Artificial Intelligence (AI)

Lec07: Artificial Neural Network

충북대학교

문성태 (지능로봇공학과)

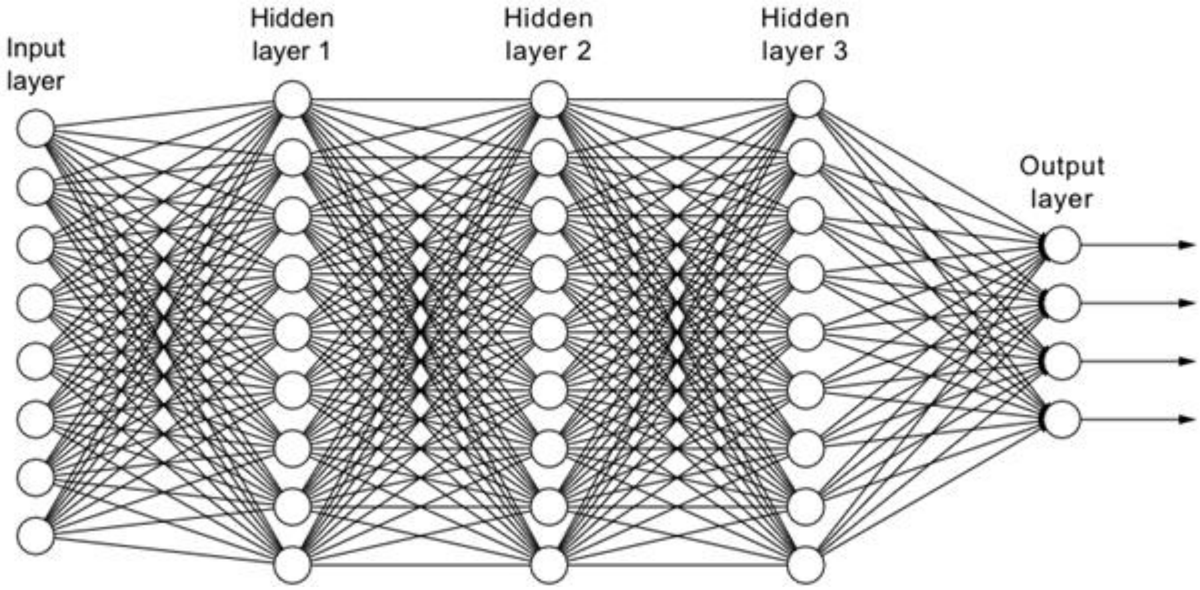
stmoon@cbnu.ac.kr

01

Backpropagation Analysis

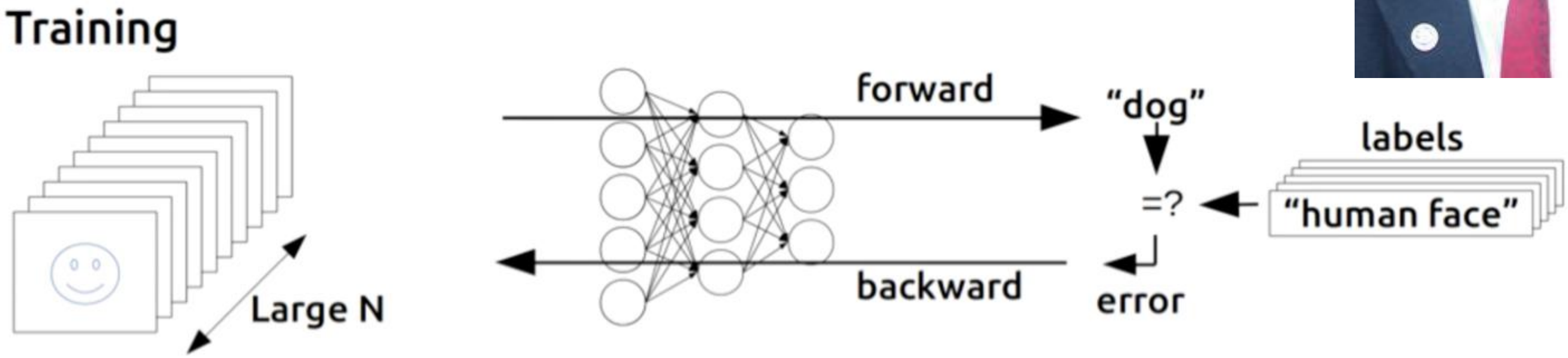
Recap: Backpropagation

모든 레이어의 파라미터를 구하기 위해 computational graph에 chain rule을 재귀적으로 적용하는 방법



Recap: Backpropagation

- 1974, 1982 by Paul Werbos, 1986 by Hinton
  - Paul Werbos, based on his 1974 Ph.D. thesis, publicly proposes the use of Backpropagation for propagating errors during the training of Neural Networks



Simple Example

$$f(x, y, z) = (x + y)z$$

$$q = x + y$$

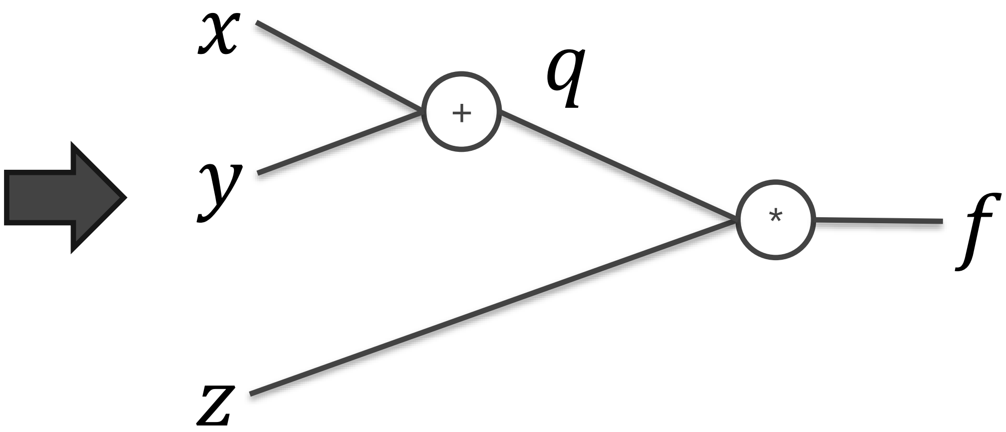
$$f = qz$$

Simple Example

$$f(x, y, z) = (x + y)z$$

$$q = x + y$$

$$f = qz$$



Target

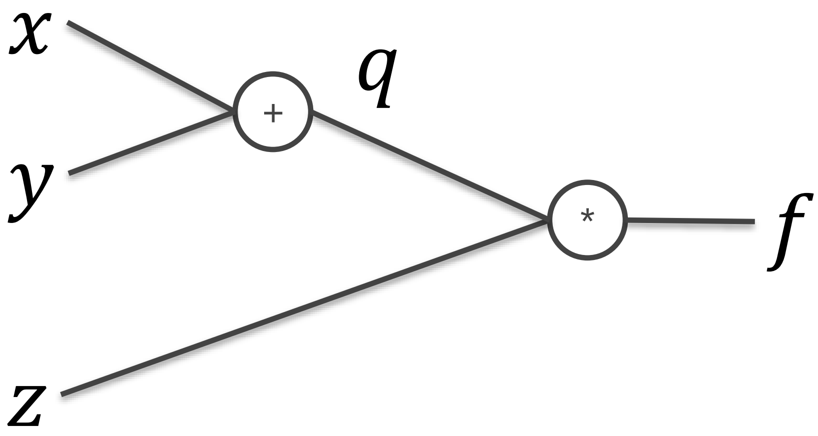
$$\frac{\partial q}{\partial x} = ? \quad \frac{\partial q}{\partial y} = ?$$

