
START
머신러닝과 딥러닝
Machine Learning deep Learning

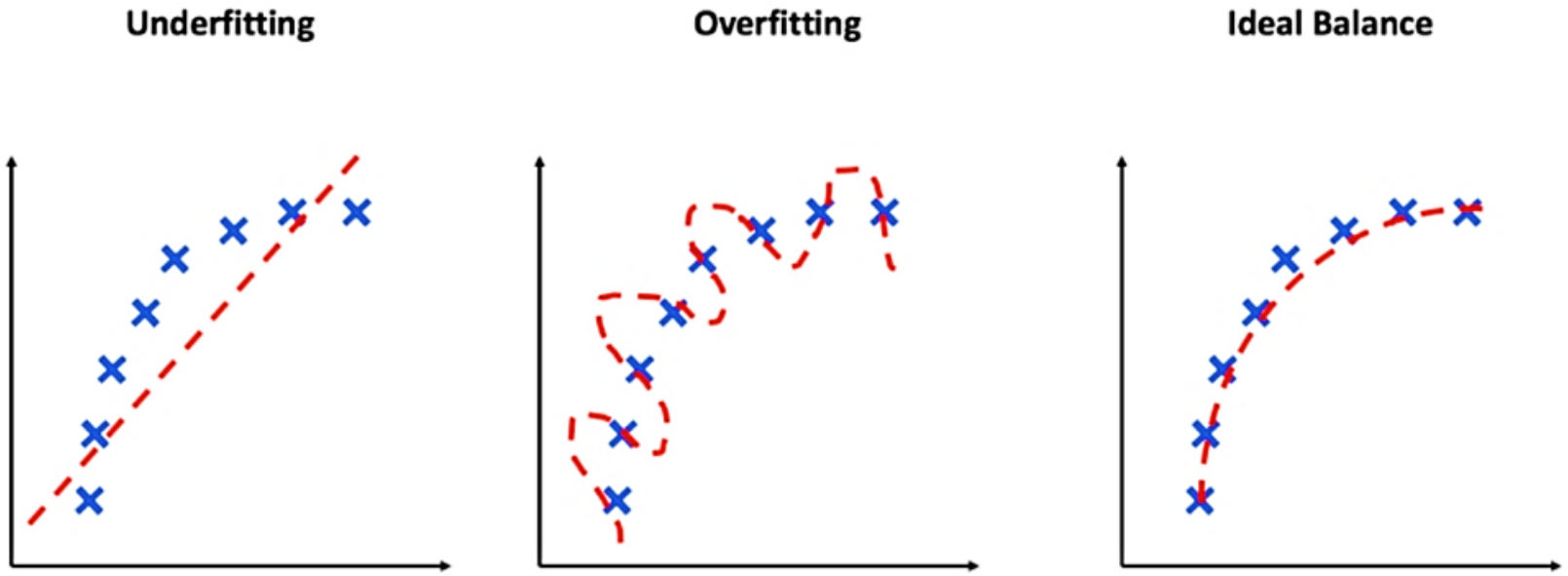
Chapter 11. 딥러닝 가이드라인

Machine Learning & Deep Learning

손영두

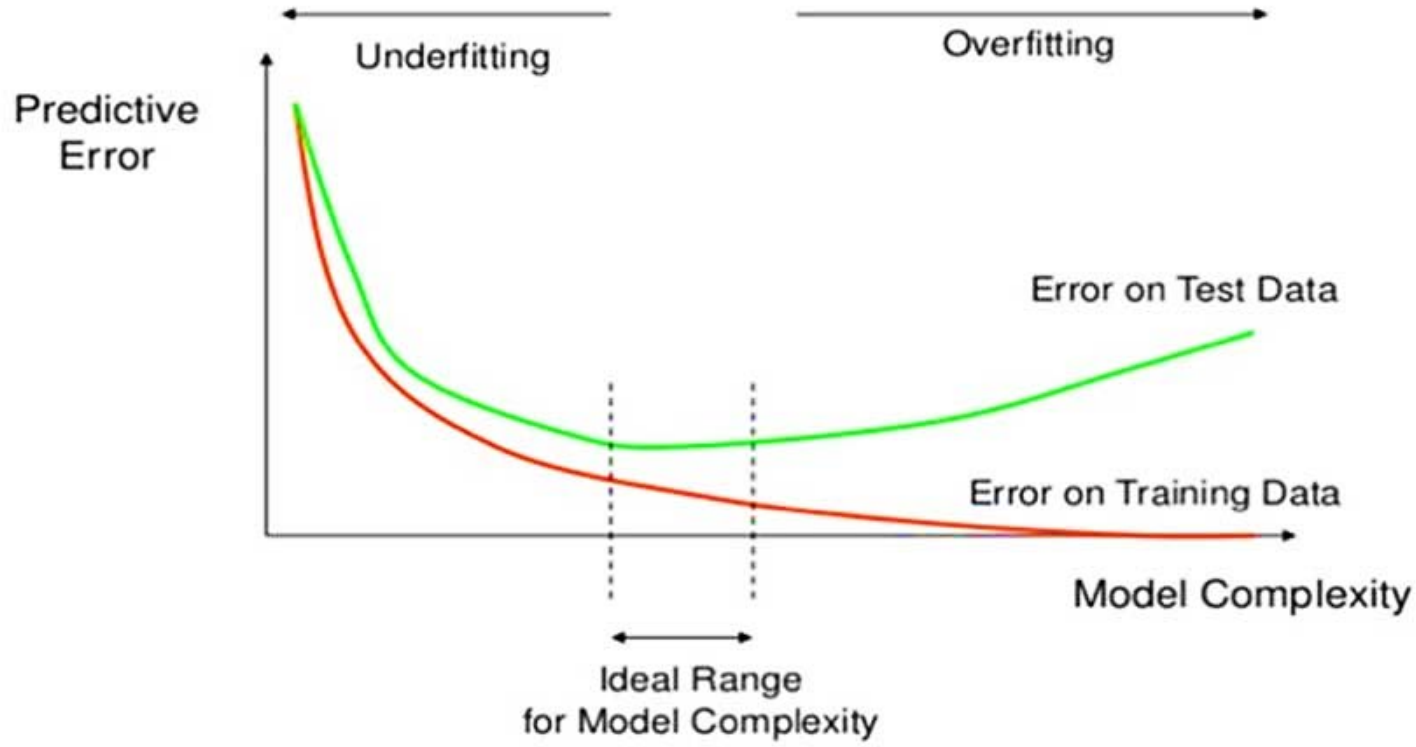
e-mail: youngdoo@dongguk.edu

Recall: Underfitting and Overfitting



- 머신러닝 알고리즘이 train data에 충분히 적합하지 못하면 패턴을 잘 설명하지 못하는 underfitting (과소적합) 문제가 발생
- 머신러닝 알고리즘의 학습에서 train data에만 너무 적합하게 학습되면, generalization performance가 감소하는 overfitting (과적합) 문제가 발생

Recall: Underfitting and Overfitting



☑ 머신러닝 알고리즘을 학습할 때에는, (1) training error, (2) generalization gap (training error와 generalization error의 차이), 두 가지를 최소화하여야 함

Overfitting in Deep Learning

- 모델의 복잡도가 큰 딥러닝에서는 overfitting 문제가 발생하기 쉬움
- 따라서 이를 해결하기 위한 많은 방법들이 regularization 연구됨
  - Parameter norm penalty
  - Data augmentation
  - Multitask learning
  - Early stopping
  - Dropout

Regularization

- Regularization(정규화, 정칙화)는 학습하고자하는 파라미터의 값에 제약을 주어 모델의 복잡도를 낮추는 방법
- 처음부터 적당한 복잡도의 모델을 선택하는 대신, 복잡도가 높은 모델의 파라미터에 적절한 제약을 주는 방식으로 최적적합 모형을 탐색
- Parameter Norm Penalty
  - 이러한 regularization 방법 중에는 학습하고자하는 parameter들의 norm penalty를 기존의 cost function에 더하여 새로운 objective function을 만드는 방법이 가장 널리 사용됨

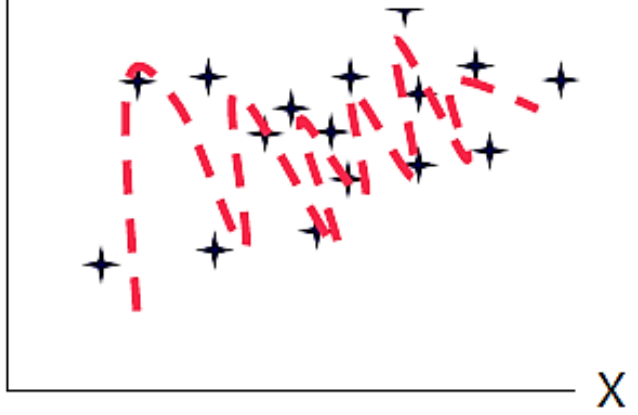
$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

Parameter Norm Penalty

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

$\lambda = 0.$

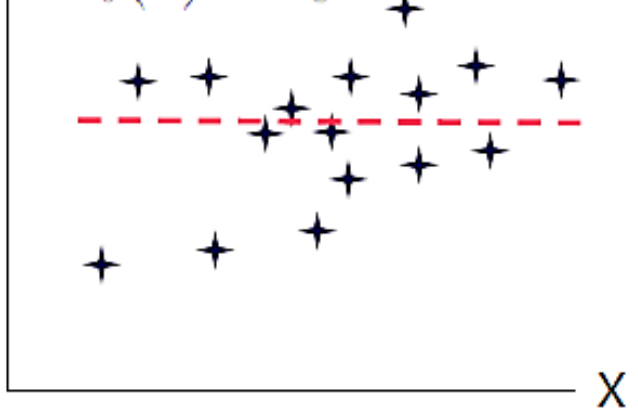
$h_\theta(x) = \theta_0 + \theta_1 x^1 + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \ldots$



