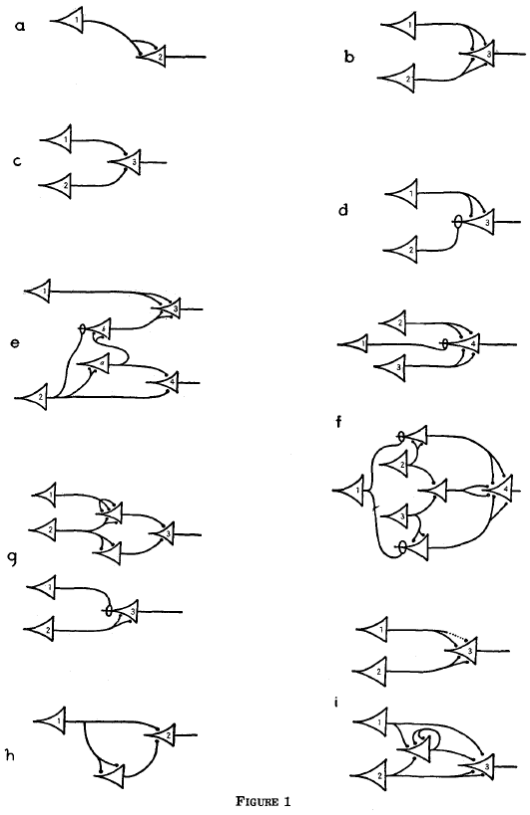
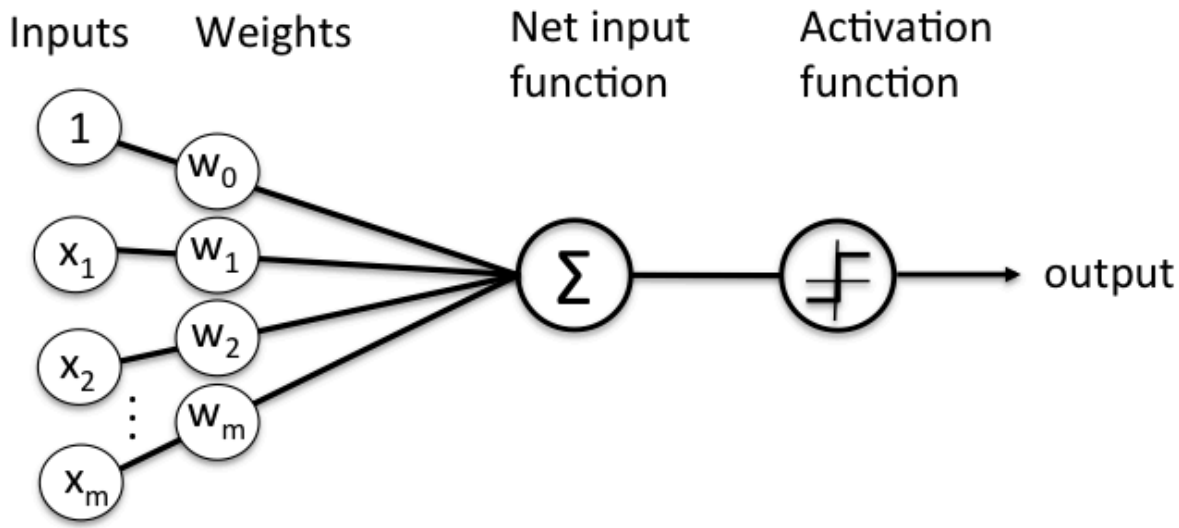


FIGURE 1

간략히 살펴보는 딥러닝의 역사 (1세대 : 1943~1986년)

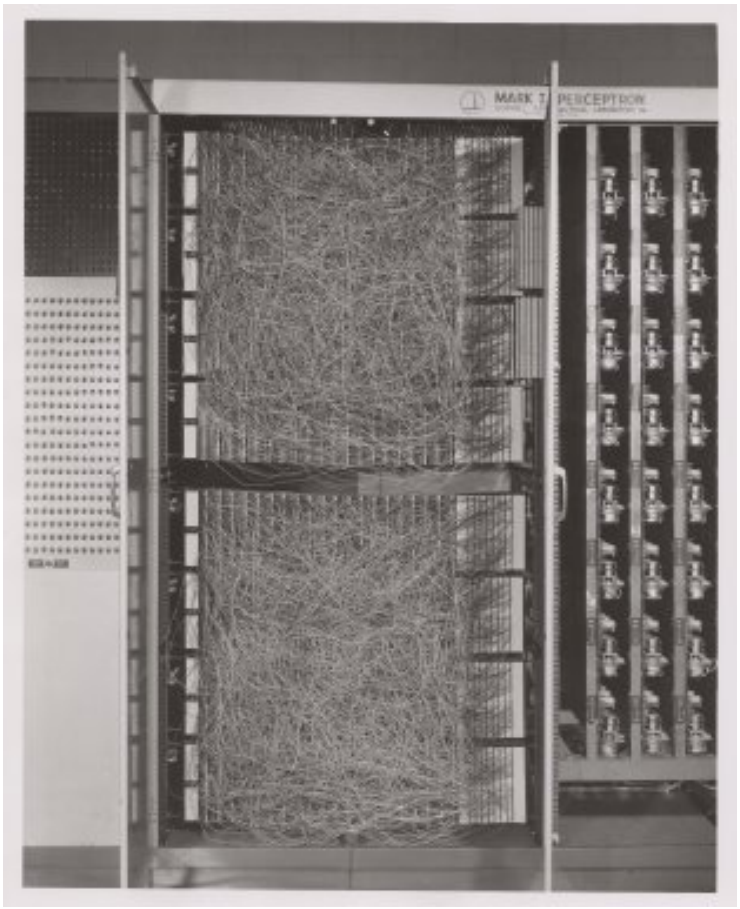
- 1958년 Frank Rosenblatt이 퍼셉트론 모형을 제안



Schematic of Rosenblatt's perceptron.

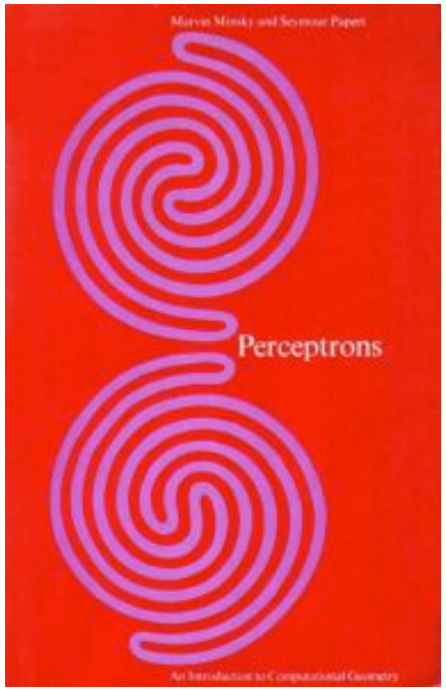
간략히 살펴보는 딥러닝의 역사 (1세대 : 1943~1986년)

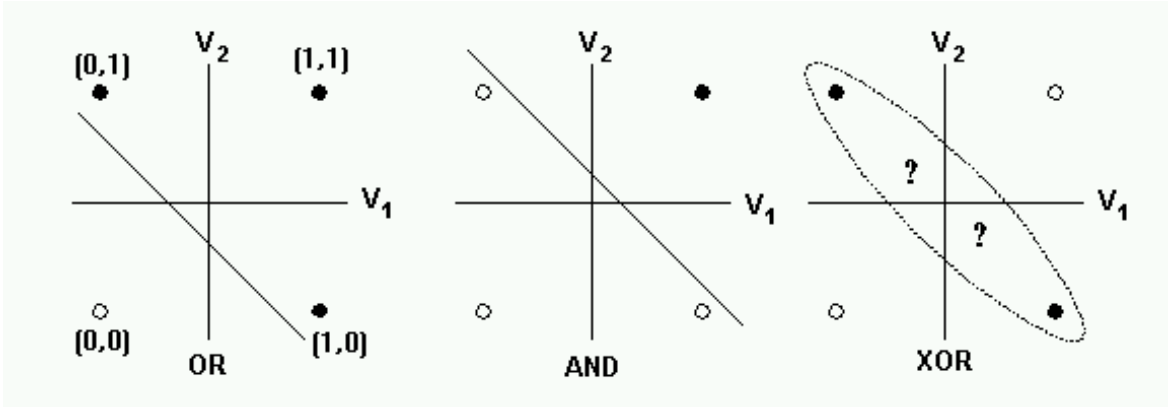
- 1958년 Frank Rosenblatt이 퍼셉트론 모형을 제안



간략히 살펴보는 딥러닝의 역사 (1세대 : 1943~1986년)

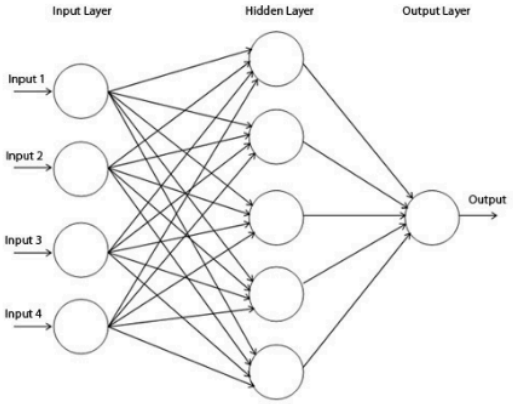
- 1969년 Marvin Minsky와 Seymour Papert가 Perceptron 모델의 한계를 지적(선형 분리 불가능)하면서 퍼셉트론의 기대는 사그라들고 인공지능 연구자들은 다른 방법을 탐구 함

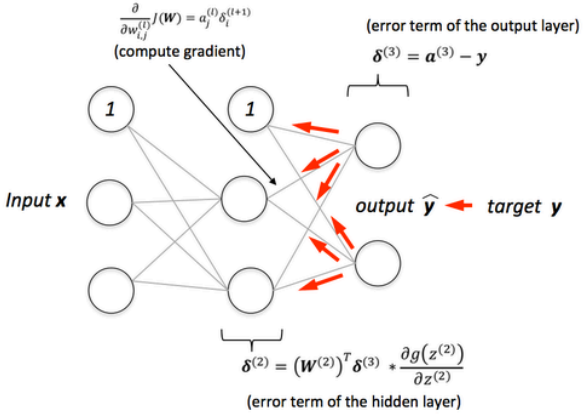


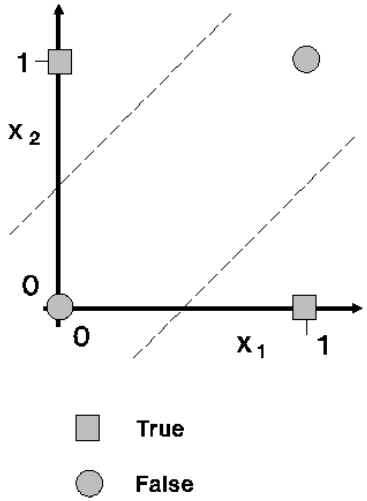


간략히 살펴보는 딥러닝의 역사 (2세대 : 1986~2006년)

- 1986년 Multi-Layer Perceptron(MLP)과 Backpropagation Algorithm이 제안되고 이를 통해 통해 XOR 문제를 해결할 수 있게 됨. (선형분리가능)

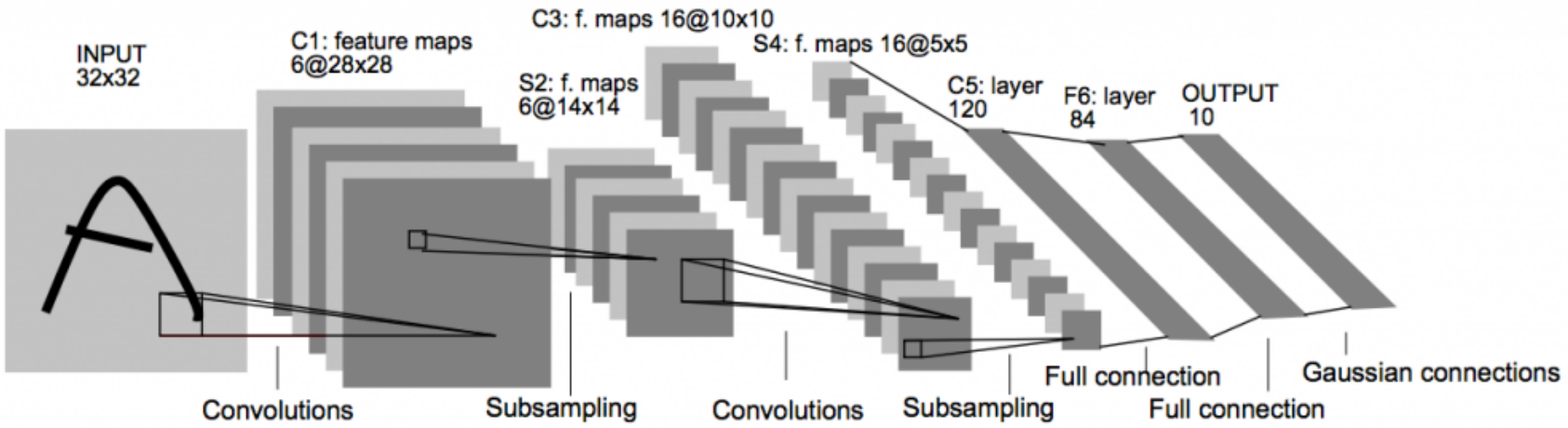






간략히 살펴보는 딥러닝의 역사 (2세대 : 1986~2006년)

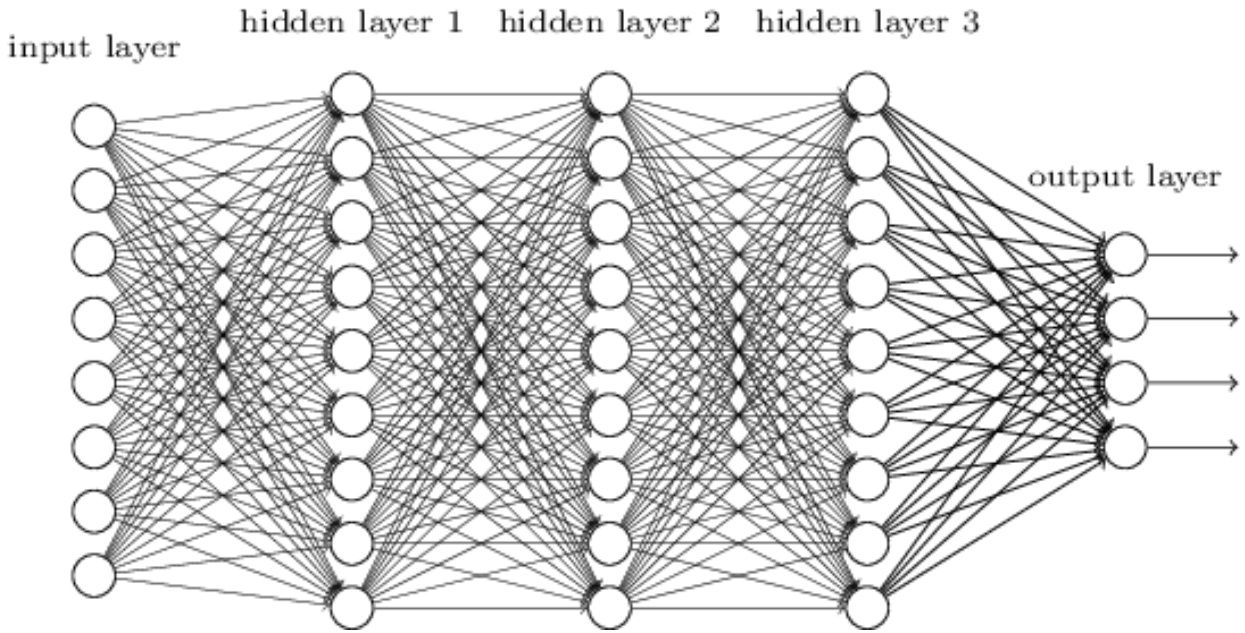
- 1998년에 Yann Lecun에 의해 제안된 Convolutional Neural Networks(CNNs) 구조인 LeNet은 수표 인식 문제 등에 실용적으로 사용 됨



간략히 살펴보는 딥러닝의 역사 (2세대 : 1986~2006년)

- 하지만 Layer를 깊게 쌓을 수록 Gradient가 사라져서 학습이 잘 되지 않는 Vanishing Gradient Problem이 발생.

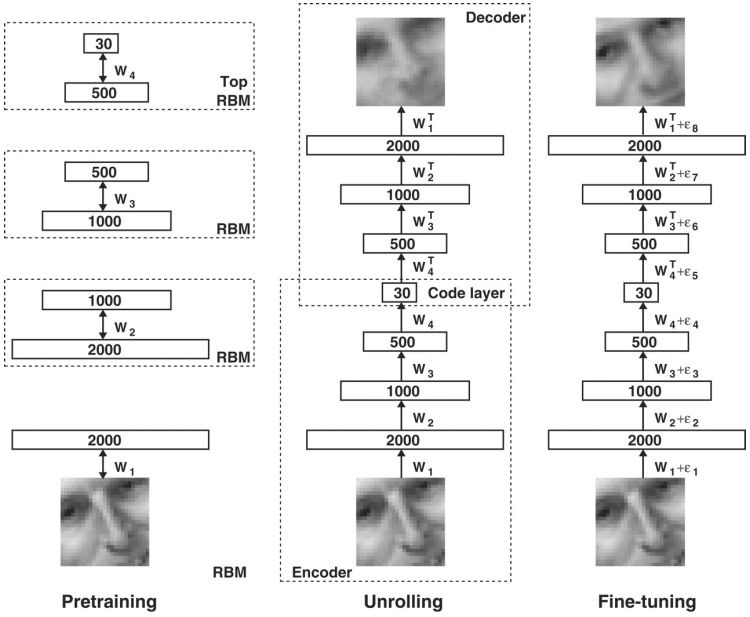
- 이로 인해 Deep Learning은 불가능했고 인공지능 연구자들은 SVM과 같은 다른 기법들을 탐구하기 시작함.



간략히 살펴보는 딥러닝의 역사 (3세대 : 2006~2012년)

- ANNs에 대한 인공지능 연구자들의 관심이 사그라들었지만 Geoffrey Hinton과 딥러닝 선구자들은 묵묵히 연구를 지속함.

- Hinton은 Pre-training을 이용해 학습한 Deep Autoencoders에 관한 논문을 2006년에 Science지에 발표.



간략히 살펴보는 딥러닝의 역사 (3세대 : 2006~2012년)

1. Hinton의 Science 논문을 시발점으로 RBM, Pre-training, Dropout, ReLU 등 Overfitting을 방지하기 위한 새로운 알고리즘들의 등장

2. 인터넷의 발전으로 인한 학습에 사용할 빅데이터를 손쉽게 구할수 있는 환경의 도래

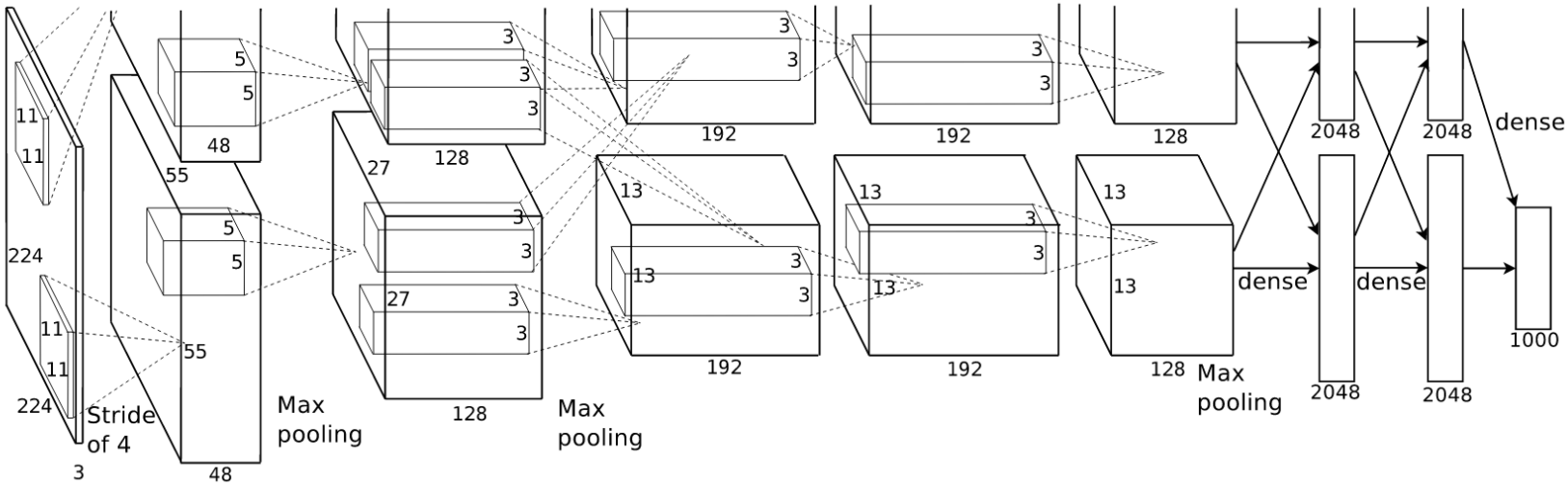
3. 빠르게 대용량 연산을 처리할 수 있는 GPU 하드웨어의 발전

위 세가지 조합에 힘입어 딥러닝 기법이 다시 부흥기를 맞게 됨

간략히 살펴보는 딥러닝의 역사 (3세대 : 2006~2012년)

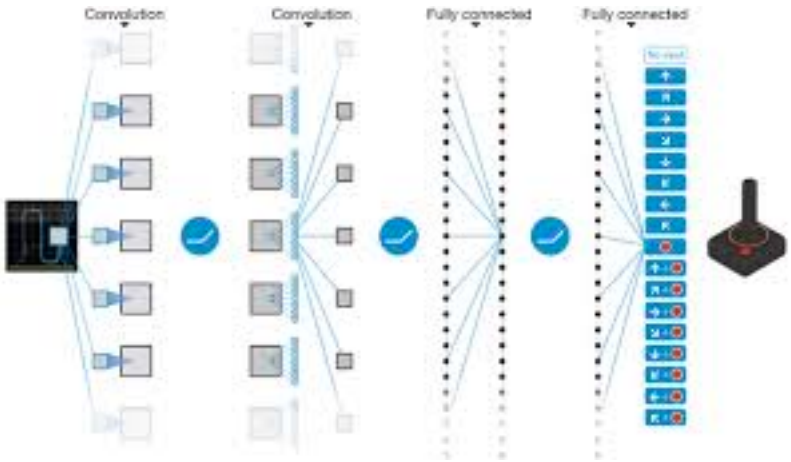
- 2012년 Hinton이 제자인 Alex krizhevsky와 함께 제안한 AlexNet은 ILSVRC-2012 대회에서 다른 기법들을 압도적 격차로 이기고 우승을 차지함.

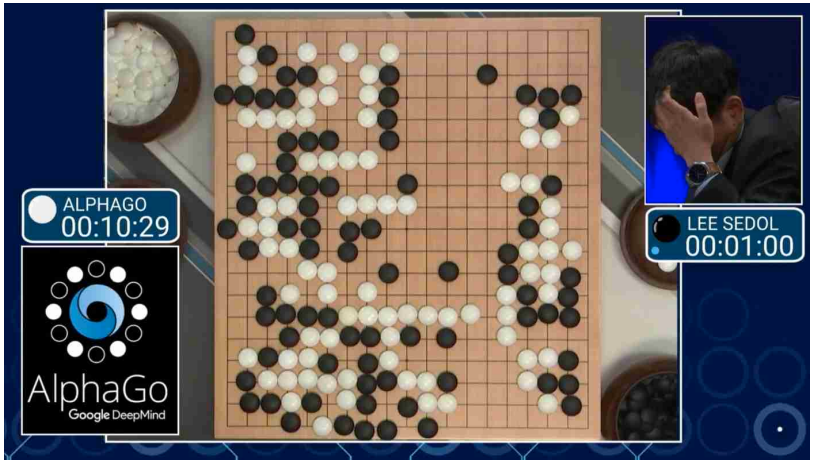
- 이로써 딥러닝에 가능성을 본 글로벌 IT 기업들이 딥러닝에 대규모 투자를 집행하게 됨.



간략히 살펴보는 딥러닝의 역사 (4세대 : 2012~현재)

- DeepMind는 Deep ANNs와 Reinforcement Learning를 결합한 Deep-Q-Networks(DQN)을 제안
- 이는 Reinforcement Learning에 대한 관심을 다시 불러일으키게 된다.





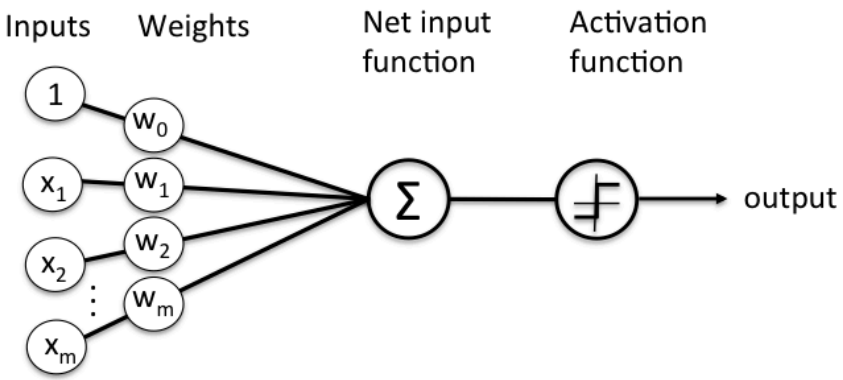
간략히 살펴보는 딥러닝의 역사 (4세대 : 2012~현재)

- 최근에는 Generative Model-Generative Adversarial Networks(GAN)-에 대한 연구가 활발히 진행중

Perceptron의 구조

- Perceptron의 구조

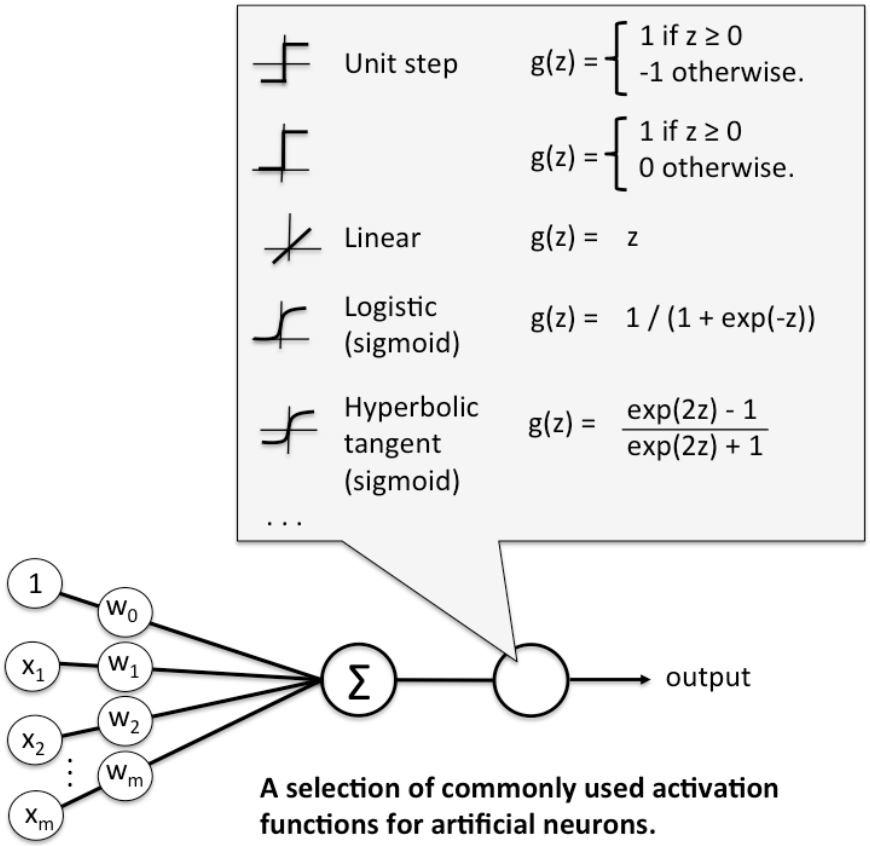
$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$



Schematic of Rosenblatt's perceptron.

