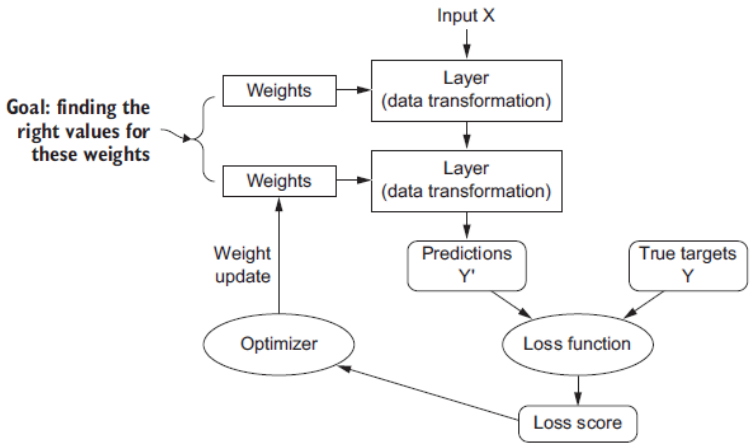
Deep Neural Network

심층신경망의 구조와 학습 과정

1. 네트워크의 학습 절차

심층신경망은 입력층과 출력층 사이에 다수의 은닉층을 포함하여 다양한 비선형적 관계를 학습할 수 있다. 알고리즘에 따라 여러 개의 완전 연결 은닉층을 쌓은 다층 퍼셉트론(Multi-Layer Perceptron; MLP), 이미지와 같은 2차원 데이터 처리에서 뛰어난 성능을 보이며 지역적인 특징을 학습할 수 있는 합성곱 신경망(Convolution Neural Network; CNN) 등이 주요 네트워크로 사용되고 있다. 신경망의 학습은 전체 네트워크의 층과 층 사이에서의 연산에 사용되는 **모든 가중치들의 최적값을 찾는 것**이며 모델에 따라 정의한 LOSS가 충분히 줄어들 때까지 가중치의 업데이트를 반복하는 것이다.



학습의 과정은 다음의 절차를 정해진 횟수만큼 반복함으로써 이루어진다.

1) 배치 사이즈*만큼의 훈련 샘플 (x, y)를 가져와 입력층에 데이터 x를 입력한다.
2) 전체 네트워크를 따라 가중치(weight)를 곱하고 활성함수를 거쳐 입력 데이터를 변환하고 출력 값 $\hat{y}$를 계산한다.
3) 출력된 $\hat{y}$와 실제 y 값의 오차를 사용해 현재 배치 안에서의 평균 LOSS를 계산한다. (MSE 또는 Cross Entropy 등)
4) 출력층부터 입력층까지 역방향으로 되돌아가며 가중치 파라미터에 대한 LOSS의 그래디언트를 계산하고, LOSS가 줄어드는 방향으로 가중치를 업데이트한다.
5) 다음 배치 사이즈만큼의 훈련 샘플을 가져와 반복한다.

A. 파라미터 초기화 (INITIALIZATION)

사전에 학습된 모형이 없을 경우 일반적으로 전체 네트워크의 파라미터(가중치 및 편향)에 랜덤한 초기값을 설정한다. 초기값에 따라 학습의 성능이 크게 좌우될 수 있으므로 주의가 필요하다.

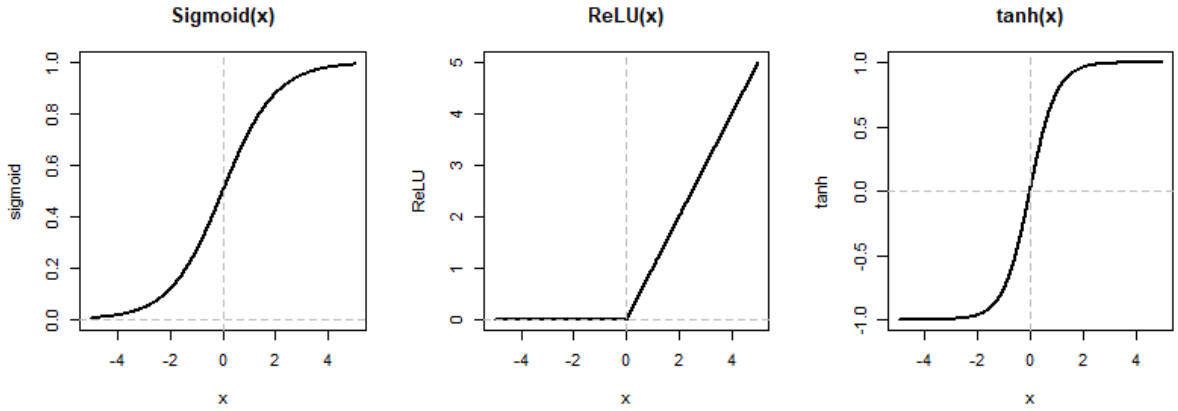
※ **잘못된 초기화**

- 모두 0으로 놓는 경우: 만약 가중치 행렬 W의 초기값을 단순히 모두 0으로 놓는다면 W의 원소들은 전방향 연산에서 출력 값 계산에 사용되고, 이 값을 이용해 역방향으로 그래디언트를 구하는 과정에서 그래디언트가 모두 0이 되어 사라진다. 따라서 이것은 네트워크 학습이 이루어지지 않게 하므로 옳지 않은 방법이다.

- 모두 동일한 값(상수)로 지정하는 경우: 초기값이 동일하면 모든 뉴런이 동일한 출력 값을 내보낼 것이며 모두 동일한 그래디언트 값을 가지게 된다. 따라서 뉴런이 여러 개라도 하나의 뉴런처럼 작동하므로 학습이 잘 되지 않는다.