Regularization을 안 하는 것과 일치

$\lambda \to \infty$

$h_\theta(x) = \theta_0$



Regularization을 최대치까지 활용

Parameter Norm Penalty

Parameter Norm Penalty in Deep Learning

- 일반적으로 신경망에서는 node 간의 연결의 가중치에만 penalty를 부여하고 bias는 penalty에 포함시키지 않음
- Bias를 penalty에 포함시킬 경우 underfitting이 심각하게 발생할 가능성이 높음
- 신경망에서 층별로 서로 다른 종류의 norm penalty를 적용하는 방식 또한 있으나, hyperparameter 선택에 관한 비용이 증가하기 때문에 일반적으로는 모든 층에 같은 종류의 penalty를 적용함

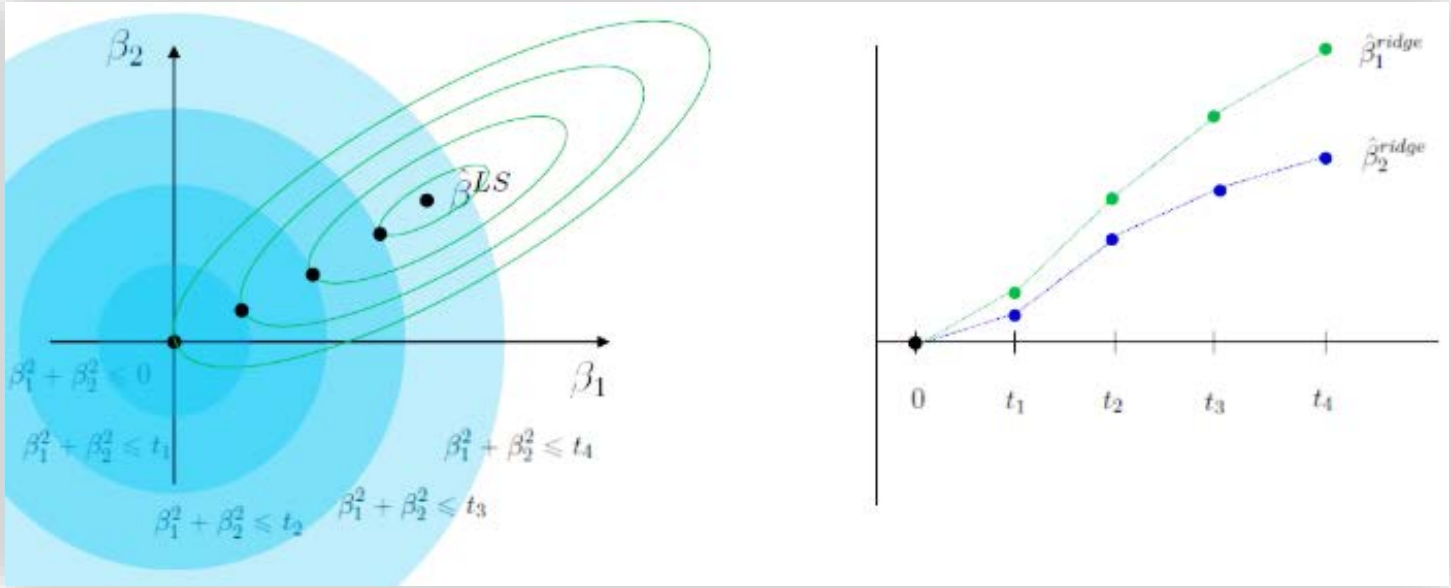
Parameter Norm Penalty

L_2-norm Regularization

$$\text{regularization term } \Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \frac{\alpha}{2}\boldsymbol{w}^\top \boldsymbol{w} + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

- Ridge-style norm penalty
- Weight-decay (shrinkage)



Parameter Norm Penalty

L_1-norm Regularization

$$\text{regularization term} \quad \Omega(\boldsymbol{\theta}) = ||\boldsymbol{w}||_1 = \sum_i |w_i|$$

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha||\boldsymbol{w}||_1 + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}),$$

- Lasso-style norm penalty
- Parameter selection (sparsity)

Parameter Norm Penalty

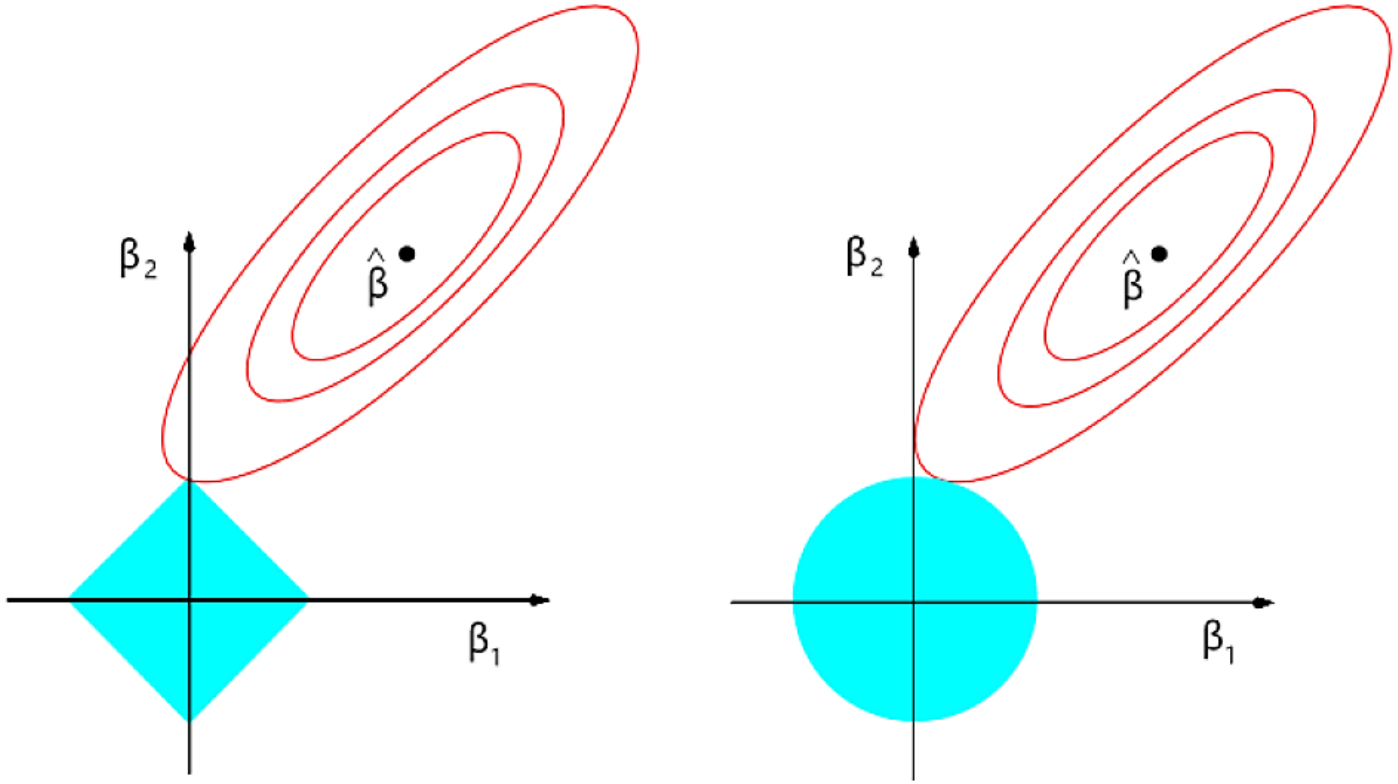
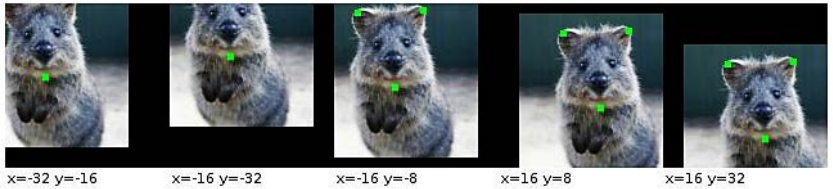


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \leq t$ *and* $\beta_1^2 + \beta_2^2 \leq t^2$*, respectively, while the red ellipses are the contours of the least squares error function.*

Data Augmentation

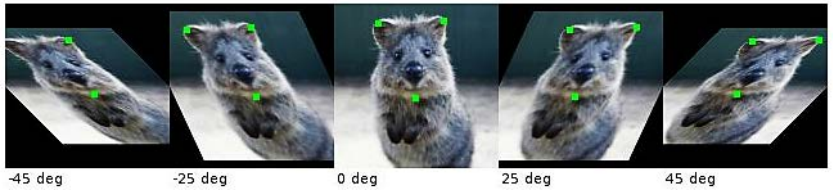
- 기계학습 방법론의 학습에서 model의 generalization performance를 개선하는 가장 좋은 방법 중 하나는 (아마도) 적절한 데이터를 추가하는 것
- 그러나 실제로 사용할 수 있는 데이터의 수는 한정적임
- Data Augmentation
  - 기존의 training data를 조작하여 새로운 (fake) training data를 생성
  - 예: 데이터의 random noise를 추가하여 학습하여 robust model을 구축
  - 예: image data의 경우

