딥러닝 #02

Keras 활용

문자열 입력 부분은 모두 함수로 대체가 가능

- tf.keras.activations
- tf.keras.optimizers
- tf.keras.losses
- tf.keras.metrics

```
tf.keras.backend.clear_session()

# 모델 생성 or 선택
X = tf.keras.layers.Input(shape=(784, ))
H = tf.keras.layers.Dense(128, activation="relu")(X)
H = tf.keras.layers.Dense(128, activation="relu")(H)
H = tf.keras.layers.Dense(128, activation="relu")(H)
Y = tf.keras.layers.Dense(10, activation="softmax")(H)

model = tf.keras.models.Model(X, Y)
model.compile(
    optimizer="adam",
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)
```

문자열 입력 부분은 모두 함수로 대체가 가능

- tf.keras.activations
- tf.keras.optimizers
- tf.keras.losses
- tf.keras.metrics

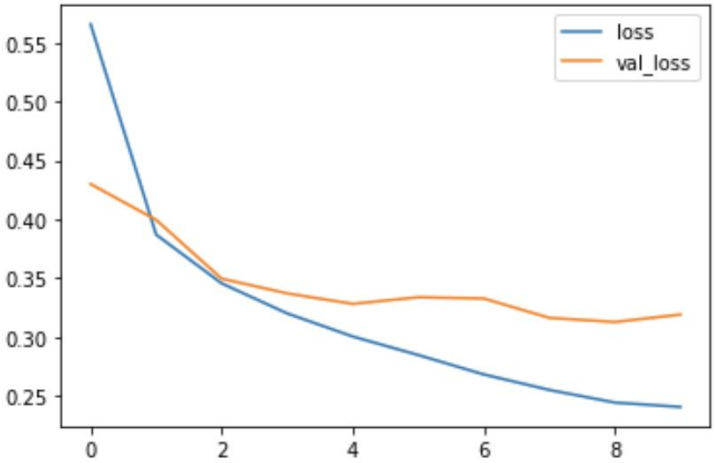
```
tf.keras.backend.clear_session()

# 모델 생성 or 선택
X = tf.keras.layers.Input(shape=(784, ))
H = tf.keras.layers.Dense(128, activation=tf.keras.activations.relu)(X)
H = tf.keras.layers.Dense(128, activation=tf.keras.activations.relu)(H)
H = tf.keras.layers.Dense(128, activation=tf.keras.activations.relu)(H)
Y = tf.keras.layers.Dense(10, activation=tf.keras.activations.softmax)(H)

model = tf.keras.models.Model(X, Y)
model.compile(
    optimizer=tf.keras.optimizers.Adam(0.001),
    loss=tf.keras.losses.categorical_crossentropy,
    metrics=[tf.keras.metrics.categorical_accuracy]
)
```

history

- 학습 진행 log 확인 가능
- loss와 metrics으로 지정된 값이 로깅된다.




```
result = model.fit(x_train, y_train,
                   epochs=10, batch_size=128,
                   validation_split=0.2)
model.evaluate(x_test, y_test)
```

```
import matplotlib.pyplot as plt

plt.plot(result.history['loss'])
plt.plot(result.history['val_loss'])
plt.legend(['loss', 'val_loss'])
```

Custom Loss / Metrics

- 직접 함수를 작성하여 적용
- y_true, y_pred 파라미터를 받는다.

```
17 model.compile(
18     loss=my_entropy,
19     optimizer=tf.keras.optimizers.Adam(0.001),
20     metrics=[my_accuracy]
21 )
```

```
1 def my_entropy(y_true, y_pred):
2     loss = tf.keras.backend.binary_crossentropy(y_true, y_pred)
3     return tf.reduce_mean(loss, axis=-1)
4
5 def my_accuracy(y_true, y_pred):
6     # tf.print(y_true, y_pred > 0.5)
7     equals = tf.math.equal(y_true, tf.cast(y_pred > 0.5, tf.float32))
8     return tf.reduce_mean(tf.cast(equals, tf.float32))
9
```

Model Save & Load

방법 1.
가중치와 모델을 한꺼번에
저장했다가 꺼내는 방법

```
model.save('model.h5', include_optimizer=True)
```

```
model = tf.keras.models.load_model('my_model.h5')
```

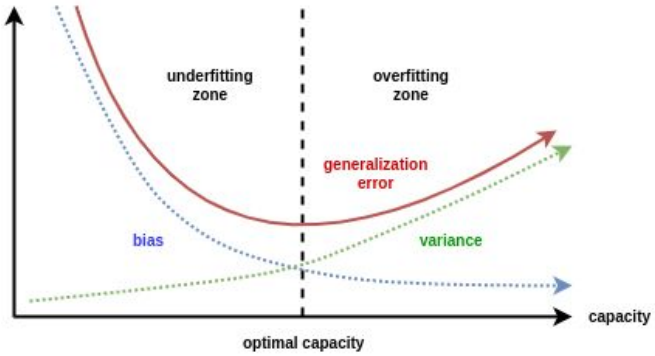
방법 2.
가중치와 모델을 따로
저장했다가 꺼내는 방법

```
model.save_weights('model_weights.h5')
with open('model.json', 'w') as f:
    f.write(model.to_json())
```

```
with open('model.json', 'r') as f:
    model = tf.keras.models.model_from_json(f.read())
model.load_weights('model_weights.h5')
```