Perceptron의 다양한 활성 함수들

- 퍼셉트론의 다양한 활성 함수들(activation functions)



A selection of commonly used activation functions for artificial neurons.

Perceptron의 동작과정의 직관적 이해

- **Perceptron의 가중치** : **Input의 중요도**를 나타낸다.

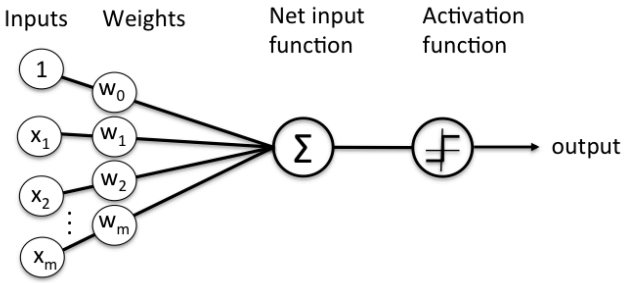
- 예를 들어, "주말에 집에서 나가 데이트를 할것인가?"에 대한 의사결정 모델을 Perceptron을 이용해서 만든다고 생각해보자.

- 이때 의사결정 고려사항은 다음과 같이 세가지라고 가정하자.

1. 날씨가 좋은가? (W1)

2. 남자친구 혹은 여자친구가 바쁜 일이 없는가? (W2)

3. 데이트 장소가 집에서 가까운가? (W3)



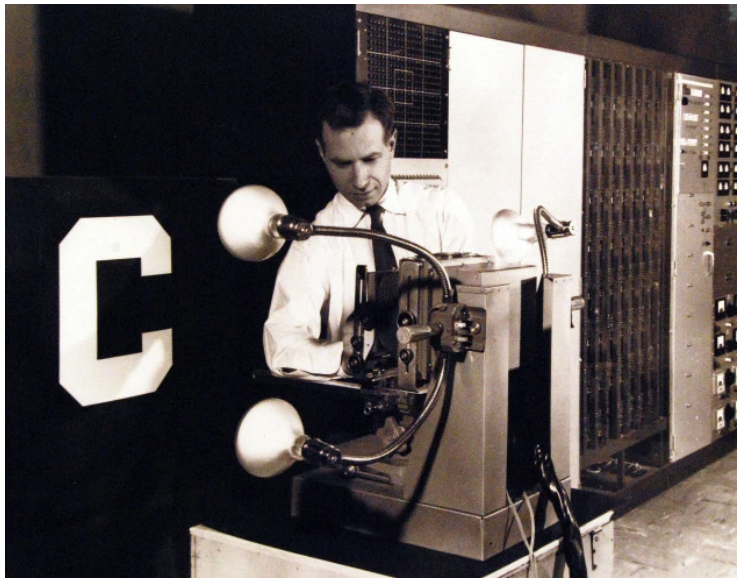
Schematic of Rosenblatt's perceptron.

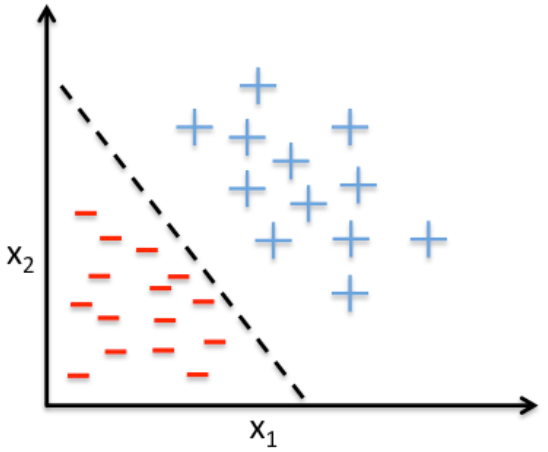
- 이렇게 세가지 고려사항을 Perceptron의 Input으로 넣을 수 있다. 이때 날씨를 가장 중요하게 고려하는 사람이라면 W1=6,W2=2,W3=2의 가중치를 줄 수 있다.

- 만약 데이트 장소가 집에서 가까운 것을 가장 중요하게 고려하는 사람이라면 W1=2,W2=2,W3=6의 가중치를 줄 수 있다.

Perceptron의 응용사례

- Perceptron의 응용 : 문자 인식, 분류(Classification)

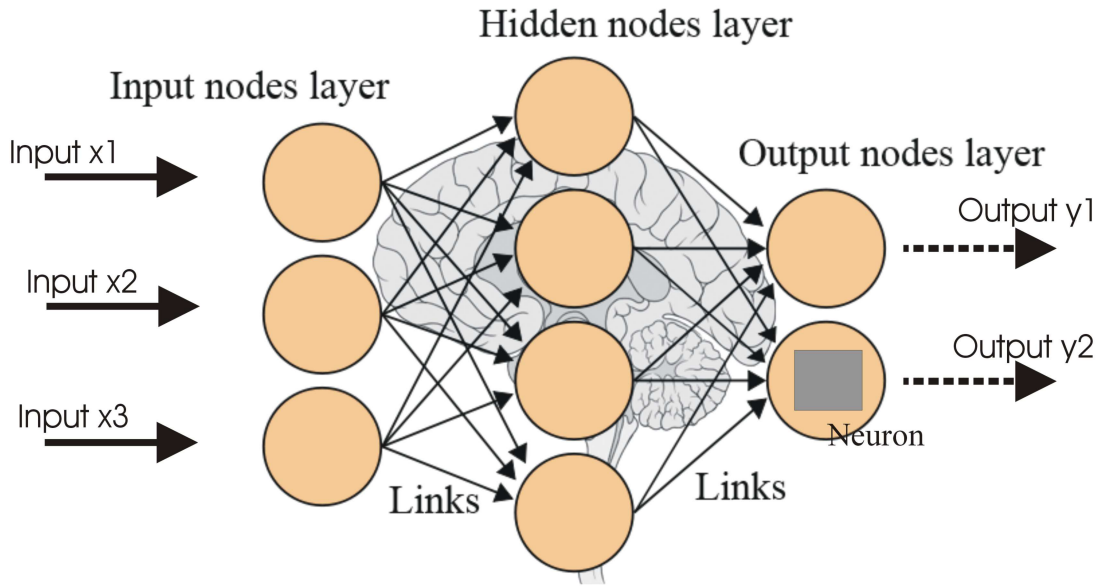




Example of a linear decision boundary for binary classification.

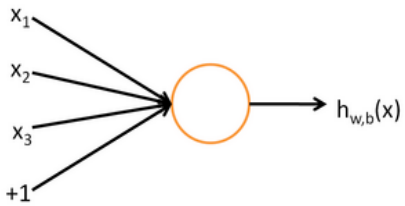
Artificial Neural Networks(ANNs)

- Artificial Neural Networks(ANNs) = Multi-Layer Perceptron(MLP)



Artificial Neural Networks(ANNs)

- 가장 간단한 Simple Neuron의 경우

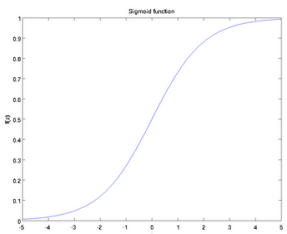


- Output의 값은 아래와 같다.

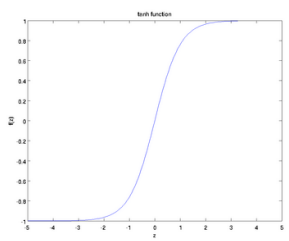
$$h_{W,b}(x) = f(W^T x) = f(\sum_{i=1}^{3} W_i x_i + b)$$

- $f(\cdot)$ 는 activation function을 나타낸다. 보통 sigmoid나 tanh을 사용한다

$$f(z) = \frac{1}{1+\exp(-z)}.$$

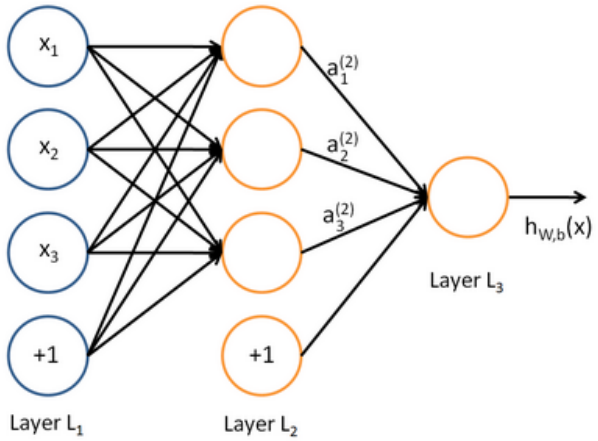


$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}},$$



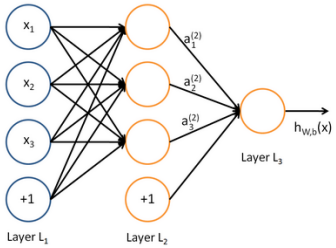
Artificial Neural Networks(ANNs)

- **Artificial Neural Networks(ANNs)**는 이런 simple neuron들을 여러개 쌓은 형태이다. (아래 그림의 경우, 3 input units, 3 hidden units, 1 output unit을 가진 3-Layer Neural Networks이다.)



- (W,b) = (W(1),b(1),W(2),b(2)), $W_{ij}^{(l)}$ 는 layer l 에서의 unit j, 그리고 layer l + 1 에서 unit i간의 connection을 나타낸다.

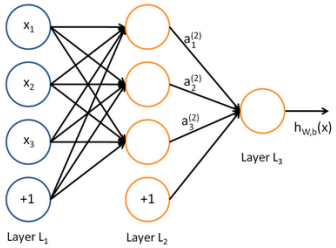
Artificial Neural Networks(ANNs)



- $a_i^{(l)}$ 는 layer l에서 unit i의활성값(activation)-출력값(output value)-을 나타낸다. l = 1에서는 $a_i^{(1)} = x_i$ 이고 i번째 input을 나타낸다.

$$
\begin{aligned}
a_1^{(2)} &= f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}) \\
a_2^{(2)} &= f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}) \\
a_3^{(2)} &= f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)}) \\
h_{W,b}(x) &= a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})
\end{aligned}
$$

Artificial Neural Networks(ANNs)



- $z_i^{(l)}$ 는 layer l에서 unit i의 input의 weighted sum과 bias를 더한 값을 나타낸다. (e.g. $z_i^{(2)} = \sum_{j=1}^{n} W_{ij}^{(1)} x_j + b_i^{(1)}$) $a_i^{(l)} = f(z_i^{(l)})$

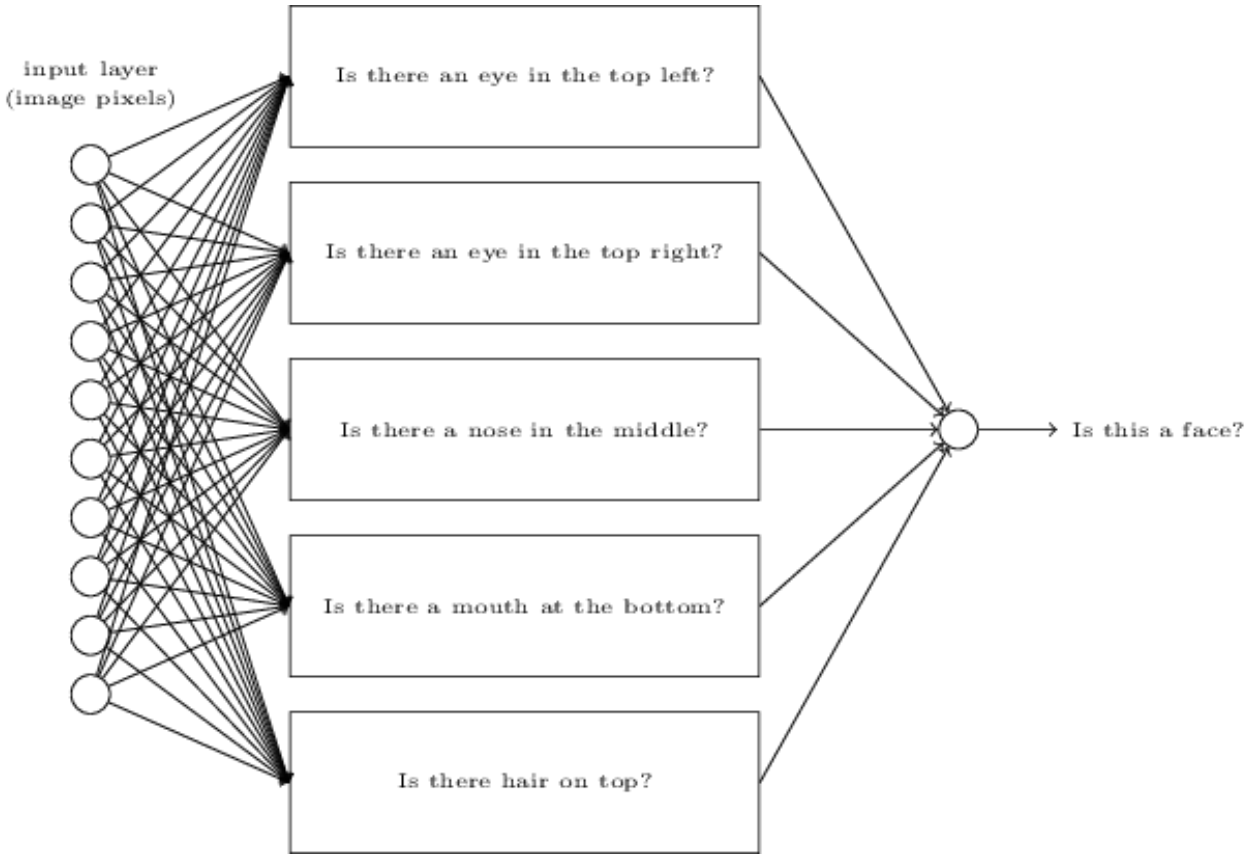
- 이렇게 output을 계산하는 과정을 **forward propagation**이라고 한다.

$$z^{(2)} = W^{(1)}x + b^{(1)}$$
$$a^{(2)} = f(z^{(2)})$$
$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$
$$h_{W,b}(x) = a^{(3)} = f(z^{(3)})$$

$$z^{(l+1)} = W^{(l)}a^{(l)} + b^{(l)}$$
$$a^{(l+1)} = f(z^{(l+1)})$$

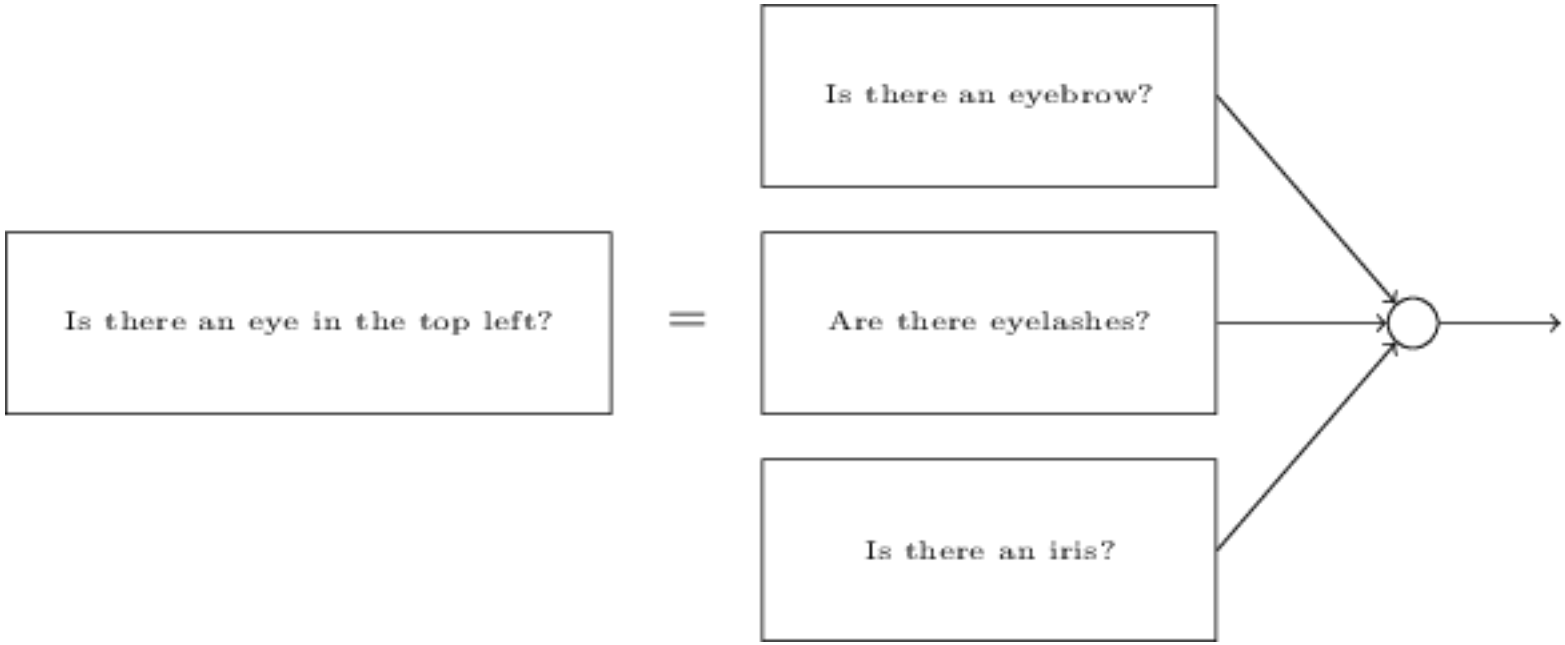
Artificial Neural Networks(ANNs) 의 동작과정의 직관적 이해 (1/2)

- Face Recognition(얼굴 인식) 문제에 적용했을 경우



Artificial Neural Networks(ANNs) 의 동작과정의 직관적 이해 (2/2)

- 각각의 질문들은 더 구체적이고 작은 질문들로 분해(Decompose) 될 수 있다.
- 이 과정들은 ANNs의 Weight 조정을 통해서 이루어진다.



Cost Function in ANNs

- ANNs의 Single example (x, y)에 대한 Minimum Squared-Error(MSE) cost function을 다음과 같이 정의할 수 있다.

$$J(W,b;x,y) = \frac{1}{2}\left\|h_{W,b}(x) - y\right\|^2.$$

- 전체 M개의 example에 대해서는 cost function을 아래와 같이 정의할 수 있다.

$$\begin{aligned} J(W,b) &= \left[\frac{1}{m}\sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)})\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2 \\ &= \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\left\|h_{W,b}(x^{(i)}) - y^{(i)}\right\|^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2 \end{aligned}$$

Cost Function in ANNs

$$J(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)})\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2$$

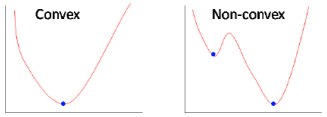
$$= \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\left\|h_{W,b}(x^{(i)}) - y^{(i)}\right\|^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2$$

- 여기서 첫번째 Term은 average sum-of-squared error, **두번째 Term은 regularization term (또는 weight decay term)**- overfitting을 방지하기 위해서 weights의 강도를 감소시킨다.- 이라고 부른다.

- **weight decay term parameter λ** 가 두 term들간의 상대적 중요도를 조절한다.

Gradient Descent In Neural Networks

- 우리의 목적은 J(W,b)를 minimize하는 것이다. 이를 위해 gradient descent algorithm을 이용한다. J(W,b)는 **non-convex function**이므로 gradient descent는 local optima에 빠지기 쉽지만, 실제 문제에 대해서는 대부분 잘 작동한다.



- **Gradient descent**의 한 iteration은 W,b를 다음과 같이 업데이트한다.

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b)$$

$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W,b)$$

- α 는 learning rate이다. 위 **수식의 핵심은 partial derivatives를 계산**하는 것이다. 이제 partial derivatives를 계산하기 위한 **backpropagation algorithm**을 살펴보자

Cost Function의 derivative

- 전체 Cost Function J(W,b)의 derivative는 아래와 같이 표현할 수 있다.

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b) = \left[\frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x^{(i)},y^{(i)}) \right] + \lambda W_{ij}^{(l)}$$

$$\frac{\partial}{\partial b_{i}^{(l)}} J(W,b) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial b_{i}^{(l)}} J(W,b;x^{(i)},y^{(i)})$$

$$\begin{aligned} J(W,b) &= \left[\frac{1}{m} \sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)}) \right] + \frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(W_{ji}^{(l)} \right)^2 \\ &= \left[\frac{1}{m} \sum_{i=1}^{m} \left(\frac{1}{2} \left\| h_{W,b}(x^{(i)}) - y^{(i)} \right\|^2 \right) \right] + \frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(W_{ji}^{(l)} \right)^2 \end{aligned}$$

Cost Function

Backpropagation Algorithm (오류역전파 알고리즘)

- Partial Derivative를 계산하기 위한 Backpropagation Algorithm은 다음과 같다.

1. Perform a feedforward pass, computing the activations for layers L_2, L_3, and so on up to the output layer L_{n_l}
2. For each output unit i in layer n_l (the output layer), set

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \|y - h_{W,b}(x)\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

3. For $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$

   For each node i in layer l, set

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})$$

4. Compute the desired partial derivatives, which are given as:

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y) = a_j^{(l)} \delta_i^{(l+1)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W,b;x,y) = \delta_i^{(l+1)}.$$

Backpropagation Algorithm (오류역전파 알고리즘) 예제

- F'(z) 구하기(activation function으로 sigmoid를 쓸 경우 $\sigma'(z) = \sigma(z)(1-\sigma(z))$)

1. Perform a feedforward pass, computing the activations for layers L_2, L_3, and so on up to the output layer L_{n_l}.
2. For each output unit i in layer n_l (the output layer), set
$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \|y - h_{W,b}(x)\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$
3. For $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$
For each node i in layer l set
$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})$$
4. Compute the desired partial derivatives, which are given as:
$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y) = a_j^{(l)} \delta_i^{(l+1)}$$
$$\frac{\partial}{\partial b_i^{(l)}} J(W,b;x,y) = \delta_i^{(l+1)}.$$

$$\frac{\partial J}{\partial W^{(2)}} = -(y - \hat{y}) \frac{\partial \hat{y}}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial W^{(2)}}$$

$$f(z) = \frac{1^u}{1 + e^{-z^v}}$$

$$f'(z) = \frac{u'v - uv'}{v^2} = \frac{0 \cdot v - 1 \cdot -e^{-z}}{(1+e^{-z})^2}$$

$$f'(z) = \frac{e^{-z}}{(1+e^{-z})^2}$$

Backpropagation Algorithm (오류역전파 알고리즘) 예제

- 우변을 F'(z^(3))로 다시 표현

1. Perform a feedforward pass, computing the activations for layers L_2, L_3, and so on up to the output layer L_{n_l}
2. For each output unit i in layer n_l (the output layer), set

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \|y - h_{W,b}(x)\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

3. For $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$
   For each node i in layer l, set

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})$$

4. Compute the desired partial derivatives, which are given as:

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y) = a_j^{(l)} \delta_i^{(l+1)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W,b;x,y) = \delta_i^{(l+1)}.$$

$$\frac{\partial J}{\partial W^{(2)}} = -(y - \hat{y}) f'(z^{(3)}) \frac{\partial z^{(3)}}{\partial W^{(2)}}$$

Backpropagation Algorithm (오류역전파 알고리즘) 예제

- 계산 과정을 Matrix로 나타냄

1. Perform a feedforward pass, computing the activations for layers L_2, L_3, and so on up to the output layer L_{n_l}
2. For each output unit i in layer n_l (the output layer), set

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \|y - h_{W,b}(x)\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

3. For $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$
   For each node i in layer l, set

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})$$

4. Compute the desired partial derivatives, which are given as:

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y) = a_j^{(l)} \delta_i^{(l+1)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W,b;x,y) = \delta_i^{(l+1)}.$$

$$\frac{\partial J}{\partial W^{(2)}} = -(y - \hat{y}) f'(z^{(3)}) \frac{\partial z^{(3)}}{\partial W^{(2)}}$$

$$\begin{bmatrix} -y_1 - \hat{y}_1 \\ -y_2 - \hat{y}_2 \\ -y_3 - \hat{y}_3 \end{bmatrix} \begin{bmatrix} f'(z_1^{(3)}) \\ f'(z_2^{(3)}) \\ f'(z_3^{(3)}) \end{bmatrix}$$

Backpropagation Algorithm (오류역전파 알고리즘) 예제

- 계산 과정을 Matrix로 나타냄

1. Perform a feedforward pass, computing the activations for layers L_2, L_3, and so on up to the output layer L_{n_l}
2. For each output unit i in layer n_l (the output layer), set

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \|y - h_{W,b}(x)\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

3. For $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$
   For each node i in layer l, set

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})$$

4. Compute the desired partial derivatives, which are given as:

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y) = a_j^{(l)} \delta_i^{(l+1)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W,b;x,y) = \delta_i^{(l+1)}.$$

$$\frac{\partial J}{\partial W^{(2)}} = -(y - \hat{y}) f'(z^{(3)}) \frac{\partial z^{(3)}}{\partial W^{(2)}}$$

$$\begin{bmatrix} \delta_1^{(3)} \\ \delta_2^{(3)} \\ \delta_3^{(3)} \end{bmatrix} = \delta^{(3)}$$

Backpropagation Algorithm (오류역전파 알고리즘) 예제

- 계산 과정을 Matrix로 나타냄

1. Perform a feedforward pass, computing the activations for layers L_2, L_3, and so on up to the output layer L_{n_l}.
2. For each output unit i in layer n_l (the output layer), set

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \|y - h_{W,b}(x)\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

