WEEK 4

# Standalone Deep-Leaning

- [#12.Lab] Run Pytorch on GPU
- [#13.Lec] Overfitting, Regularization
- [#14.Lec] Hyperparameter Tuning Guide

2021. 08. 03. 여해인

# CPU VS GPU Architecture

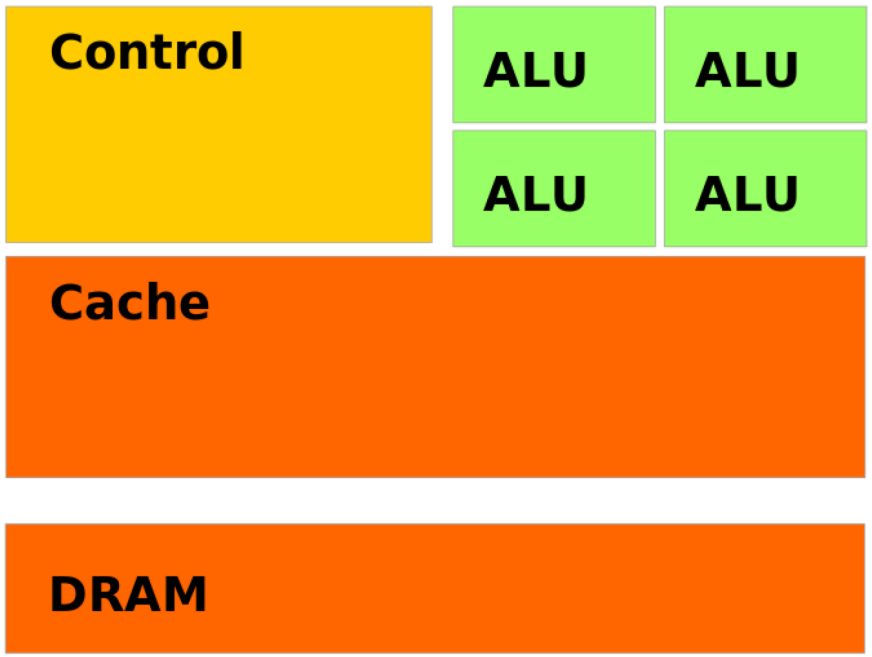
## CPU

Central Processing Unit

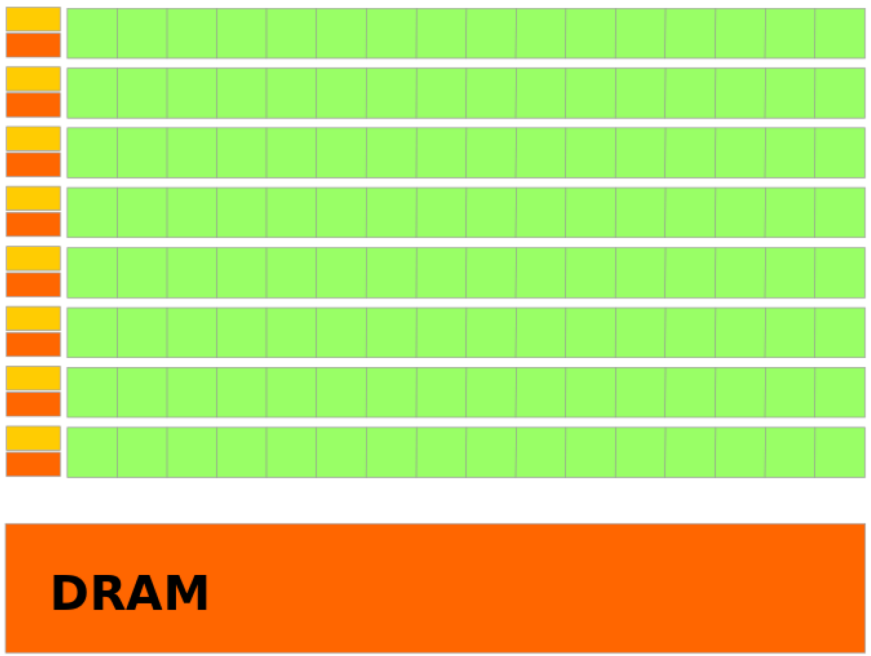