Early Stopping

- overfitting이 발생하면 멈추고 멈추기 전 최적의 모델을 적용한다.




```python
es = tf.keras.callbacks.EarlyStopping(
    monitor = 'val_loss',
    min_delta = 0, # 개선되고 있다고 판단하기 위한 최소 변화량
    patience = 10, # 개선 없는 epoch 얼마나 기달려 줄거야?
    verbose = 1
)

# 학습
result = model.fit(
    x_train, y_train,
    epochs=20000, batch_size=128, validation_split=0.2,
    callbacks=[es]
)

model.evaluate(x_test, y_test)
```

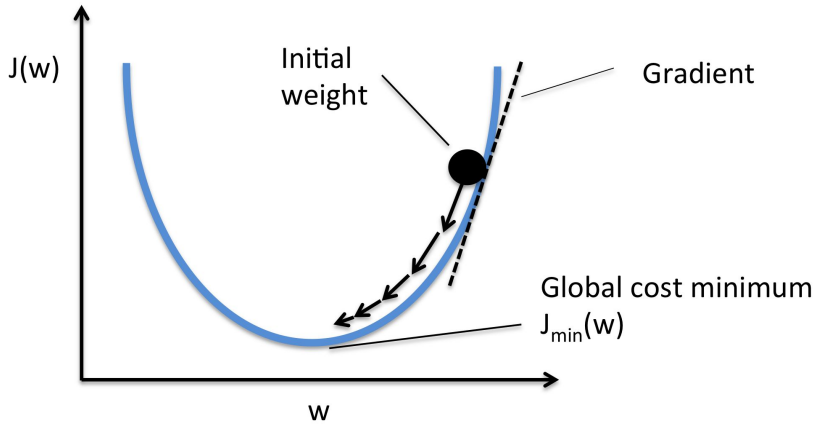
Optimizer

Gradient Descent (경사하강법)

함수의 기울기(경사)를 구하여 함수의 극값에 이를 때까지 기울기가 낮은 쪽으로 반복하여 이동하는 방법.

$$W(t+1) = W(t) - \alpha \frac{\partial}{\partial w} Cost(w)$$

- SGD

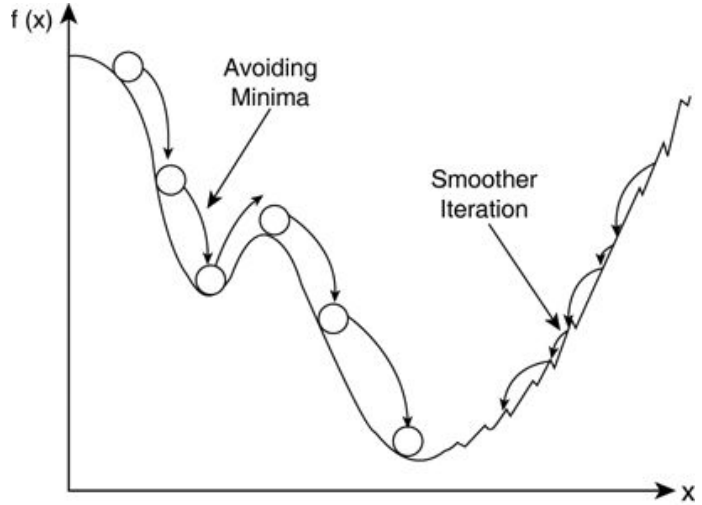


Momentum (관성)

이전에 이동했던 방향을 기억해서 다음 이동의 방향에 반영.

- NAG

$$V(t) = m * V(t-1) - \alpha \frac{\partial}{\partial w} Cost(w)$$
$$W(t+1) = W(t) + V(t)$$



Adagrad (Adaptive Gradient)

많이 이동한 변수(w)는 최적값에 근접했을 것이라는 가정하에, 많이 이동한 변수(w)를 기억해서 다음 이동의 거리를 줄인다.

- RMSprop

$$G(t) = G(t-1) + \left(\frac{\partial}{\partial w(t)} Cost(w(t))\right)^2$$

$$= \sum_{i=0}^{t} \left(\frac{\partial}{\partial w(i)} Cost(w(i))\right)^2$$

$$W(t+1) = W(t) - \alpha * \frac{1}{\sqrt{G(t)+\epsilon}} * \frac{\partial}{\partial w(i)} Cost(w(i))$$

Adam (RMSprop + Momentum)

