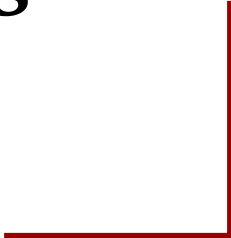



# Basics of Deep Learning



- 
1. Deep Learning
  2. Perceptron
  3. Activation Function
  4. Training Neural Network
  5. Problems in Training
  6. Skills for Training

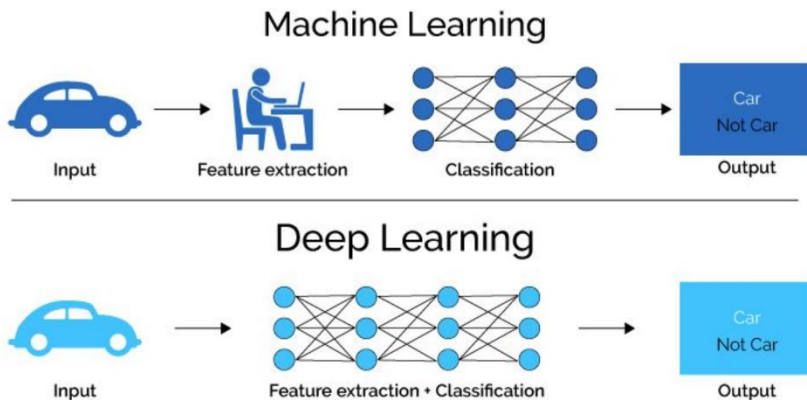
# Deep Learning & Machine Learning

- Deep Learning  $\subset$  Machine Learning

- 심층신경망 (Deep Neural Network) 를 이용한 기법
- 딥러닝은 더 많은 데이터를 처리할 때 유리하고, 인간의 힘을 덜 들여도 된다.

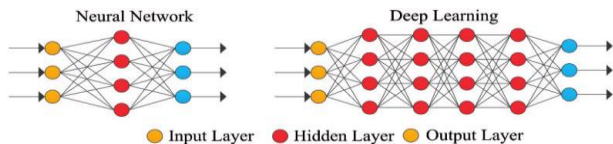
- 머신러닝과의 차이점

- 1) 데이터
- 2) 하드웨어
- 3) 변수 추출 (Feature Extraction)

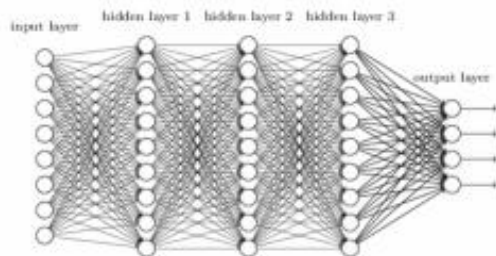


출처: <https://www.quora.com/What-is-the-difference-between-deep-learning-and-usual-machine-learning>

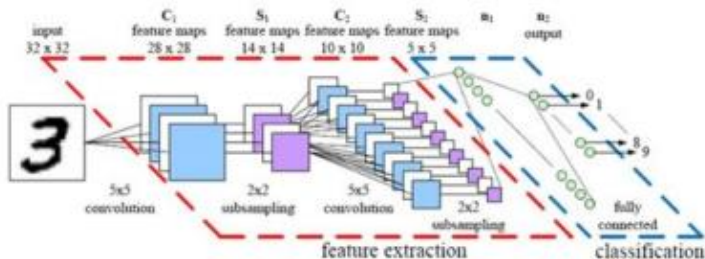
# Deep Learning 의 구조



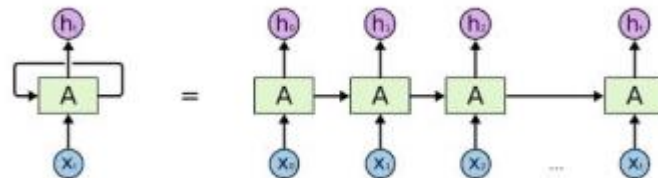
- DNN (Deep Neural Networks)



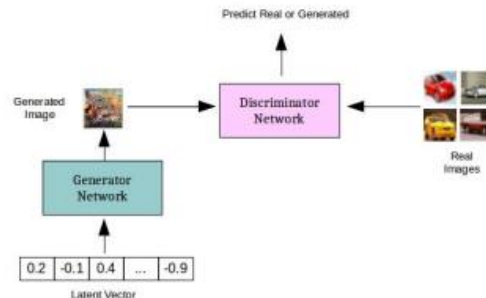
- CNN (Convolutional Neural Networks)



