

딥러닝

Deep Learning

이건명
충북대학교 대학원 산업인공지능학과

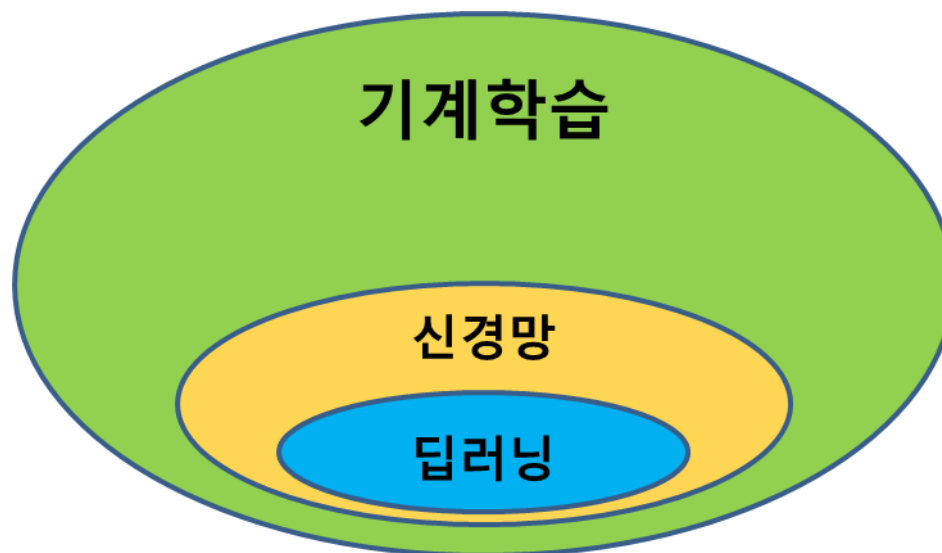
학습 내용

- 딥러닝 신경망의 특성에 대해서 알아본다.
- 기울기 소멸 문제와 해결 방안에 대해서 알아본다.
- 가중치 초기화 방법에 대해서 알아본다.
- 신경망의 과적합 완화 기법들에 대해서 알아본다.
- 신경망 학습에 사용되는 경사 하강법의 여러가지 변형 형태에 대해서 알아본다.

1. 딥러닝

❖ 딥러닝(deep learning)

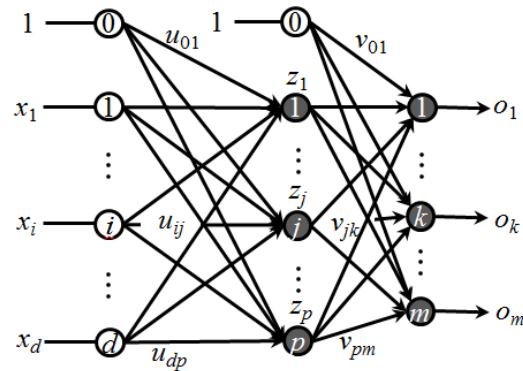
- 비교적 최근에 개발된 **딥러닝 신경망 모델** 기반의 기계 학습 기법을 차별화하여 일컫는 말
- **다수의 층**을 갖는 신경망 구조 사용
- 복잡한 구조의 신경망을 학습시키기 위해 **많은 데이터**와 **많은 컴퓨팅 자원** 사용
- 다양한 분야에서 기존 기계학습 방법보다 **훨씬 뛰어난 성능**을 보이면서 많은 관심



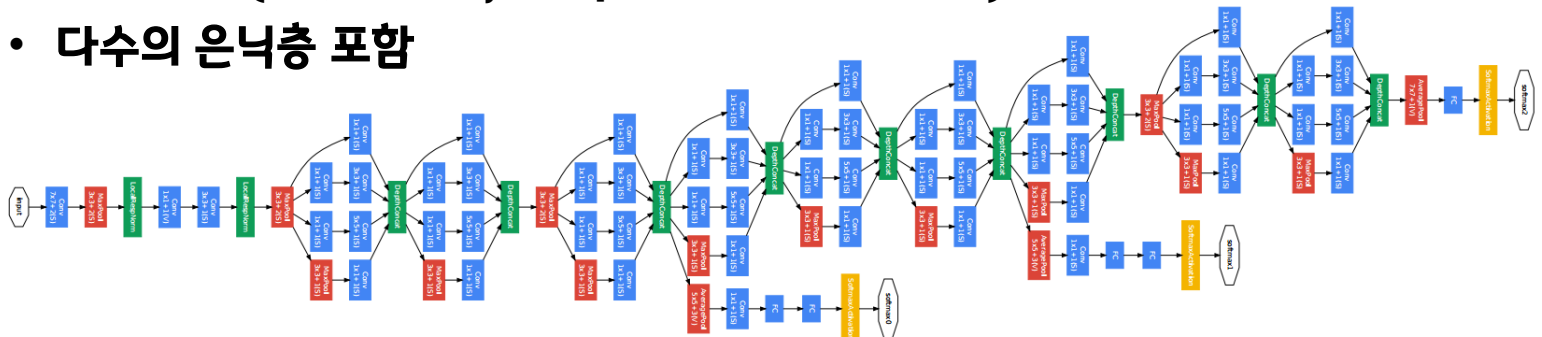
딥러닝

❖ 딥러닝(deep learning)

- 일반 신경망
 - 소수의 은닉층 포함



- 딥러닝 신경망 (심층 신경망, deep neural network)
 - 다수의 은닉층 포함

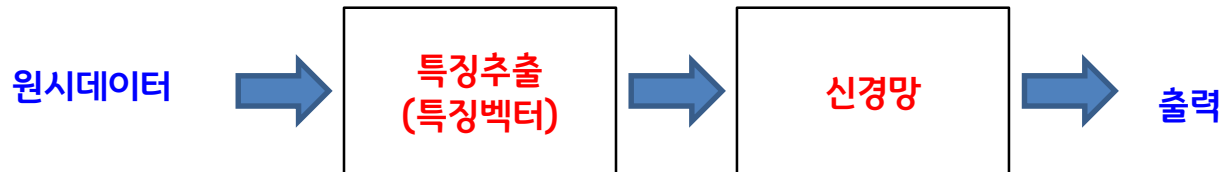


딥러닝

❖ 일반 신경망과 딥러닝 신경망

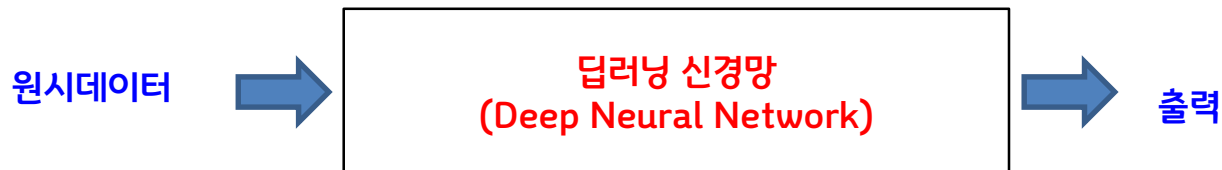
▪ 일반 신경망 모델

- 원시 데이터(original data)에서 **직접 특징(handcrafted feature)**을 추출해서 만든 **특징 벡터(feature vector)**를 신경망 모델 학습을 위한 입력으로 사용
- 특징 벡터들의 품질에 영향



▪ 딥러닝 신경망

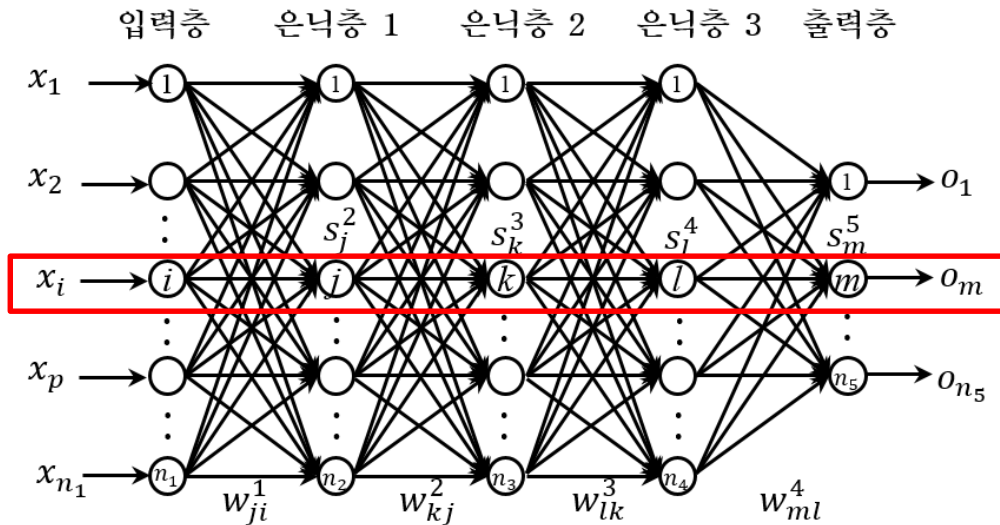
- **특징추출**과 **문제 해결**을 위한 모델의 **학습**을 함께 수행
- 데이터로부터 문제해결에 **효과적인 특징**을 학습을 통해 추출
→ 우수한 성능



2. 기울기 소멸 문제

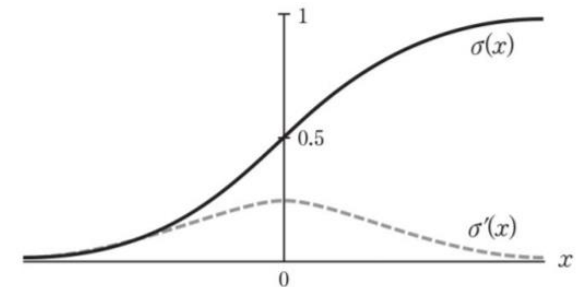
❖ 기울기 소멸 문제(Vanishing gradient problem)

- 은닉층이 많은 다층 퍼셉트론에서, 출력층에서 아래 층으로 갈 수록 전달되는 오차가 크게 줄어들어, 학습이 되지 않는 현상

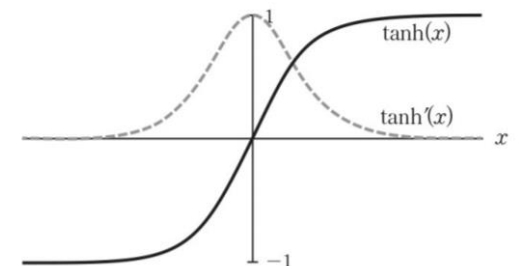


$$E = \frac{1}{2} \sum_{m=1}^{n_5} (y_m - o_m)^2$$

$$\frac{\partial E}{\partial w_{ji}^1} = \sum_{m=1}^{n_5} \sum_{l=1}^{n_4} \sum_{k=1}^{n_3} \sum_{j=1}^{n_2} (y_m - o_m) f'(s_m^5) w_{ml}^4 f'(s_l^4) w_{lk}^3 f'(s_k^3) w_{kj}^2 f'(s_j^2) x_i$$



