강의계획표

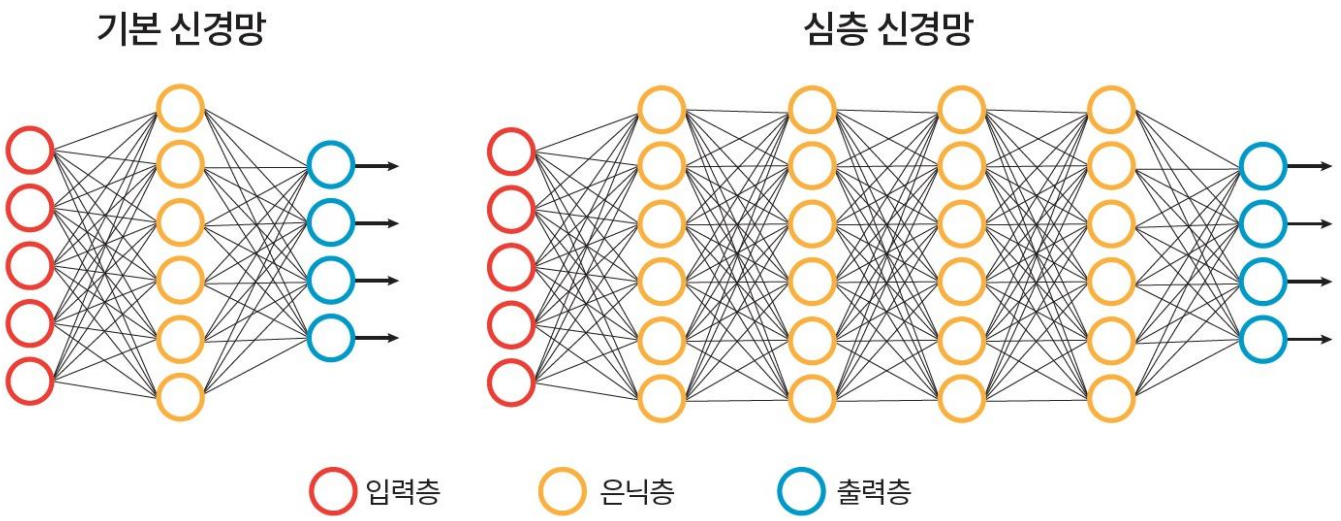
주	해당 장	주제
1	1장 2장, 3장	머신러닝이란
2		머신러닝을 위한 기초지식, 구현을 위한 도구
3	4장	선형 회귀로 이해하는 지도학습
4	5장	분류와 군집화로 이해하는 지도 학습과 비지도 학습
5	6장	다양한 머신러닝 기법들 - 다항 회귀, Logistic Regression - 정보이론, 결정트리 - SVM, Ensemble
6		
7		
8		중간고사 (04-20)
9	7장	인공 신경망 기초 - 문제와 돌파구
10	8장	고급 인공 신경망 구현
11	9장	신경망 부흥의 시작, 합성곱 신경망
12	10장	순환 신경망
13	11장	차원축소와 매니폴드 학습
14	12장	오토인코더와 잠재표현 학습
15		보강주
16		기말고사

8장 고급 인공 신경망 구현
10 주차

Deep Neural Network (1)

❖ **DNN**

- MLP(다층 퍼셉트론)에서 은닉층의 개수를 증가
  - 기술적 문제
  - 컴퓨터 성능의 문제
- 컴퓨터 시각, 음성 인식, 자연어 처리, 소셜 네트워크 필터링, 기계 번역 등에서 뛰어난 성능 (예, ImageNet의 image 인식대회)

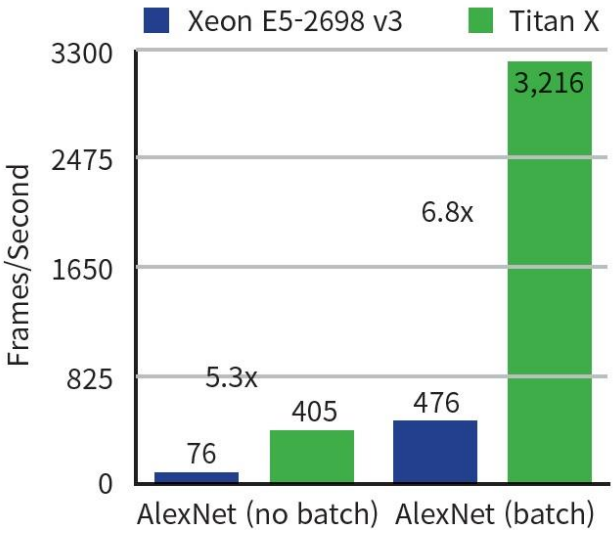


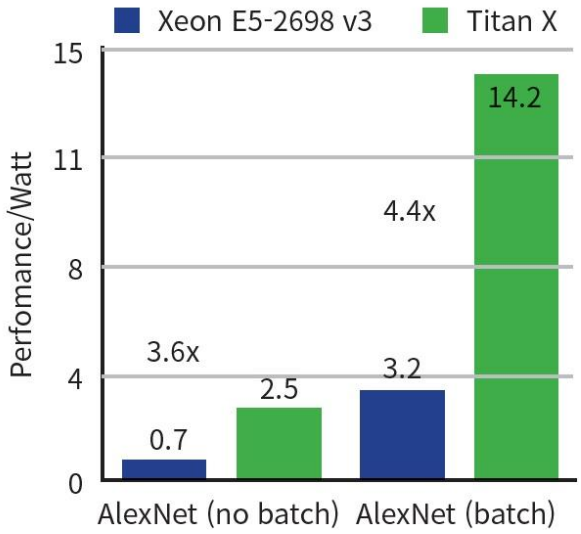
Deep Neural Network (2)

❖ **DNN의 성능개선 원인**

- MLP의 문제점 개선
  - 그래디언트 소멸(vanishing gradien) 문제: 활성화 함수
  - 초기 가중치
  - 손실함수
  - 과잉 적합(over fitting)
- Big Data
- GPU 성능 개선

CPU와 GPU의 성능비교

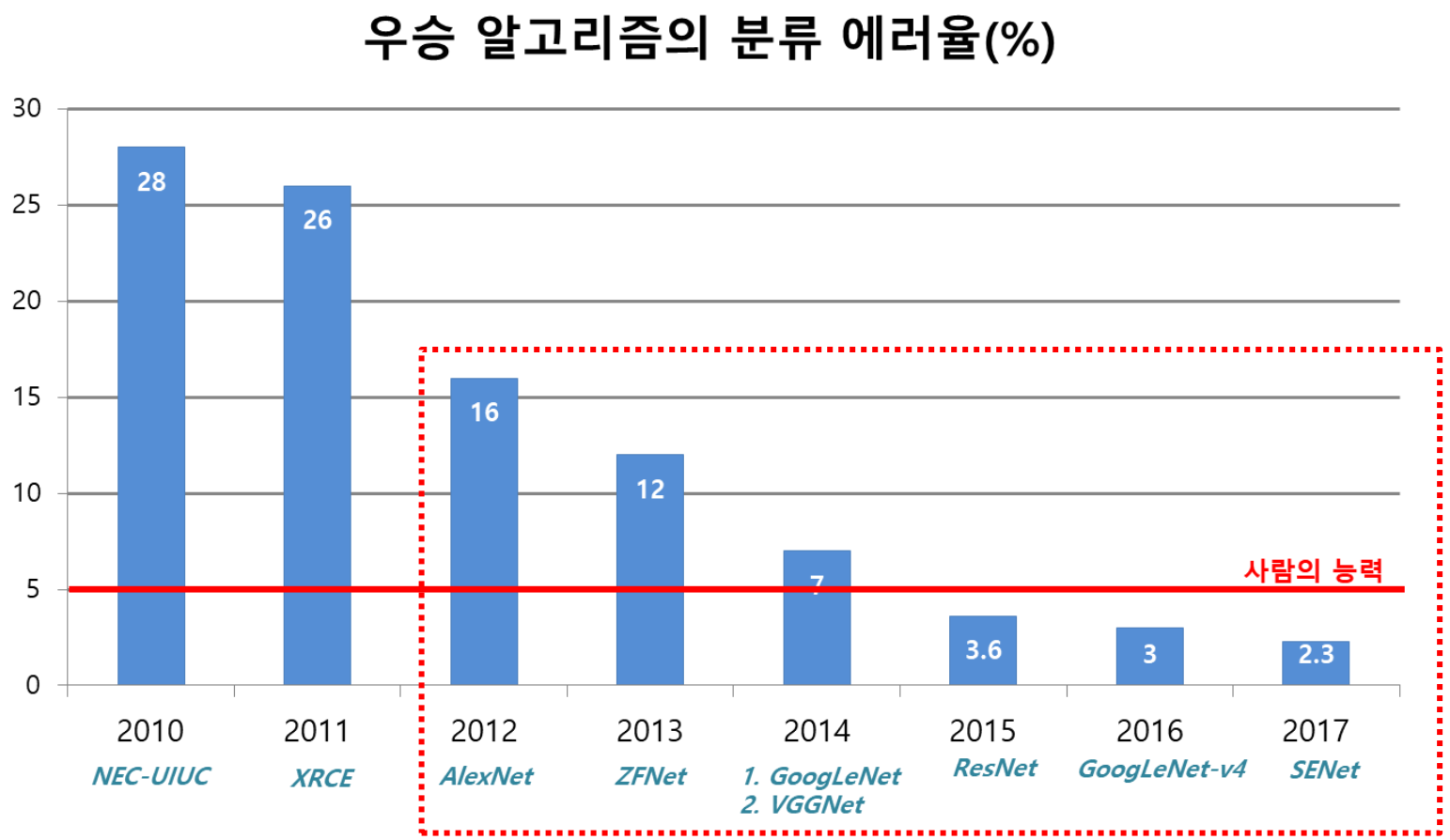




Deep Neural Network (3)

❖ **DNN 성능**

- 컴퓨터 시각, 음성 인식, 자연어 처리, 소셜 네트워크 필터링, 기계 번역 등에서 뛰어난 성능 (예, ImageNet의 image 인식대회)



Deep Neural Network (4)

❖ 은닉층의 역할

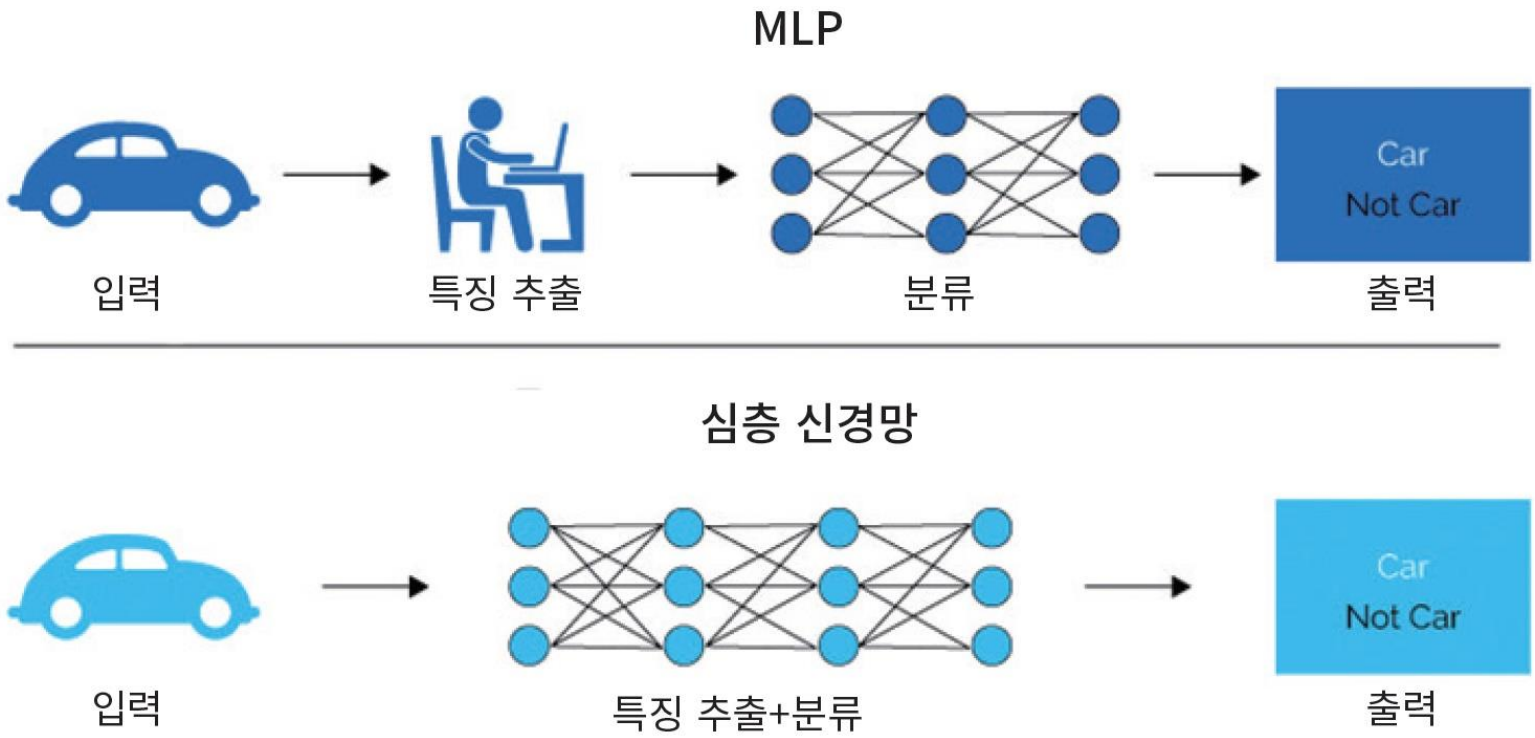


그림 14–3 MLP와 심층 신경망의 비교

Deep Neural Network (5)