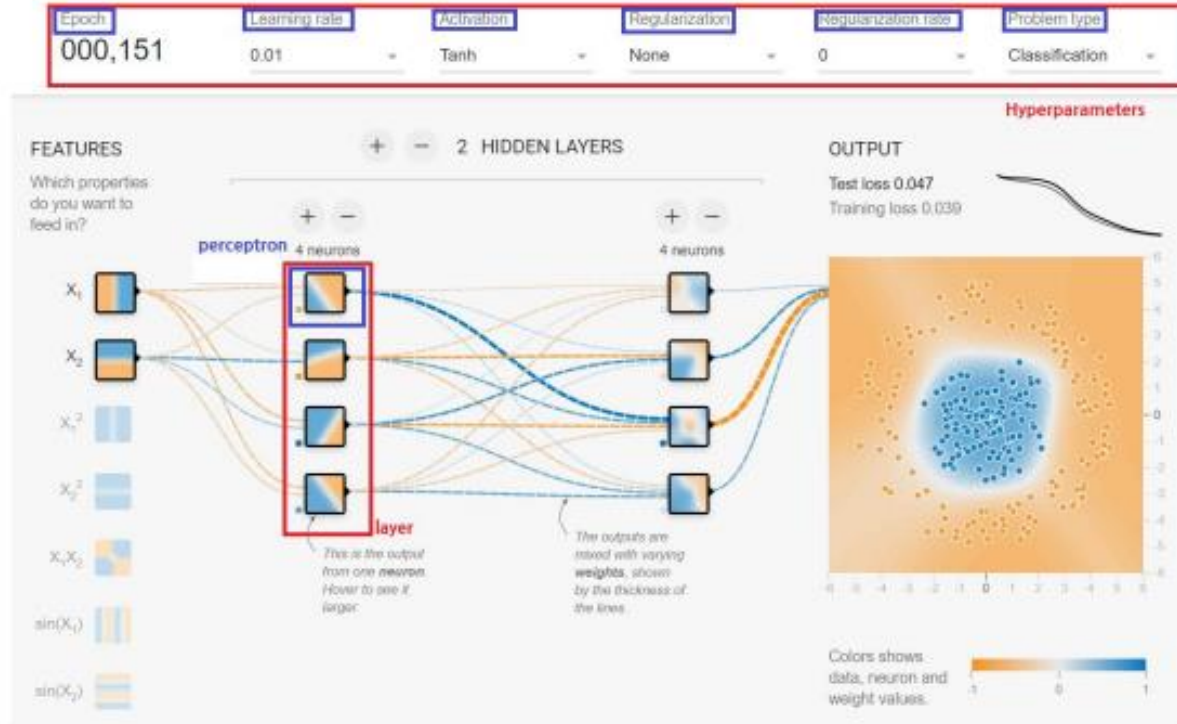
- RNN (Recurrent Neural Networks)



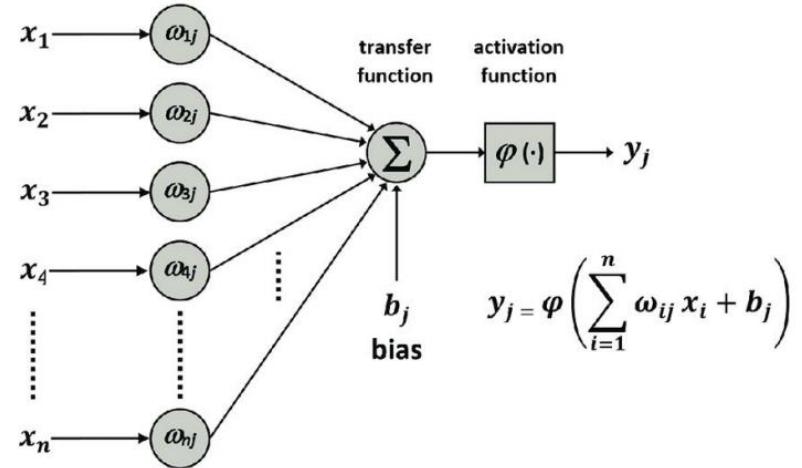
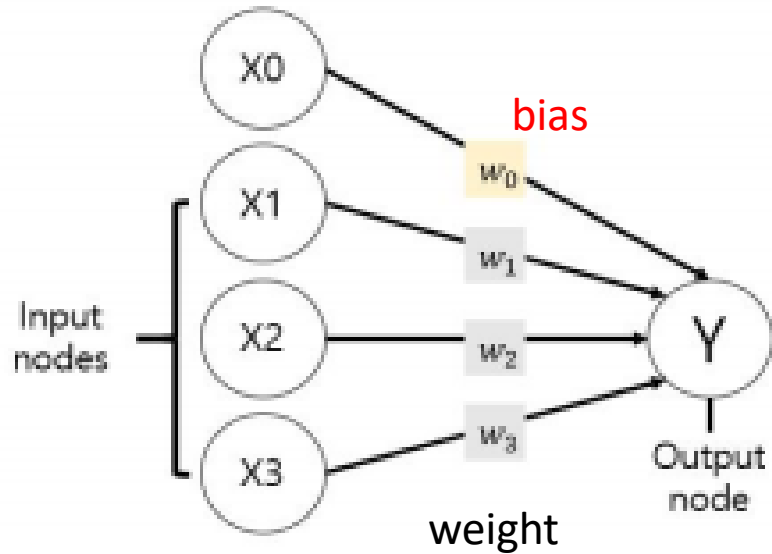
- GAN (Generative Adversarial Networks)