※ **Random Generator**

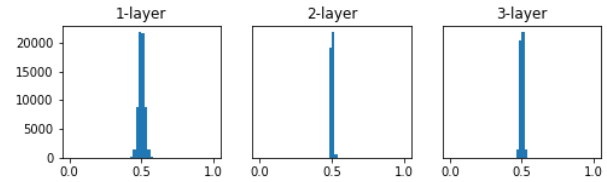
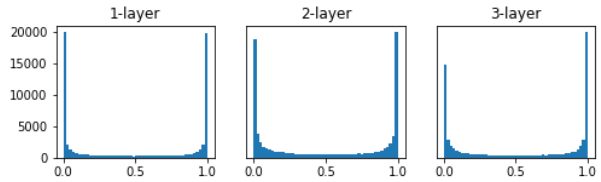
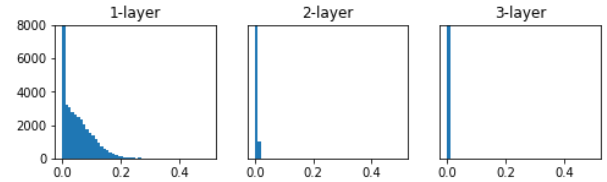
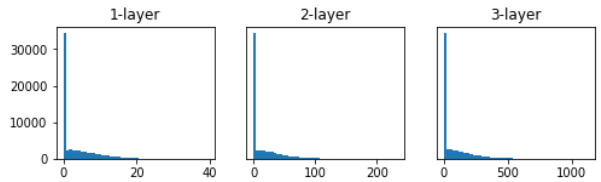
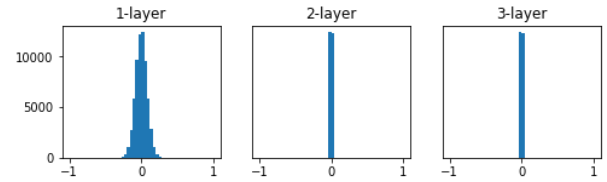
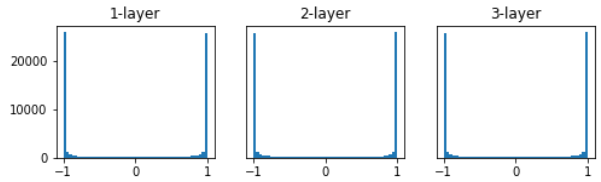
가중치의 초기값을 서로 다른 랜덤한 값으로 구성하기 위해 일반적으로 Uniform, 또는 Normal 분포에서 난수를 생성한다. 주의할 점은 적절한 분산을 설정하여 값의 크기를 적당하게 초기화해야 한다는 것이다. 초기화 분포의 선택은 활성화 함수의 영향을 받는다. 현재 많이 사용되고 있는 활성화함수는 다음과 같다.



$$\text{sigmoid}(x) = \frac{1}{1+\exp(-x)} \qquad \text{ReLU}(x) = \max\{0,\ x\} \qquad \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

만약 활성화함수가 sigmoid일 경우 가중치 초기값의 크기(절댓값)이 너무 크다면 0과 1로 수렴하기 때문에 그래디언트 소실이 발생하게 된다. 활성화함수가 ReLU일 경우, 절댓값이 크다면 음수일 경우 dead ReLU 문제가 발생하고 양수일 경우 그래디언트 폭주(exploding)가 일어난다.

크기가 너무 작은 값으로 초기값을 지정한다면 간단한 신경망에서는 잘 작동하는 편이나 신경망의 깊이가 깊어질수록 문제가 발생한다. 다음 그림에서 상위 신경망으로 갈수록 누적 곱해진 가중치들의 값이 점점 모두 동일한 값을 출력하는 것을 확인할 수 있다. 따라서 이 경우 역시 모든 뉴런의 그래디언트 값이 동일해지기 때문에 학습이 잘 이루어지지 않는다.

Normal(mean=0, sd=0.1)	Normal(mean=0, sd=1)
sigmoid	
	
ReLU	
	
tanh	
	

따라서 적절한 크기의 서로 다른 값으로 가중치의 초깃값을 설정하는 것이 좋다. 가중치를 초기화하는 방법은 지속적으로 연구가 진행되고 있는 분야로, 현재는 랜덤한 값을 생성할 Uniform 또는 Normal 분포의 **분산**을 각 층의 입력과 출력 노드의 개수에 따라 결정하도록 하는 Xavier (2010)이나 He (2015) 방법을 주로 사용한다.

- Xavier (Glorot)

$$W_i \sim Uniform\left[-\sqrt{\frac{2}{n_{in}+n_{out}}},\ \sqrt{\frac{2}{n_{in}+n_{out}}}\right]\cdot\sqrt{3} \quad or \quad W_i \sim Truncated\ Normal\left(0,\ \sqrt{\frac{2}{n_{in}+n_{out}}}\right)$$

Xavier Glorot가 제안한 방법은 깊은 신경망 모형에서 그래디언트의 소실 및 폭주 문제를 해결하기 위해 만들어졌다. Xavier는 기존의 초기값 생성 분포에서 난수를 생성할 경우 층이 깊어짐에 따라 sigmoid 또는 softsign 등의 비선형 활성화 값의 분산이 점점 작아지는 것을 발견하였다. 또한 가중치 업데이트를 위해 순전파와 역전파를 반복하므로 각 층 사이의 가중치의 분산이 입력층의 노드의 개수뿐만 아니라 출력 층의 노드의 개수에도 영향을 받음을 발견하였다. 따라서 순전파 시 가중치의 분산의 합과 역전파의 가중치의 분산의 합을 모두 1으로 하기 위해 i번째 층과 i+1번째 층 사이의 가중치 분산 $\mathrm{Var}(W_i) = 2/(n_i + n_{i+1})$이 되도록 하였다. 또한 Uniform 분포에서 분산을 모두 1으로 맞추기 위한 정규화 상수 $\sqrt{3}$을 곱하는 방법을 도입하였다.

Xavier 방법은 이렇게 간단한 발견으로 sigmoid 또는 tanh, softsign 등의 비선형함수에서 효과적인 결과를 나타냈으나 ReLU 함수에서 사용 시 출력 값이 0으로 수렴하게 되는 문제를 해결하지 못했다.

- He

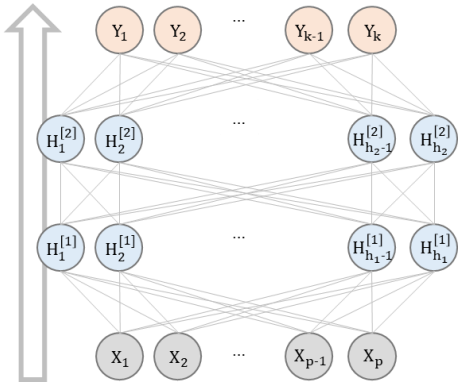
$$W_i \sim Uniform\left[-\sqrt{\frac{2}{n_{in}}},\ \sqrt{\frac{2}{n_{in}}}\right]\cdot\sqrt{3} \quad or \quad W_i \sim Truncated\ Normal\left(0,\ \sqrt{\frac{2}{n_{in}}}\right)$$

ReLU를 사용한 모형에서의 추정이 잘 되지 않는 점을 해결하기 위해 Kaiming He가 제안한 좀더 로버스트한 초기화 방법으로 Glorot과 유사하지만 뉴런의 출력 사이즈를 고려하지 않는다. ReLU가 0 이하의 신호를 제거하기 때문에 줄어드는 분산을 유지하기 위해 분산을 좀 더 크게 설정하는 방법이라고 할 수 있다.

B. 출력 값 계산 및 오차 평가 (FORWARD)

파라미터들의 초기값이 주어지면 각 훈련 샘플을 네트워크에 주입하고 연속되는 각 층마다 출력을 계산한다. 각 층에서는 $\text{output} = f(W \cdot \text{input} + b)$의 연산이 이루어진다. 전체 네트워크를 따라 하나의 입력 데이터(하나의 관측치)에 대한 하나의 최종 출력 값을 계산할 수 있다. 또한 계산된 출력 값을 이용해 실제 출력과의 오차 및 Loss를 계산한다.