Affine: Translate



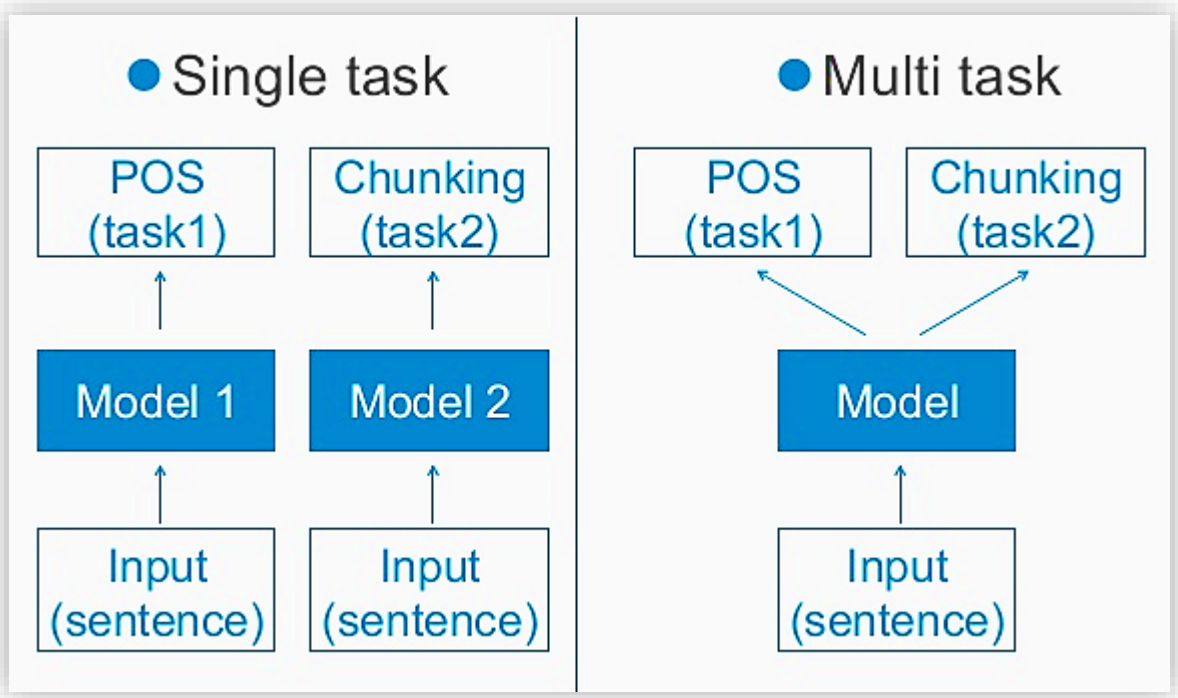
Affine: Rotate

Affine: Shear



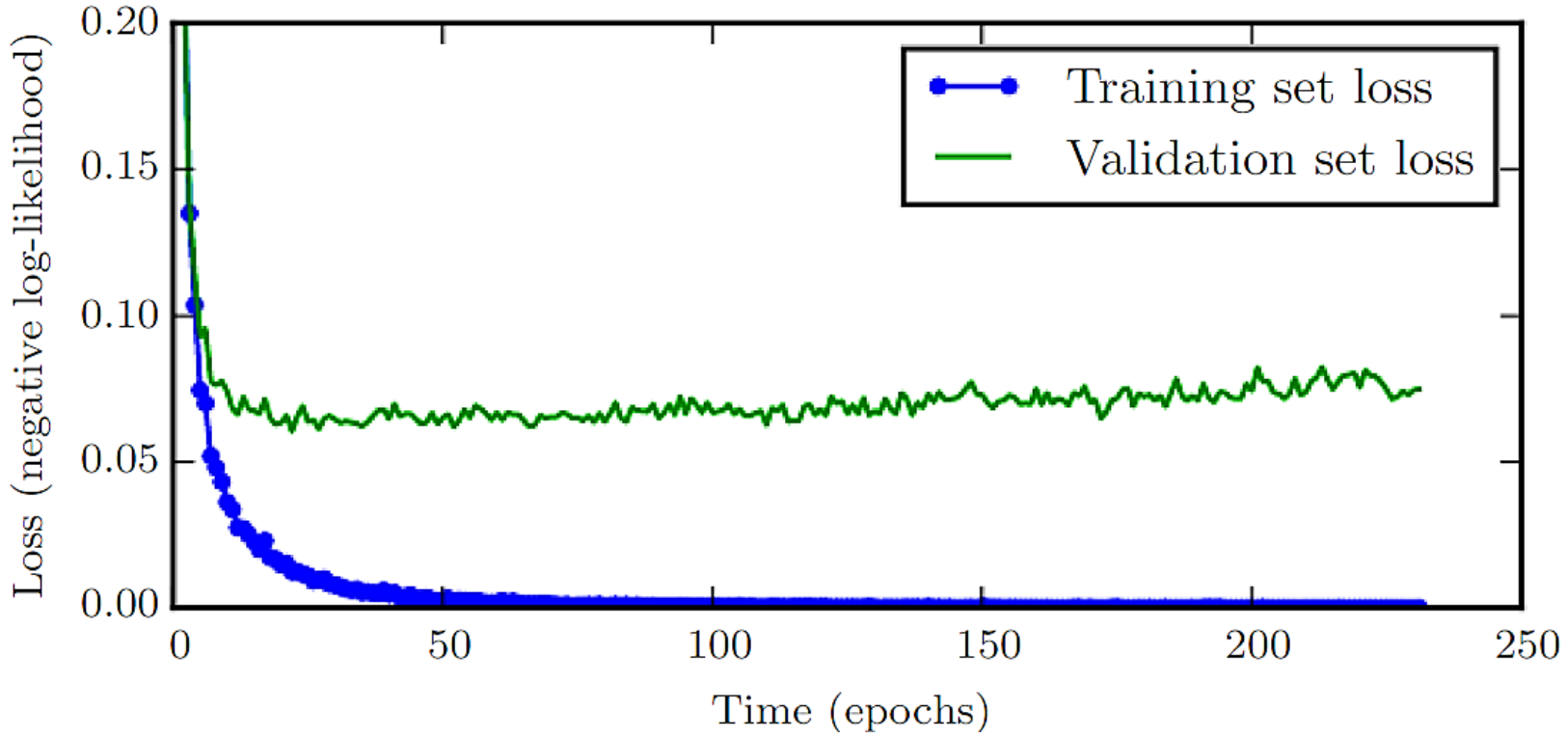
Multitask Learning

- Multitask learning은 같은 입력을 사용하는 여러 task들을 동시에 수행하는 방법론을 의미
- 일반적으로 모형이 여러 task를 동시에 수행하게 될 경우, 서로 연관되거나 공유하고 있는 부분이 하나의 task에 적합되지 않고 여러 task에 고르게 잘 작동하려고 학습되기 때문에 regularization 효과가 나타남



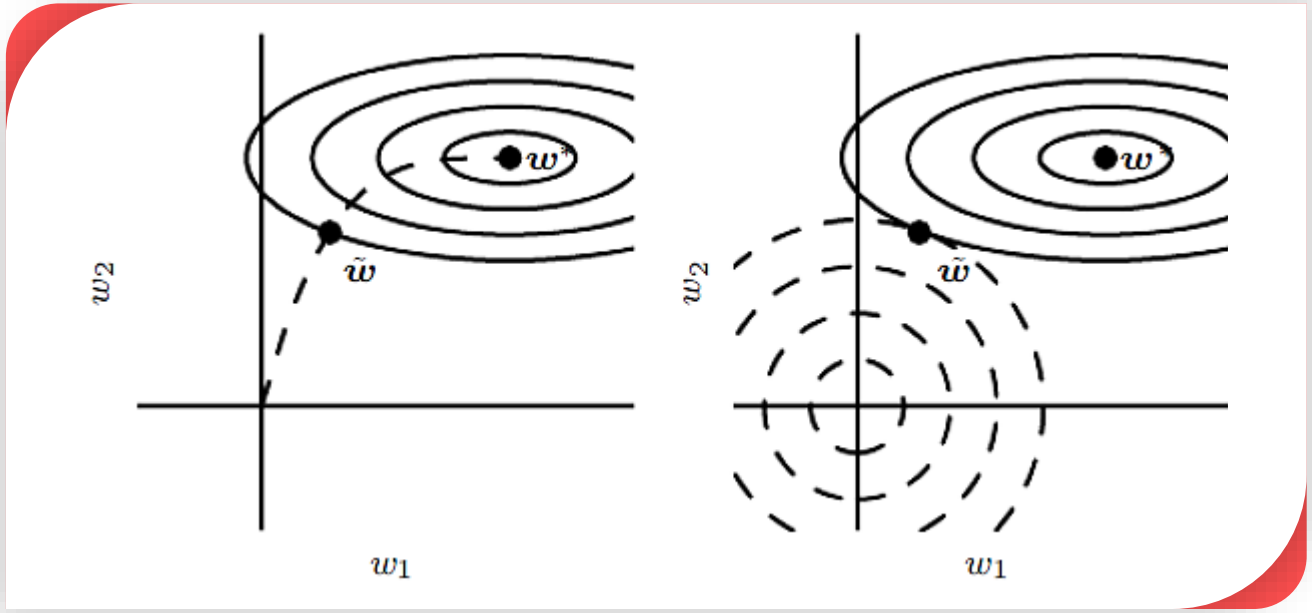
Early Stopping

- 신경망의 경우 training epoch이 길어질수록 training data에 과적합되는 현상이 강함
- 이를 막기 위하여 별도의 validation set을 사용, validation set의 error가 감소하지 않는 경우 학습을 중단



Early Stopping

- 구체적으로 validation loss가 개선될 때마다 해당 지점에서의 모수들을 기억하고, validation loss가 개선되지 않으면 최저의 validation loss를 나타내었던 모수들을 호출하여 사용
- 가장 간단하며 기본적인 regularization 전략으로 널리 사용됨
- Early stopping은 다른 regularization 방법과의 결합도 손쉬움
- Early stopping은 모수의 탐색 공간을 초기값 주위의 작은 공간으로 한정하여 norm penalty와 비슷한 효과를 낼 수 있음