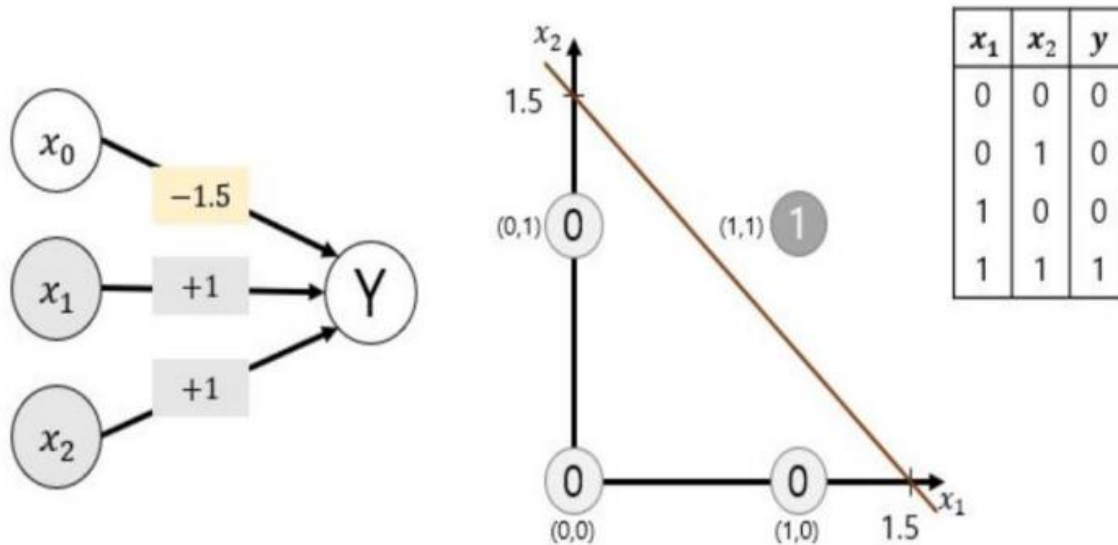
# Neural Network



# Perceptron



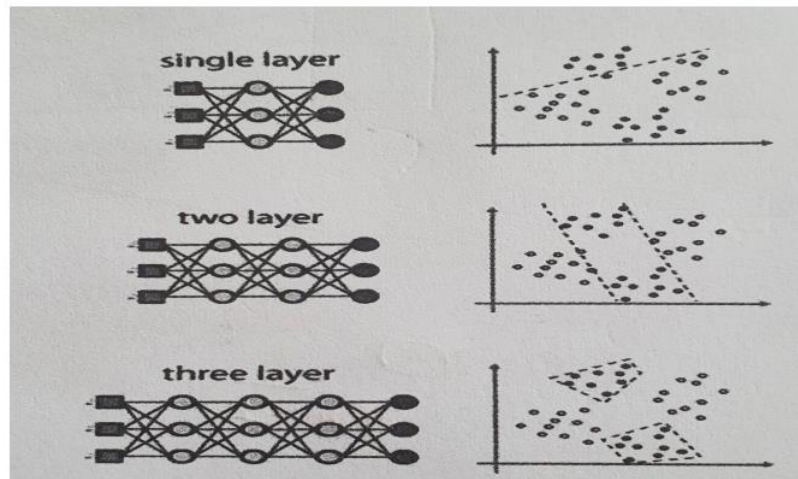
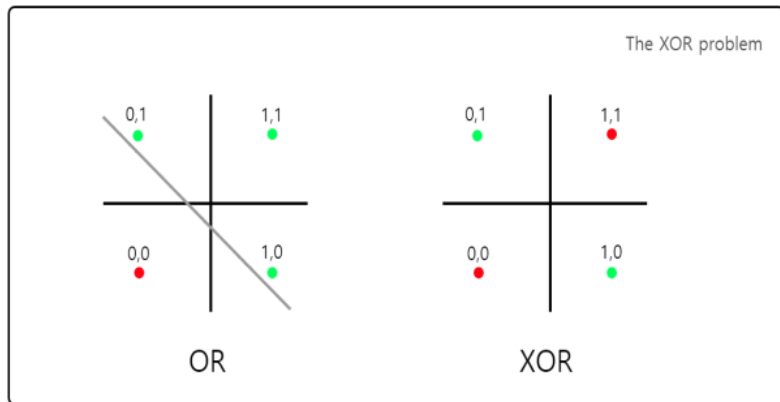
# Perceptron



**경사 하강법(Gradient Descent):**

$y$ 의 예측값과 실제  $y$ 값의 차이를 최소화하는 모델

# Multilayer Perceptron (MLP)

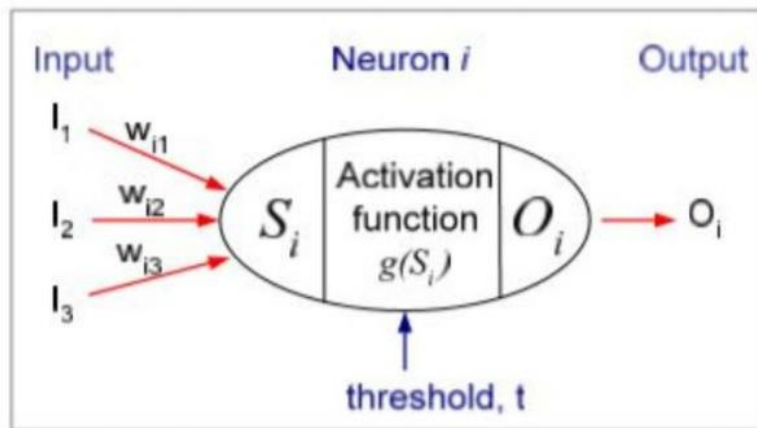


- XOR의 경우 어떤 방법을 써도 기준선 한 개로 Classification을 수행 불가  
→ 여러 개의 **decision boundary** 필요!
- **MLP**: Input layer와 Output layer 사이에 여러 개의 Hidden Layer를 추가한 Perceptron 구조로, hidden layer의 층 수 = decision boundary의 개수



# Activation Function (활성화 함수)

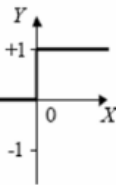
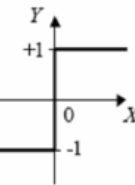
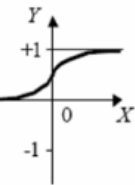
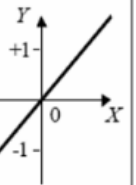
---



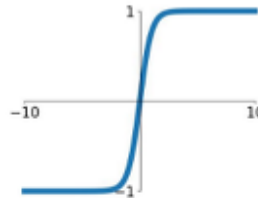
- Activation function을 적용한 한 perceptron

Activation function → 선형 boundary를 flexible 하게 해준다.