입력층과 2 개의 은닉층, 출력층으로 이루어진 분류모형을 예시로 사용해보자. MLP 에서 각 층과 층사이의 노드는 완전 연결되어 있다.



INPUT – HIDDEN[1] – HIDDEN[2] – OUTPUT

m	배치 사이즈* (한번의 연산에 사용되는 입력 데이터의 수)
p	입력층의 노드 개수 (=입력 특징의 수)
$\mathbf{h_1}$	은닉층 1 의 노드 개수
$\mathbf{h_2}$	은닉층 2 의 노드 개수
k	출력층의 노드 개수(=출력 클래스의 수)

* 학습 한번에 모든 데이터를 사용할 수도 있지만 한번에 일부 데이터(m 개)만을 사용해 적합하는 미니배치 방법론을 주로 많이 사용하므로 이 방법론으로 연산 과정을 설명하였다. 배치 사이즈에 대한 자세한 설명은 뒤에서 다시 다루기로 한다.

i. INPUT – HIDDEN[1]

$$X_{batch} \cdot W^{[1]} + b^{[1]} = z^{[1]}, \qquad a^{[1]} = activation(z^{[1]})$$

$$\begin{bmatrix} x_{1,1} & \cdots & x_{1,p} \\ \vdots & & \\ x_{m,1} & \cdots & x_{m,p} \end{bmatrix} \begin{bmatrix} w_{1,1} & \cdots & w_{1,h_1} \\ \vdots & & \\ w_{p,1} & \cdots & w_{p,h_1} \end{bmatrix}^{[1]} + \begin{bmatrix} b_1 \\ \vdots \\ b_{h_1} \end{bmatrix}^{[1]\prime} = \begin{bmatrix} z_{1,1} & \cdots & z_{1,h_1} \\ \vdots & & \\ z_{m,1} & \cdots & z_{m,h_1} \end{bmatrix}^{[1]} \xrightarrow{f} \begin{bmatrix} a_{1,1} & \cdots & a_{1,h_1} \\ \vdots & & \\ a_{m,1} & \cdots & a_{m,h_1} \end{bmatrix}^{[1]}$$

$$(m,p) \times (p,h_1) \qquad (1,h_1) \qquad (m,h_1)$$

ii. HIDDEN[1] – HIDDEN[2]

$$a^{[1]} \cdot W^{[2]} + b^{[2]} = z^{[2]}, \qquad a^{[2]} = activation(z^{[2]})$$

$$\begin{bmatrix} a_{1,1} & \cdots & a_{1,h_1} \\ \vdots & & \\ a_{m,1} & \cdots & a_{m,h_1} \end{bmatrix}^{[1]} \begin{bmatrix} w_{1,1} & \cdots & w_{1,h_2} \\ \vdots & & \\ w_{h_1,1} & \cdots & w_{h_1,h_2} \end{bmatrix}^{[2]} + \begin{bmatrix} b_1 \\ \vdots \\ b_{h_2} \end{bmatrix}^{[2]\prime} = \begin{bmatrix} z_{1,1} & \cdots & z_{1,h_2} \\ \vdots & & \\ z_{m,1} & \cdots & z_{m,h_2} \end{bmatrix}^{[2]} \xrightarrow{f} \begin{bmatrix} a_{1,1} & \cdots & a_{1,h_2} \\ \vdots & & \\ a_{m,1} & \cdots & a_{m,h_2} \end{bmatrix}^{[2]}$$

$$(m,h_1) \times (h_1,h_2) \qquad (1,h_2) \qquad (m,h_2)$$

iii. HIDDEN[2] – OUPTUT

$$a^{[2]} \cdot W^{[3]} + b^{[3]} = z^{[3]}, \qquad \hat{p} = softmax(z^{[3]}) = \frac{\exp(z^{[3]})}{\sum \exp(z^{[3]})}$$

$$\begin{bmatrix} a_{1,1} & \cdots & a_{1,h_2} \\ \vdots & & \\ a_{m,1} & \cdots & a_{m,h_2} \end{bmatrix}^{[2]} \begin{bmatrix} w_{1,1} & \cdots & w_{1,k} \\ \vdots & & \\ w_{h_2,1} & \cdots & w_{h_2,k} \end{bmatrix}^{[3]} + \begin{bmatrix} b_1 \\ \vdots \\ b_k \end{bmatrix}^{[3]\prime} = \begin{bmatrix} z_{1,1} & \cdots & z_{1,k} \\ \vdots & & \\ z_{m,1} & \cdots & z_{m,k} \end{bmatrix}^{[3]} \xrightarrow{softmax} \begin{bmatrix} \frac{\exp(z_{1,1})}{\sum_i \exp(z_{1,i})} & \cdots & \frac{\exp(z_{1,k})}{\sum_i \exp(z_{1,i})} \\ \vdots & & \\ \frac{\exp(z_{m,1})}{\sum_i \exp(z_{m,i})} & \cdots & \frac{\exp(z_{m,k})}{\sum_i \exp(z_{m,i})} \end{bmatrix}$$

$$(m,h_2) \times (h_2,k) \qquad (1,k) \qquad (m,k)$$

iv. LOSS

입력한 관측치 하나당 정의되는 LOSS 를 계산하고 미니배치 안에서 m 개 관측치의 평균 LOSS (E)를 구한다.

$$\text{관측치 i 의 LOSS}_{\text{i}} = -\sum_{k} I(y_i \in C_k) \cdot \log \hat{p}_i\,, \qquad E = \frac{1}{m}\sum_{i=1}^{m} LOSS_i$$

C. 경사 하강 및 역전파 (BACKWARD) → W의 Update

▶ 경사 하강법 (Gradient Descent)

많은 머신러닝 알고리즘은 경사 하강법을 이용해 목적 함수를 최소화하는 방향으로 파라미터들을 반복적으로 업데이트 한다. 심층 신경망에서 전체 네트워크의 LOSS 함수(E)는 파라미터 W(개별 가중치들의 집합)에 대해 미분 가능한 함수의 형태이다. 이를 이용해 각 층의 가중치들에 대한 Loss의 미분, 즉 그래디언트를 계산할 수 있다. Loss의 계산에 포함된 어떤 가중치 w에 대해 그래디언트를 구하면 w로부터 단위(Δw)당 변화량을 계산할 수 있고, 이것이 감소하는 방향(descent)으로 가중치를 업데이트한다.

$$w_{ij}^{*} = w_{ij} - \eta \cdot \frac{\partial E}{\partial w_{ij}}$$

여기서 η는 하이파라미터인 학습률(learning rate)에 의해 결정되는 조정 크기이다. 학습률이 너무 크면 현재 위치에서 보폭을 매우 크게 하는 것으로 최적 값으로 수렴하지 않고 더 큰 값으로 발산할 가능성이 있으며, 학습률이 매우 작을 경우 알고리즘이 수렴하기 위해 반복을 많이 진행해야 하므로 시간이 많이 걸린다.