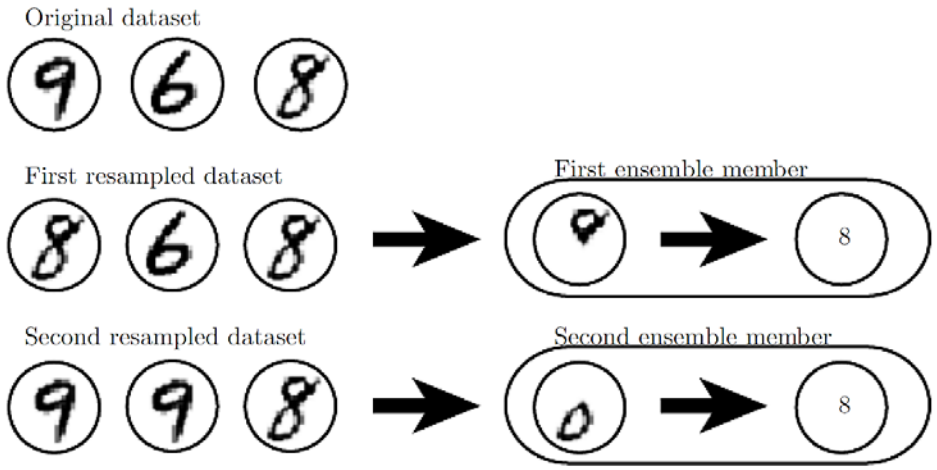


Ensemble

앙상블 방법 또한 overfitting을 막는 대표적인 방법으로 딥러닝에도 적용이 가능

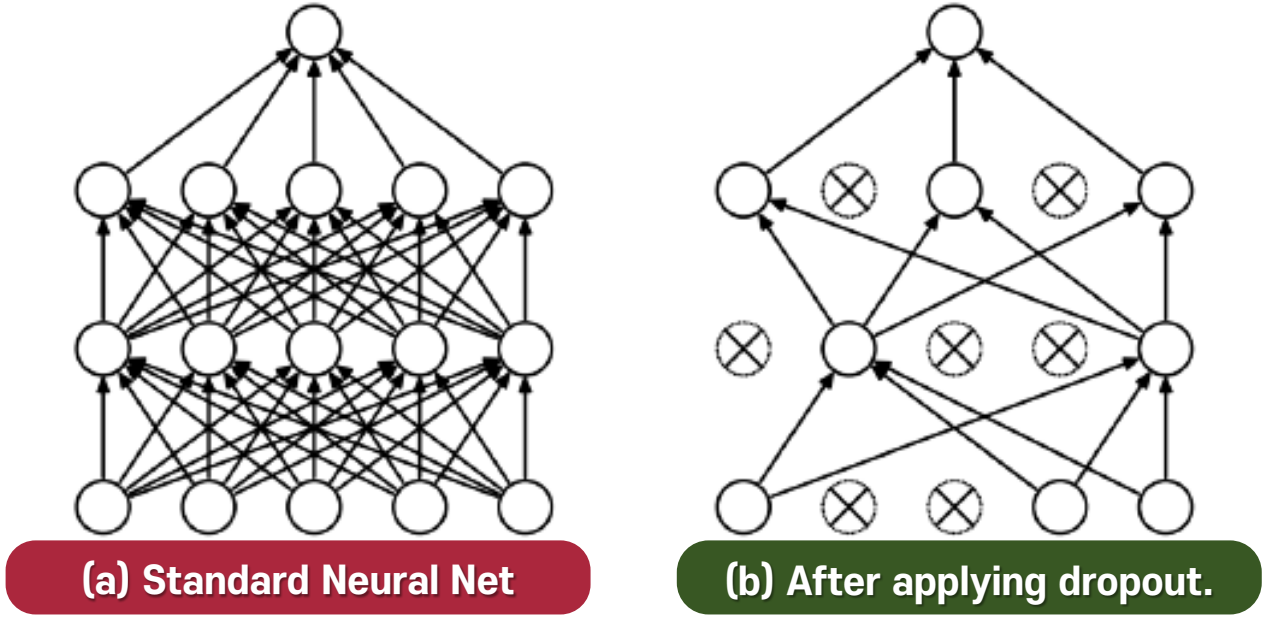
$$\mathbb{E}\left[\left(\frac{1}{k}\sum_i \epsilon_i\right)^2\right] = \frac{1}{k^2}\mathbb{E}\left[\sum_i\left(\epsilon_i^2 + \sum_{j\neq i}\epsilon_i\epsilon_j\right)\right]$$
$$= \frac{1}{k}v + \frac{k-1}{k}c.$$

- 오차들의 상관관계가 작을수록 적은 기대오차 제곱값을 가짐
- Bagging의 경우 데이터셋에 어떤 데이터들이 포함 되었느냐에 따라 모형 학습의 결과가 달라짐



Dropout

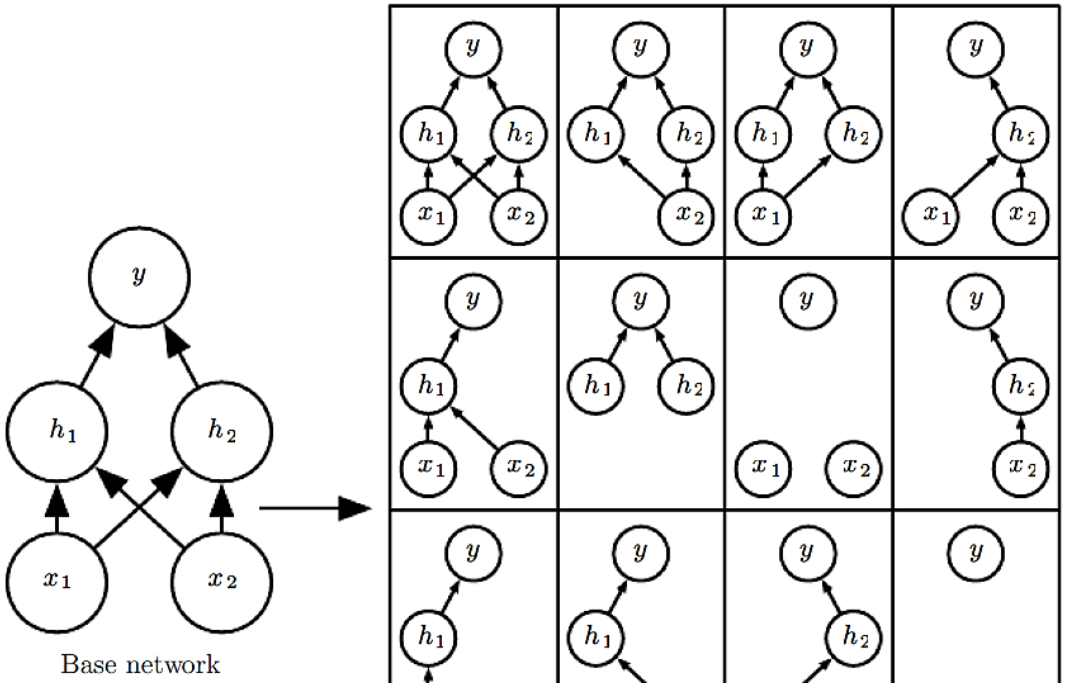
- 가장 성공적인 딥러닝 regularization 방법 중 하나
- 여러 network 들의 ensemble을 현실적으로 구현하는 방법
- Output node가 아닌 다른 node들을 제거하여 만들 수 있는 subnetwork들로 구성된 ensemble model이 학습되도록 조절
- Node들을 일정확률로 선택/또는 선택하지 않고 weight를 학습하는 방식으로 수행



(a) Standard Neural Net

(b) After applying dropout.

Dropout

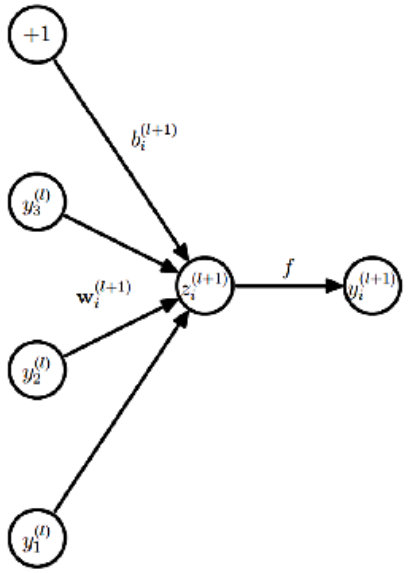


Base network

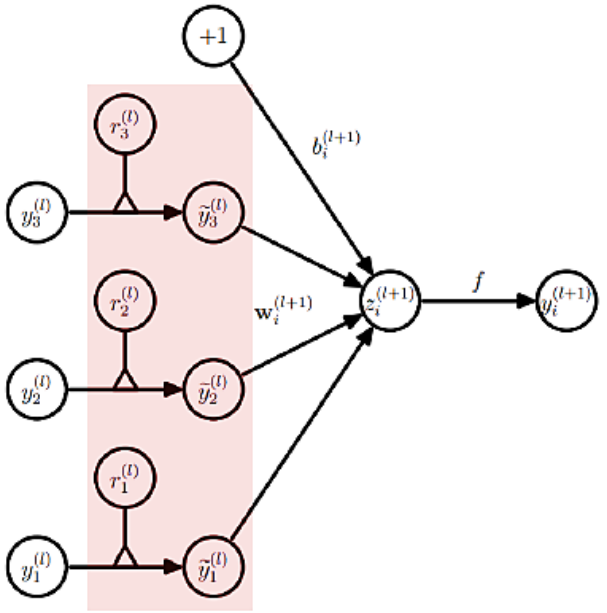
Ensemble of subnetworks

Dropout

Training phase : 각 iteration마다, hidden node를 일정 확률 p를 이용하여 보존 (1-p의 확률로 제거)