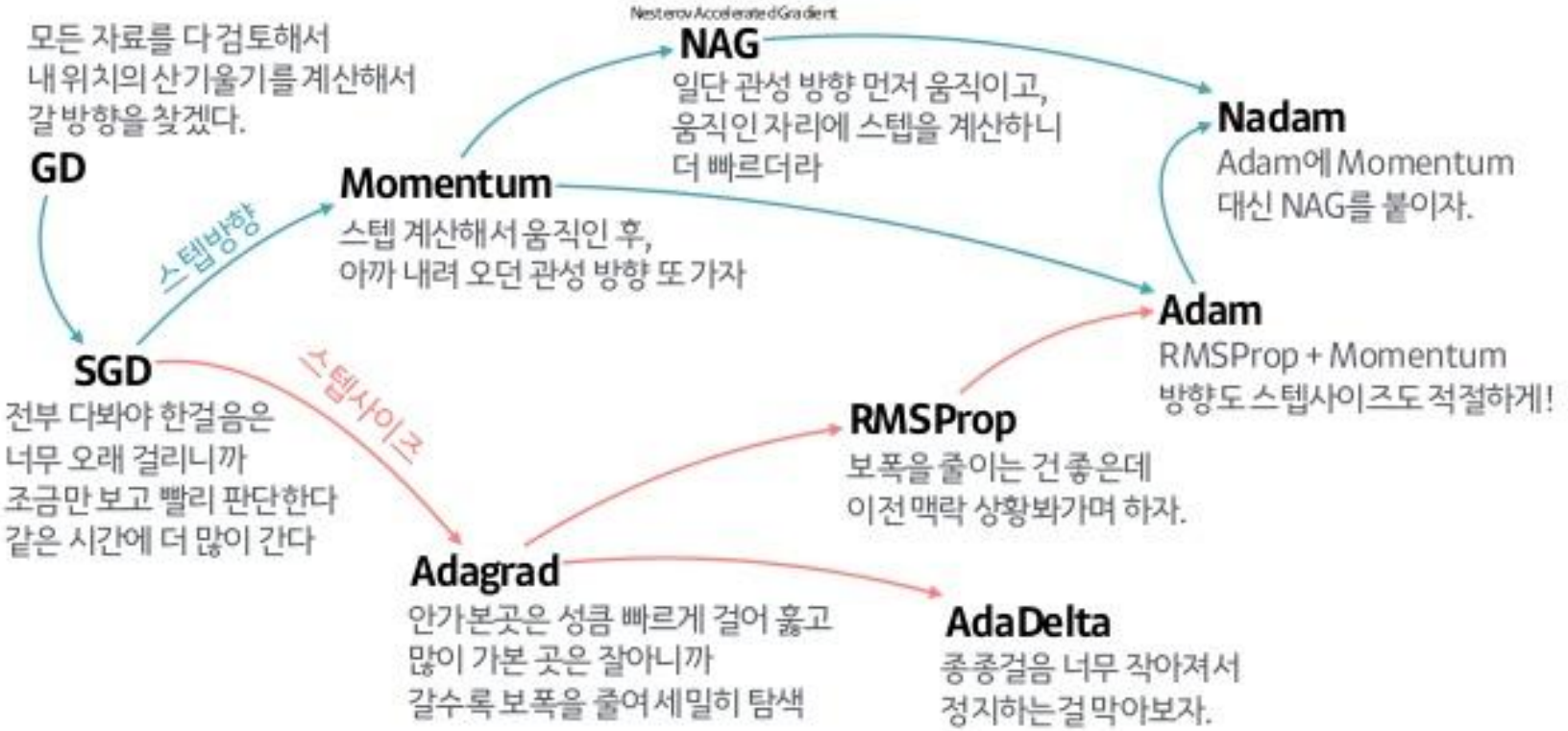
$$\mathrm{M(t)} = \beta_1 M(t-1) + (1-\beta_1)\frac{\partial}{\partial w(t)} Cost(w(t))$$

$$\mathrm{V(t)} = \beta_2 V(t-1) + (1-\beta_2)\left(\frac{\partial}{\partial w(i)} Cost(w(i))\right)^2$$

$$\widehat{M}(t) = \frac{M(\mathrm{t})}{1-{\beta_1}^t} \qquad \hat{V}(t) = \frac{V(\mathrm{t})}{1-{\beta_2}^t}$$

$$\mathrm{W(t+1)} = W(t) - \alpha * \frac{\widehat{M}(t)}{\sqrt{\hat{V}(t) + \epsilon}}$$