시그모이드(sigmoid)

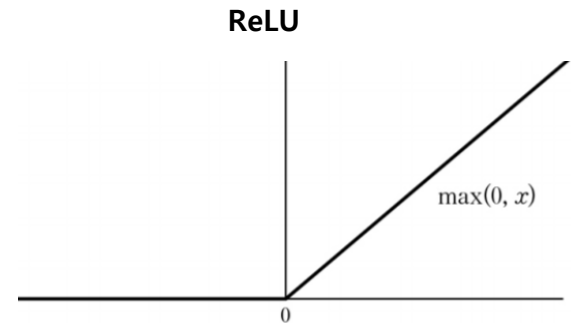
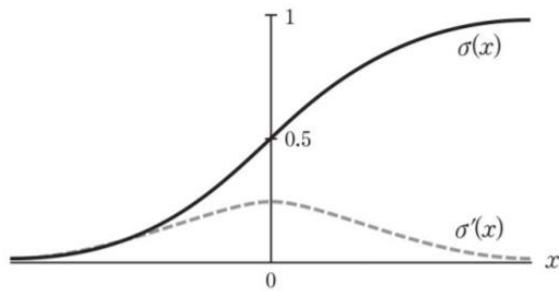


쌍곡 탄젠트(hypertangent)

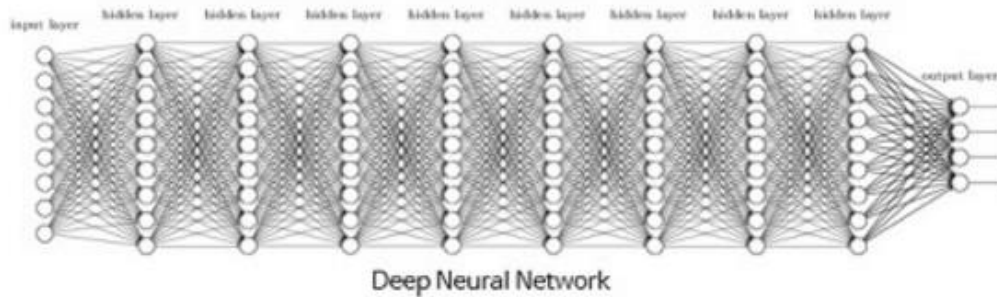
기울기 소멸 문제

❖ 기울기 소멸 문제 완화

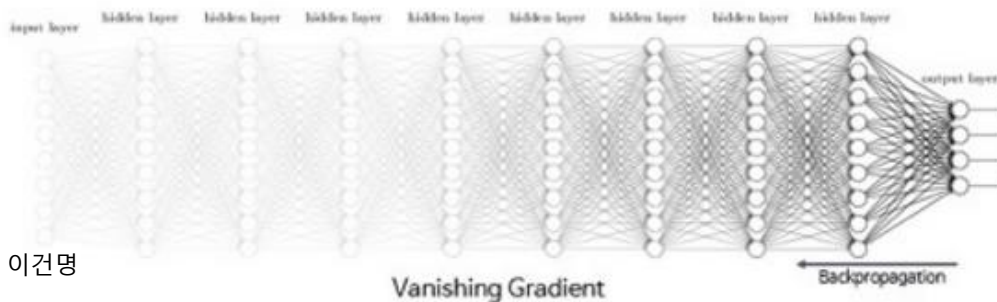
- 시그모이드나 쌍곡 탄젠트 대신 **ReLU**(Rectified Linear Unit) 함수 사용



```
def ReLU(x):  
    return np.maximum(0,x)
```



ReLU 사용



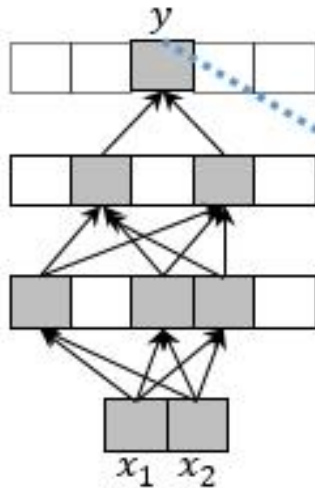
시그모이드 사용

기울기 소멸 문제

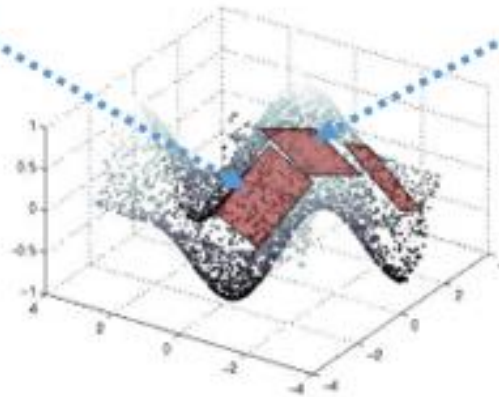
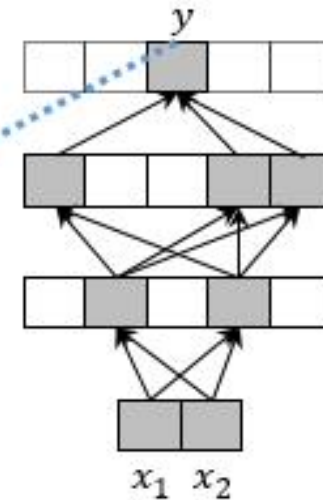
❖ ReLU 함수 사용과 함수 근사

- 함수를 부분적인 평면 타일들로 근사(approximate)하는 형태
- 출력이 0이상인 것들에 의해 계산되는 결과
 - 입력의 선형변환(입력과 가중치 행렬의 곱으로 표현)의 결과

$$y = []_{1 \times 2} \cdot []_{2 \times 3} \cdot []_{3 \times 2} x$$



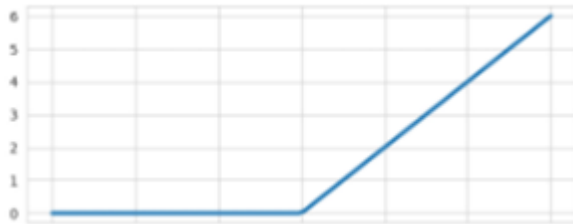
$$y = []_{1 \times 3} \cdot []_{3 \times 2} \cdot []_{2 \times 2} x$$



기울기 소멸 문제

❖ ReLU와 변형된 형태의 활성화 함수

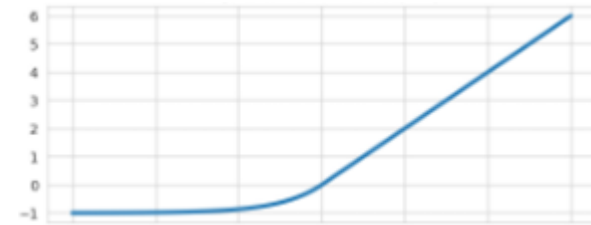
ReLU



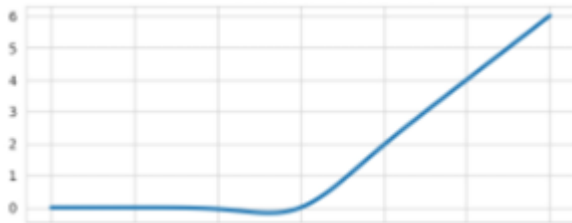
누수 ReLU(Leaky ReLU)



ELU(exponential Linear Unit)



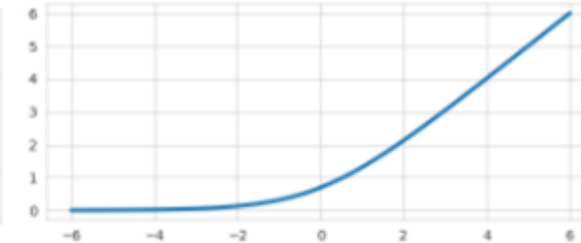
GELU(Gaussian Error Linear Unit)