❖ 은닉층의 역할

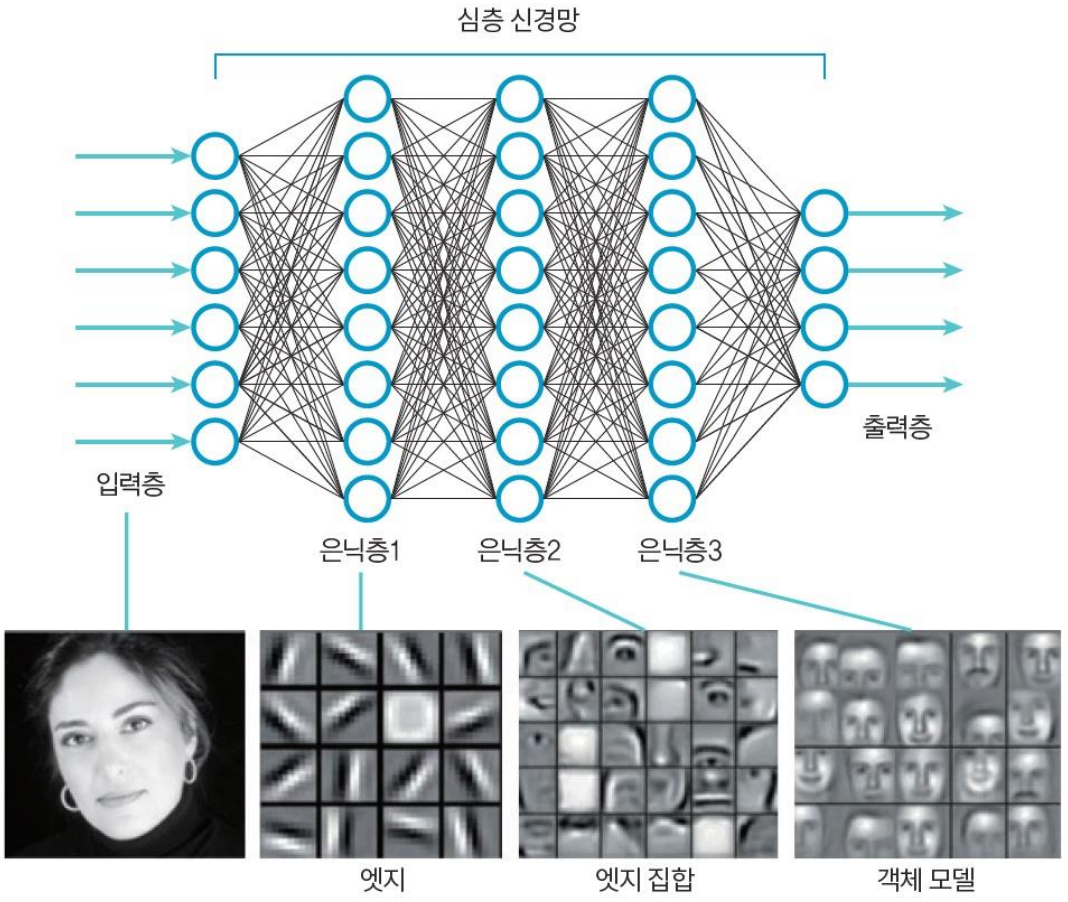
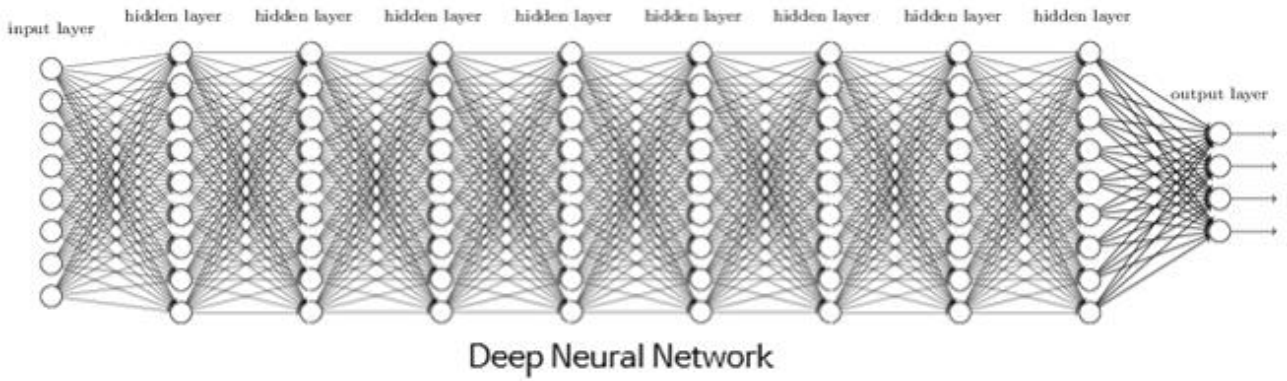
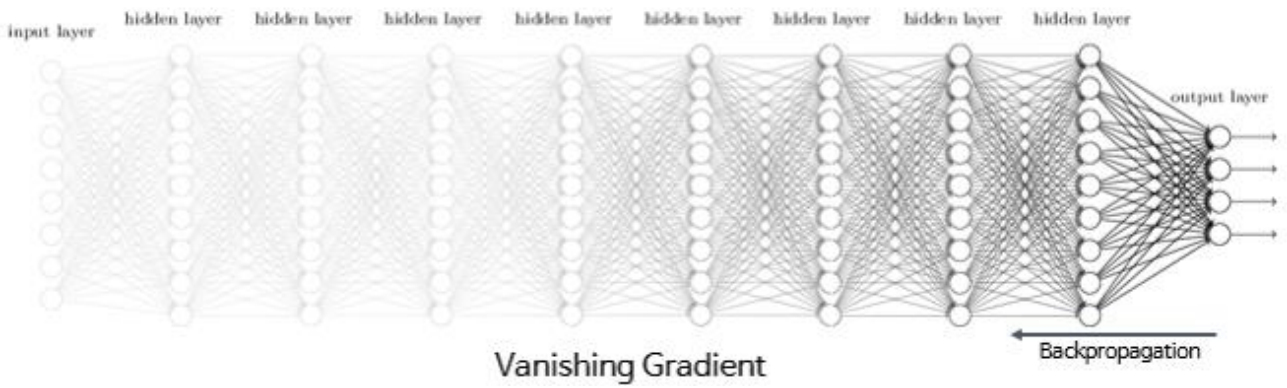


그림 14–2 은닉층의 역할(*출처: 위키미디어)

MLP의 문제점 (1)

- **Vanishing gradient 현상**
  - 오차가 아래 단계(이전 단계)로 역전파될수록 점점 오차가 소실됨
- **Exploding gradient 현상**
  - 기울기가 아래 단계로 내려가면서 급격히 커져서 아래 단계(이전 단계)의 연결강도가 지나치게 커지게 됨

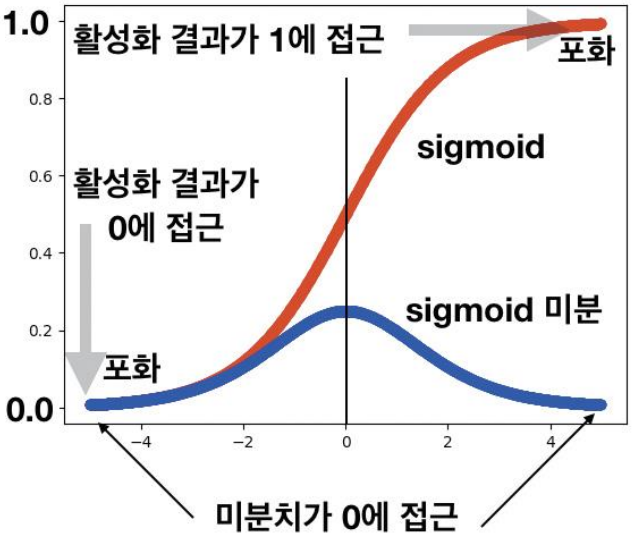


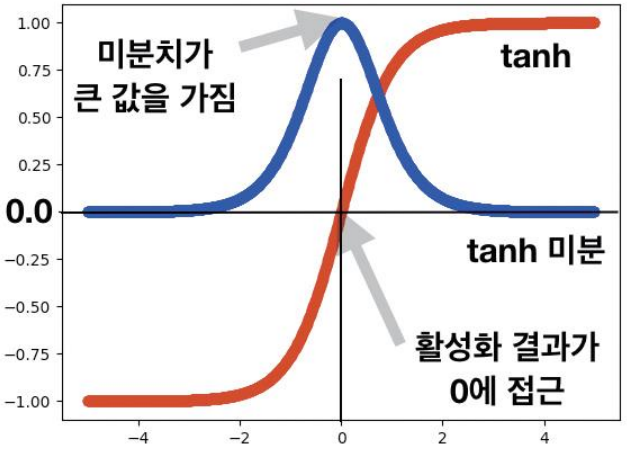


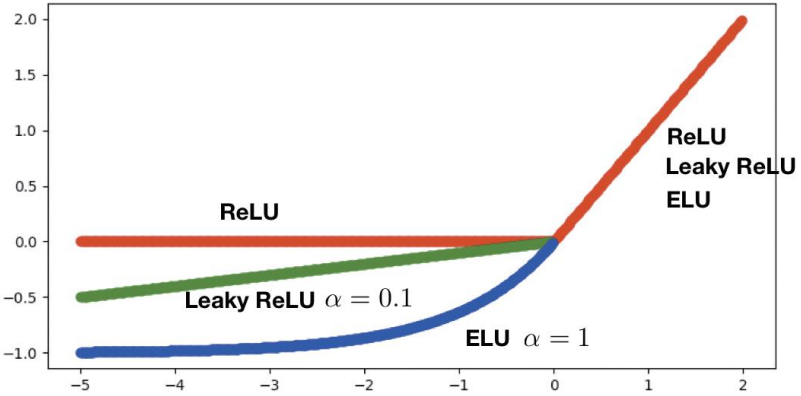
MLP의 문제점 (2)

❖ 활성화 함수

- Sigmoid 활성화 함수의 값이 0이나 1에 접근하면 미분치는 0에 가까워짐.
- 오차 역전파는 활성화 함수의 미분치를 곱해 오차를 아래로 전달하는 것이므로 sigmoid 활성화 함수가 포화 상태에 가까워지면 아래 단계로 오차를 전파하지 못함
- 해결책
  - tanh[hyperbolic tangent] 대칭 함수 사용(Glorot, Bengio): 활성화 함수의 결과가 0인 지점에 접근해도 미분치는 큰 값을 유지
  - ReLU[Rectified Linear Unit] 변형





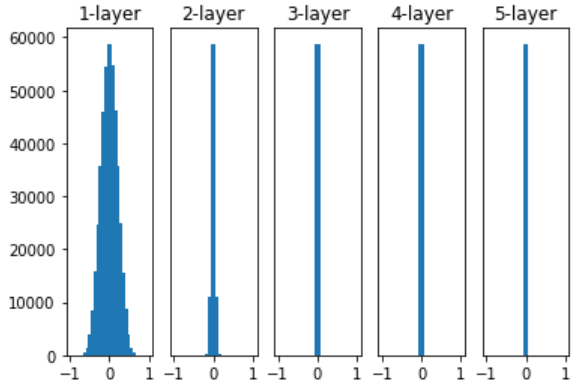


MLP의 문제점 (3)

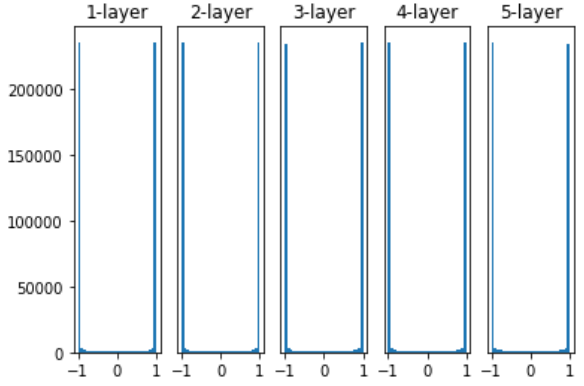
❖ 가중치 초기화의 영향

- 가중치를 동일하게 주면 모든 오차가 동일

  => 모든 가중치가 동일하게 변경
- 가중치가 대칭적이면 대칭적인 가중치와 관련한 노드는 동일한 일을 함
- 가중치의 초기값은 random하게 부여(표준편차 1, 평균 0인 균등분포)
  - 각 계층의 노드 수가 달라지면 입력에서 출력으로 가는 순전파 신호와 출력에서 입력으로 내려오는 오차 역전파 신호의 분산이 왜곡되어 전달
  - 활성화 함수: tanh



평균이 0, 표준편차가 0.01인 정규분포
(신경망의 깊이가 깊어질수록 문제가 발생)

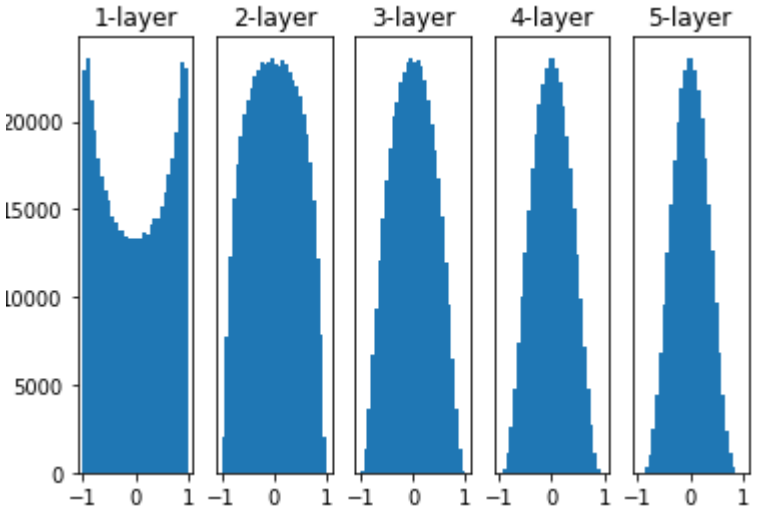


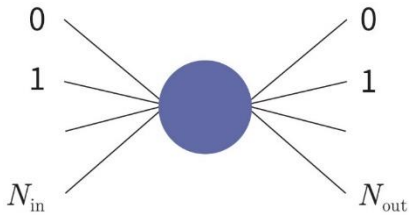
평균이 0, 표준편차가 1인 정규분포

MLP의 문제점 (4)