3. For $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$
   For each node i in layer l, set

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})$$

4. Compute the desired partial derivatives, which are given as:

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y) = a_j^{(l)} \delta_i^{(l+1)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W,b;x,y) = \delta_i^{(l+1)}.$$

$$z^{(2)} = W^{(1)} x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

$$z^{(3)} = W^{(2)} a^{(2)} + b^{(2)}$$

$$h_{W,b}(x) = a^{(3)} = f(z^{(3)})$$

$$\frac{\partial J}{\partial W^{(2)}} = -(y - \hat{y}) f'(z^{(3)}) \frac{\partial z^{(3)}}{\partial W^{(2)}}$$

$$= \begin{bmatrix} \delta_1^{(3)} \\ \delta_2^{(3)} \\ \delta_3^{(3)} \end{bmatrix} \begin{bmatrix} a_{11}^{(2)} & a_{12}^{(2)} & a_{13}^{(2)} \\ a_{21}^{(2)} & a_{22}^{(2)} & a_{23}^{(2)} \\ a_{31}^{(2)} & a_{32}^{(2)} & a_{33}^{(2)} \end{bmatrix}$$

Backpropagation Algorithm (오류역전파 알고리즘) 예제

- 계산 과정을 Matrix로 나타냄

1. Perform a feedforward pass, computing the activations for layers L_2, L_3, and so on up to the output layer L_{n_l}.
2. For each output unit i in layer n_l (the output layer), set

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \|y - h_{W,b}(x)\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

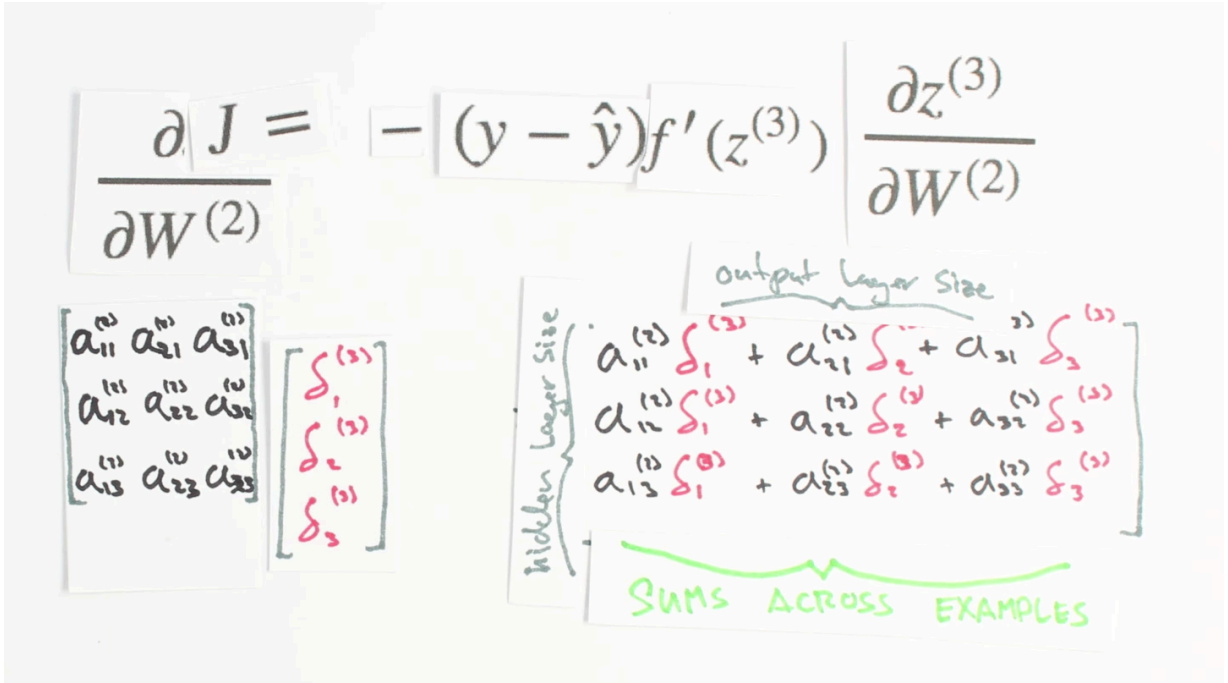
3. For $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$
   For each node i in layer l, set

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})$$

4. Compute the desired partial derivatives, which are given as:

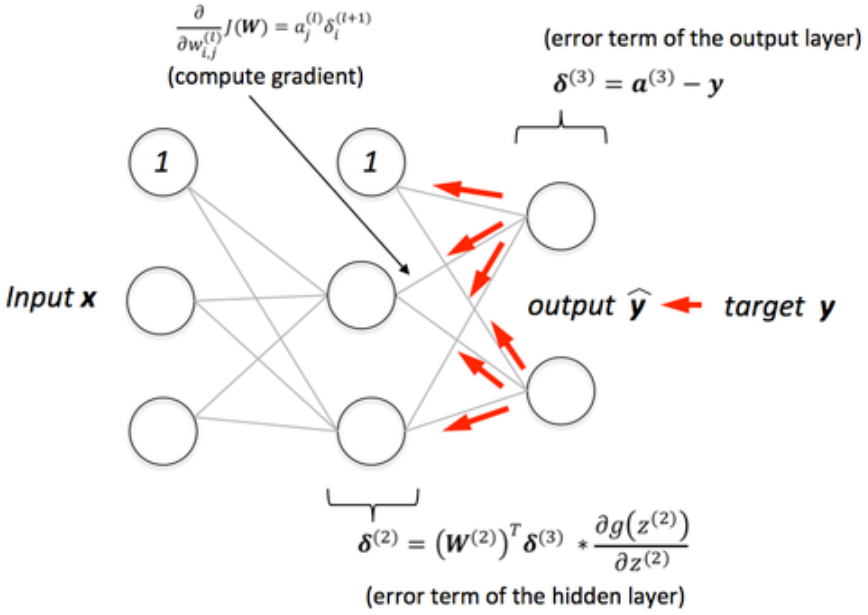
$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y) = a_j^{(l)} \delta_i^{(l+1)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W,b;x,y) = \delta_i^{(l+1)}.$$



Backpropagation Algorithm (오류역전파 알고리즘)

- 마지막 Output Layer로부터 오류(Error)를 역전파(Back-Propagate)하기 때문에 **Backpropagation Algorithm(오류역전파 알고리즘)**이라고 한다. (Output Layer -> Hidden Layer 2 -> Hidden Layer 1-> Input Layer)



Backpropagation Algorithm (오류역전파 알고리즘)

- Backpropagation Algorithm (matrix-vectorial notation)

1. Perform a feedforward pass, computing the activations for layers L_2, L_3, up to the output layer L_{n_l}, using the equations defining the forward propagation steps
2. For the output layer (layer n_l), set

$$\delta^{(n_l)} = -(y - a^{(n_l)}) \bullet f'(z^{(n_l)})$$

3. For $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$

   Set

$$\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)}\right) \bullet f'(z^{(l)})$$

4. Compute the desired partial derivatives:

$$\nabla_{W^{(l)}} J(W, b; x, y) = \delta^{(l+1)} (a^{(l)})^T,$$

compute : 계산, 계산하다, 평가하다, 어림하다, 추정하다

$$\nabla_{b^{(l)}} J(W, b; x, y) = \delta^{(l+1)}.$$

1. Perform a feedforward pass, computing the activations for layers L_2, L_3, and so on up to the output layer L_{n_l}
2. For each output unit i in layer n_l (the output layer), set

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \|y - h_{W,b}(x)\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

3. For $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$

   For each node i in layer l, set

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})$$

4. Compute the desired partial derivatives, which are given as:

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) = a_j^{(l)} \delta_i^{(l+1)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W, b; x, y) = \delta_i^{(l+1)}.$$

Gradient Descent Algorithm in ANNS

- Backpropagation Algorithm을 이용해서 계산한 gradient를 토대로 전체 Gradient Descent Algorithm을 나타내면 아래와 같다. (m개의 examples, layer l)

1. Set $\Delta W^{(l)} := 0$, $\Delta b^{(l)} := 0$ (matrix/vector of zeros) for all l.
2. For $i = 1$ to m,
   a. Use backpropagation to compute $\nabla_{W^{(l)}} J(W, b; x, y)$ and $\nabla_{b^{(l)}} J(W, b; x, y)$.
   b. Set $\Delta W^{(l)} := \Delta W^{(l)} + \nabla_{W^{(l)}} J(W, b; x, y)$.
   c. Set $\Delta b^{(l)} := \Delta b^{(l)} + \nabla_{b^{(l)}} J(W, b; x, y)$.
3. Update the parameters:

$$W^{(l)} = W^{(l)} - \alpha \left[\left(\frac{1}{m} \Delta W^{(l)} \right) + \lambda W^{(l)} \right]$$

$$b^{(l)} = b^{(l)} - \alpha \left[\frac{1}{m} \Delta b^{(l)} \right]$$

Gradient Descent in ANNs

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b)$$

$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W, b)$$

Partial Derivative in Cost Function

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) = a_j^{(l)} \delta_i^{(l+1)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W, b; x, y) = \delta_i^{(l+1)}.$$

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) = W_{ij}^{(l)} - \alpha \cdot a_j^{(l)} \delta_i^{(l+1)}$$

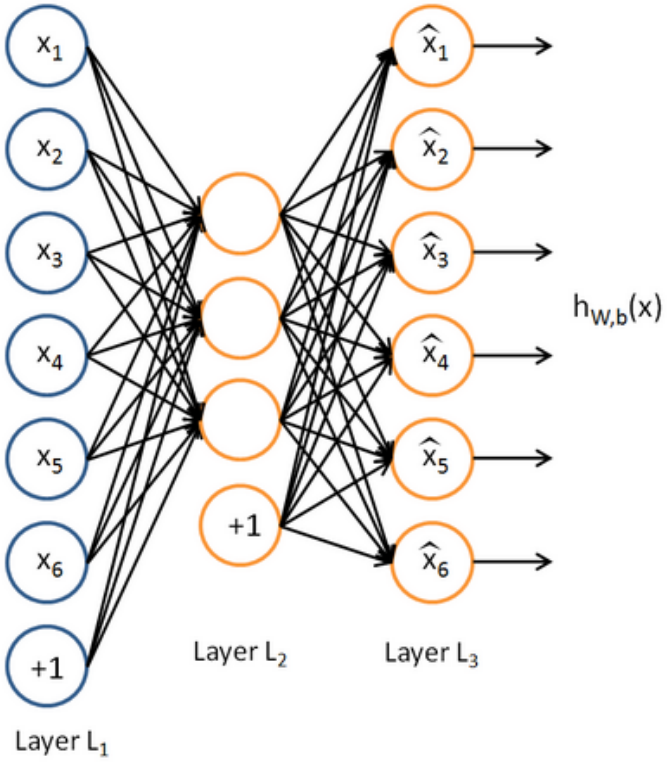
$$b_{ij}^{(l)} = b_{ij}^{(l)} - \alpha \frac{\partial}{\partial b_{ij}^{(l)}} J(W, b; x, y) = b_{ij}^{(l)} - \alpha \cdot \delta_i^{(l+1)}$$

다양한 Neural Networks 구조들

- Autoencoder
- Convolutional Neural Networks(CNNs)
- Recurrent Neural Networks(RNNs)
- Long-Short Term Memory(LSTM) Networks
- Generative Adversarial Networks(GAN)

Autoencoder

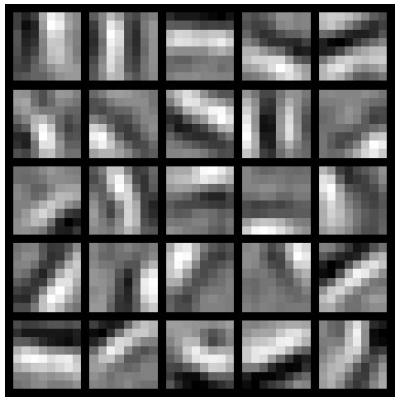
- Autoencoder : Input Unit의 개수와 Output Unit의 개수가 동일한 구조의 Neural Networks



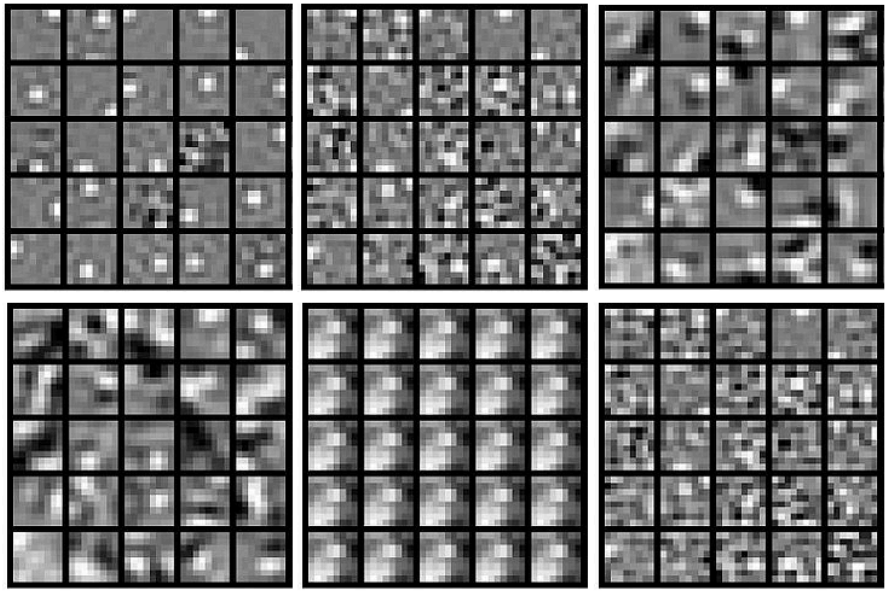
Autoencoder 학습결과

- Hidden Layer는 인풋에 대한 정보를 압축적으로 저장하기 때문에 아래와 같이 Feature를 학습하게 된다.

잘 학습된 경우



잘못 학습된 경우

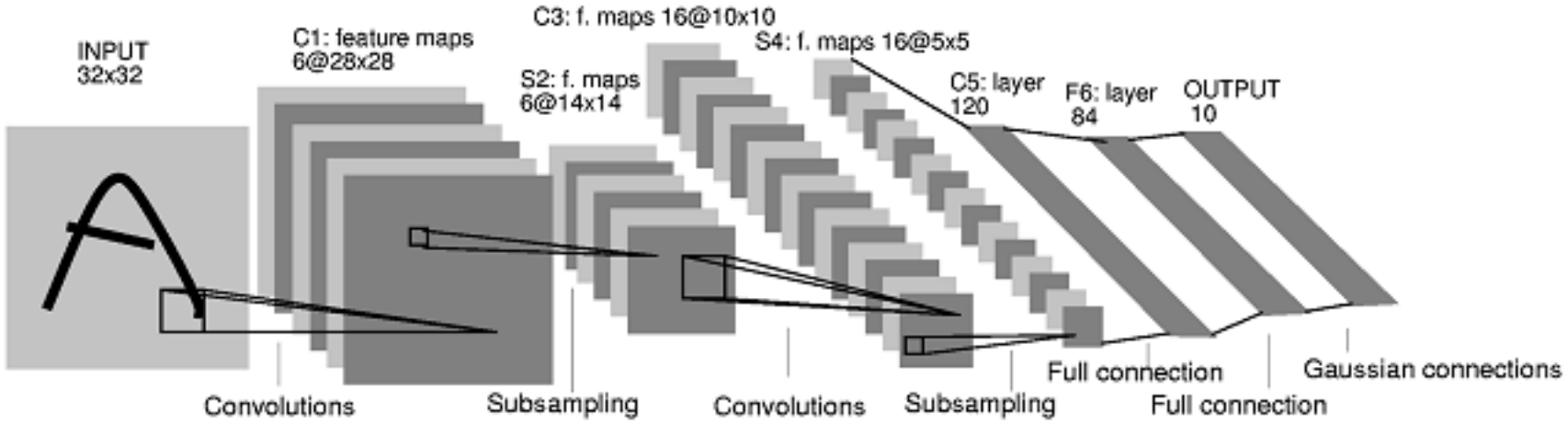


Autoencoder의 특징

- Autoencoder 주요 응용 분야 : Feature Extraction
- 특징 : 레이블(Labels)이 없는 데이터에 대한 Unsupervised Learning이 가능 함

Convolutional Neural Networks(CNNs)

- Convolutional Neural Networks(CNNs)의 구조

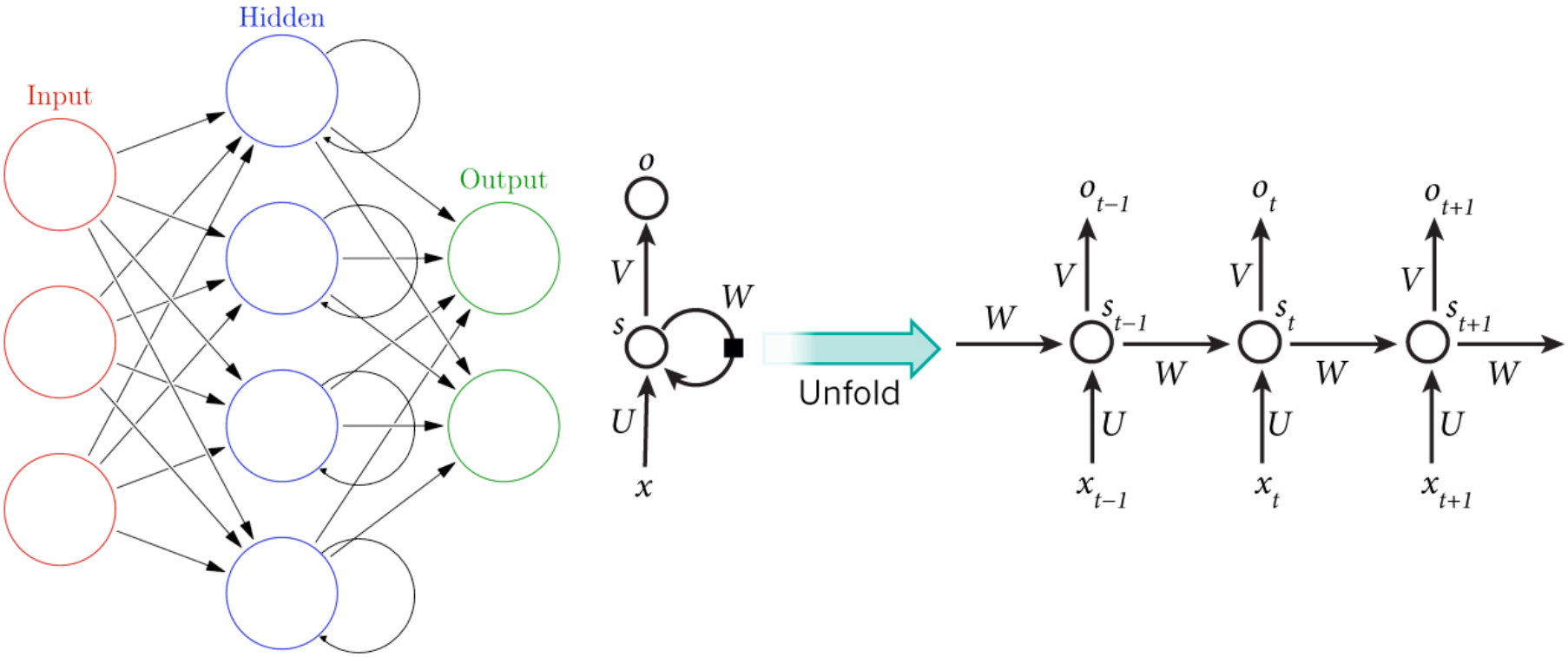


Convolutional Neural Networks(CNNs)의 특징

- Convolutional Neural Networks(CNNs) 주요 응용 분야 : 컴퓨터 비전
- 장점 : Convolution을 이용해서 이미지와 같이 dimension이 큰 데이터도 다룰 수 있게 됨
- 단점 : Pooling 과정에서 정보가 손실 됨

Recurrent Neural Networks(RNNs)

- Recurrent Neural Networks (RNNs)의 구조



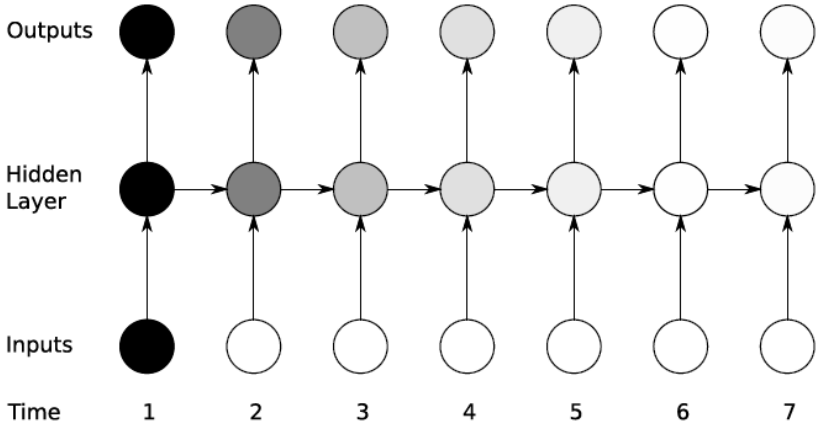
Recurrent Neural Networks(RNNs)의 특징

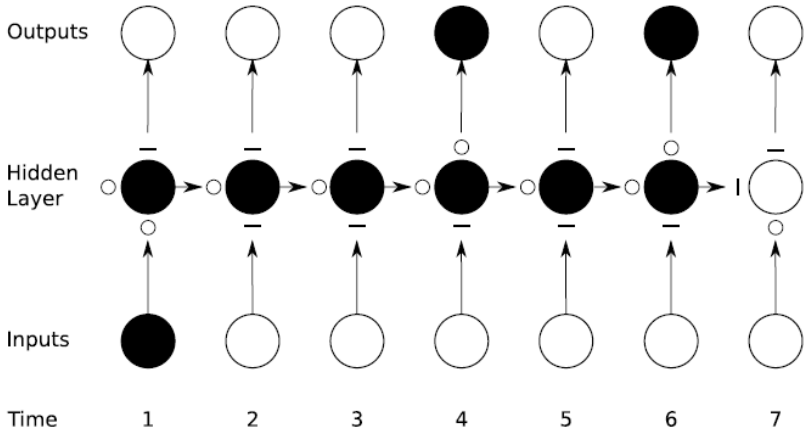
- Recurrent Neural Networks(RNNs) 주요 응용 분야 : 자연어 처리(시계열 데이터)
- 장점 : 시간 축이 추가 됨으로써 시간적 연속성을 가진 데이터를 잘 처리할 수 있게 됨. 이전 시간의 상태에 대한 정보를 일종의 메모리 형태로 가지고 있을 수 있다.
- 단점 : 연산량이 증가함, Vanishing Gradient Problem

Vanishing Gradient Problem

- Vanishing Gradient Problem in RNNs : 시간이 지나면서 이전 시간의 gradient의 영향력이 점점 작아지다가 사라지는 문제

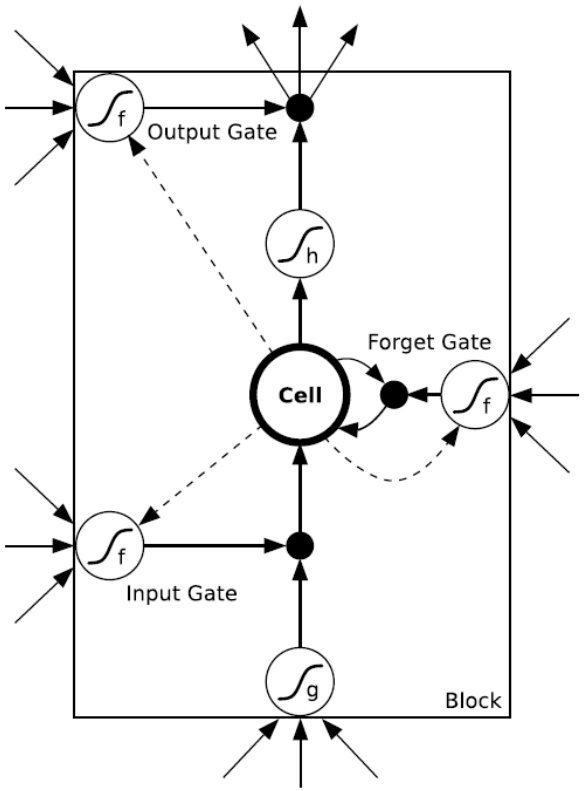
- LSTM을 이용한 해결 : 아래 그림과 같이, 시간 1에서의 인풋데이터를 받은 이후에 Input Gate를 닫아버려서 새로운 인풋을 받지 않고, Forget Gate를 열어놔서 시간 1에서의 인풋데이터를 계속해서 전달받으면, 시간 1에서의 인풋의 영향력을 계속해서 가져갈 수 있다.