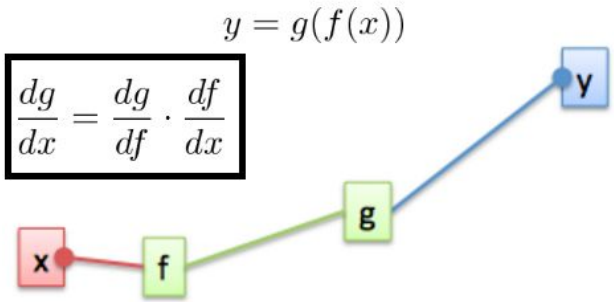
산 내려오는 작은 오솔길 잘찾기(Optimizer)의 발달 계보

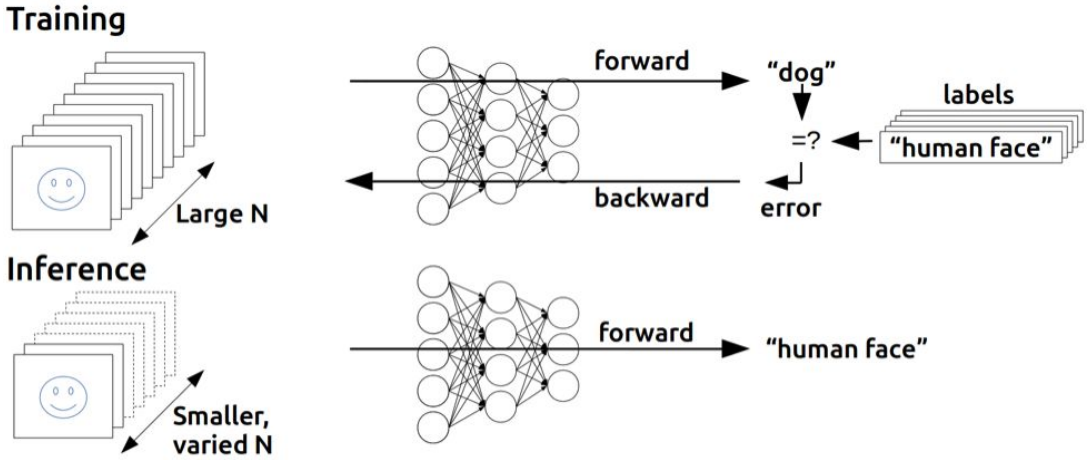


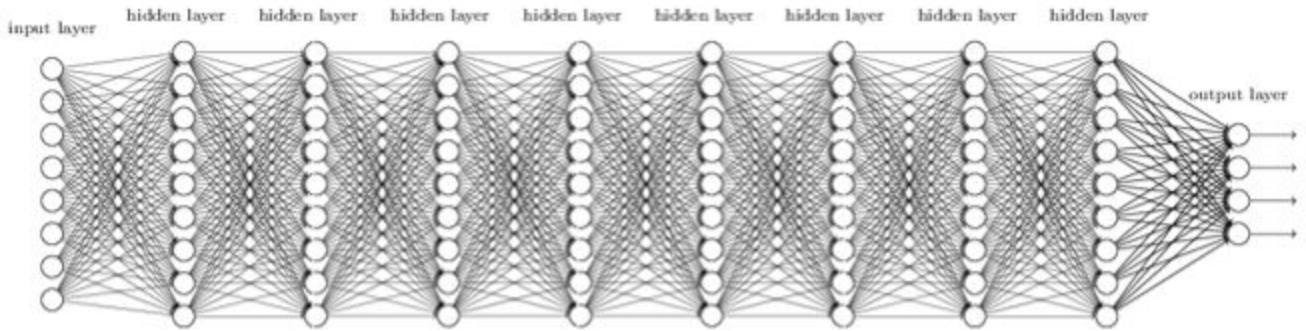
출처: 하용호, 자습해도 모르겠던 딥러닝, 머리속에 인스톨 시켜드립니다

Chain Rule & Back Propagation

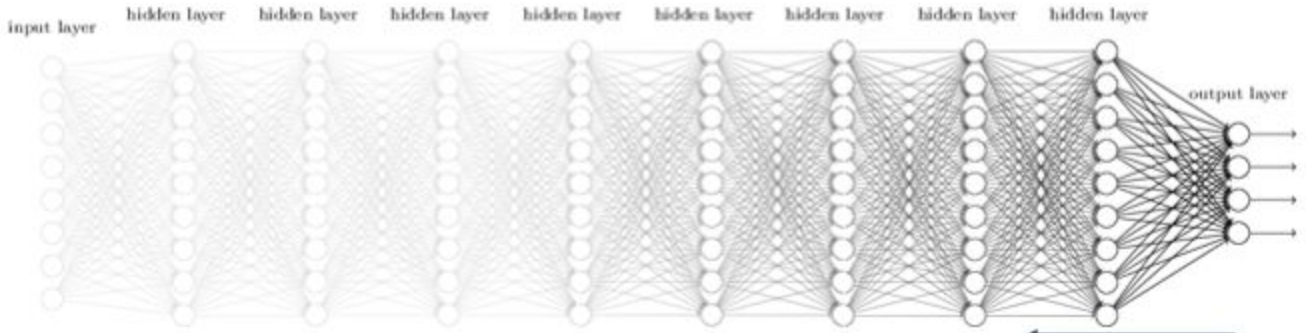
- 학습 = 'Cost'를 줄이는 방향으로 가중치를 조절한다.
- 가중치를 a만큼 변화시키면 Error는 어떻게 변화할까 → **미분을 해보면 된다.**
- 미분이 복잡해. 함수 안에 함수가 있고 그 안에 또… → **Chain Rule을 쓰면 된다!**







Deep Neural Network



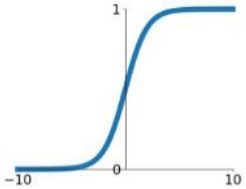
Vanishing Gradient

Activation

Activation (활성화 함수)

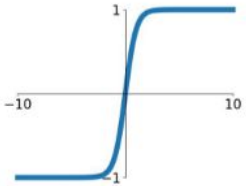
Sigmoid

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



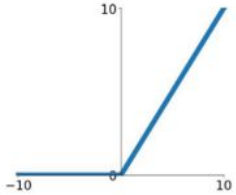
tanh

$$\tanh(x)$$



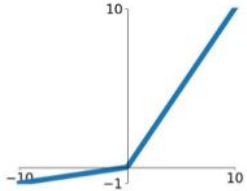
ReLU

$$\max(0, x)$$



Leaky ReLU

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

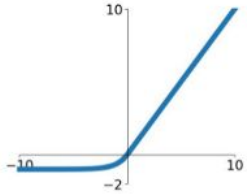


Maxout

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

ELU

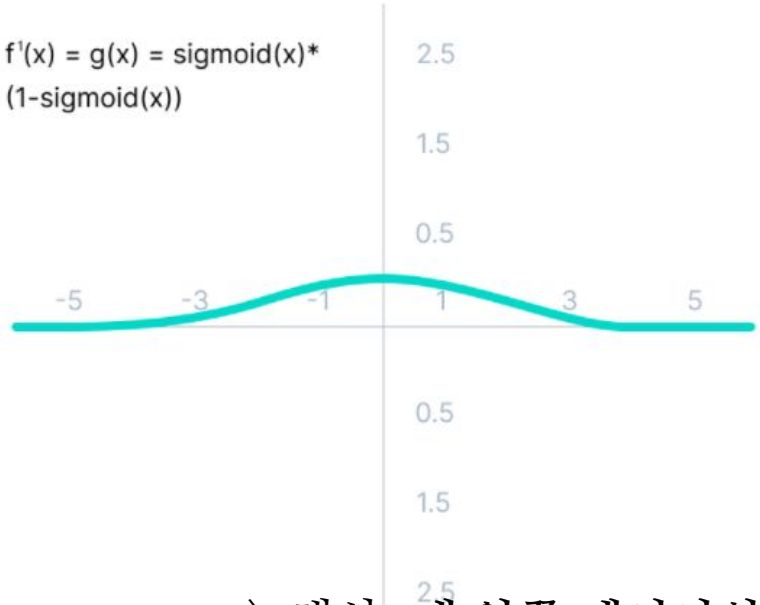
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