❖ **Xavier 초기화(Glorot 초기화)**

- 각 레이어의 출력에 대한 분산이 입력에 대한 분산이 동일(은닉층의 크기가 동일)
- 역전파에서 레이어를 통과하기 전과 후의 그래디언트 분산이 동일
- 평균이 0이고 표준편차 $\sigma = \sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$ 인 정규분포
- $r = \sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$ 일 때 $-r$과 $+r$ 사이의 균등분포
- 입력과 출력의 연결 개수가 비슷할 경우 $\sigma = 1/\sqrt{n_{\text{inputs}}}$ 또는 $r = \sqrt{3}/\sqrt{n_{\text{inputs}}}$ 사용
- Tanh 시용시의 각 layer의 출력 분포

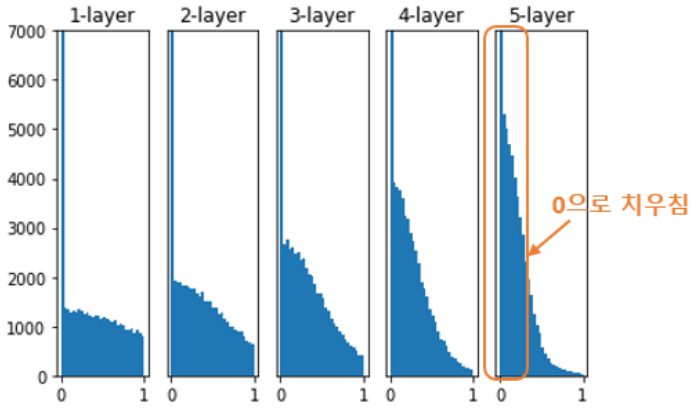




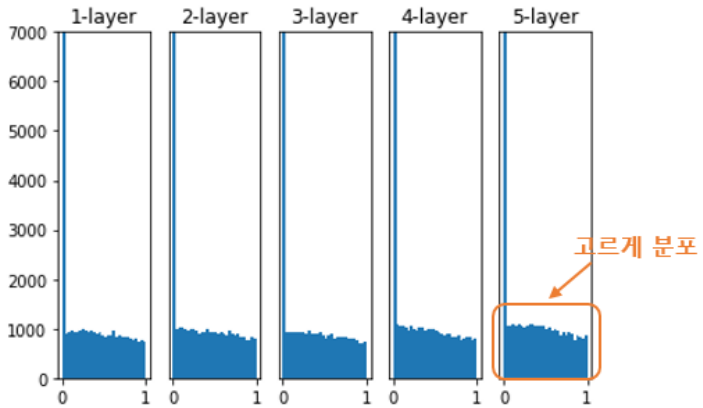
MLP의 문제점 (5)

❖ He 초기화

- Xavier 초기값은 ReLU 활성화 함수 사용시 각 layer의 출력 분포



<ReLU에서 Xavier 초기값의 문제>



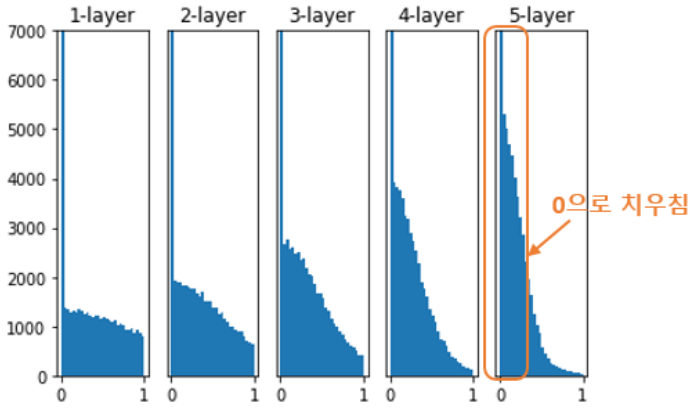
<ReLU에서 He 초기값 사용>

- 평균이 0이고 표준편차 $\sigma = \sqrt{2} \cdot \sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$ 인 정규분포
- $r = \sqrt{2} \cdot \sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$ 일 때 $-r$과 $+r$ 사이의 균등분포
- 입력의 연결 개수와 출력의 연결 개수가 비슷할 경우
  $\sigma = \sqrt{2}/\sqrt{n_{\text{inputs}}}$ 또는 $r = \sqrt{2} \cdot \sqrt{3}/\sqrt{n_{\text{inputs}}}$ 사용

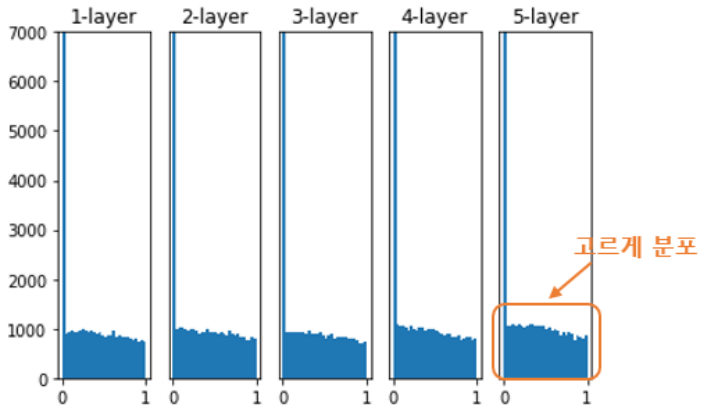
MLP의 문제점 (6)

❖ He 초기화

- Xavier 초기값은 ReLU 활성화 함수 사용시 각 layer의 출력 분포



<ReLU에서 Xavier 초기값의 문제>



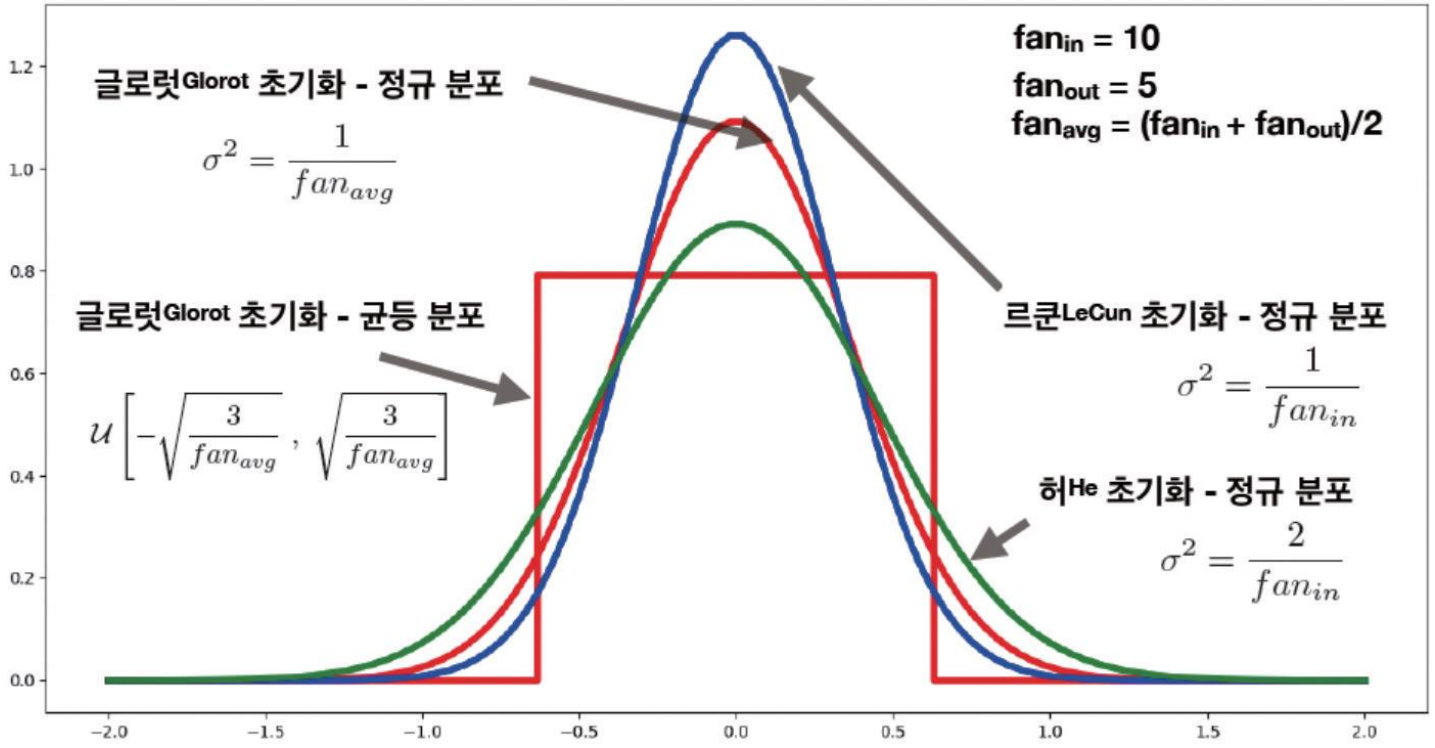
<ReLU에서 He 초기값 사용>

- 평균이 0이고 표준편차 $\sigma = \sqrt{2} \cdot \sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$ 인 정규분포
- $r = \sqrt{2} \cdot \sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$ 일 때 $-r$과 $+r$ 사이의 균등분포
- 입력의 연결 개수와 출력의 연결 개수가 비슷할 경우
  $\sigma = \sqrt{2}/\sqrt{n_{\text{inputs}}}$ 또는 $r = \sqrt{2} \cdot \sqrt{3}/\sqrt{n_{\text{inputs}}}$ 사용

MLP의 문제점 (7)

❖ 활성화 함수의 다양화

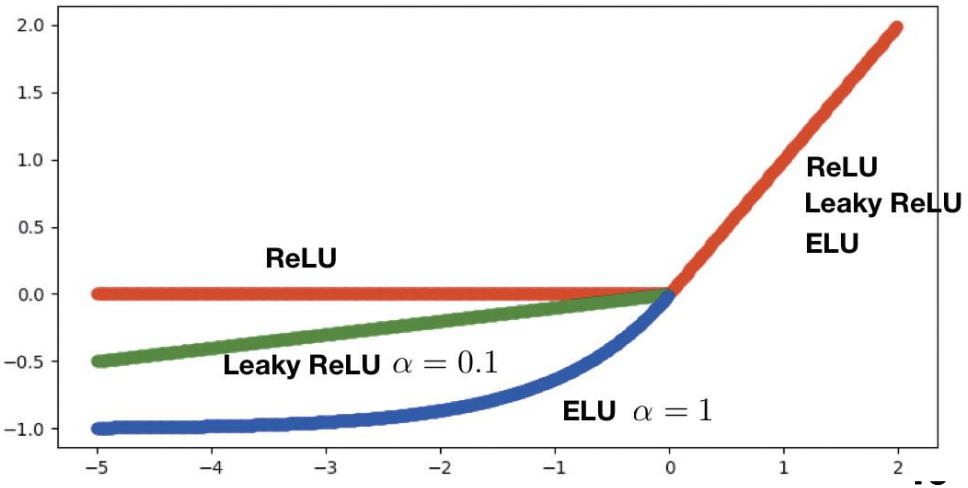
- 글로럿의 초기화 방법은 균등 분포로 제안되었는데, 가중치의 표준편차가 $\sigma = 1/\sqrt{fan_{avg}}$가 되도록 한 것 ($fan_{avg}$는 fan_{in}과 fan_{out}의 평균)



MLP의 문제점 (8)