Simple Example

$$f(x, y, z) = (x + y)z$$

$$q = x + y$$

$$f = qz$$



Ex) x=−2, y=5, z=−4

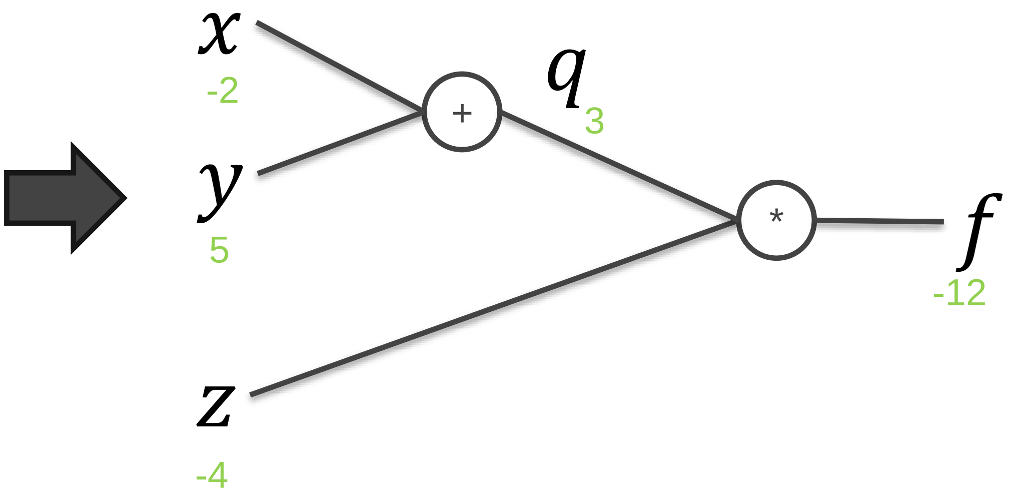
Simple Example

$$f(x, y, z) = (x + y)z$$

$$q = x + y$$

$$f = qz$$

Ex) x=−2, y=5, z=−4

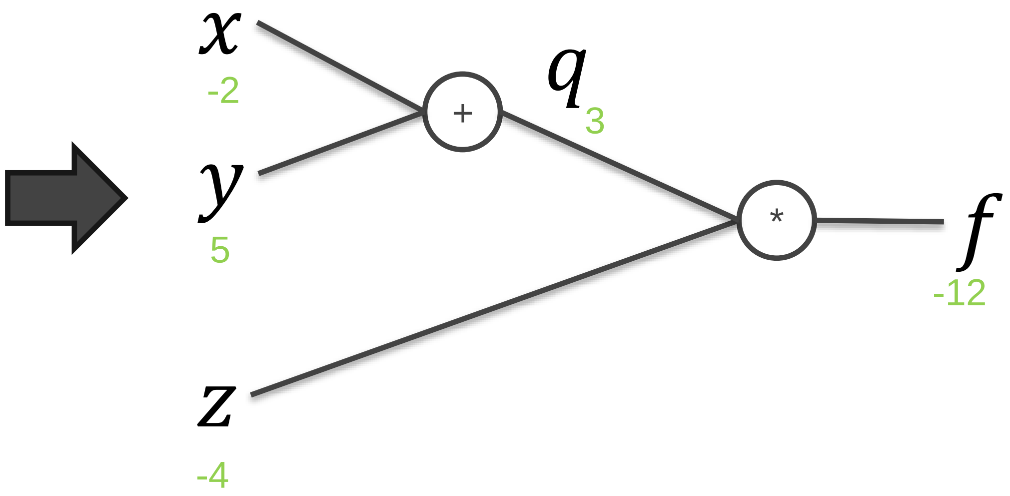


Simple Example

$$f(x, y, z) = (x + y)z$$

$$q = x + y$$

$$f = qz$$



Ex) x=−2, y=5, z=−4

Simple Example

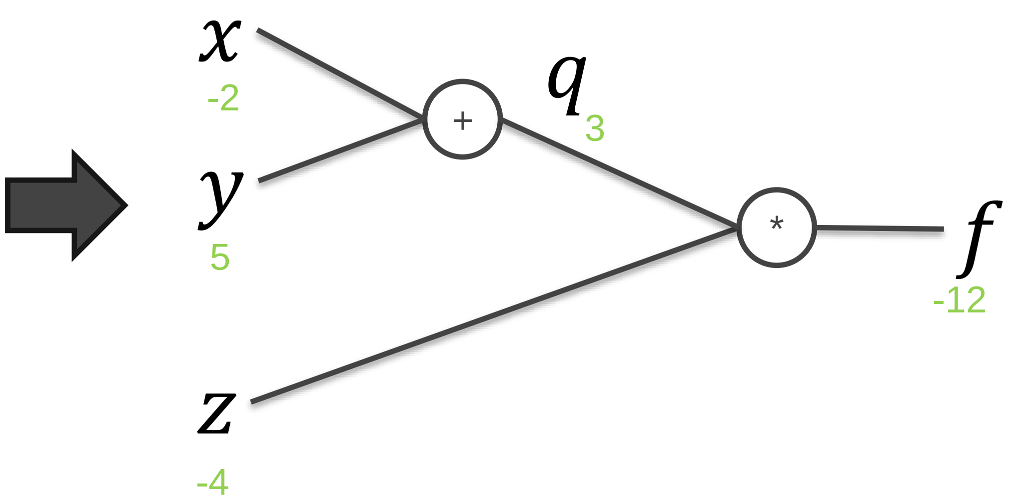
$$f(x, y, z) = (x + y)z$$

$$q = x + y$$

$$f = qz$$

Ex) x=−2, y=5, z=−4

$$\frac{\partial f}{\partial x} = ? \qquad \frac{\partial f}{\partial y} = ? \qquad \frac{\partial f}{\partial z} = ?$$



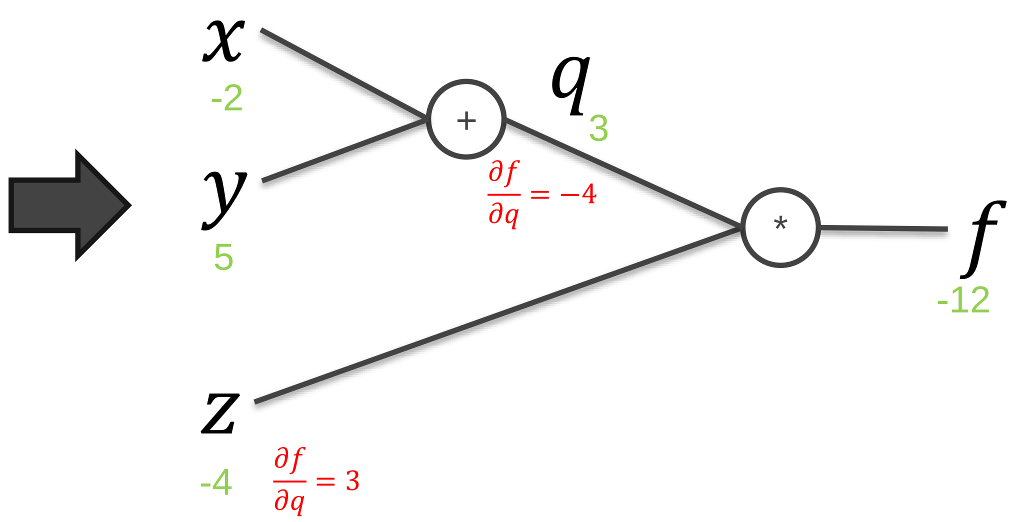
Simple Example

$$f(x, y, z) = (x + y)z$$

$$q = x + y$$

$$f = qz$$

Ex) x=−2, y=5, z=−4



$$\frac{\partial f}{\partial q} = z = -4 \qquad \frac{\partial f}{\partial z} = q = (x + y) = -2 + 5 = 3$$

