

Neural Network

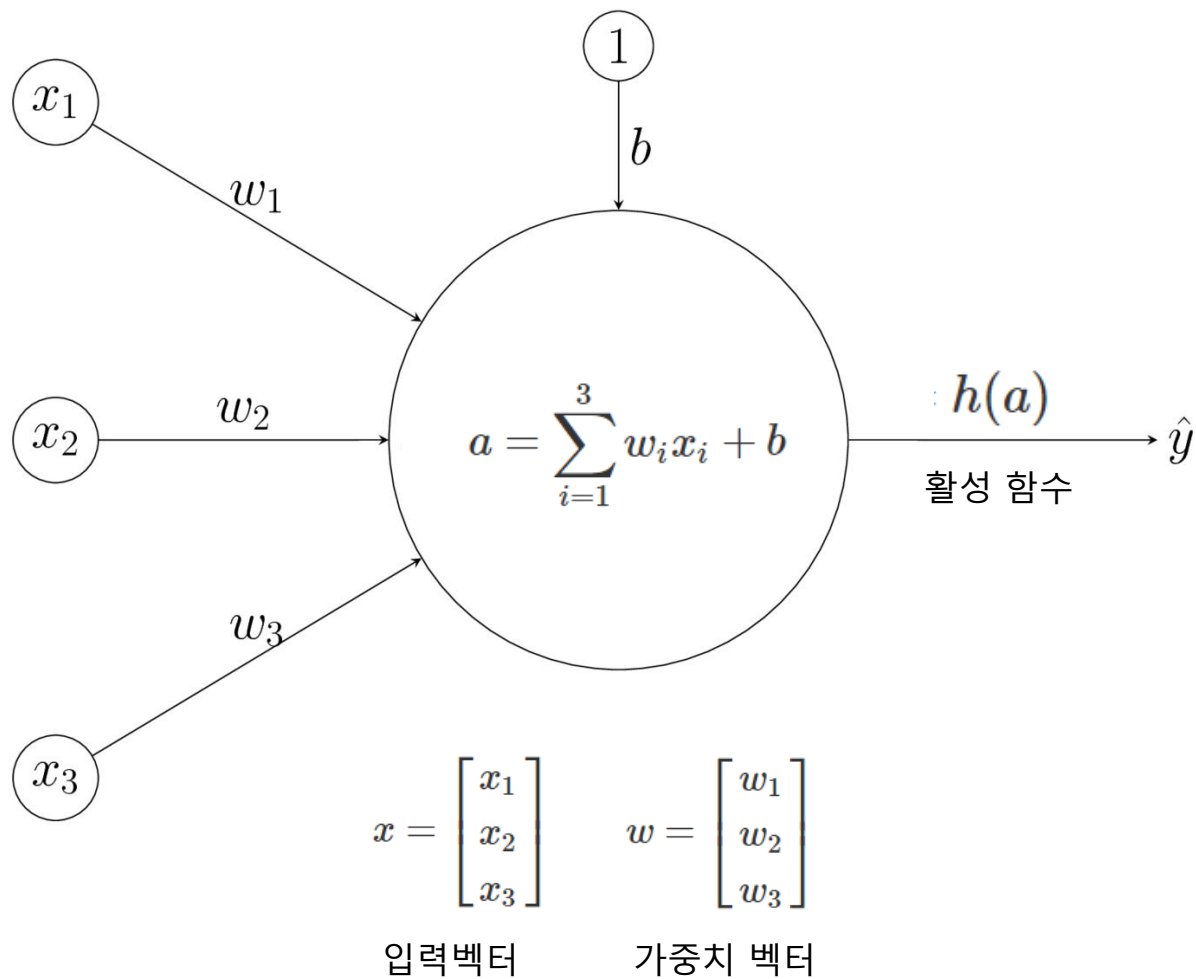
구름

도시공학과 일반대학원

한양대학교

Neural Network 기초

뉴런의 구조

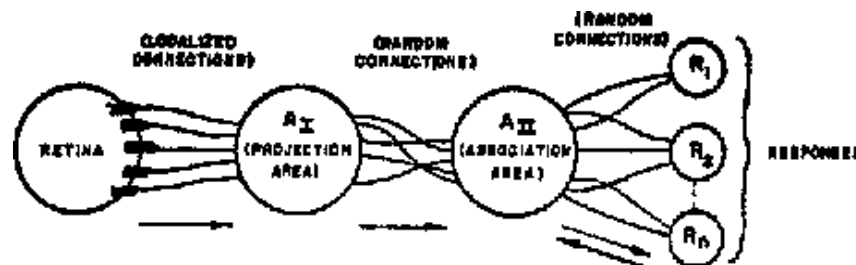


Neural Network 기초

단층퍼셉트론

퍼셉트론은 1957년 코넬 항공 연구소(Cornell Aeronautical Lab)의 프랑크 로젠블라트(Frank Rosenblatt)에 의해 고안된 인공신경망이다. 로젠블라트에 의해 제안된 것은 가장 간단한 형태의 단층 퍼셉트론(single-layer perceptron)으로 입력 벡터를 두 부류로 구분하는 선형분류기이다.

Rosenblatt, Frank (1958), The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain, Cornell Aeronautical Laboratory, Psychological Review, v65, No. 6,



Neural Network 기초

단층퍼셉트론

임계치(threshold): 어떠한 값이 활성화되기 위한 최소값을 임계치라고 한다.

가중치(weight): 퍼셉트론의 학습 목표는 학습 벡터를 두 부류로 선형 분류하기 위한 선형 경계를 찾는 것이다. 가중치는 이러한 선형 경계의 방향성 또는 형태를 나타내는 값이다.

바이어스(bias): 선형 경계의 절편을 나타내는 값으로써, 직선의 경우는 y절편을 나타낸다.

net값: 입력값과 가중치의 곱을 모두 합한 값으로써, 기하학적으로 해석하면 선형 경계의 방정식과 같다.

$$net = \sum_i^N w_i x_i + w_0 x_0$$

활성함수(activation function): 뉴런에서 계산된 net값이 임계치보다 크면 1을 출력하고, 임계치보다 작은 경우에는 0을 출력하는 함수이다. 이 정의는 단층 퍼셉트론에서만 유효하며, 다층 퍼셉트론에서는 다른 형태의 활성화함수를 이용한다.

$$f(net) = \begin{cases} 1, & net \geq threshold \\ 0, & net < threshold \end{cases} \tag{1}$$

뉴런(neuron): 인공신경망을 구성하는 가장 작은 요소로써, net값이 임계치보다 크면 활성화되면서 1을 출력하고, 반대의 경우에는 비활성화되면서 0을 출력한다.

학습 연산

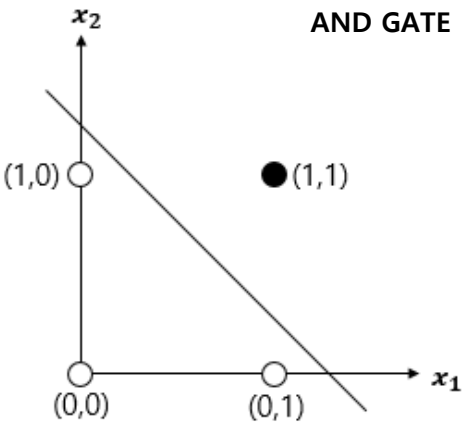
$$w_i = w_i + \eta x_i (t - f(net))$$

(2)

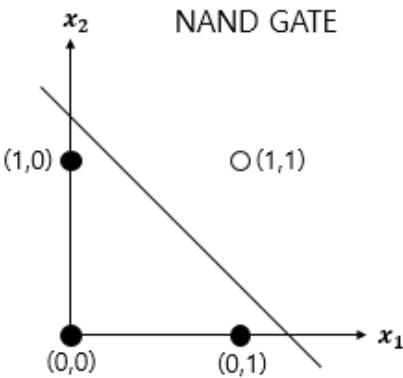
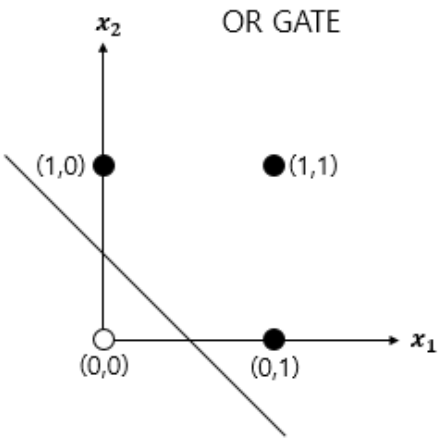
$\eta = learning\ rate$

$t = target\ value$

Neural Network 기초



x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1



01 단층퍼셉트론/SLP Sources.py

```
ds=[[0,0,0],[0,1,0],[1,0,0],[1,1,1]] #AND
#ds=[[0,0,0],[0,1,1],[1,0,1],[1,1,1]] #OR
#ds=[[0,0,0],[0,1,1],[1,0,1],[1,1,0]] #XOR

w0, w1, w2 = 0.3, 0.4, 0.1
x0 = -1
threshold = 0
learning_rate = 0.05

t = 0

while t < 10 :
    print(str(t) + ' 번째 루프')
    bLearn = False

    for x1,x2,y in ds:
        if x0*w0 + x1*w1 + x2*w2 >= threshold:
            y_hat = 1
        else:
            y_hat = 0

        print("x1:{0:0f} x2:{1:0f} y:{2:0f} y^{3:0f}".format(x1,x2,y, y_hat))

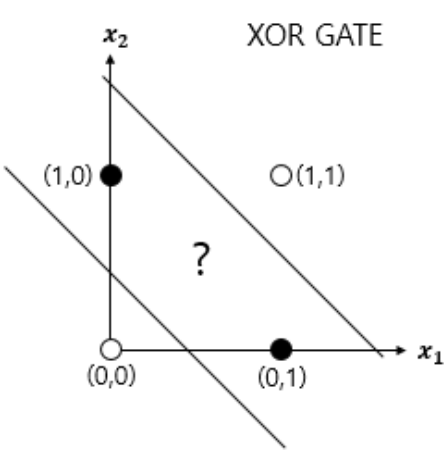
    if y != y_hat:
        print('학습 필요 w0:{0:2f} w1:{1:2f} w2:{2:2f}'.format(w0,w1,w2))
        print('학습량 w0:{0:2f} w1:{1:2f} w2:{2:2f}'.format(
            x0*(y - y_hat),x1*(y - y_hat), x2*(y - y_hat)))

        w0 = w0 + learning_rate*x0*(y - y_hat)
        w1 = w1 + learning_rate*x1*(y - y_hat)
        w2 = w2 + learning_rate*x2*(y - y_hat)

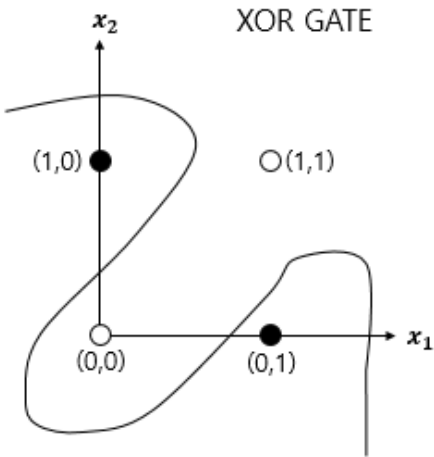
    print('학습 결과 w0:{0:2f} w1:{1:2f} w2:{2:2f}'.format(w0,w1,w2))
    bLearn = True
    t+=1
    if not bLearn : break
```