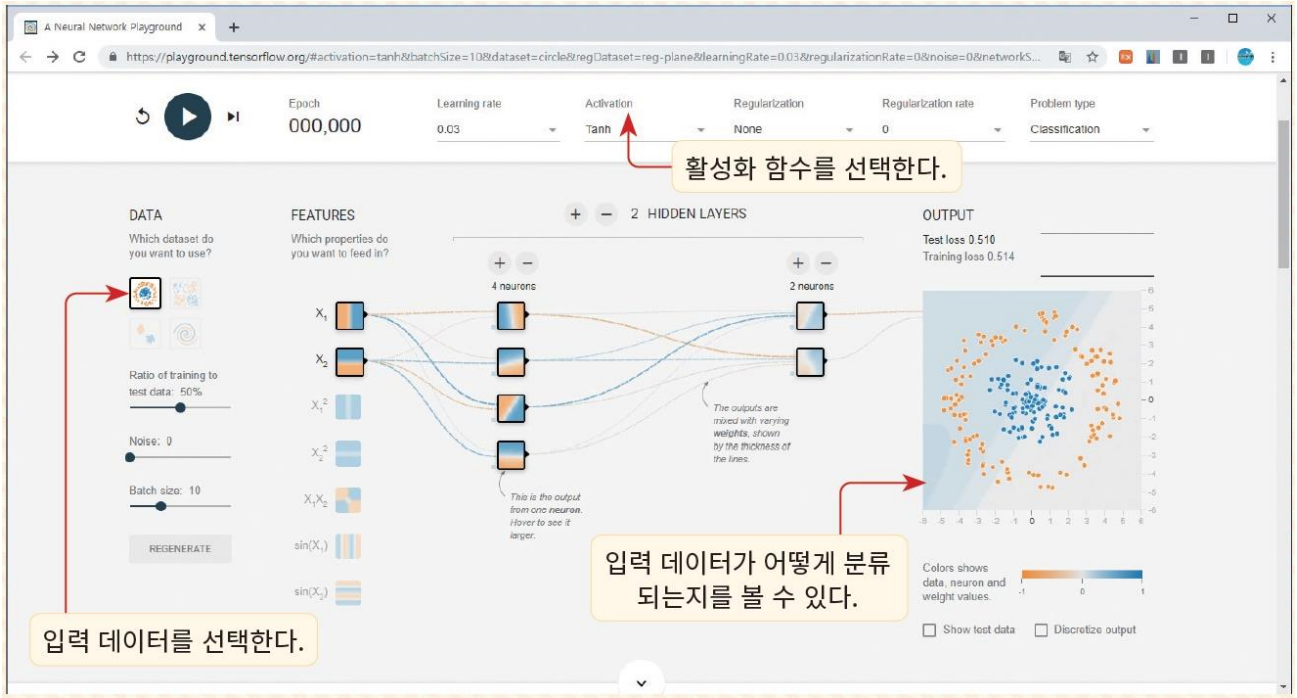
❖ 활성화 함수의 다양화

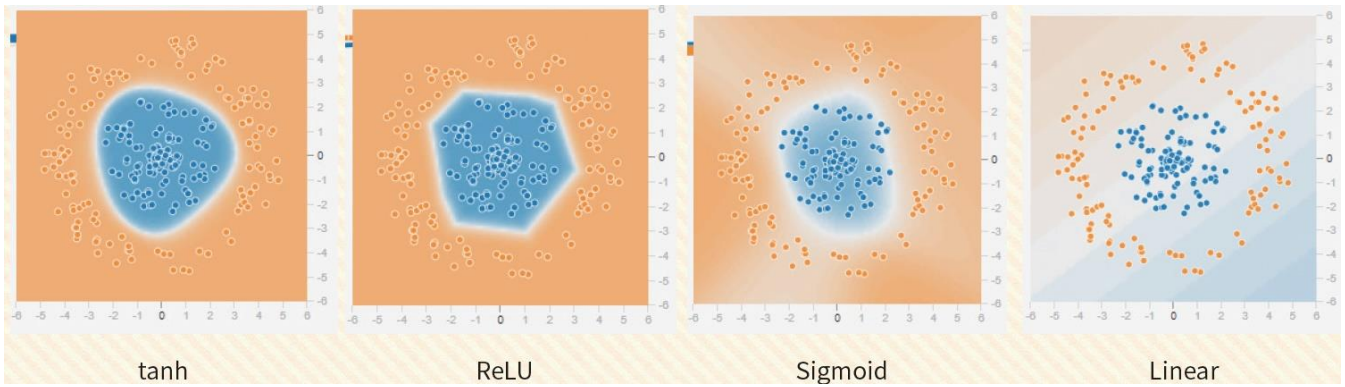
- ReLU $ReLU(x) = \max(0, x)$
  - 사용하는 노드가 0을 출력하게 되면, 해당 노드의 미분은 0 오차를 아래로 전달하지 못해 계속해서 0을 출력하는 노드로 남게 되는 것
- Leaky ReLU $LeakyReLU_{\alpha}(x) = \max(\alpha x, x),\ 0 < \alpha < 1$
  - 입력이 음수인 곳에서도 0을 출력하지 않고 1보다 작은 기울기로 약간의 신호를 내어 보냄
  - 이 구간의 미분은 α가 되고, 이 값은 0이 아니기 때문에 오차를 내려 보내어 연결강도 조정하는 일이 가능
- PReLU parametric ReLU : α를 파라미터화
- ELU $ELU(x) = \begin{cases} x & , x \geq 0 \\ \alpha(e^x - 1), & x < 0 \end{cases}$



MLP의 문제점 (9)

❖ **활성화 함수 실험** (https://playground.tensorflow.org)





Optimizer (1)

❖ **경사하강법에 의한 가중치 변경**

- 파라미터가 w일 때 오차를 측정하는 손실함수 $J(w)$를 정의하고, 파라미터에 대한 손실함수의 기울기에 따라 파라미터를 수정하는 방법

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

learning rate : 한번에 얼마나 학습할지

gradient : 어떤 방향으로 학습할지

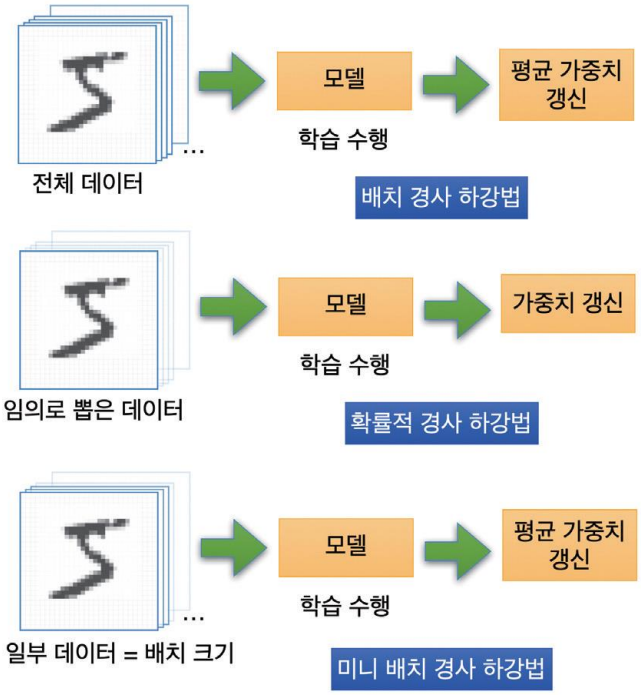
- 가중치 변경을 위한 학습데이터 셋 사용하는 방법
- 학습률 η 영향
- 기울기 결정방법

Optimizer (2)

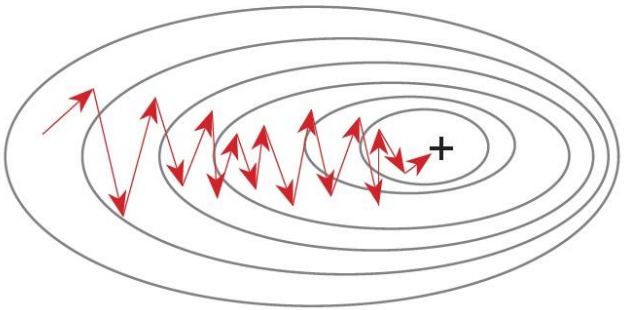
❖ **가중치 변경 방법(**학습데이터 셋 사용하는 방법)

- **Stochastic Gradient Descent(SGD)**
  - 하나의 샘플(랜덤)이 주어지면 오차를 계산하여서 바로 가중치를 변경
  - 잡음에 취약
  - 부분 극소점local minimum으로 잘못 수렴할 가능성
- **Batch Learning(GD)**
  - 모든 샘플을 모두 보여준 후에 개별 샘플의 그래디언트를 전부 더해서 이것을 바탕으로 모델의 가중치를 변경
  - 모델 update 되는데 시간이 많이 걸림
  - 전체 data가 memory에 들어가지 못할 수 있음
- **Mini Batch**
  - 훈련 데이터를 작은 배치들로 분리시켜서 하나의 배치가 끝날 때마다 가중치 update를 수행
  - 1 < size(매니 배치) < size(훈련 데이터)

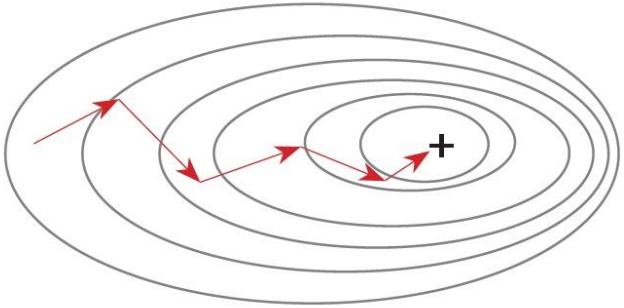
Optimizer (3)



확률적 경사 하강법(SGD)



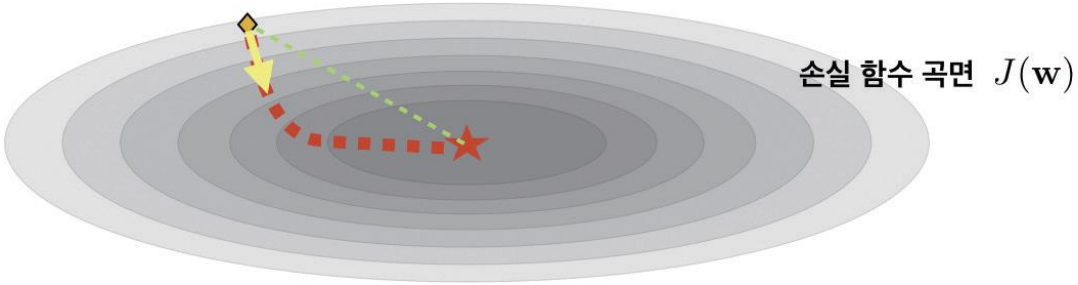
미니 배치 경사 하강법



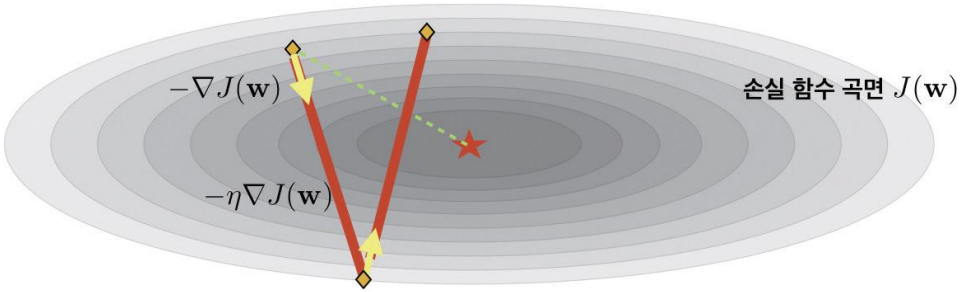
Optimizer (4)

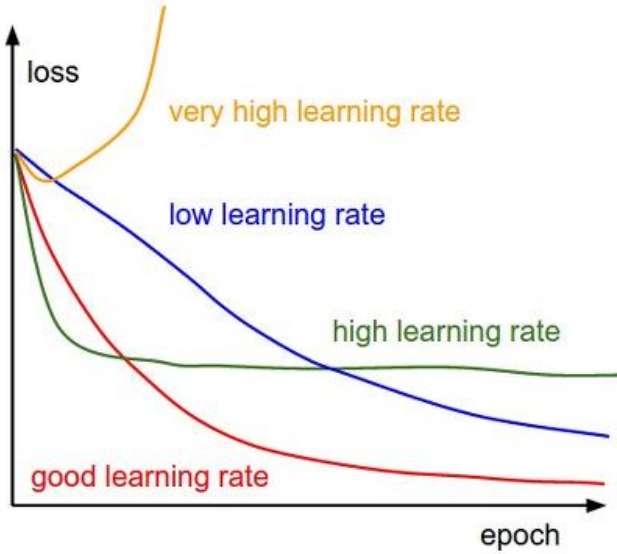
❖ 학습률 η 영향

- 적을 경우: 느림, 급한 경사를 먼저 내려간 뒤에 골을 따라 최적해로 이동.



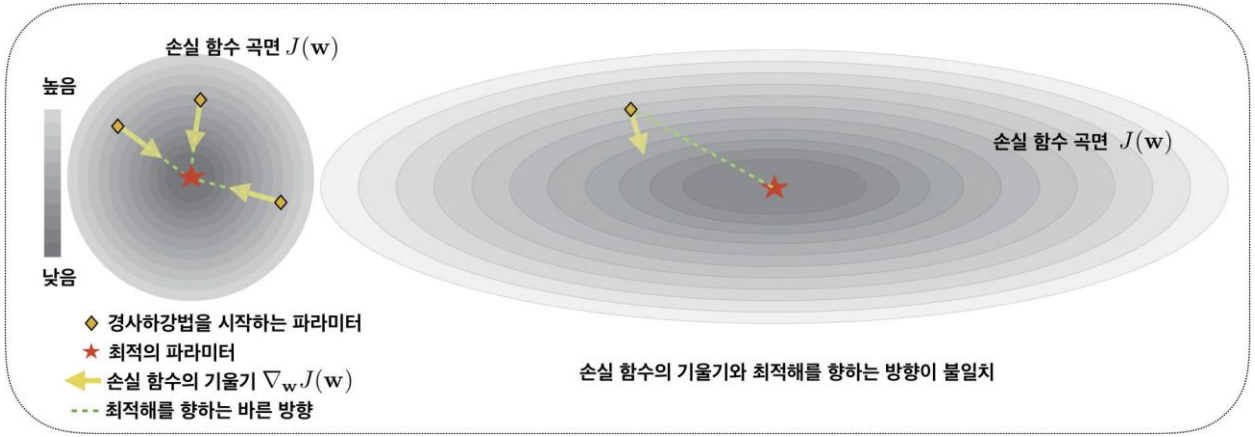
- 큰 경우: 골을 여러번 건너며 빠르게 수렴하지 못하거나, 심지어 발산해 버릴 수도 있음





Optimizer (5)

- 기울기 (descent gradient) 방향
  - 학습률이 너무 적으면 최적해까지 도달하는데 많은 기울기 계산
  - 곡면이 오른쪽과 같이 특정한 방향으로 늘어난 형태라고 생각하면 **기울기를 따라 가는 방법은 최적해를 향하는 가장 빠른 길을 따르지 못 함**



Optimizer (6)