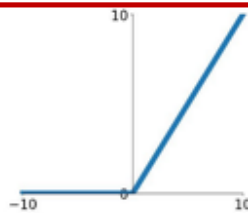
# Activation Function (활성화 함수)

Step function	Sign function	Sigmoid function	Linear function
			
$y^{step} = \begin{cases} 1, & \text{if } X \geq 0 \\ 0, & \text{if } X < 0 \end{cases}$	$y^{sign} = \begin{cases} +1, & \text{if } X \geq 0 \\ -1, & \text{if } X < 0 \end{cases}$	$y^{sigmoid} = \frac{1}{1+e^{-X}}$	$y^{linear} = X$

**tanh**  
 $\tanh(x)$



**ReLU**  
 $\max(0, x)$



- 음수 Input에서는 0, 양수에서는 Linear한 구조
- 기울기가 0 또는 1 → Gradient Vanishing 문제 X
- Non-Linear Function → Layer를 깊게 쌓을 수 있음
- 간단한 수식 → 빠른 연산능력

# Training Neural Network

## 28x28 픽셀 이미지 dataset을 학습하는 예제

# 0. 사용할 패키지 불러오기

```
from keras.utils import np_utils
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Activation
```

# 1. 데이터셋 생성하기

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(60000, 784).astype('float32') / 255.0
x_test = x_test.reshape(10000, 784).astype('float32') / 255.0
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
```

# 2. 모델 구성하기

```
model = Sequential()
model.add(Dense(units=64, input_dim=28*28, activation='relu'))
model.add(Dense(units=10, activation='softmax'))
```

# 3. 모델 학습과정 설정하기

```
model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
```

# 4. 모델 학습시키기

```
hist = model.fit(x_train, y_train, epochs=5, batch_size=32)
```

1. trainset, test set 생성

2. 딥러닝 모델 구성  
활성함수로 Relu와  
Softmax 사용

3. Loss function과  
optimizer 정의

4. 학습 진행

# Training Neural Network

---

Keras의 fit() method에는 원래 무려 19개의 parameter가 필요하다.  
그 중 일반적으로 가장 많이 조정하는 대표적인 parameter들을 살펴보자.

---

```
model.fit(x, y, batch_size=32, epochs=10)
```

- x = 입력 데이터
  - y = 라벨 값
  - batch\_size = 몇 개의 샘플로 가중치를 갱신할 것인지 설정
  - epochs = 학습 반복 횟수
-

# Training Neural Network

---

```
hist = model.fit(x_train, y_train, epochs = 5, batch_size = 32)
```

- **x**: Input data. It could be:
  - A Numpy array (or array-like), or a list of arrays (in case the model has multiple inputs).
  - A dict mapping input names to the corresponding array/tensors, if the model has named inputs.
  - A generator or `keras.utils.Sequence` returning `(inputs, targets)` Or `(inputs, targets, sample weights)` .
  - None (default) if feeding from framework-native tensors (e.g. TensorFlow data tensors).
- **y**: Target data. Like the input data `x`, it could be either Numpy array(s), framework-native tensor(s), list of Numpy arrays (if the model has multiple outputs) or None (default) if feeding from framework-native tensors (e.g. TensorFlow data tensors). If output layers in the model are named, you can also pass a dictionary mapping output names to Numpy arrays. If `x` is a generator, or `keras.utils.Sequence` instance, `y` should not be specified (since targets will be obtained from `x`).
- **batch\_size**: Integer or `None` . Number of samples per gradient update. If unspecified, `batch_size` will default to 32. Do not specify the `batch_size` if your data is in the form of symbolic tensors, generators, or `Sequence` instances (since they generate batches).
- **epochs**: Integer. Number of epochs to train the model. An epoch is an iteration over the entire `x` and `y` data provided. Note that in conjunction with `initial_epoch` , `epochs` is to be understood as "final epoch". The model is not trained for a number of iterations given by `epochs` , but merely until the epoch of index `epochs` is reached.

# Training Neural Network

---

쉽게 이해하기 위해서 우리가 모의고사 문제를 푸는 것에 비유를 해보자.