Neural Network 기초

단층퍼셉트론으로는 XOR 문제 해결 어려움



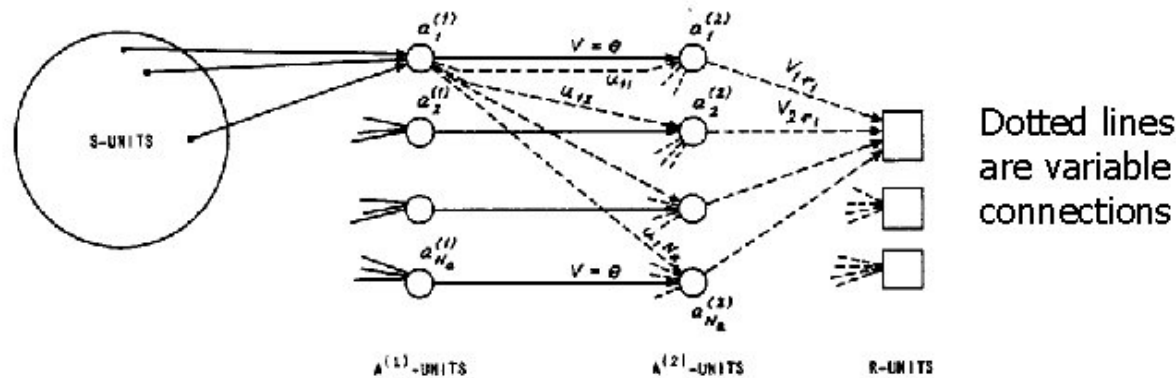
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



Neural Network 기초

다층퍼셉트론

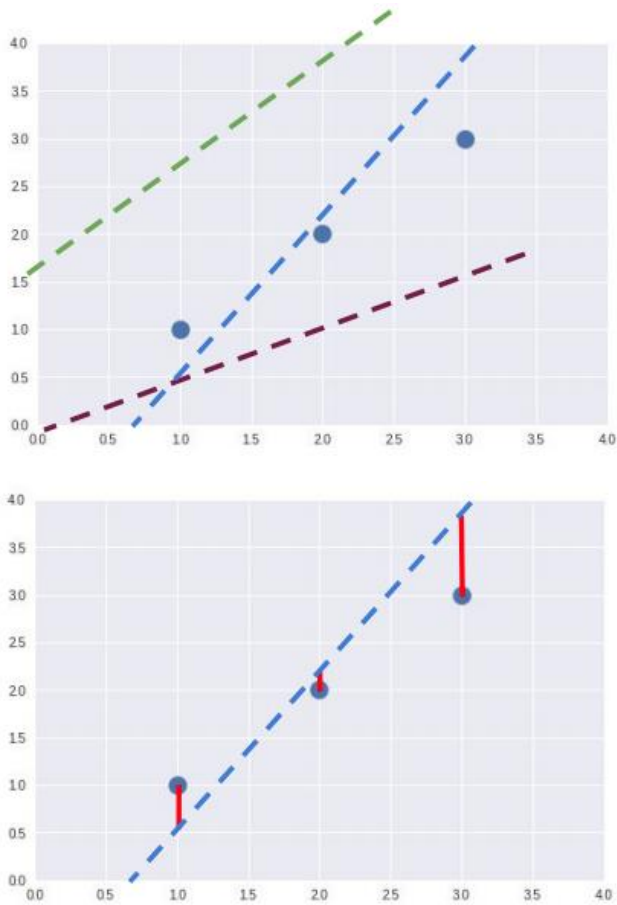
1962년 로센블레트는 자신의 퍼셉트론 이론을 보완하고 정리해서 "신경 역학의 법칙(Principles of Neurodynamics)"이라는 책을 출간하여 다중 퍼셉트론 개념을 제시
그러나 마땅한 수학적인 학습방법을 제시하지 못한 채 아이디어 수준에서 머무름



Neural Network 기초

오류 함수 (Loss Function)

모델의 예측과 실제 관측 값의 차이를 수치화 하기 위해 도입된 함수



$$MAE = \frac{1}{N} \sum_{i=1}^n |\hat{y}_i - y_i|$$

$$MSE = \frac{1}{N} \sum_{i=1}^n (\hat{y}_i - y_i)^2$$

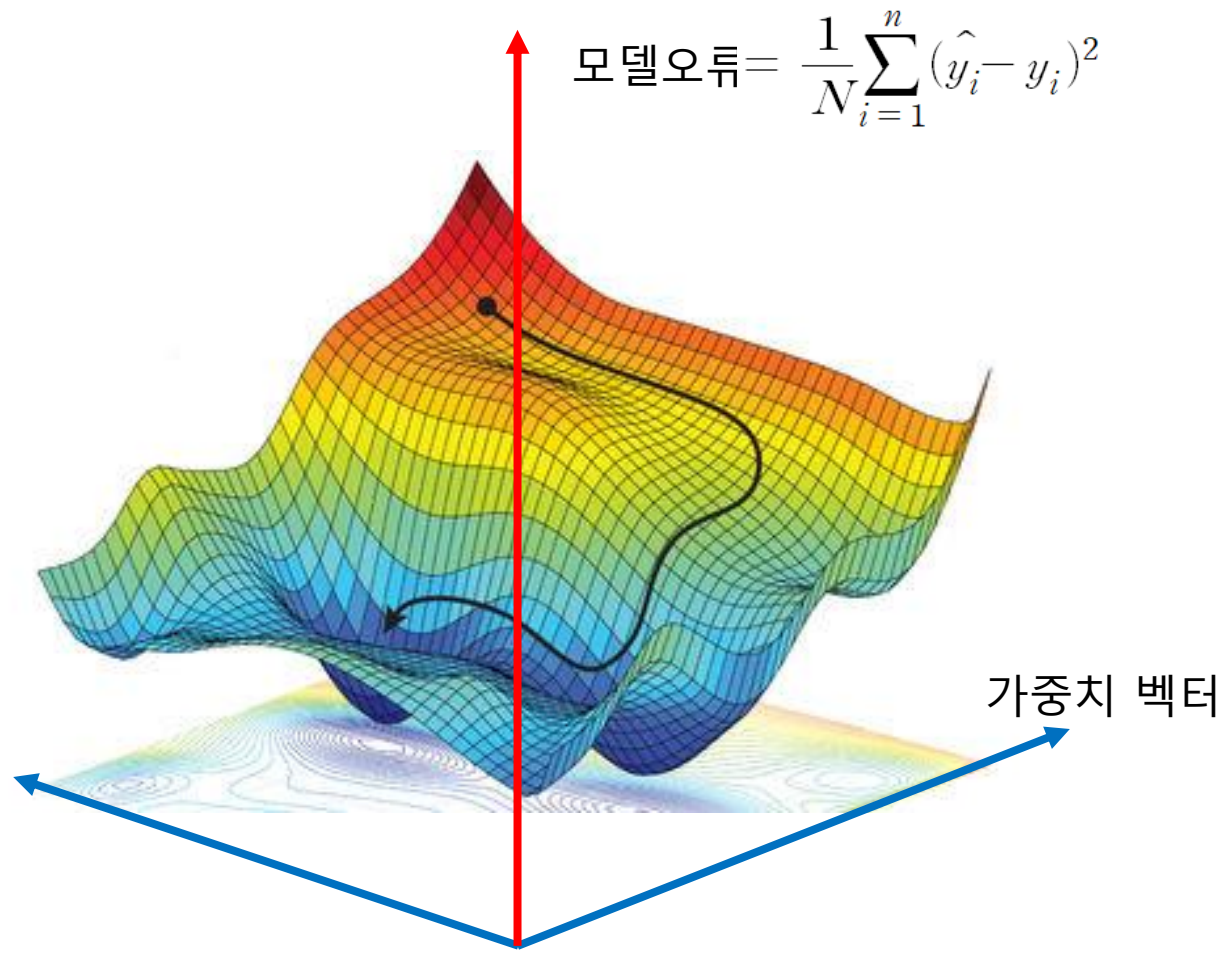
$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^n (\hat{y}_i - y_i)^2}$$

$$BCE = -\frac{1}{N} \sum_{i=0}^N y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)$$

$$CCE = -\frac{1}{N} \sum_{i=0}^N \sum_{j=0}^J y_j \cdot \log(\hat{y}_j) + (1 - y_j) \cdot \log(1 - \hat{y}_j)$$

Neural Network 기초

경사하강법

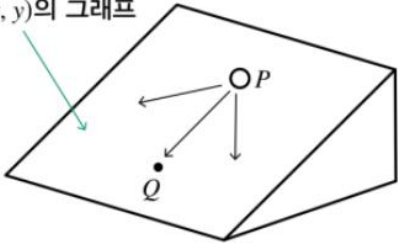


Neural Network 기초

경사하강법을 위해 각 가중치별 편미분

그림 2-47

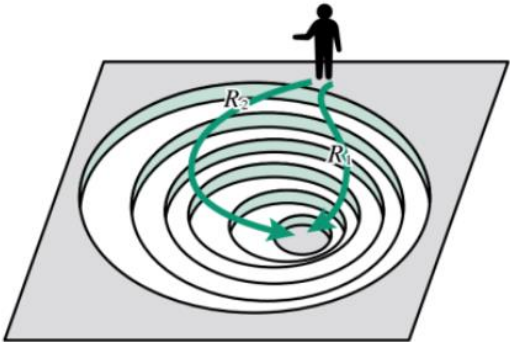
$z = f(x, y)$ 의 그래프



함수의 그래프 일부를 확대해 경사면으로 나타낸 그림. 탁구공은 가장 급한 비탈(PQ 방향)을 찾아 구르기 시작합니다.

이 탁구공의 움직임을 흉내 낸 것이 경사하강법입니다.

그림 2-48



탁구공의 움직임을 사람이 따라간다고 가정하면 사람은 최단 경로 R_1 으로 경사면의 끝(최솟값인 점)에 도착합니다.

Neural Network 기초

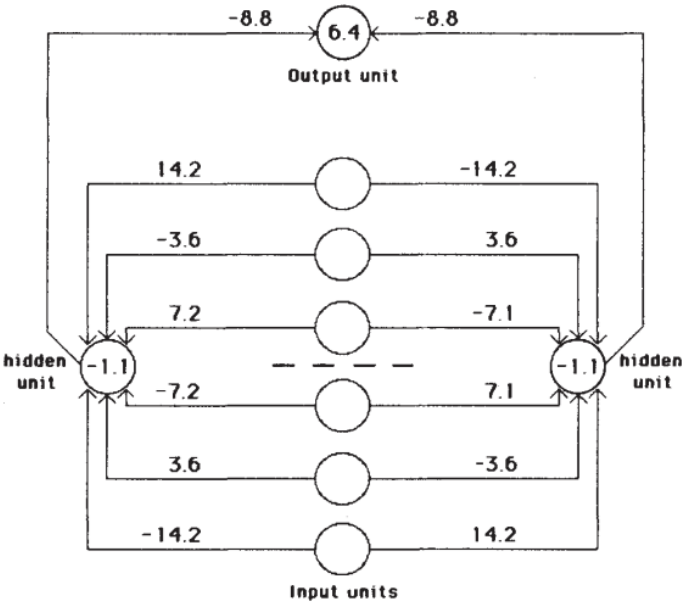
역전파 알고리즘 (back-propagation)

1986년 데이비드 러멜하트(David Rumelhart), 제프리 힌튼(Geoffrey Hinton), 로날드 윌리엄스(Ronald Williams)가 "Learning representations by back-propagating errors"라는 논문을 네이처지에 발표