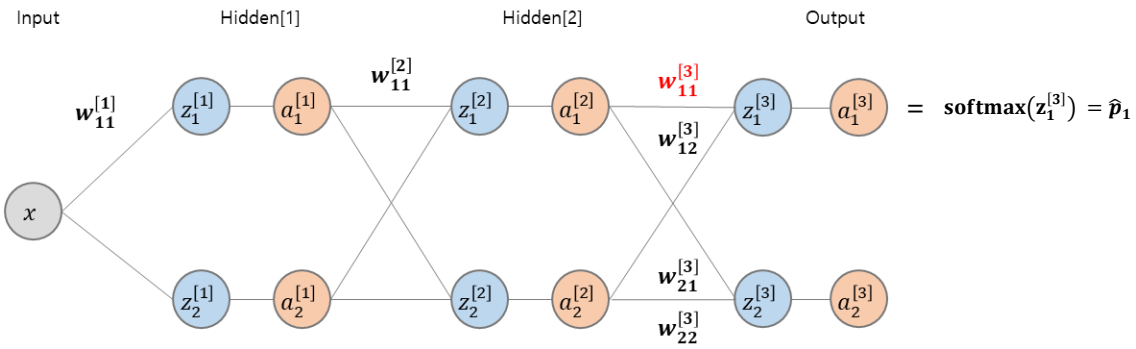
▶ 역전파 (Back Propagation)

심층신경망에는 층과 층 사이에 매우 많은 가중치 파라미터가 존재하고, 학습 한번 당 각 층의 모든 가중치를 경사 하강법을 이용해 업데이트해야 한다. 이 연산을 빠르고 효율적으로 하는 방법으로 역전파(Back Propagation) 알고리즘을 사용한다.

역전파는 앞에서 입력 데이터에 대해 전방향 연산(Forward Propagation)을 통해 계산된 오차를 네트워크에 존재하는 각각의 노드에 역으로 전파하는 과정이라고 할 수 있다. 마지막 은닉층에서의 각 뉴런들의 가중치에 대해 Loss의 편미분을 계산하고, 다시 이전 은닉층의 뉴런의 가중치에 대한 해당 그래디언트의 편미분을 계산하는 과정을 반복하여 입력 층에 도달할 때까지 역으로 계산한다. 이 역방향 과정은 오차 그래디언트를 후방으로 전파한다고 하여 '역전파'라고 한다.

k-1번째 층(입력층)의 i번째 노드와 k번째 층(출력층)의 j번째 노드를 연결하는 가중치 w를 $w_{ij}^{[k]}$라고 한다. FORWARD 연산과 동일하게 입력층과 출력층 사이에 2개의 은닉층이 있는 신경망을 예시로 사용하였다.

$\mathbf{h_1}$	$\mathbf{h_2}$	k
2	2	2



FORWARD 스텝에서 입력 x에 대해 마지막 층과 층 사이에서는 다음의 연산이 일어난다.

$$z_1^{[3]} = a_1^{[2]} w_{11}^{[3]} + a_2^{[2]} w_{21}^{[3]}, \qquad z_2^{[3]} = a_1^{[2]} w_{12}^{[3]} + a_2^{[2]} w_{22}^{[3]}$$

$$a_1^{[3]} = f\left(z_1^{[3]}\right), \qquad a_2^{[3]} = f\left(z_2^{[3]}\right)$$

마지막 활성화를 거쳐 계산된 출력, 즉 예측 값과 실제 값을 사용해 LOSS E를 구하고 나면 역전파 알고리즘을 시작한다. 전체 가중치의 업데이트를 위해 먼저 출력층부터 마지막 은닉층으로 출발한다. $w_{11}^{[3]}$을 업데이트하기 위한 $\partial E/\partial w_{11}^{[3]}$의 계산은 Chain Rule에 의해 다음과 같이 계산할 수 있다.

$$\frac{\partial E}{\partial w_{11}^{[3]}} = \frac{\partial E}{\partial a_1^{[3]}} \cdot \frac{\partial a_1^{[3]}}{\partial z_1^{[3]}} \cdot \frac{\partial z_1^{[3]}}{\partial w_{11}^{[3]}}$$

$$\frac{\partial E}{\partial a_1^{[3]}} = \frac{\partial}{\partial a_1^{[3]}} E\left(a_1^{[3]}, y_1\right),$$

$$\frac{\partial a_1^{[3]}}{\partial z_1^{[3]}} = f\left(z_1^{[3]}\right) \cdot f'\left(z_1^{[3]}\right) = a_1^{[3]} f'\left(f^{-1}\left(a_1^{[3]}\right)\right)$$

$$\frac{\partial z_1^{[3]}}{\partial w_{11}^{[3]}} = a_1^{[2]}$$

이 과정에서 E의 가중치에 대한 미분 값이 결국 역전파의 출발 노드의 활성함수 값과 도착 노드의 활성함수 값과 실제 출력 값으로만 표현 가능하다는 것을 알 수 있다. 계산된 그래디언트 $\frac{\partial E}{\partial a_1^{[3]}} \cdot \frac{\partial a_1^{[3]}}{\partial z_1^{[3]}}$을 $\delta_1^{[3]}$라고 정의하고 이것을 전 단계로 전파하여 마지막 은닉층으로부터의 가중치 $w_{11}^{[3]}$와 $w_{21}^{[3]}$을 업데이트한다.

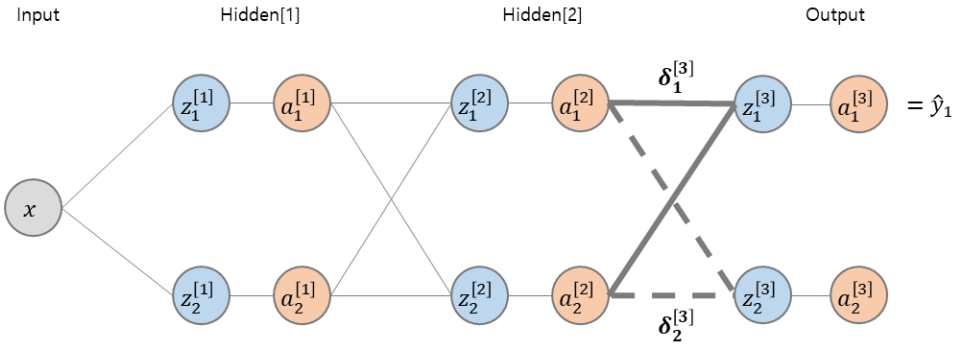
$$w_{11}^{[3]} \leftarrow w_{11}^{[3]} - \eta \cdot \delta_1^{[3]} \cdot a_1^{[2]}$$

$$w_{21}^{[3]} \leftarrow w_{21}^{[3]} - \eta \cdot \delta_1^{[3]} \cdot a_1^{[2]}$$