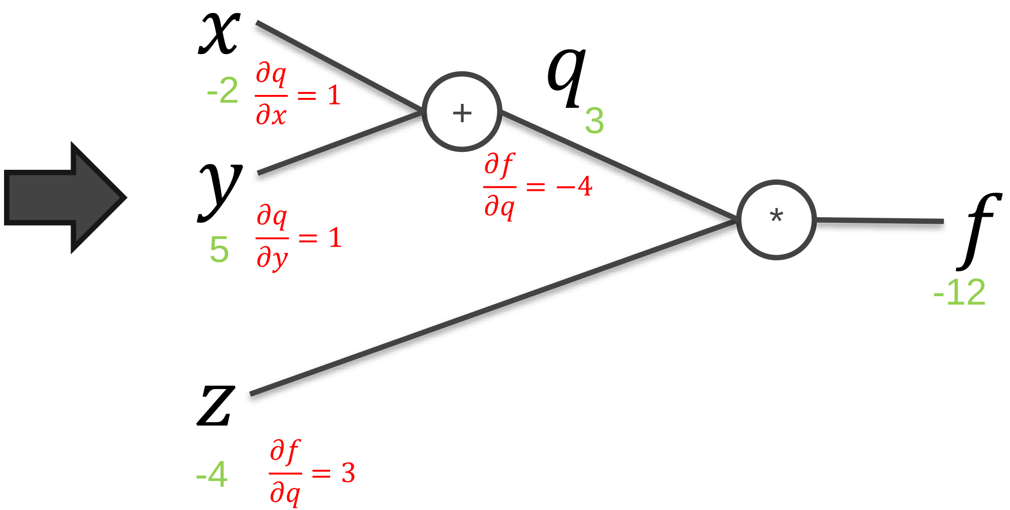
Simple Example

$$f(x, y, z) = (x + y)z$$

$$q = x + y$$

$$f = qz$$

Ex) x=−2, y=5, z=−4

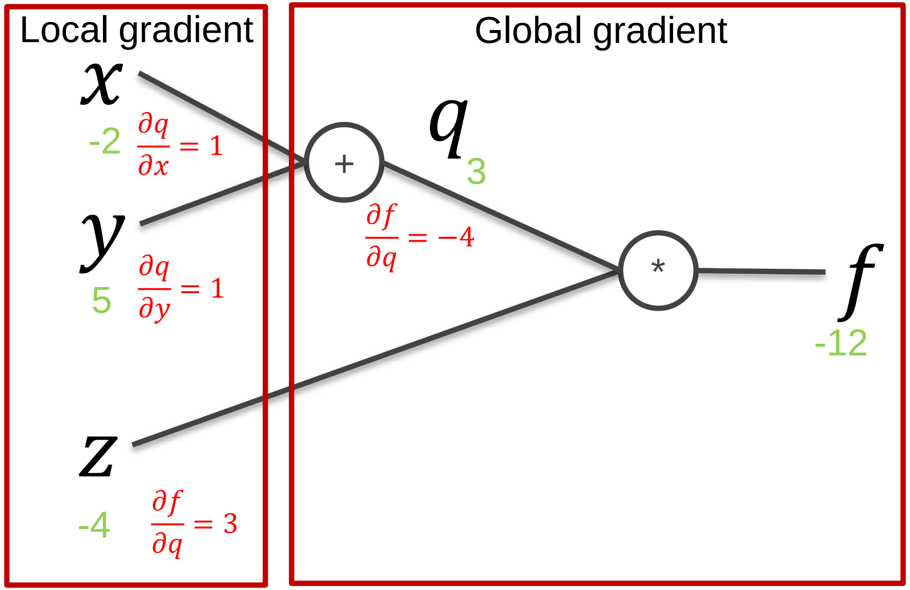


$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} = (-4) * 1 = -4$$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y} = (-4) * 1 = -4$$

수행 방식

- 많은 층으로 구성된 Neural Network에서 Chain Rule을 이용하면 미분값을 쉽게 구할 수 있다.
- Forward Propagation 에서는 Local gradient를 미리 구하고 저장한다.
- Backward Propagation 에서 Local gradient X Global gradient를 곱하여 계산한다.
    - 복잡한 연산을 단순하게 처리 가능하다



Recap: XOR Problem

Define and Initialize weights

```
## Initialize weights
n_x = 2 # Number of inputs
n_y = 1 # Number of neurons in output layer
n_h = 2 # Number of neurons in hidden layer

w1 = np.random.rand(n_h, n_x)   # Weight matrix for hidden layer
w2 = np.random.rand(n_y, n_h)   # Weight matrix for output layer
b1 = np.random.rand(2, 1)
b2 = np.random.rand(1, 1)
```

Learning

```
epoch = 10000
losses = []
m = y.shape[1]      # # of data set
lr = 0.1            # Learning rate

for i in range(epoch):
    z1, a1, z2, y_hat = forward_prop(w1, w2, b1, b2, X)
    loss = -(1/m)*np.sum(y*np.log(y_hat) + (1-y)*np.log(1-y_hat))
    losses.append(loss)

    dw1, db1, dw2, db2 = back_prop(m,w1,w2,z1,a1,z2,y_hat,X,y)
    w2 = w2 - lr*dw2
    w1 = w1 - lr*dw1
    b2 = b2 - lr*db2
    b1 = b1 - lr*db1
```

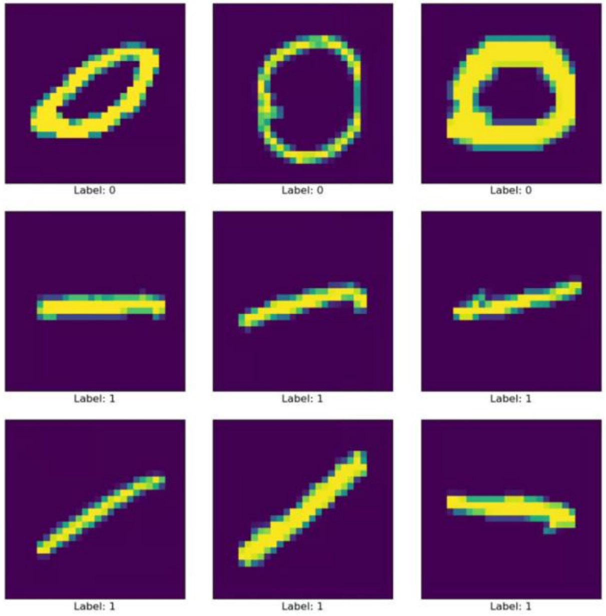
Recap: XOR Problem

Functions

```
[ ]
def sigmoid(z):
    z = 1 / ( 1 + np.exp(-z))
    return z

def forward_prop(w1, w2, b1, b2, x):
    z1 = w1 @ x + b1
    h1 = sigmoid(z1)

    z2 = w2 @ h1 + b2
    y_hat = sigmoid(z2)

    return z1, h1, z2, y_hat

def back_prop(m,w1,w2,z1,h1,z2,y_hat,x,y):
    dz2 = y_hat - y
    dw2 = dz2 @ h1.T
    db2 = dz2 @ np.ones((m, 1))

    dz1 = w2.T @ dz2 * h1 * (1 - h1)
    dw1 = dz1 @ x.T
    db1 = dz1 @ np.ones((m, 1))

    return dw1, db1, dw2, db2
```

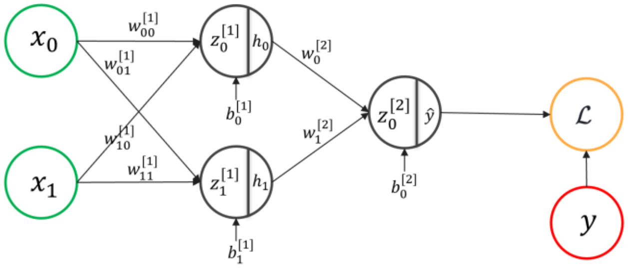
MNIST problem

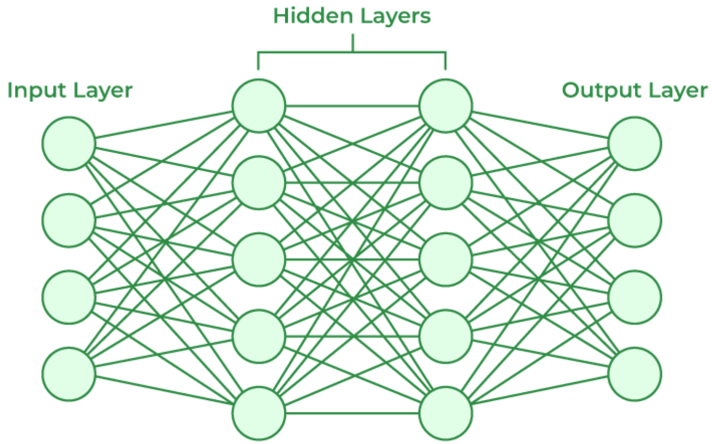
- A collection of handwritten digits from 0-9 (70,000 images)
- Grayscale image of size 28 X 28 pixel