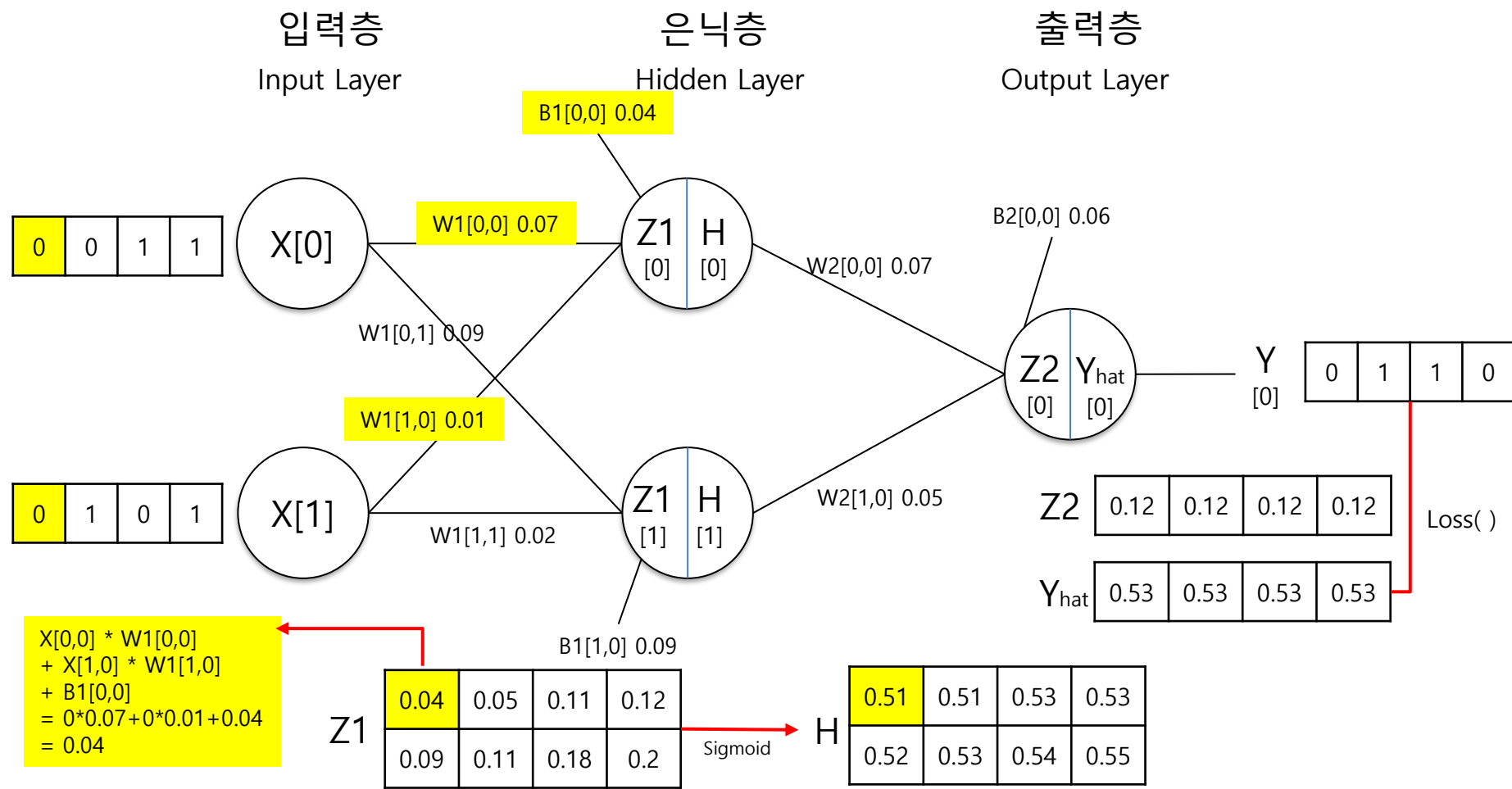
연쇄법칙(chain rule)을 이용해 오류를 출력층부터 순차적으로 전파하는 방법을 제시

$f(g(x)),$

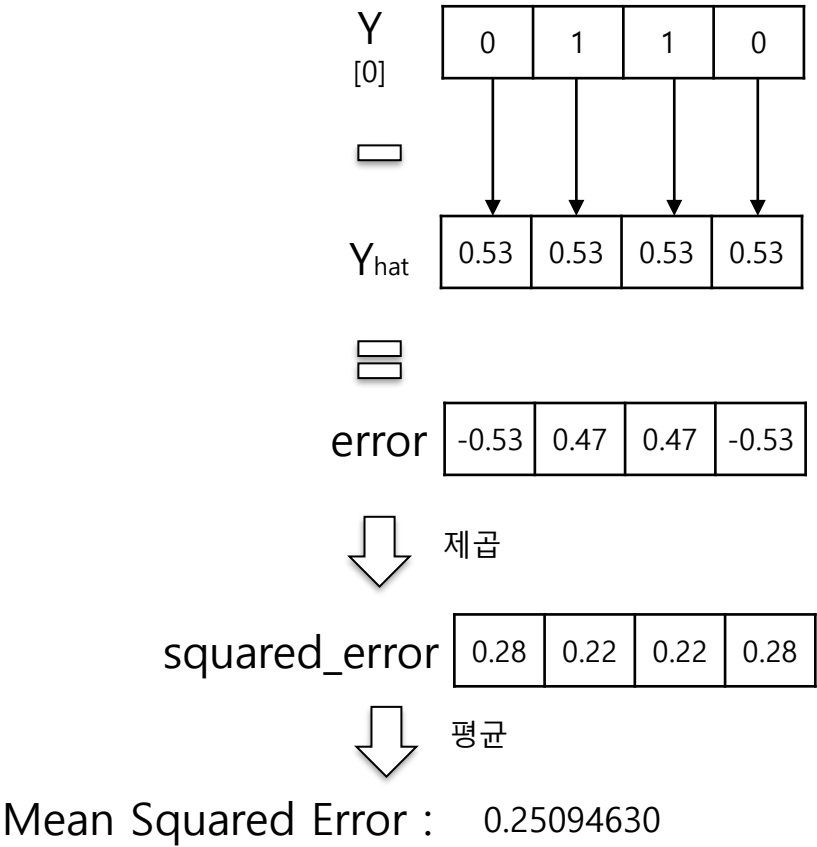
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} \times \frac{\partial g}{\partial x}$$



역전파 이해 1) Feed Forward



역전파 이해 2) Loss Function



역전파 이해 3) Back-propagation

Loss Function 값을 줄이기 위해 W2[0,0] 가중치를 얼마나 줄이면 될지를 편미분을 통해 도출

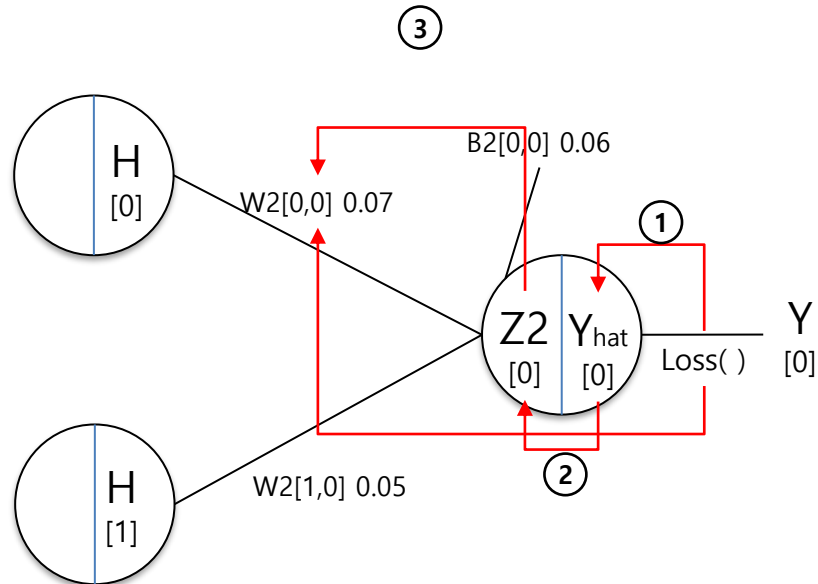
Chain Rule을 이용하면

$$\frac{\partial Loss}{\partial W2[0,0]} = \frac{\partial Loss}{\partial Yhat} \times \frac{\partial Yhat}{\partial Z2} \times \frac{\partial Z2}{\partial W2[0,0]}$$

①

②

③



① $\frac{\partial Loss}{\partial Yhat} = (Y - Yhat)^2 \frac{\partial Loss}{\partial Yhat}$

$$= 2 \times (Y - Yhat)^{2-1} \times (-1)$$
$$= -2(Y - Yhat)$$

② $\frac{\partial Yhat}{\partial Z2} = Sigmoid(Z2) * (1 - Sigmoid(z2))$

시그모이드 함수 미분 참조 (분수함수 미분공식)
http://taewan.kim/post/sigmoid_diff/

③ $\frac{\partial Z2}{\partial W2[0,0]} = W2[0,0] \cdot H[0] + W2[1,0] \cdot H[1] + B2[0,0] \frac{\partial Z2}{\partial W2[0,0]}$

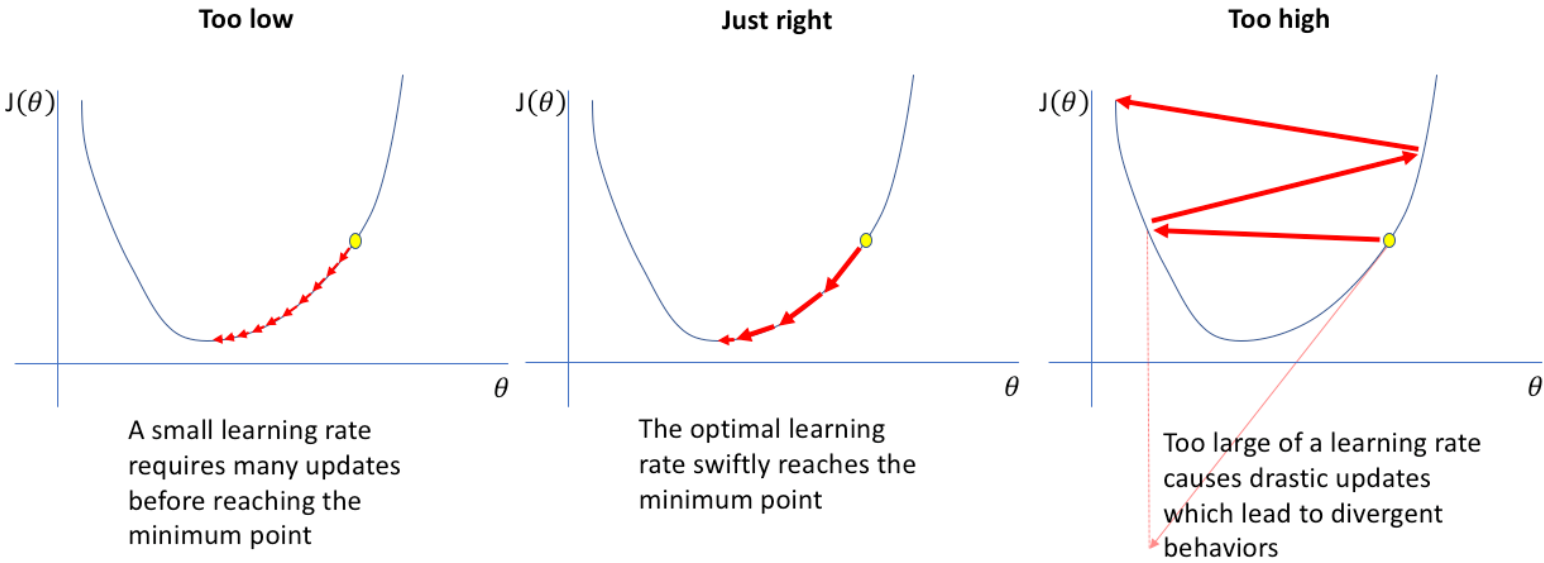
$$= H[0] + 0 + 0 = H[0]$$

역전파 이해 4) Learning Rate

편미분 값은 가중치와 바이어스 각각이 1단위 변경시 전체 손실함수에 미치는 영향을 표시
편미분 값이 큰 가중치와 바이어스를 더 많이 변경 시키는 방법이 경사하강법

미분은 분자 분모 소량의 움직임으로 순간 기울기를 계산한 결과이므로 학습률을 이용하여 학습해야 함

학습률의 크기에 따라 손실함수가 수렴하거나 발산할 수 있음



Neural Network 기초

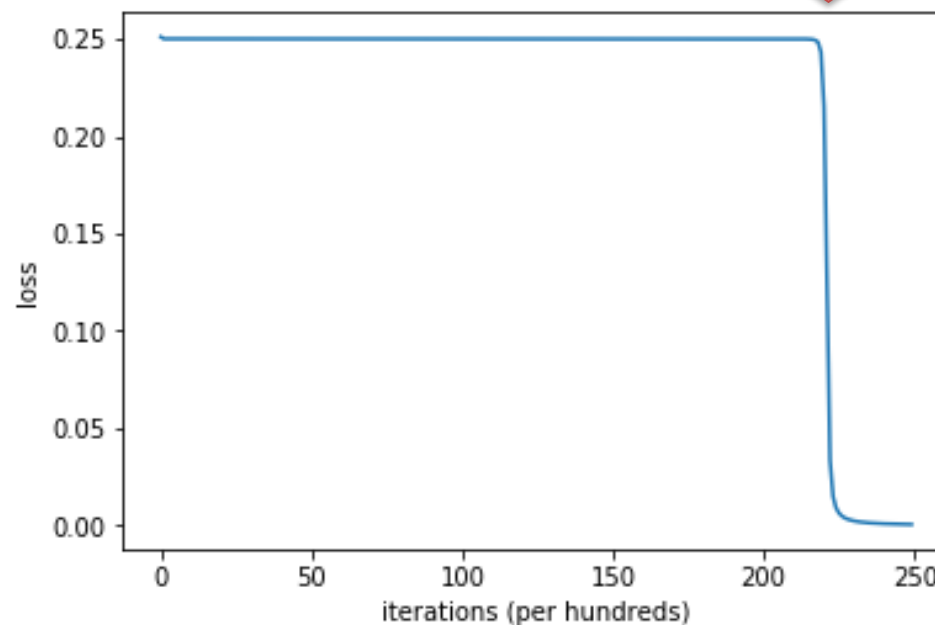
역전파 이해 5) 학습

02 다층퍼셉트론/MLP Sigmoid.py

```
new_params, loss_trace, Y_hat_predict = optimize([W1, B1, W2, B2],
learning_rate, 250000)
```