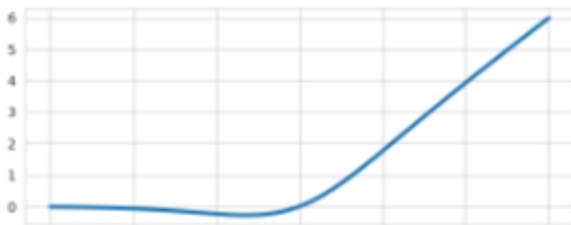
SELU



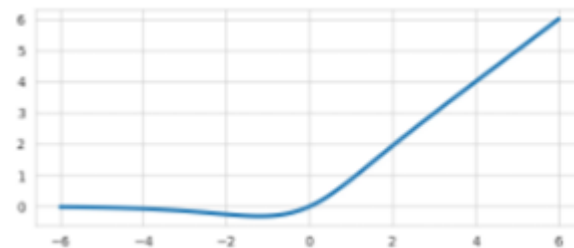
SoftPlus



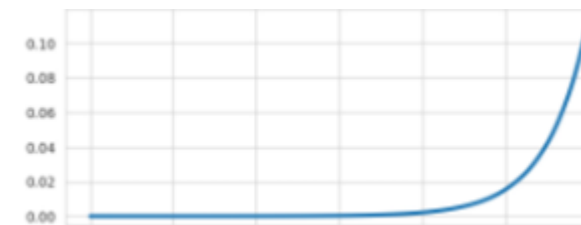
Swish



Mish

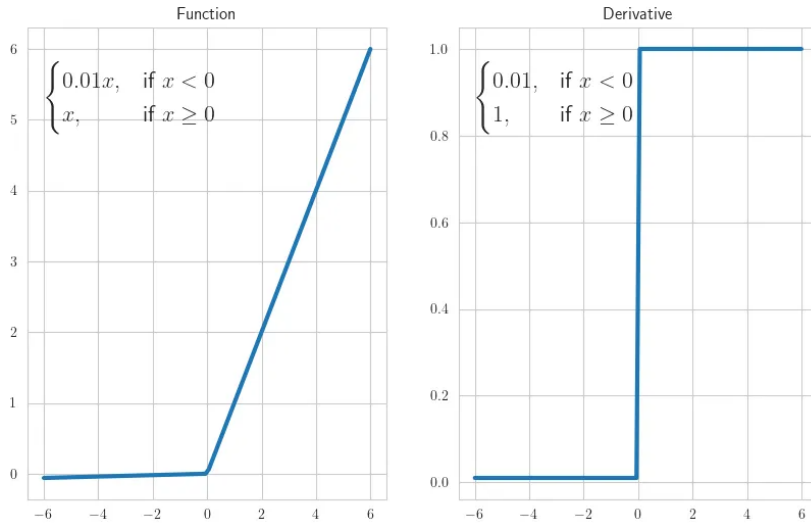


Softmax

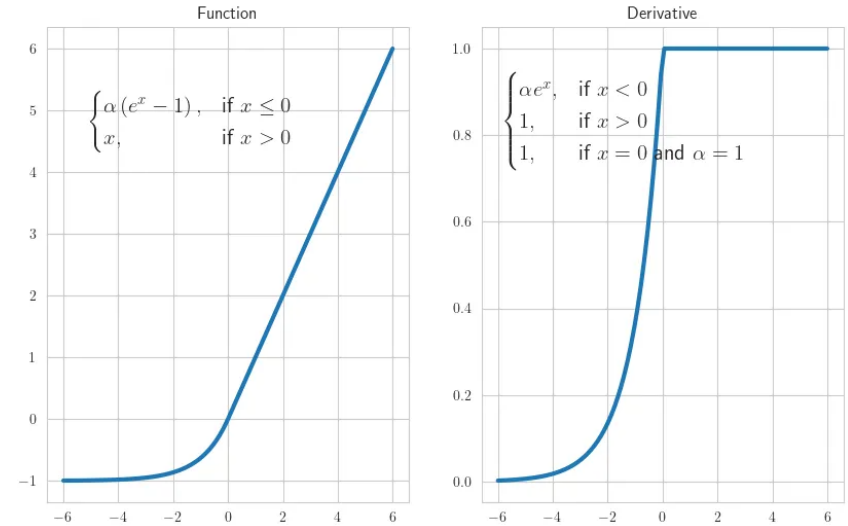


기울기 소멸 문제

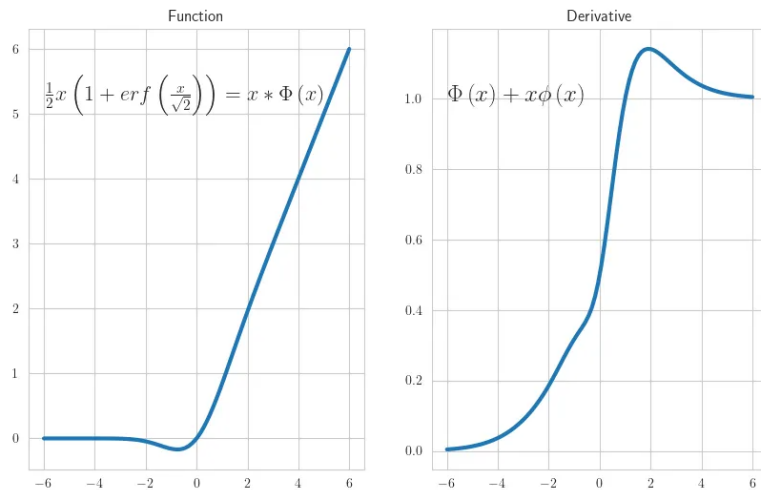
Leaky ReLU



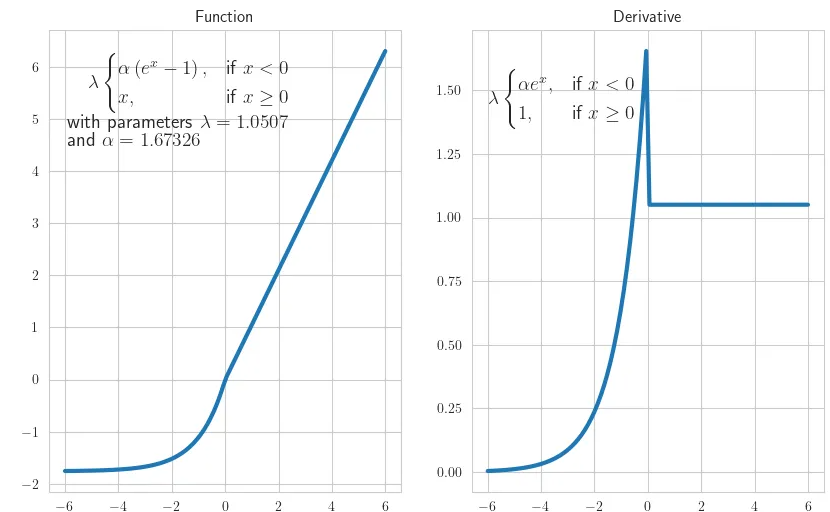
Exponential Linear Unit (ELU)



Gaussian Error Linear Unit (GELU)



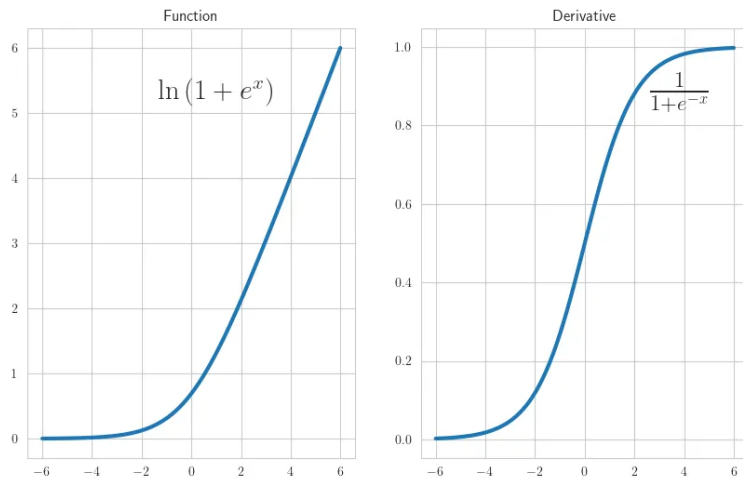
Scaled Exponential Linear Unit (SELU)



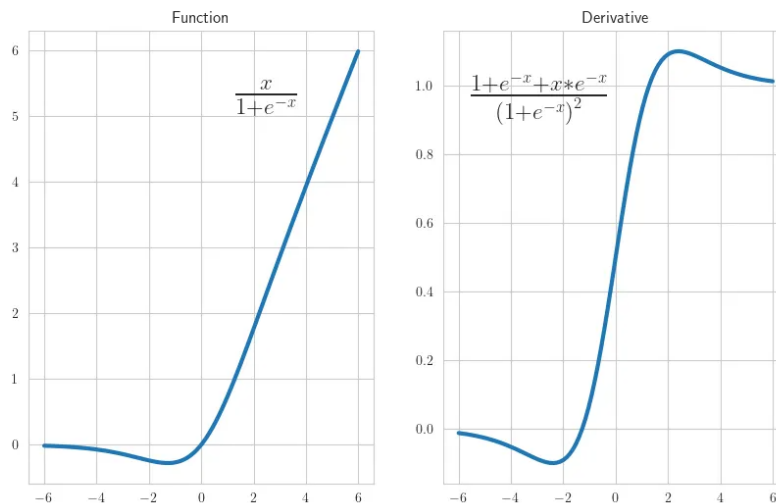
기울기 소멸 문제

❖ 활성화 함수와 도함수

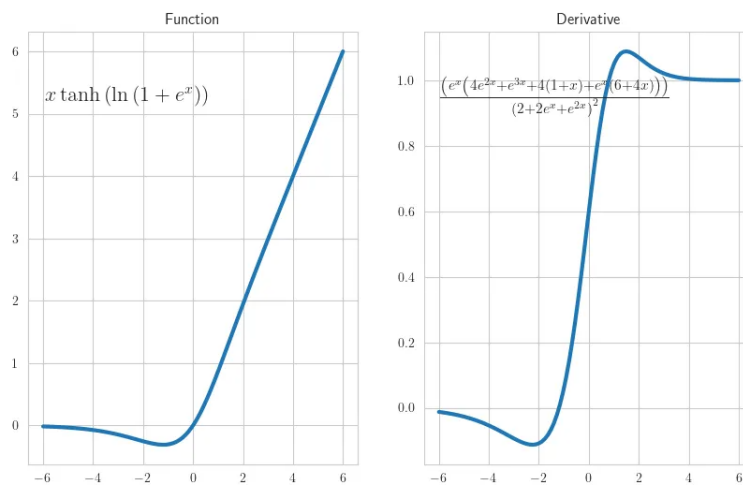
SoftPlus



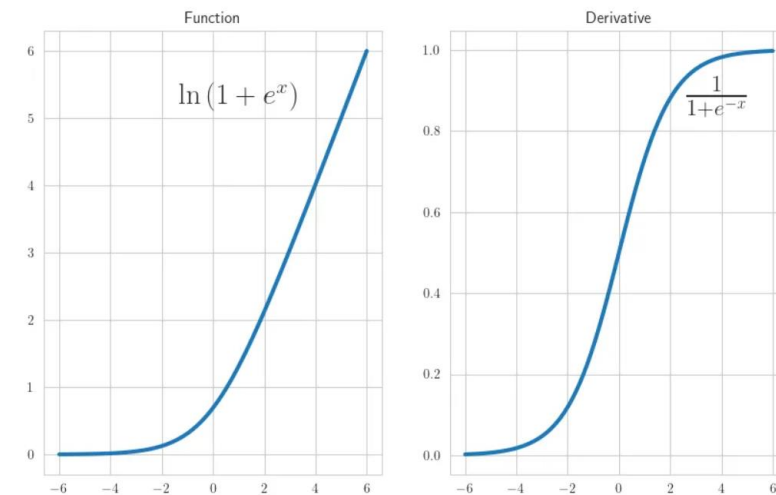
Sigmoid-Weighted Linear Unit (SiLU) / Swish



Mish Function



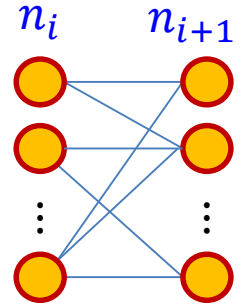
SoftPlus



3. 가중치 초기화

❖ 가중치 초기화

- 신경망의 성능에 큰 영향을 주는 요소
- 보통 가중치의 초기값으로 0에 가까운 무작위 값 사용



❖ 개선된 가중치 초기화 방법

- 각 노드의 **입력 노드 개수** n_i 와 **출력 노드의 개수** n_{i+1} 를 사용하는 방법

- **균등 분포** 초기화

$$U\left[-\sqrt{\frac{6}{n_i + n_{i+1}}}, \sqrt{\frac{6}{n_i + n_{i+1}}}\right] \text{에서 무작위로 선택}$$

- **제이비어(Xavier)** 초기화

$$\frac{N(0,1)}{\sqrt{n_i}} \text{에서 무작위로 선택}$$

$N(0,1)$: 표준 정규분포

- **허(He)** 초기화

$$\frac{N(0,1)}{\sqrt{n_i/2}} \text{에서 무작위로 선택}$$

```
sd = np.sqrt(6/(layer_size[i+1] + layer_size[i]))
W = np.random.uniform(-sd, sd, layer_size[i]*
    layer_size[i+1]).reshape(layer_size[i+1], layer_size[i])
```