## GPU

Graphics Processing unit

# CPU VS GPU Architecture




CPU



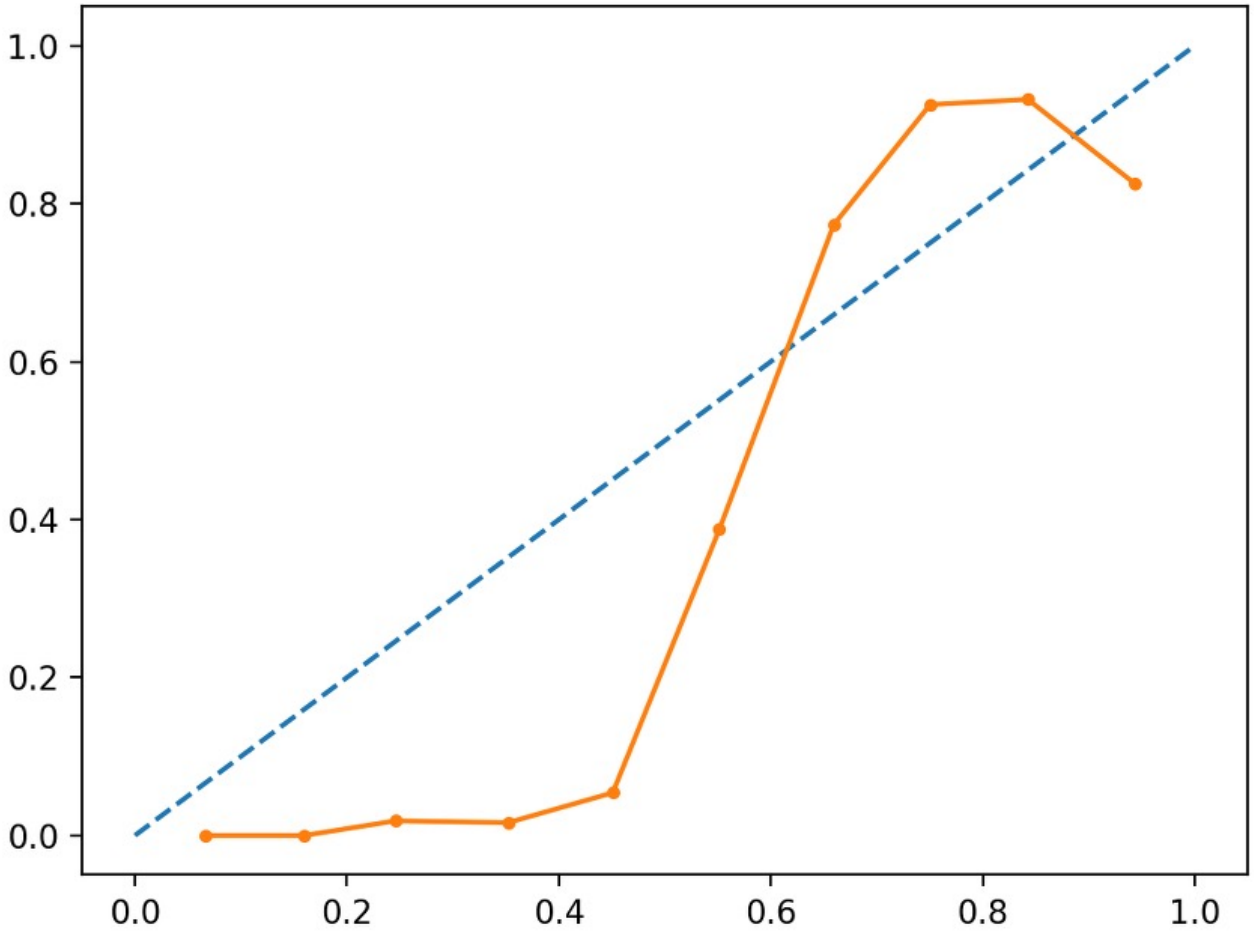

GPU

# How to use GPU with Pytorch?

In [ ]:
```
model = MLPModel(784, 10, [1000])
device = torch.device('cuda:0' if torch.cuda.is_available() else 'gpu')
model.to(device)
```