마찬가지로, $\delta_2^{[3]} = \frac{\partial E}{\partial a_2^{[3]}} \cdot \frac{\partial a_2^{[3]}}{\partial z_2^{[3]}}$을 이용해 $w_{12}^{[3]}$와 $w_{22}^{[3]}$을 업데이트한다. 이와 같이 역전파를 통해 전달되는 값이 $\delta_1^{[3]}$과 $\delta_2^{[3]}$ 뿐이더라도 해당 층에서 관련된 가중치를 모두 업데이트할 수 있다.

$$w_{12}^{[3]} \leftarrow w_{12}^{[3]} - \eta \cdot \delta_2^{[3]} \cdot a_2^{[2]}$$

$$w_{22}^{[3]} \leftarrow w_{22}^{[3]} - \eta \cdot \delta_2^{[3]} \cdot a_2^{[2]}$$



다음으로 이전 단계의 가중치 $w_{11}^{[2]}$을 업데이트하기 위해서는 $\frac{\partial E}{\partial w_{11}^{[2]}}$을 계산해야 한다.

$$\frac{\partial E}{\partial w_{11}^{[2]}} = \frac{\partial E}{\partial a_1^{[2]}} \cdot \frac{\partial a_1^{[2]}}{\partial z_1^{[2]}} \cdot \frac{\partial z_1^{[2]}}{\partial w_{11}^{[2]}}$$

여기서 첫번째 편미분인 $\frac{\partial E}{\partial a_1^{[2]}}$은 다음과 같이 분할 가능하고 이것은 $a_1^{[2]}$가 전체 E 계산에 영향을 미치는 모든 경로의 합이라고 할 수 있다.

$$\frac{\partial E}{\partial a_1^{[2]}} = \frac{\partial E_1}{\partial a_1^{[2]}} + \frac{\partial E_2}{\partial a_1^{[2]}} = \frac{\partial E_1}{\partial z_1^{[3]}} \cdot \frac{\partial z_1^{[3]}}{\partial a_2^{[2]}} + \frac{\partial E_2}{\partial z_2^{[3]}} \cdot \frac{\partial z_2^{[3]}}{\partial a_1^{[2]}} = \delta_1^{[3]} w_{11}^{[3]} + \delta_2^{[3]} w_{12}^{[3]}$$

두번째, 세번째 편미분의 계산 역시 이전 과정과 동일한 과정을 거쳐 $w_{11}^{[2]}$의 업데이트는 다음과 같게 된다.

$$w_{11}^{[2]} \leftarrow w_{11}^{[2]} - \delta_1^{[2]} a_1^{[1]}$$

이렇게 역으로 오차의 그래디언트를 연달아 전파하여 입력층까지 연결되는 모든 가중치들을 업데이트할 수 있다.

D. ITERATION

네트워크의 모든 가중치가 업데이트되고 나면 다시 FORWARD 스텝으로 돌아가 **다음으로 입력되는 미니배치**의 샘플에 대한 출력 값 및 오차를 정방향으로 계산하고, BACKWARD 스텝에서 그래디언트 계산 및 역전파를 거쳐 가중치를 다시 업데이트한다. 즉 학습 과정은 LOSS가 충분히 작아질 때까지, 또는 정해진 반복 수에 도달할 때까지 정방향과 역방향 연산을 번갈아 진행하며 파라미터의 값을 조정하는 과정이라고 할 수 있다.

이 반복적인 연산을 이용한 학습 과정은 다수의 완전 연결 층으로만 이루어진 MLP 모형뿐만 아니라 합성곱 계층과 풀링 계층을 포함하는 CNN의 구조에서도 수학적으로 동일하게 이루어진다.

2. 배치 사이즈와 확률적 경사 하강법(SGD)

앞에서 연산 한번에 처리하는 샘플의 개수는 조정할 수 있으며, 이것은 곧 '배치 사이즈'로서 네트워크의 학습률, 뉴런의 수, 활성화 함수의 선택 등과 함께 학습에 영향을 미치는 하이퍼파라미터 중 하나이다. 배치 사이즈가 전체 훈련 샘플의 수 N일 경우 역전파 단계에서는 배치 경사 하강법, 1인 경우 확률적 경사 하강법, $1 < m < N$인 경우 미니배치 경사 하강법을 이용한다. 신경망 학습에서의 컴퓨터 연산은 주로 행렬 연산으로, 배치 사이즈는 학습의 속도 및 안정성에 영향을 미친다.

A. 배치 경사 하강법 (Batch Gradient Descent)

X가 N개의 sample로 이루어진 전체 훈련 셋이라면 배치 경사 하강법은 모두 사용해서 그래디언트 벡터를 계산한다. 또한 Loss의 계산 시 전체 훈련 셋에 포함된 각각의 샘플에 대한 개별 오차를 전체 훈련 셋에 걸쳐 평균을 계산한다. 일괄적으로 처리한다고 하여 이 알고리즘을 **배치 경사 하강법**이라고 한다. 만약 훈련 데이터 셋의 크기가 매우 크면 속도가 매우 떨어지며, 복잡한 모형일수록 심한 속도 저하가 발생한다. 그러나 점진적으로 최솟값으로 수렴하고 알고리즘이 안정적이다.

B. 확률적 경사 하강법 (Stochastic Gradient Descent)

배치 경사 하강법의 속도 저하 문제가 발생하지 않는 정반대의 방법인 확률적 경사 하강법은 매 스텝에서 하나의 샘플을 랜덤으로 선택해 그 샘플에 대한 그래디언트를 계산한다. 매 반복에서 매우 적은 데이터만을 처리하기 때문에 알고리즘이 훨씬 빠르며 메모리 소모 비용 또한 적다. 그러나 이 알고리즘은 배치 경사 하강법보다 훨씬 불안정하며, Loss가 최소에 다다를 때까지 부드럽게 감소하지 않고 위아래로 요동치면서 평균적으로 감소한다. 만약 Loss가 전역과 지역 최솟값을 모두 가지는 형태라면 배치 방법은 로컬 값에 빠질 경우 빠져나오기 힘들지만 SGD는 확률적으로 로컬 값에서 벗어날 수 있도록 도와준다. 따라서 배치 방법보다 전역 최솟값을 빠르게 찾을 가능성이 높지만, 학습을 멈추기까지 찾지 못하고 불안정하게 종료될 수 있다. 일반적으로 학습의 한 epoch에서 훈련 셋의 샘플의 개수 n번 되풀이되도록 한다. 샘플을 무작위로 선택하기 때문에 어떤 샘플은 한 epoch에서 여러 번 선택되거나 아예 선택되지 않을 수 있다. 따라서 훈련 셋 안의 각각의 샘플들이 서로 동질적이라면 계산 비용을 효과적으로 절약하면서도 좋은 파라미터의 추정치를 찾을 수 있으나,

그렇지 않은 경우 노이즈가 발생한다. 각 epoch마다 모든 샘플을 사용하게 하려면 훈련 셋을 랜덤하게 섞은 후(shuffle) 차례대로 하나씩 선택하게 할 수 있다.