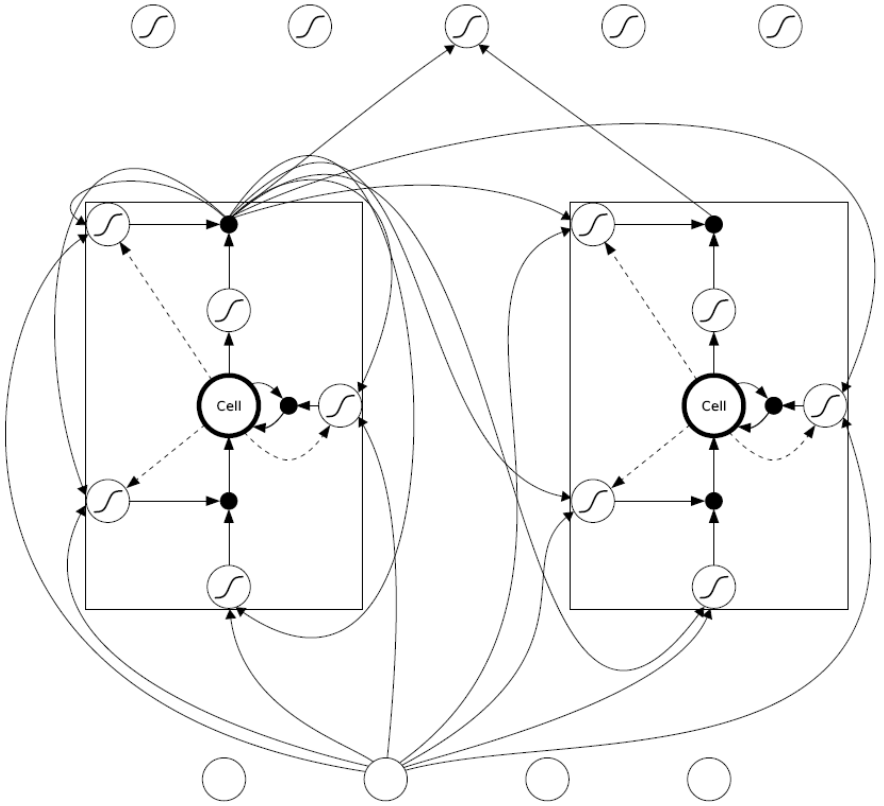




Long-Short Term Memory(LSTM) Networks

- Long-Short Term Memory(LSTM) Networks의 구조



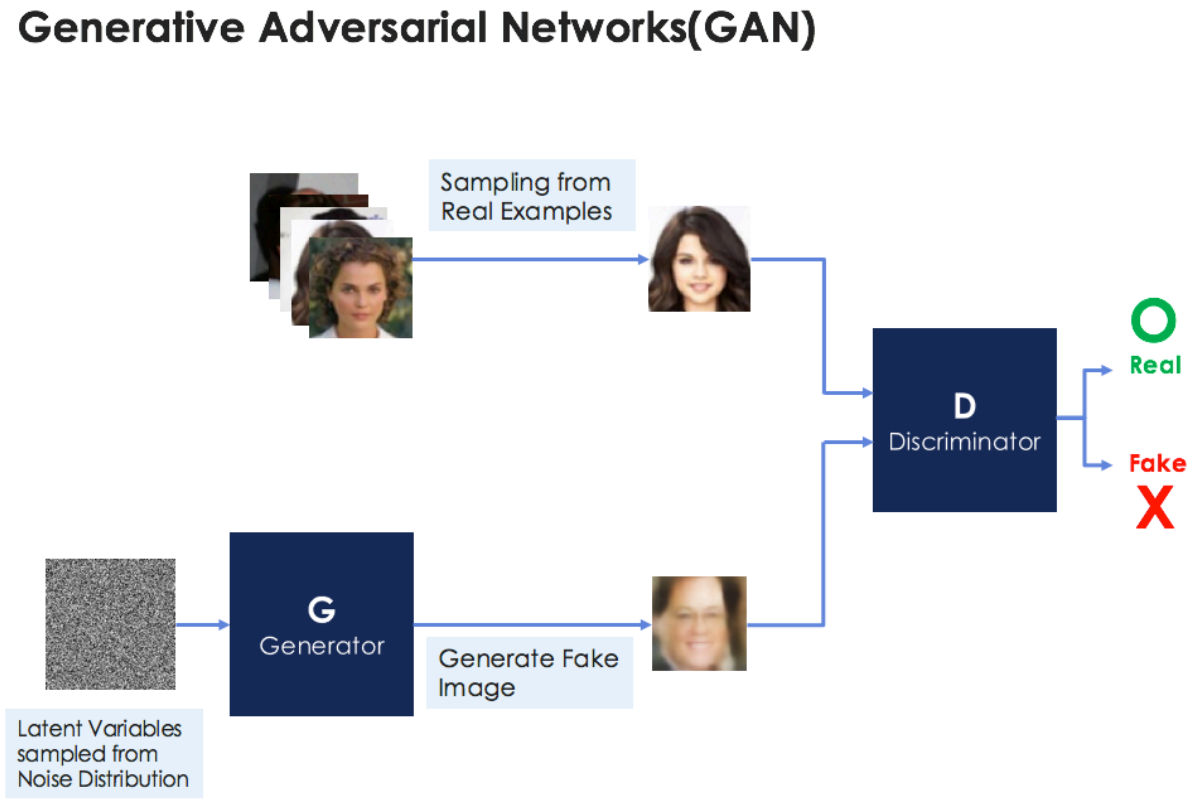


Long-Short Term Memory(LSTM) Networks의 특징

- Long-Short Term Memory(LSTM) Networks 주요 응용 분야 : 자연어 처리(시계열 데이터)
- 장점 : RNNs에 비해 좀 더 긴 시간의 dependency를 저장할 수 있다. (장기 기억 가능)
- 단점 : 연산량이 증가함

Generative Adversarial Networks(GAN)

- 경찰(Discriminator)과 위조지폐생성도둑(Generator)
- **Generator(생성자)** – Discriminator를 속이기 위한 이미지를 생성하도록 학습 된다.
- **Discriminator(구분자)** – 주어진 이미지가 진짜 이미지 인지 Generator가 생성한 가짜 이미지인지를 구분하도록 학습 된다.



Generative Adversarial Networks(GAN)의 특징

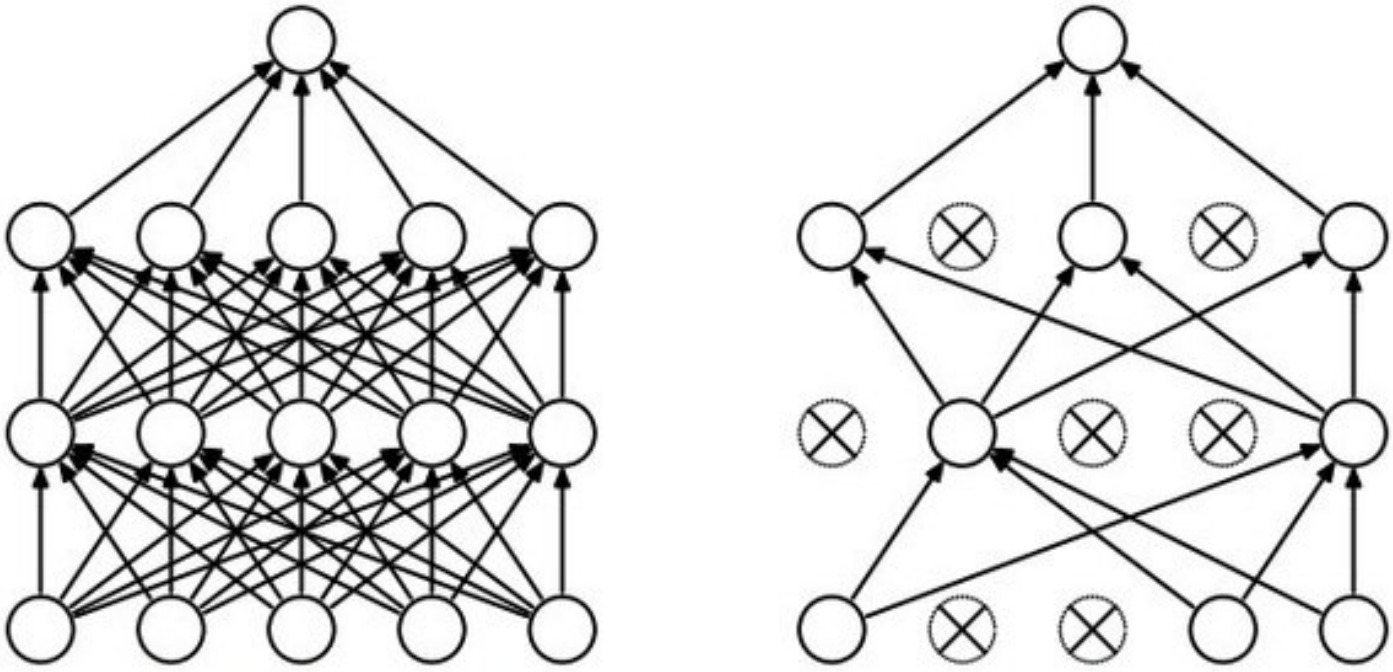
- Generative Adversarial Networks(GAN) 주요 응용 분야 : 이미지 생성(Image Generation)
- 장점 : 새로운 이미지를 생성할 수 있다.
- 단점 : 학습이 어려움

효율적인 학습을 위한 테크닉들

- Dropout
- ReLU(Rectified Linear Unit) activation function
- Fine-tuning(Transfer Learning)

Dropout

- Overfitting을 방지하기 위한 기법. 학습과정에서 node들을 일정 확률로 Drop(사용하지 않음)한다.



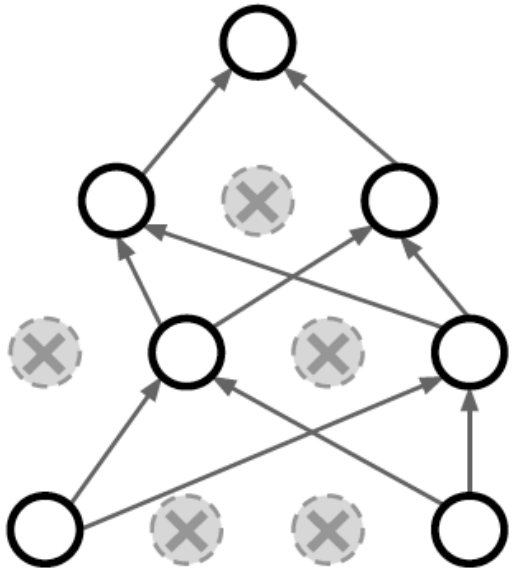
(a) Standard Neural Net

(b) After applying dropout.

Dropout 동작 과정의 직관적 이해

Regularization: Dropout

How can this possibly be a good idea?



Forces the network to have a redundant representation;
Prevents co-adaptation of features

Slide credit: CS231n (https://goo.gl/QSA36t)

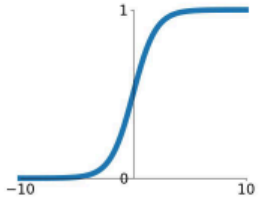
다양한 Activation Function들

- Overfitting을 방지하기 위해 최근에 제안된 다양한 activation function들

Activation Functions

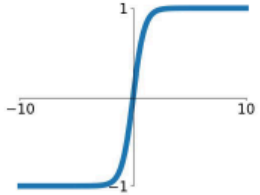
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



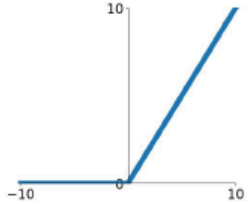
tanh

$$\tanh(x)$$



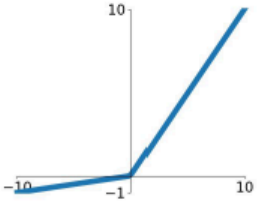
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

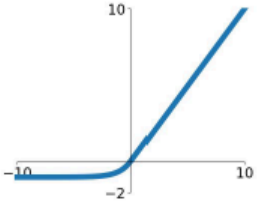


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

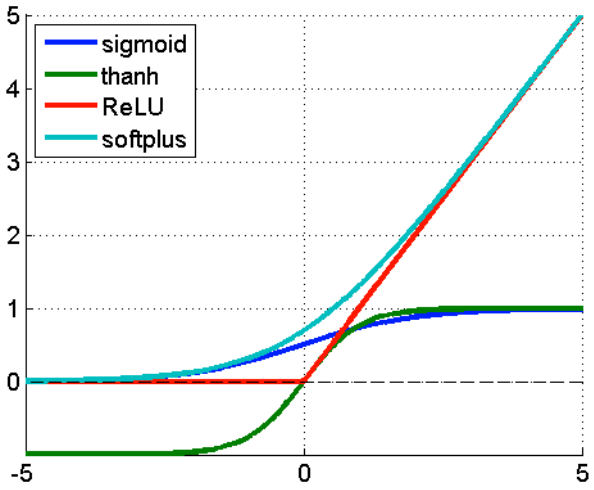
ELU

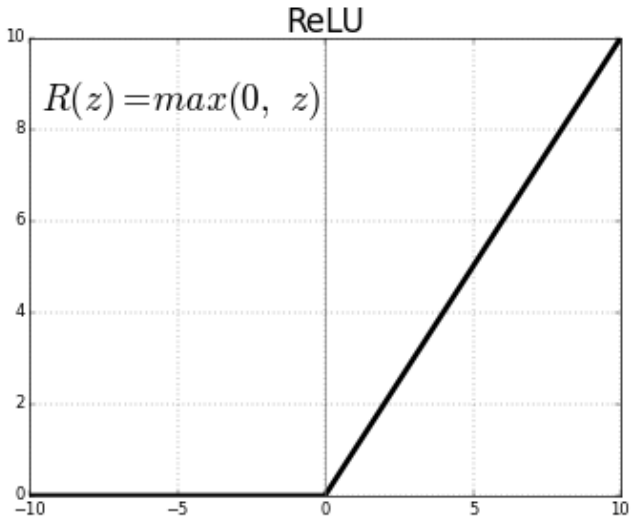
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



ReLU(Rectified Linear Unit) activation function

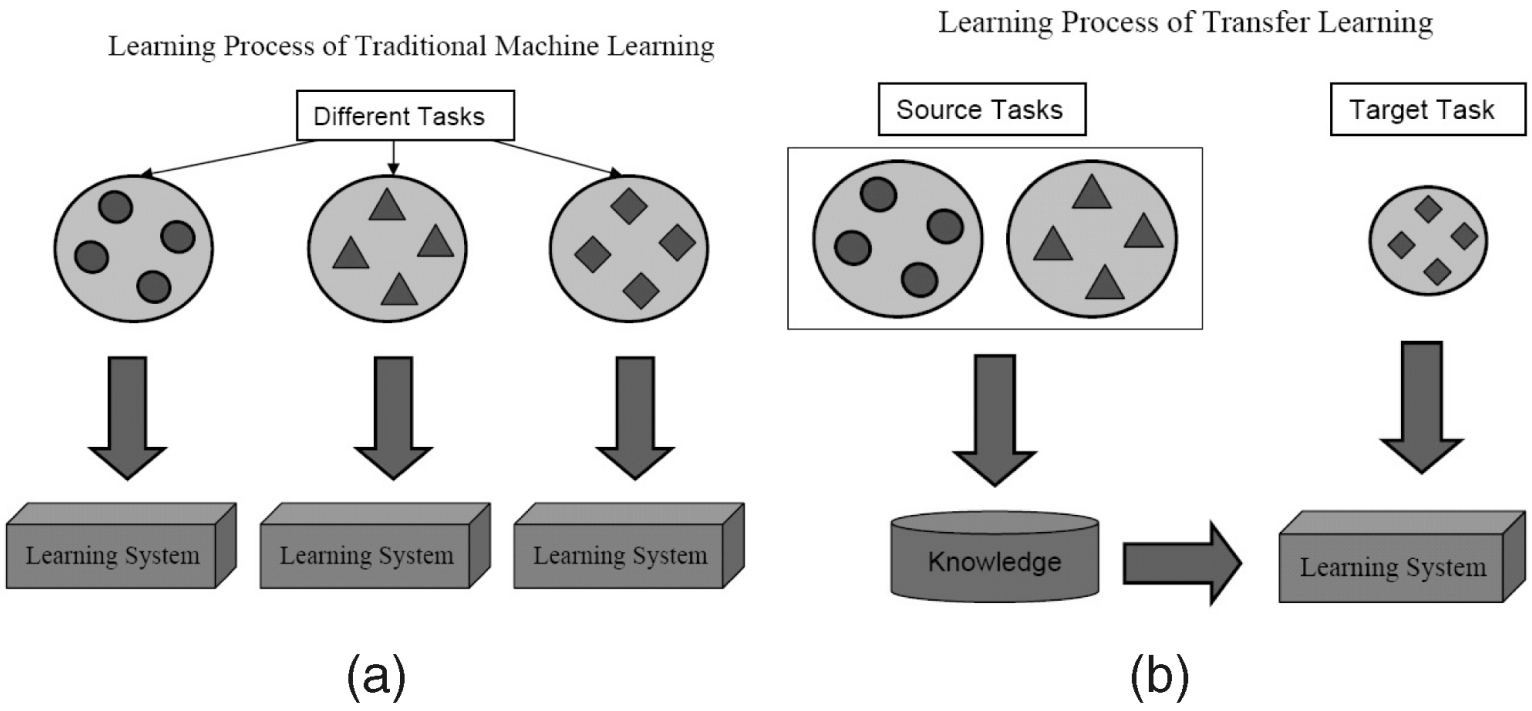
- Overfitting을 방지하기 위해 최근에 제안된 activation function.
- Sigmoid activation function의 문제점을 해결하기 위해 제안 됨
- 실험결과 sigmoid 함수보다 약 6배 빠른 학습 속도를 보임





Fine-tuning(Transfer Learning)

- 이미 학습된 Neural Networks를 새로운 Task에 맞게 다시 미세조정(Fine-Tuning)한다.

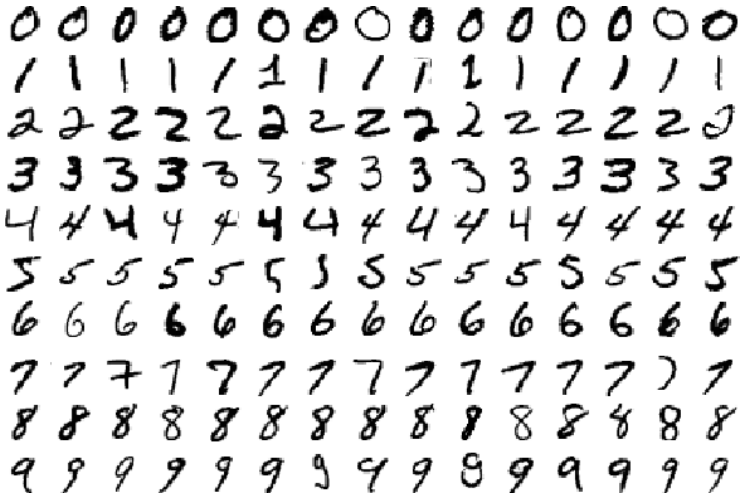


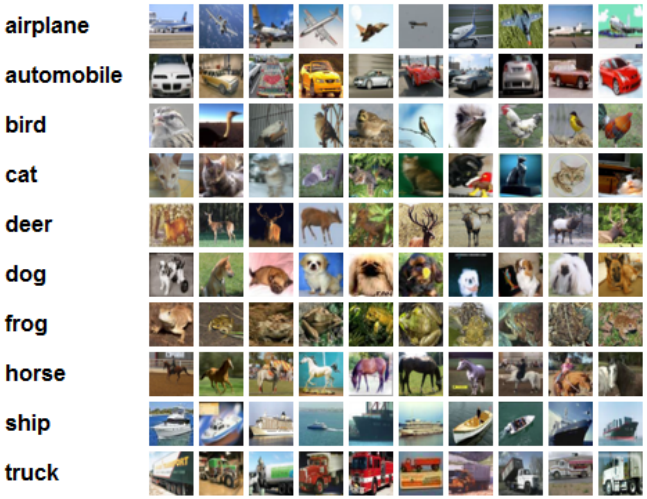
수업에서 다룰 컴퓨터비전 문제들

- Image Classification (MNIST 숫자 분류, CIFAR-10 이미지 분류)
- Inception v3 Retraining
- 한글 Char-RNN (한글 희곡 생성)
- Visual Q&A
- Image Captioning
- Deconvolution, DeepDream, Neural Style Transfer
- Generative Model – Variational AutoEncoder(VAE) & Generative Adversarial Networks(GAN)
- AutoPilot
- Brain Tumor Segmentation

Image Classification(MNIST, CIFAR-10)

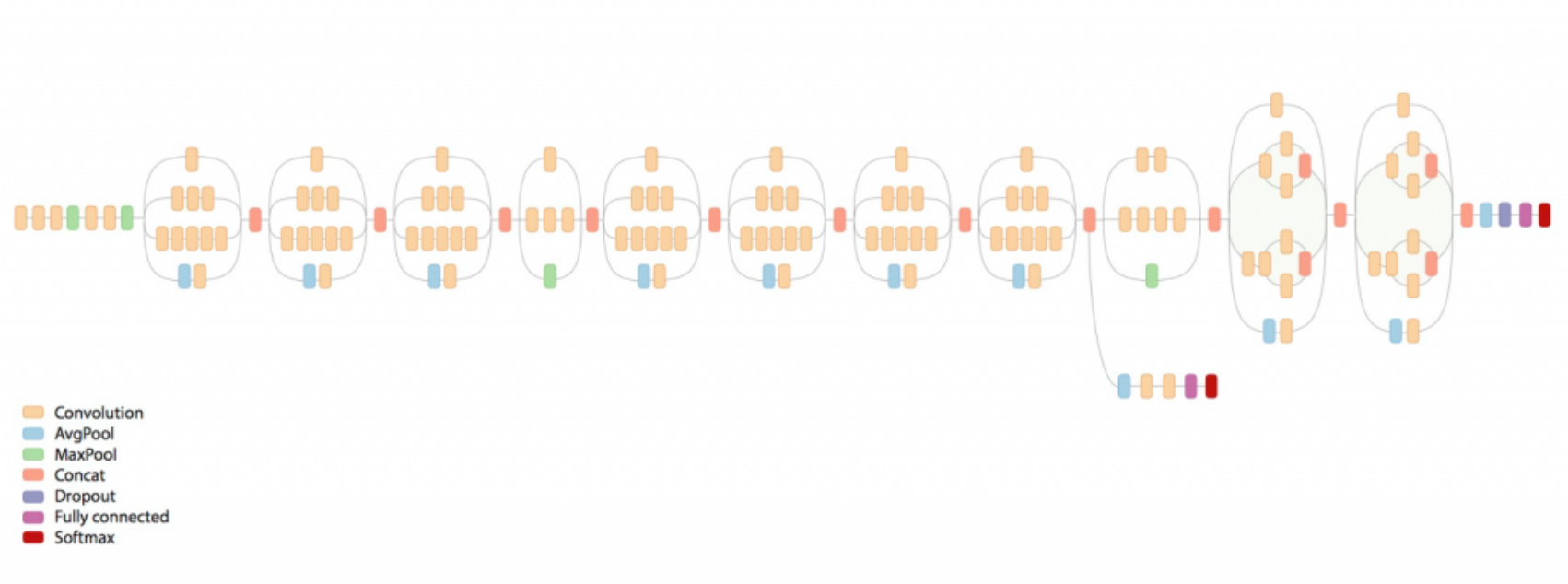
- MNIST, CIFAR-10 이미지 분류 문제





Inception v3 Retraining

- Inception v3 Retraining



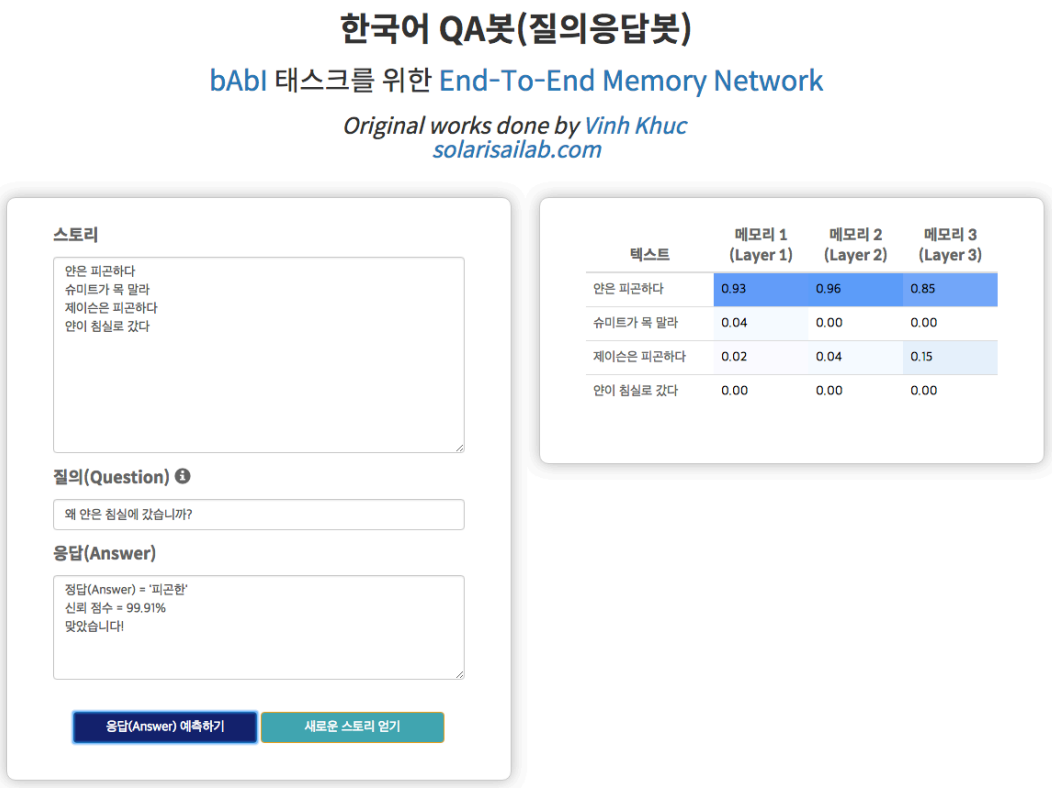
한글 Char-RNN

- RNN을 이용한 한글 희곡 생성(generation)(<고도를 기다리며> 텍스트 데이터를 이용한 학습)

```
에스트라공: (무대 옆에서 몸을 비꼰다.) 앞으로!
블라디미르: 나는 우릴 고맙다 (헐떡내 다친다) 이것 사실이오. (럭키에게) 하지만 생
각을 할까? 무슨 소리를 터터봐
블라디미르: 회연하는 성을 거일도 좋니?
에스트라공: 벌써 미칠대를 럭키의 주위를 지면줘요!
블라디미르: 빨리빨리! 이놈 좀 일으켜 세워요!
그는 발을 멈추고 블라디미르 모자를 놓고 럭키의 모자를 고우지 않을 거들어 줘!
블라디미르가 끈과 소년은 물러서며) 이번 한테전 것도 없었냐?
에스트라공: 안 그래요!
블라디미르: 조용히!
침묵)
에스트라공: 왜 왜 누구?
블라디미르: 고도를 기다려야지.
에스트라공: 참 그렇구 말야?
블라디미르: 고도가 싫다니까. 벌써 시간이 흐르는 게 있으면
```

NLP Q&A

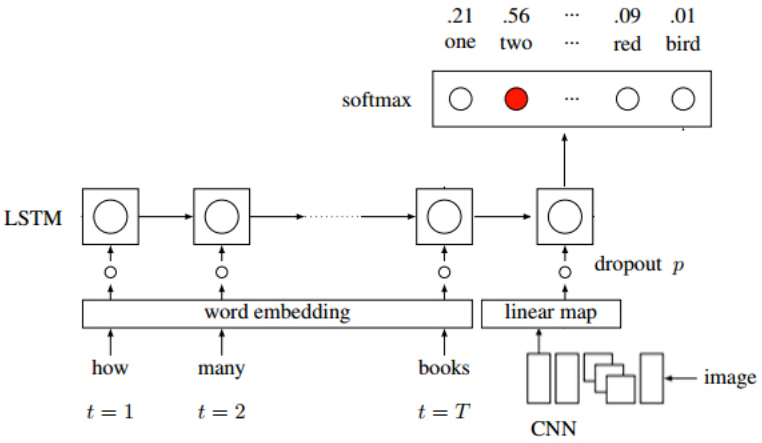
- "왜 얀은 침실로 갔습니까?"(Question) -> "피곤해서"(Answer)



http://solaris33.pythonanywhere.com/

Visual Q&A

- Visual Q&A



DAQUAR 1553
What is there in front of the sofa?
Ground truth: table
IMG+BOW: table (0.74)
2-VIS+BLSTM: table (0.88)
LSTM: chair (0.47)

COCOQA 5078
How many leftover donuts is the red bicycle holding?
Ground truth: three
IMG+BOW: two (0.51)
2-VIS+BLSTM: three (0.27)
BOW: one (0.29)

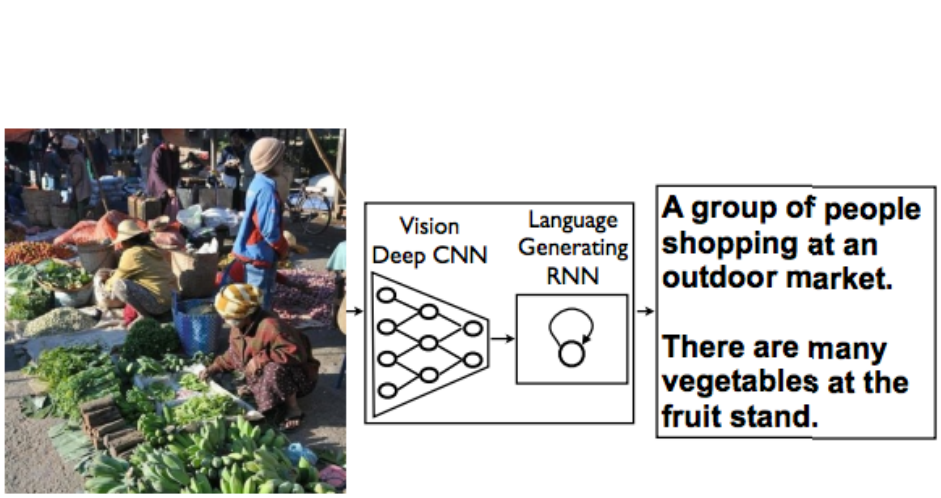
COCOQA 1238
What is the color of the tee-shirt?
Ground truth: blue
IMG+BOW: blue (0.31)
2-VIS+BLSTM: orange (0.43)
BOW: green (0.38)

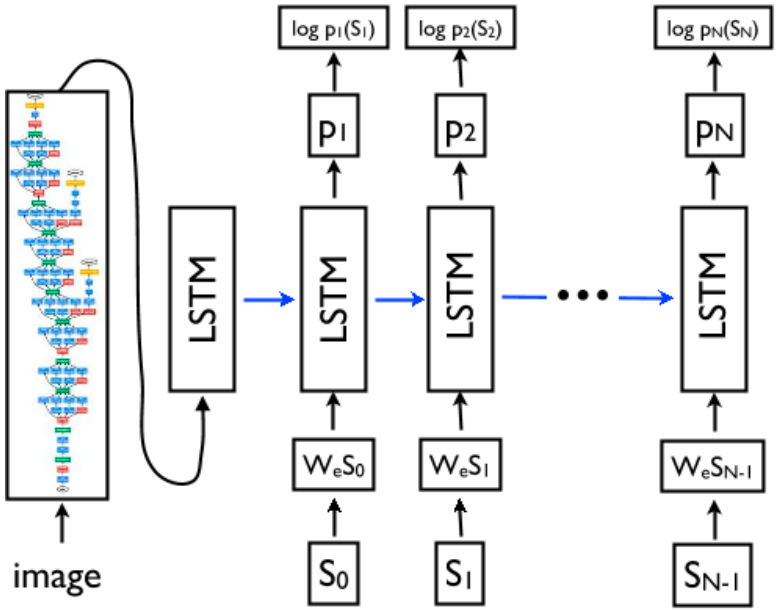
COCOQA 26088
Where is the gray cat sitting?
Ground truth: window
IMG+BOW: window (0.78)
2-VIS+BLSTM: window (0.68)
BOW: suitcase (0.31)

Figure 1: Sample questions and responses of a variety of models. Correct answers are in green and incorrect in red. The numbers in parentheses are the probabilities assigned to the top-ranked answer by the given model. The leftmost example is from the DAQUAR dataset, and the others are from our new COCO-QA dataset.