❖ **Momentum**

- 기울기의 이동한 경로를 누적한 값
- 학습속도를 가속시킴
- 전역최소값을 찾는 데 도움

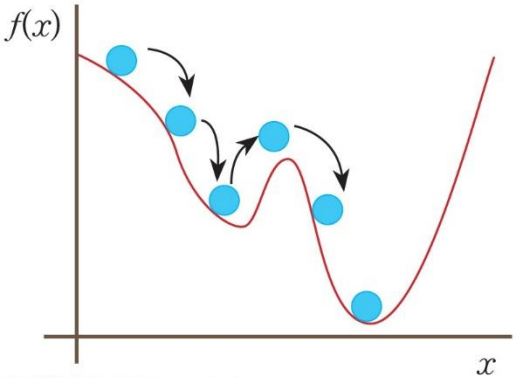
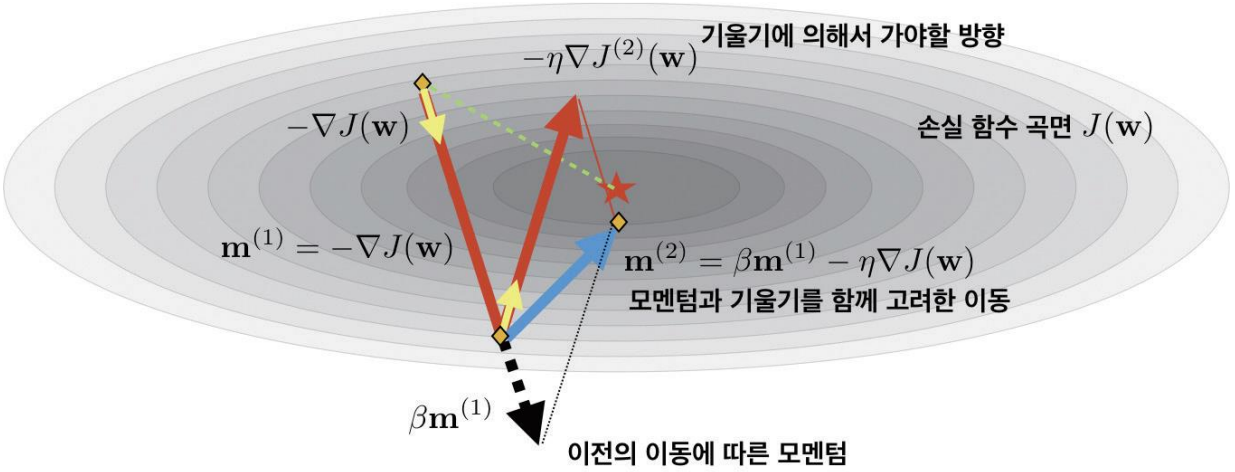


그림 14–16 모멘텀

$$\mathbf{m} \leftarrow \beta\mathbf{m} - \eta\nabla J(\mathbf{m})$$
$$\mathbf{w} \leftarrow \mathbf{w} + \mathbf{m}$$

$$w_{ij}(t+1) = w_{ij}(t) - \eta\frac{\partial E}{\partial w_{ij}} + momentum^{*} w_{ij}(t)$$



Optimizer (7)

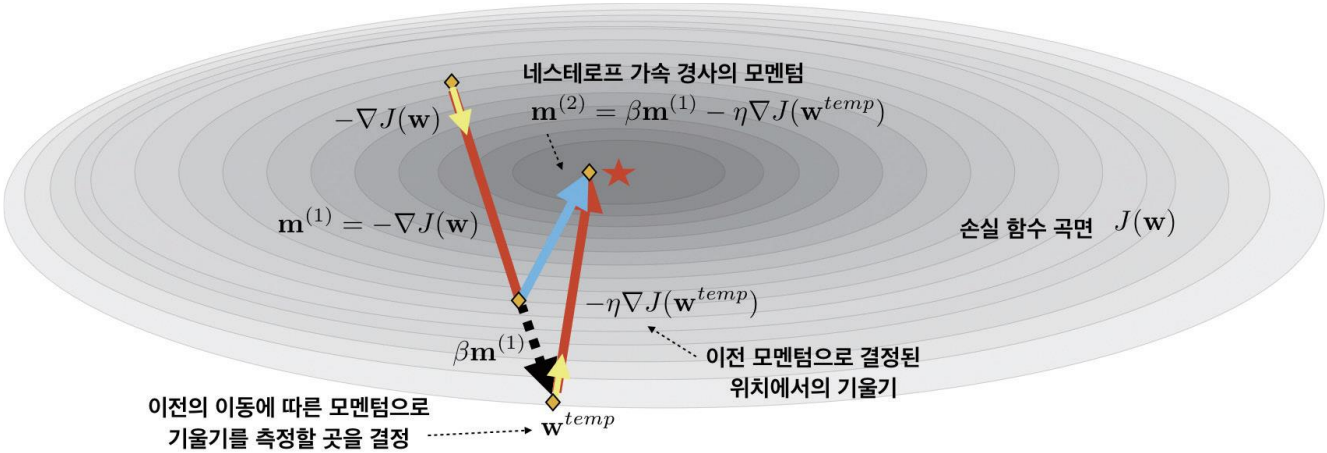
❖ 네스테로프 가속 경사Nesterov accelerated gradient

- 손실 함수 곡면의 기울기를 계산할 때 이전 모멘텀만큼 이동한 곳에서 기울기를 측정하여 이 기울기와 모멘텀을 함께 고려하여 최종적으로 이동할 곳을 찾는 것

$$\mathbf{w}^{temp} \leftarrow \mathbf{w} + \beta\mathbf{m}$$

$$\mathbf{m} \leftarrow \beta \leftarrow \mathbf{m} - \eta\nabla J(\mathbf{w}^{temp})$$

$$\mathbf{w} \leftarrow \mathbf{w}^{temp} + \mathbf{m}$$



Optimizer (8)

❖ **적응적 경사adaptive gradient , AdaGra 기법**

- 길게 늘어난 있는 축 방향으로 손실 함수 곡면의 크기를 줄여주면 더 좋은 수렴 성능을 보일 수 있음
- 기울기를 구할 때마다 기울기 벡터의 i번째 요소의 제곱값을 구해서 s_i 누적한다.
- 이 누적값은 해당 차원의 경사가 급할수록 큰 값이 되므로 기울기를 따라 파라미터를 이동할 때 이 s_i값으로 학습률을 조정하여 경사가 급한 곳은 천천히 이동하게 하면 손실 함수 곡면의 왜곡을 보정하는 역할

$$s_i \leftarrow s_i + \nabla J(\mathbf{w})_i^2$$

$$\mathbf{w}_i \leftarrow \mathbf{w}_i - \frac{\eta}{\sqrt{s_i + \epsilon}} \nabla J(\mathbf{w})_i$$

Optimizer (9)

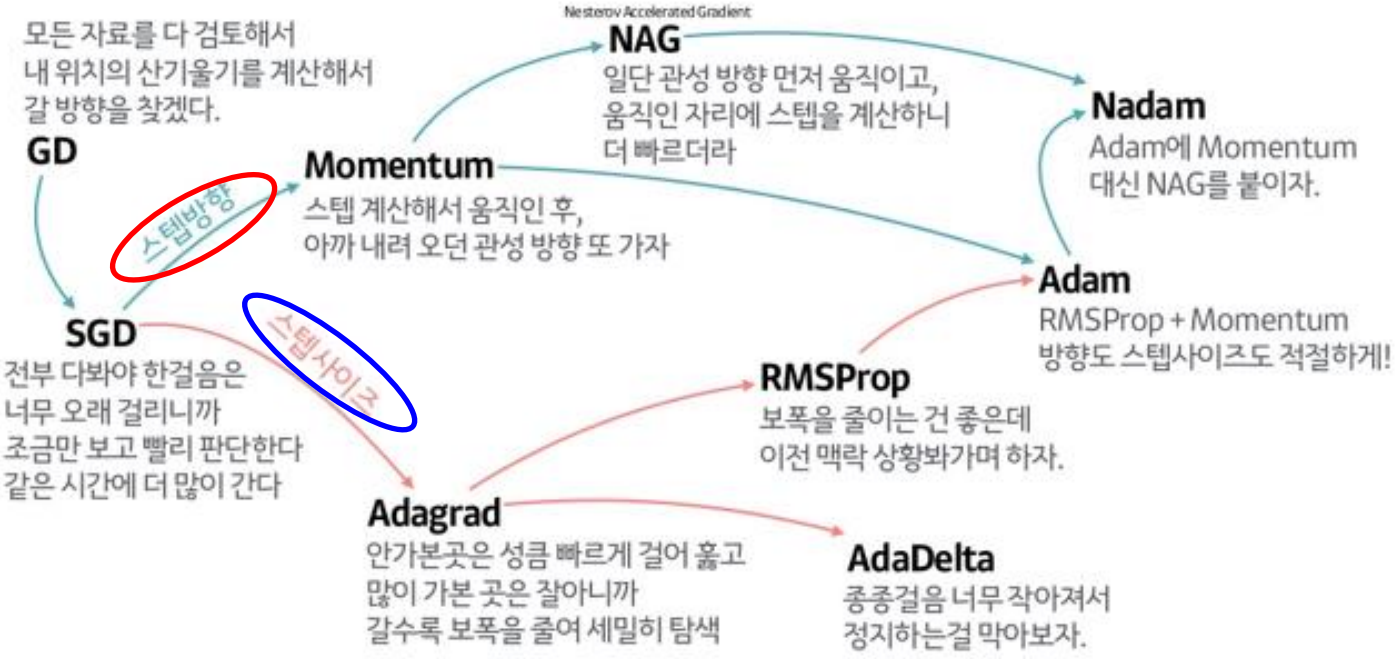
❖ **RMSProp 알고리즘**

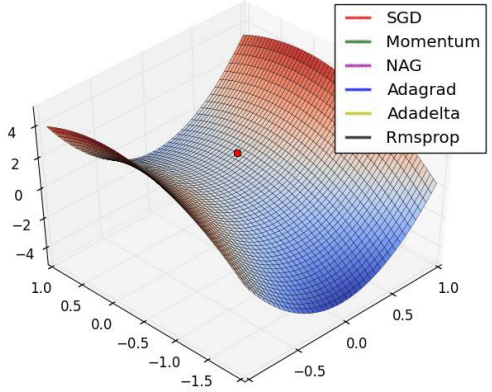
- 제프리 힌튼
- 적응적 경사 기법은 손실함수 곡면의 왜곡을 수정하지만, s는 지속적으로 증가하게 되고, 학습의 후반부에 학습률은 큰 수로 나누어지기 때문에 점점 해를 찾는 속도가 느려짐
- 간단히 이전 s_i가 일정한 비율로 쇠퇴하게 만들고 새로운 값이 쇠퇴한 만큼 추가되도록 하는 것으로 쇠퇴 비율 ρ는 보통 0.9 정도를 사용

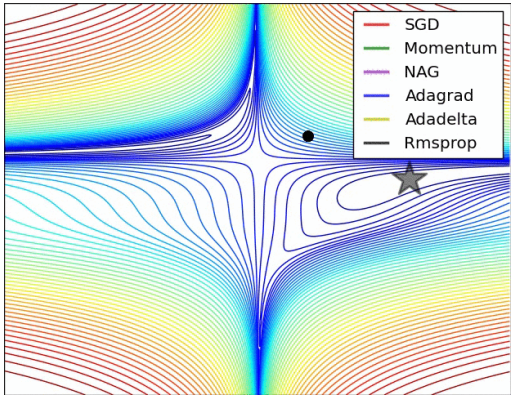
$$s_i \leftarrow \rho \cdot s_i + (1-\rho) \cdot \nabla J(\mathbf{w})_i^2$$

Optimizer (10)

❖ **Optimizer 종류**



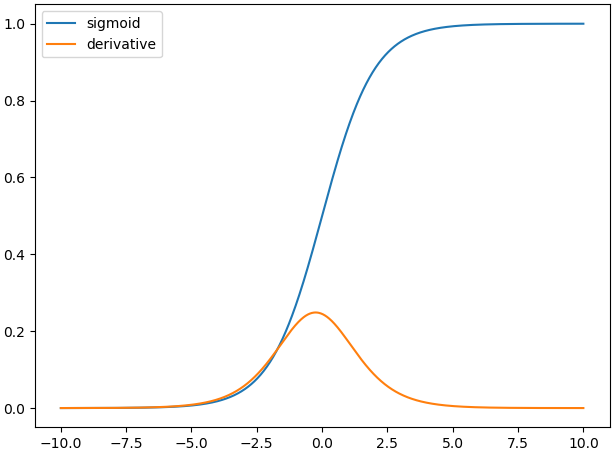




Loss Function Problem (1)

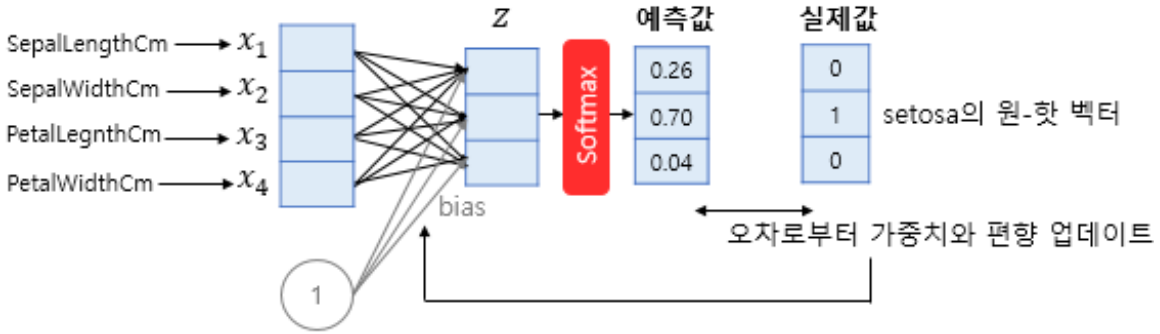
❖ **손실함수 MSE** $\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$ N: 출력노드 수

- 활성화 함수를 sigmoid로 사용할 경우
  - 손실함수를 $E = \frac{1}{2}(y - \hat{y})^2$로 단순화
  - 활성화 함수 $\varphi = \frac{1}{1+e^{-x}}$, $z = wx + b$
  - $\frac{\partial E}{\partial w} = \big(y - \varphi(z)\big)\varphi'(z)x$
  - $\varphi'(z) = 0$ 일 때 $\varphi'(z)$가 최대값
  - $\varphi'(z)$가 5보다 커거나 -5보다 작으면 거의 0



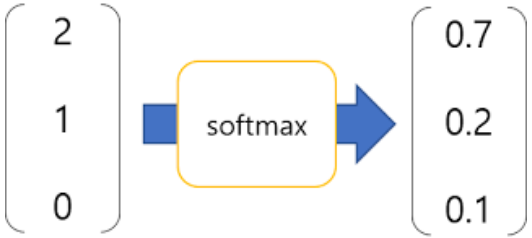
- 활성화 함수가 sigmoid이면 저속 수렴 (slow convergence)

❖ 출력층의 활성화 함수: SoftMax, 손실 함수: Cross Entropy



Loss Function Problem (2)