C. 미니배치 경사 하강법 (Mini-Batch Gradient Descent)

미니배치는 배치와 SGD의 결합으로 각 스텝에서 전체 훈련 셋에서 미니배치라고 부르는 임의의 작은 샘플 셋에 대해 그래디언트를 계산한다. SGD에 비해 얻는 주요 장점은 행렬 연산에 최적화된 GPU에서 얻는 성능 향상이다. 미니배치 사이즈를 어느 정도 크게 하면 파라미터의 공간에서 SGD보다 덜 불규칙하게 움직이며 SGD보다 최솟값에 더 가까이 도달하게 될 것이다. 따라서 연산량을 줄이면서도 어느 정도 안정적인 학습을 위하여 일반적으로 16, 32, 64, 128 등의 배치 사이즈를 이용한 미니배치 학습을 주로 사용한다.

SGD와 미니배치 방법에서 가중치의 변동을 줄이기 위한 방법으로 학습률을 점진적으로 감소시키는 방법이 있다. 초기 학습 단계에서는 학습률을 크게 해 수렴을 빠르게 하고 지역 최솟값에 빠지지 않게 하며, 점차 작게 줄여서 전역 최솟값에 도달하도록 하는 것이다.

cf. 배치 정규화(Batch Normalization, BN) (2015)

학습 동안 이전 층의 파라미터가 변함에 따라 각 층에 들어오는 입력의 분포가 변화하는 문제(내부 공변량의 변화)를 발견하고 이를 해결하기 위한 기법으로 개발되었다. 입력의 분포가 학습할 때마다 변하면서 가중치가 엉뚱한 방향으로 갱신될 문제가 발생할 수 있다. 일반적인 DNN에서, 각 층의 노드 하나에서는 활성화 함수를 통과하기 전 계산된 $\mathbf{x}^{(i)}$들을 미니배치 단위로 표준화한 후 각 층에서 두 개의 새로운 파라미터 γ (scale), β (bias)로 결과 값의 스케일을 조정하고 이동시킨다. 각 층의 연산 $x = Wu + b$에 배치 정규화를 적용하는 형태로 표현할 수 있다. 이는 곧 $Wu + b$를 정규화 하는 것이 되므로, b의 영향이 사라지기 때문에 b를 무시한다. 따라서 이 연산으로 네트워크는 층마다 현재의 미니배치에서의 평균과 표준편차를 평가하며, 최적의 $\boldsymbol{\gamma}, \boldsymbol{\beta}$을 학습한다. 이를 통해 학습률을 높게 설정해도 그래디언트가 폭주/소실되거나 나쁜 local minima에 빠지는 문제가 없어 학습 속도가 개선되며, 가중치 초기값 선택의 의존성이 적어진다. 또한 과대적합 위험을 줄이고 그래디언트 소실 문제를 해결할 수 있다.

$B = \{x^{(1)}, \ldots, x^{(m_B)}\}$: 활성화 함수를 통과하기 전 현재 미니배치에서 계산된 모든 출력 값의 집합

1. $\mu_B = \frac{1}{m_B}\sum_{i=1}^{m_B} x^{(i)}$
2. $\sigma_B^2 = \frac{1}{m_B}\sum_{i=1}^{m_B}\left(x^{(i)} - \mu_B\right)^2$
3. $\hat{x}^{(i)} = \frac{x^{(i)} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$
4. $z^{(i)} = \gamma\hat{x}^{(i)} + \beta$

만약 현재 층에서 활성화 직전 계산된 (미니배치 샘플 수, 출력 채널의 수) 크기의 행렬이 있다면 배치 정규화에서 출력 채널 하나당 평균과 분산을 계산한다. 즉 배치 정규화는 각 층의 노드 하나하나에서 독립적으로 진행되는 것이다. 층에서 출력되는 채널마다 독립적으로 사용되는 γ와 β을 각각 학습한다.

합성곱 층에서는 어떤 위치에 필터를 적용하든지 같은 파라미터 값을 공유하기 때문에 배치 정규화가 조금 다르게 이루어진다. 일반적인 배치정규화와 동일하다면 미니 배치의 크기를 m, 하나의 출력 특성맵의 크기를 (p, q)라고 할 때 (p, q)의 각 원소에 대해 각각 미니배치 정규화가 진행될 것이다. 그러나 합성곱 층에서의 (p, q)를 하나의 단위로 보고 (m, p, q)에 대해 각각 하나의 평균과 분산을 구하여 정규화한다.

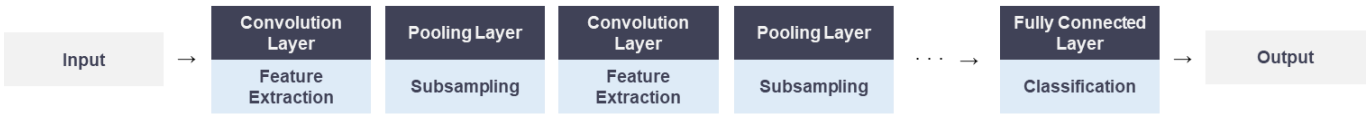
1. INTRODUCTION

이미지 인식 및 분류에서 특히 뛰어난 성능을 보이는 합성곱 신경망(CNN)은 입력 데이터의 모든 특징을 한번에 학습하지 않고 부분적인 특징을 여러 구간에서 학습하고자 하는 알고리즘으로, CNN을 구성하는 층은 크게 다음과 같이 세 종류의 층으로 구분할 수 있다.

(1) 합성곱 계층(Convolution Layer) (2) 풀링 계층(Pooling Layer) (3) 완전 연결 계층(Fully Connected Layer)

먼저 합성곱 계층에서는 일정한 사이즈로 입력 데이터를 지역적으로 구분한 후 다수의 필터와 곱 연산을 통해 특성 맵을 추출한다. 풀링 계층에서는 추출된 특성 맵을 지정한 풀링 사이즈로 구분하여 각 구역당 최대값(Max Pooling) 또는 평균값(Average Pooling) 하나만을 남기는 방식으로 특성 맵의 크기를 축소한다. 마지막 합성곱(및 풀링) 층에서 출력된 특성 맵을 일렬로 변형하고 이후 완전 연결 층에서 출력 클래스들과의 완전 연결 연산을 통한 예측 및 분류가 이루어진다.

일반적으로 분류 문제를 위한 CNN은 다음과 같이 합성곱-풀링 계층을 반복하여 입력 데이터에 대한 특징을 추출하고 난 뒤 마지막 단계에 1개 이상의 완전 연결 층을 두어 클래스 분류가 이루어지도록 하는 구조를 가지고 있다.