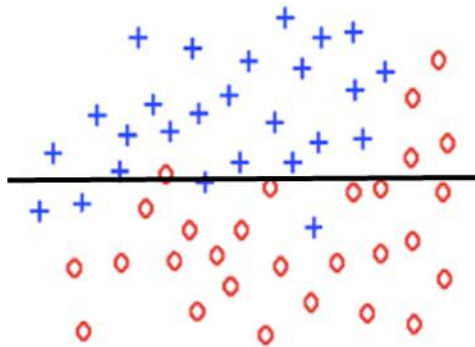
```
W = np.random.randn(layer_size[i+1], layer_size[i])
    * np.sqrt(1/layer_size[i])
```

```
W = np.random.randn(layer_size[i+1], layer_size[i])
    * np.sqrt(2/layer_size[i])
```

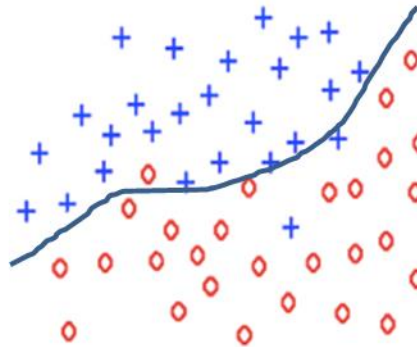
4. 과적합 문제

❖ 과적합(Overfitting)

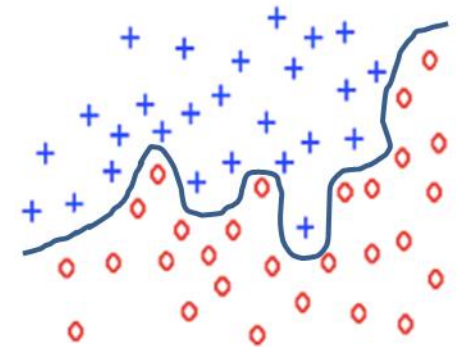
- 모델이 학습 데이터에 지나치게 맞추어진 상태
- 데이터에는 잡음이나 오류가 포함
 - 과적합된 모델은 학습되지 않는 데이터에 대해 성능 저하



부적합(underfitting)



정적합(good fitting)



과적합(overfitting)

- 과적합 문제 완화 기법
 - 규제화
 - 드롭아웃
 - 배치 정규화

과적합 문제

1. 규제화(Regularization) 기법

- 오차 함수를 오차(error) 항과 모델 복잡도(model complexity) 항으로 정의
 - 모델이 복잡해 지면 과적합이 될 수 있으므로, 모델 복잡도를 벌점(penalty) 항으로 추가

$$\text{오차 함수} = (\text{오차 항}) + \alpha(\text{모델 복잡도 항})$$

α : 상대적인 반영비율을 조정

- 신경망 학습의 모델 복잡도
 - 절대값이 큰 가중치에 벌점(penalty) 부여

$$\mathbf{w} = (w_1, w_2, \dots, w_n)$$

- 모델 복잡도 정의

$$\sum_{i=1}^n w_i^2 \quad \text{또는} \quad \sum_{i=1}^n |w_i|$$

cf. ridge/lasso regression

과적합 문제

2. 드롭아웃(Dropout) 기법

- **학습**할 때 일정 확률로 노드들을 무작위로 선택하여, 선택된 노드의 앞뒤로 연결된 가중치 연결선은 없는 것으로 간주
- **미니배치**(mini-batch)나 **학습주기**(epoch) 마다 드롭아웃 할 즉, 없는 것으로 간주할 노드들을 새롭게 선택하여 학습
- **추론**을 할 때는 드롭아웃을 하지 않고 전체 학습된 신경망을 사용하여 출력 계산

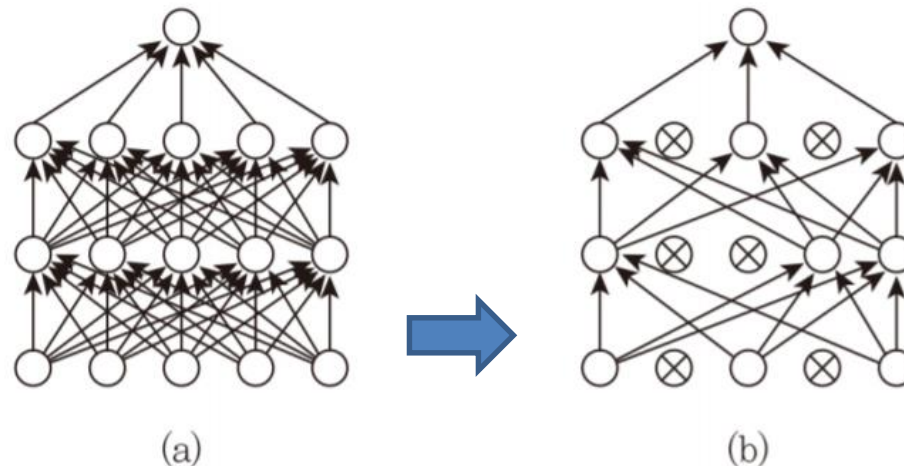


그림 5.6 드롭아웃 방법

(b)는 (a)의 신경망에 드롭아웃을 적용한 결과를 나타냄

과적합 문제

❖ 학습주기(epoch, 에포크)

- 전체 데이터에 대해서 신경망 모델을 **한번의 학습 과정**을 완료하는 것

❖ 배치(batch)

- 신경망의 **가중치**를 **한번** 수정할 때 **사용**되는 데이터
- **배치 크기(batch size)**
 - 하나의 배치에 포함되는 데이터의 개수

❖ iteration(반복)

- **한 번의 학습주기를 완료**하기 위해 수행되는 **배치**의 **처리 회수**
 - $\text{iteration 개수} = (\text{전체 데이터 개수}) / (\text{배치 크기})$

과적합 문제

❖ 학습 데이터 단위에 따른 가중치 갱신 전략

- **확률적 갱신** (stochastic update, stochastic gradient descent)
 - 한번에 **하나의 학습 데이터**에 대한 그레디언트를 계산하여 가중치 갱신
 - 가중치의 변동이 심하여 성능 개선 저하
- **배치 갱신** (batch update, batch gradient descent)
 - 전체 학습 데이터에 대한 그레디언트의 평균을 구하여 가중치 갱신
 - 느린 학습
- **미니배치 갱신** (mini-batch update, mini-batch gradient descent)
 - **일정 개수의 학습 데이터**, 미니배치에 대해 그레디언트의 평균을 구하여 가중치 갱신
 - 예. 10개 데이터로 구성된 미니배치의 그레디언드
$$\nabla g = \frac{1}{10} \sum_{i=1}^{10} \nabla g_i$$
 - 과적합 문제 완화에 도움

배치 vs 미니배치

❖ 가중치 갱신 단위

- 그래디언트 계산 단위

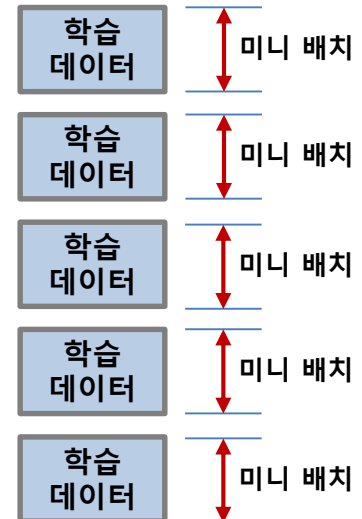
확률적 갱신
(stochastic update)

단일 학습 데이터 

배치 갱신
(batch update)



미니배치 갱신
(mini-batch update)



배치 vs 미니배치

❖ 확률적 갱신

```
for i in range(m):    # m: 에포크 수
    for one_data in training_data:
        gradient = evaluate_gradient(one_data)
        weight = weight - learning_rate * gradient
```

❖ 미니배치 갱신

```
for i in range(m):
    for one_batch in get_mini_batches(training_data, one_batch_size=32):
        gradient = evaluate_gradient(one_batch)
        weight = weight - learning_rate * gradient
```

❖ 배치 갱신

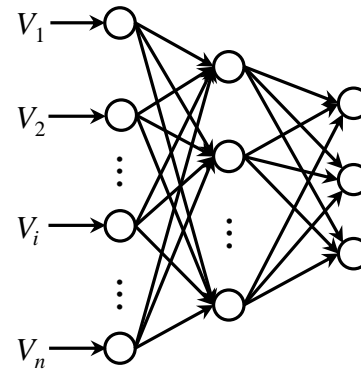
```
for i in range(m):
    gradient = evaluate_gradient(training_data)
    weight = weight - learning_rate * gradient
```

과적합 문제

3. 배치 정규화(Batch normalization) 기법

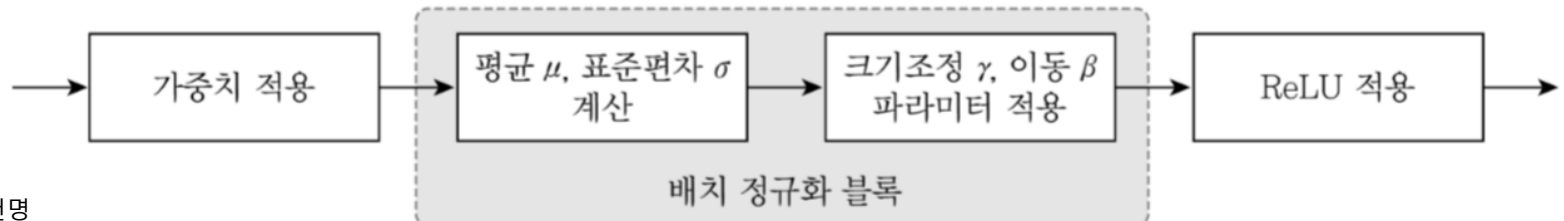
■ 내부 공변량 이동(internal covariate shift) 문제

- 오차 역전파 알고리즘 적용 학습
- 이전 층들의 학습에 의해 이들 층의 가중치가 바뀌게 되면, 현재 층에 전달되는 데이터의 분포가 현재 층이 학습했던 시점의 분포와 차이 발생 → 학습 속도 저하



■ 배치 정규화

- 신경망의 각 층에서 미니배치 B 의 각 데이터에 가중치 연산을 적용한 결과인 x_i 의 분포를 정규화하는 것



과적합 문제

❖ 배치 정규화 기법 – cont.