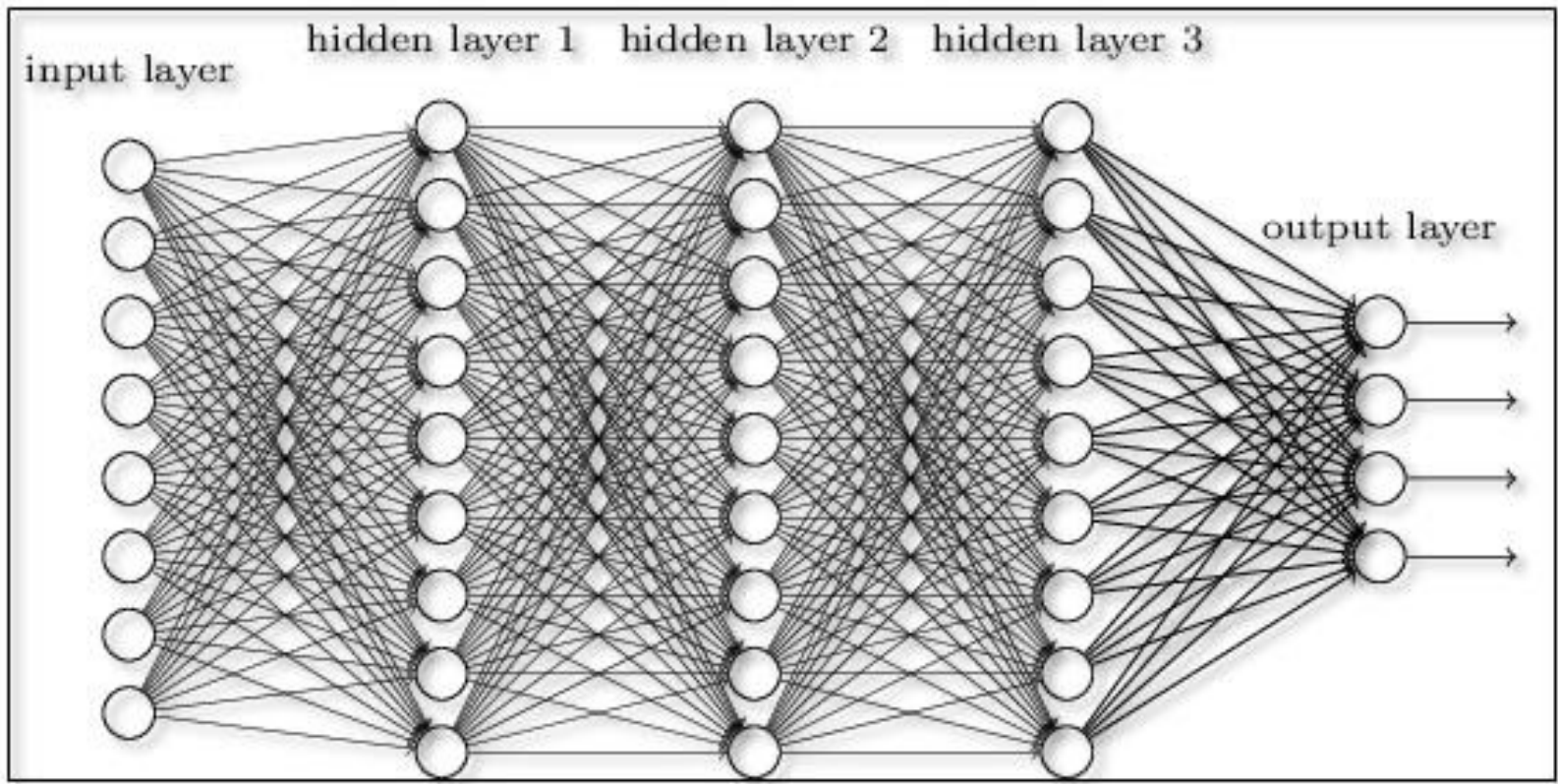
```
print(Y_hat_predict)
# Plot learning curve (with costs)
plt.plot(loss_trace)
plt.ylabel('loss')
plt.xlabel('iterations (per hundreds)')
plt.show()
```

학습 과정



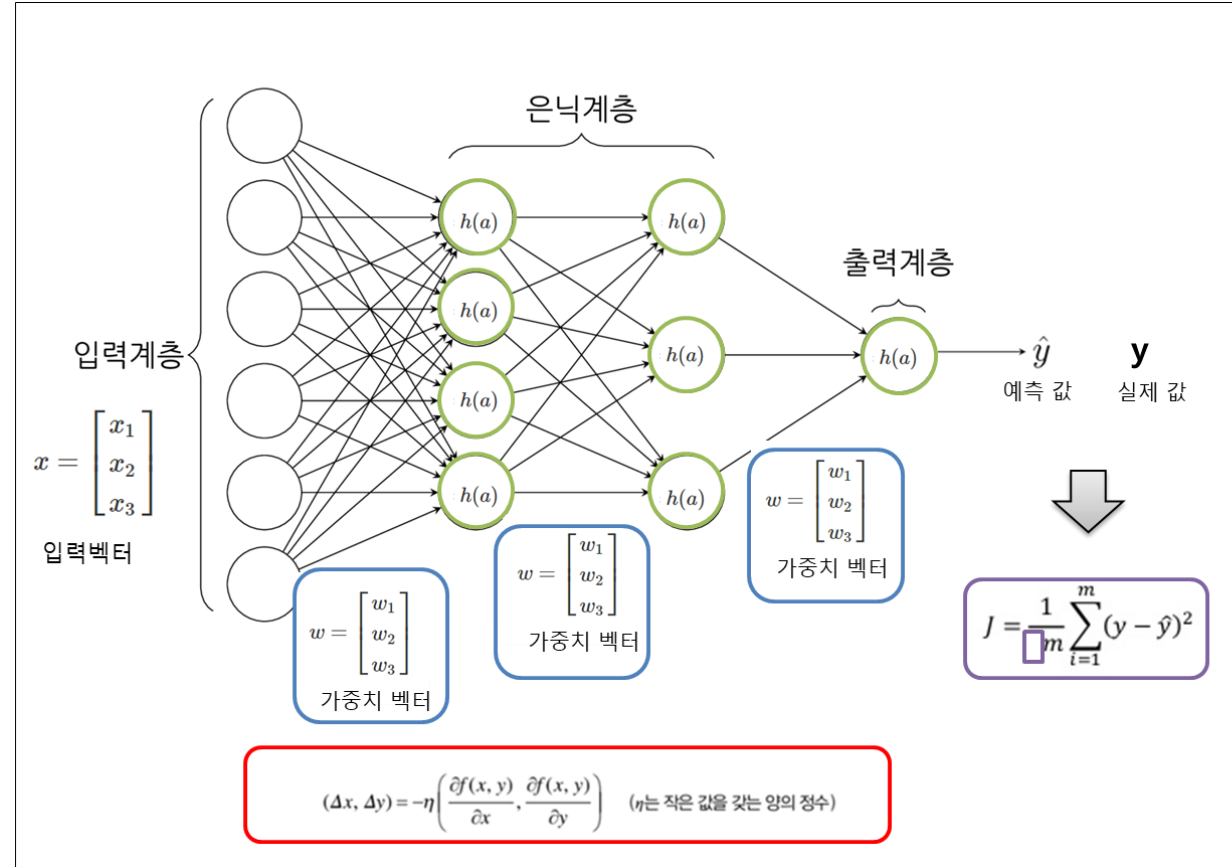
심층신경망의 한계

1. 네트워크가 깊어짐에 따라 학습시간이 너무 느리다
2. Sigmoid함수가 극단치에서 변동성이 극히 작아 학습에서 Vanishing 현상 발생
3. 초기 가중치로 인한 Local Optimal 에 빠지거나 발산하는 문제



Neural Network 기초

인공신경망의 주요 옵션





인공신경망 학습시 주요 설정 옵션


☐ 가중치 초기화 알고리즘 : HE, Glorot, Xavier


☒ 활성화함수 정의


Activation Functions


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

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

tanh
 $\tanh(x)$ 

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

ReLU
 $\max(0, x)$ 

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

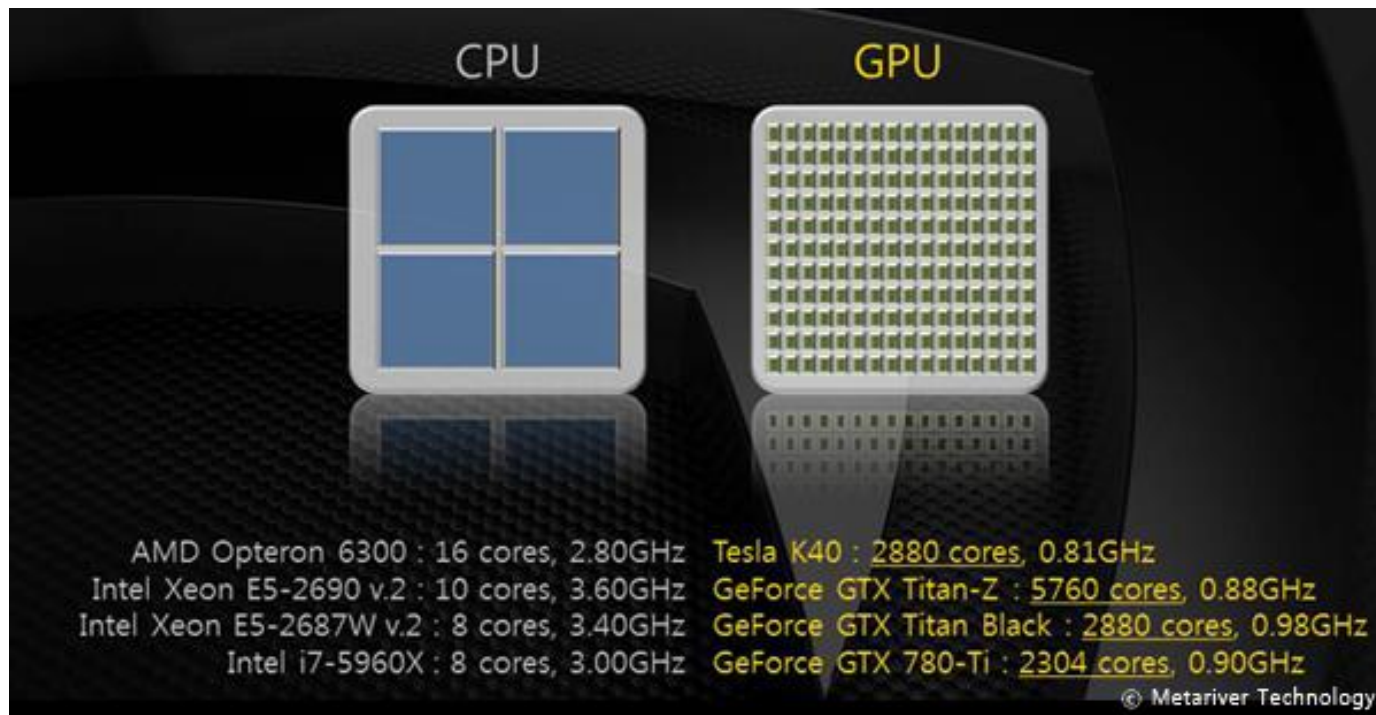
☐ Optimizer 선택(학습률)
SGD, Momentum, NAG, ADAGrad, ADADelta
Rmsprop, ADAM