(a) Standard network



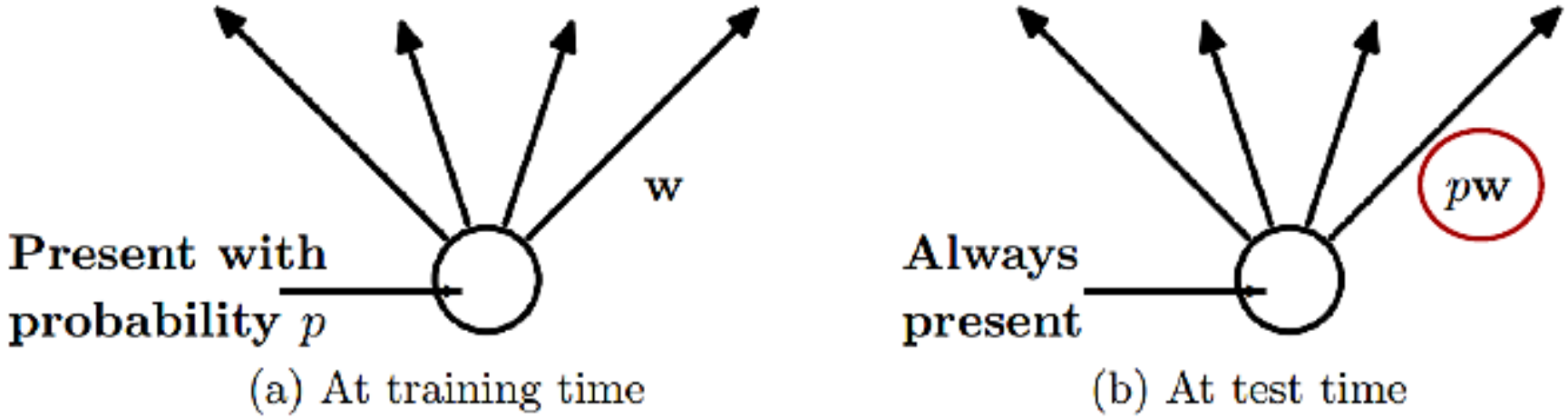
(b) Dropout network

$$
\begin{aligned}
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)}\mathbf{y}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}),
\end{aligned}
$$

$$
\begin{aligned}
r_j^{(l)} &\sim \text{Bernoulli}(p), \\
\tilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)}, \\
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)}\tilde{\mathbf{y}}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}).
\end{aligned}
$$

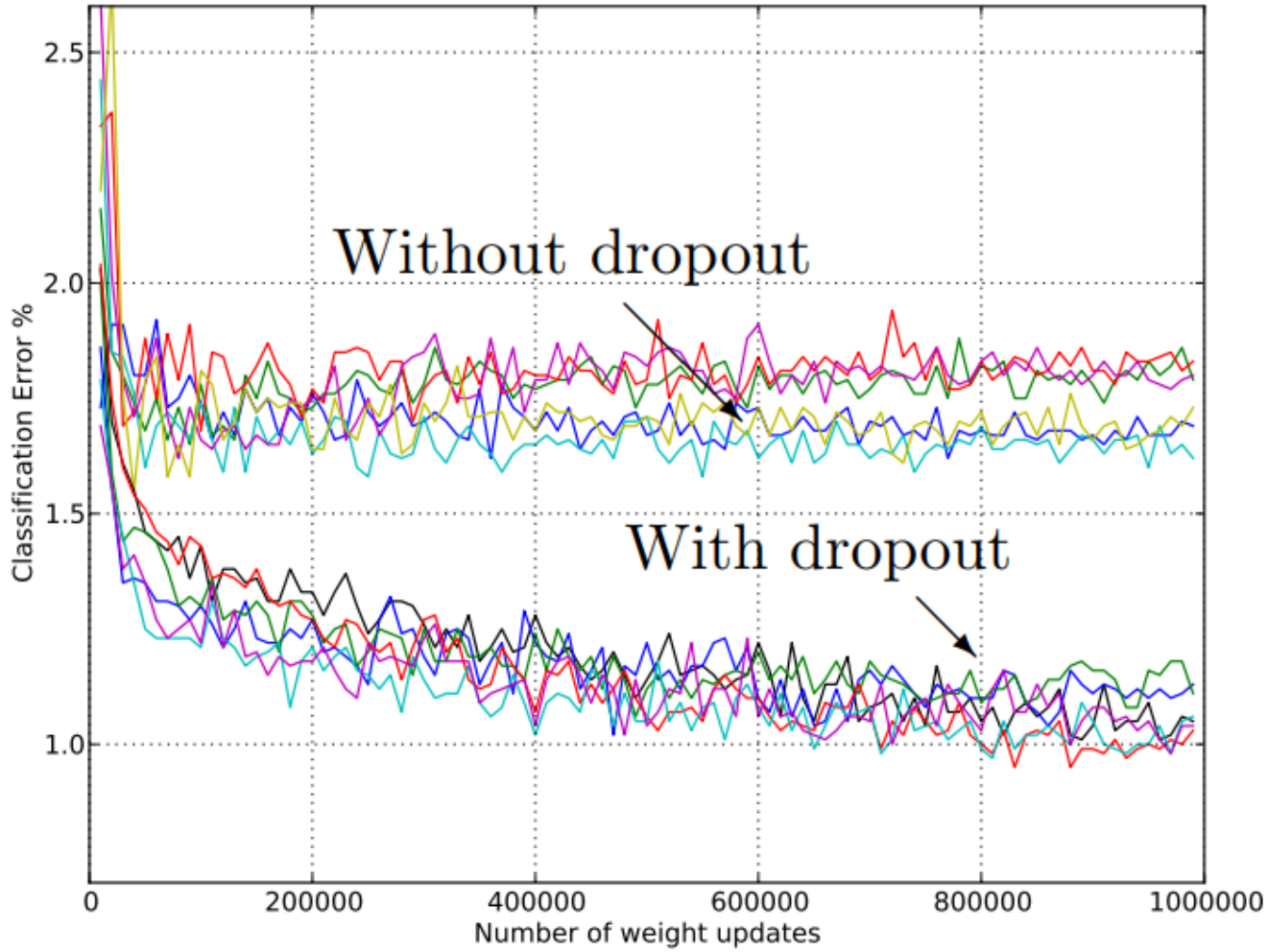
Dropout

Test phase : 각 weight에는 p만큼 보정됨



(a) At training time (b) At test time

Dropout


Without dropout
With dropout
Classification Error %
2.5
2.0
1.5
1.0
0
200000
400000
600000
800000
1000000
Number of weight updates