# Overfitting

# Model Capacity

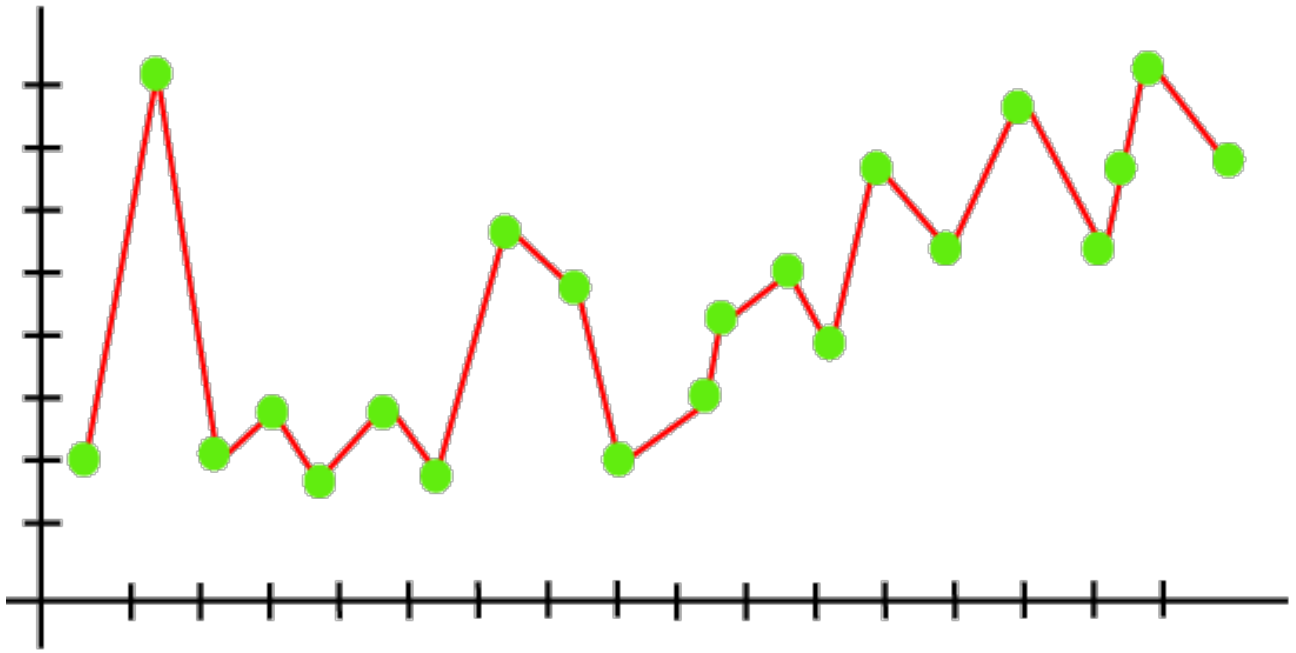


# Model Capacity

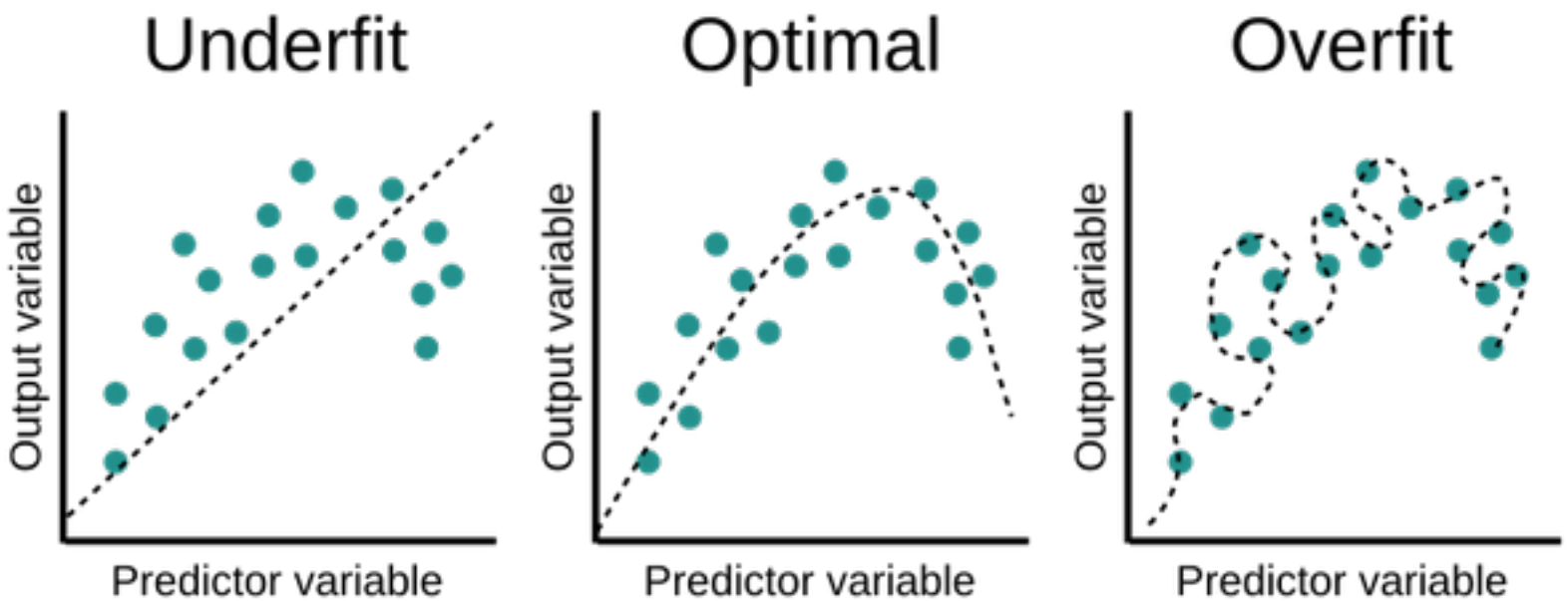
Model Capacity가 높아질 수록 더 복잡한 현상을 예측할 수 있다.

계속 capacity를 늘리면?

# Overfitting



# Overfitting



# Overfitting

## True Risk

- 실제 loss 값
- 모든 데이터 다 구할 수 없음

  ex) 전세계 모든 사람의 얼굴 표정

