딥 러닝을 이용한 자연어 처리 입문

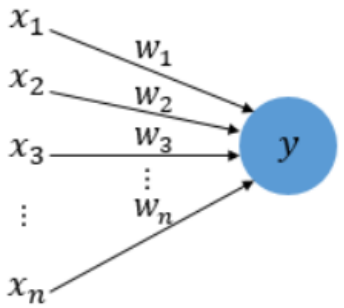
7장 딥 러닝

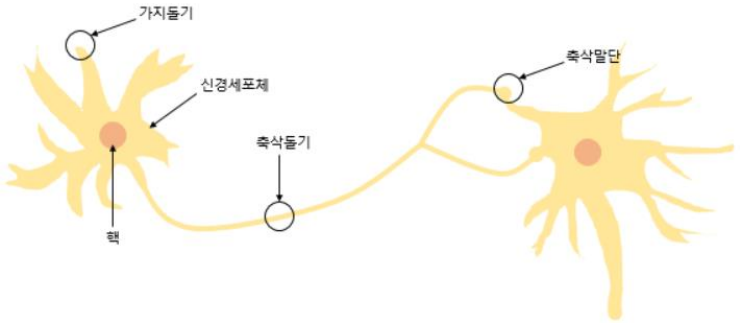
발표자 : 김성윤

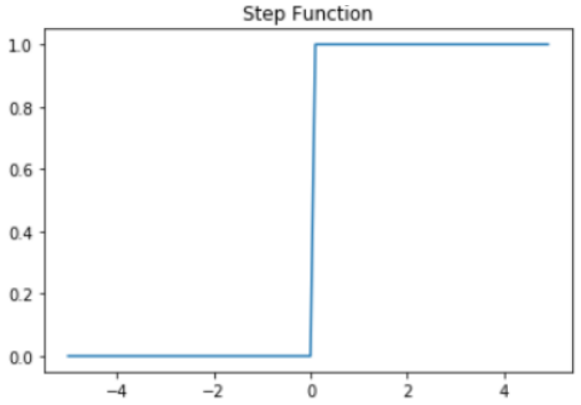
목차

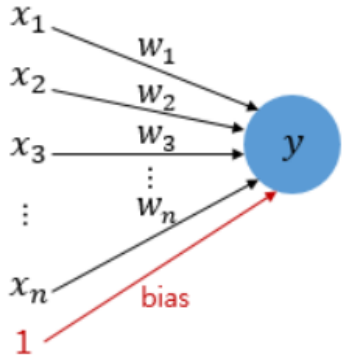
1. 퍼셉트론 - 개념

- 초기 형태의 인공 신경망으로 다수의 입력으로부터 하나의 결과를 내보내는 알고리즘
- 뉴런의 동작 원리와 유사하게 일정치 이상의 크기를 가지면 신호 전달
- 단층 퍼셉트론 & 다층 퍼셉트론









x : 입력값 (가지돌기)
w : 가중치 (축삭돌기)
y : 출력값

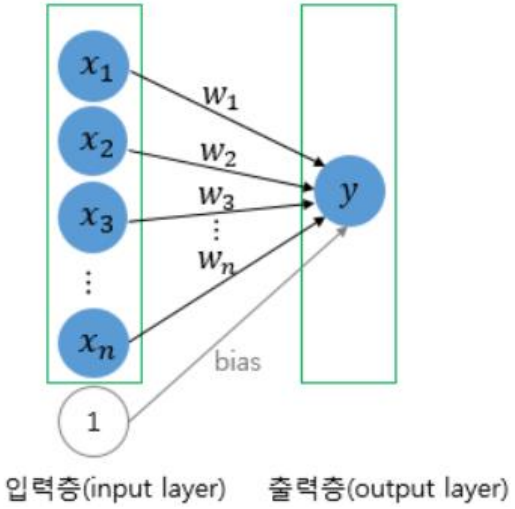
$$if \sum_i^n w_i x_i + b \geq 0 \rightarrow y = 1$$

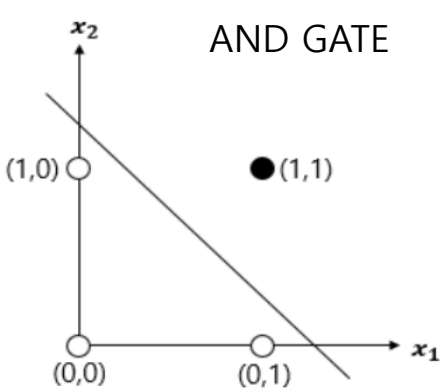
$$if \sum_i^n w_i x_i + b < 0 \rightarrow y = 0$$