❖ **SoftMax 함수**



$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$

exp(x)로 인해 큰 수는 더 커지고 작은 수는 작아짐. 모든 수가 (0,1)

- hard argmax ([2, 1, 0]) = [1, 0, 0]
- soft argmax ([2, 1, 0]) ≈ [0.7 0.2 0.1]
  => 다른 출력값의 확률값도 유지

❖ 예

```
a = np.array([0.5, 4.1, 2.5, 5.6, 1.2])
print('신경망의 예측값 :', a)
print('소프트맥스 함수의 출력 :', softmax(a))
```

```
신경망의 예측값 : [0.5 4.1 2.5 5.6 1.2]
소프트맥스 함수의 출력 : [0.00473882 0.17343248 0.03501541 0.77727047 0.00954281]
```

```
# 소프트맥스 함수의 입력값을 두 배로 증가시켜보자
a = np.array([0.5, 4.1, 2.5, 5.6, 1.2]) * 2
print('신경망의 예측값 :', a)
print('소프트맥스 함수의 출력 :', softmax(a))
print('소프트맥스 함수의 최댓값 :', np.max(softmax(a)))
```

```
신경망의 예측값 : [ 1.   8.2  5.  11.2  2.4]
소프트맥스 함수의 출력 : [3.53328547e-05 4.73259126e-02 1.92910850e-03 9.50566364e-01
 1.43281791e-04]
소프트맥스 함수의 최댓값 : 0.9505663642857384
```

Loss Function Problem (3)

❖ 교차 엔트로피(cross-entropy)

- **두 개(목표, 실제)의 확률분포의 차이**를 구하기 위해 사용
- $P(x)$: 목표 확률분포는 **one-hot encoding**으로 표현


```
from keras.utils import to_categorical

data = np.array([0, 1, 2, 3, 4])  # 수치 데이터 0에서 4까지의 값
print('인코딩할 원본 데이터',data)
encoded = to_categorical(data)    # 원-핫 인코딩된 범주형 데이터 생성
print('원-핫 인코딩된 데이터 :')
print(encoded)
```

```
인코딩할 원본 데이터 [0 1 2 3 4]
원-핫 인코딩된 데이터 :
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
```

- $q(x)$: ML의 추정치의 확률
- **Cross-entropy**
  - 두 개의 확률을 교차

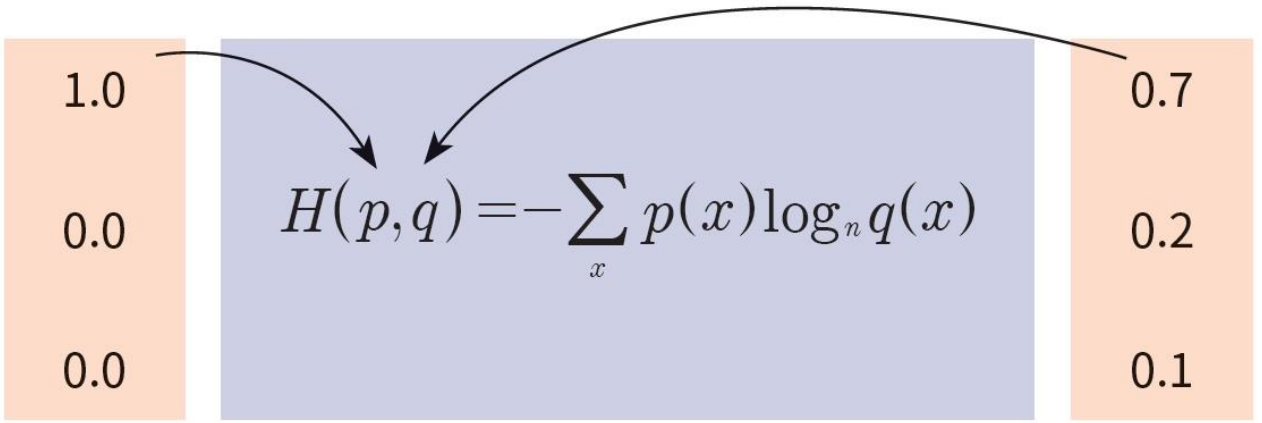
$$H(p,q) = -\sum_{x} p(x)\ \log q(x)$$

- 하나의 변수에 대한 entropy

$$H = -\sum p(x) \log p(x)$$

Loss Function Problem (4)

❖ **Cross-Entropy Loss 함수**



$$H(p,q) = -\sum_x p(x)\log_n q(x)$$
$$= -(1.0*\log 0.7 + 0.0*\log 0.2 + 0.0*\log 0.1)$$
$$= 0.154901$$

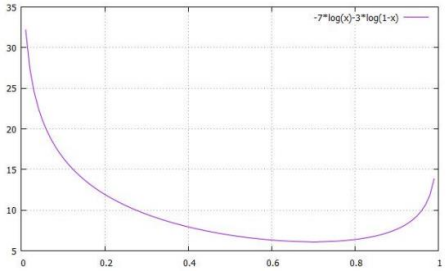
- 교차 엔트로피가 작으면 2개의 확률 분포가 거의 일치
  => 손실함수로 사용
  => MSE에 비해 오차값이 더 큼 => 학습속도를 더 빠르게 함
  => 예) 프로그램 참조

Loss Function Problem (5)

❖ Cross-Entropy Loss

$$E(\boldsymbol{w}) \equiv \frac{1}{|D|}\sum_{d\in D}\left(-y_d log(\hat{y}_d) - (1-y_d)log(1-\hat{y}_d)\right)$$

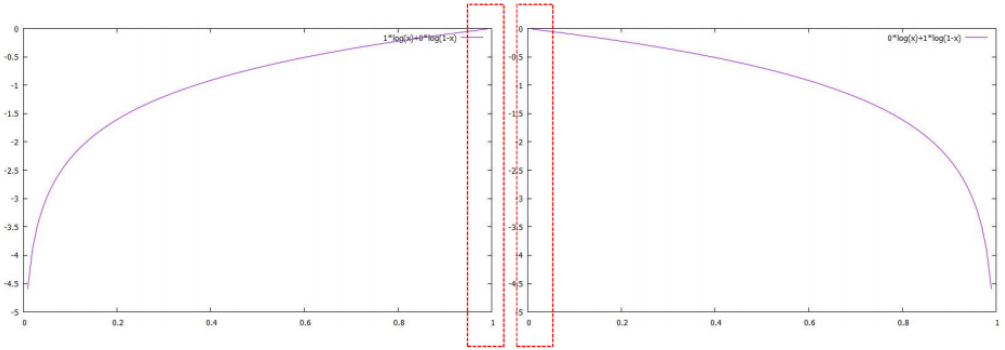
0~1사이 0 또는 1



❖ Cross-Entropy의 그래프

- 최소값을 찾는 문제
- 부호를 바꾸면 최대값을 찾는 문제: $+ylog(\hat{y}) + (1-y)log(1-\hat{y})$

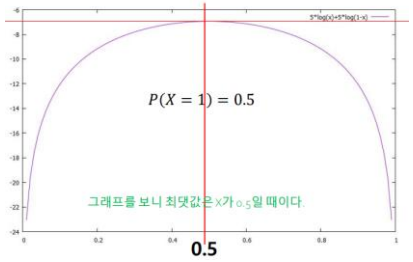
최댓값

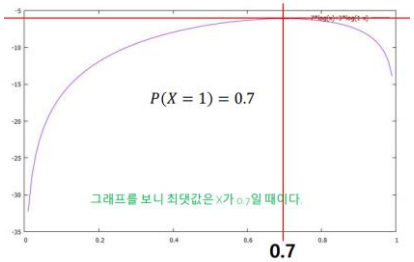


성공만 나온 경우
$P(X=1)=1$
성공과 실패가 반반 나온 경우

1 0

실패만 나온 경우
$P(X=1)=0$
성공 70%과 실패 30%나온 경우

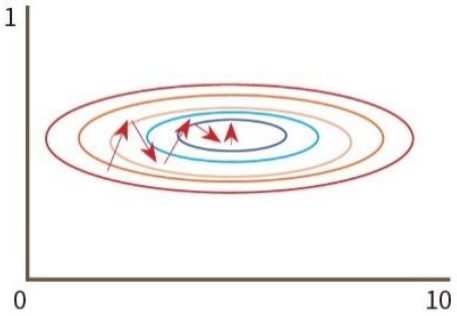




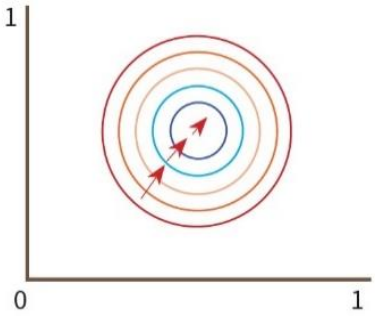
Data Normalization (1)

❖ 경사하강법과 Data 속성의 scale

- Data의 scale(단위)이 다를 경우, 큰 범위의 입력값의 그래디언트가 매개변수 갱신을 주도

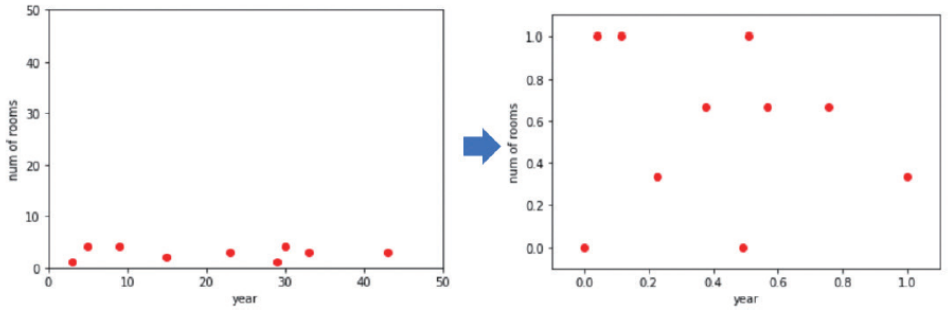


특징값의 범위가
다른 경우



특징값이 [0, 1]에서
움직이는 경우

연도	15	30	23	5	9	43	33	29	3	56
방의 수	2	4	3	4	4	3	3	1	1	2
가격	3.2	3.4	5	3.4	3.7	1.1	1.5	3.9	5.3	3.0

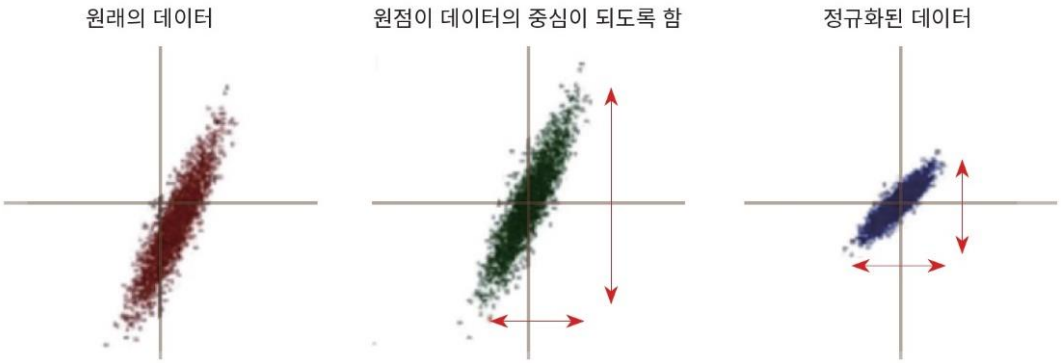


Data Normalization (2)

❖ **Data Normalization**

- Data의 scale(단위)를 [0,1] 사이로 조정
- Sklearn.preprocessing.MinMaxScaler()

$$\tilde{x} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$



❖ **Standardization**, mean removal and variance scaling

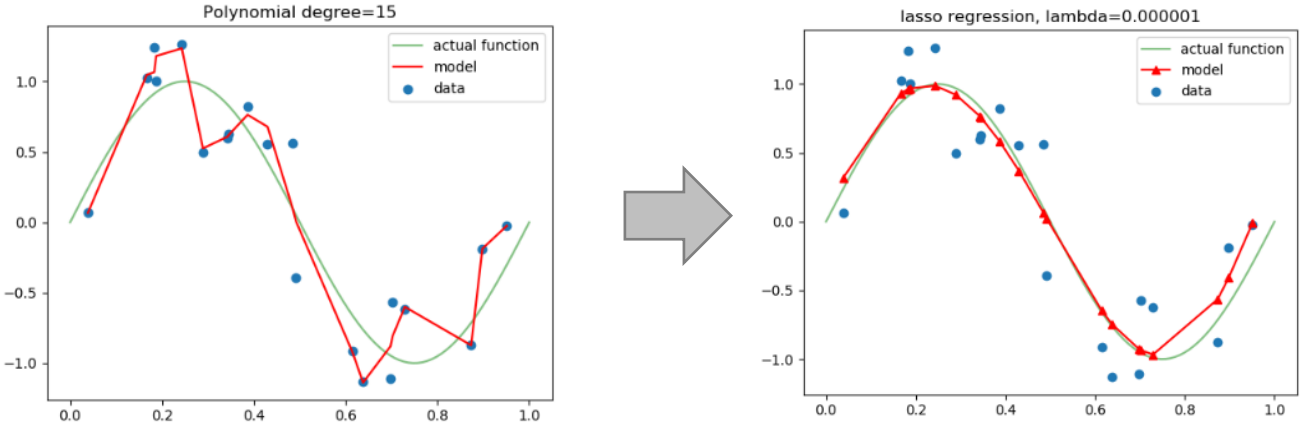
- 평균이 0이고 표준편차가 1인 정규분포를 가지도록 데이터를 조정
- Whitening: 전체 데이터를 균일하게 분포시킴
- Sklearn.preprocessing.StandardScaler()

$$x' = \frac{x - \mu_x}{\sigma_x}$$

Regularization (1)

❖ Regularization(규제화)