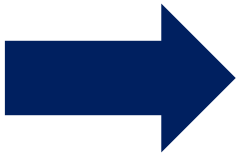
## Empirical Risk

- training set에 대한 loss 값
- 구할 수 있는 데이터

# Overfitting

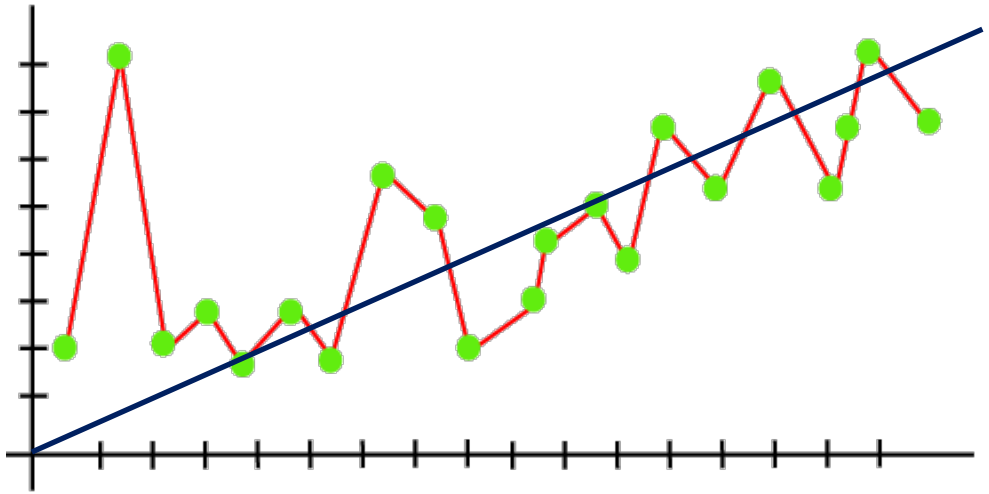
Empirical Risk ⬇️

True Risk ⬆️



**Overfitting**

# Overfitting을 해결하려면?



- ✅ Test set과 Training set의 분리
- ✅ 적절한 capacity 찾기
- ✅ 너무 깊게 학습하지 않도록 Training 중단
- ✅ Regularization