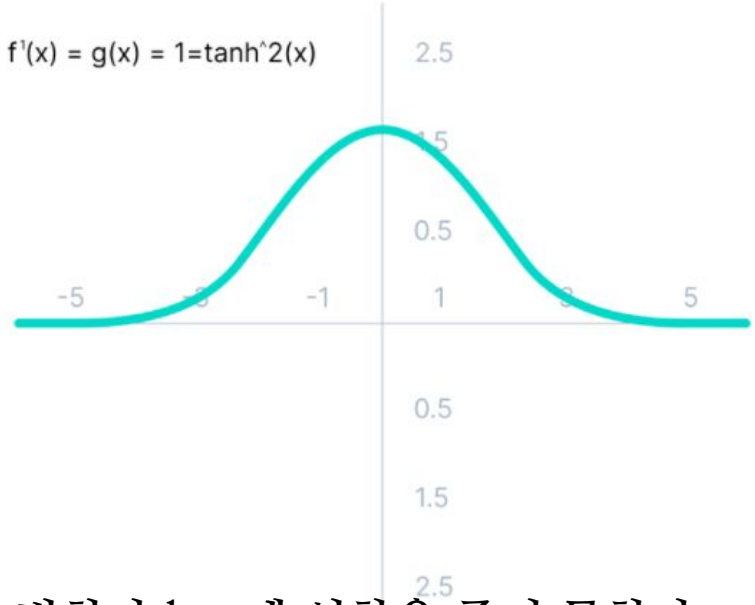


vanishing gradient 이유

아래는 sigmoid, tanh의 미분 함수

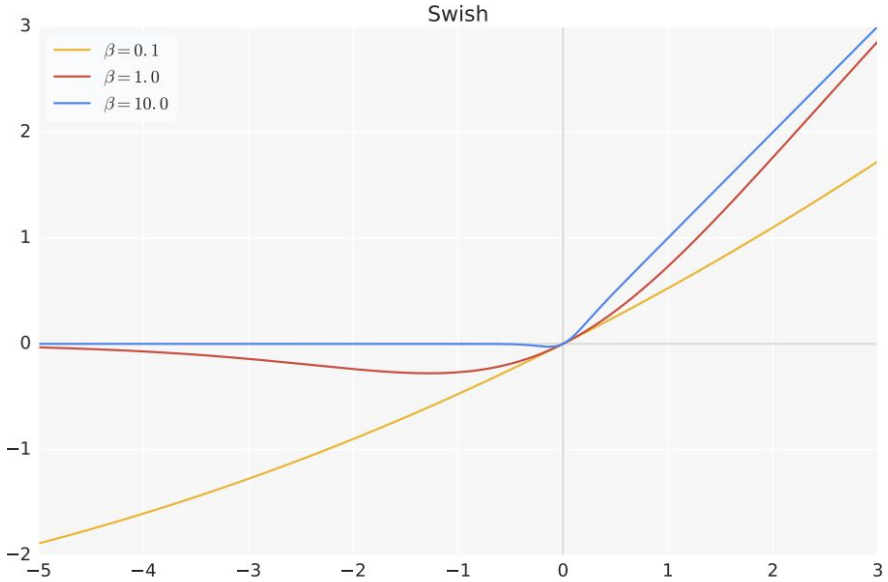
0.25/0.5 보다 작은 값이 중첩되어 곱해지면서 미분값이 0에 수렴하게 된다.



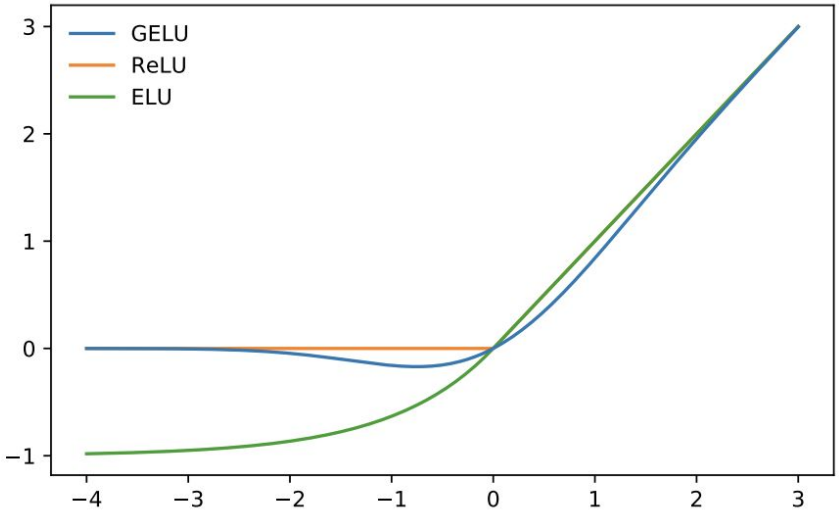


=〉 해석: **맨 앞쪽 레이어의 weight의 변화가 loss에 영향을 주지 못한다.**

최근의 좋은 활성화 함수



$$f(x) = x \cdot \text{sigmoid}(\beta x)$$



$$\text{GELU}(x) = x \cdot \frac{1}{2}\left[1 + \text{erf}(x/\sqrt{2})\right]$$

Loss Function

- MSE (Mean Square Error)

$$loss_i = (y_i - \hat{y}_i)^2$$

- Cross Entropy

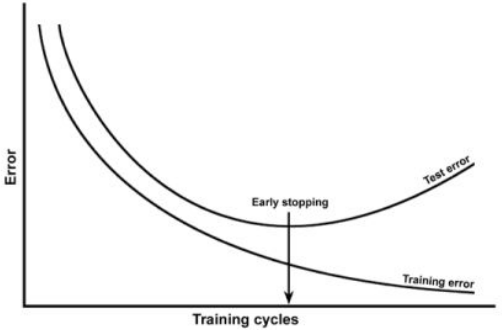
$$error = -\sum_i^n y_i \log \hat{y}_i \qquad loss_i = -(\, y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i))$$