MNIST problem

- Input layer
  - 28 x 28 = 784
- Hidden layer (1 ~)
  - 15 or 50 or 100
- Output layer
  - 0 ~ 9 digit = 10





Implementation (hidden layer 1)

Initialization

```python
## Initialize weights
n_x = X_train.shape[1]
n_y = y_train.shape[1]
n_h = 100

w1 = np.random.rand(n_x, n_h) - 0.5
w2 = np.random.rand(n_h, n_y) - 0.5
b1 = np.random.rand(1, n_h) - 0.5
b2 = np.random.rand(1, n_y) - 0.5
```

Implementation (hidden layer 1)

```
def forward_prop(w1, w2, b1, b2, x):
    z1 = x @ w1 + b1
    h1 = sigmoid(z1)

    z2 = h1 @ w2 + b2
    y_hat = sigmoid(z2)

    return z1,h1,z2,y_hat

def back_prop(m,w1,w2,z1,h1,z2,y_hat,x,y):
    dz2 = y_hat - y
    dw2 = h1.T @ dz2
    db2 = np.ones((1,m)) @ dz2  / m

    dz1 = (dz2 @ w2.T) * sigmoid_prime(h1)
    dw1 = x.T @ dz1

    db1 = np.ones((1,m)) @ dz1 / m

    return dw1, db1, dw2, db2
```

Implementation (hidden layer 1)

Main Loop

```
[ ]
epoch = 30
losses = []
m = y_train.shape[0]        # # of data set
lr = 0.01                   # Learning rate

y_train_true = np.argmax(y_train, axis=1)
y_test_true = np.argmax(y_test, axis=1)
for i in range(epoch):
    z1, a1, z2, y_hat = forward_prop(w1, w2, b1, b2, X_train)
    loss = -(1/m)*np.sum(y_train*np.log(y_hat + 1e-10) + (1-y_train)*np.log(1-y_hat + 1e-10))

    losses.append(loss)

    dw1, db1, dw2, db2 = back_prop(m, w1, w2, z1, a1, z2, y_hat, X_train, y_train)
    w2 = w2 - lr*dw2
    w1 = w1 - lr*dw1

    b2 = b2 - lr*db2
    b1 = b1 - lr*db1

    # print(w1[0, :5])
    # print(w2[0, :5])
    print(f'loss: {loss}')
```

02

Vanishing Gradient Problem

Vanishing Graident

Define and Initialize weights

[ ]

```
## Initialize weights
n_x = 2 # Number of inputs
n_y = 1 # Number of neurons in output layer
n_h = 2 # Number of neurons in hidden layer

w1 = np.random.rand(n_h, n_x)   # Weight matrix for hidden layer
w2 = np.random.rand(n_y, n_h)   # Weight matrix for output layer
b1 = np.random.rand(2, 1)
b2 = np.random.rand(1, 1)
```

Vanishing Gradient

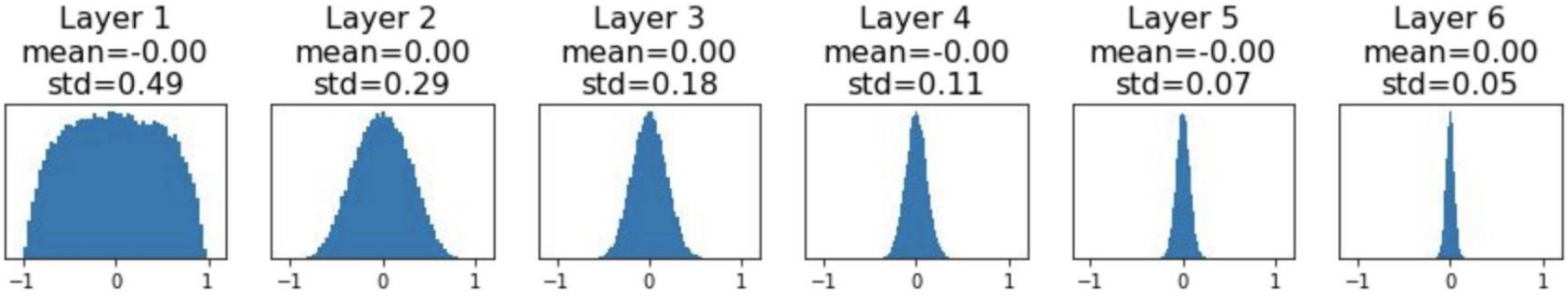
- Weight 분석
  - Weight가 상대적으로 작다면?

```
w1 = np.random.rand(n_h, n_x)
w2 = np.random.rand(n_y, n_h)
b1 = np.random.rand(2, 1)
b2 = np.random.rand(1, 1)
```

Almost zero activations at top layers!

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}} \cdot \frac{\partial \mathbf{h}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}} \cdot (\mathbf{z}^{\mathrm{T}}) = 0$$

→ No learning!



Vanishing Gradient

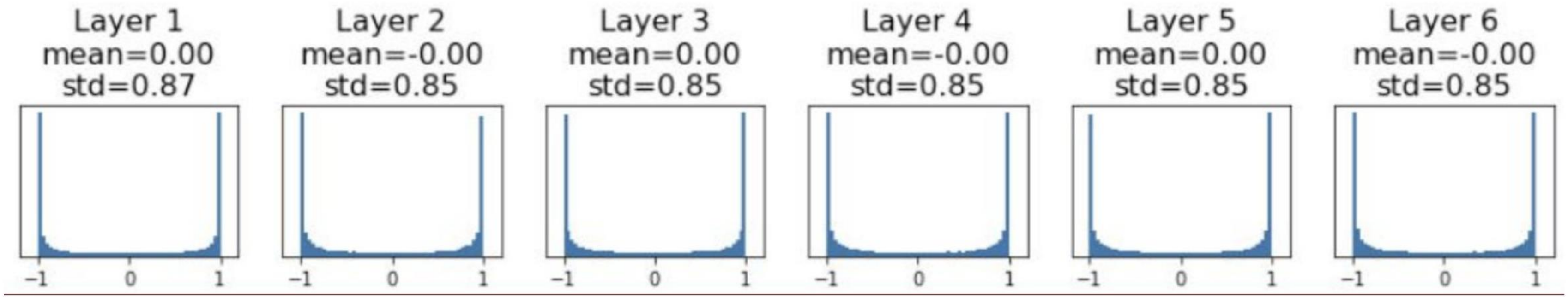
- Weight 분석
  - Weight가 상대적으로 크다면?

```
w1 = np.random.rand(n_h, n_x)
w2 = np.random.rand(n_y, n_h)
b1 = np.random.rand(2, 1)
b2 = np.random.rand(1, 1)
```

Almost zero gradient due to saturation in nonlinear function!