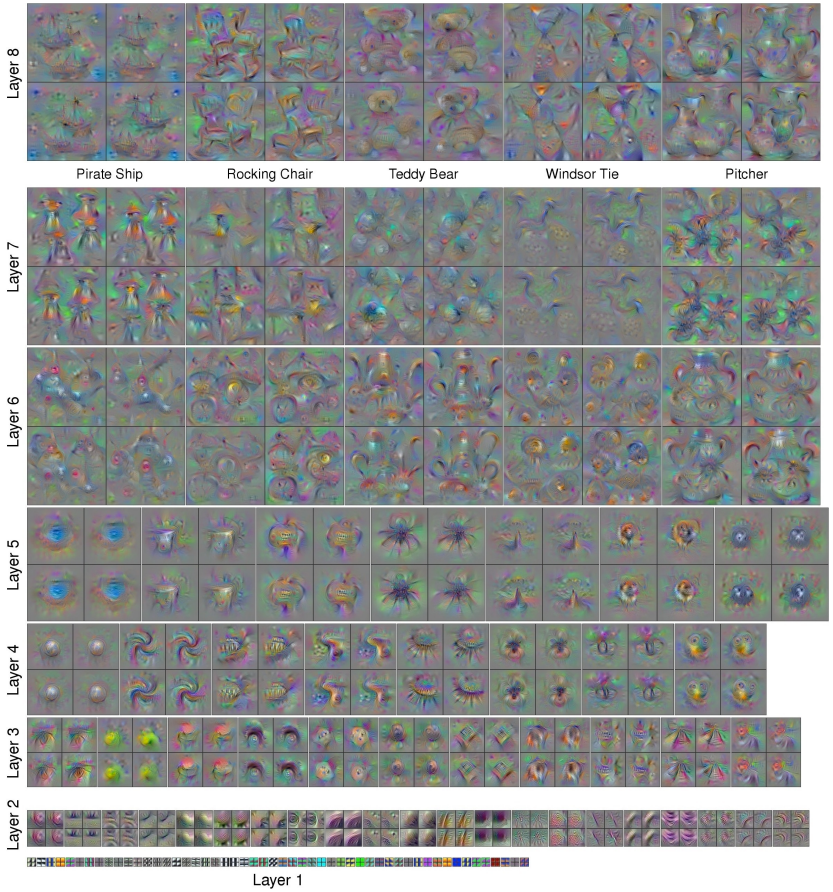
Image Captioning

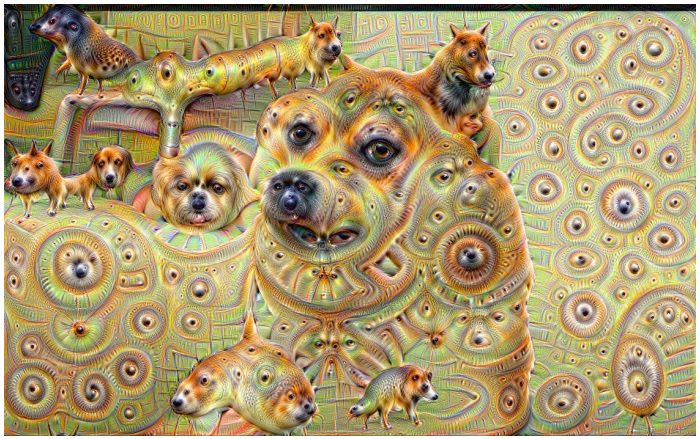
- Image Captioning





Deconvolution, DeepDream





Neural Style Transfer

Neural Style Transfer

Content Image

This image is licensed under CC-BY 3.0

+

Style Image

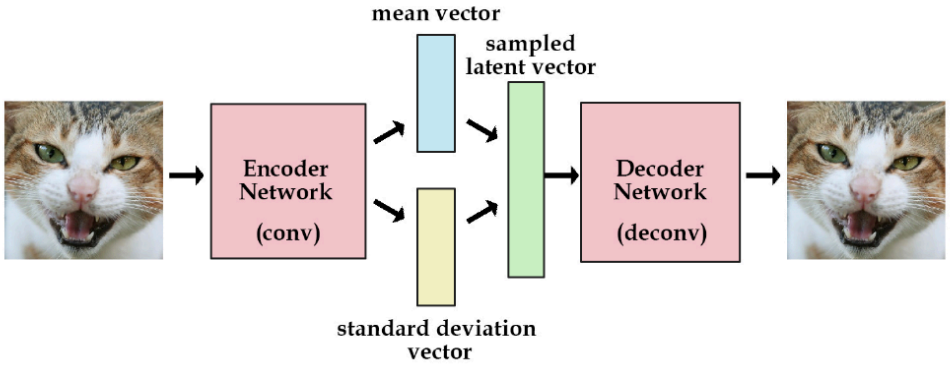
Starry Night by Van Gogh is in the public domain

=

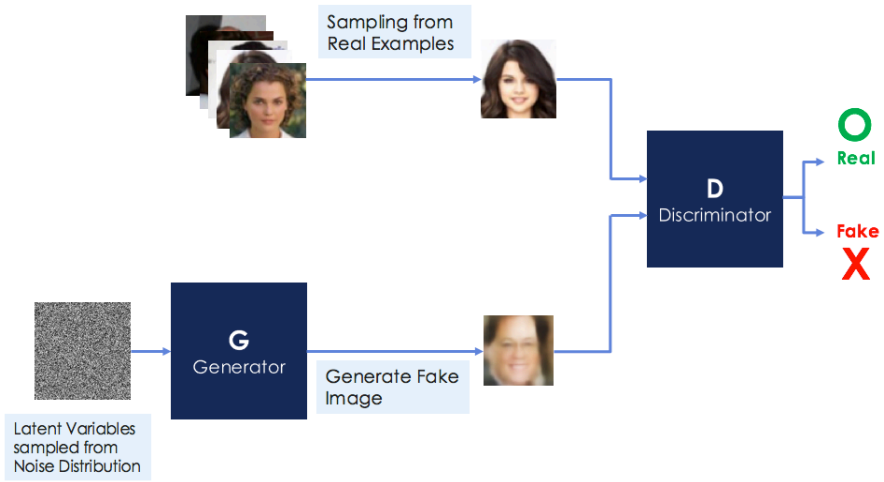
Style Transfer!

This image copyright Justin Johnson, 2015. Reproduced with permission.

Variational AutoEncoder(VAE) & Generative Adversarial Networks(GAN)

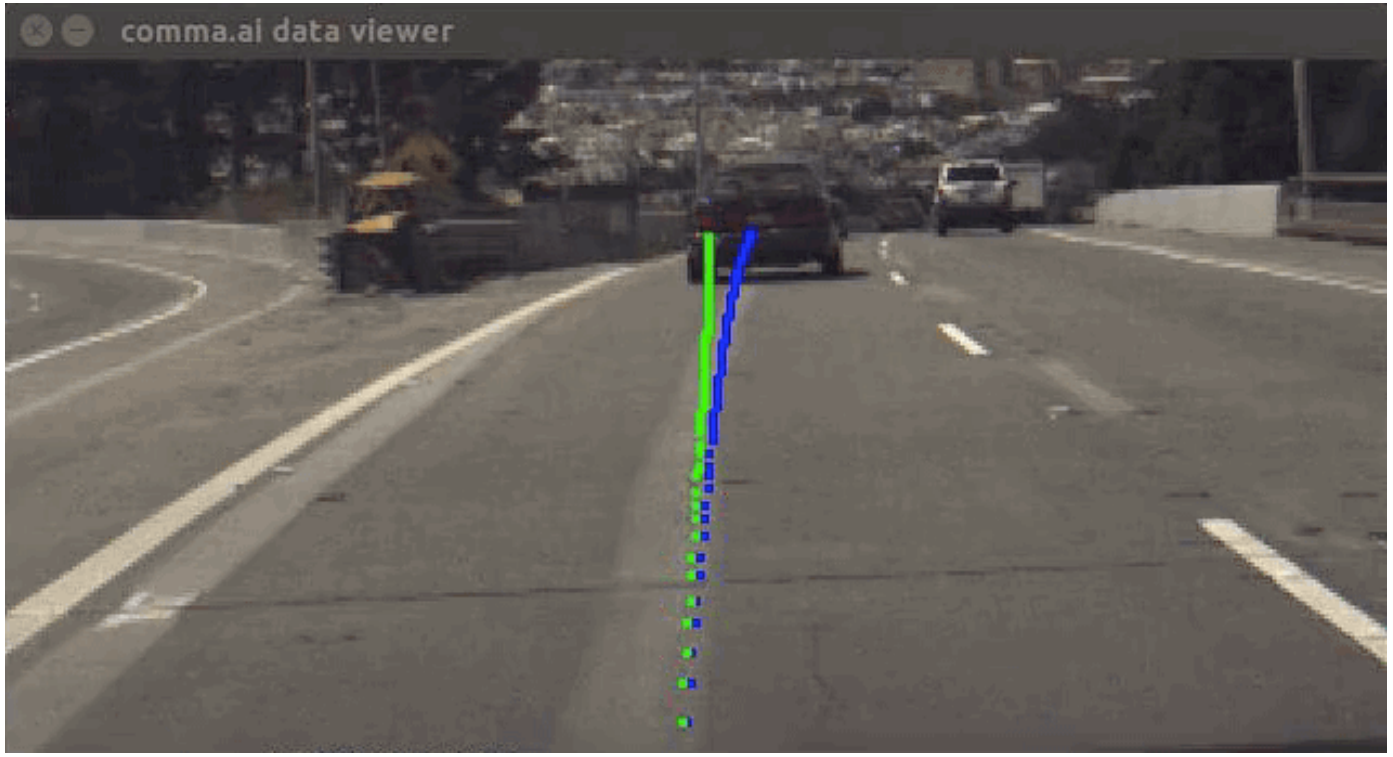


Generative Adversarial Networks(GAN)



AutoPilot

- AutoPilot



Semantic Image Segmentation

- Fully Convolutional Networks(FCNs)

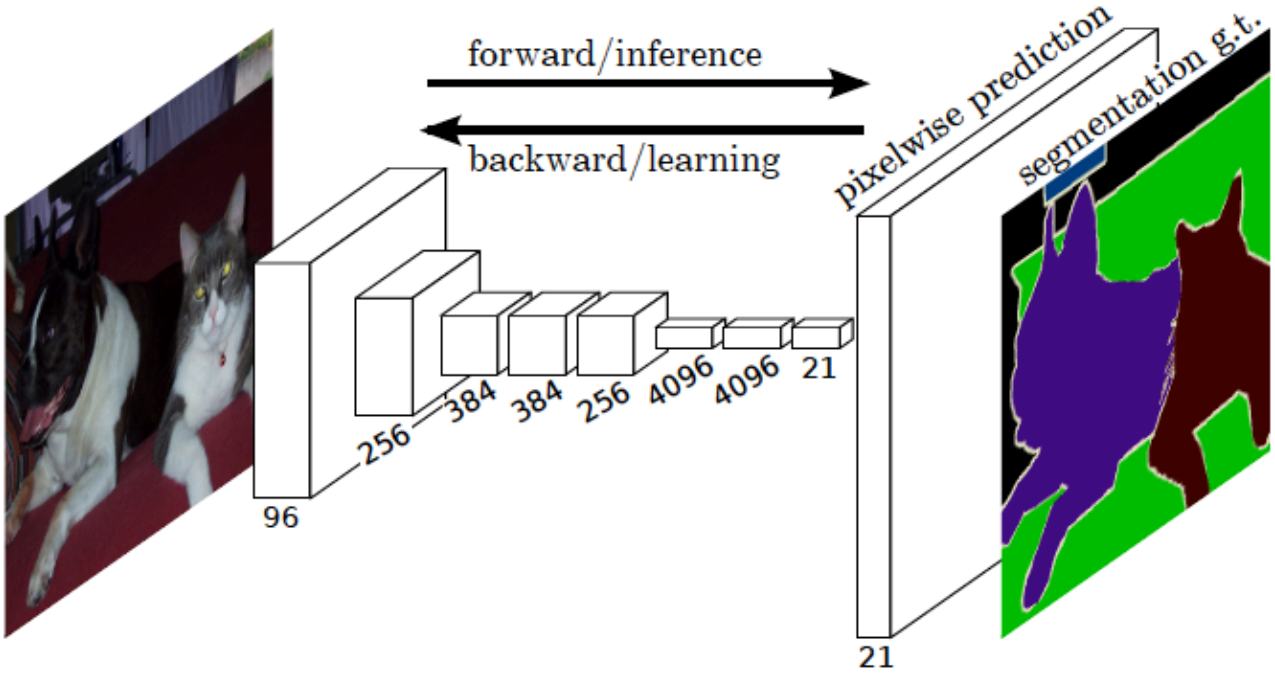
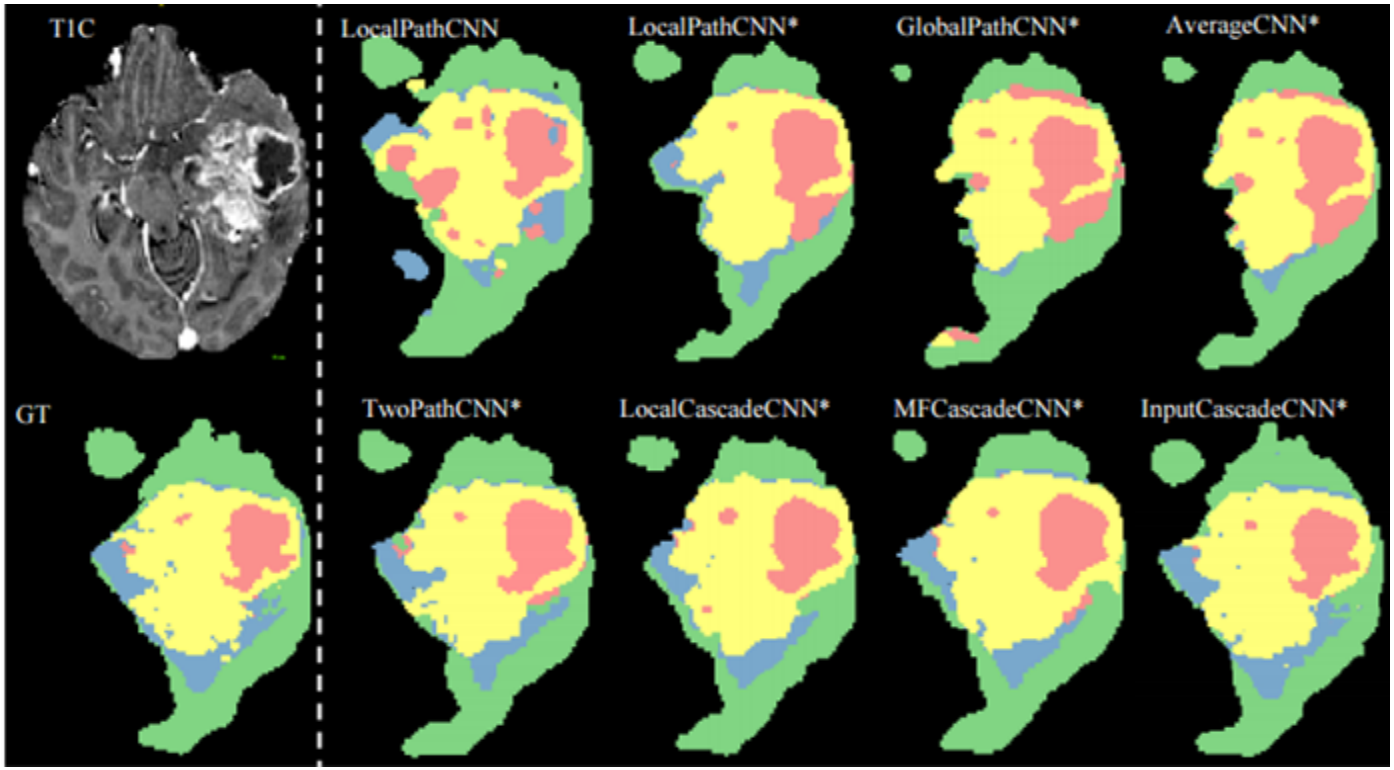


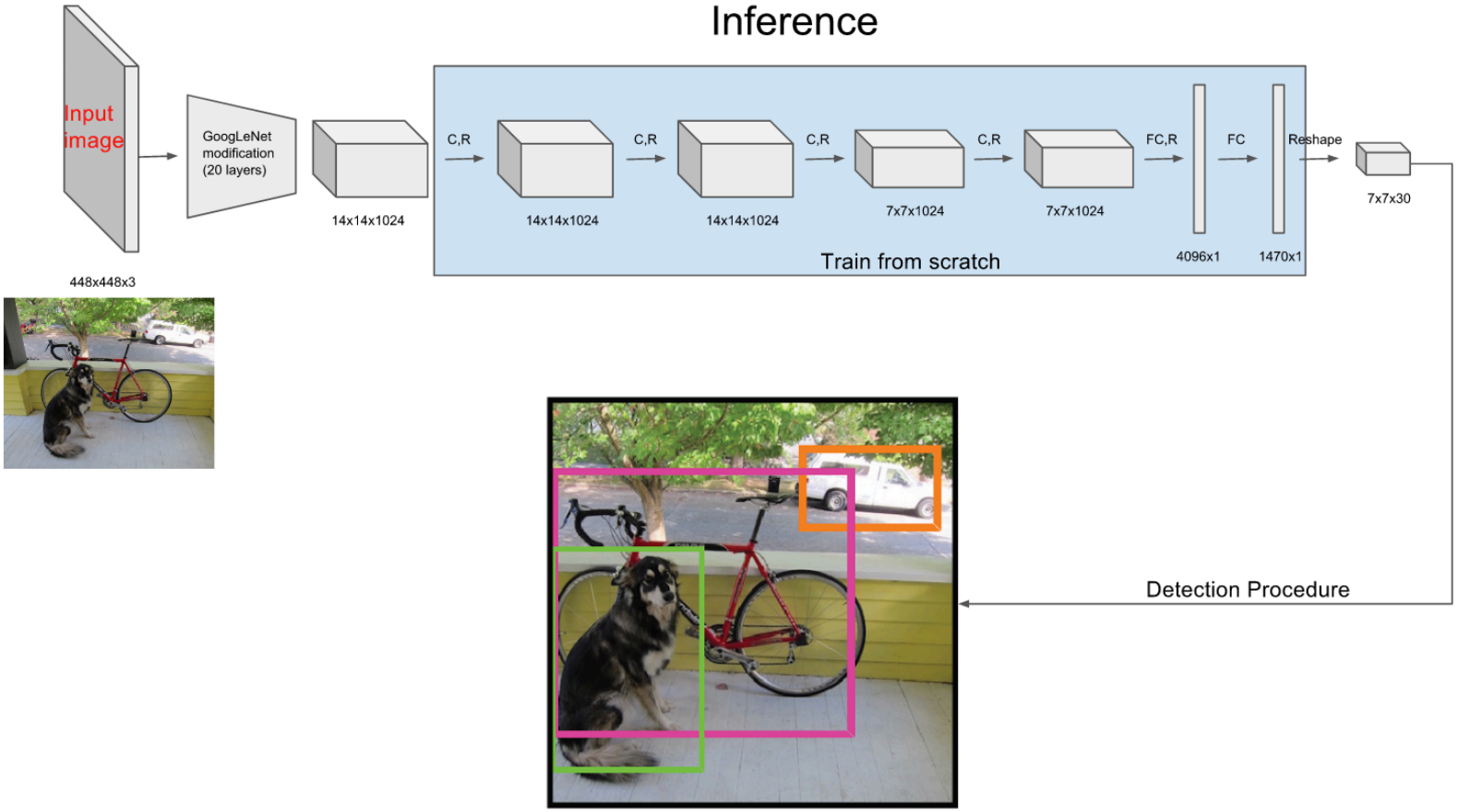
Figure 1. Fully convolutional networks can efficiently learn to make dense predictions for per-pixel tasks like semantic segmentation.

Brain Tumor Segmentation

- Brain Tumor Segmentation

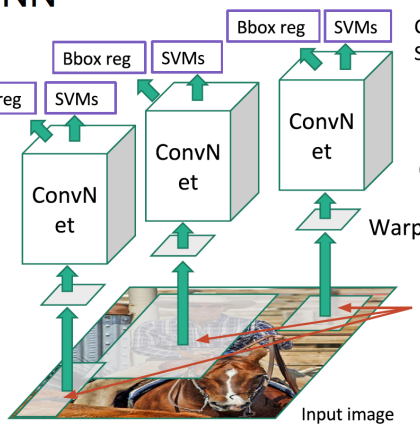


Object Detection - YOLO



Object Detection - R-CNN, Fast R-CNN, Faster R-CNN

R-CNN



Linear Regression for bounding box offsets

Classify regions with SVMs

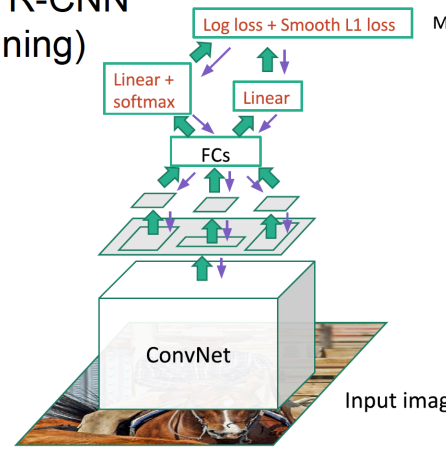
Forward each region through ConvNet

Warped image regions

Regions of Interest (RoI) from a proposal method (~2k)

Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Fast R-CNN (Training)



Multi-task loss

Girshick, "Fast R-CNN", ICCV 2015.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

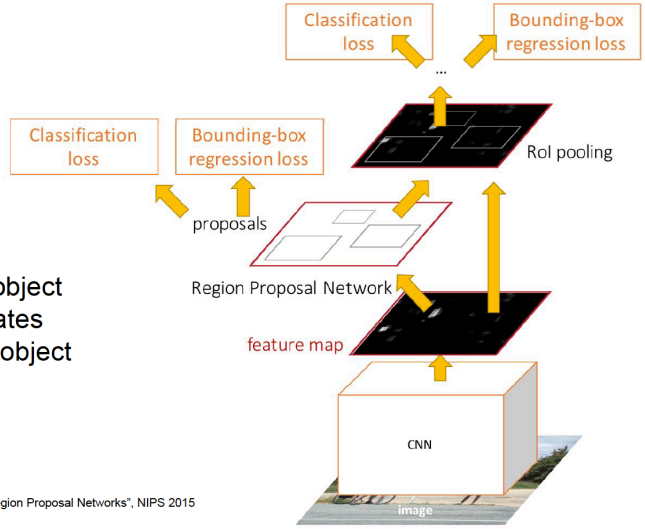
Faster R-CNN:

Make CNN do proposals!