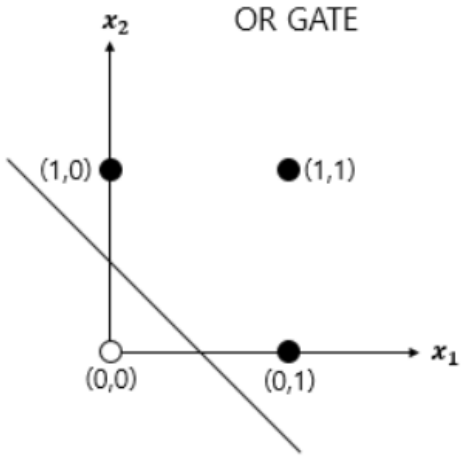
b : 편향

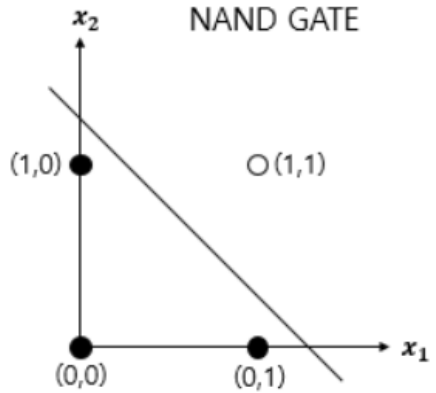
1. 퍼셉트론 – 단층 퍼셉트론

- 2개의 층(입력층과 출력층)으로만 이루어진 상태
- 직선 하나로 두 영역을 나누는 문제에서만 구현 가능
- XOR GATE 구현 불가능



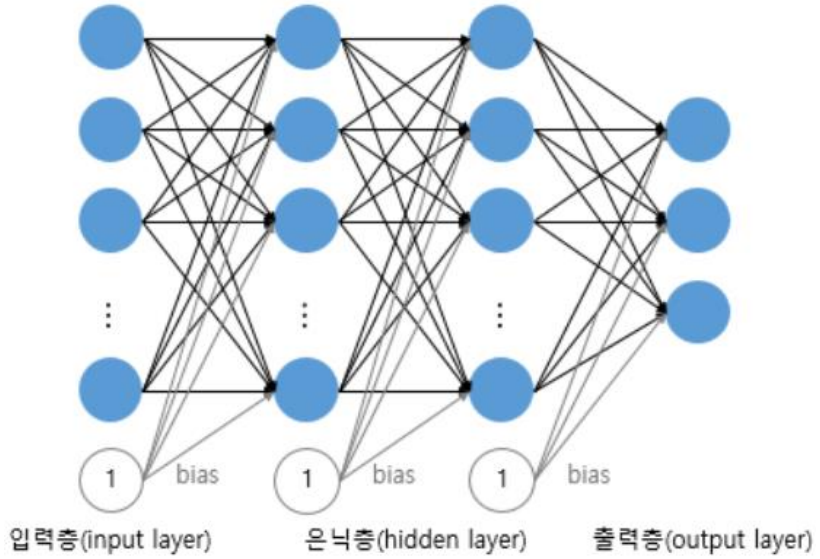




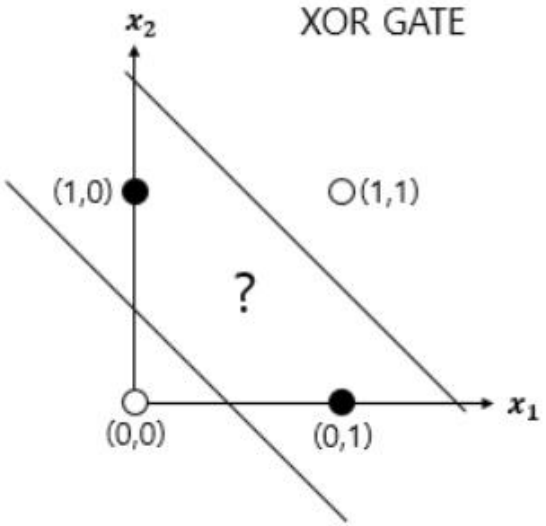


1. 퍼셉트론 – 다층 퍼셉트론(MLP)