```
hist = model.fit(x_train, y_train, epochs = 5, batch_size = 32)
```

x\_train : 100문항의 문제들

y\_train : 100문항의 정답들

epochs : 주어진 모의고사를 반복 풀이하는 횟수

batch\_size : 채점 사이클(몇 문항마다 채점을 할까?)

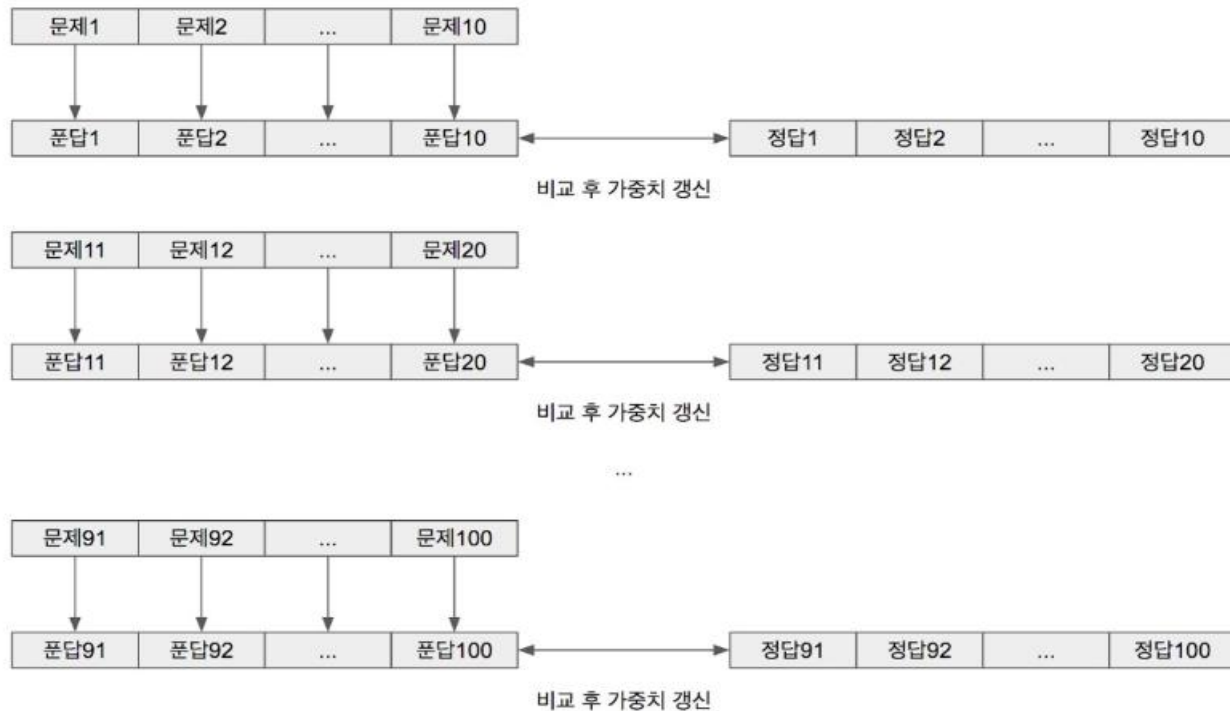
문제지

문제1	문제2	문제3	문제4	문제5	문제6	...	문제100
-----	-----	-----	-----	-----	-----	-----	-------

해답지

정답1	정답2	정답3	정답4	정답5	정답6	...	정답100
-----	-----	-----	-----	-----	-----	-----	-------

# Training Neural Network



일반적으로 batch size를 줄이면 가중치 갱신을 자주 할 수 있 으니까 학습 성능이 더 좋다?



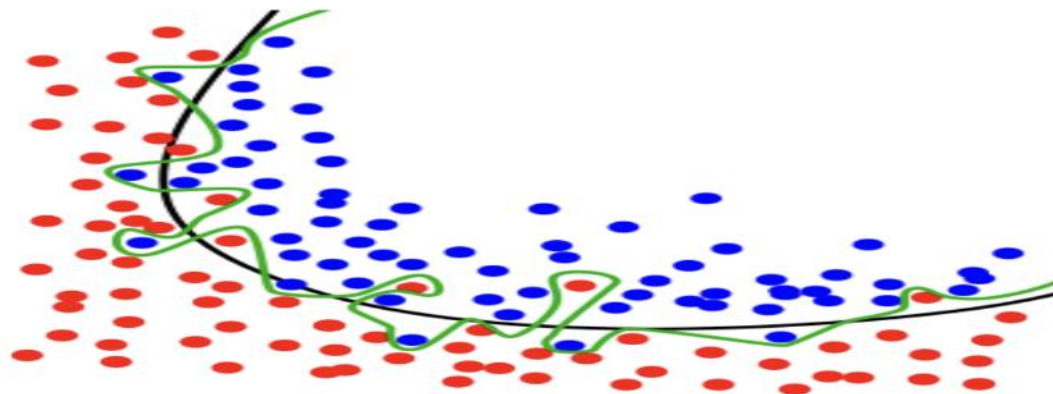
Gradient Vanishing Problem

# Problems In Training - Overfitting

---

- Overfitting

Layer 수가 많아지고, train epoch 수가 많아진다고 해서 무조건적으로 최종 학습 성능이 향상되는 것은 아니다. -> 결국 train data만 잘 학습하는 것일 뿐  
(모의고사 잘 치면 뭐하나 수능을 잘 쳐야지)



(출처 : <https://ko.wikipedia.org/wiki/%EA%B3%BC%EC%A0%81%ED%95%A9>)



# Problems In Training - Overfitting

- Solution of Overfitting, Dropout (2012년에 발표된 일종의 regularization)



몇 개의 neuron들을 일부러 제거하는 것

1. 제거되는 neuron들은 어떻게 선정?

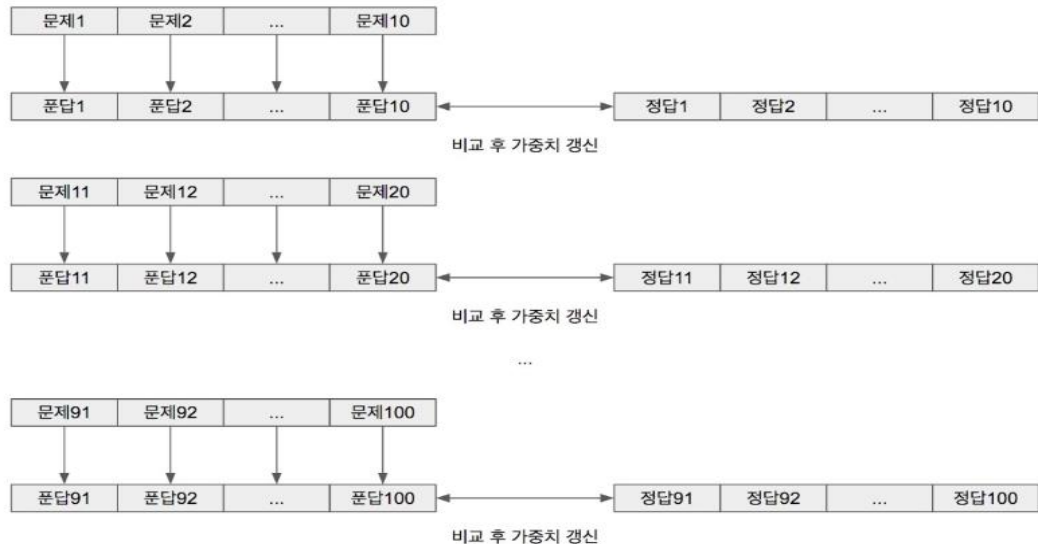
-> random으로!

2. 효과는?

-> train set에 대한 설명력은 떨어지지만  
전반적인 학습 성능은 향상됨

# Problems In Training – Gradient Vanishing

- Gradient Vanishing



Batch size를 줄일수록, 자주  
backpropagation을 수행함

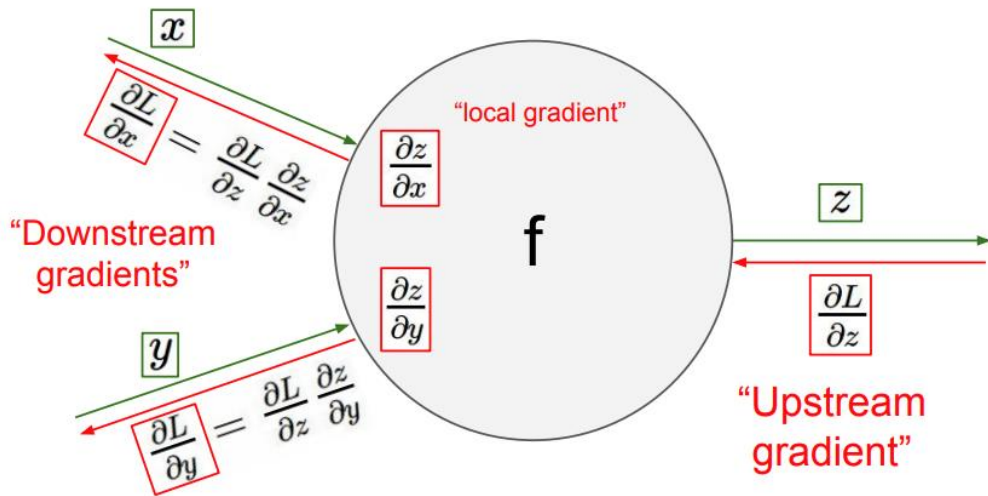