☐ 손실함수 정의
Regression Loss Functions : MSE, MAE
Binary Classification Loss Functions
Multi-Class Classification Loss Functions

Neural Network 기초

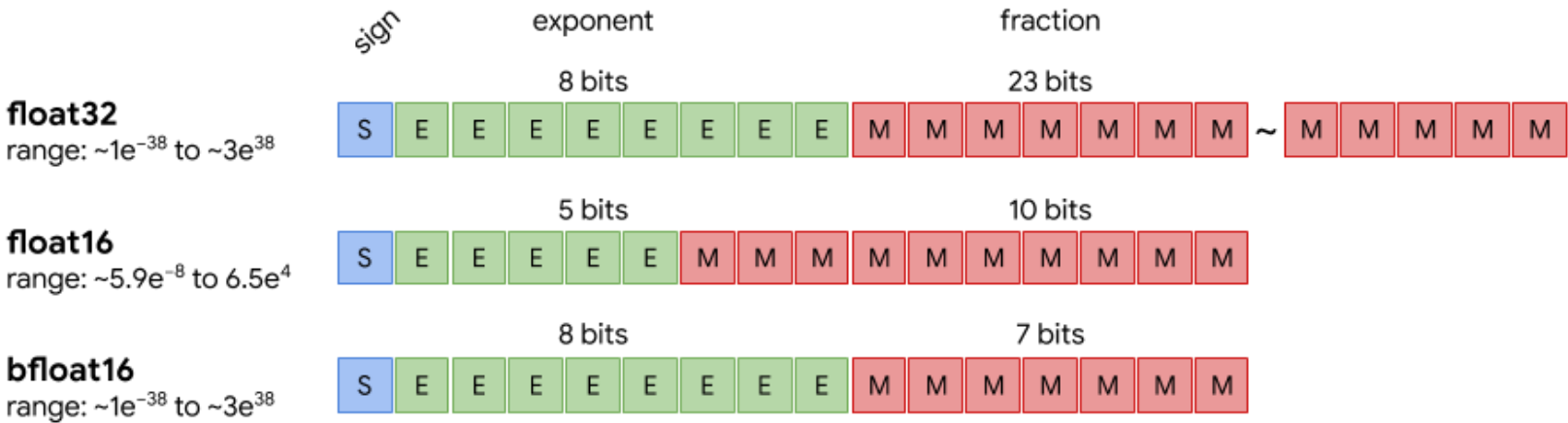
GPU > CPU 연산속도가 높은 이유 = 병렬 연산

폴리곤 단위 FLOAT 연산 > Core 개수 늘리는 방향으로 발전
3d 폴리곤 연산과 인공신경망 편미분 계산이 유사한 float 연산 과정



Neural Network 기초

CPU(float32), GPU(float16), TPU(bfloat16)



CPU



GPU



TPU

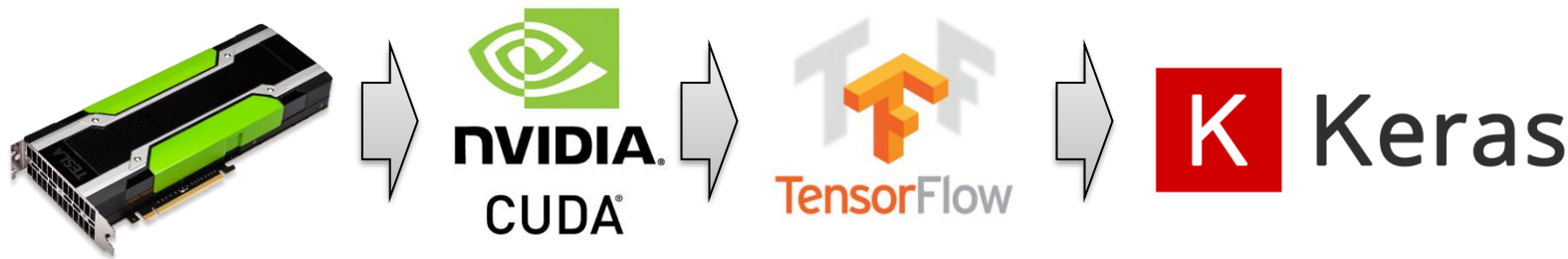
Platform	Unit	Version	Mem Type	Mem (GB)	Mem Bdw (GB/s)	Peak FLOPS
CPU	1 VM	Skylake	DDR4	120	16.6	2T SP [†]
GPU (DGX-1)	1 Pkg	V100 (SXM2)	HBM2	16	900	125T
TPU	1 Board (8 cores)	v2	HBM	8	2400	180T
TPUv3	8 cores	v3	HBM	16	3600*	420T




[†] Single precision: $2 \text{ FMA} \times 32 \text{ SP} \times 16 \text{ cores} \times 2 \text{G frequency} = 2 \text{ SP TFLOPS}$
* Estimated based on empirical results (Section 4.5).

Table 3: Hardware platforms under study.

Neural Network 기초

딥러닝 프레임워크를 활용하면 매우 간단함



	Keras 	TensorFlow 	PyTorch 
Level of API	high-level API ¹	Both high & low level APIs	Lower-level API ²
Speed	Slow	High	High
Architecture	Simple, more readable and concise	Not very easy to use	Complex ³
Debugging	No need to debug	Difficult to debugging	Good debugging capabilities
Dataset Compatibility	Slow & Small	Fast speed & large	Fast speed & large datasets
Popularity Rank	1	2	3
Uniqueness	Multiple back-end support	Object Detection Functionality	Flexibility & Short Training Duration
Created By	Not a library on its own	Created by Google	Created by Facebook ⁴
Ease of use	User-friendly	Incomprehensive API	Integrated with Python language
Computational graphs used	Static graphs	Static graphs	Dynamic computation graphs ⁵

1. 신경망 Neural Network

2. keras

Neural Network 이해

Keras 프레임워크 사용

인공신경망 생산성이 가장 빠른 keras를 사용

더 상세한 튜닝 및 커스터마이징을 하고 싶은 경우 pytorch(페이스북)나 tensorflow(구글)를 사용



<https://keras.io/api/>

<https://keras.io/ko/> << 한글도 있음.. 일부는 없음

케라스 documentation 을 기준으로 작성

상세 내용은 공식 다큐먼트 참고

Neural Network 이해

Keras 프레임워크 사용

Keras를 이용하면 NN 기초에서 했던 XOR 신경망 모델을 5줄이면 구현 및 학습이 가능
다양한 옵션을 이용하면 손쉽게 다양한 인공신경망 분석이 가능
연구 속도면에서 뛰어남

KERAS XOR 모델.py

```
# -*- coding: utf-8 -*-
from keras.models import Sequential #개별 레이어를 선형적으로 적제하기 위한 모델
from keras.layers import Dense #일반적인 형태의 뉴럴네트워크 계층 / 앞선 학습에 사용한 은닉/출력층에 해당
from keras import optimizers

model = Sequential() #모델을 선언한다
model.add(Dense(units=2, activation='sigmoid', input_shape=(2,))) #은닉층 2개 추가, 활성화함수 시그모이드
model.add(Dense(units=1, activation='sigmoid')) # 출력층 1개 추가, 활성화함수 시그모이드

model.summary() #모델 요약

#loss함수와 학습률결정모델(optimizer) 선택하여 모델 컴파일
model.compile(loss='mse', optimizer=optimizers.SGD(learning_rate=0.1))

#학습을 시작한다. 백번 돌아 주세요.
history = model.fit(x_train, y_train, epochs=100, batch_size=1)
```


Neural Network 이해

Regression vs Neural Network

회귀분석 과정과 NN 분석 과정의 차이를 이해
생활인구 집계구역별 건축물용도와 토지용도 중 공원의 면적비율을 이용하여 18년~20년 20대 인구차이 분석

```
sm.ols(formula = '인구차이 ~ 주택 + 업무 + 상가 + 교육 + 공원 + 1', data = learning).fit()
```

절편이 0인 회귀분석 진행

Summary를 통해 분석 결과 확인

OLS Regression Results						
=====						
Dep. Variable:	인구차이		R-squared:	0.052		
Model:	OLS		Adj. R-squared:	0.032		
Method:	Least Squares		F-statistic:	2.590		
Date:	Thu, 21 Oct 2021		Prob (F-statistic):	0.0265		
Time:	00:30:50		Log-Likelihood:	703.97		
No. Observations:	242		AIC:	-1396.		
Df Residuals:	236		BIC:	-1375.		
Df Model:	5					
Covariance Type:	nonrobust					
=====						
	coef	std err	t	P> t	[0.025	0.975]

Intercept	-0.0090	0.001	-7.305	0.000	-0.011	-0.007
주택	-0.0003	0.001	-0.440	0.661	-0.001	0.001
업무	0.0020	0.001	2.431	0.016	0.000	0.004
상가	0.0001	5.86e-05	2.191	0.029	1.29e-05	0.000
교육	-0.0029	0.002	-1.225	0.222	-0.007	0.002
공원	-0.0242	0.024	-1.004	0.316	-0.072	0.023
=====						
Omnibus:	33.283		Durbin-Watson:	1.780		
Prob(Omnibus):	0.000		Jarque-Bera (JB):	41.823		
Skew:	0.994		Prob(JB):	8.28e-10		
Kurtosis:	2.561		Cond. No.	418.		
=====						

Intercept	주택	업무	상가	교육	공원	rSquared
-0.00895863	-0.000268483	0.00199243	0.000128375	-0.00287556	-0.0241726	5.2011

Neural Network 이해

독립변수와 종속변수 세트를 numpy 배열로 변경

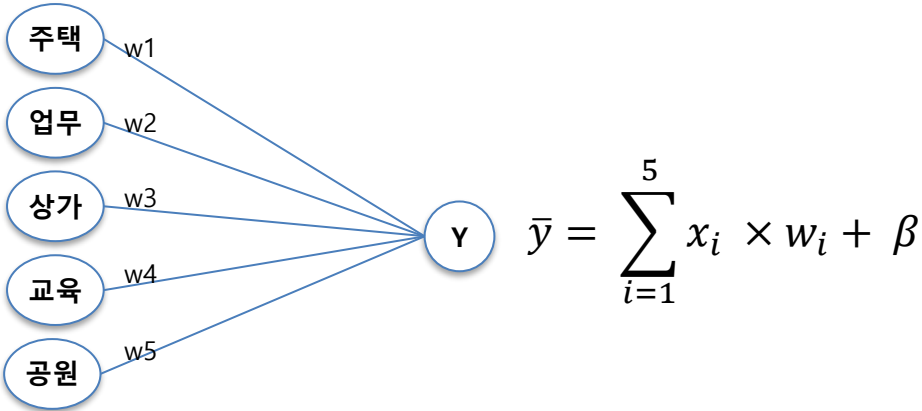
```
x_train = np.array(learning2[['주택', '업무', '상가', '교육', '공원']].values)
y_train = np.array(learning2[['인구차이']].values)
```