- 모델이 복잡하여 생긴 과적합을 해결

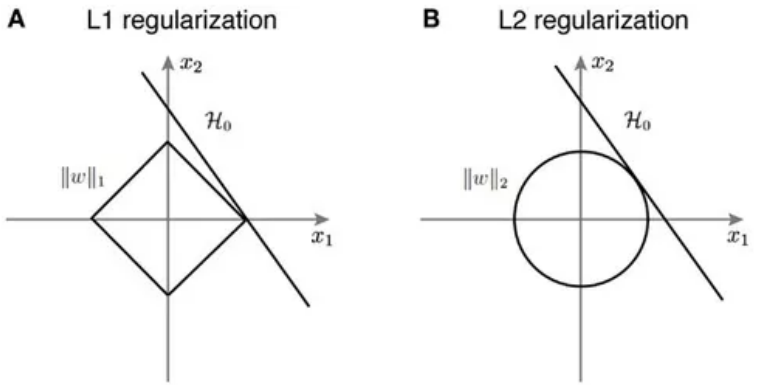


- L1 Regularization (Lasso)

$$Cost = \frac{1}{n}\sum_{i=1}^{n}\{L(y_i, \hat{y_i}) + \frac{\lambda}{2}|w|\}$$

- L2 Regularization (Ridge)

$$Cost = \frac{1}{n}\sum_{i=1}^{n}\{L(y_i, \hat{y_i}) + \frac{\lambda}{2}|w|^2\}$$



L1 Regularization 의 경우 위 그림처럼 미분 불가능한 점이 있기 때문에 Gradient-base learning 에는 주의가 필요

Regularization (2)

❖ **Data Augmentation**

- 데이터가 부족하여 생긴 과적합을 해결

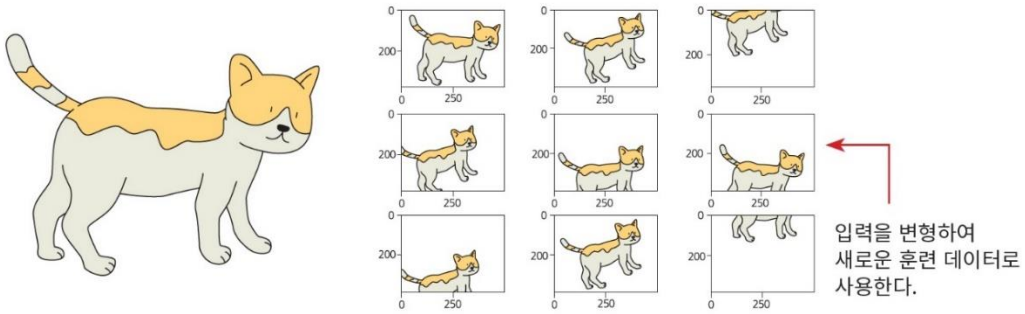


그림 14–18 데이터 증강 방법

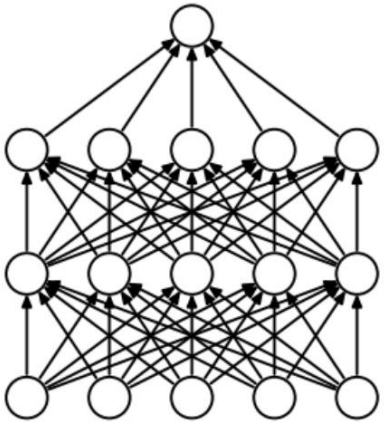
❖ https://playground.tensorflow.org

- Batch size
- Learning Rate
- Regularization, Regularization rate

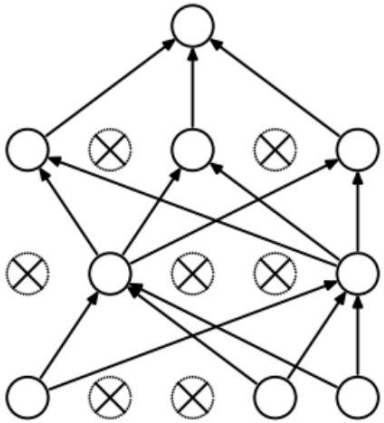
Regularization (3)

❖ **Drop Out**

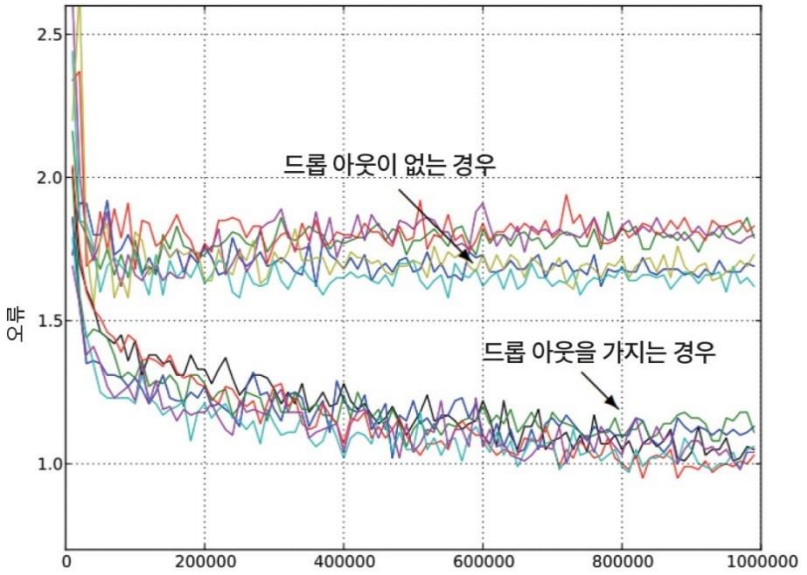
- 은닉층의 노드가 너무 많아서 생긴 과적합을 해결
- 학습 시 전체 신경만 중 일부만 사용 (동일한 입력에 대해 여러 노드가 중복하여 특징을 학습)
- 예측을 할 때는 모든 노드를 사용



(a) Standard Neural Net



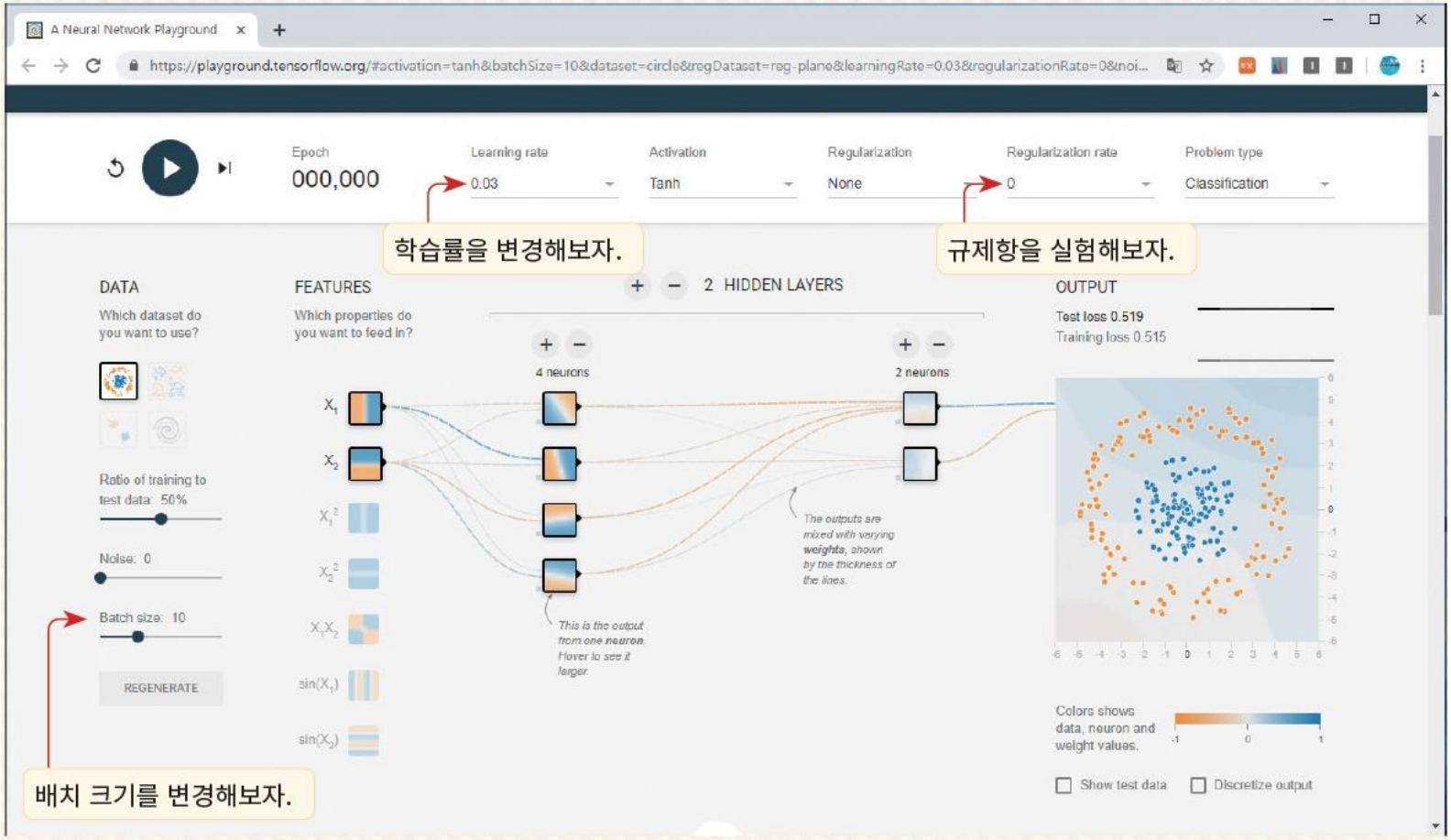
(b) After applying dropout.



Regularization (4)

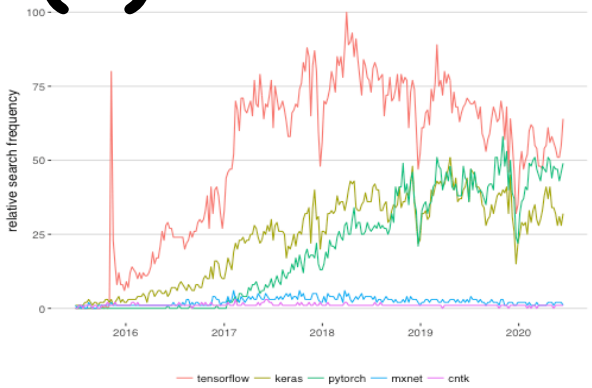
❖ **Regularization 실험**

- https://playground.tensorflow.org
- Batch size, Learning Rate, Regularization, Regularization rate



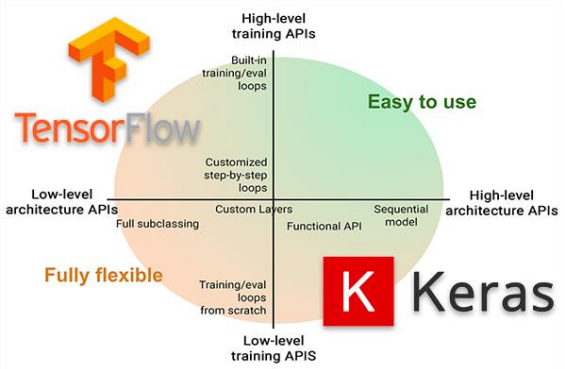
TensorFlow & Keras (1)

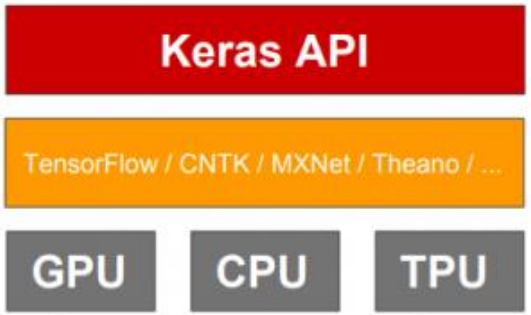
- Deep Learning programming architecture
  - Tensorflow: low-level architecture
  - Keras, Pytorch, … : high-level architecture



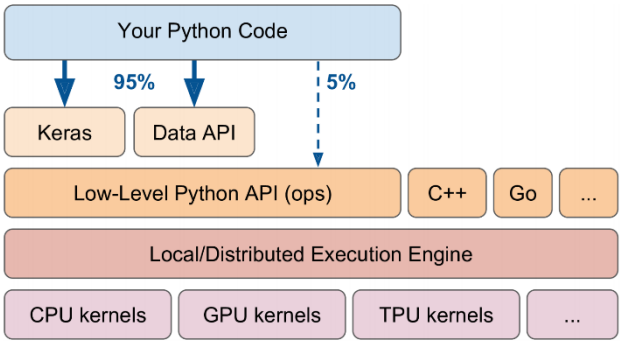
https://opendatascience.com/deep-learning-with-tensorflow-2-pytorch/

- Keras
  - https://keras.io
  - https://www.youtube.com/watch?v=UYRBHFAvLSs
  - https://www.youtube.com/watch?v=uhzGTijaw8A





https://punchplatform.com/2019/07/08/tensorflow-keras-pml-pipeline/

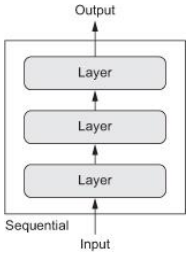


TensorFlow & Keras (2)

❖ Keras API reference https://keras.io/api

TensorFlow & Keras (3)

❖ Sequential API:




```
model = Sequential()
model.add(LSTM(50, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
model.fit(X_train, y_train, epochs=500, verbose=0)
y_pred = model.predict(X_test)
```