Gradient가 0으로 converge  
하는 경우가 발생하고, 가  
중치 update가 더 이상 수  
행되지 못함(특히 sigmoid..)

# Problems In Training – Gradient Vanishing

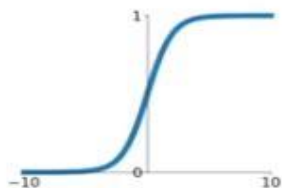
- Backpropagation

chain rule (연쇄 법칙)으로 gradient를 계산하는 방법 -> 꼬리에 꼬리를 문다고 생각하면 됨.



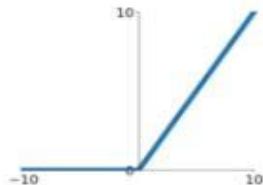
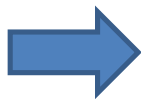
즉, Layer가 깊어지고,  
backpropagation 연산이 많아지  
면, 활성화 함수의 기울기가 0으  
로 converge

# Problems In Training – Gradient Vanishing



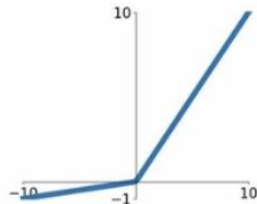
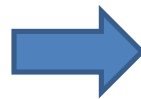
Sigmoid

음, 양의 극한으로  
갈수록 기울기가  
0으로 수렴 ->  
gradient vanishing  
심각  
(얘는 제발 버리자)



ReLU  
(Rectified Linear Unit)

양수일 때 기울기  
가 항상 '1' ->  
그런데 음수일 땐  
기울기가 항상 0  
인데 어떡할 거임?



Leaky ReLU

또 그럴 줄 알고  
음수일 때 기울기  
를 살려 줬음. 한  
번 써보자.  
하지만, 실제로 ReLU로  
충분히 해결할 수 있고,  
널리 쓰인다.

# Problems In Training – Time Complexity

---

- Neural Networks에 대하여 ML을 통해 최적의 bias값과 가중치 값들을 찾아야 함.



- 그렇다면 당연히 COST FUNCTION 최소화 작업 들어가야 함.

하지만, 기존의 Gradient Descent와 같은 일반적인 방법을 쓰면 DL의 어마 무시한 연산량을 감당하지 못할 것임

# Problems In Training – Time Complexity

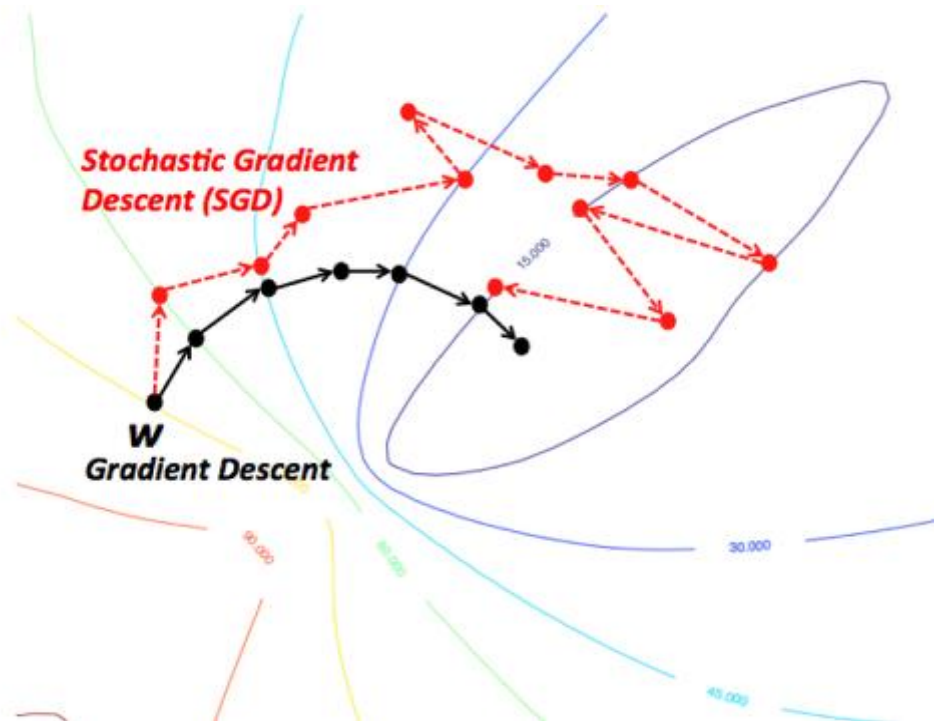
---

- 만약 가중치 update를 한 번할 때마다, 모든 데이터를 다 훑어본다면?
- > 데이터의 양이 방대해질 수록, 가중치 update를 위한 연산량이 매우 증가