- activation function – softmax

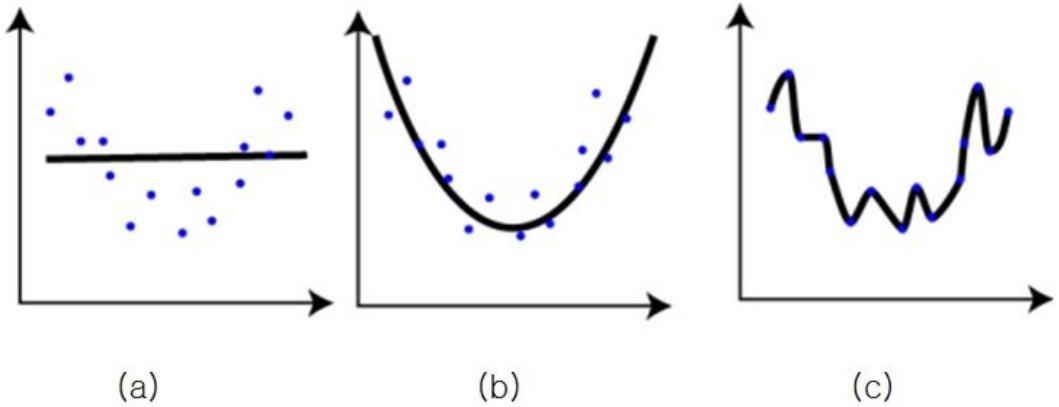
$$softmax(x_i) = \frac{e^{x_i}}{e^{x_1} + e^{x_2} + ... + e^{x_n}} = \frac{e^{x_i}}{\sum_c^n e^{x_c}}$$

Overfitting

훈련 데이터에 특화된 학습을 하게 되어 새로운 데이터에 대한 예측이 오히려 나빠지거나 학습의 효과가 나타나지 않는 상태



- 충분한 데이터 확보
- Dropout / Ensemble
- Batch Normalization
- L1, L2 Regularization
- Data Augmentation

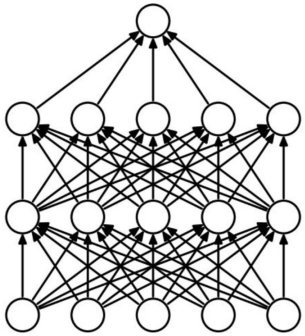


Dropout

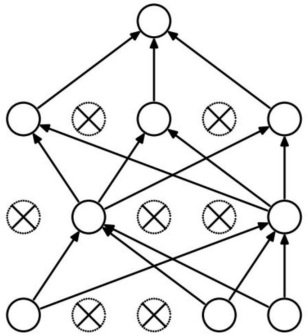
- 지정된 확률의 개수만큼 노드를 랜덤하게 제외하고 학습을 진행.
- 완성된 모델에서는 모든 노드를 사용.
- 효과
  - 오버피팅 방지
  - 성능 향상
  - 앙상블 효과

학습이 된다면 최대한 높게 줄수록 좋다.

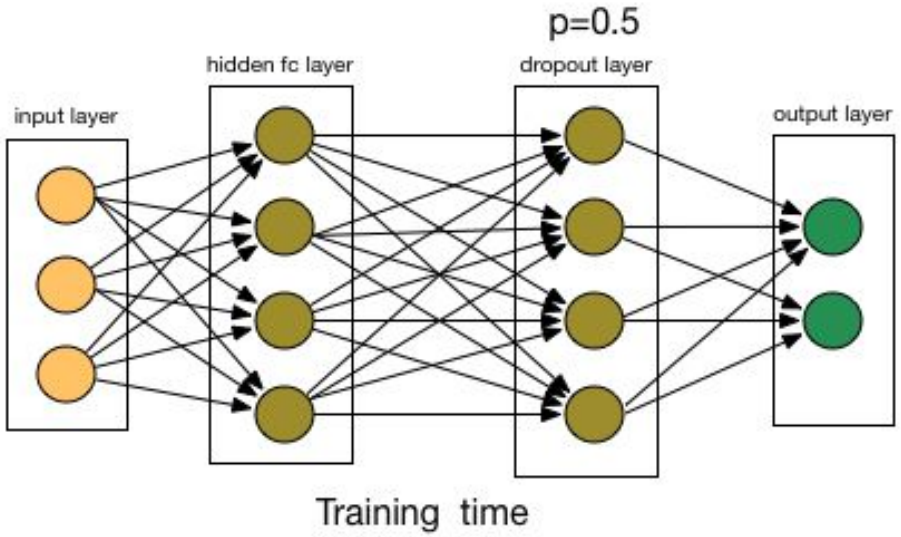
데이터가 충분하면 0.4 ~ 0.6



(a) Standard Neural Net



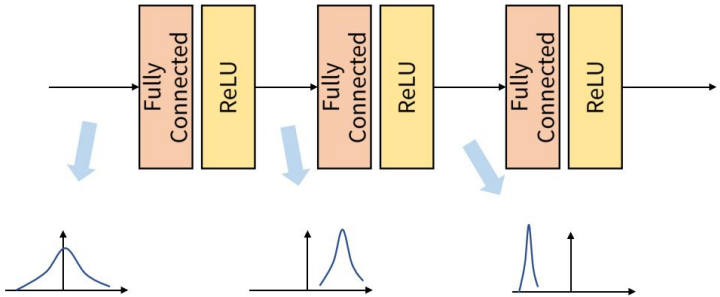
(b) After applying dropout.