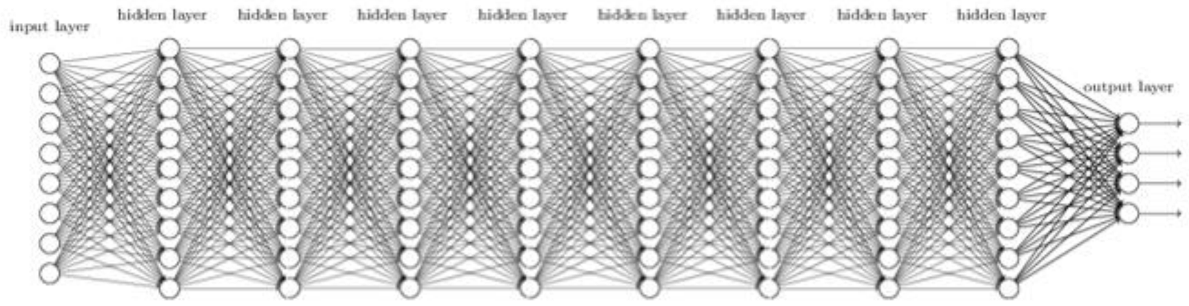
$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}} \cdot \frac{\partial \mathbf{h}}{\partial \mathbf{z}} = (\overbrace{\sigma'(\cdot)}^{=0}\mathbf{W}^{\mathrm{T}}) \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{h}} \quad =0$$

→ No learning!

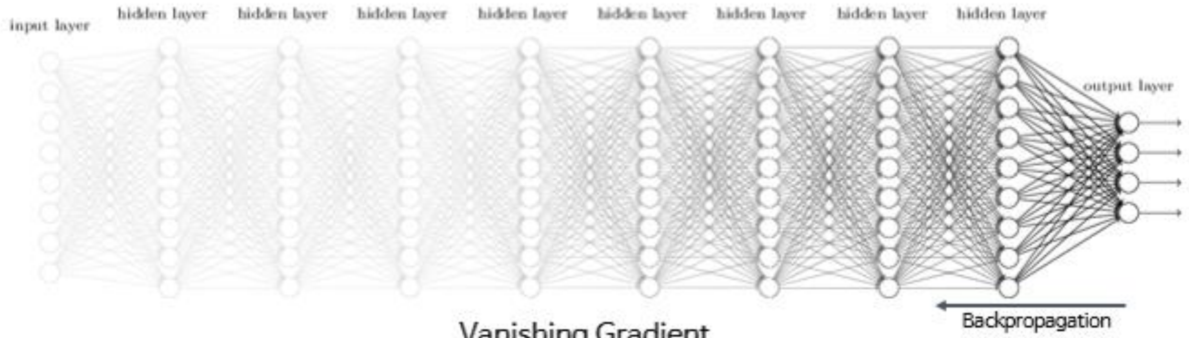


Vanishing Gradient

- Backpropagation 과정에서 출력층에서 멀어질수록 Gradient 값이 매우 작아지는 현상



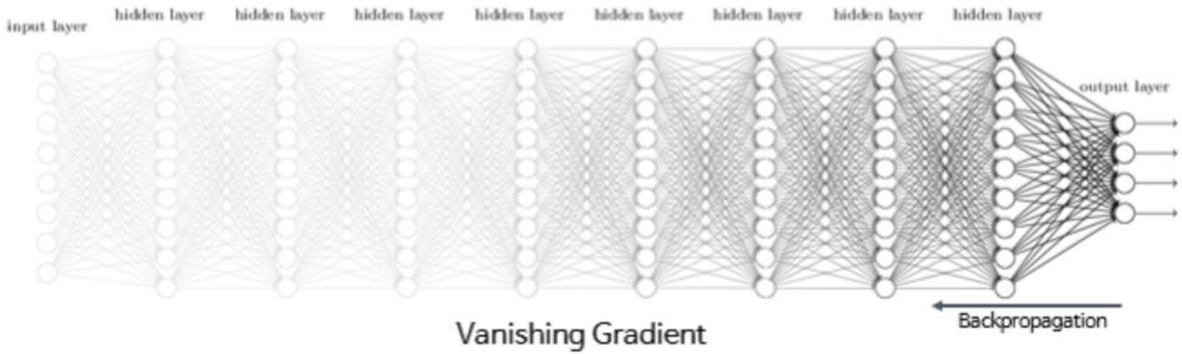
Deep Neural Network



Vanishing Gradient

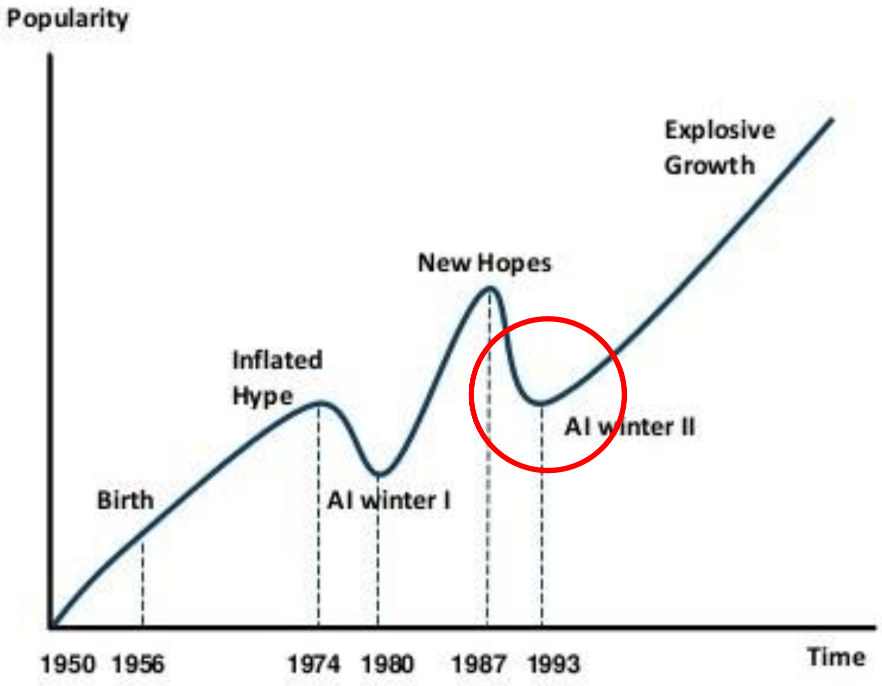
AI Winter II

- Backpropagation just did not work well for normal neural nets with many layers
- Other rising machine learning algorithms: SVM, RandomForest, etc.
- 1995 "Comparison of Learning Algorithms For Handwritten Digit Recognition" by LeCun et al. found that this new approach (SVM, RandomForest) worked better



AI Winter II

AI HAS A LONG HISTORY OF BEING "THE NEXT BIG THING"...



Timeline of AI Development

- **1950s-1960s**: First AI boom - the age of reasoning, prototype AI developed
- **1970s**: AI winter I
- **1980s-1990s**: Second AI boom: the age of Knowledge representation (appearance of expert systems capable of reproducing human decision-making)
- **1990s**: AI winter II
- **1997**: Deep Blue beats Gary Kasparov
- **2006**: University of Toronto develops Deep Learning
- **2011**: IBM's Watson won Jeopardy
- **2016**: Go software based on Deep Learning beats world's champions

CIFAR

- Canadian Institute for Advanced Research (CIFAR)
- CIFAR encourages basic research without direct application, was what motivated Hinton to move to Canada in 1987, and funded his work afterward.