- 전체 데이터를 다 훑지 말고, 일부(mini batch)만 훑고 update!
- > 최적의 해를 찾아가는 데에 기존의 Gradient Descent보다 속도는 빠르지만,  
But, 수렴안정성이 낮고, Local Optima에 빠지는 문제가 발생!

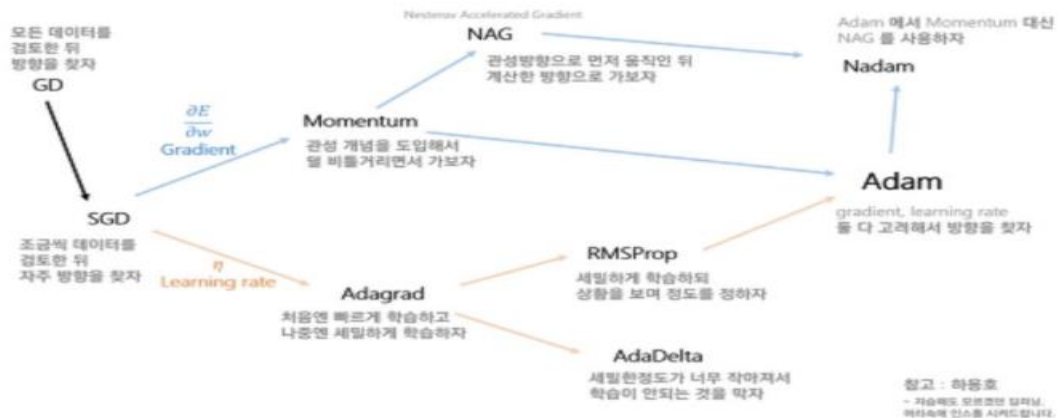
# Problems In Training – Time Complexity



S.G.D가 먼 길 돌아가는 것처럼 보이지만, 사실 속도는 더 빠름.

하지만 지그재그로 수렴하고 있어서 수렴안정성이 떨어지고, 중간에 딴길로 썰 수도 있음  
(Local Optima)

# Problems In Training – Time Complexity



- 요즘엔 더 좋은 Optimizer들이 많다

관성 중심:

Momentum, NAG

속도 중심:

Adagrad, RMSProp, AdaDelta

둘의 장점을 합침:

Adam, Nadam

$$w^+ = w - \boxed{\eta} * \boxed{\frac{\partial E}{\partial w}}$$

learning rate : 한번에 얼마나 학습할지

gradient : 어떤 방향으로 학습할지

(출처 : <https://gomguard.tistory.com/187>, Optimizer별 간략한 정리)



# Skills For Training

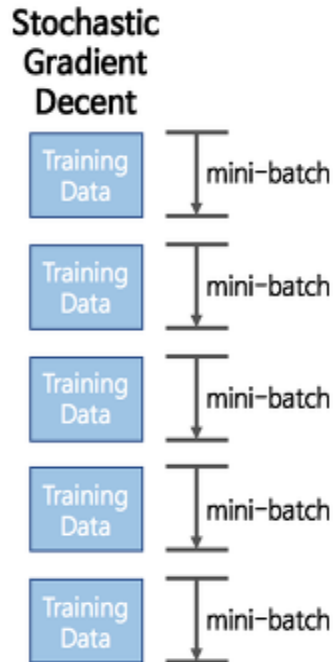
---

1. Mini-batch
2. Weight Initialization
3. Batch Normalization

# Skills For Training - Mini-batch

## - Mini-batch

앞서 소개한 s.G.D에 적용한 것 처럼,  
전체 데이터를 사용하지 않고,  
일부 sample에 대해서 가중치 update



# Skills For Training – Weight Initialization

---

## - Weight Initialization

1. 가중치의 초기값을 전부 0으로 하거나, 동일한 값으로 설정



Neuron의 가중치가 비대칭적이 되지 못하므로 마치 neuron 하나로 학습하는 것과 같아 짐

2. 작은 값을 가지는 random한 수들로 설정



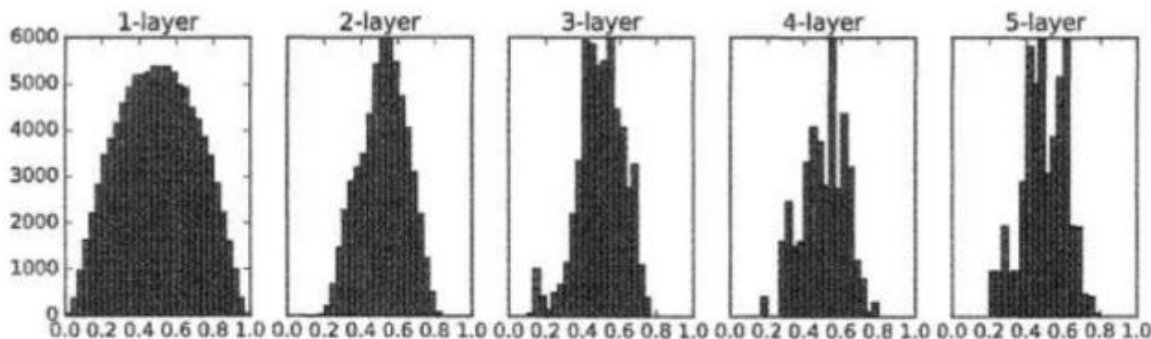
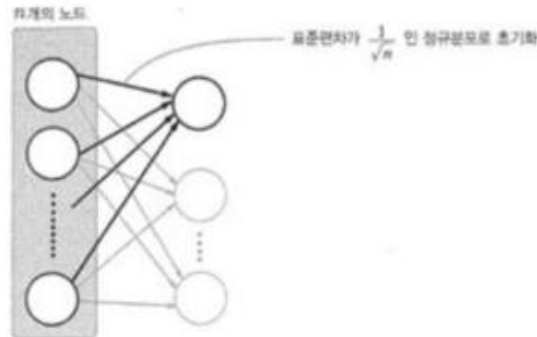
random하니까 동일한 가중치 값을 가지지 않음  