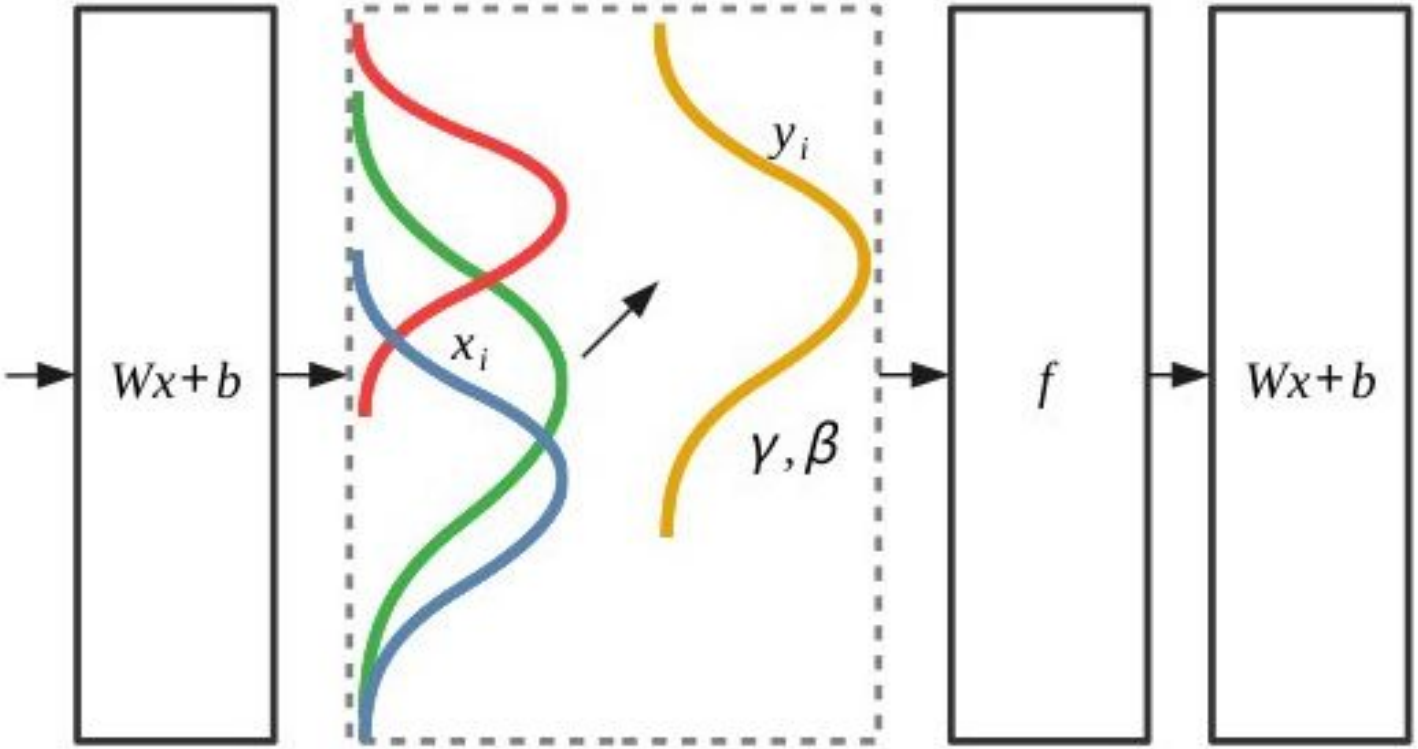
Batch Normalization

- Layer 사이에 중간 결과 데이터들을 표준정규화하는 계층
  - 학습 단계에서는 batch 단위로 정규화하고
  - 모델 적용시의 σ, μ 는 사용하던 값들의 이동평균값을 이용한다.
- Internal Covariant Shift 문제를 해결한다.
- 무조건 쓴다고 가정하자.

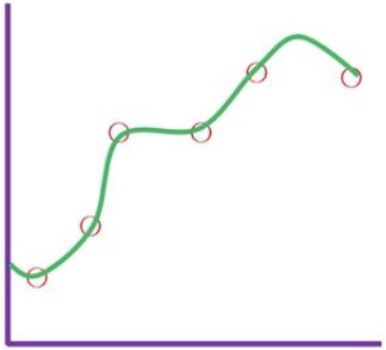
$$BN(\mathbf{x}_i) = \gamma \odot \left(\frac{\mathbf{x}_i - \mu_B}{\sigma_B}\right) + \beta,$$



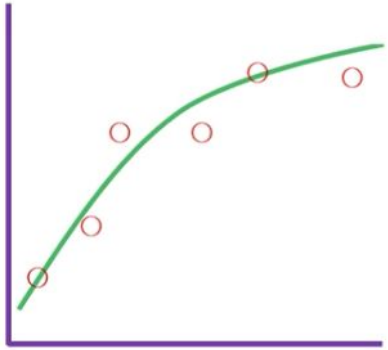
Batch Normalization



L1, L2 Regularization



$$\beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 x^4$$



$$\beta_0 + \beta_1 x + \beta_2 x^2$$

$$\min_{\beta} \sum_{i=1} (y_i - \hat{y}_i)^2 + 5000{\beta_3}^2 + 5000{\beta_4}^2$$