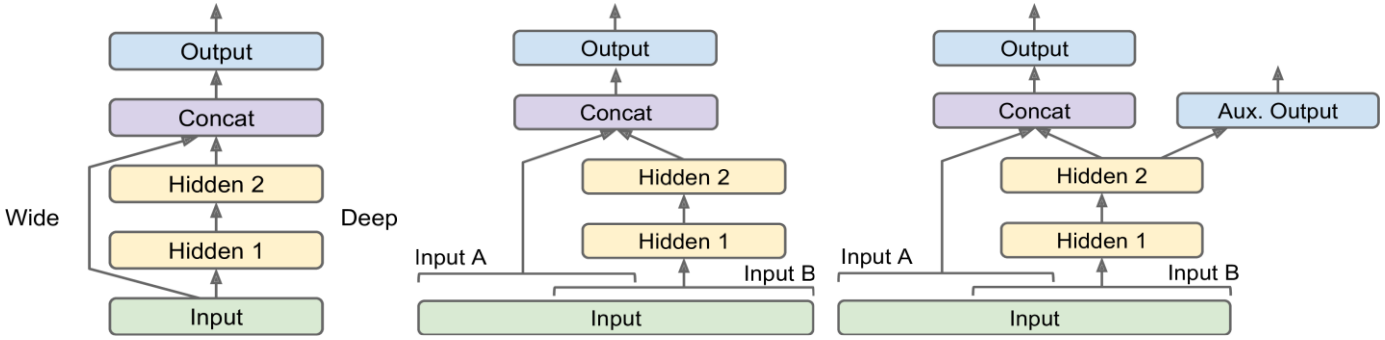
initialize model

add layers & compile model

fit model

make prediction

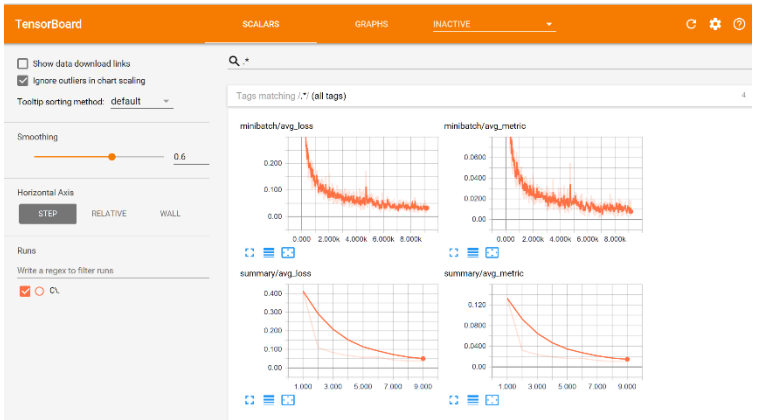
❖ Functional API: Building Complex Models



❖ Subclassing API: Building Dynamic Models

❖ Callbacks

❖ Visualization Using TensorBoard



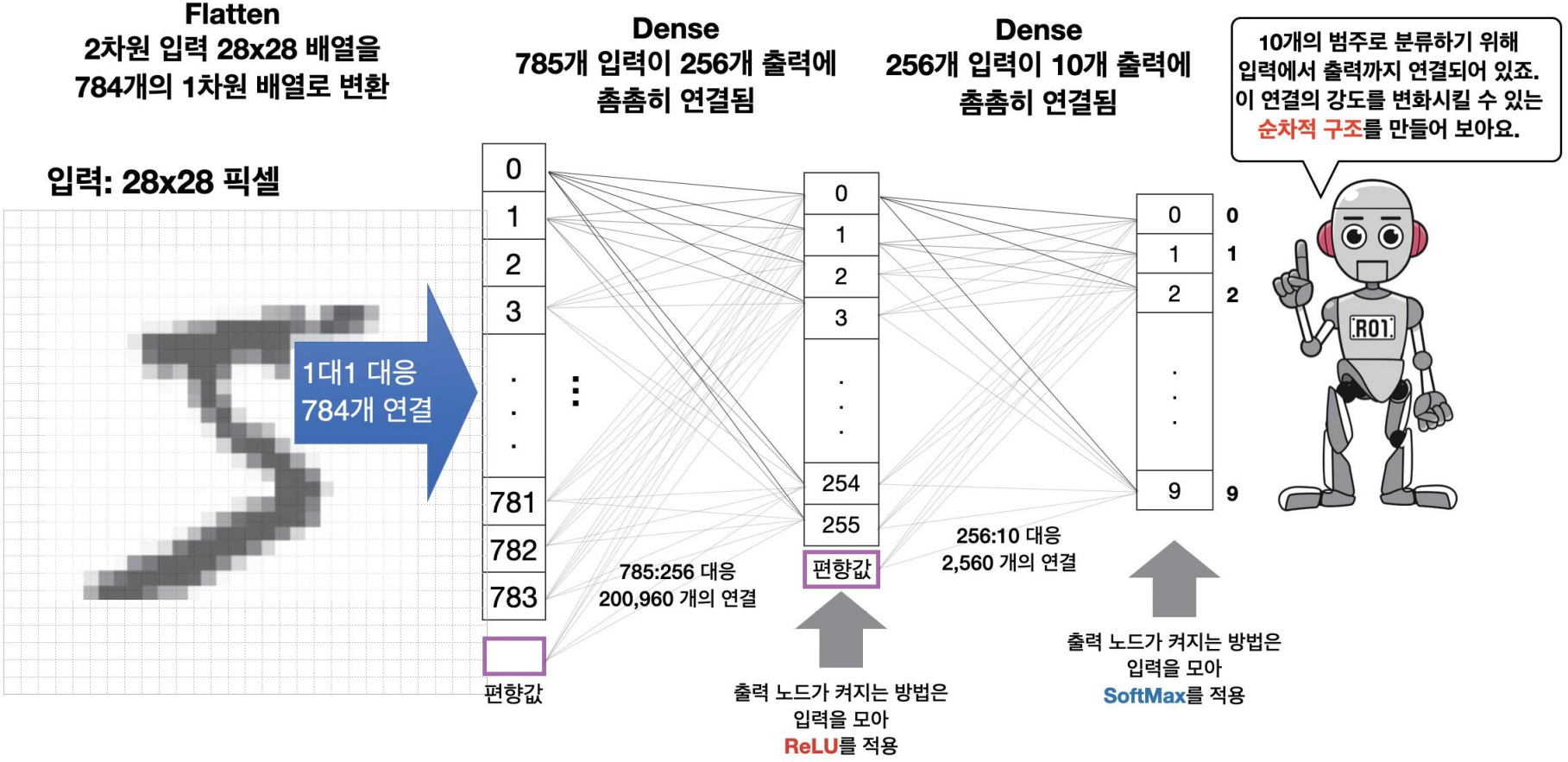
TensorFlow & Keras (4)

- Fine-Tuning Neural Network Hyperparameters
  - Number of Hidden Layers
  - Number of neurons per layer
  - Learning Rate
  - Optimizer
  - Batch Size
  - Activation function
  - Number of iterations
  - Weight initialization logic
  - Loss function
  - metrics

Program (1)

❖ Tutorial TensorFlow & Keras

- https://www.tensorflow.org/tutorials?hl=ko
- https://colab.research.google.com/drive/1bmzbi5JC-r3AYRp4SfdotyG6t8SzrDAX



Program (1)

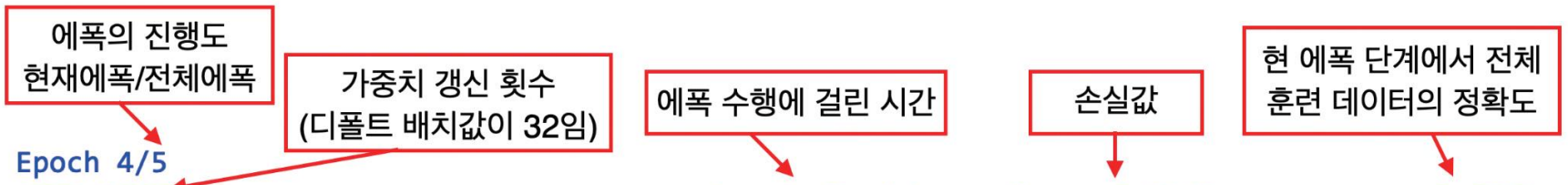
```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt

mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255, x_test / 255  # 입력값 정규화

model = keras.models.Sequential( [
    keras.layers.Flatten(input_shape = (28, 28)),
    keras.layers.Dense(256, activation = 'relu'),
    keras.layers.Dense(10, activation = 'softmax')])

# 학습을 위한 최적화 함수, 손실 함수등을 가진 모델을 컴파일
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, epochs = 4)
```

Mini batch: 32



Program (1)

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 flatten_1 (Flatten)          (None, 784)               0

 dense_2 (Dense)              (None, 256)               200960

 dense_3 (Dense)              (None, 10)                2570

=================================================================
Total params: 203,530
Trainable params: 203,530
Non-trainable params: 0
```

```
print('신경망 모델의 학습 결과 :')
eval_loss, eval_acc = model.evaluate(x_test, y_test)
print('test 데이터의 손실값', eval_loss, 'test 데이터의 정확도', eval_acc)
```

```
신경망 모델의 학습 결과 :
313/313 [==============================] - 1s 2ms/step - loss: 0.0709 - accuracy: 0.9775
test 데이터의 손실값 0.07090707123279572 test 데이터의 정확도 0.9775000214576721
```

Program (1)

❖ 정확률 개선: layer 추가, epochs = 10

```
model = keras.models.Sequential( [
    keras.layers.Flatten(input_shape = (28, 28)),
    keras.layers.Dense(64, activation = 'relu'),
    keras.layers.Dense(64, activation = 'relu'),
    keras.layers.Dense(64, activation = 'relu'),
    keras.layers.Dense(10, activation = 'softmax'),
])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history = model.fit(x_train, y_train, epochs = 10, validation_data = (x_test, y_test))
```

```
Epoch 9/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0332 - accuracy: 0.9893 - val_loss: 0.1020 - val_accuracy: 0.9739
Epoch 10/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0303 - accuracy: 0.9901 - val_loss: 0.1045 - val_accuracy: 0.9746
```

```
print('신경망 모델의 학습 결과 :')
eval_loss, eval_acc = model.evaluate(x_test, y_test)
print('test 데이터의 손실값', eval_loss, 'test 데이터의 정확도', eval_acc)
```

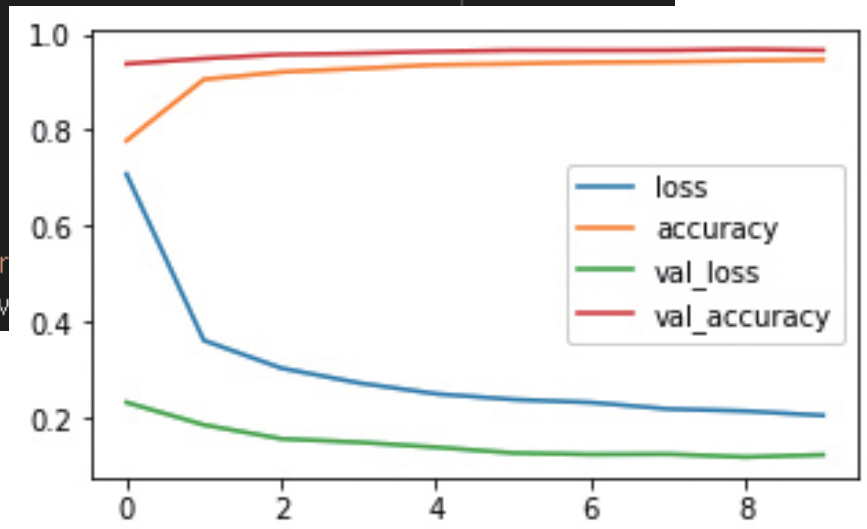
```
신경망 모델의 학습 결과 :
313/313 [==============================] - 1s 2ms/step - loss: 0.1045 - accuracy: 0.9746
test 데이터의 손실값 0.10447640717029572 test 데이터의 정확도 0.9746000170707703
```

- 과적합 => 규제화 필요 (dropout)

Program (1)

❖ 규제화: dropout

```
# 드롭아웃 계층을 가진 신경망 모델 만들기
model = keras.models.Sequential( [
    keras.layers.Flatten(input_shape = (28, 28)),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(64, activation = 'relu'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(64, activation = 'relu'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(64, activation = 'relu'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(64, activation = 'relu'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(64, activation = 'relu'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(64, activation = 'relu'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(10, activation = 'softmax'),
])
model.compile(optimizer='adam', loss='sparse_categor
history = model.fit(x_train, y_train, epochs = 10, v
```

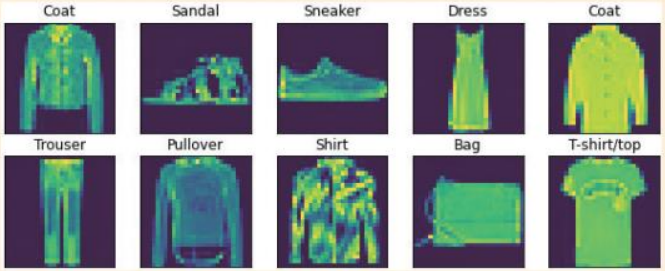


Program (2)

- https://colab.research.google.com/drive/1bmzbi5JC-r3AYRp4SfdotyG6t8SzrDAX
- **LAB8-1** Fashion-MNIST 데이터 분류하기

실습 목표

케라스 모듈에는 다음과 같이 **운동화나 셔츠같은 옷과 신발의 이미지와 레이블을 제공**하고 있다. 이 이미지의 갯수는 MNIST와 동일하게 6만장이며 28x28 픽셀 크기로 저장되어 있다. 그리고 테스트를 위한 데이터로 1만장의 이미지도 따로 준비되어 있다.



Fashion MNIST 데이터를 이용하여 다음과 같은 신경망을 생성하고, 학습시킨 다음 정확도를 구해 보자.(이때 에폭의 수는 10으로하고 배치의 크기는 64로 두자)

1. 각각 128개와 32개의 노드를 가진 두 개의 은닉층
2. 입력층에서 128개 노드의 은닉층으로 가는 연결은 20%의 드롭아웃을 사용
3. 10개의 출력 노드 각각은 10개의 클래스에 해당

힌트

이 랩에서는 Fashion MNIST 데이터를 읽어오는 부분을 제외한다면 이전의 코드와 유사한 코드로 문제를 해결할 수 있다. Fashion MNIST 데이터는 다음과 같은 방법으로 훈련 이미지와 레이블, 검증 이미지와 레이블을 읽어올 수 있다.

```
fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_
mnist.load_data()
```