BUT, layer가 깊어질 수록 Gradient Vanishing 문제가 심화됨.

# Skills For Training – Weight Initialization

## - Xavier Initialization

출력값이 고른 분포를 보여주지만, 이것은 어디까지나 활성화함수로 tanh를 사용했을 경우이다.

-> ReLU를 사용하면, Layer가 깊어질 수록, 출력값이 0에 가까워진다



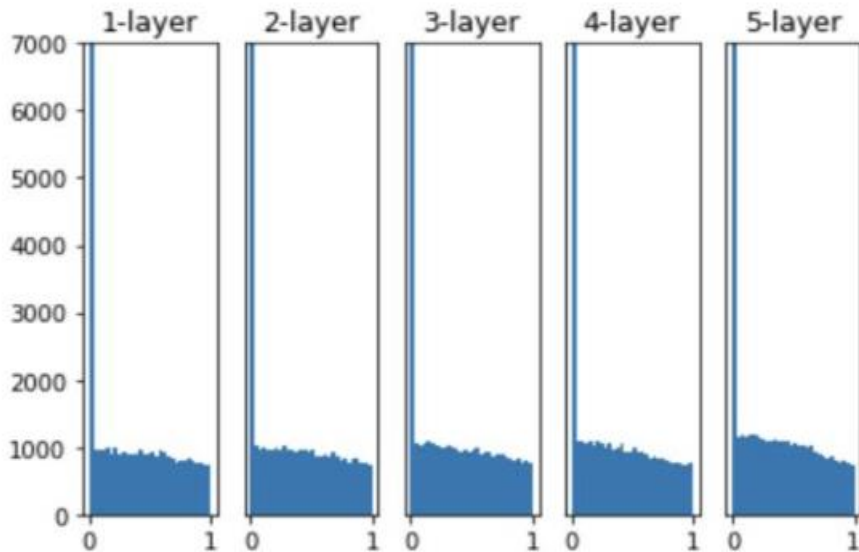
# Skills For Training – Weight Initialization

- He Initialization

Xavier 초기값에다가 각각  $\sqrt{2}$ 를 곱해주면 됨.

->ReLU 함수에 적용할 때,  
입력값이 음수면, 출력값이  
전부 0이므로, 가중치들을  
좀 더 넓게 분포시켜야 함.

ReLU함수를 적용했을 때,  
가중치들이 고르게 분포됨.

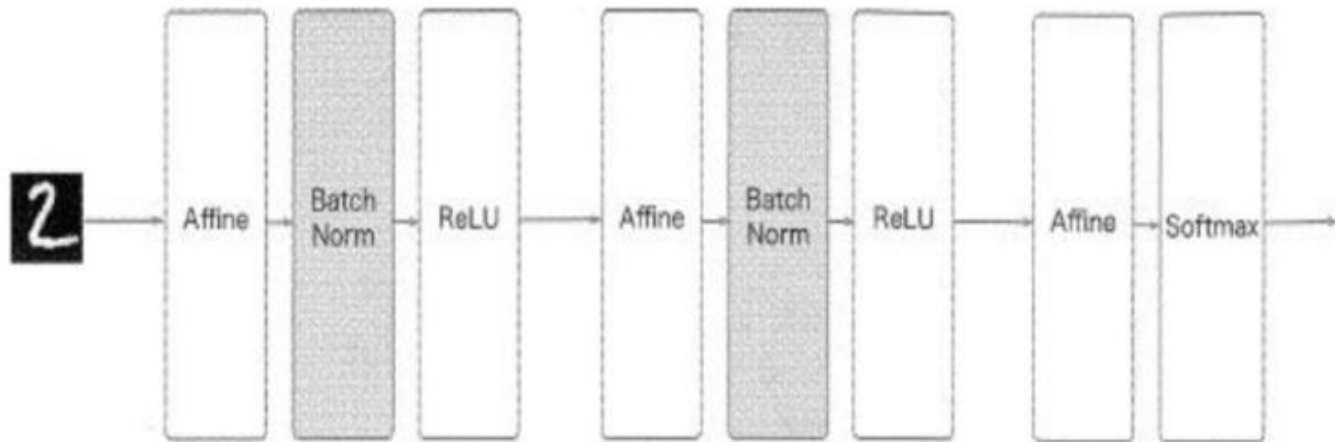


(출처 : <https://excelsior-cjh.tistory.com/177>)

# Skills For Training – Batch Normalization

- Batch Normalization

데이터의 분포를 정규화(배치 정규화 계층) -> 각 Layer에 활성화 값을 고르게 분포.



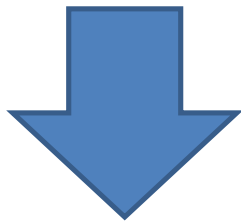
이름에 걸맞게 학습 시에 mini-batch단위로 Normalization 진행 -> 평균 = 0, 분산 = 1

# Skills For Training – Batch Normalization

---

- Batch Normalization

데이터의 분포를 정규화(배치 정규화 계층) -> 각 Layer에 활성화 값을 고르게 분포.



1. 빠른 학습 진행
2. 초기값 의존도가 낮아짐
3. Overfitting 억제

# Reference And Source

---

<https://excelsior-cjh.tistory.com/>

<https://seamless.tistory.com/38>

cs231n in Stanford university education

[https://tykimos.github.io/2017/01/27/Keras\\_Talk/](https://tykimos.github.io/2017/01/27/Keras_Talk/)

Keras official document



Q & A