Insert **Region Proposal Network (RPN)** to predict proposals from features

Jointly train with 4 losses:

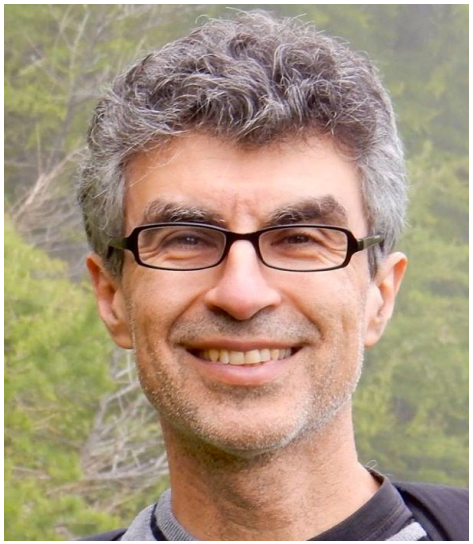
1. RPN classify object / not object
2. RPN regress box coordinates
3. Final classification score (object classes)
4. Final box coordinates



Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015
Figure copyright 2015, Ross Girshick; reproduced with permission

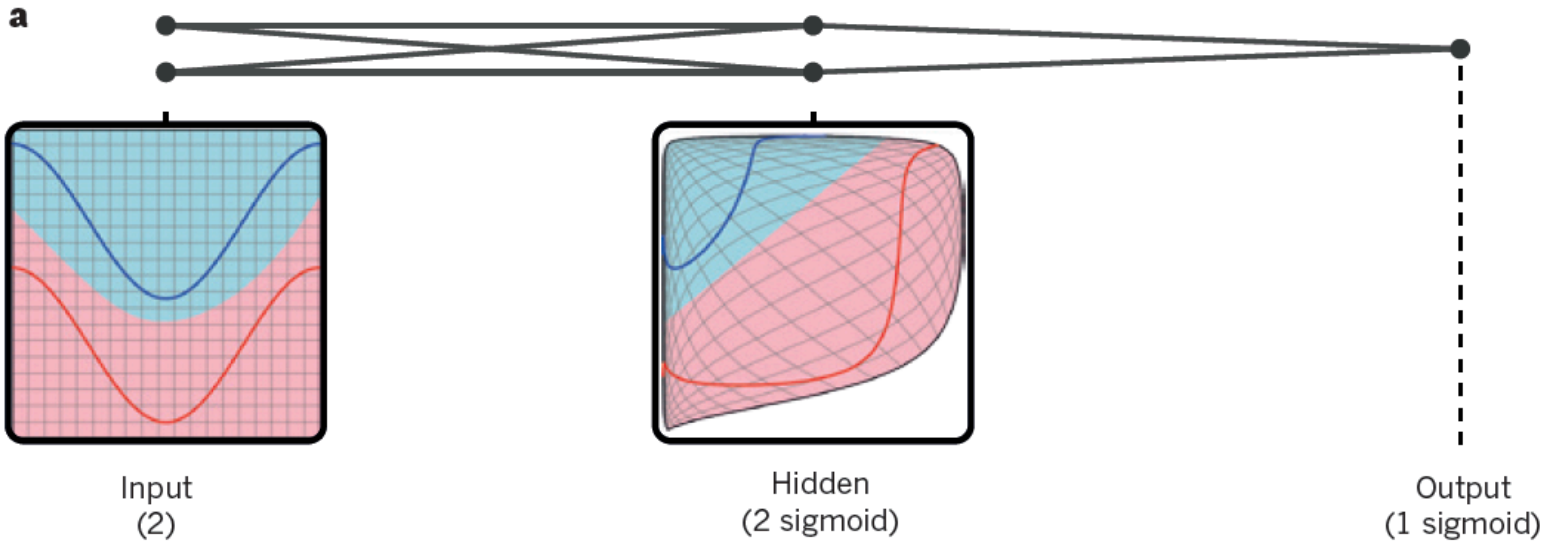
논문 리뷰 - Deep learning

- LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning.", Nature 2015
- https://www.cs.toronto.edu/~hinton/absps/NatureDeepReview.pdf
- 딥러닝 대가 3인이 소개하는 딥러닝

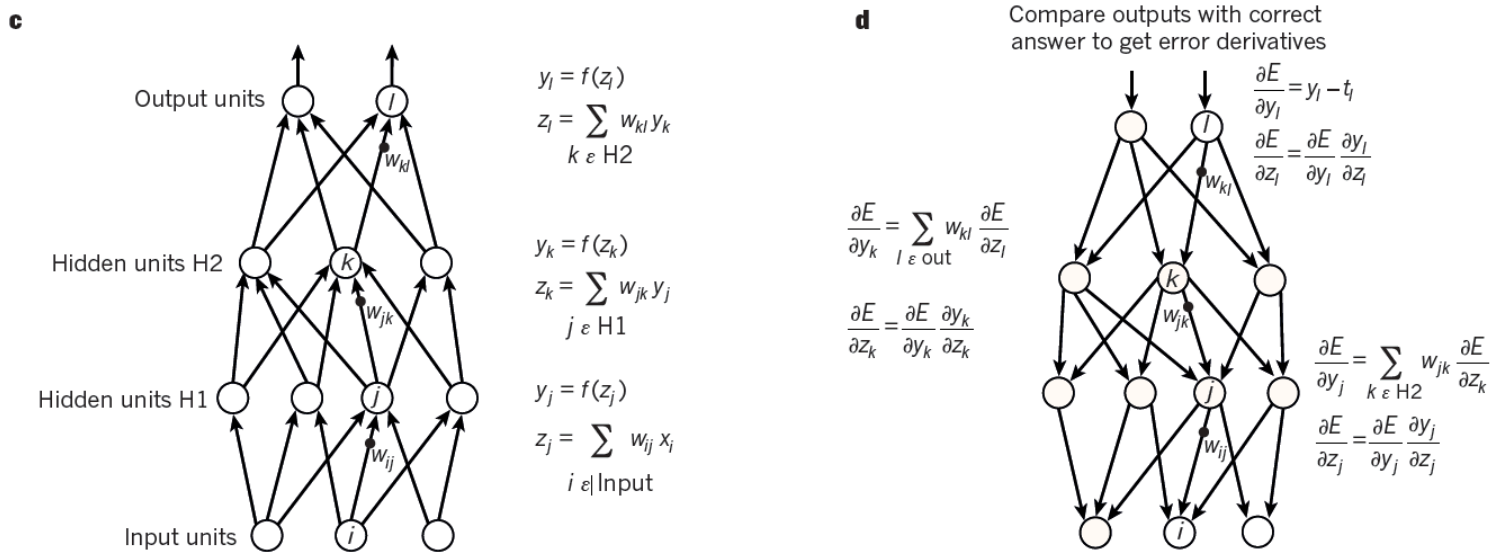

Distort input space

- **a**, A multi-layer neural network (shown by the connected dots) can **distort the input space** to make the classes of data (examples of which are on the red and blue lines) **linearly separable.** Note how a regular grid (shown on the left) in input space is also transformed (shown in the middle panel) by hidden units. This is an illustrative example with only two input units, two hidden units and one output unit, but the networks used for object recognition or natural language processing contain tens or hundreds of thousands of units.



Backpropagation Algorithm

- **d**, The equations used for computing the backward pass. At each hidden layer we compute the error derivative with respect to the output of each unit, which is a weighted sum of the error derivatives with respect to the total inputs to the units in the layer above. We then convert **the error derivative with respect to the output into the error derivative with respect to the input by multiplying it by the gradient of *f*(z).** At the output layer, the error derivative with respect to the output of a unit is computed by differentiating the cost function. This gives *yl* − *tl* if the cost function for unit *l* is 0.5(*yl* − *tl*)2, where *tl* is the target value. Once the *∂E*/*∂zk* is known, the error-derivative for the weight *wjk* on the connection from unit *j* in the layer below is just *yj ∂E*/*∂zk*.



Convolutional Neural Networks(CNNs)

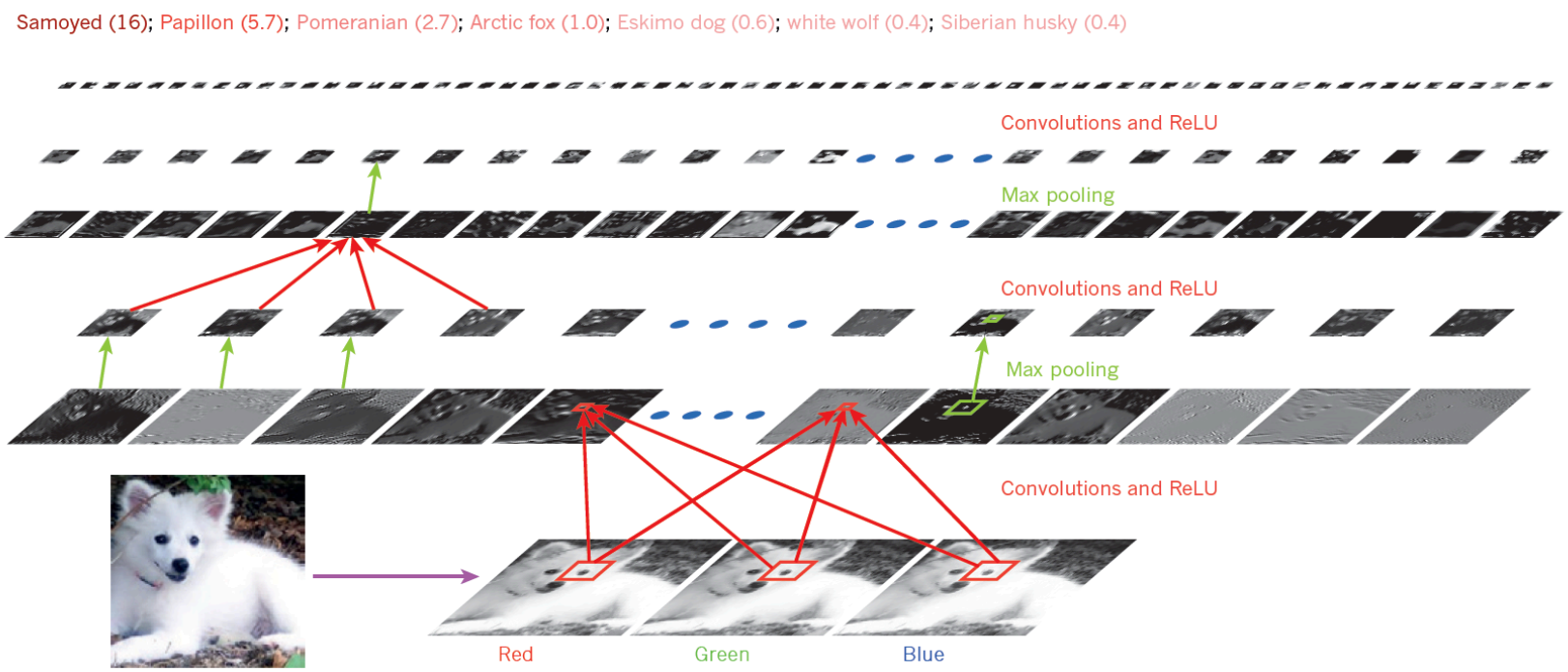


Figure 2 | Inside a convolutional network. The outputs (not the filters) of each layer (horizontally) of a typical convolutional network architecture applied to the image of a Samoyed dog (bottom left; and RGB (red, green, blue) inputs, bottom right). Each rectangular image is a feature map corresponding to the output for one of the learned features, detected at each of the image positions. Information flows bottom up, with lower-level features acting as oriented edge detectors, and a score is computed for each image class in output. ReLU, rectified linear unit.

Image Captioning

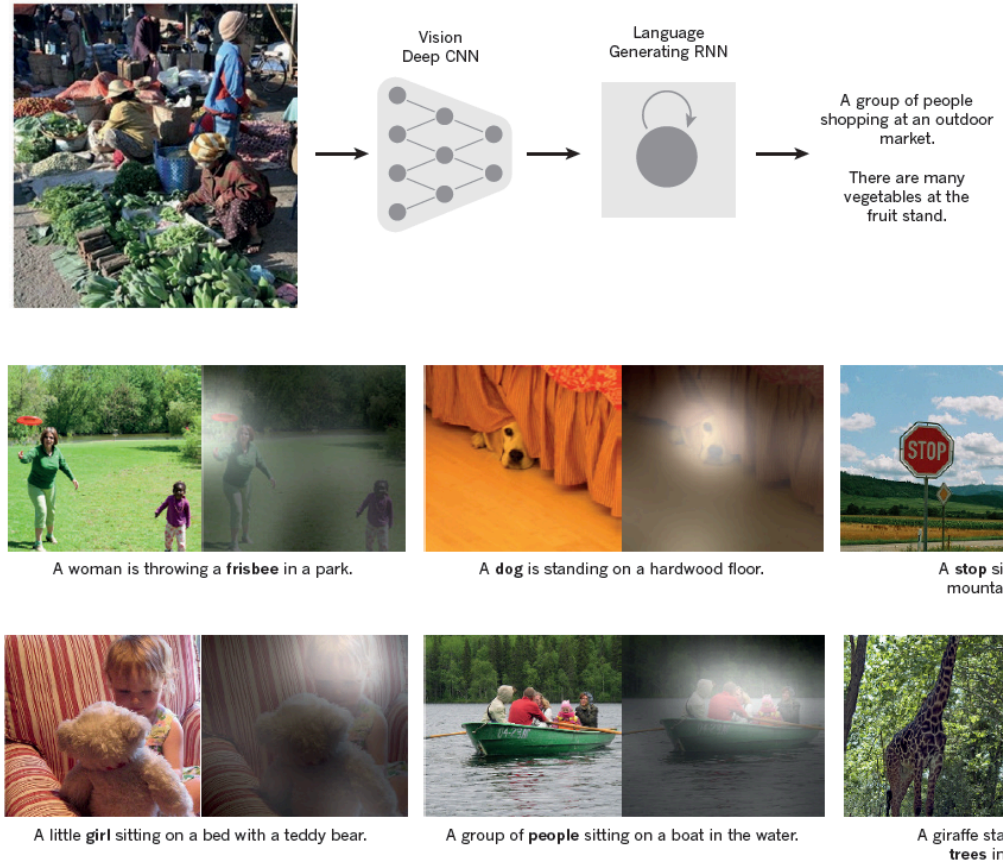
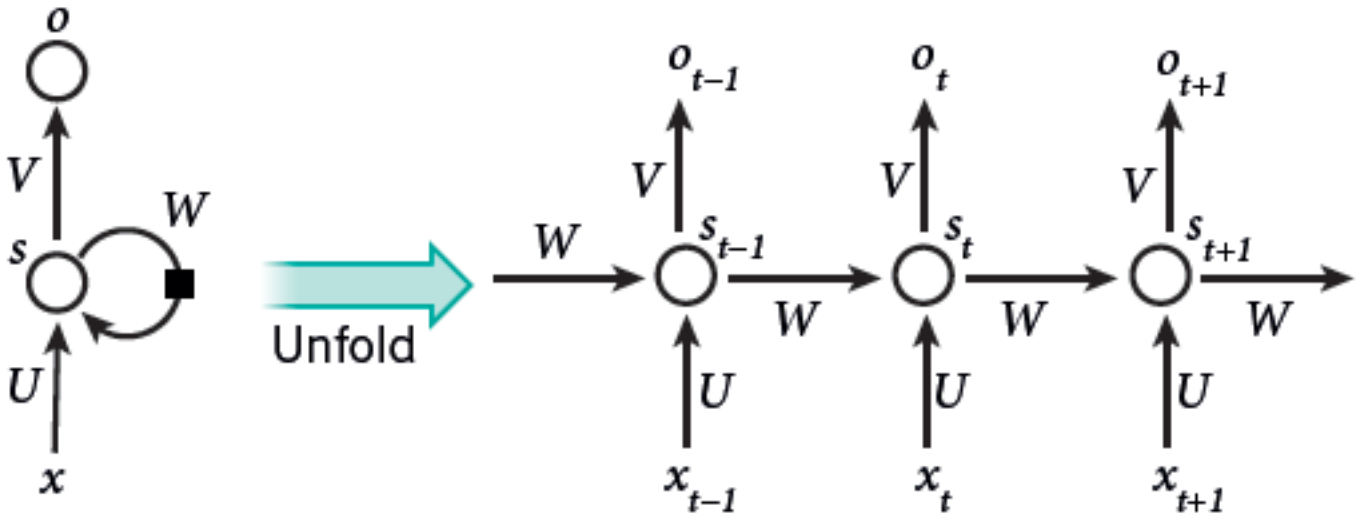


Figure 3 | From image to text. Captions generated by a recurrent neural network (RNN) taking, as extra input, the representation extracted by a deep convolution neural network (CNN) from a test image, with the RNN trained to 'translate' high-level representations of images into captions (top). Reproduced with permission from ref. 102. When the RNN is given the ability to focus its attention on a different location in the input image (middle and bottom; the lighter patches were given more attention) as it generates each word (bold), we found[86] that it exploits this to achieve better 'translation' of images into captions.

Recurrent Neural Networks(RNNs)

- **A recurrent neural network and the unfolding in time of the computation involved in its forward computation.** The artificial neurons (for example, hidden units grouped under node *s* with values *st* at time *t*) get inputs from other neurons at previous time steps (this is represented with the black square, representing a delay of one time step, on the left). In this way, a recurrent neural network can map an input sequence with elements *xt* into an output sequence with elements *ot*, with each *ot* depending on all the previous *xt'* (for $t' \leq t$). The same parameters (matrices *U*,*V*,*W*) are used at each time step. Many other architectures are possible, including a variant in which the network can generate a sequence of outputs (for example, words), each of which is used as inputs for the next time step.



Softmax Function

- Classification 문제를 풀때, Neural Networks의 마지막 Layer에 주로 사용되는 Activation function 중 하나.
- 모두 더하면 1이되는 normalized 된 함수이다.
- **Softmax Function의 Output은 각각의 Label에 대해 예측한 확률값이 된다.** 예를 들어, 어떤 사진이 [개, 고양이, 호랑이]인지를 분류하는 Classification 문제의 Softmax Function의 출력값이 [0.7, 0.2, 0.1]이라면 우리가 만든 분류기가 이 사진이 개일 확률을 70%, 고양이인 확률은 20%, 호랑이일 확률은 10%로 예측했다고 해석할 수 있다.

$$softmax(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j)}$$

Softmax Regression

- Softmax Regression은 어떤 input x가 주어졌을때 그것이 class i일거라고 확신하는 정도(evidence)를 아래의 식으로 나타낸다.

$$evidence_i = \sum_j W_{i,j} x_j + b_i$$

- 위의 evidence를 softmax function을 통해 프로그램이 레이블(label)을 y라고 예측할 확률로 바꾼다.

$$y = softmax(evidence)$$

- 위 식에서 softmax 함수 일종의 link 함수로써 선형함수의 결과(evidence)를 우리가 원하는 형태로 바꾸어준다.(MNIST 예제의 경우, 10개의 case(0~9)에 대한 확률 분포) 이를 거칠게 수식으로 나타내면 아래와 같다.

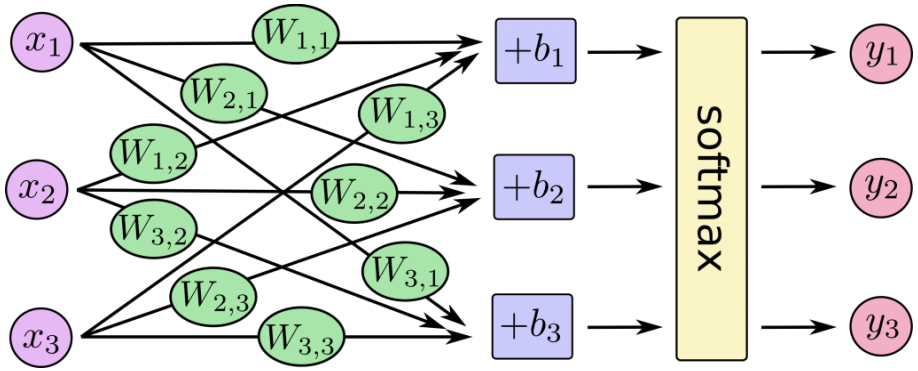
$$softmax(x) = normalize(exp(x))$$

$$softmax(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j)}$$

Softmax Regression

- Softmax Regression 과정을 그림으로 나타내면 아래와 같다.

$$y = softmax(Wx + b)$$



$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

Cross-entropy Cost Function

- 이제 우리가 만든 softmax regression 모델이 잘 학습하고 있는지를 평가하는 evaluation function을 정의해야 한다. 가장 대표적으로 쓰이는 evaluation function은 cross-entropy이다. cross-entropy는 아래와 같이 정의된다.

$$H_{y'}(y) = -\sum_i y'_i log(y_i)$$

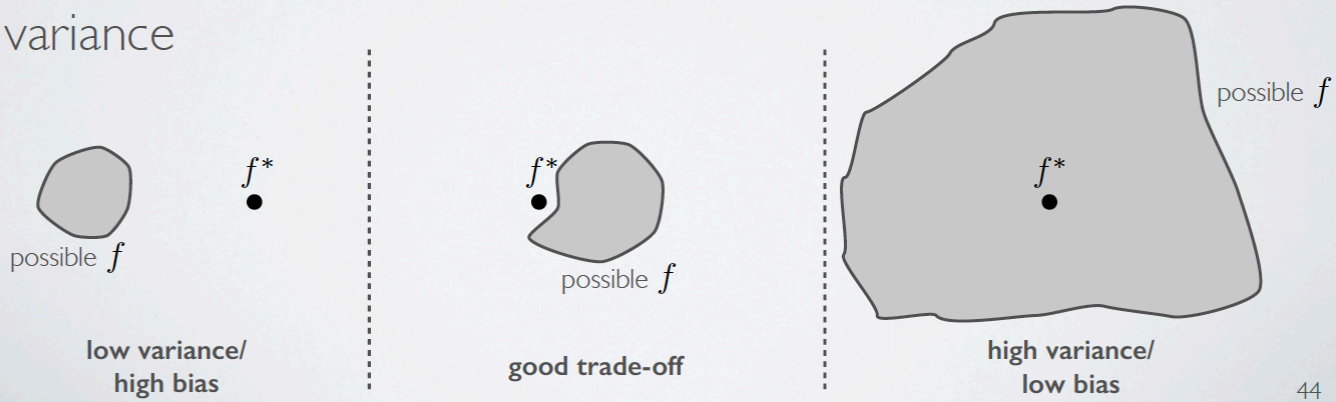
- cross-entropy를 간단히 설명하면 **모델의 예측값(prediction)이 실제 참값(truth)을 설명하는데 얼마나 비효율적인지(inefficient)**를 나타낸다. 따라서, cross-entropy가 낮을수록 좋은 모델이다. 즉, 우리는 cross-entropy를 최소화하는 방향으로 학습을 진행한다.

학습에서의 bias-variance tradeoff

MACHINE LEARNING

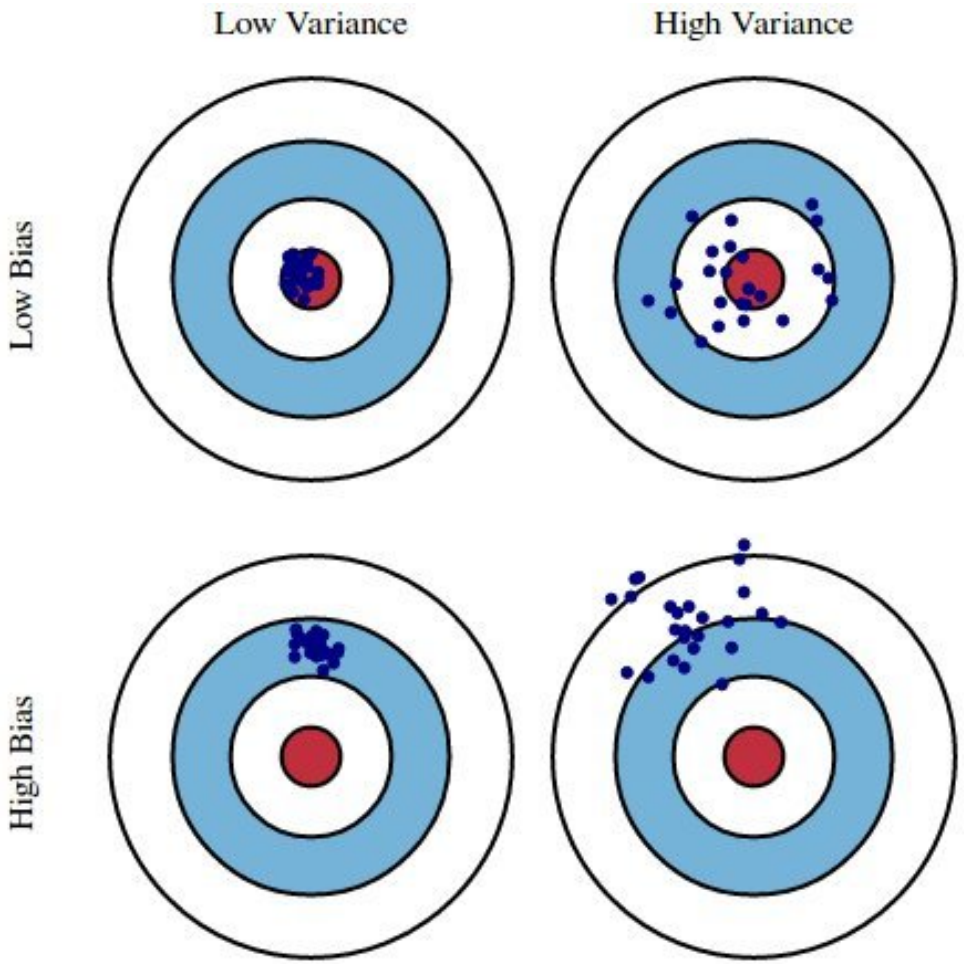
Topics: bias-variance trade-off

- **Variance** of trained model: does it vary a lot if the training set changes
- **Bias** of trained model: is the average model close to the true solution?
- Generalization error can be seen as the sum of bias and the variance



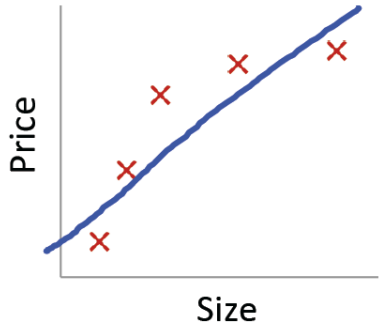
44

학습에서의 bias-variance tradeoff



학습에서의 bias-variance tradeoff

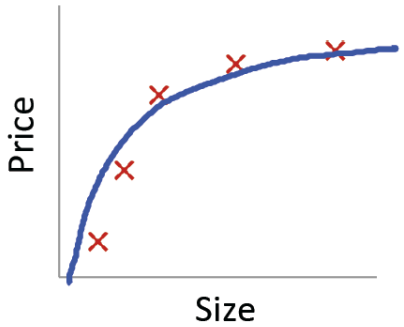
Bias/variance



$\theta_0 + \theta_1 x$

High bias
(underfit)

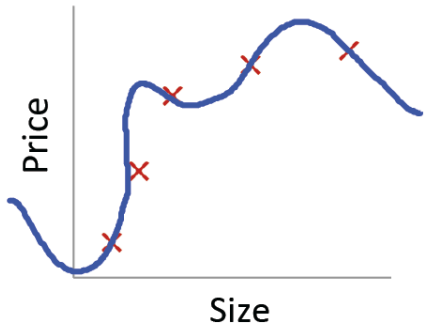
$d=1$



$\theta_0 + \theta_1 x + \theta_2 x^2$

"Just right"

$d=2$



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

High variance
(overfit)

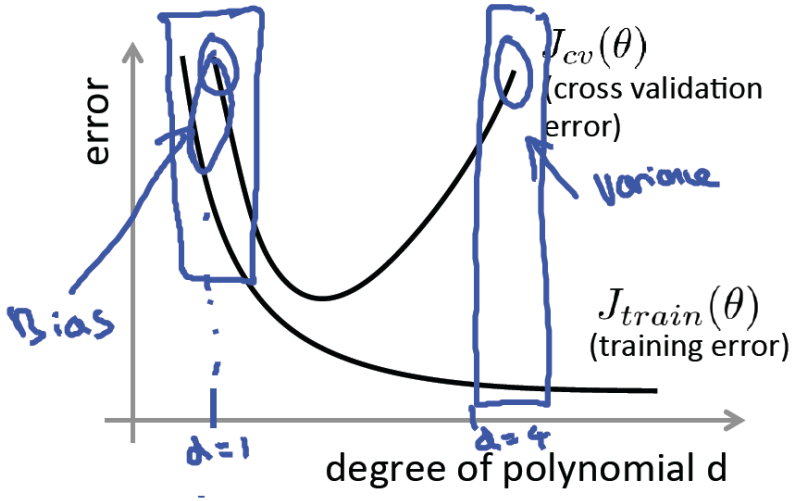
$d=4$

Andrew Ng

학습에서의 bias-variance tradeoff

Diagnosing bias vs. variance

Suppose your learning algorithm is performing less well than you were hoping. ($J_{cv}(\theta)$ or $J_{test}(\theta)$ is high.) Is it a bias problem or a variance problem?