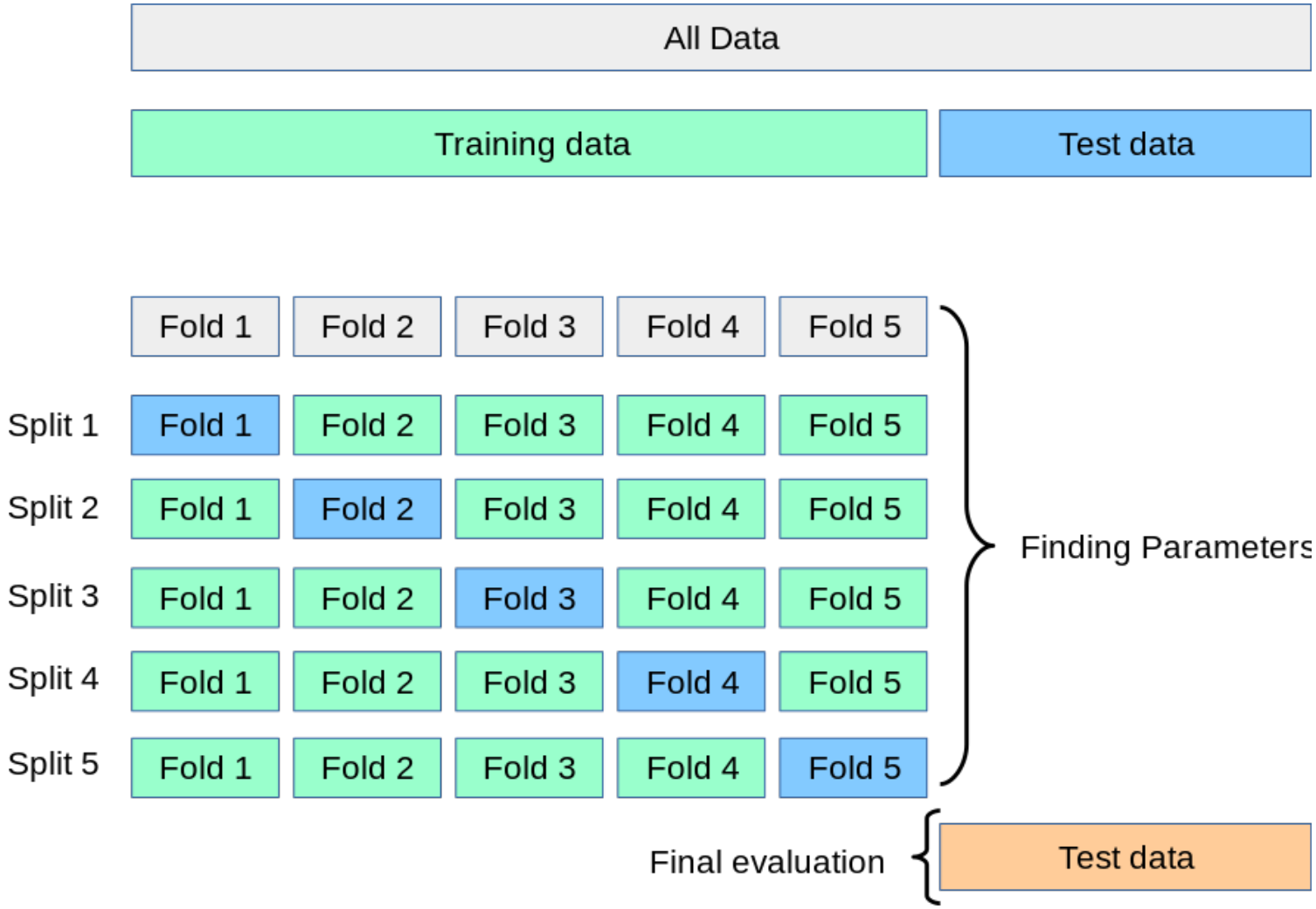
Hyperparameter

- **Hyperparameter (초매개변수) :** 학습 과정에서 변경되지 않고, 모델의 학습을 위하여 미리 결정해야하는 값들
- 최종 학습이 완료된 모델의 성능을 결정
- **예:** 신경망의 구조 (깊이, 넓이), step size, regularization hyperparameter 등
- 이러한 hyperparameter의 selection을 위하여 validation set을 이용
- **예:** underfitting과 overfitting을 막기 위한 신경망의 구조
- Logistic regression, SVM은 hyperparameter가 적은 모델들
- 데이터의 수가 적어 validation set을 따로 나누기 어려운 경우에는 k-fold cross validation 방법이 널리 사용됨

K-fold cross validation

- 데이터를 k개의 서로 겹치지 않는 부분집합으로 분할
- 이 중 k-1개의 부분집합을 training, 나머지 1개의 부분집합을 test set으로 이용하여 오차를 추정
- 이를 서로 각기 다른 조합에 대하여 k번 반복하여 평균 오차를 확인
- k=n : jack-knife CV

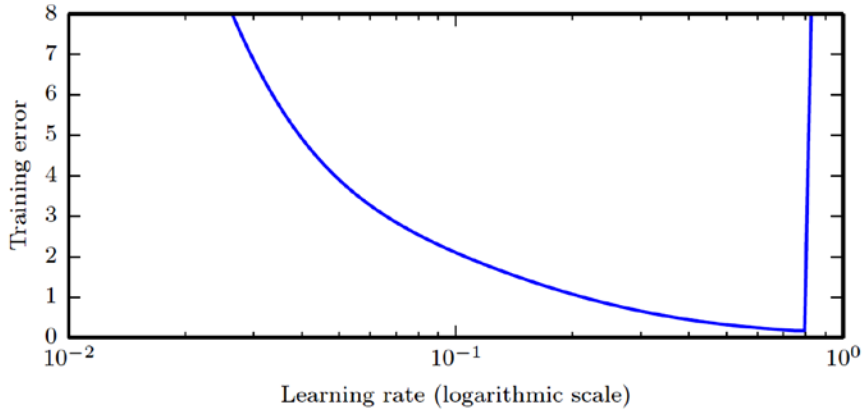
K-fold cross validation

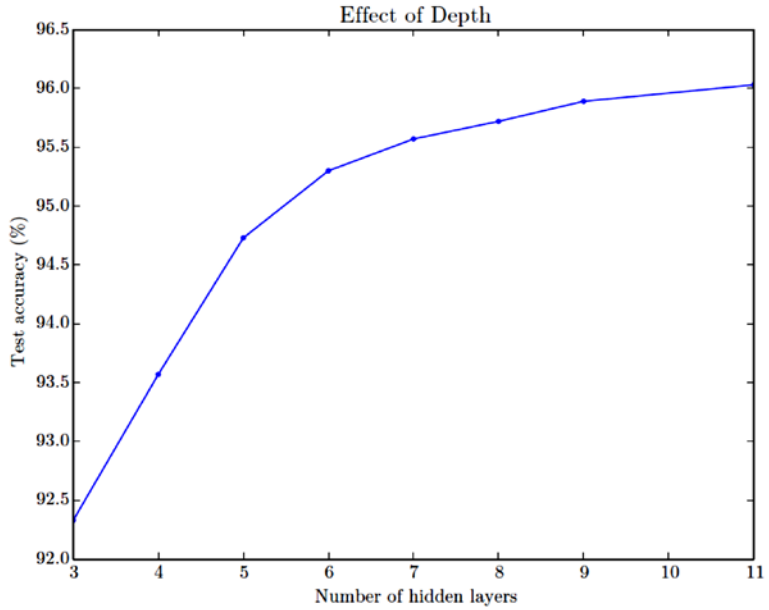


Hyperparameters in Deep Learning

딥러닝에서는 일반적으로 다음과 같은 hyperparameter들을 조절

- 은닉층의 수와 포함된 node의 수: 은닉층의 수와 node의 수가 증가할수록 모델의 복잡도가 증가
- Step size: 너무 크거나 작은 step size를 사용할 경우 최적해를 도출하는 데에 실패할 수 있음
- Regularization coefficient: 정규화를 위한 계수를 적절히 선택하여 모델의 복잡도를 조절 가능
- Dropout rate: dropout 시 각 node가 학습에 사용될 확률을 조절





Hyperparameter Tuning

- Hyperparameter 및 model에 대한 깊은 이해가 있는 경우 이러한 hyperparameter들을 수동으로 결정 가능

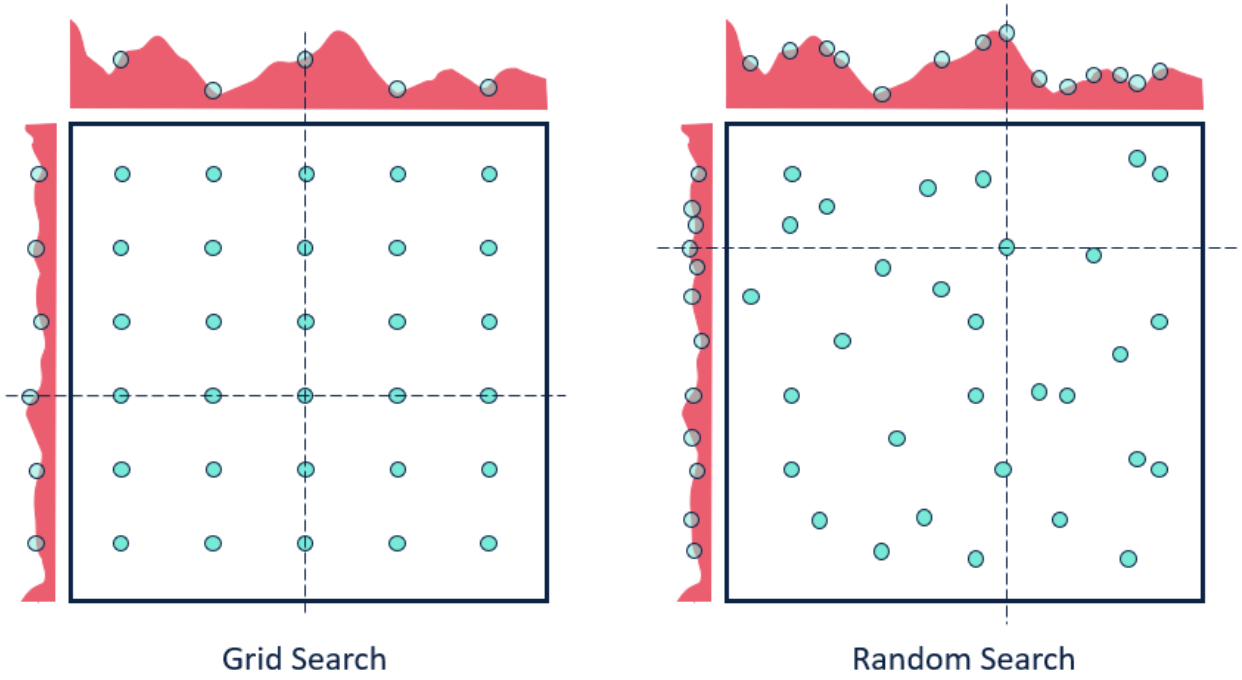
- 그러나 일반적으로 자동화된 규칙을 통하여 최적 hyperparmeter를 탐색

- Grid search
  - Hyperparameter들 마다 적절한 값을 선택하고,
    이들의 Catersian product (grid point)에 대하여 validation loss를 탐색
  - 탐색 하고자하는 hyperparameter의 수가 많아질수록 탐색의 수는 지수적으로 증가
  - 각 trial은 독립적이므로 쉽게 병렬화가 가능

Hyperparameter Tuning

Random search

- Hyperparameter space의 임의의 점에 대하여 random search
- 때때로 grid search보다 효과적임
- 각 trial은 독립적이므로 쉽게 병렬화가 가능



Hyperparameter Tuning

Model-based optimization

- Hyperparameter selection을 하나의 optimization 문제로 바라볼 수 있음
  Objective function : validation error
  Decision variable : hyperparameters

- 일반적으로 위의 objective function은 미분이 불가능함

- 따라서 validation error에 expectation 대한 모델을 각 hyperparmaeter에 대한 함수로 만들어 최적화를 수행하는 방법을 널리 사용 (Bayesian linear regression model 사용)

- 일반적으로 시간이 오래 걸리고 성과가 좋지 않아 널리 사용되지는 않음

Model Selection

수많은 머신러닝/딥러닝 알고리즘들 중 내 데이터를 분석하기 위하여 어떤 알고리즘을 사용 해야할까?