# Regularizations

Regularization

# 가중치 벌칙

$$\underbrace{J_{regularized}(\boldsymbol{\Theta}; \mathbb{X}, \mathbb{Y})}_{\text{규제를 적용한 목적함수}} = \underbrace{J(\boldsymbol{\Theta}; \mathbb{X}, \mathbb{Y})}_{\text{목적함수}} + \lambda \underbrace{R(\boldsymbol{\Theta})}_{\text{규제 항}}$$

- 규제항은 훈련집합과 무관하며, 데이터 생성 과정에 내재한 사전 지식에 해당
- 규제항은 매개변수를 작은 값으로 유지하므로 모델의 용량을 제한하는 역할 (수치적 용량을 제한함)
- 큰 가중치에 벌칙을 가해 작은 가중치를 유지하려고 주로 L2놈이나 L1놈을 사용

# Regularization

## L1 놈

$$\underbrace{J_{regularized}(\boldsymbol{\Theta}; \mathbb{X}, \mathbb{Y})}_{\text{규제를 적용한 목적함수}} = \underbrace{J(\boldsymbol{\Theta}; \mathbb{X}, \mathbb{Y})}_{\text{목적함수}} + \lambda \underbrace{\|\boldsymbol{\Theta}\|_1}_{\text{규제 항}}$$

$$\nabla J_{regularized}(\boldsymbol{\Theta}; \mathbb{X}, \mathbb{Y}) = \nabla J(\boldsymbol{\Theta}; \mathbb{X}, \mathbb{Y}) + \lambda \mathbf{sign}(\boldsymbol{\Theta})$$

# Regularization

## L2 놈

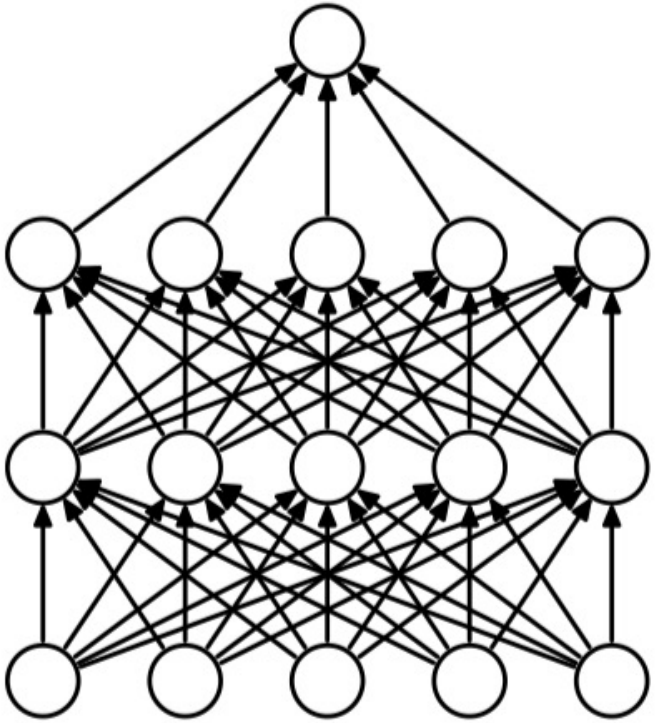
$$\underbrace{J_{regularized}(\boldsymbol{\Theta}; \mathbb{X}, \mathbb{Y})}_{\text{규제를 적용한 목적함수}} = \underbrace{J(\boldsymbol{\Theta}; \mathbb{X}, \mathbb{Y})}_{\text{목적함수}} + \lambda \underbrace{\|\boldsymbol{\Theta}\|_2^2}_{\text{규제 항}}$$

$$\nabla J_{regularized}(\boldsymbol{\Theta}; \mathbb{X}, \mathbb{Y}) = \nabla J(\boldsymbol{\Theta}; \mathbb{X}, \mathbb{Y}) + 2\lambda\boldsymbol{\Theta}$$