▷ 합성곱 층의 2가지 핵심 하이퍼 파라미터

(1) 입력으로부터 필터를 적용할 패치의 크기(kernel size)
(2) 특성 맵의 출력 깊이: 합성곱 연산에 사용할 필터의 수(filters)

CNN의 학습 목적은 층마다 설정된 개수의 필터 가중치를 학습함으로써 클래스를 구별하는 데 영향을 미치는 중요한 특징을 뽑아낼 수 있도록 하는 것이다. 각 필터의 크기를 크게 설정한다면 한번에 넓은 구역에서 클래스를 구별하는 특징을 찾도록 하는 것이고 작게 설정한다면 좀더 작은 구역에서의 특징을 찾아내도록 하는 것이다. 따라서 모형에서 기대하는 바에 따라 필터를 적용할 패치의 크기라는 하이퍼파라미터를 다르게 설정할 수 있다.

입력 데이터가 이미지라고 할 때, 하나의 필터는 이미지의 전 구역을 돌아다니며 동일한 연산을 진행한다. 즉 필터를 공유 파라미터로 사용하기 때문에 기존의 완전 연결로만 이루어진 신경망에 비해 추정 파라미터의 수를 대폭 줄일 수 있다.

개별 필터를 구성하는 각 원소의 초기값은 일반적인 심층신경망과 같이 랜덤하게 결정하고 반복학습에 따라 값을 업데이트한다. 이미지 분류 모형을 구성하는 경우 'Alexnet', 'VGG' 등과 같이 대규모 이미지 분류 경연대회에서 우승한 사전학습된 모형의 합성곱 층을 불러와 초기 단계부터 부분적 특징을 효과적으로 학습하고, 여기에 예측을 위한 완전 연결 층을 쌓는 형태로 사용하기도 한다.

합성곱 신경망은 2D의 이미지 뿐 아니라 1D의 시퀀스 형태의 데이터에도 적용이 가능하다. 대표적으로 시계열 시퀀스나 문장, 즉 단어들의 시퀀스 형태가 있다. 이 경우 적용하는 필터 역시 1D로 적용되며 연산 과정은 2D CNN과 동일하다.

2. CNN 구조의 도식화

▶ 2D CNN 예시 그림

CNN의 각 층에서의 연산을 그림으로 표현하기 위해 먼저 이미지처럼 2D의 형태를 예시로 사용하였다.

하나의 입력 이미지의 넓이와 높이가 각각 28픽셀로 이루어져 있다고 하자. 컬러 이미지의 경우 각 픽셀에는 RGB, 3가지 차원의 특징이 존재하므로 입력의 깊이, 즉 입력 특징(feature)의 수는 3이다. 흑백 이미지의 경우 픽셀 별로 1가지 음영 값 만을 가지므로 깊이는 1이 된다. 이미지가 분류될 클래스는 10개가 존재한다고 할 때, 다음과 같이 두 개의 합성곱 및 풀링 계층을 거치는 CNN 모형의 구조를 예시로 고려하였다.

▷ 하이퍼파라미터 설정

합성곱 층	필터의 개수	필터 사이즈
1	32	(3, 3)
2	64	(3, 3)

※ **패딩**(Padding)
합성곱 연산에서 입력과 출력의 사이즈가 동일하도록 입력 데이터의 가장자리를 0으로 채우는 것을 말한다.
패딩을 하지 않을 경우 합성곱 층을 거칠수록 특징 맵의 크기는 줄어든다. 본 예시에서는 패딩을 하지 않는 것('valid')으로 가정한다.

※ **스트라이드**(Stride)
필터가 한번에 이동하는 보폭을 의미한다. **합성곱** 층에서는 **스트라이드를 1**로 하여 한 칸씩 이동하며, **풀링** 층에서는 스트라이드를 필터 사이즈와 동일하게 설정하여 **서로 겹치지 않도록**(No Overlapping)하는 것이 기본 설정이며 일반적으로 사용된다.

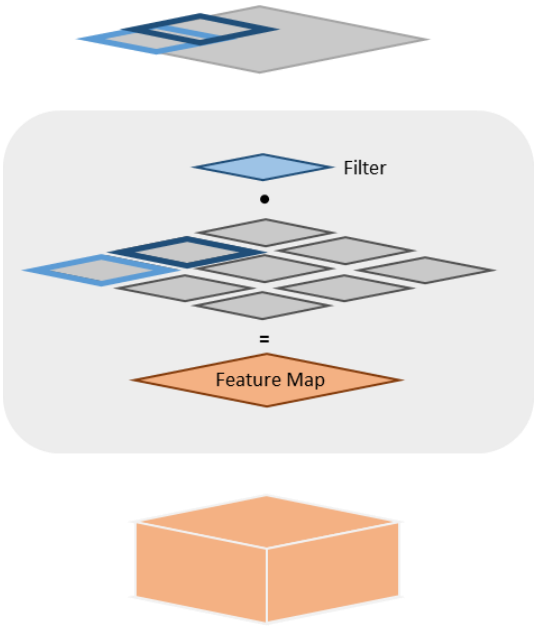
▷ 구조 요약

Layer	Output Shape	Parameters
Input	(None, 28, 28, 3)	
Conv2D(filters=**32**, kernel_size=**(3,3)**, activation='**relu**', input_shape=(28,28,3))	(None, 26, 26, 32)	896
MaxPooling2D(pool_size=(2,2))	(None, 13, 13, 32)	0
Conv2D(filters=**64**, kernel_size=**(3,3)**, activation='**relu**')	(None, 11, 11, 64)	18496
MaxPooling2D(pool_size=(2,2))	(None, 5, 5, 64)	0
Flatten	(None, 1600)	0
Dense(units=10, activation='**softmax**')	(None, 10)	16010

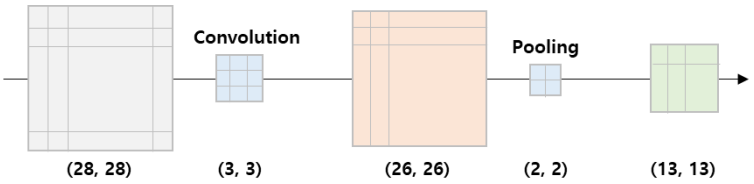
cf. Layer 별 옵션 설정: [Keras] https://keras.rstudio.com/reference/index.html

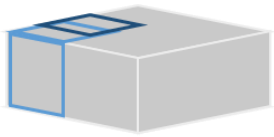
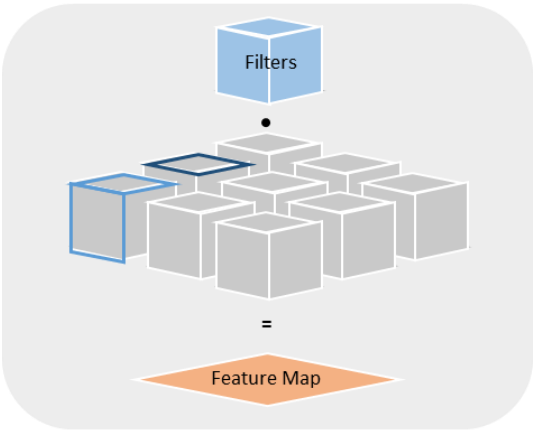
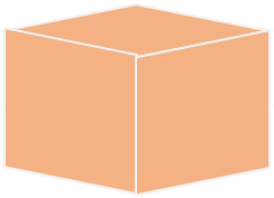
- 합성곱(2D) 층에서 입력의 높이 및 너비가 n이고 필터의 높이 및 너비가 l이라고 할 때, 패딩을 하지 않고 (n, n) 이미지(또는 특성 맵)에 (l, l) 필터를 적용한 결과 출력되는 특징 맵의 크기는 $(n-l+1, n-l+1)$이 된다.