"CIFAR had a huge impact in forming a community around deep learning," by LeCun

CIFAR

CANADIAN INSTITUTE for ADVANCED RESEARCH

- In 2006, Hinton, Simon Osindero, and Yee-Whye Teh published, "A fast learning algorithm for deep belief nets"
- Yoshua Bengio et al. in 2007 with "Greedy Layer-Wise Training of Deep Networks"

Breakthrough in 2006 and 2007 by Hinton and Bengio

- Neural networks with many layers really could be trained well, if the weights are initialized in a clever way rather than randomly.
- Deep machine learning methods are more efficient for difficult problems than shallow methods.
- Rebranding to Deep Nets, Deep Learning

How to solve vanishing gradient

Initialization + Activation Function

Xavier (or Glorot) Initialization

- Xavier (or Glorot) initialization (Xavier Glorot, Yoshua Bengio, 2010)
  - Xavier Normal initialization

$$W \sim N(0, Var(W))$$

$$Var(W) = \sqrt{\frac{2}{n_{in}+n_{out}}}$$

(n_{in} : 이전 layer(input)의 노드 수, n_{out} : 다음 layer의 노드 수)

  - Xavier Uniform initialization

$$W \sim U(-\sqrt{\frac{6}{n_{in}+n_{out}}}, +\sqrt{\frac{6}{n_{in}+n_{out}}})$$

(n_{in} : 이전 layer(input)의 노드 수, n_{out} : 다음 layer의 노드 수)

➔ 비선형함수(ex. sigmoid, tanh)에서 효과적인 결과

➔ ReLU함수에서 사용 시 출력 값이 0으로 수렴하게 되는 현상

He Initialization

- He initialization (Kaiming He et al, 2015)
  - Xavier Normal initialization

$$W \sim N(0, Var(W))$$

$$Var(W) = \sqrt{\frac{2}{n_{in}}}$$

(n_{in} : 이전 layer(input)의 노드 수, n_{out} : 다음 layer의 노드 수)

  - Xavier Uniform initialization

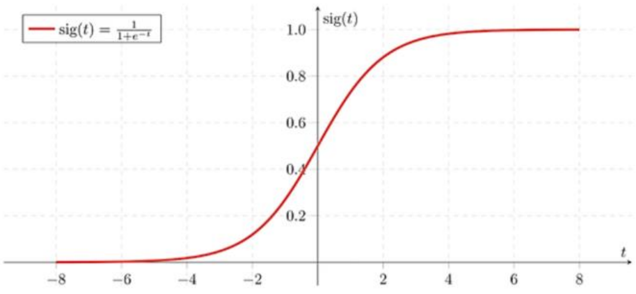
$$W \sim U(-\sqrt{\frac{6}{n_{in}}}, +\sqrt{\frac{6}{n_{in}}})$$

(n_{in} : 이전 layer(input)의 노드 수, n_{out} : 다음 layer의 노드 수)

Analysis of Activation Function

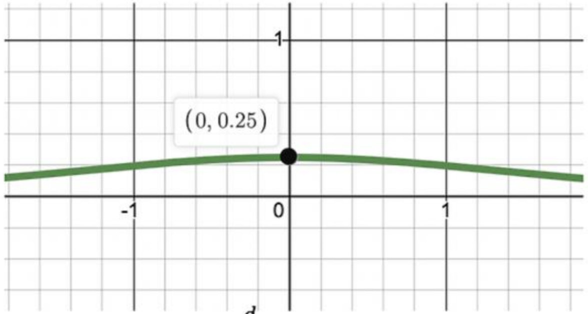
- Sigmoid
  - 미분 값은 입력값이 0일 때 가장 크지만 0.25에 불과
  - X 값이 크거나 작아짐에 따라 기울기는 0에 수렴

➔ 역전파 과정에서 Sigmoid 함수의 미분값이 거듭 곱해지면
출력층과 멀어질수록 Gradient 값이 매우 작아질 수밖에 없다!



sigmoid

https://heytech.tistory.com/

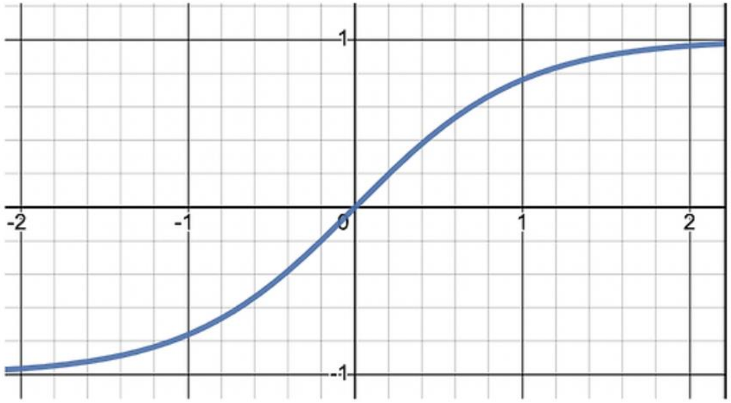


$\frac{d}{dx}sigmoid$

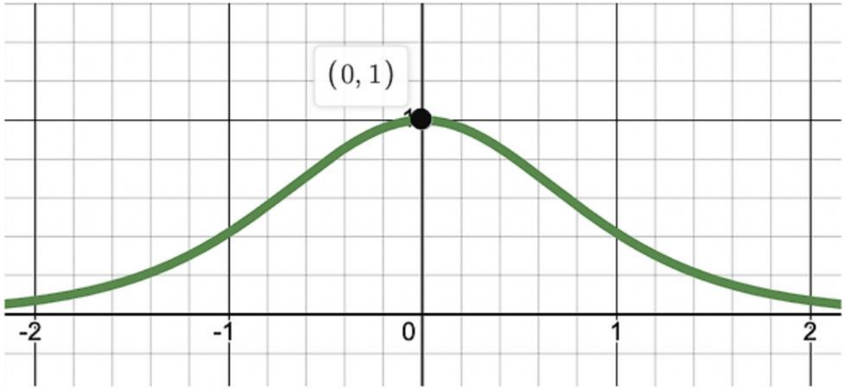
tanh Activation Function

- Tahh
  - Sigmoid 함수의 대안
  - 출력값: -1 ~ 1 까지 2배 증가 (sigmoid : 0 ~ 1)
  - 기울기: 0 ~ 1까지 4배 증가 (sigmoid: 0 ~ 0.25)

➔ 값이 크거나 작아짐에 따라 기울기 크기가 크게 작아지게 때문에 여전히 기울기 소실 문제 방지할 수 없음



$tanh$



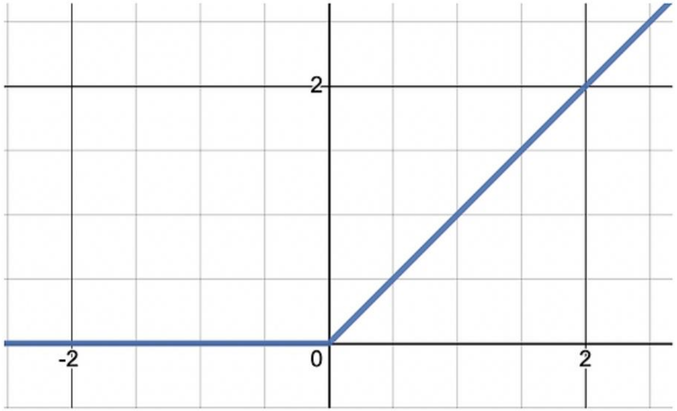
$\frac{d}{dx}tanh$

https://heytech.tistory.com/

ReLU Activation Function

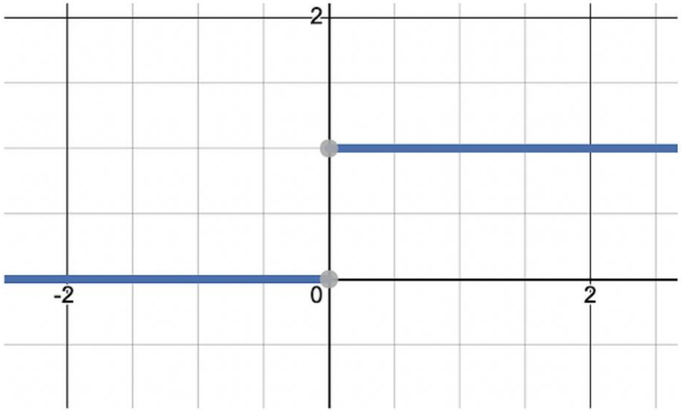
- ReLU (Rectified Linear Unit)
  - 입력값이 양수일 경우, 입력값에 상관 없이 항상 동일한 미분 값: 0 ➔ 기울기 손실 발생 X
  - 특별한 연산이 없어서 연산 속도가 빠름

➔ 하지만, 입력값이 음수인 경우 항상 0 ➔ 입력값이 음수이면 다시는 회생 불가능!