- 미니배치 B 에 대해

1. x_i 의 평균 μ_B 가 0이 되고 표준편차 σ_B 는 1가 되도록 변환
2. 크기조정(scaling) 파라미터 γ 와 이동(shift) 파라미터 β 적용
3. 변환된 데이터 y_i 생성

- 가중치 연산 결과의 미니 배치 : $B = \{x_1, x_2, \dots, x_m\}$
- 배치 정규화 적용 결과 : $\{y_1, y_2, \dots, y_m\}$

미니배치의 평균 : $\mu_B = \frac{1}{m} \sum_{i=1}^m x_i$

미니배치의 분산 : $\sigma_B^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2$

정규화 : $\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$

크기조정 및 이동변환 : $y_i = \gamma \hat{x}_i + \beta$

미니 배치 B
 $m=3$

feature ₁	feature ₂	feature ₃	feature ₄
x_{11}	x_{12}	x_{13}	x_{14}
x_{21}	x_{22}	x_{23}	x_{24}
x_{31}	x_{32}	x_{33}	x_{34}

평균(mean)

...
-----	-----	-----	-----

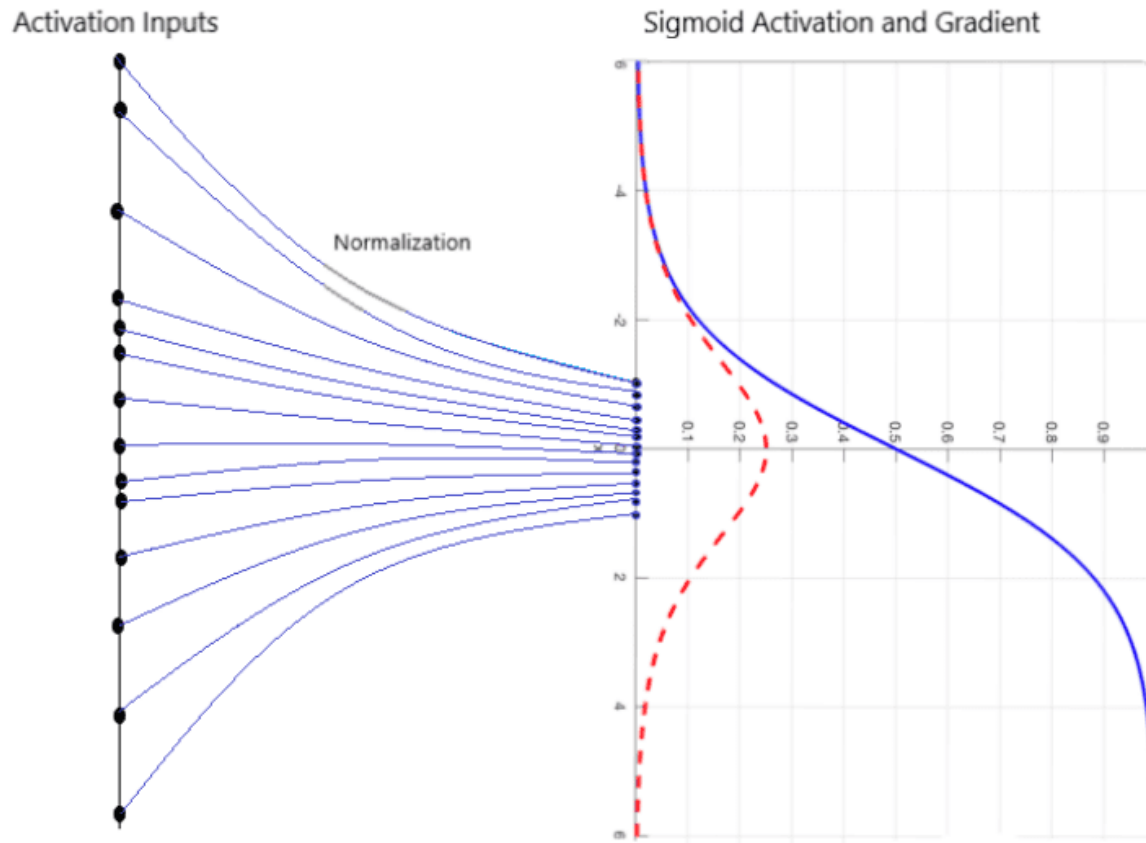
표준편차(std)

...
-----	-----	-----	-----

γ, β : 학습 대상 파라미터

과적합 문제

❖ 배치 정규화 기법 – cont.



5. 가중치 학습 기법(optimizer)

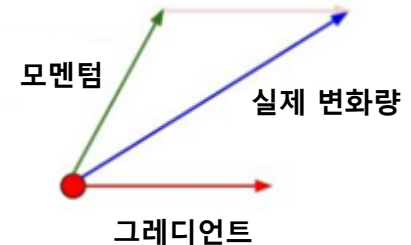
❖ 경사 하강법(Gradient descent method)

$$w^{(t+1)} = w^{(t)} - \eta \frac{\partial E(w^{(t)})}{\partial w}$$

❖ 모멘텀 사용 경사 하강법(Gradient descent method with Momentum)

$$\Delta^{(t)} = \alpha \Delta^{(t-1)} + \eta \frac{\partial E(w^{(t)})}{\partial w}$$

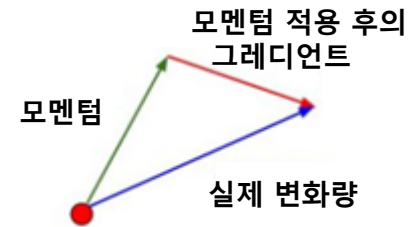
$$w^{(t+1)} = w^{(t)} - \Delta^{(t)}$$



❖ NAG(Nesterov accelerated gradient)

$$\Delta^{(t)} = \alpha \Delta^{(t-1)} + \eta \frac{\partial E(w^{(t)} - \alpha \Delta^{(t-1)})}{\partial w}$$

$$w^{(t+1)} = w^{(t)} - \Delta^{(t)}$$



가중치 학습 기법

❖ Adagrad (Adaptive Gradient Algorithm)

- 가중치 별도 다른 학습을 사용

$$g_i^{(t)} = \frac{\partial E(\mathbf{w}^{(t)})}{\partial w_i} \quad G_i^{(t)} = G_i^{(t-1)} + (g_i^{(t)})^2$$

$$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta}{\sqrt{G_i^{(t)} + \epsilon}} g_i^{(t)}$$

❖ Adadelata

- Adagrad의 확장
- 과거 그레디언트의 영향을 점점 줄이면서 그레디언트 제곱합 계산

$$E[g_i^2]_t = \gamma E[g_i^2]_{t-1} + (1 - \gamma)(g_i^{(t)})^2 \quad RMS[g_i]^{(t)} = \sqrt{E[g_i^2] + \epsilon}$$

$$E[w_i^2]_t = \gamma E[w_i^2]_{t-1} + (1 - \gamma) \left(\frac{\eta}{RMS[g_i]^{(t)}} g_i^{(t)} \right)^2 \quad RMS[w_i]^{(t)} = \sqrt{E[w_i^2] + \epsilon}$$

$$w_i^{(t+1)} = w_i^{(t)} - \frac{RMS[w_i]^{(t-1)}}{RMS[g_i]^{(t)}} g_i^{(t)}$$

가중치 학습 기법

❖ RMSprop

- 가중치별 다른 학습율 사용
- 결합된 그레디언트 제곱의 합의 제곱근을 학습율로 사용

$$E[g_i^2]_t = \gamma E[g_i^2]_{t-1} + (1 - \gamma)(g_i^{(t)})^2$$

$$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta}{\sqrt{E[g_i^2]^{(t)} + \epsilon}} g_i^{(t)}$$

❖ ADAM (Adaptive Moment Estimation)

- 가중치별 다른 학습율 사용
- 그레디언트의 1차, 2차 모멘텀 사용

$$m^{(t)} = \beta_1 m^{(t-1)} + (1 - \beta_1) g_i^{(t)}$$

$$v^{(t)} = \beta_2 v^{(t-1)} + (1 - \beta_2) (g_i^{(t)})^2$$

$$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta}{\sqrt{\hat{v}^{(t)} + \epsilon}} \hat{m}^{(t)}$$

$$\hat{m}^{(t)} = \frac{m^{(t)}}{1 - \beta_1^{(t)}}$$

$$\hat{v}^{(t)} = \frac{v^{(t)}}{1 - \beta_2^{(t)}}$$

가중치 학습 기법

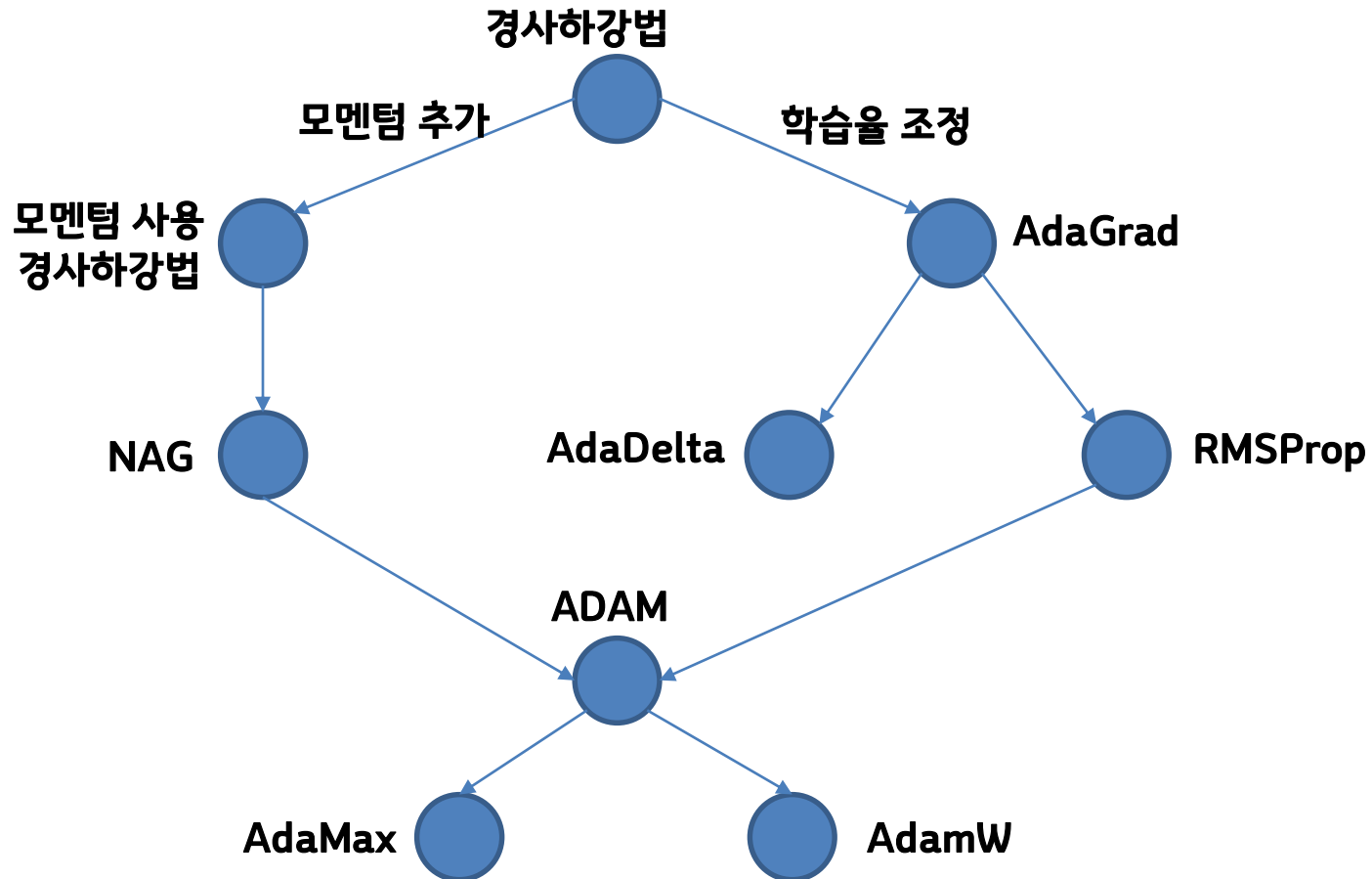
❖ AdamW

- L2 정규화와 가중치 감소(weight decay) 방법을 함께 적용한 Adam

Algorithm 2 Adam with L₂ regularization and Adam with decoupled weight decay (AdamW)