Bias (underfit):

→ $J_{train}(\theta)$ will be high

$J_{cv}(\theta) \approx J_{train}(\theta)$

Variance (overfit):

→ $J_{train}(\theta)$ will be low

$J_{cv}(\theta) \gg J_{train}(\theta)$

>>

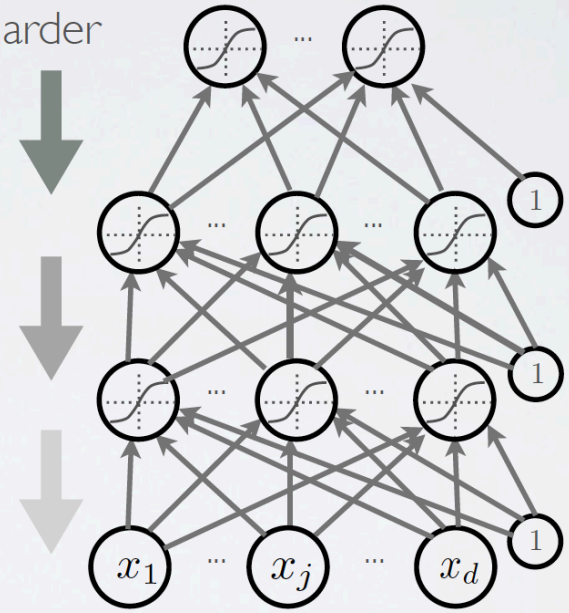
Andrew Ng

bias-variance tradeoff 관점에서 본 Deep Learning

DEEP LEARNING

Topics: why training is hard

- First hypothesis: optimization is harder (underfitting)
  - vanishing gradient problem
  - saturated units block gradient propagation
- This is a well known problem in recurrent neural networks

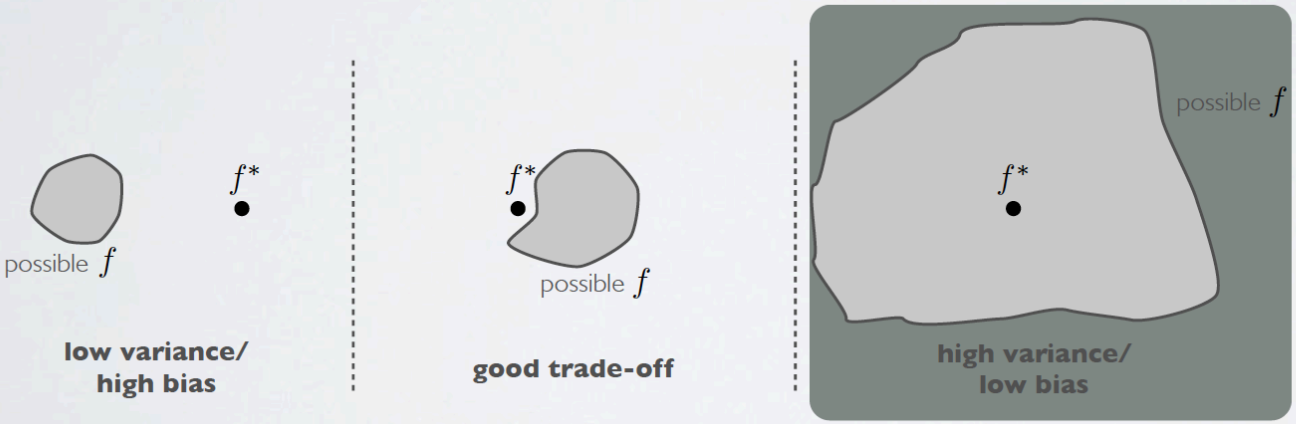


bias-variance tradeoff 관점에서 본 Deep Learning

DEEP LEARNING

Topics: why training is hard

- Second hypothesis: overfitting
  - we are exploring a space of complex functions
  - deep nets usually have lots of parameters
- Might be in a high variance / low bias situation



bias-variance tradeoff 관점에서 본 Deep Learning

DEEP LEARNING

Topics: why training is hard

- Depending on the problem, one or the other situation will tend to dominate

- If first hypothesis (underfitting): better optimize
  - use better optimization methods
  - use GPUs

- If second hypothesis (overfitting): use better regularization
  - unsupervised pre-training
  - stochastic «dropout» training

TensorFlow를 사용하는 이유

Why TensorFlow?

- Python API
- Portability: deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API
- Flexibility: from Raspberry Pi, Android, Windows, iOS, Linux to server farms
- Visualization (TensorBoard is da bomb)
- Checkpoints (for managing experiments)
- Auto-differentiation *autodiff* (no more taking derivatives by hand. Yay)
- Large community (> 10,000 commits and > 3000 TF-related repos in 1 year)
- Awesome projects already using TensorFlow

10

Slide credit: CS 20SI
(https://goo.gl/Ez8wRq)

TensorFlow에서의 데이터 표현

- TensorFlow는 데이터를 Graph 형태(Node + Edge)로 나타낸다.

Slide credit: CS 20SI
(https://goo.gl/Ez8wRq)

- **Node** : operators, variables, and constants를 나타낸다.

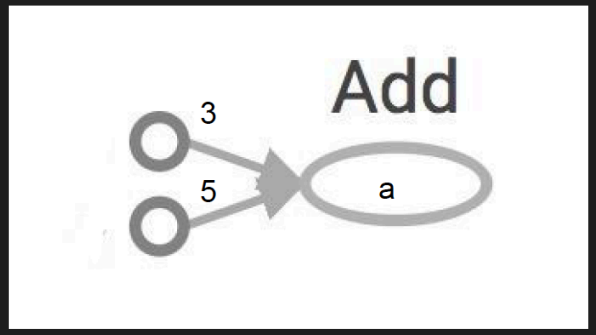
  Edge : tensors를 나타낸다.

Data Flow Graphs

```
import tensorflow as tf

a = tf.add(3, 5)
```

Interpreted?



Nodes: operators, variables, and constants
Edges: tensors

Tensors are data.
Data Flow -> Tensor Flow (I know, mind=blown)

32

Tensor = n차원 행렬

What's a tensor?

An n-dimensional array

0-d tensor: scalar (number)

1-d tensor: vector

2-d tensor: matrix

and so on

29

Slide credit: CS 20SI
(https://goo.gl/Ez8wRq)

TensorFlow 실행과정

1. 그래프(Graph) 생성

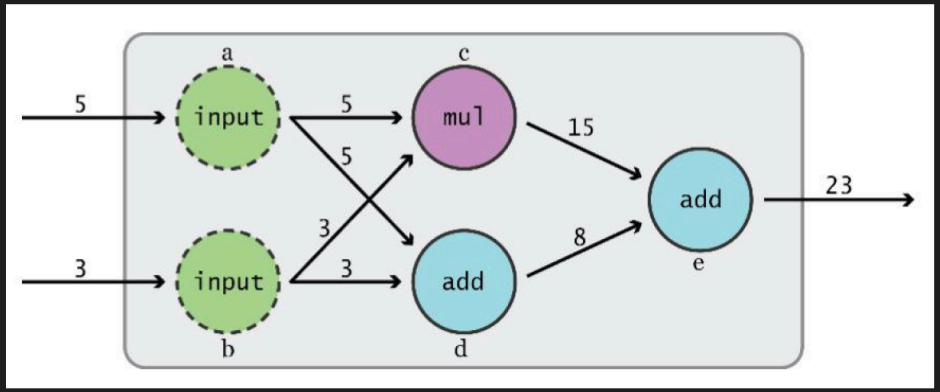
Slide credit: CS 20SI
(https://goo.gl/Ez8wRq)

2. Session을 이용한 그래프 상에 연산(operation) 실행

Data Flow Graphs

Phase 1: assemble a graph

Phase 2: use a session to execute operations in the graph.



Graph by TFFMI

27

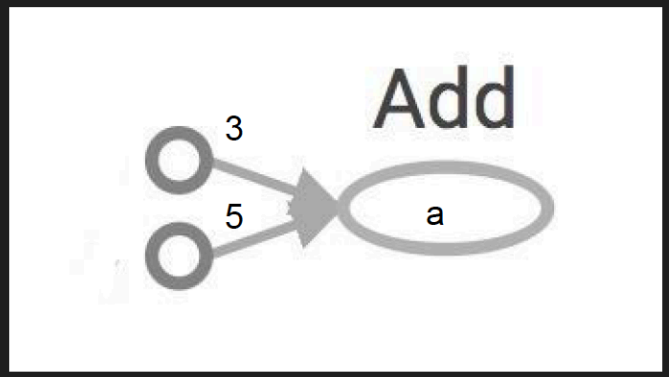
Session을 실행하기 전

Data Flow Graphs

```
import tensorflow as tf

a = tf.add(3, 5)

print  a
```




```
>> Tensor("Add:0", shape=(), dtype=int32)
```

(Not 8)

33

Slide credit: CS 20SI
(https://goo.gl/Ez8wRq)

Session을 이용한 실행

How to get the value of a?

Create a **session**, assign it to variable sess so we can call it later

Within the session, evaluate the graph to fetch the value of a

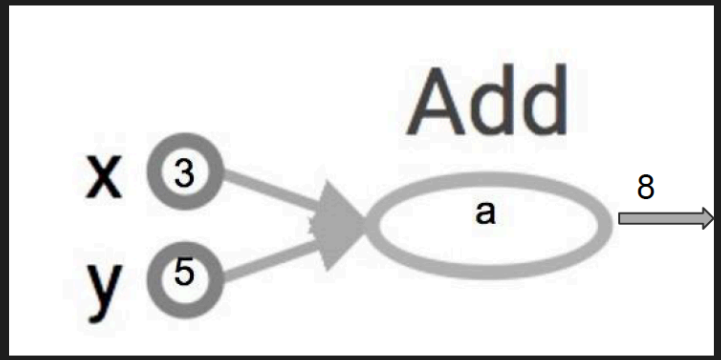
```
import tensorflow as tf

a = tf.add(3, 5)

sess = tf.Session()

print sess.run(a)        >> 8

sess.close()
```



The session will look at the graph, trying to think: hmm, how can I get the value of a, then it computes all the nodes that leads to a.

36

Slide credit: CS 20SI (https://goo.gl/Ez8wRq)

MNIST Database

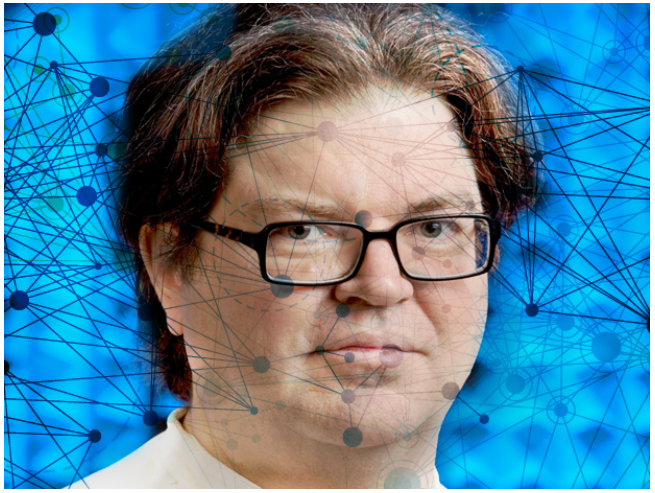
- MNIST 숫자 분류 = Machine Learning에서의 Hello World!

MNIST : Mixed National Institute of Standards and Technology database

- MNIST Database :

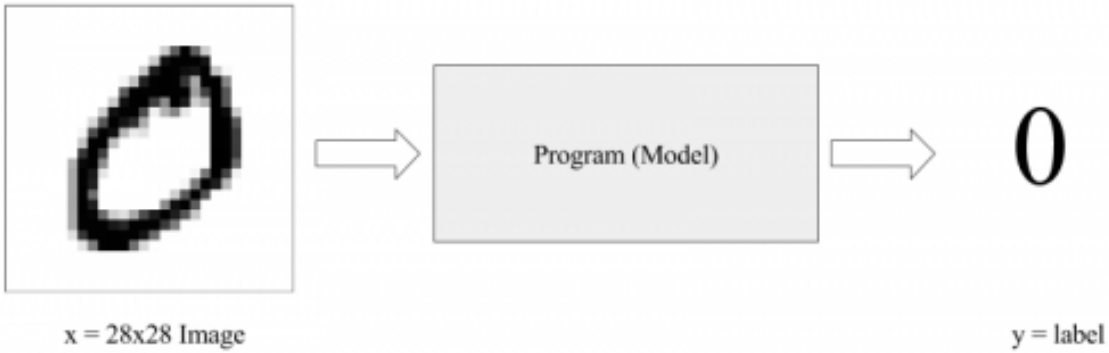
Image : 28x28 크기의 숫자 이미지

Labels : 0~9 숫자



MNIST 숫자 분류 문제 소개

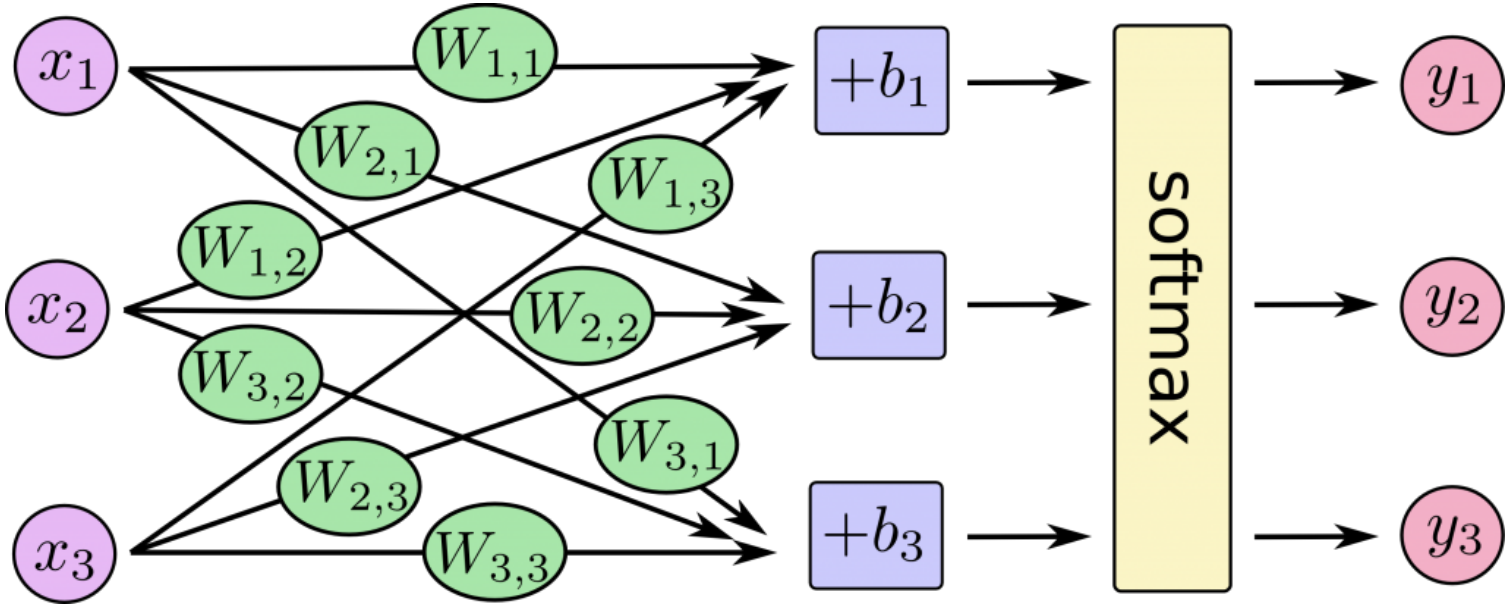
- 해결하고자 하는 문제 : MNIST 숫자 분류



MNIST 숫자분류를 위한 Softmax Classifier

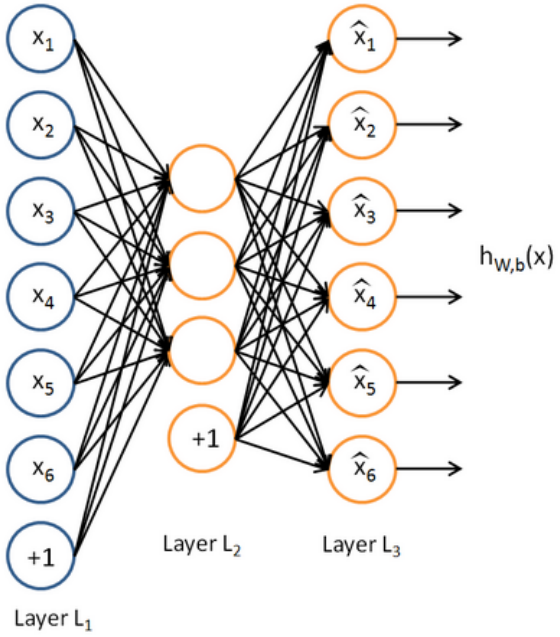
- https://github.com/solaris33/dl_cv_tensorflow_10weeks/blob/master/week1/mnist_classification_with_softmax_classifier.py

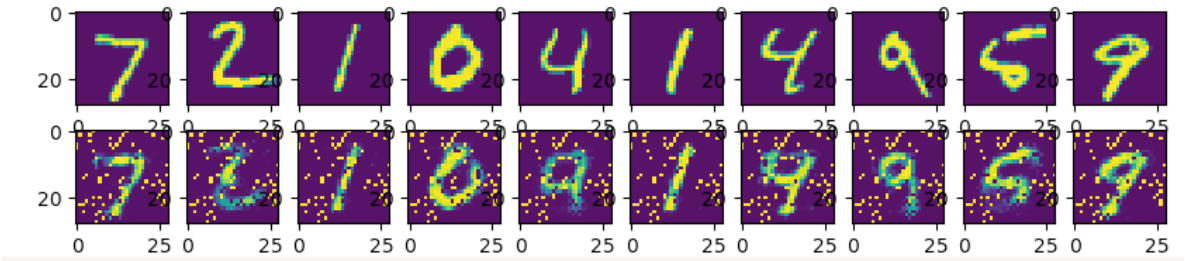
- 실험결과 : 약 **91.8%** 정확도



Autoencoder를 이용한 MNIST Reconstruction

- https://github.com/solaris33/dl_cv_tensorflow_10weeks/blob/master/week1/mnist_autoencoder_reconstruction.py

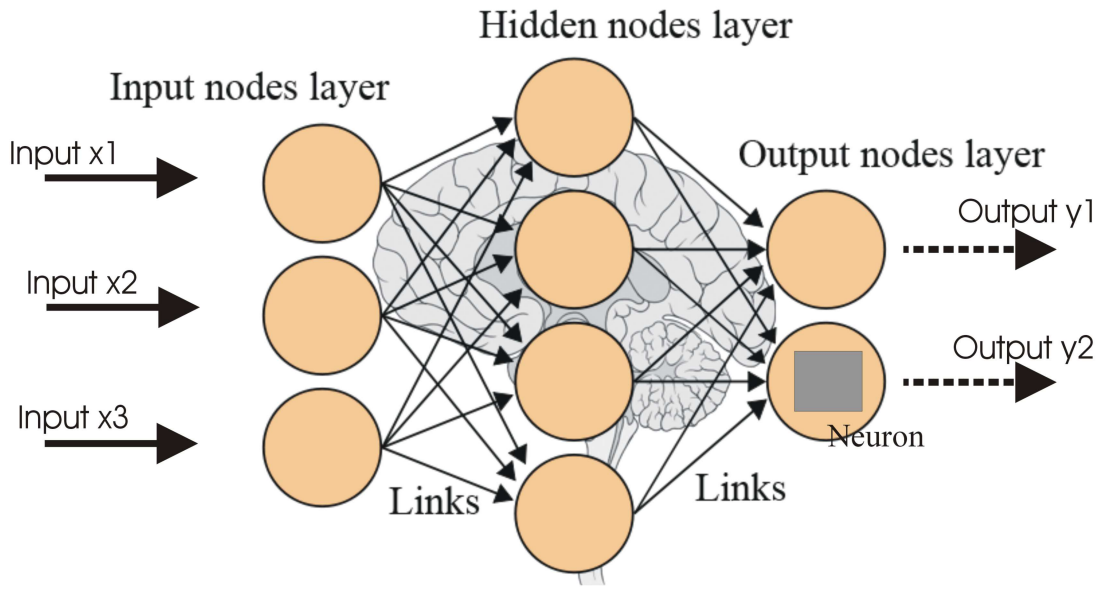




MNIST 숫자분류를 위한 Multi-Layer Perceptron(MLP)

- https://github.com/solaris33/dl_cv_tensorflow_10weeks/blob/master/week1/mnist_classification_with_MLP.py

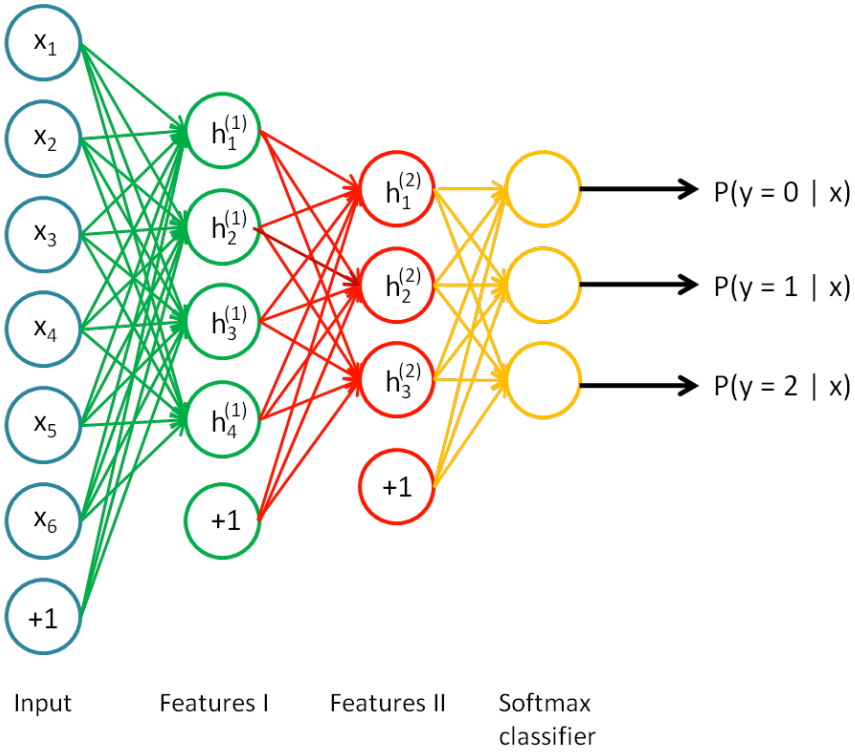
- 실험결과 : 약 **94.5%** 정확도



MNIST 숫자분류를 위한 Stacked Autoencoder + Softmax Classifier

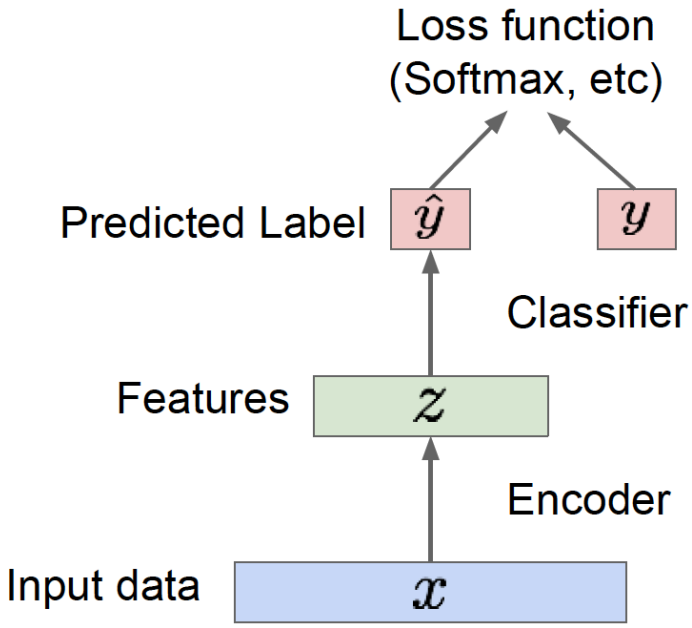
- https://github.com/solaris33/dl_cv_tensorflow_10weeks/blob/master/week1/mnist_classification_with_stacked_autoencoders_and_softmax_classifier.py

- 실험결과 : 약 **96.6%** 정확도



Supervised Autoencoder

Some background first: Autoencoders

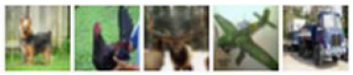


Encoder can be used to initialize a **supervised** model

Fine-tune encoder jointly with classifier

bird plane
dog deer truck

Train for final task (sometimes with small data)

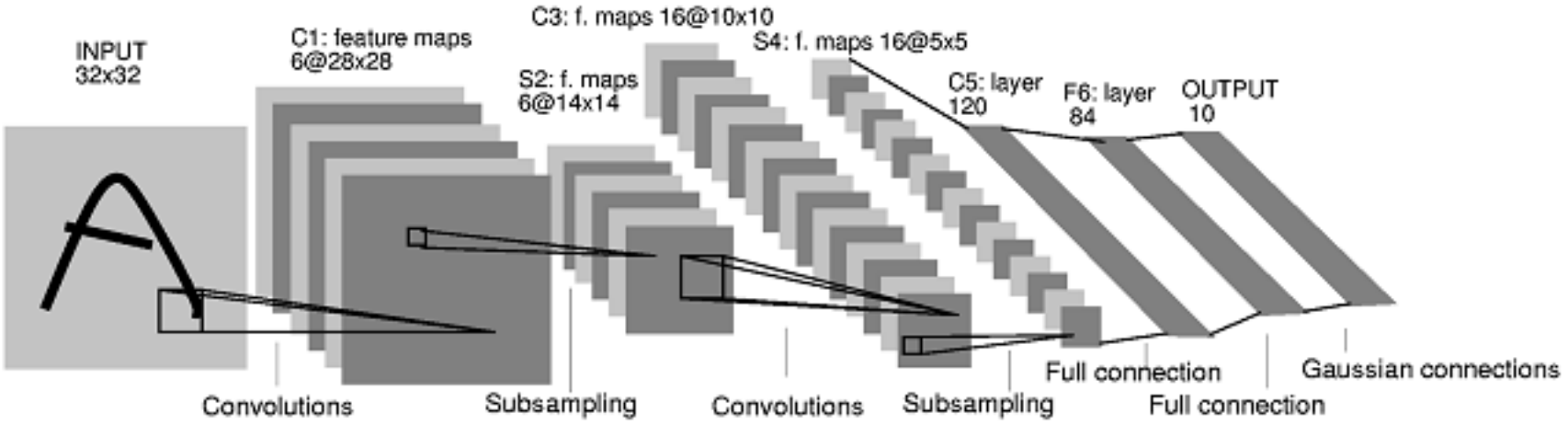


Slide credit: CS231n (https://goo.gl/QSA36t)

MNIST 숫자분류를 위한 Convolutional Neural Networks(CNNs)

- https://github.com/solaris33/dl_cv_tensorflow_10weeks/blob/master/week1/mnist_classification_with_convolutional_neural_networks.py

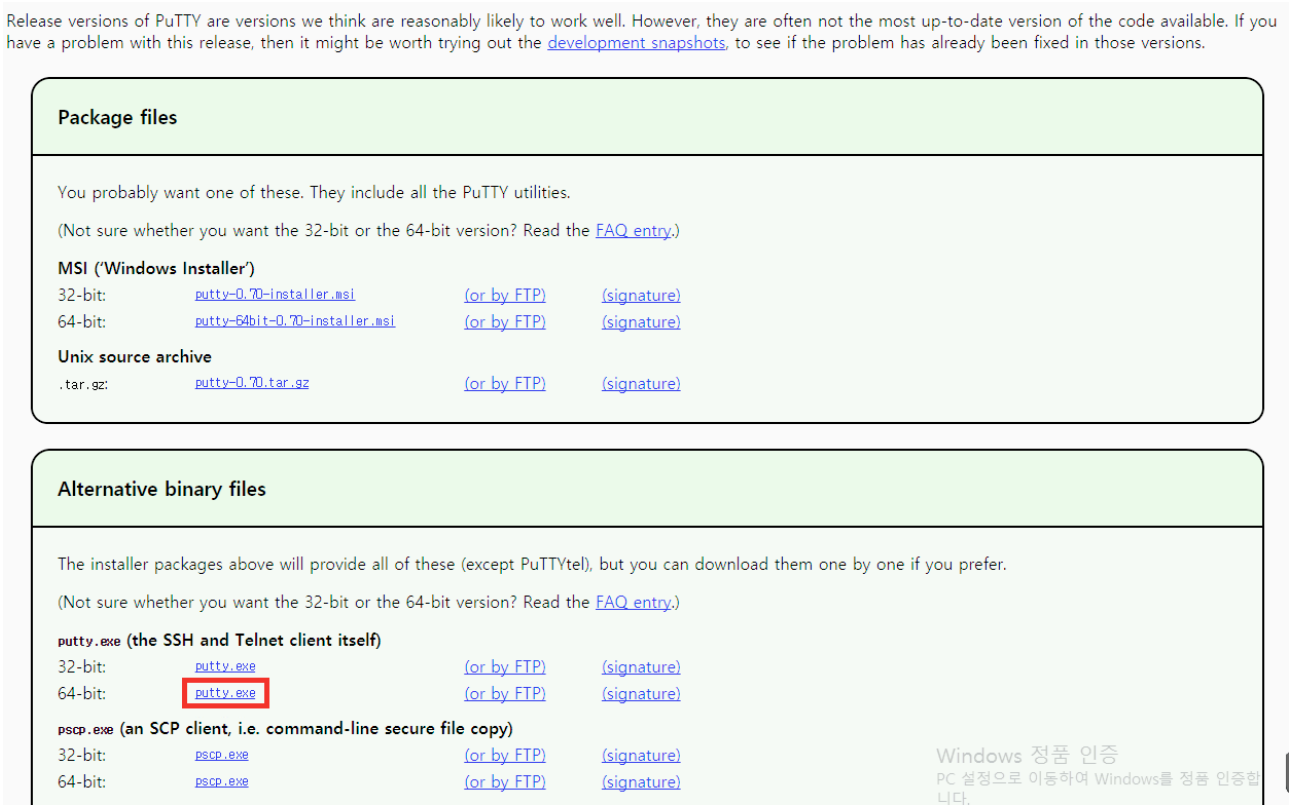
- 약 **99.1%** 정확도



Appendix A – Putty를 이용한 AWS 접속 (Windows)

- 아래 링크에서 Putty Binary 파일을 다운 받는다.