- 합성곱 층의 파라미터의 개수는 (입력 채널 수)*(필터 넓이)*(필터 높이)*(출력 채널 수)의 가중치(weight)와 출력 채널의 수와 동일한 개수의 편향(bias)을 더한 만큼 존재한다.
- 완전 연결 층의 파라미터는 (입력 층 노드 수)*(출력 층 노드 수)의 가중치와 출력 노드 수만큼의 편향으로 이뤄진다.

Convolution Layer 1		Shape
	(1) 입력 이미지는 한번에 필터와 동일한 크기의 윈도우로 구분된다. 예시의 경우 필터 사이즈가 (3, 3)이므로 패딩을 하지 않고 스트라이드가 1인 경우 총 26*26개의 구역이 생긴다. (앞서 구조에서는 3차원으로 입력 채널을 가정하였으나 이해를 돕기 위해 그림에서는 입력 채널 1개에 대한 그림으로 표현하였다.) (2) 하나의 필터는 이미지를 슬라이딩하며 각 구역당 점곱을 통해 하나의 값을 계산하고 각 값에 활성화 함수를 적용하여 26*26 크기의 특성 맵 하나를 형성한다. (3) 총 32개의 필터에 대해 각각 26*26 크기의 특성 맵이 생성되므로 출력되는 특징 맵은 (26, 26, 32) 차원의 배열이 된다.	(26, 26, 1) (26, 26, 32)
Pooling Layer 1		**Shape**
	(4) 풀링 사이즈는 2로 최대 풀링할 경우 출력된 각 특징 맵의 2칸 당 큰 값만을 추출하여 특징 맵의 사이즈를 축소한다. (특징 맵의 개수는 동일하게 유지된다.) 이 층에서는 파라미터를 사용한 연산이 일어나지 않는다.	(13, 13, 32)

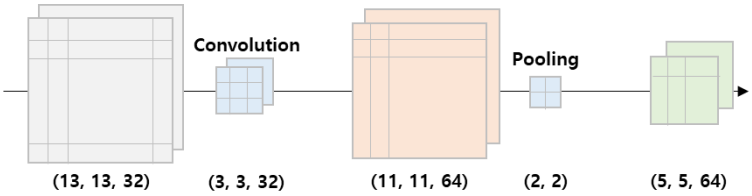
* 필터 하나의 연산

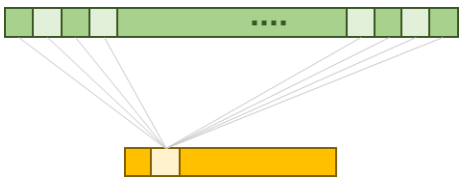


Convolution Layer 2		Shape
	(5) 이전 층에서 출력된 특징 맵이 두번째 합성곱 층의 입력 특징 맵이 된다. 첫번째 합성곱 층과 같은 과정으로, 필터 사이즈와 동일한 크기로 특징 맵의 구역이 구분되며 채널의 깊이는 유지된다.	(13, 13, 32)
	(6) 각 필터는 입력되는 특징 맵과 동일한 깊이를 가진다. 개별 필터가 입력 특징 맵의 구역을 이동하며 점곱과 활성화 함수를 거쳐 3차원의 윈도우 당 하나의 값을 계산함으로써 11*11 크기의 특징 맵 하나가 형성된다.	
	(7) 두 번째 합성곱 층에서 설정한 필터의 개수는 64개이므로 11*11 크기의 특징 맵이 64개 형성되어 출력되는 특징 맵은 (11, 11, 64)의 차원을 가진다.	(11, 11, 64)

Pooling Layer 1		Shape
	(8) 마찬가지로 특징 맵의 각 두 칸 당 큰 값만을 추출함으로써 각 특징 맵의 크기를 축소하며 개수는 유지된다.	(5, 5, 64)

* 필터 하나의 연산



Flatten & Output Layer		Shape
	(9) 클래스 분류를 위한 마지막 출력 계층과 완전 연결하기 위해 이전 층에서 출력된 (5, 5, 64)의 특징 맵을 1차원의 배열로 변형한다.	(1600,)
	(10) 1600개의 모든 특징과 10개의 클래스를 완전 연결한 연산을 통해 각 클래스로 분류될 Score를 계산하고 활성함수로 Softmax를 취하여 값이 제일 큰 클래스로 분류한다.	(10,)

▶ 1D CNN 예시 그림 (합성곱 및 풀링)

1D CNN 역시 2D와 동일한 연산 과정을 거치며 차이점은 입출력되는 데이터 및 특징맵과 필터의 차원에 있다. 1D CNN에서 필터는 데이터와 동일하게 길이 차원만 존재하며, 입력 시퀀스를 한 방향으로 슬라이딩하며 합성곱 연산을 진행한다. 하나의 데이터가 총 길이가 200인 시퀀스 하나로 이루어진 형태라고 하자. 데이터 하나에 대한 특징이 여러 개라면 여러 개의 시퀀스가 동시에 채널로서 존재할 것이다. 다음에서 이를 2가지 클래스로 분류하는 모형의 구조를 예시로 사용하여 합성곱 층과 풀링 층에서 일어나는 연산을 도식화하였다. 마지막 출력된 특징맵의 일렬 배열과 완전연결 층의 연산은 2D와 동일하다. 합성곱 층에서 학습할 파라미터는 (입력 채널 수)*(시퀀스 길이)*(출력 채널 수) 만큼의 가중치와 출력 채널 수만큼의 편향으로 구성된다.

▷ 하이퍼파라미터 설정

합성곱 층	필터의 개수	필터 사이즈
1	32	5
2	64	5

▷ 구조 요약

Layer	Output Shape	Parameters
Input	(None, 200, 1)	
Conv1D(filters=**32**, kernel_size=**5**, activation='**relu**', input_shape=(200, 1))	(None, 196, 32)	192
MaxPooling1D(pool_size=2)	(None, 98, 32)	0
Conv1D(filters=**64**, kernel_size=**5**, activation='**relu**')	(None, 94, 64)	10304
MaxPooling1D(pool_size=2)	(None, 47, 64)	0
Flatten	(None, 3008)	0
Dense(units=2, activation='**softmax**')	(None, 2)	6018