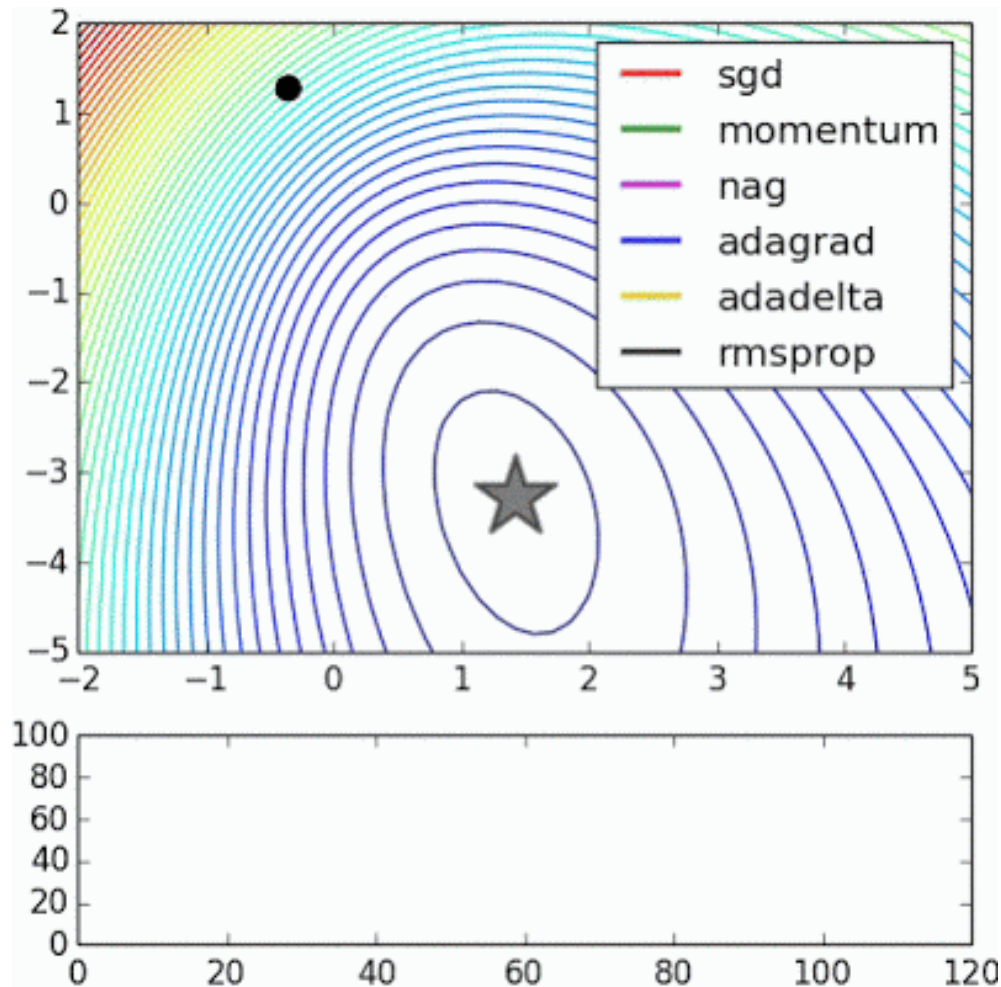
```
1: given  $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$ 
2: initialize time step  $t \leftarrow 0$ , parameter vector  $\theta_{t=0} \in \mathbb{R}^n$ , first moment vector  $m_{t=0} \leftarrow \mathbf{0}$ , second moment vector  $v_{t=0} \leftarrow \mathbf{0}$ , schedule multiplier  $\eta_{t=0} \in \mathbb{R}$ 
3: repeat
4:    $t \leftarrow t + 1$ 
5:    $\nabla f_t(\theta_{t-1}) \leftarrow \text{SelectBatch}(\theta_{t-1})$  ▷ select batch and return the corresponding gradient
6:    $g_t \leftarrow \nabla f_t(\theta_{t-1}) + \lambda \theta_{t-1}$ 
7:    $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$  ▷ here and below all operations are element-wise
8:    $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$ 
9:    $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$  ▷  $\beta_1$  is taken to the power of  $t$ 
10:   $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$  ▷  $\beta_2$  is taken to the power of  $t$ 
11:   $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$  ▷ can be fixed, decay, or also be used for warm restarts
12:   $\theta_t \leftarrow \theta_{t-1} - \eta_t \left( \alpha \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon) + \lambda \theta_{t-1} \right)$ 
13: until stopping criterion is met
14: return optimized parameters  $\theta_t$ 
```