NN를 1개 Dense Layer로 Activation 함수가 없는 Linear모델로 Bias 없이 진행하면 Regression 과 동일
model.add(Dense(units=1, activation='linear', input_shape=(5,), use_bias=False))

```
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 1)	5
Total params: 5		
Trainable params: 5		
Non-trainable params: 0		



Neural Network 이해

역전파 학습을 진행한 후 학습 가중치를 확인하면 회귀분석과 비교 가능

	Intercept	주택	업무	상가	교육	공원	rSquared
회귀분석	-0.00895863	-0.000268483	0.00199243	0.000128375	-0.00287556	-0.0241726	5.2011
Neural Net	-0.00751105	-0.00102376	0.00177757	0.000139327	-0.00381043	-0.0449258	4.20124

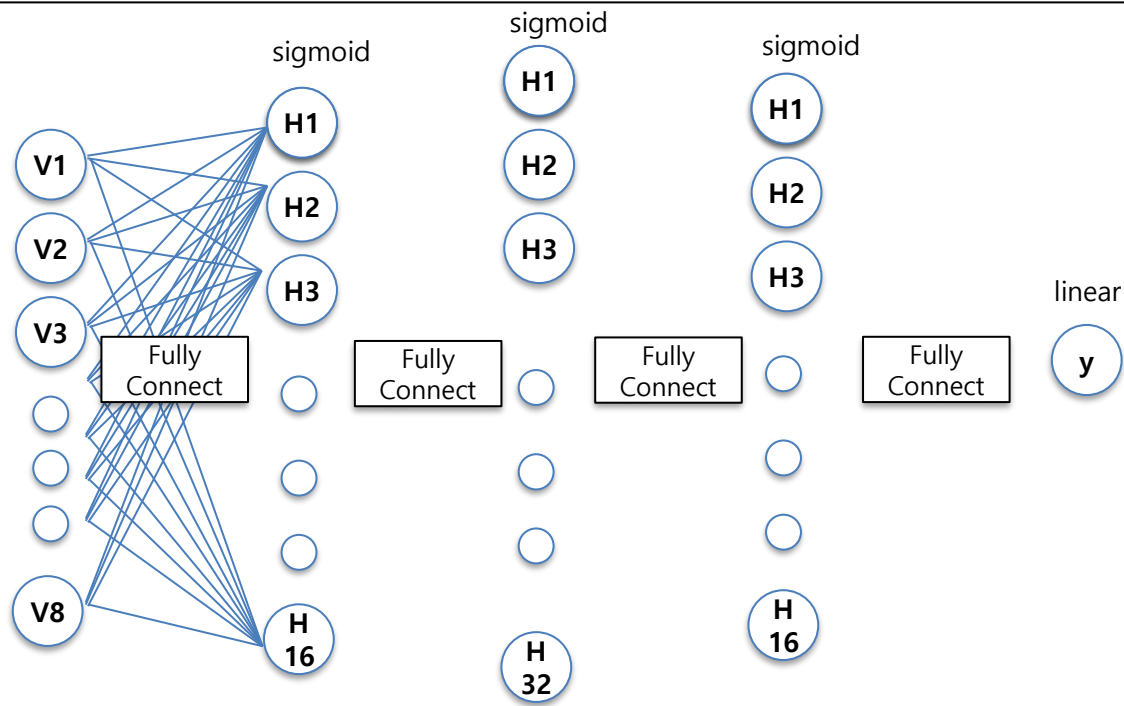
`model.layers[0].get_weights()` << 모델의 첫번째 레이어의 가중치 목록을 가져온다
`model.layers[0].set_weights()` << 모델의 첫번째 레이어 가중치를 변경한다.

회귀분석 결과를 첫번째 가중치 값으로 강제 할당하면 NN의 예측 결과와 회귀분석 예측결과가 동일하게 나옴

Neural Network 이해

NN의 비선형성을 반영하기 위해서 Hidden Layer를 추가하려면
Dense Layer를 중간에 추가하고 비선형성 관계를 나타내기 위해 활성화함수를 relu로 지정
비선형성이 추가되어 학습 횟수가 늘어나면 학습률이 개선됨

```
model = Sequential() #모델을 선언한다
model.add(Dense(units=16, activation=relu', input_shape=(5,)))
model.add(Dense(units=32, activation= relu '))
model.add(Dense(units=16, activation= relu '))
model.add(Dense(units=1, activation='linear'))
model.summary() #모델 요약
```



Neural Network 이해

<https://keras.io/api/models/> 참고

keras.models

뉴럴네트워크를 구성하기 위해 최상위에 선언해야 하는 클래스

Sequential() 함수를 통해 인스턴스를 생성하면 일렬로 나열된 모델을 생성할 수 있고

순서대로 Layers를 이어붙이면 순차형 뉴럴네트워크를 구성합니다.

첫 번째 Layers에는 input_shape를 통해 독립변수 구조를 지정해 줘야 합니다.

두 번째부터는 앞의 Layers의 units 수만큼을 input으로 받는 순차 모델이 됩니다.

```
model = Sequential() #모델을 선언한다
model.add(Dense(units=16, activation='sigmoid', input_shape=(5,), use_bias=False))
model.add(Dense(units=32, activation='sigmoid', use_bias=False))
model.add(Dense(units=16, activation='sigmoid', use_bias=False))
model.add(Dense(units=1, activation='linear', use_bias=False))
model.summary() #모델 요약
```

Model()함수를 통해서도 모델 인스턴스를 생성할 수 있으며,

Input 클래스와 Layers클래스로 구성하여 Model을 생성할 수 있습니다.

위와 동일한 모델을 Model()함수로 선언하면 아래와 같다. 요약을 보면 인풋 레이어가 추가되어 있음