$ReLU$

https://heytech.tistory.com/

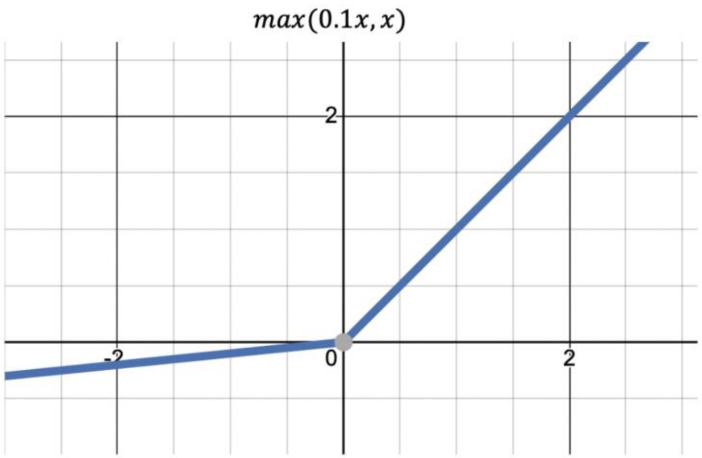


$\frac{d}{dx}ReLU$

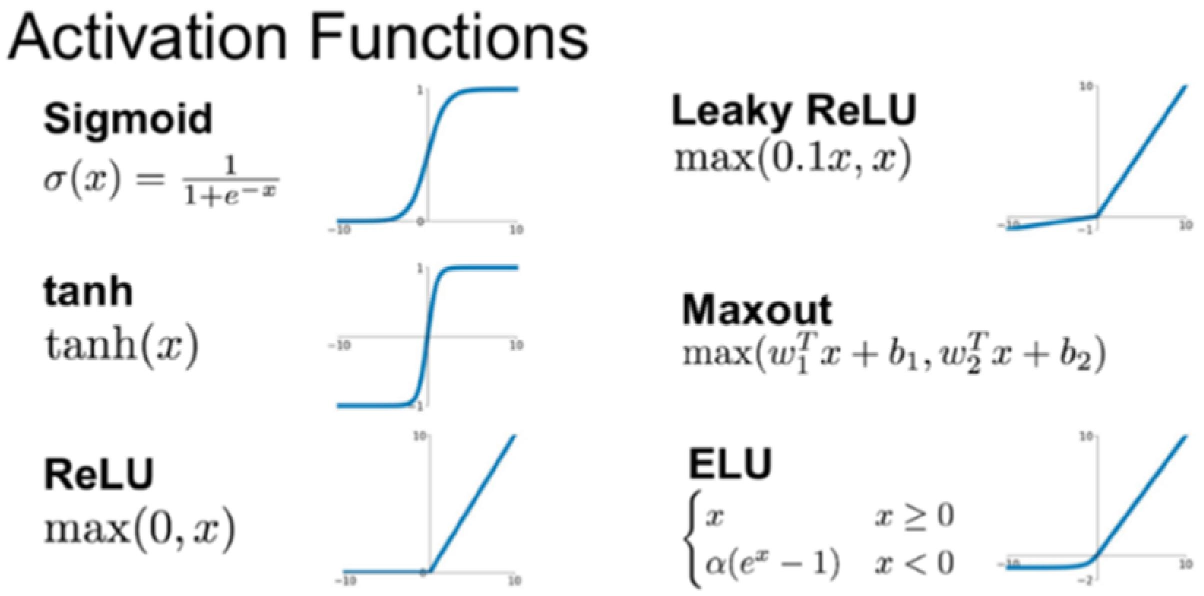
Leaky ReLU Activation Function

- Leaky ReLU
  - 입력값이 음수일 때 출력값을 0 이 아닌 매우 작은 값을 출력하도록 설정

➔ 입력값이 음수라도 기울기가 0 이 되지 않아 뉴런이 죽는 현상을 방지



Leaky ReLU https://heytech.tistory.com/

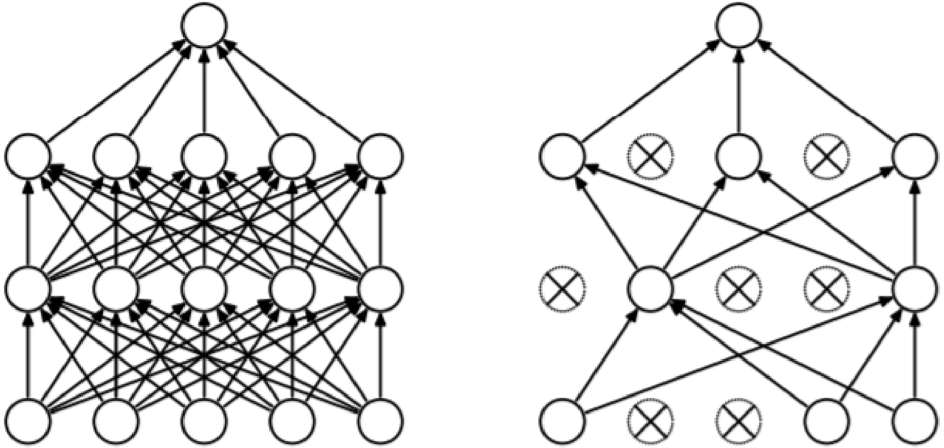


Different Activation Functions and their Graphs

Dropout

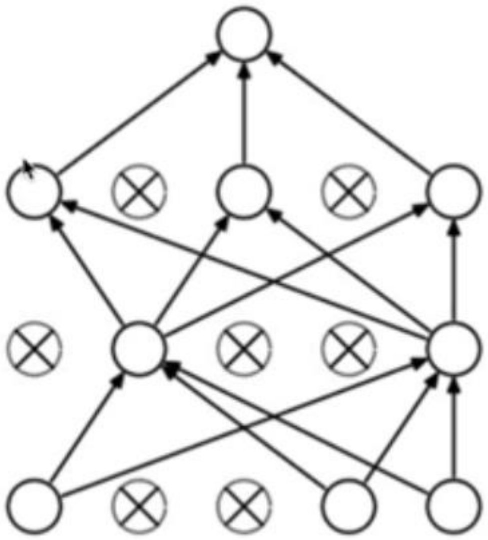
Dropout

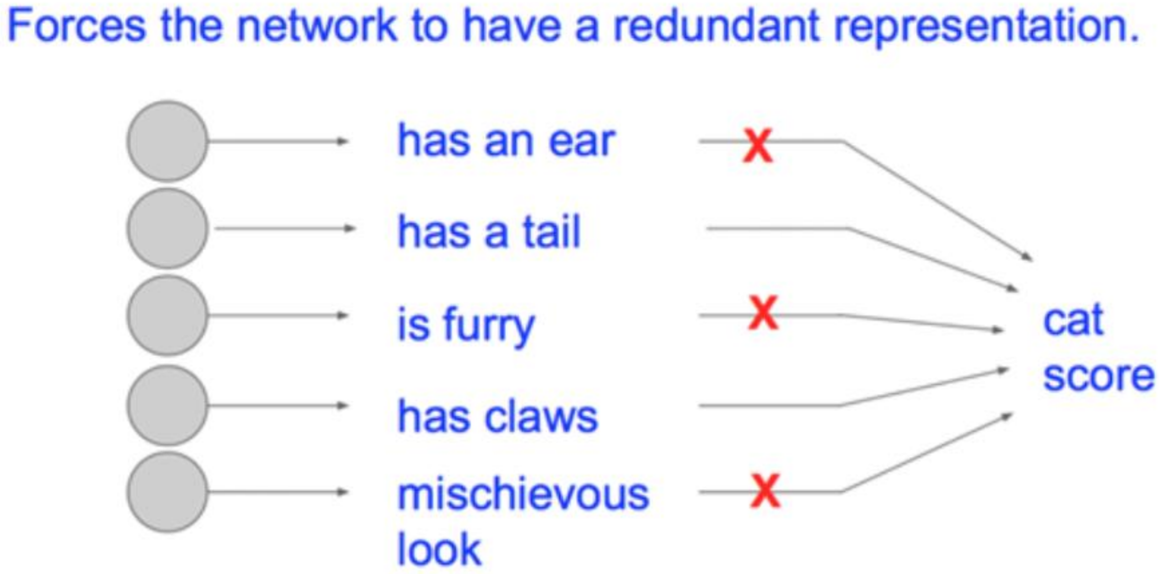
- Dropout: A simple way to prevent neural networks from overfitting [Srivastava et al. 2014]
- *"Randomly set some neurons to zero in the forward pass"*



(a) Standard Neural Net (b) After applying dropout.