가중치 학습 기법



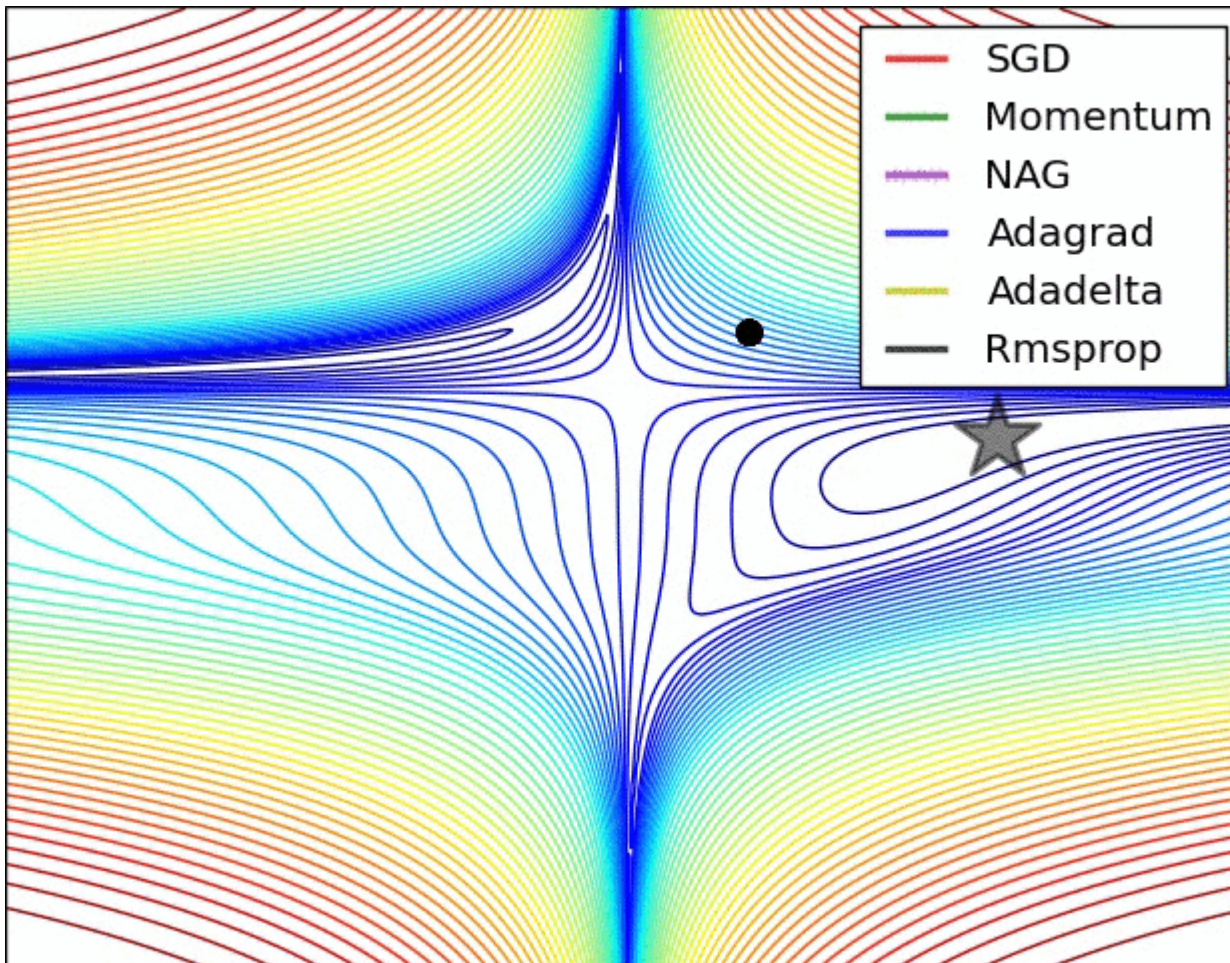
가중치 학습 기법

❖ 여러 경사 하강법의 동작 형태



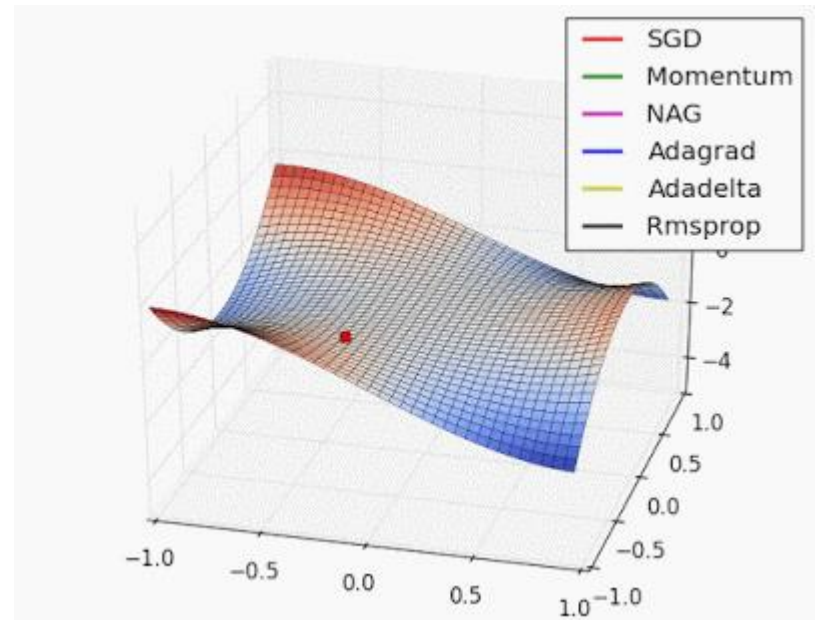
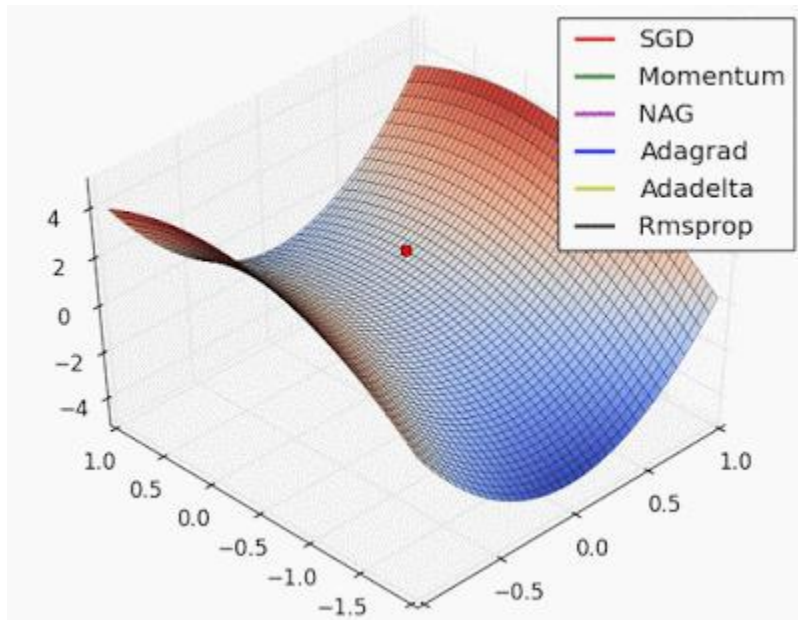
가중치 학습 기법

❖ 여러 경사 하강법의 동작 형태



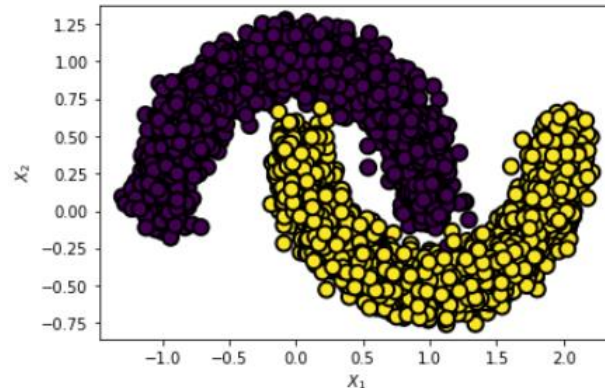
가중치 학습 기법

❖ 여러 경사 하강법의 동작 형태



[프로그래밍 실습: Gradient Descent]

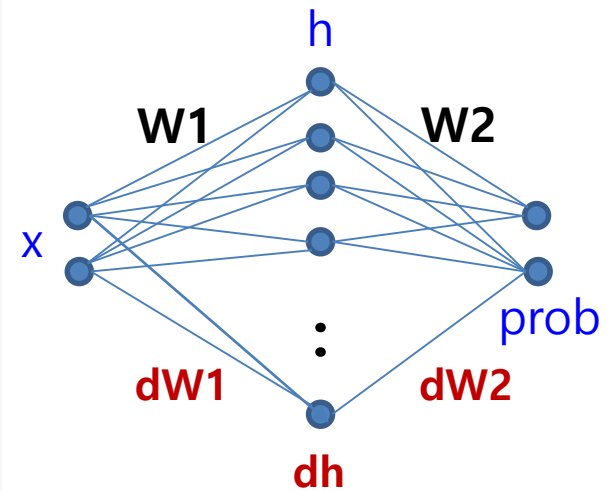
```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.datasets import make_moons
4 from sklearn.model_selection import train_test_split
5
6 n_samples = 5000          # 데이터 개수
7 minibatch_size = 50       # 미니배치 크기
8
9 n_feature = 2
10 n_class = 2
11 X, y = make_moons(n_samples=5000, random_state=42, noise=0.1)
12 plt.scatter(X[:, 0], X[:, 1], marker='o', c=y, s=100,
13            edgecolor="k", linewidth=2)
14 plt.xlabel("$X_1$")
15 plt.ylabel("$X_2$")
16 plt.show()
17
18 X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.75, random_state=42)
```



```

1 def make_network(n_hidden=100):
2     model = dict(
3         W1 = np.random.randn(n_feature, n_hidden),
4         W2 = np.random.randn(n_hidden, n_class)
5     )
6     return model
7
8 def softmax(x):
9     return np.exp(x) / np.exp(x).sum()
10
11 def forward(x, model):
12     h = x @ model['W1']           # 입력과 가중치의 행렬곱
13     h[h < 0] = 0                 # ReLU 연산 적용
14     prob = softmax(h @ model['W2']) # 출력층 계산
15     return h, prob
16
17 def backward(model, xs, hs, errs):
18     # xs, hs, errs : 미니배치의 전체 (input, hidden state, error)
19     dW2 = hs.T @ errs
20     dh = errs @ model['W2'].T
21     dh[hs <= 0] = 0
22     dW1 = xs.T @ dh
23     return dict(W1=dW1, W2=dW2)

```




```

1 n_iteration = int(len(X_train)/minibatch_size)    # iteration 수
2
3 def get_minibatch_grad(model, X_train, y_train):
4     xs, hs, errs = [], [], []
5
6     for x, cls_idx in zip(X_train, y_train):
7         h, y_pred = forward(x, model)
8
9         y_true = np.zeros(n_class)
10        y_true[int(cls_idx)] = 1.
11
12        err = y_true - y_pred    # 오차
13        xs.append(x)            # 입력 저장
14        hs.append(h)            # 은닉층 출력 저장
15        errs.append(err)        # 오차 저장
16
17    # 현재 미니배치를 이용한 Backprop
18    return backward(model, np.array(xs), np.array(hs), np.array(errs))

```

Gradient Descent

```
1 def shuffle(X, y):
2     indices = np.arange(X.shape[0])
3     np.random.shuffle(indices)
4     X = X[indices]
5     y = y[indices]
6     return X, y
7
8 def get_minibatch(X, y, minibatch_size):
9     minibatches = []
10    X, y = shuffle(X, y)
11
12    for i in range(0, X.shape[0], minibatch_size):
13        X_mini = X[i:i + minibatch_size]
14        y_mini = y[i:i + minibatch_size]
15        minibatches.append((X_mini, y_mini))
16
17    return minibatches
18
19 def GradientDescent(model, X_train, y_train, minibatch_size, eta=1e-4):
20     minibatches = get_minibatch(X_train, y_train, minibatch_size)
21     for idx in range(0, n_iteration):
22         X_mini, y_mini = minibatches[idx]
23         grad = get_minibatch_grad(model, X_mini, y_mini)
24
25         for layer in grad:
26             model[layer] += eta * grad[layer]
27     return model
```

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \frac{\partial E(\mathbf{w}^{(t)})}{\partial \mathbf{w}}$$

```

1 n_experiment = 100          # 실험회수
2 learning_rate = 1e-4        # 학습율
3 accs = np.zeros(n_experiment) # 정확도 저장
4
5 for k in range(n_experiment):
6     model = make_network()    # 신경망 구성
7     model = GradientDescent(model, X_train, y_train, minibatch_size, learning_rate)
8     y_pred = np.zeros_like(y_test)
9
10    for i, x in enumerate(X_test):
11        _, prob = forward(x, model)
12        y = np.argmax(prob)
13        y_pred[i] = y
14
15    accs[k] = (y_pred == y_test).sum() / y_test.size # 정확도 계산
16
17 print('정확도 평균: {}, 표준편차: {}'.format(accs.mean(), accs.std()))

```

정확도 평균: 0.8193520000000001, 표준편차: 0.07025199851961508

Quiz

❖ 일반 신경망과 딥러닝 신경망에 대한 설명으로 옳지 않은 것은?

- ① 일반 신경망인 다층 퍼셉트론에서 은닉층의 개수는 2~3개 정도로 적다.
- ② 일반 신경망에서 특징 추출을 별도로 할 경우 개발자가 이를 처리해줘야 한다.
- ③ 딥러닝 신경망에서는 데이터에 대한 특징 추출 방법이 학습을 통해서 결정될 수 있다.
- ④ 기울기 소멸 문제는 계단 모양 활성화 함수를 사용하여 완화시킬 수 있다.

❖ 다음 신경망의 학습에 대한 설명으로 옳지 않은 것은?

- ① 시그모이드 함수의 미분값은 0.25이하인 양의 값이기 때문에, 이를 활성화 함수로 사용하면 기울기 소멸 문제가 발생할 수 있다.
- ② ReLU 함수의 출력값은 0이 될 수 없다.
- ③ 제이비어(Xavier) 기법을 사용할 때 가중치의 초기값은 층(layer)에 있는 노드 개수에 영향을 받는다.
- ④ 동일한 학습 알고리즘을 적용하더라도 신경망에 있는 가중치의 초기값은 성능에 영향을 크게 줄 수 있다.

Quiz

❖ 과적합에 대한 설명으로 옳지 않은 것은?

- ① 과적합 상태이면 테스트 데이터에 대한 성능이 학습 데이터에 대한 성능보다 좋다.
- ② 오차함수를 오차항과 모델 복잡도항으로 구성함으로써 과적합을 완화시킬 수 있다.
- ③ 학습할 때 일정 확률로 노드들을 무작위로 선택하여 해당 노드에 대한 연결선이 없는 것처럼 학습에 배제하는 드롭아웃은 과적합 해소에 도움이 된다.
- ④ 미니배치 단위로 가중치를 갱신하는 학습을 하면 과적합을 완화시키는데 도움이 될 수 있다.

❖ 다음 가중치 학습 기법에 대한 설명으로 옳지 않은 것은?

- ① 딥러닝 신경망의 학습 알고리즘은 기본적으로 경사 하강법에 기반한다.
- ② 모멘텀 사용 경사 하강법에서는 직전 시점의 가중치 갱신 정보를 일부 활용하여 가중치를 갱신한다.
- ③ Adadelata 알고리즘은 가중치별로 학습율에 다르게 적용될 수 있도록 한다.
- ④ 현재 알려진 가중치 학습 기법 중 ADAM이 가장 성능이 우수한 기법이다.

Quiz

❖ **딥러닝 모델의 성능을 향상시키기 위해 사용하는 기법은 아닌 것은?**

- ① 데이터 확대
- ② 정규화
- ③ 배치 정규화
- ④ 특성 삭제

❖ **일반신경망과 딥러닝 신경망에 대한 설명으로 옳지 않은 것은?**

- ① 딥러닝 신경망은 층의 수가 많을수록 과적합에 취약하다.
- ② 일반 신경망은 과적합에 상대적으로 덜 취약하다.
- ③ 딥러닝 신경망은 자동으로 특징을 추출한다.
- ④ 일반 신경망은 깊은 구조를 가진다.