- 은닉층이 1개 이상인 퍼셉트론
- 딥 러닝 : 가중치를 스스로 찾도록 심층 신경막을 학습



심층 신경막



단층 퍼셉트론의 XOR 문제

*심층 신경막 : 은닉층이 2개 이상인 신경망

1. 퍼셉트론 – 다층 퍼셉트론(MLP) 실습

- texts_to_matrix()

```
print(tokenizer.texts_to_matrix(texts, mode = 'count'))
```

```
[[0. 0. 1. 1. 1. 0. 0. 0. 0. 0.]
 [0. 1. 1. 1. 0. 0. 0. 0. 0. 0.]
 [0. 2. 0. 0. 0. 1. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 1. 1.]]
```

```
print(tokenizer.texts_to_matrix(texts, mode = 'binary'))
```

```
[[0. 0. 1. 1. 1. 0. 0. 0. 0. 0.]
 [0. 1. 1. 1. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 1. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 1. 1.]]
```

```
print(tokenizer.texts_to_matrix(texts, mode = 'tfidf').round(2))
```

```
[[0.   0.   0.85 0.85 1.1  0.   0.   0.   0.   0.  ]
 [0.   0.85 0.85 0.85 0.   0.   0.   0.   0.   0.  ]
 [0.   1.43 0.   0.   0.   1.1  1.1  0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.   0.   1.1  1.1  1.1 ]]
```

```
print(tokenizer.texts_to_matrix(texts, mode = 'freq').round(2))
```

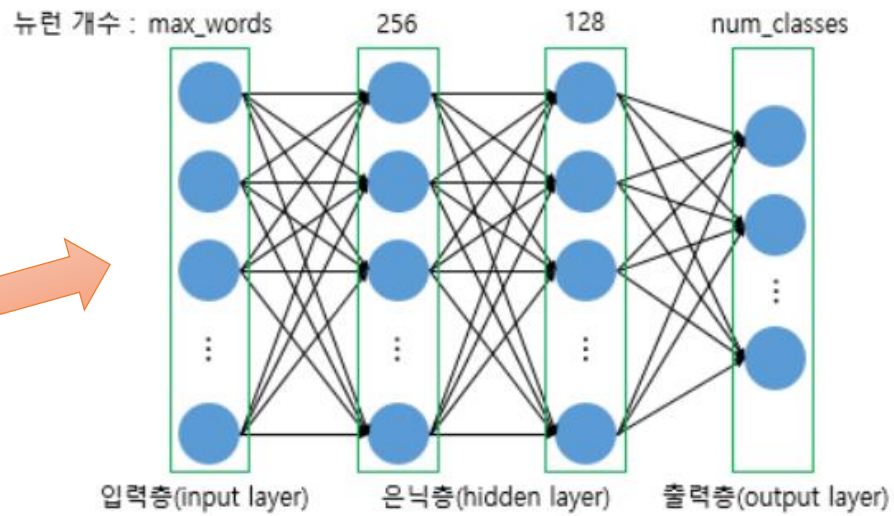
```
[[0.   0.   0.33 0.33 0.33 0.   0.   0.   0.   0.  ]
 [0.   0.33 0.33 0.33 0.   0.   0.   0.   0.   0.  ]
 [0.   0.5  0.   0.   0.   0.25 0.25 0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.   0.   0.33 0.33 0.33]]
```

1. 퍼셉트론 – 다층 퍼셉트론(MLP) 실습

```
def prepare_data(train_data, test_data, mode): # 전처리 함수
    tokenizer = Tokenizer(num_words = vocab_size) # vocab_size 개수만큼의 단어만 사용한다.
    tokenizer.fit_on_texts(train_data)
    X_train = tokenizer.texts_to_matrix(train_data, mode=mode) # 샘플 수 × vocab_size 크기의 행렬 생성
    X_test = tokenizer.texts_to_matrix(test_data, mode=mode) # 샘플 수 × vocab_size 크기의 행렬 생성
    return X_train, X_test, tokenizer.index_word
```

```
def fit_and_evaluate(X_train, y_train, X_test, y_test):
    model = Sequential()
    model.add(Dense(256, input_shape=(vocab_size,), activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))

    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    model.fit(X_train, y_train, batch_size=128, epochs=5, verbose=1, validation_split=0.1)
    score = model.evaluate(X_test, y_test, batch_size=128, verbose=0)
    return score[1]
```



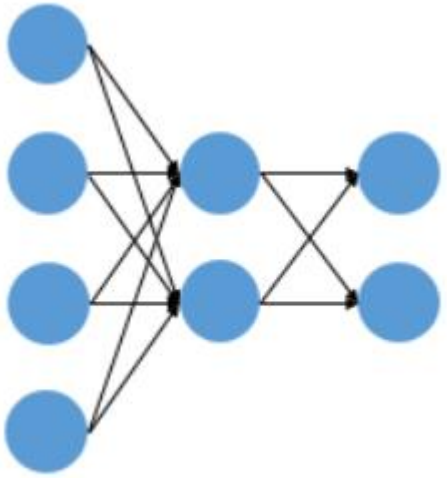

```
modes = ['binary', 'count', 'tfidf', 'freq'] # 4개의 모드를 리스트에 저장.

for mode in modes: # 4개의 모드에 대해서 각각 아래의 작업을 반복한다.
    X_train, X_test, _ = prepare_data(train_email, test_email, mode) # 모드에 따라서 데이터를 전처리
    score = fit_and_evaluate(X_train, y_train, X_test, y_test) # 모델을 훈련하고 평가.
    print(mode+' 모드의 테스트 정확도:', score)
```

```
binary 모드의 테스트 정확도: 0.8312533
count 모드의 테스트 정확도: 0.8239511
tfidf 모드의 테스트 정확도: 0.8381572
freq 모드의 테스트 정확도: 0.6902549
```