문제의 복잡도에 따라 모델을 결정

- 문제가 복잡하지 않다면 로지스틱 회귀 등 단순한 통계 기반의 머신러닝 모형을 사용
- 물체 인식, 음성 인식, 기계 번역 등 복잡한 AI task인 경우 적절한 딥러닝 모형으로부터 시작

딥러닝 모형을 사용시 가이드라인

- 기본적인 MLP, CNN, LSTM 등의 모델로부터 시작하여 점점 복잡한 방법론들을 적용
- Regularization은 처음부터 사용하는 것을 추천
- Early stopping은 거의 모든 경우 하는 것이 좋으며, dropout 또한 많은 경우에 추천
- 기존 연구에서 좋은 알고리즘이 존재하면 이를 이용하여 시작 (transfer learning)

Model Selection

모수의 초기 값: 대칭성의 파괴가 필요

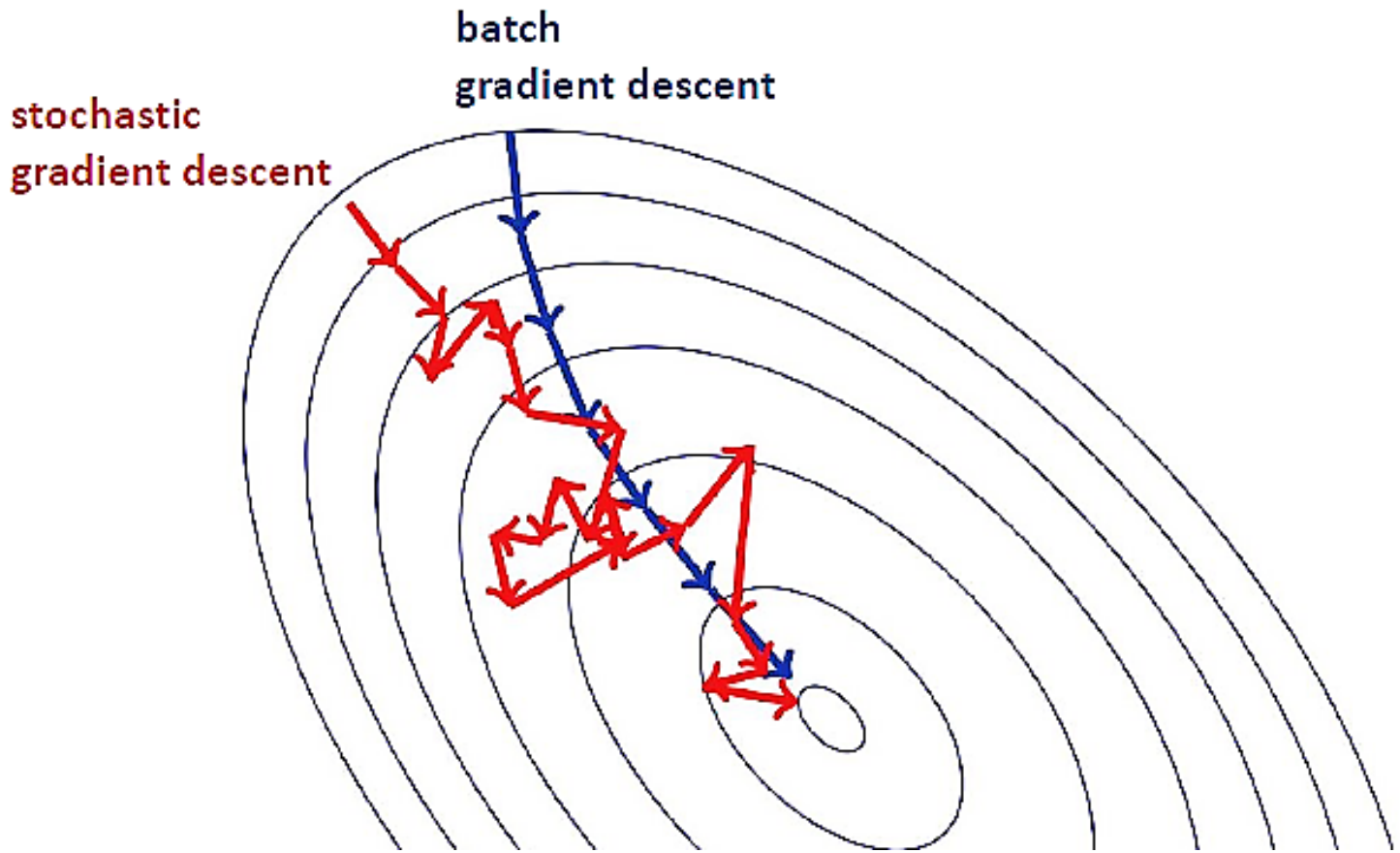
$$W_{i,j} \sim U\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right)$$

최적화 방법론

- (full) Batch gradient
- Stochastic gradient
- Mini-batch gradient
- Momentum methods

Model Selection

Batch gradient vs Stochastic gradient

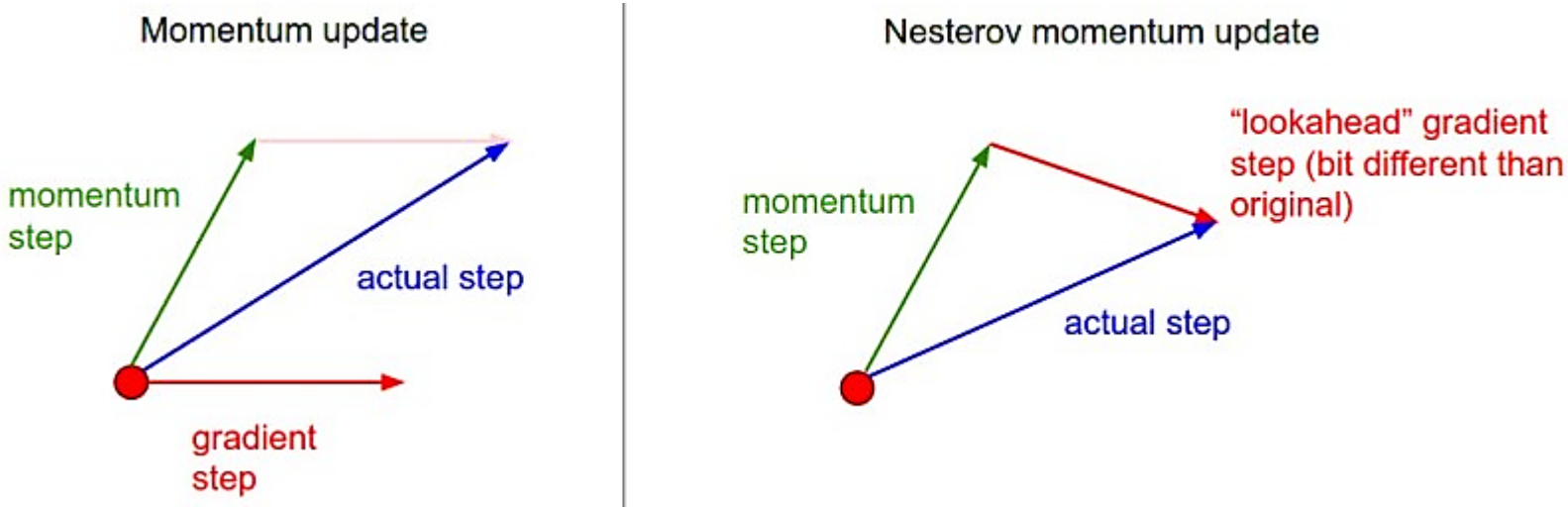


Model Selection

Momentum

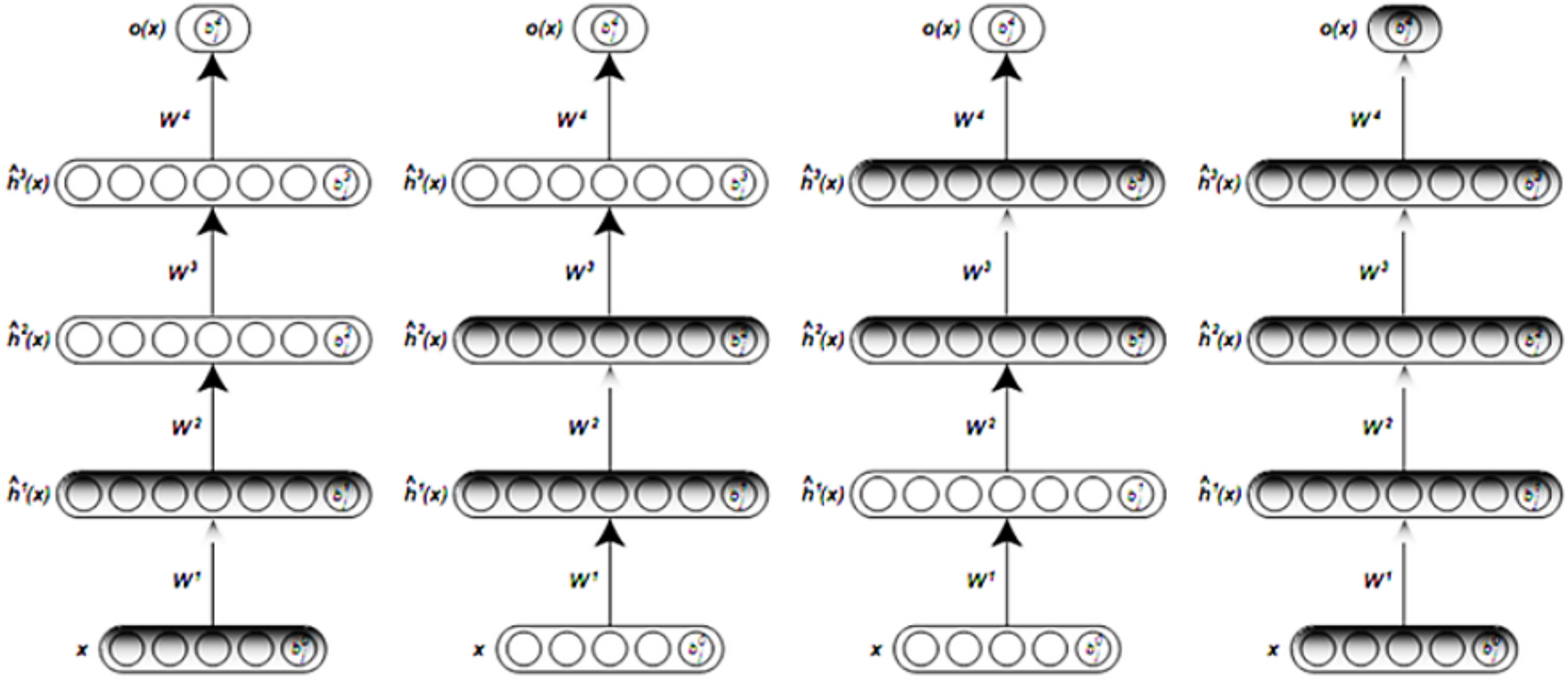
$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right)$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}.$$



Model Selection

모수의 초기 값 : greedy layerwise pre-training (참고)



Data Gathering

- 많은 경우 데이터의 추가 수집이 모델의 수정보다 더 좋은 성능 개선을 만들어내는 효과가 있음
- 언제 데이터의 추가 수집이 효과적일까?