```
inputs = Input(shape=(5,))
h1 = Dense(16, activation='sigmoid', use_bias=False)(inputs)
h2 = Dense(32, activation='sigmoid', use_bias=False)(h1)
h3 = Dense(16, activation='sigmoid', use_bias=False)(h2)
outputs = Dense(1, activation='linear', use_bias=False)(h3)
model = Model(inputs, outputs)
model.summary()
```

Neural Network 이해

<https://keras.io/api/layers/> 참고

keras.layers

layers는 뉴럴네트워크의 한 개 층을 구성하는 클래스로 layers.Layer를 상속받아서 다양하게 제공

Core Layer

Input Object : 입력 레이어 Model() 선언할때 필요

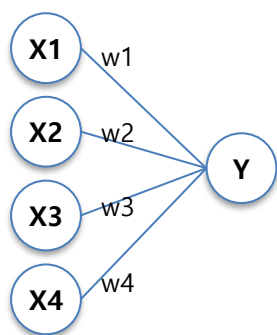
Dense Layer : Fully Connected 레이어를 생성

Activation Layer : 입력된 모든 값을 각각의 Activation 함수를 통과시켜서 동일한 수의 Output을 뱉는다

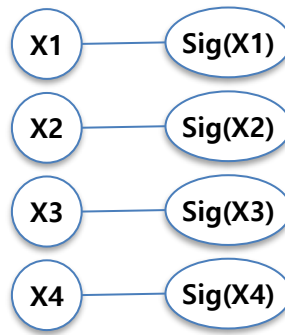
Embedding Layer : 자연어처리에서 쓰이며 문자열 개수 만큼 원핫 인코딩 생성

Masking Layer : 자연어 문장 입력 이나 시계열 자료의 결측치를 특정 값으로 채울 때 사용

Lambda Layer : 사용자 정의 레이어 함수나 표현식으로 레이어를 생성할때 사용



Dense Layer



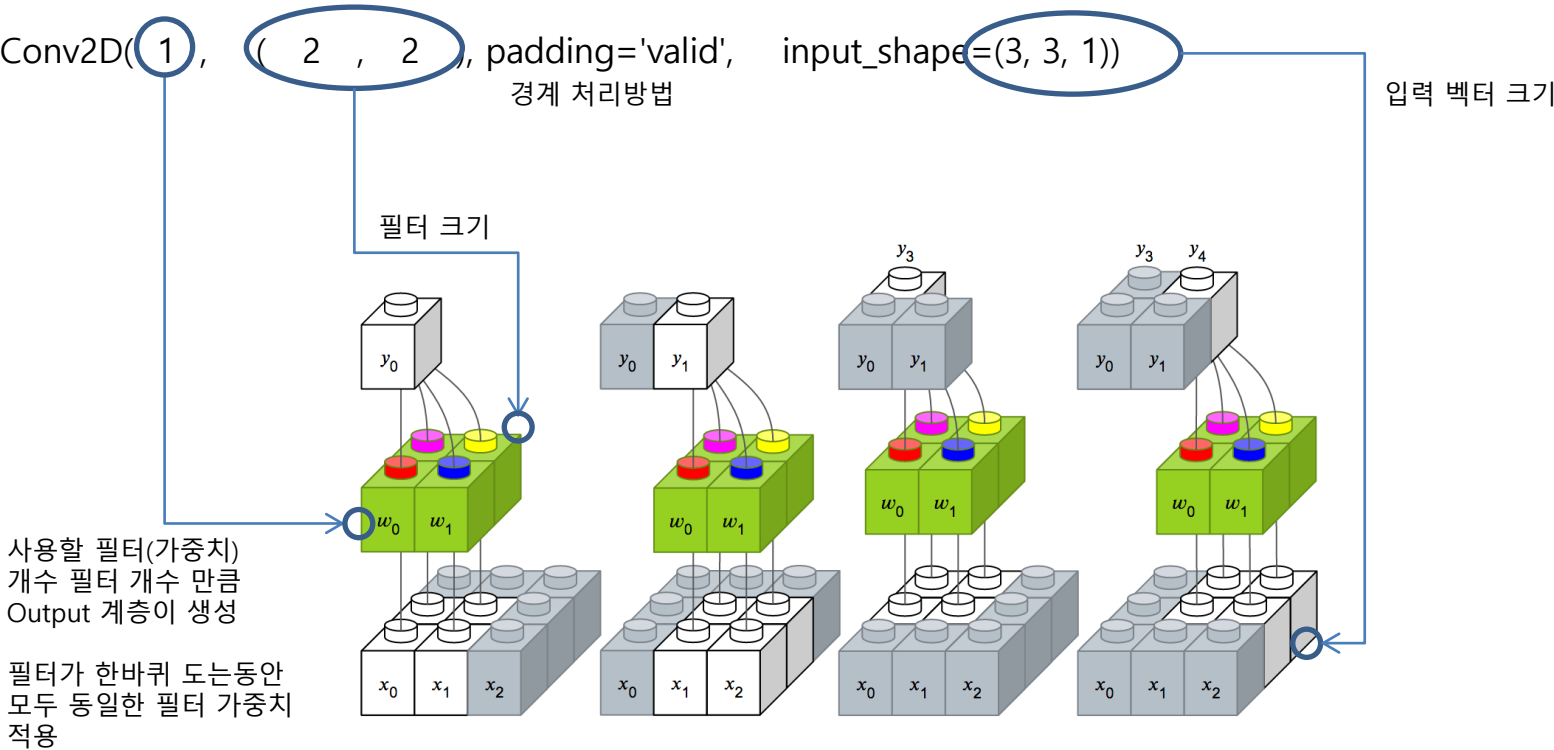
Activation Layer

Neural Network 이해

https://tykimos.github.io/2017/01/27/CNN_Layer_Talk/

keras.layers

Convolution Layer : 1차원~3차원 벡터에 활용하는 합성곱 레이어, 이미지 인식에 많이 활용



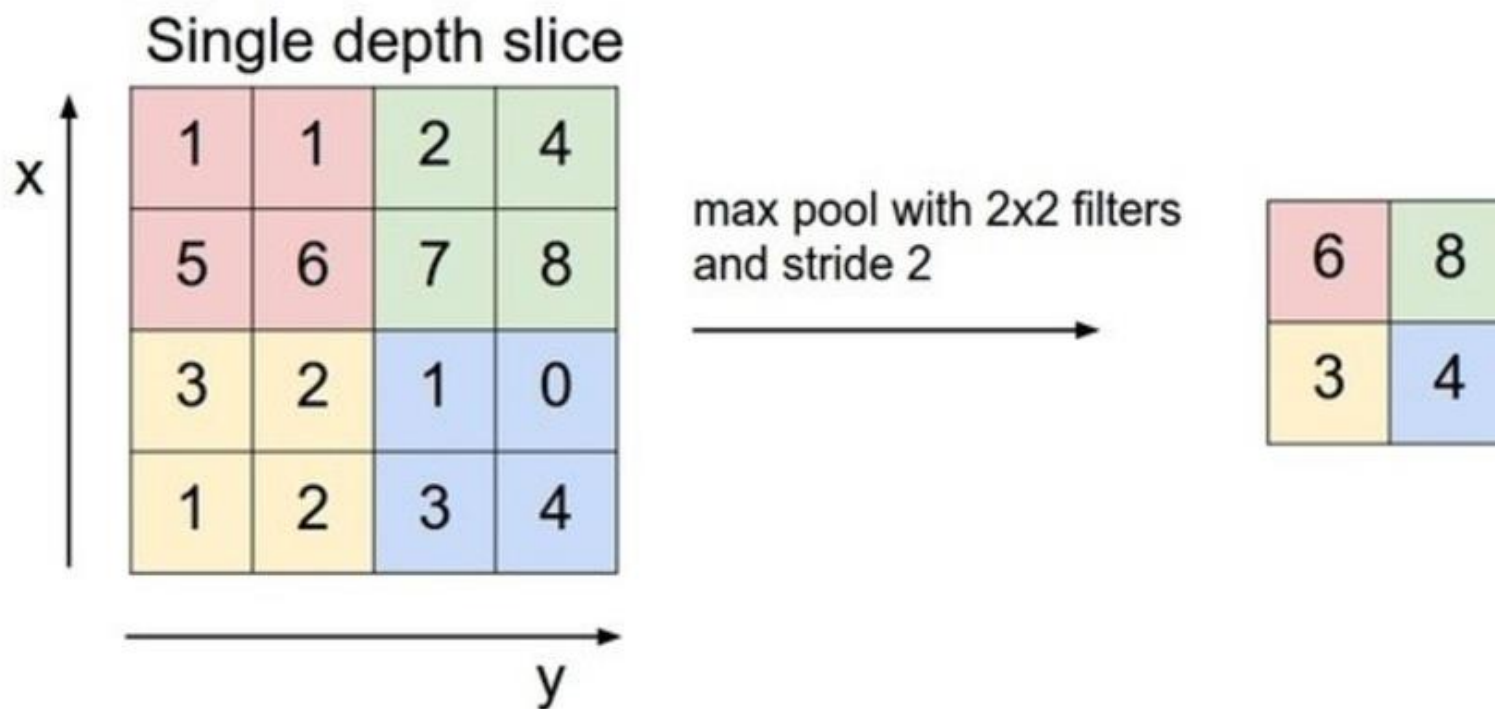
Neural Network 이해

<https://bcho.tistory.com/1149>

keras.layers

Pooling Layer : 인풋정보를 특정한 규칙으로 단순화 시켜 정보량을 축소 시킴, 이미지인식에 활용

아래는 maxpooling을 사용하는 경우



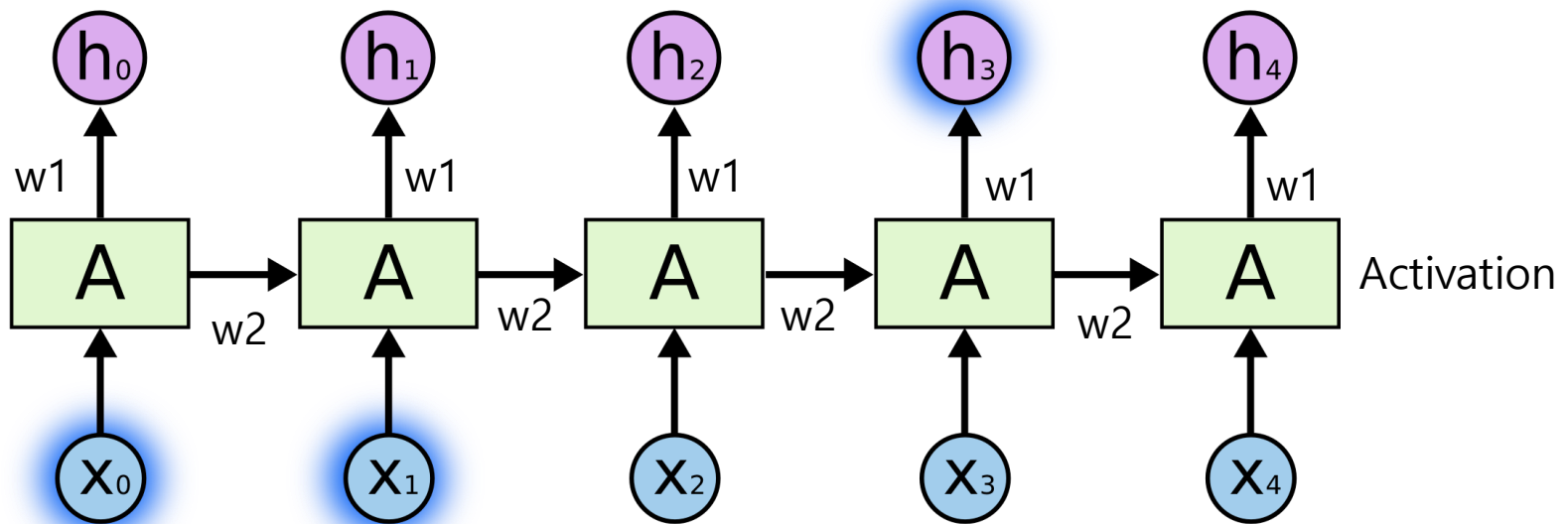
Neural Network 이해

<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

keras.layers

Recurrent Layer : 순환신경망 레이어로 input 열을 동일한

동일한 가중치와 활성화함수로 입력 벡터 수와 출력 벡터 수 만큼 순환 학습 하는 신경망 층
이전 학습



Neural Network 이해

<https://keras.io/api/layers/> 참고

keras.layers

학습 전 전처리가 가능한 레이어

Preprocessing Layer

[Resizing layer](#)

[Rescaling layer](#)

[CenterCrop layer](#)

[RandomCrop layer](#)

[RandomFlip layer](#)

[RandomTranslation layer](#)

[RandomRotation layer](#)

[RandomZoom layer](#)

[RandomHeight layer](#)

[RandomWidth layer](#)

Normalization Layer

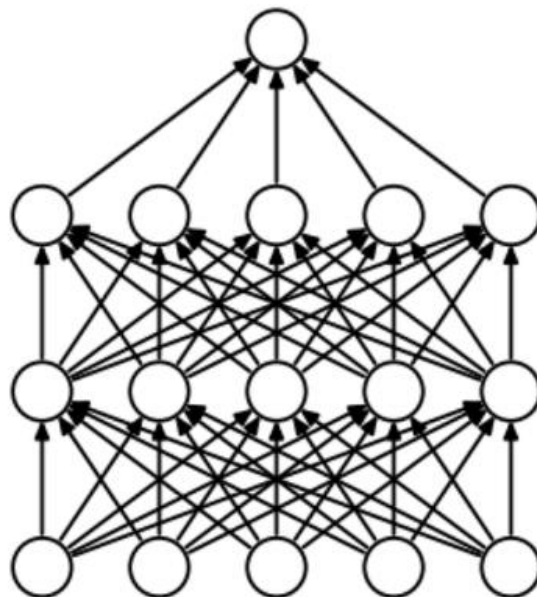
Neural Network 이해

<https://keras.io/api/layers/> 참고

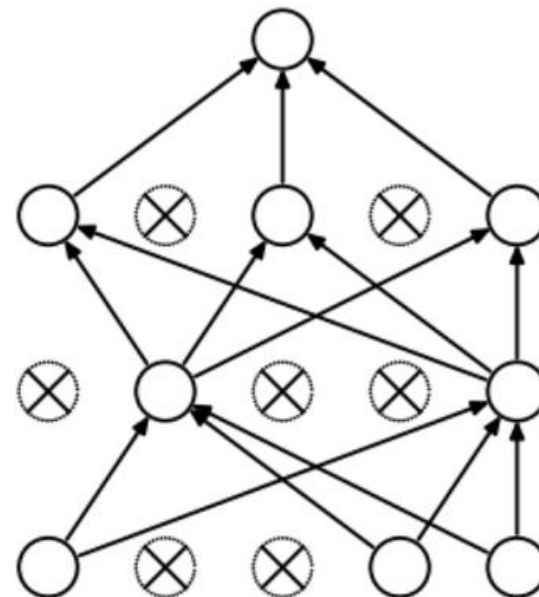
keras.layers

과적합을 피하기 위해 사용하는 레이어

Regularizaion Layer = Dropout Layer



(a) Standard Neural Net



(b) After applying dropout.

Neural Network 이해

<https://keras.io/api/layers/> 참고

keras.layers

이전 레이어에서 받은 벡터의 차원을 변경하는 레이어

Reshaping Layer : reshape, flatten 많이 사용

Conv2d에서 Flatten을 통해 Dense로 이동

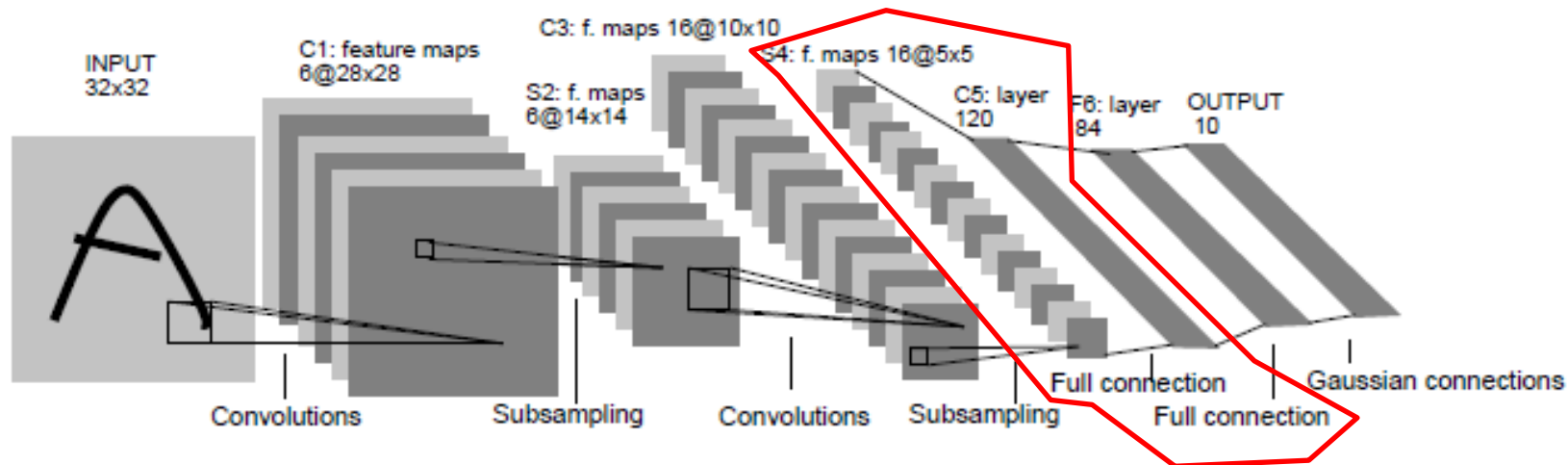


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Neural Network 이해

<https://keras.io/api/layers/> 참고

keras.layers

두 개 레이어를 연결하는데 활용 가능한 레이어

Merging Layer

Concatenate Layer : 두 개 이상의 네트워크를 횡으로 연결
아웃풋 벡터 구조 달라도 가능

Average Layer : 동일한 아웃풋 벡터의 각 평균 값

Maximum Layer

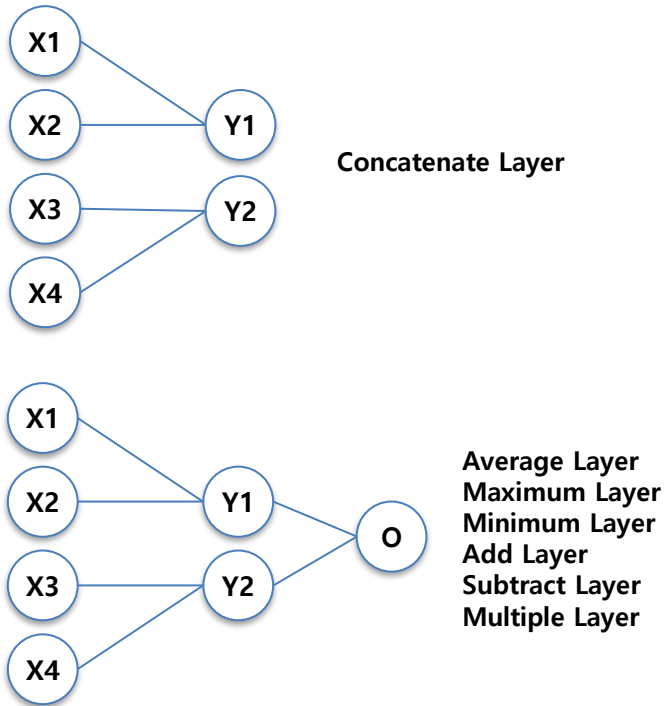
Minimum Layer

Add Layer

Subtract Layer

Multiply Layer

Dot Layer : 두 네트워크 아웃풋의 행렬 곱 연산



Neural Network 이해

<https://keras.io/api/layers/> 참고

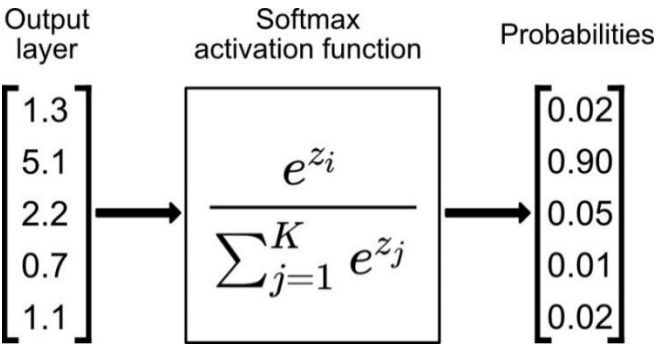
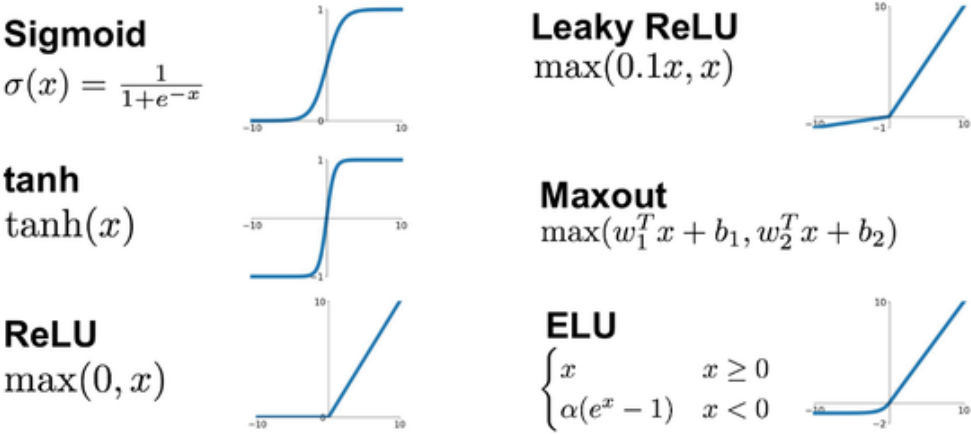
keras.layers.activation

활성함수는 역전파 학습을 위해 미분가능 함수를 사용해야 함

keras에서는 기본 활성함수로 relu, sigmoid, softmax, softplus, softsign, tanh, selu, elu, exponential 을 제공

고급 활성함수로 LeakyReLU, PReLU, ThresholdedReLU 등 제공

Activation Functions



Neural Network 이해

<https://keras.io/api/layers/> 참고

keras.layers.initializers