How could this possibly be a good idea?





- Only use dropout during training

Gradient Descent

Gradient Descent

Learning

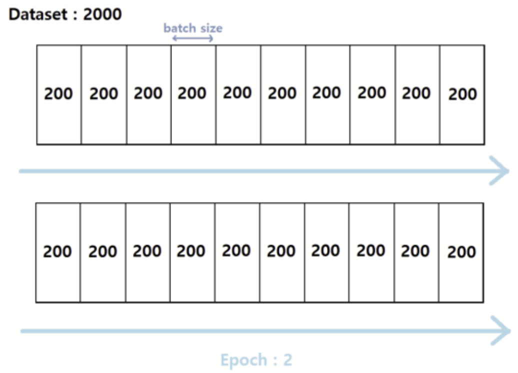
```
epoch = 10000
losses = []
m = y.shape[1]          # # of data set
lr = 0.1                # Learning rate

for i in range(epoch):
    z1, a1, z2, y_hat = forward_prop(w1, w2, b1, b2, X)
    loss = -(1/m)*np.sum(y*np.log(y_hat) + (1-y)*np.log(1-y_hat))
    losses.append(loss)

    dw1, db1, dw2, db2 = back_prop(m,w1,w2,z1,a1,z2,y_hat,X,y)
    w2 = w2 - lr*dw2
    w1 = w1 - lr*dw1
    b2 = b2 - lr*db2
    b1 = b1 - lr*db1
```

Epoch/Batch size/Iteration

In the neural network terminology:

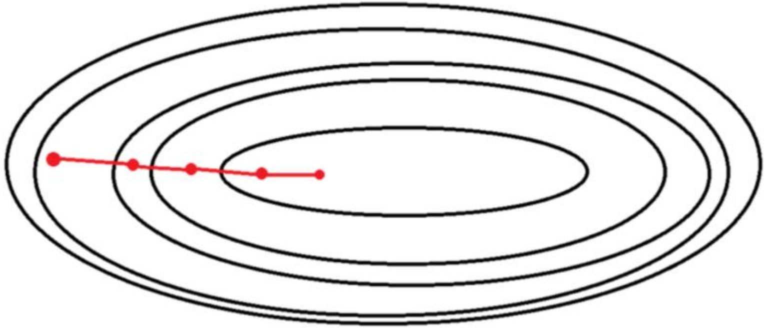
- one **epoch** = one forward pass and one backward pass of *all* the training examples
- **batch size** = the number of training examples in one forward/backward pass. The higher the batch size, the more memory space you'll need.
- number of **iterations** = number of passes, each pass using [batch size] number of examples. To be clear, one pass = one forward pass + one backward pass (we do not count the forward pass and backward pass as two different passes).

Example: if you have 1000 training examples, and your batch size is 500, then it will take 2 iterations to complete 1 epoch.



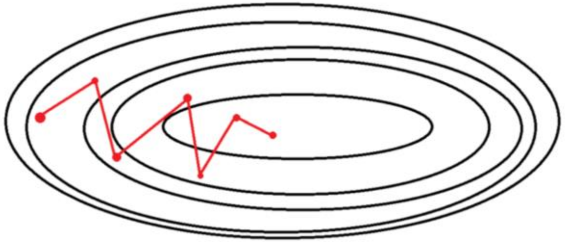
Gradient Descent

- 배치 경사하강법 (Batch Gradient Descent, BGD)
  - 배치 사이즈가 훈련세트 사이즈와 동일한 경사하강법
  - 전체 훈련세트를 한 번에 처리해 기울기를 업데이트
  - 특징
    - 항상 같은 데이터에 대해 경사를 구하기 때문에, 수렴이 안정적
    - 전체 훈련세트를 한 번에 처리하기 때문에, 메모리가 가장 많이 필요
    - 긴 시간이 소요



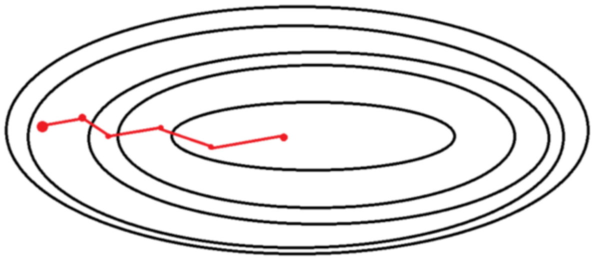
Gradient Descent

- 확률적 경사하강법 (Stochastic Gradient Descent, SGD)
  - 배치 사이즈가 1개인 경사하강법
  - 전체 훈련세트 중, 랜덤하게 하나의 데이터를 선택해 기울기를 업데이트 하기 때문에 확률적 경사하강법이라고 부름
  - 1개의 훈련데이터만 처리해 기울기를 업데이트
  - 특징
    - 수렴에 Shooting이 발생
    - 전역 최저점(Global Minimum)에 수렴하기는 어렵지만, 지역 최저점(Local Minimum)에 빠질 확률 감소
    - 훈련데이터를 1개씩 처리하기 때문에, 벡터화 과정에서 대부분의 속도를 잃으며 GPU의 병렬 처리 활용 어려움



Gradient Descent

- 미니 배치 확률적 경사하강법 (Mini-Batch Stochastic Gradient Descent, MSGD)
    - 배치 경사하강법과 확률적 경사하강법의 절충안으로, 전체 훈련세트를 1~M 사이의 적절한 batch size로 나누어 학습
    - 특징
        - 훈련세트의 사이즈가 클 경우, 배치 경사하강법보다 속도가 빠릅니다
        - Shooting이 적당히 발생 (Local Minimum 회피 가능)



Learning Rate

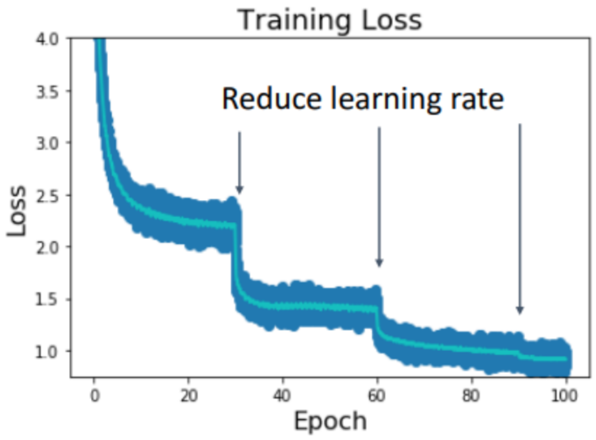
Learning Rate

Learning

```
epoch = 10000
losses = []
m = y.shape[1]        # # of data set
lr = 0.1              # Learning rate

for i in range(epoch):
    z1, a1, z2, y_hat = forward_prop(w1, w2, b1, b2, X)
    loss = -(1/m)*np.sum(y*np.log(y_hat) + (1-y)*np.log(1-y_hat))
    losses.append(loss)

    dw1, db1, dw2, db2 = back_prop(m,w1,w2,z1,a1,z2,y_hat,X,y)
    w2 = w2 - lr*dw2
    w1 = w1 - lr*dw1
    b2 = b2 - lr*db2
    b1 = b1 - lr*db1
```

Learning Rate Decay

- 딥 러닝 신경망이 확률적 경사 하강법(SGD : Stochastic Gradient Descent) 최적화 알고리즘을 사용하여 훈련하는데서 나온 파라미터
- 모델의 weight이 업데이트 될 때마다 예상 오류에 대한 응답으로 모델을 조정하고 제어하면서 모델 학습에 영향을 주는 하이퍼 파라미터



Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.