(1) train data로 학습한 결과가 괜찮은 지 판단

- Train data에 대한 결과가 좋지 못한 경우에는 현재 데이터에 대하여 model이 충분히 학습하지 못하고 있기 때문에 model의 복잡도를 증가 시켜야 함
  => hyperparameter 조절
- 그래도 나아지지 않는다면 data의 품질에 대한 점검이 필요
  => noise 제거, 더 많은 feature 등

(2) train data에서 학습한 결과가 괜찮다면 test data에서 판단

- Test data에서의 결과가 괜찮다면 모델 학습 완료
- 그렇지 않다면, 이 때 데이터의 추가 수집이 유의미할 수 있음

실습 _ 전체 프로세스

실습 _ 필수 module import

```
[2]  #module import
     from keras.datasets import mnist
     from keras import models
     from keras import layers
     from keras.utils import to_categorical
```

데이터 로드

모델 생성 및 초기화

모델 생성 및 초기화

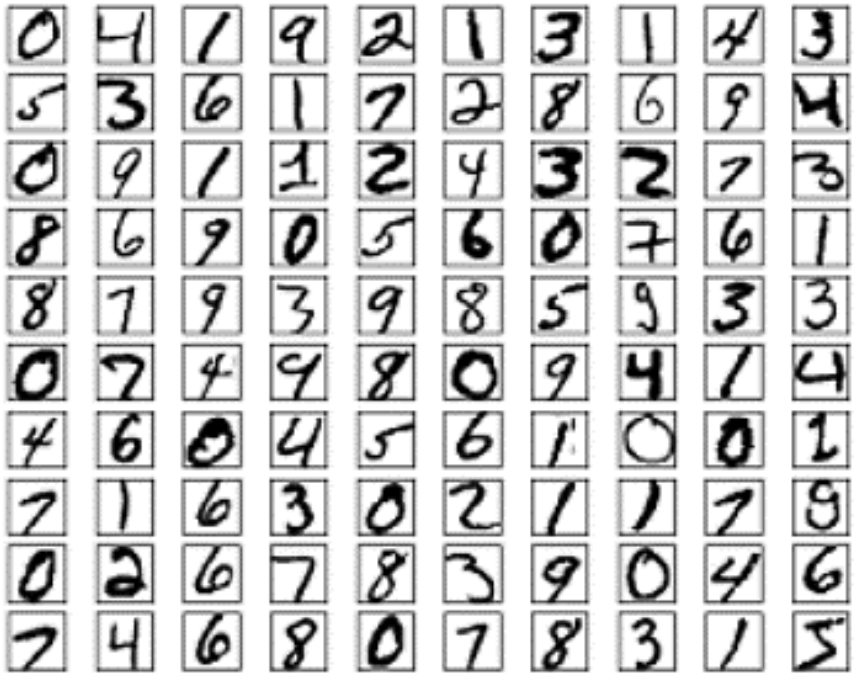
데이터 처리

실습 _ 데이터 로드(MNIST)

```
# data preprocess
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

MNIST?

- 0~9 사이의 숫자
- 이미지 파일



실습 _ 데이터 전처리

데이터 전처리(사이즈 조절 및 scale 맞추기)

```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
```

데이터 전처리(범주형 변수로 변경)

```
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

실습 _ 모델 생성 및 초기화

- 모델 선언(모델 구조 선언)

```
#model train
model = models.Sequential()
model.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
model.add(layers.Dense(10, activation='softmax'))
```

- Hideen layer의 output 노드 수는 512개
- Activation function을 ‘relu’로 지정함
- 마지막 layer의 output 노드 수는 10으로 고정해야 함 (class가 10개)
- 모델의 loss 지정

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

- 모델의 loss와 optimizer 설정

실습 _ 모델 생성 및 초기화

모델 구조 확인(선택적)

```
[5]  model.summary()

Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 512)               401920
_________________________________________________________________
dense_4 (Dense)              (None, 10)                5130
=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
_________________________________________________________________
```

Hidden layer(512)

Output layer(10)

Parameter에 대한 정보도 주어짐

실습 _ 모델 학습

모델의 학습과 관련된 매개변수 조절 및 학습

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


model.fit(train_images, train_labels, epochs=10, batch_size=128)
```

밑에 표시되는 Train loss 와 Train accuracy를 토대로 학습 경과 확인 가능

실습 _ 모델 평가

학습된 모델의 Test accuracy를 토대로 모델 평가

```
#model test
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('test_acc: ', test_acc)
```

```
test_acc:  0.9836999773979187
```

실습 _ 전체 코드 - 1

```
#module import
from keras.datasets import mnist
from keras import models
from keras import layers
from keras.utils import to_categorical

# data preprocess
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
```

실습 _ 전체 코드 - 2

```
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)


#model train
model = models.Sequential()
model.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
model.add(layers.Dense(10, activation='softmax'))
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=10, batch_size=128)


#model test
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('test_acc: ', test_acc)
```

실습 _ Overfitting 확인하기

Epoch 수 증가

1
```
model.fit(train_images, train_labels, epochs=10, batch_size=128)
```

2
```
model.fit(train_images, train_labels, epochs=20, batch_size=128)
```

실습 _ Overfitting 확인하기

1

```
Epoch 7/10
60000/60000 [==============================] - 1s 20us/step - loss: 0.0223 - accuracy: 0.9934
Epoch 8/10
60000/60000 [==============================] - 1s 21us/step - loss: 0.0173 - accuracy: 0.9949
Epoch 9/10
60000/60000 [==============================] - 1s 22us/step - loss: 0.0127 - accuracy: 0.9962
Epoch 10/10
60000/60000 [==============================] - 1s 21us/step - loss: 0.0105 - accuracy: 0.9971
```

☑ **Train accuracy는 2번 모델이 높은 것을 확인**

2

```
Epoch 17/20
60000/60000 [==============================] - 1s 20us/step - loss: 0.0017 - accuracy: 0.9995
Epoch 18/20
60000/60000 [==============================] - 1s 20us/step - loss: 0.0014 - accuracy: 0.9997
Epoch 19/20
60000/60000 [==============================] - 1s 20us/step - loss: 0.0011 - accuracy: 0.9997
Epoch 20/20
60000/60000 [==============================] - 1s 21us/step - loss: 9.4112e-04 - accuracy: 0.9998
10000/10000 [==============================] - 1s 51us/step
```

실습 _ Overfitting 확인하기

1 test_acc: 0.9840000027179718

2 test_acc: 0.9811999797821045

Overfitting 으로 인한 Test 성능의 저하를 확인할 수 있음

실습 _ Dropout

2

```
model = models.Sequential()
model.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
model.add(layers.Dense(10, activation='softmax'))
```

3

```
model = models.Sequential()
model.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
model.add(layers.Dropout(0.2,noise_shape=None,seed=None))
model.add(layers.Dense(10, activation='softmax'))
```

Dropout layer 추가 (2번 모델 3번째 줄, dropout rate=0.2)

실습 _ Dropout

2

```
Epoch 17/20
60000/60000 [==============================] - 1s 20us/step - loss: 0.0017 - accuracy: 0.9995
Epoch 18/20
60000/60000 [==============================] - 1s 20us/step - loss: 0.0014 - accuracy: 0.9997
Epoch 19/20
60000/60000 [==============================] - 1s 20us/step - loss: 0.0011 - accuracy: 0.9997
Epoch 20/20
60000/60000 [==============================] - 1s 21us/step - loss: 9.4112e-04 - accuracy: 0.9998
10000/10000 [==============================] - 1s 51us/step
```

3

```
Epoch 17/20
60000/60000 [==============================] - 1s 23us/step - loss: 0.0104 - accuracy: 0.9966
Epoch 18/20
60000/60000 [==============================] - 1s 23us/step - loss: 0.0100 - accuracy: 0.9971
Epoch 19/20
60000/60000 [==============================] - 1s 22us/step - loss: 0.0087 - accuracy: 0.9974
Epoch 20/20
60000/60000 [==============================] - 1s 23us/step - loss: 0.0081 - accuracy: 0.9976
```

Dropout 추가 후 Train error는 안 좋아진 상황

실습 _ Dropout

2 test_acc: 0.9811999797821045

3 test_acc: 0.9837999939918518

Dropout 추가 후 Test error의 향상이 있음을 확인할 수 있음

실습 _ 3가지 모델 비교

1 test_acc: 0.9840000027179718

2 test_acc: 0.9811999797821045

3 test_acc: 0.9837999939918518