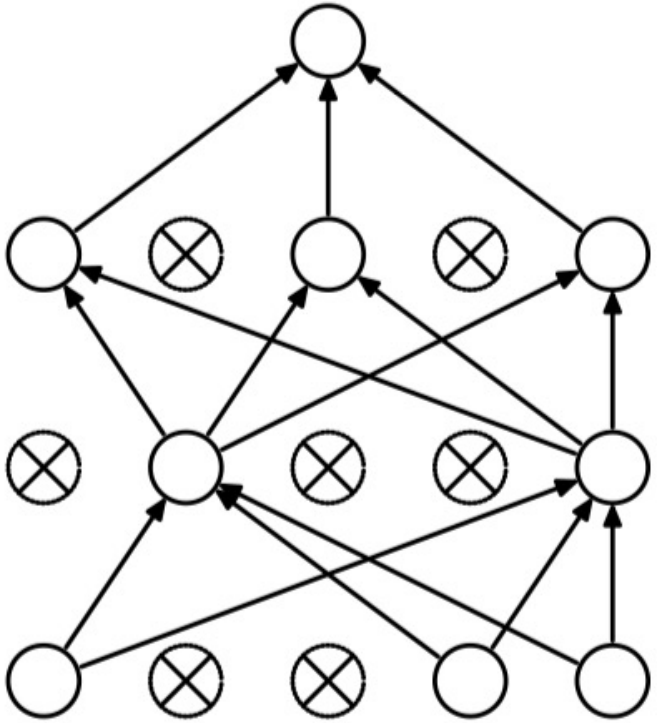
## Regularization

| Ridge | Lasso |
| --- | --- |
| L2-norm regularization | L1-norm regularization |
| 변수 선택 불가능 | 변수 선택 가능 |

# Dropout



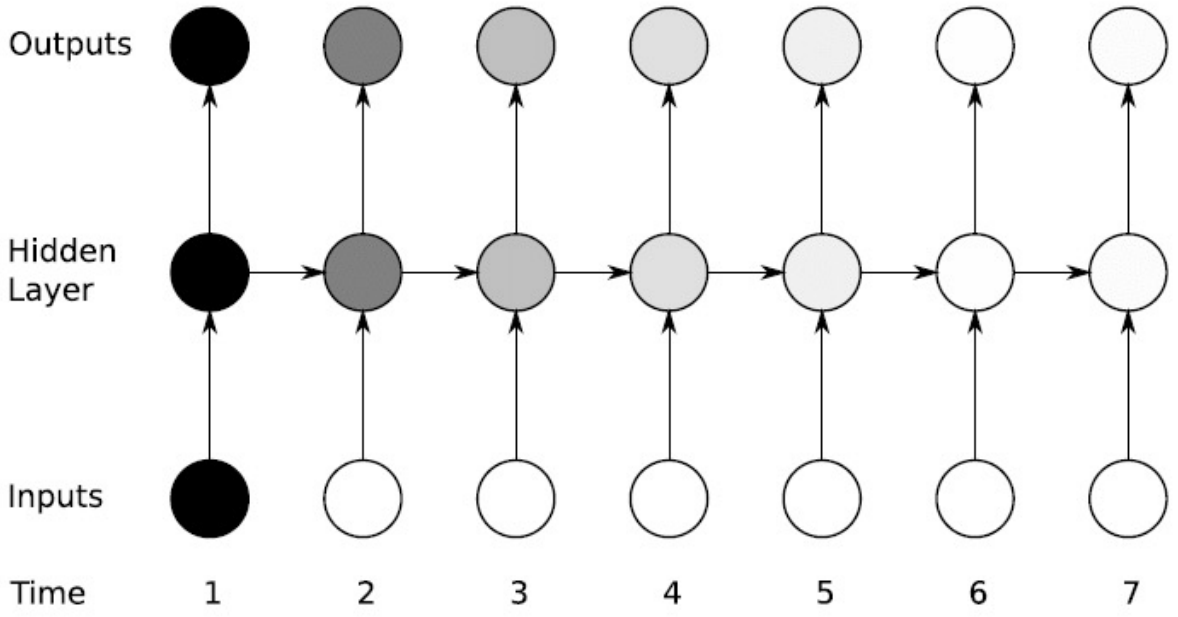
(a) Standard Neural Net



(b) After applying dropout.