❖ **일반 신경망과 딥러닝 신경망의 주요 차이점은?**

- ① 사용되는 뉴런의 수
- ② 네트워크의 깊이
- ③ 활성화 함수의 종류
- ④ 그래디언트 소멸 문제의 발생

Quiz

❖ 기울기 소멸 문제는 주로 어떤 모델에서 발생하는가?

- ① 얇은 신경망
- ② 깊은 신경망
- ③ 단일 계층 신경망
- ④ SVM

❖ 어떤 모델이 주로 복잡한 계층적 특징을 학습하는 데 적합한가?

- ① 얇은 신경망
- ② 깊은 신경망
- ③ 단일 계층 신경망
- ④ 선형 회귀 모델

❖ 딥러닝 신경망에서는 주로 어떤 유형의 활성화 함수를 사용하여 그래디언트 소멸 문제를 완화하는가?

- ① Sigmoid 함수
- ② Step 함수
- ③ ReLU
- ④ 선형 함수

Quiz

❖ 기울기 소멸 문제가 발생할 때, 가중치 업데이트는 어떤 특성을 보이는가?

- ① 지나치게 큰 업데이트
- ② 매우 작은 업데이트
- ③ 무작위한 업데이트
- ④ 일정한 크기의 업데이트

❖ 기울기 소멸 문제는 어떤 활성화 함수와 함께 발생할 가능성이 높은가?

- ① ReLU
- ② Leaky ReLU
- ③ Sigmoid
- ④ Softmax

❖ 어떤 활성화 함수가 입력 값의 모든 부분에서 미분 가능한가?

- ① ReLU
- ② Leaky ReLU
- ③ Sigmoid
- ④ Step function

Quiz

❖ 기울기 소멸 문제가 주로 발생하는 경우는?

- ① 층이 많은 신경망에서
- ② 학습률이 너무 높을 때
- ③ 가중치가 무작위로 초기화될 때
- ④ 활성화 함수로 ReLU를 사용할 때

❖ 기울기 소멸 문제를 일으키는 활성화 함수는?

- ① ReLU
- ② 시그모이드
- ③ Leaky ReLU
- ④ ELU

❖ 기울기 소멸 문제를 완화하는 방법 중 가장 잘못된 것은?

- ① 적절한 가중치 초기화 방법 사용
- ② 활성화 함수로 ReLU 계열 함수 사용
- ③ 학습률을 점차 증가시키는 방법
- ④ 배치 정규화(Batch Normalization)

Quiz

- ❖ **신경망에서 초기 가중치 설정의 중요성에 대한 잘못된 설명은?**
 - ① 초기 가중치는 학습의 시작점을 결정한다.
 - ② 너무 큰 초기 가중치는 학습 과정을 불안정하게 만들 수 있다.
 - ③ 모든 가중치를 같은 값으로 초기화하는 것이 바람직하다.
 - ④ 초기 가중치는 무작위로 설정되거나 특정 기법을 사용하여 초기화된다.

- ❖ **신경망에서 배치 크기(Batch Size)의 역할에 대한 잘못된 설명은?**
 - ① 한 번에 처리하는 데이터의 수를 결정한다.
 - ② 너무 크면 학습 속도가 느려질 수 있다.
 - ③ 아주 작게 하면 모델의 일반화 성능이 향상된다.
 - ④ 메모리 용량과 계산 속도에 영향을 준다.

- ❖ **신경망에서 손실 함수(Loss Function)의 목적에 대한 잘못된 설명은?**
 - ① 예측값과 실제값 간의 차이를 측정한다.
 - ② 학습 과정에서 최소화되어야 한다.
 - ③ 손실 함수의 선택은 문제 유형에 따라 달라진다.
 - ④ 손실 함수의 값은 학습 과정에 영향을 주지 않는다.

Quiz

❖ 가중치 초기화에서 0으로 초기화하는 방법의 문제점은?

- ① 신경망이 대칭적인 가중치를 갖게 되어 학습이 제대로 이루어지지 않는다.
- ② 0으로 초기화하면 뉴런의 출력이 항상 0이 되어 학습이 이루어질 수 없다.
- ③ 가중치 감소(weight decay) 기법을 적용할 때 문제가 발생할 수 있다.
- ④ 모든 가중치가 같은 값을 갖기 때문에 기울기 소멸 문제가 발생한다.

❖ Xavier 초기화 방법의 주된 특징은?

- ① 가중치를 일정 범위 내에서 균등하게 분포시키는 방법이다.
- ② 초기 가중치를 큰 값으로 설정하여 활성화 함수의 비선형 영역을 활용한다.
- ③ 각 층의 입력 노드의 개수에 따라 분산의 크기를 조정한다.
- ④ 무작위성을 최대화하여 모든 가중치가 다른 값을 갖도록 한다.

❖ He 초기화 방법에서 사용하는 분산의 계산에 사용되는 요소는?

- ① 은닉층의 노드 수
- ② 학습률
- ③ 각 층의 입력 노드 수
- ④ 가중치의 개수

Quiz

❖ L1 규제화는 어떤 특징을 가지고 있는가?

- ① 가중치의 제곱을 규제
- ② 가중치의 절댓값을 규제
- ③ 가중치의 합을 규제
- ④ 가중치의 제곱근을 규제

❖ 규제화는 어떤 상황에서 효과적인가요?

- ① 훈련 데이터가 충분할 때
- ② 모델이 너무 단순할 때
- ③ 모델이 너무 복잡할 때
- ④ 훈련 데이터가 아주 적을 때

❖ 규제화를 사용할 때 주의해야 할 점은?

- ① 너무 강한 규제는 과소적합을 일으킬 수 있음
- ② 규제는 항상 모델의 성능을 향상시킴
- ③ 규제는 항상 모델의 학습 속도를 늦춤
- ④ 규제는 모든 문제에 적합하다

Quiz

❖ 배치 정규화(Batch Normalization)의 주된 목적은?

- ① 과적합 방지
- ② 학습 속도 개선
- ③ 모델의 계산 복잡도 감소
- ④ 모델의 크기 축소

❖ 배치 정규화는 어느 위치에 주로 적용되는가?

- ① 활성화 함수 전
- ② 활성화 함수 후
- ③ 손실 함수 전
- ④ 최적화 함수 후

❖ 배치 정규화에서 계산할 필요가 있는 것은?

- ① 평균과 분산
- ② 가중치와 편향
- ③ 학습률과 모멘텀
- ④ 손실 값과 그래디언트 값

Quiz

❖ 배치 정규화는 어떤 값을 이용해 정규화하는가?

- ① 한 배치 내의 데이터들의 중앙값
- ② 전체 데이터의 평균값
- ③ 한 배치 내의 데이터들의 평균값
- ④ 전체 데이터의 중앙값

❖ 배치 정규화에서 발생하는 연산이 아닌 것은?

- ① 스케일(scale) 조정
- ② 이동(shift)
- ③ 회전(rotation)
- ④ 정규화(normalization)

❖ 배치 정규화의 장점은?

- ① 더 큰 학습률을 사용할 수 있게 해줌
- ② 가중치 초기화의 영향을 줄여줌
- ③ 내부 공변량 변화를 완화
- ④ 모델의 크기를 물리적으로 줄여줌

Quiz

❖ 경사하강법(Gradient Descent)의 핵심 아이디어는?

- ① 손실 함수의 최대값을 찾는 것
- ② 모든 가중치를 무작위로 조정하는 것
- ③ 손실 함수의 그래디언트를 사용하여 가중치를 업데이트하는 것
- ④ 모든 샘플에 대해 개별적으로 가중치를 업데이트하는 것

❖ Momentum 방법을 사용할 때, 어떤 변수가 이전 그래디언트 값을 사용하는가?

- ① 학습률
- ② 가중치
- ③ 손실 함수
- ④ 속도(velocity)

❖ AdaGrad의 특징은?

- ① 과거의 모든 그래디언트의 제곱을 누적한다.
- ② 속도와 방향을 모두 고려한다.
- ③ 학습률을 일정하게 유지한다.
- ④ 모멘텀만을 사용하여 가중치를 업데이트한다.

Quiz

❖ RMSProp의 핵심 아이디어는?

- ① 과거 그레디언트의 제곱의 이동 평균을 유지한다.
- ② 모든 가중치 업데이트는 동일한 크기로 수행된다.
- ③ 학습률이 시간에 따라 증가한다.
- ④ 경사하강법과 동일한 방식으로 작동한다.

❖ Adam는 어떤 최적화 기법의 조합으로 볼 수 있는가?

- ① RMSProp + AdaGrad
- ② RMSProp + Momentum
- ③ Gradient Descent + Momentum
- ④ AdaGrad + Gradient Descent

❖ 미니배치 경사하강법은 어떤 방식으로 데이터를 처리하는가?

- ① 한 번에 하나의 샘플만 처리한다.
- ② 전체 데이터 세트를 한 번에 처리한다.
- ③ 데이터 세트의 작은 부분집합을 한 번에 처리한다.
- ④ 데이터의 순서를 고려하지 않고 무작위로 처리한다.

Quiz

❖ 어떤 최적화기가 학습률을 동적으로 조정하는 기능을 가지고 있는가?

- ① Gradient Descent
- ② Momentum
- ③ AdaGrad
- ④ Batch Gradient Descent

❖ 최적화기(optimizer)의 학습률을 너무 높게 설정하면 어떤 문제가 발생할 수 있는가?

- ① 모델이 과소적합될 수 있다.
- ② 손실 함수의 최소값에 수렴하기 위해 더 많은 시간이 걸릴 수 있다.
- ③ 손실 함수의 최소값 주변에서 진동할 수 있다.
- ④ 학습이 전혀 진행되지 않는다.

❖ saddle point(안장점)에 빠질 경우, 어떤 최적화기가 이를 해결하는 데 가장 도움이 안될까?

- ① Gradient Descent
- ② Momentum
- ③ Batch Gradient Descent
- ④ AdaGrad

Quiz

- ❖ 학습 데이터의 특징 Z 가 평균 0, 표준 편차 1로 정규화되어야 할 때, 원본 데이터 값 $[2, 8, -1, 4]$ 를 정규화하고 변환된 값을 계산하세요.
- ❖ 학습률이 0.01이고, 현재 가중치가 0.5일 때, 손실 함수의 그래디언트가 0.3으로 계산되었을 때, 다음 단계의 가중치를 계산하십시오.
- ❖ 데이터 배치가 $[1, 3, 5, 7]$ 로 주어졌을 때, 이 데이터의 평균과 표준편차를 계산하십시오.
- ❖ 위에서 계산한 평균과 표준편차를 사용하여 각 데이터 포인트를 정규화하십시오.
- ❖ 위에서 정규화된 데이터에 학습 가능한 매개변수 $\gamma = 2$ 와 $\beta = 0.5$ 를 적용하여 출력을 계산하십시오.