- https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html



Release versions of PuTTY are versions we think are reasonably likely to work well. However, they are often not the most up-to-date version of the code available. If you have a problem with this release, then it might be worth trying out the development snapshots, to see if the problem has already been fixed in those versions.

Package files

You probably want one of these. They include all the PuTTY utilities.

(Not sure whether you want the 32-bit or the 64-bit version? Read the FAQ entry.)

MSI ('Windows Installer')

32-bit:	putty-0.70-installer.msi	(or by FTP)	(signature)
64-bit:	putty-64bit-0.70-installer.msi	(or by FTP)	(signature)

Unix source archive

.tar.gz:	putty-0.70.tar.gz	(or by FTP)	(signature)

Alternative binary files

The installer packages above will provide all of these (except PuTTYtel), but you can download them one by one if you prefer.

(Not sure whether you want the 32-bit or the 64-bit version? Read the FAQ entry.)

putty.exe (the SSH and Telnet client itself)

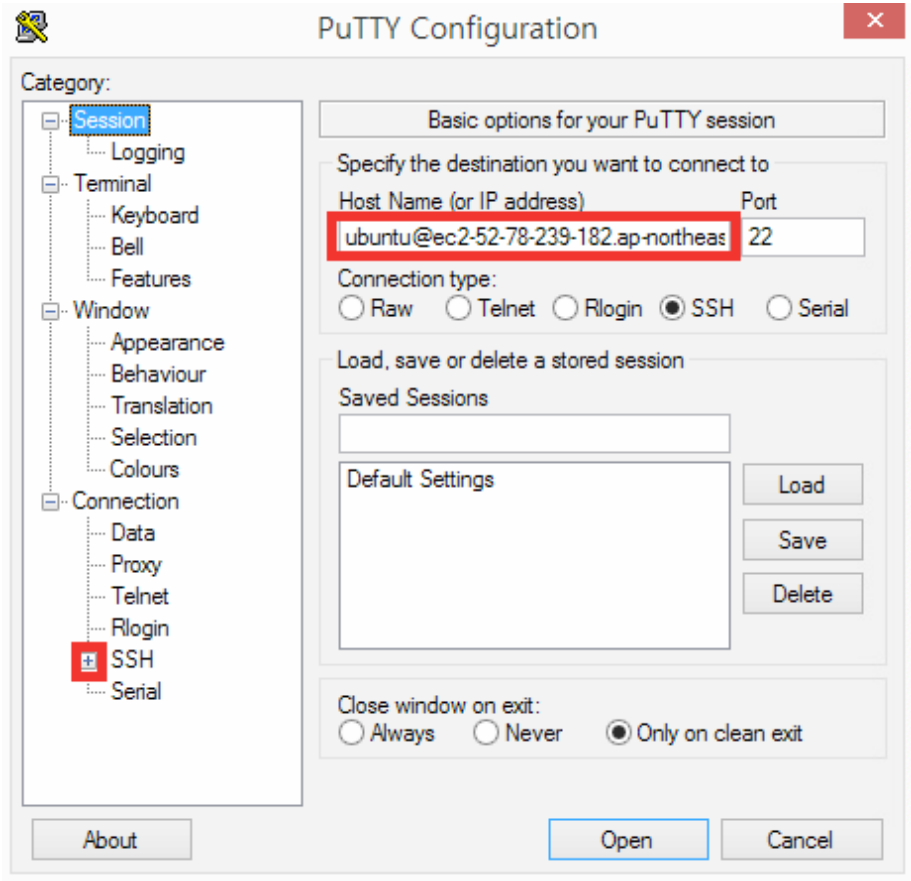
32-bit:	putty.exe	(or by FTP)	(signature)
64-bit:	putty.exe	(or by FTP)	(signature)

pscp.exe (an SCP client, i.e. command-line secure file copy)

32-bit:	pscp.exe	(or by FTP)	(signature)
64-bit:	pscp.exe	(or by FTP)	(signature)

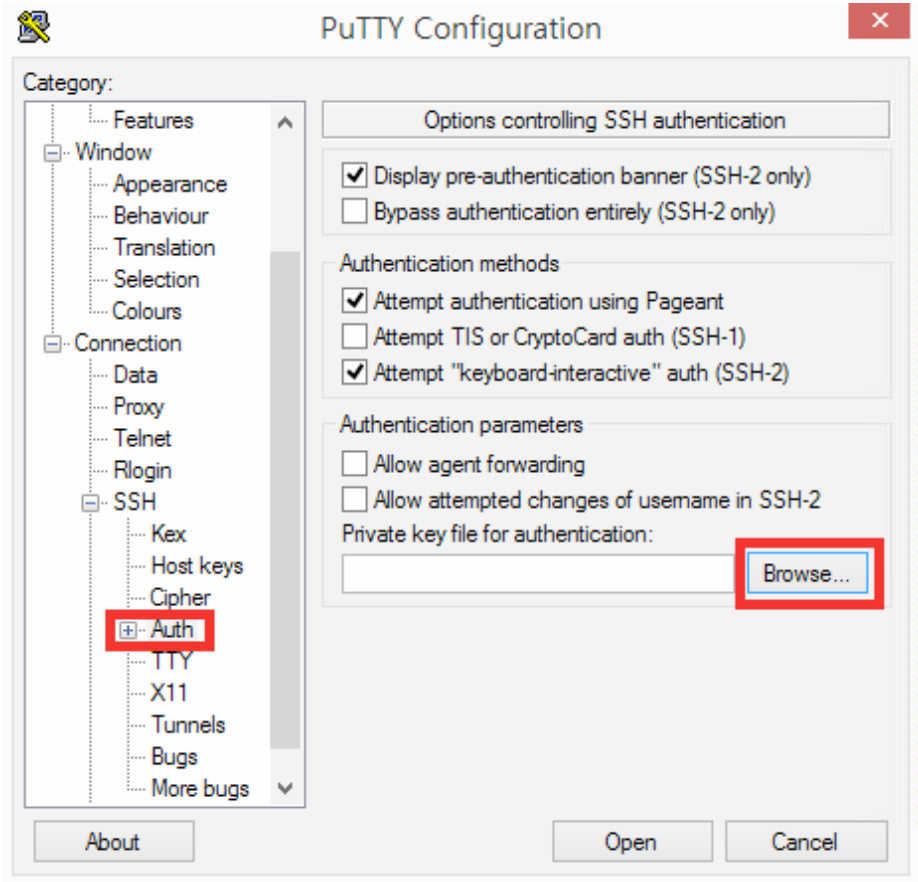
Appendix A – Putty를 이용한 AWS 접속 (Windows)

- 다운 받은 putty.exe를 실행하고 Host Name을 입력한다.
- SSH 왼쪽에 +버튼을 클릭한다.



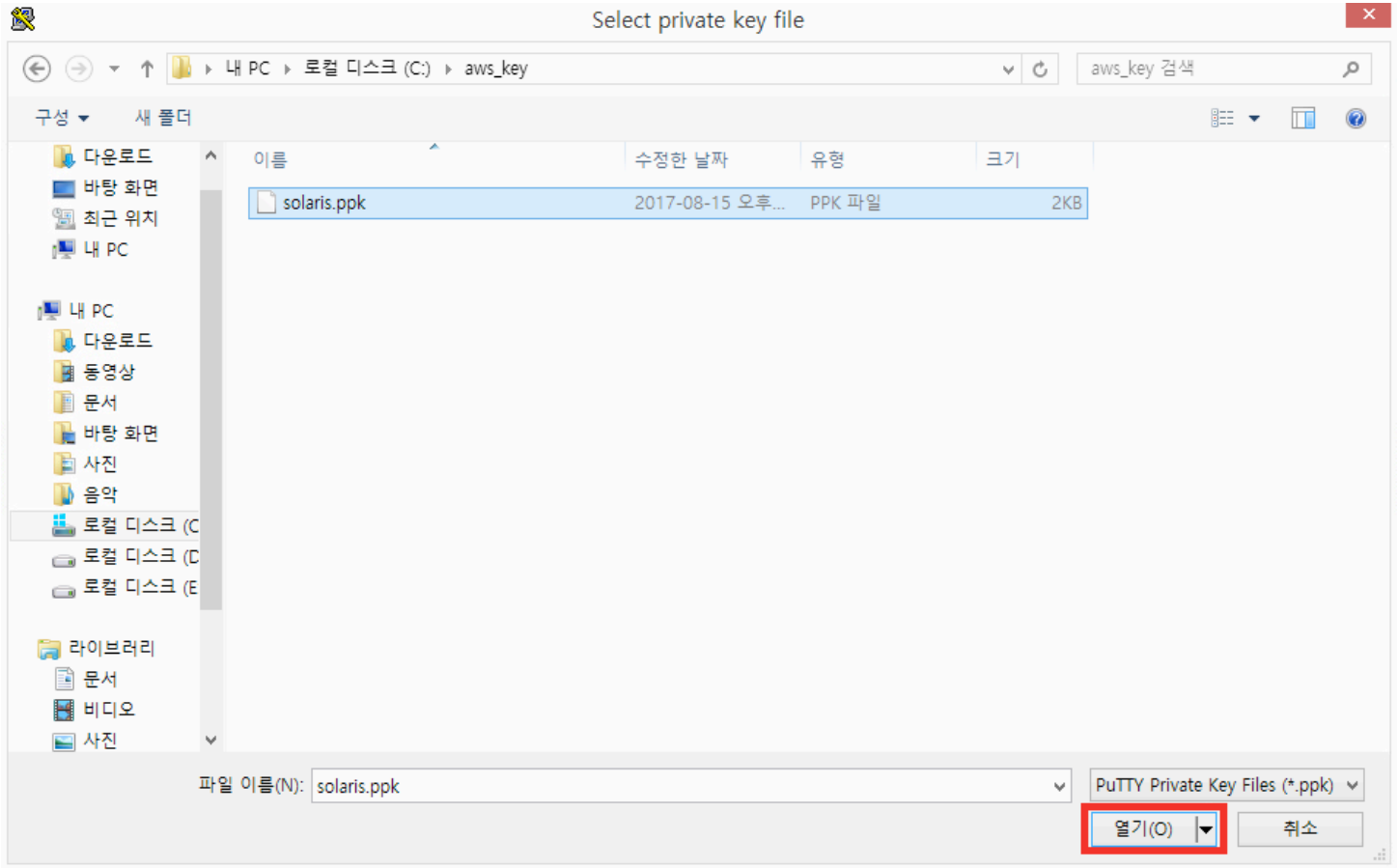
Appendix A – Putty를 이용한 AWS 접속 (Windows)

- Connection-SSH-Auth 메뉴로 들어간다.
- Private key file for authentication에 Browse 버튼을 클릭한다.



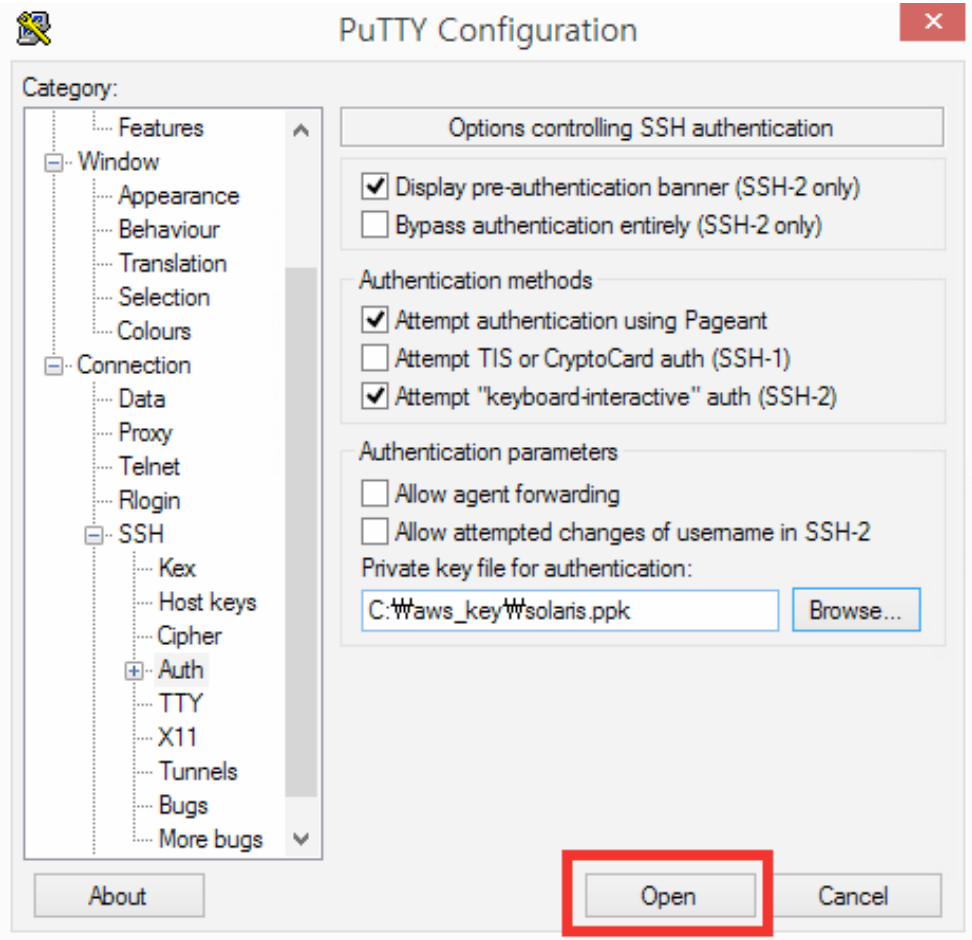
Appendix A – Putty를 이용한 AWS 접속 (Windows)

- solaris.ppk 파일을 선택한다.



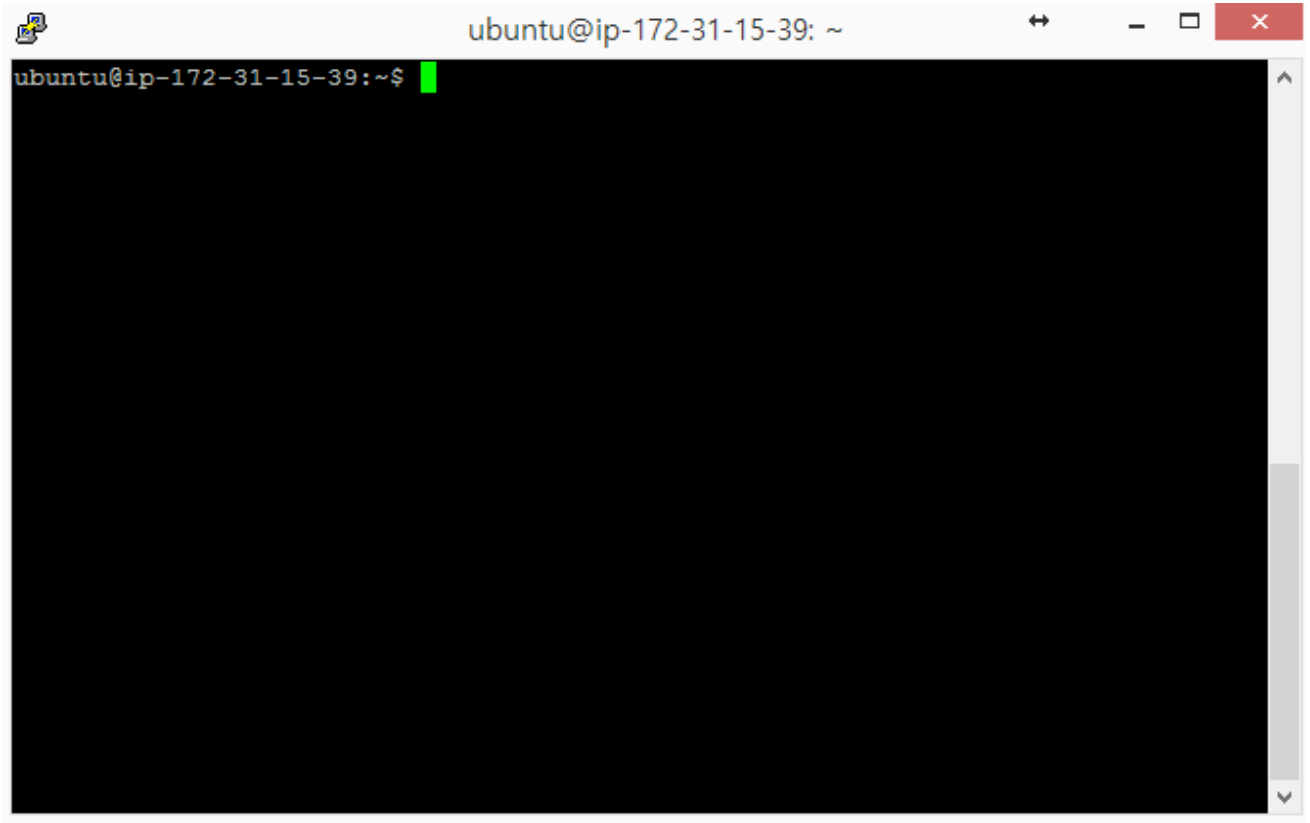
Appendix A – Putty를 이용한 AWS 접속 (Windows)

- solaris.ppk를 추가했으면 이제 Open을 클릭한다.



Appendix A – Putty를 이용한 AWS 접속 (Windows)

- Security Alert 창이 뜨는데 예(Y)를 누르면 된다.
- 그럼 아래와 같이 AWS에서 제공하는 Ubuntu 서버에 접속된 모습을 볼 수 있다.



Appendix B - MSE Cost Function vs Cross-Entropy Cost Function

- Minimum Squared Error(MSE) Cost Function은 다음과 같이 정의 된다.

$$J(W,b;x,y) = \frac{1}{2}\frac{1}{n}\sum_{x}(\sigma(z)-y)^2$$

- MSE Cost Function을 사용했을때 파라미터 W,b 미분값(derivative)은 아래와 같다.

$$\frac{\partial}{\partial W_{ij}}J(W,b;x,y) = \frac{1}{n}\sum_{x}(\sigma(x)-y)\sigma'(z)x_{ij}$$

$$\frac{\partial}{\partial b_{ij}}J(W,b;x,y) = \frac{1}{n}\sum_{x}(\sigma(x)-y)\sigma'(z)$$

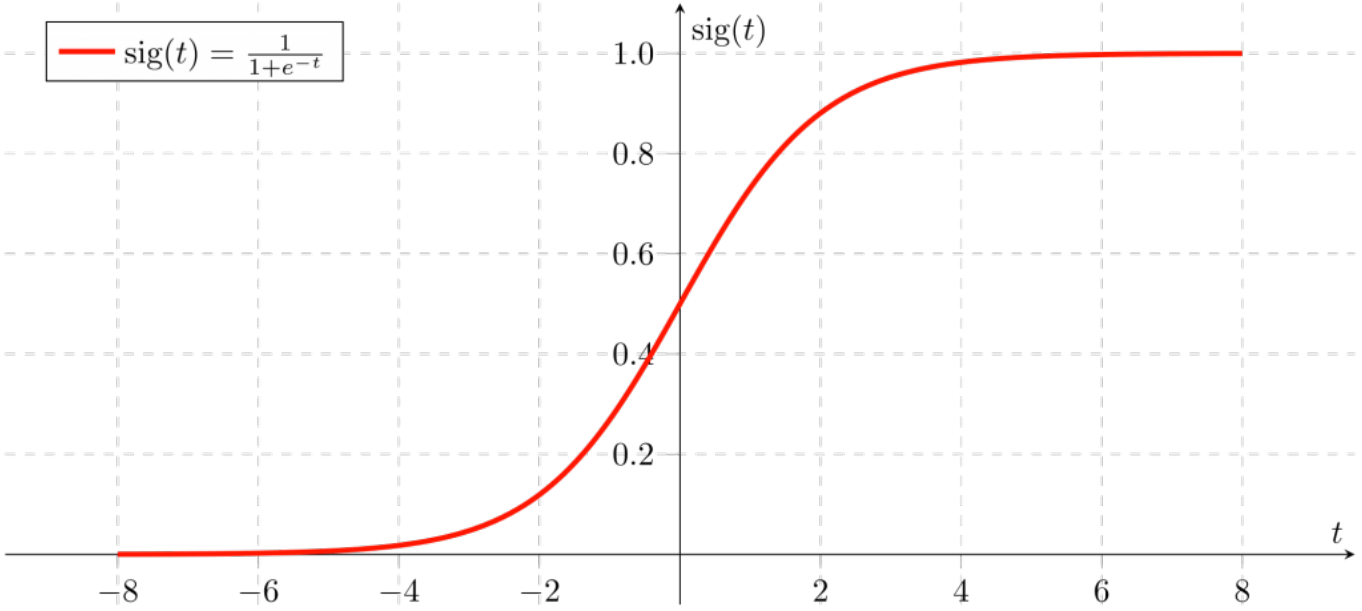
- 이때 문제점은 sigmoid function을 activation function-σ-으로 쓸 경우, z값이 너무 크거나 작아지면 미분값-$\sigma'(z)$-이 급격히 0에 가까워지는 현상이 발생한다. 따라서 이는 **학습속도가 저하(slowdown)되는 문제**를 유발하게 된다.

Appendix B - MSE Cost Function vs Cross-Entropy Cost Function

- Sigmoid Function

$$S(x) = \frac{1}{1 + e^{-x}}$$

- 인풋 값이 너무 커지거나 작아지면 해당 구간에서의 미분값(Gradient)가 0이되는 문제가 있다. (그림에서 약 -6이하, 6이상인 구간)



Appendix B - MSE Cost Function vs Cross-Entropy Cost Function

- Binary Classificiation일 경우 Cross-Entropy Cost Function은 다음과 같이 정의 된다.

$$J(W,b;x,y) = -\frac{1}{n}\sum_x [ylna + (1-y)ln(1-a)]$$

- Cross-Entropy Cost Function을 사용했을때 파라미터 W,b 미분값(derivative)은 아래와 같다.

$$\frac{\partial}{\partial W_{ij}} J(W,b;x,y) = \frac{1}{n}\sum_x x_{ij}(\sigma(z) - y)$$

$$\frac{\partial}{\partial b_{ij}} J(W,b;x,y) = \frac{1}{n}\sum_x (\sigma(z) - y)$$

- 위 식에서 파라미터 Weight와 Bias의 미분값(Derivative)은 **예측값과 실제값의 차이-$(\sigma(z) - y)$-에 비례한다. 따라서 오차가 더큰 인풋값에 대해서는 더많이 업데이트하고, 오차가 더 작은 인풋값에 대해서는 더적게 업데이트하는 결과를 얻을 수 있다.**

- **또한, 미분값에 $\sigma'(z)$ 가 포함되어 있지 않기 때문에 MSE Function을 이용해서 Cost Function을 정의했을때 발생하는 sigmoid function의 특성때문에 발생하는 학습저하(slowdown) 문제도 발생하지 않는다.**

- 따라서 Cross-Entropy Cost Function이 MSE Cost Function보다 Neural Networks를 학습시키기는데 더 적합한 Cost Function임을 알 수 있다.

과제 - IRIS 데이터셋에 대한 ANN Classifier 구현

- TensorFlow를 이용해서 Iris Dataset 분류를 위한 ANN Classifier를 구현해봅니다.

IRIS dataset

Iris Versicolor

Iris Setosa

Iris Virginica

Questions & Answers

Thank You!