# Gradient Vanishing

## 기울기 소실 문제

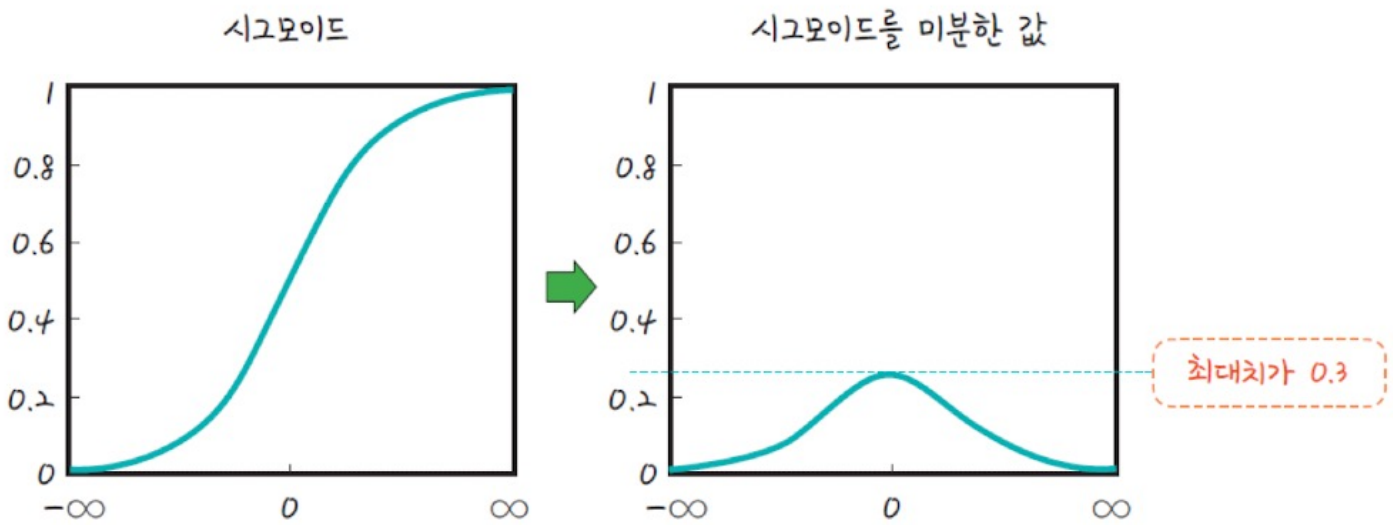



- 오차 역전파는 출력층으로부터 하나씩 앞으로 되돌아가며 각 층의 가중치를 수정하는 방법
- 가중치를 수정하려면 미분 값, 즉 기울기가 필요함
- 여러 층을 전파할수록 기울기가 점점 사라져 가중치를 수정하기가 어려운 현상

Gradient Vanishing

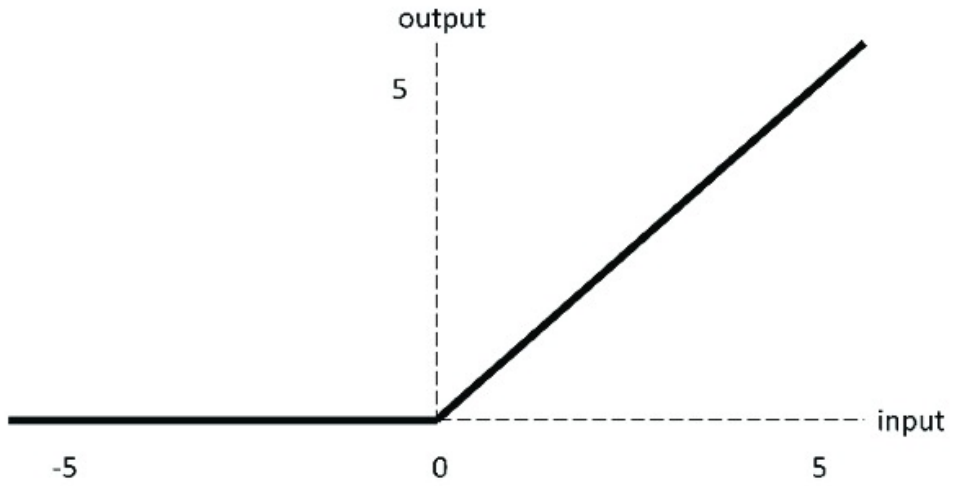
# 기울기 소실 문제의 원인

활성화 함수로 사용된 시그모이드 함수의 특성 때문
0.3 이하의 값을 반복하여 곱하는 상쇄 효과가 발생




[시그모이드 함수의 미분]