L1, L2 Regularization

- L1 (lasso)
  - 중요도가 낮은 가중치부터 0이 됨.
  - 변수 선택의 효과
- L2 (ridge)
  - 아주 큰 가중치에 페널티 부여
  - 가중치들이 전체적으로 평평해짐

L1 Regularization

$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda \sum_{j=0}^{M} |W_j|$$

L2 Regularization

$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda \sum_{j=0}^{M} W_j^2$$

Loss function

Regularization Term

Use Local Image

tqdm 패키지

- loop에서 progress bar를 이용해 loop의 진행상황을 표시해 줍니다.

```
1 !pip install tqdm
```

```
Requirement already satisfied: tqdm in /usr/local/lib/python3.6/dist-packages (4.41.1)
```

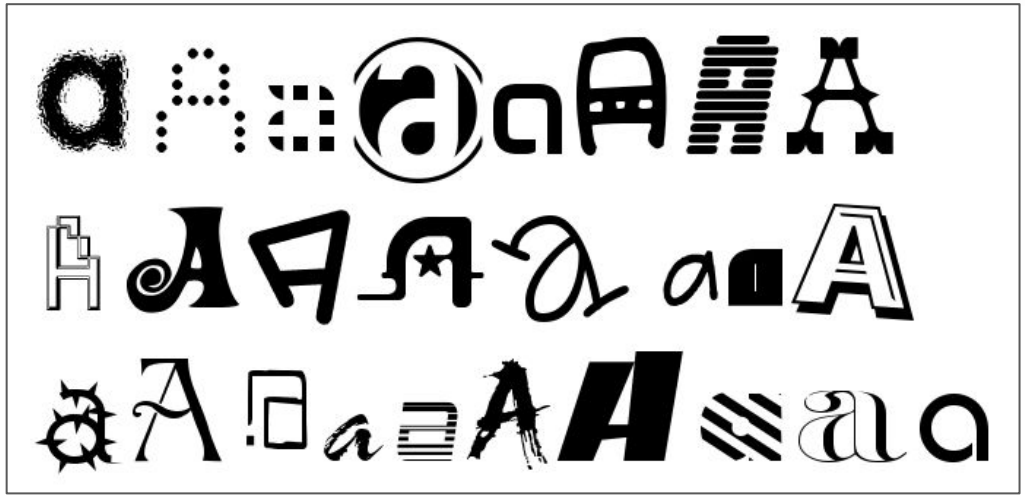
```
1 import tqdm
2 import time
3
4 for i in tqdm.notebook.tqdm(range(10000)):
5     time.sleep(0.001)
```

100%  10000/10000 [00:11<00:00, 889.32it/s]

notMNIST datasets

- http://yaroslavvb.com/upload/notMNIST/notMNIST_large.tar.gz

```
1 !wget http://yaroslavvb.com/upload/notMNIST/notMNIST_large.tar.gz
2 !tar -xvzf notMNIST_large.tar.gz
```



압축 하기

□ 사용법

```
1 | $ tar -cvzf [압축된 파일 이름] [압축할 파일이나 폴더명]
```

압축 풀기

□ 사용법

```
1 | $ tar -xvzf [압축 해제할 압축 아카이브 이름]
```

notMNIST datasets

- 파일목록 선택

```
1 import random
2 import tqdm
3 import glob
4
5 # 파일목록 가져오기
6 paths = glob.glob('./notMNIST_large/*/*.png')
7 print(len(paths))
8 print(paths[:10])
9
10 # 파일 목록 순서 섞어서
11 random.shuffle(paths)
12 # 상위 60000개의 데이터만 사용
13 paths = paths[:60000]
```

notMNIST datasets

- 이미지 로딩하기

```
 1 import numpy as np
 2 import matplotlib.pyplot as plt
 3
 4 # 이미지 데이터셋 로드
 5 x_train, y_train = [], []
 6 for p in tqdm.notebook.tqdm(paths):
 7     x_train.append(plt.imread(paths[0]))
 8     y_train.append(p.split('/')[2])
 9
10 x_train = np.array(x_train)
11 y_train = np.array(y_train)
12 print(x_train.shape, y_train.shape)
```

Data

tf.data.Dataset

- data셋을 미리 구성해주는 도구

```
train_ds = tf.data.Dataset.from_tensor_slices((x_train[:50000], y_train[:50000]))
train_ds = train_ds.shuffle(1000).batch(128)
valid_ds = tf.data.Dataset.from_tensor_slices((x_train[50000:], y_train[50000:]))
valid_ds = valid_ds.batch(128)
```

```
train_x, train_y = next(iter(train_ds))
print(train_x.shape, train_y.shape)
valid_x, valid_y = next(iter(valid_ds))
print(valid_x.shape, valid_y.shape)
```

```
(128, 28, 28, 1) (128,)
(128, 28, 28, 1) (128,)
```

ImageDataGenerator

- 기존 이미지를 변형해서 새로운 이미지를 생성해주는 도구

```
datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True
)
train_ds = datagen.flow(x_train[:50000], y_train[:50000], batch_size=128)

valid_ds = tf.data.Dataset.from_tensor_slices((x_train[50000:], y_train[50000:]))
valid_ds = valid_ds.batch(128)
```

```
train_x, train_y = next(iter(train_ds))
print(train_x.shape, train_y.shape)
valid_x, valid_y = next(iter(valid_ds))
print(valid_x.shape, valid_y.shape)
```

```
(128, 28, 28, 1) (128,)
(128, 28, 28, 1) (128,)
```