각레이어의 가중치와 bias 값의 초기 값을 설정하는 클래스

Layer weight initializers

[RandomNormal class](#)

[RandomUniform class](#)

[TruncatedNormal class](#)

[Zeros class](#)

[Ones class](#)

[GlorotNormal class](#)

[GlorotUniform class](#)

[Identity class](#)

[Orthogonal class](#)

[Constant class](#)

[VarianceScaling class](#)

Layer weight regularizers

[l1 class](#)

[l2 class](#)

[l1_l2 function](#)

Layer weight constraints

[MaxNorm class](#)

[MinMaxNorm class](#)

[NonNeg class](#)

[UnitNorm class](#)

[RadialConstraint class](#)

Neural Network 이해

https://keras.io/api/models/model_training_apis/#compile-method

keras.models.compile()

모델을 만들었으면 해당 모델을 학습하기 전에 compile()함수를 이용해서 학습 속성을 설정
대표적으로는 학습률에 관여하는 optimizer와
역전파 학습의 목표에 해당하는 loss 손실함수 설정

compile method

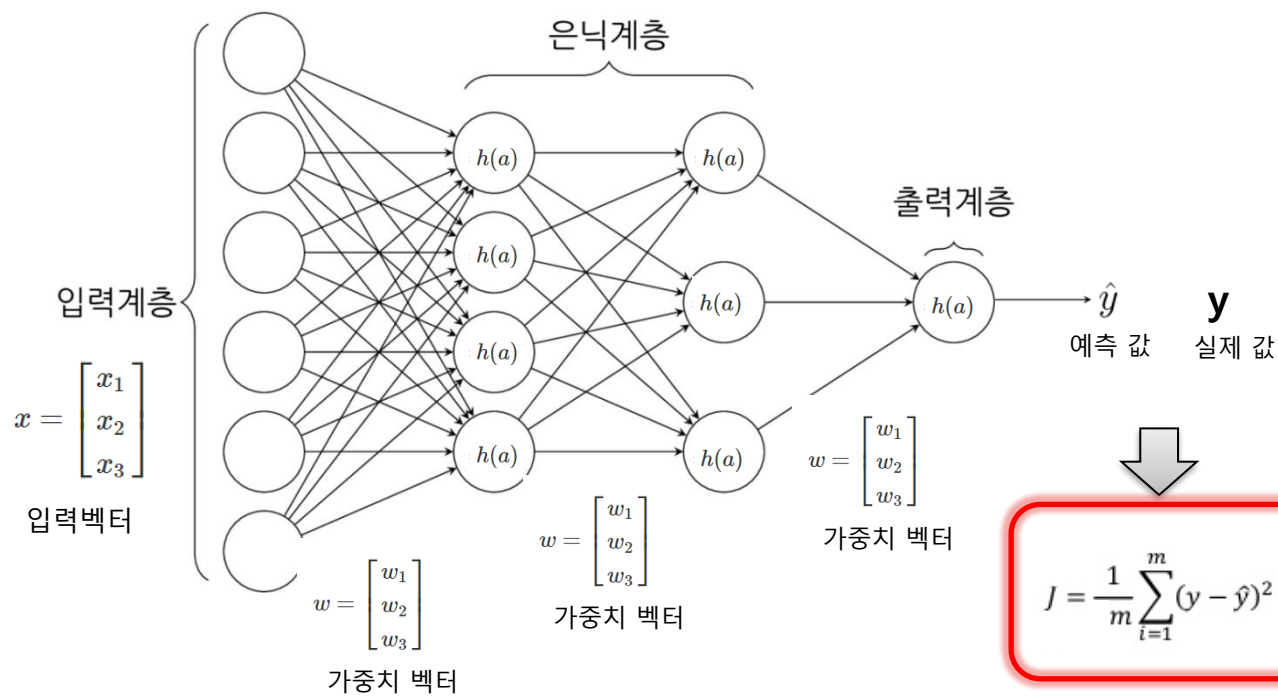
```
Model.compile(  
    optimizer="rmsprop",  
    loss=None,  
    metrics=None,  
    loss_weights=None,  
    weighted_metrics=None,  
    run_eagerly=None,  
    steps_per_execution=None,  
    **kwargs  
)
```


Neural Network 이해

<https://keras.io/api/losses/>

keras.losses

학습에 사용할 손실함수를 정의한다.
손실 함수는 미분 가능해야 하며, 손실함수가 역전파의 시작점이자 손실함수 최소화가 목표



- Regression Loss Functions
 - Mean Squared Error Loss
 - Mean Squared Logarithmic Error Loss
 - Mean Absolute Error Loss
- Binary Classification Loss Functions
 - Binary Cross-Entropy
 - Hinge Loss
 - Squared Hinge Loss
- Multi-Class Classification Loss Functions
 - Multi-Class Cross-Entropy Loss
 - Sparse Multiclass Cross-Entropy Loss
 - Kullback Leibler Divergence Loss

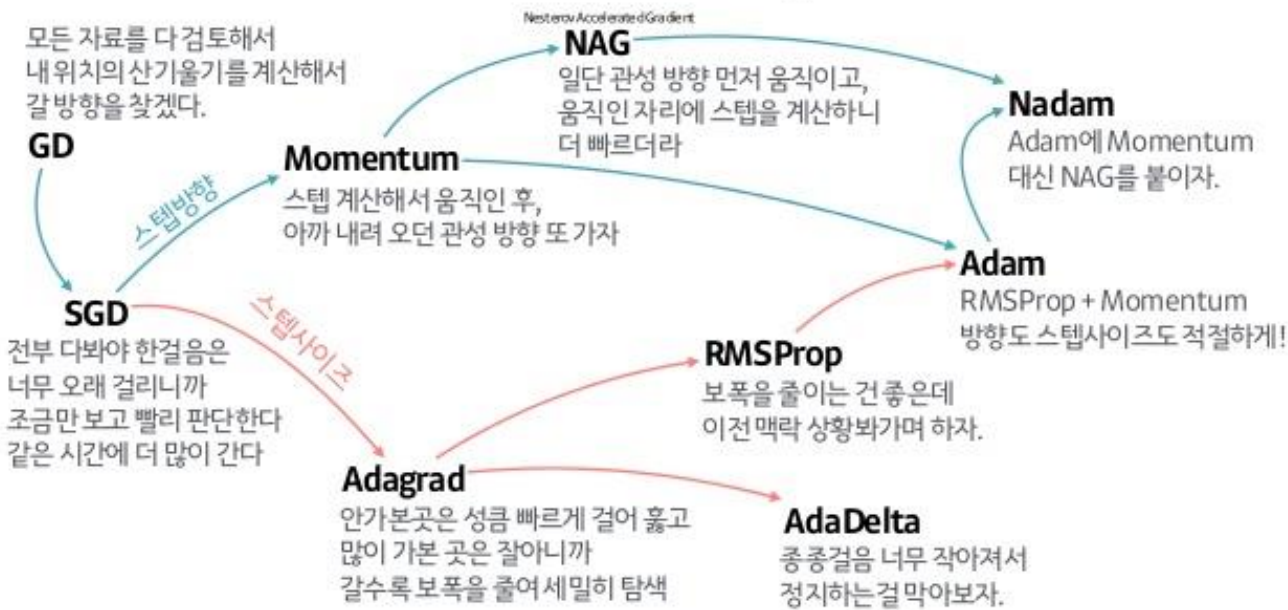
$$J = \frac{1}{m} \sum_{i=1}^m (y - \hat{y})^2$$

$$(\Delta x, \Delta y) = -\eta \left(\frac{\partial f(x, y)}{\partial x}, \frac{\partial f(x, y)}{\partial y} \right) \quad (\eta \text{는 작은 값을 갖는 양의 정수})$$

keras.optimizers

학습에 사용할 학습방법 설정(학습률 결정 방법)

산 내려오는 작은 오솔길 찾기(Optimizer)의 발달 계보



Neural Network 이해

https://keras.io/api/models/model_training_apis/#compile-method

keras.models.fit()

모델 학습을 시작하는 명령어로 학습에 사용할 x(학습 독립변수), y(학습 종속변수) 를 지정

batch_size는 한번에 역전파를 진행할 데이터 개수, epochs는 전체 학습 데이터를 몇 바퀴 학습시킬건지

Validation_data로 검증데이터를 따로 지정하거나, validation_split으로 학습데이터를 %로 나눠서 사용 가능

fit method

```
Model.fit(  
    x=None,  
    y=None,  
    batch_size=None,  
    epochs=1,  
    verbose="auto",  
    callbacks=None,  
    validation_split=0.0,  
    validation_data=None,  
    shuffle=True,  
    class_weight=None,  
    sample_weight=None,  
    initial_epoch=0,  
    steps_per_epoch=None,  
    validation_steps=None,  
    validation_batch_size=None,  
    validation_freq=1,  
    max_queue_size=10,  
    workers=1,  
    use_multiprocessing=False,  
)
```

Neural Network 이해

https://keras.io/api/models/model_training_apis/#compile-method

keras.models.predict()

학습된 모델에 독립변수 셋을 입력하여 예측 종속변수를 추출하는 함수

predict method

```
Model.predict(  
    x,  
    batch_size=None,  
    verbose=0,  
    steps=None,  
    callbacks=None,  
    max_queue_size=10,  
    workers=1,  
    use_multiprocessing=False,  
)
```

모델을 중간에서 잘라내면 중간 레이어의 output정보를 얻어 낼 수 있음

03 KERAS 예시/003 KERAS 기초 이해.py 참고

```
new_model = Model(model.input,model.layers[3].output)  
new_model.summary()  
predict_h3 = new_model.predict(x_train)
```