# ReLU

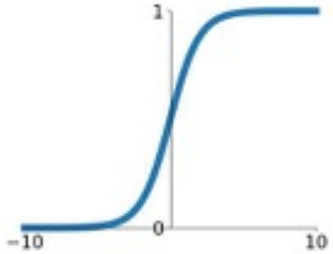



- x가 0보다 작으면 모든 값을 0으로, 0보다 크면 x를 그대로
- 현재 가장 많이 사용되는 활성화 함수
- x가 0보다 크기만 하면 미분값이 1이 됨
- 여러 은닉층을 거쳐 곱해지더라도 맨 처음 층까지 사라지지않고 남아있을 수 있음

# Activation Functions

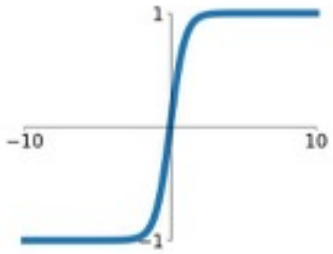
**Sigmoid**

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



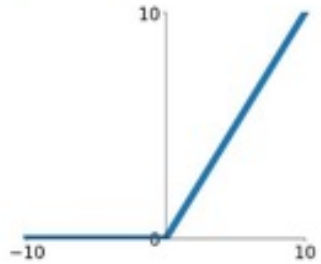

**tanh**

$$\tanh(x)$$



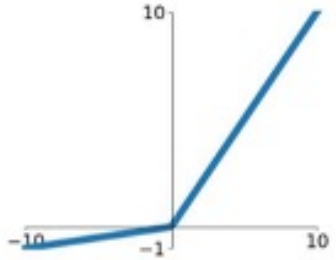

**ReLU**

$$\max(0, x)$$




**Leaky ReLU**

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

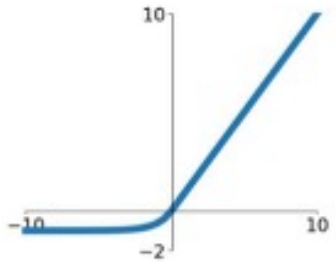



**Maxout**

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

**ELU**

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



# Hyperparameter Tuning

# Hyperparameter

최적의 딥러닝 모델 구현을 위해 학습률이나 배치크기, 훈련 반복 횟수, 가중치 초기화 방법 등

인간의 선험적 지식을 기반으로 딥러닝 모델에 설정하는 변수

# Purpose of Hyperparameter Tuning?

⬇️

Increase Model Performance

⬇️

Reduce True Risk (Generalization Error) of Model

⬇️

Reduce True Risk on Validation Set, approximately

# Hyperparameter Tuning

## Model Realated

- Number of hidden layer
- Number of hidden unit
- Activation Function

## Optimization Related

- Type of Optimizer
- Learning rate
- L2 coef
- Dropout Rate
- Batch Size
- Epoch

# Hyperparameter Tuning

```
# 컨볼루션 신경망의 설정
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), input_shape=(28, 28, 1), activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128,  activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

# Hyperparameter Tuning

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

# 모델 최적화 설정
MODEL_DIR = './model/'
if not os.path.exists(MODEL_DIR):
    os.mkdir(MODEL_DIR)

modelpath="./model/{epoch:02d}-{val_loss:.4f}.hdf5"
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_loss', verbose=1, save_best_only=True)
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=10)

# 모델의 실행
history = model.fit(X_train, Y_train, validation_data=(X_test, Y_test),
                    epochs=30, batch_size=200, verbose=0, callbacks=[early_stopping_callback,checkpointer])
```