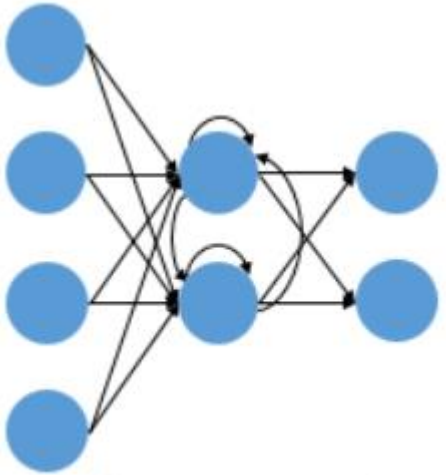
2. 인공 신경망 – 종류 및 용어

- 전결합층(=밀집층) : 어떤 층의 모든 뉴런이 이전 층의 모든 뉴런과 연결돼 있는 층



순방향 신경망(Feed-Forward Neural Network)

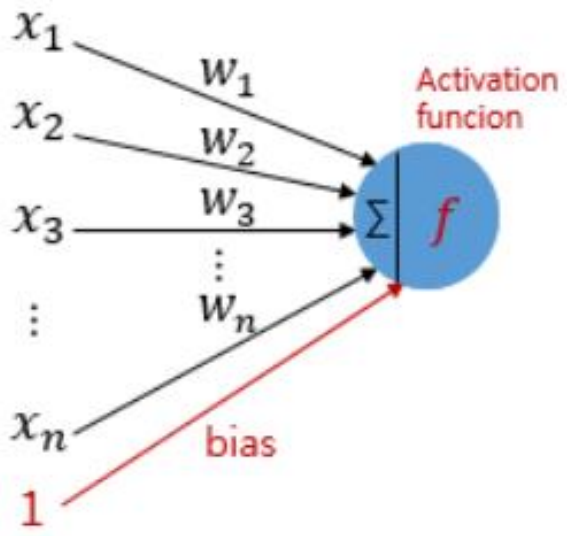
= 피드 포워드 신경망(FFNN)



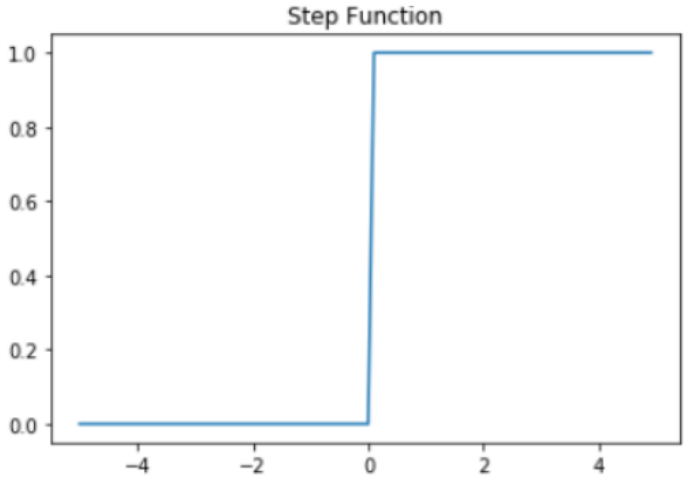
순환 신경망(Recurrent Neural Network)

2. 인공 신경망 – 활성화 함수 개념

- 은닉층과 출력층의 뉴런에서 출력값을 결정하는 함수
- 비선형 함수
- 비선형층, 은닉층 (비선형 함수 사용) vs 선형층, 투사층, 임베딩 층 (선형 함수 사용)

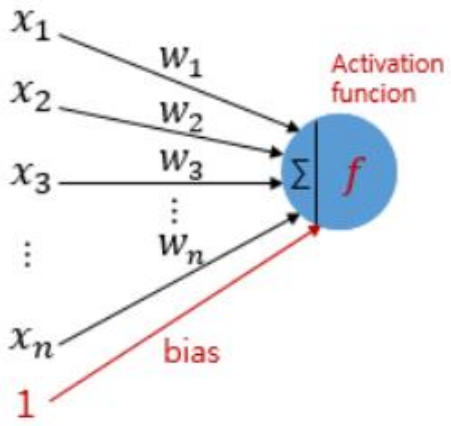


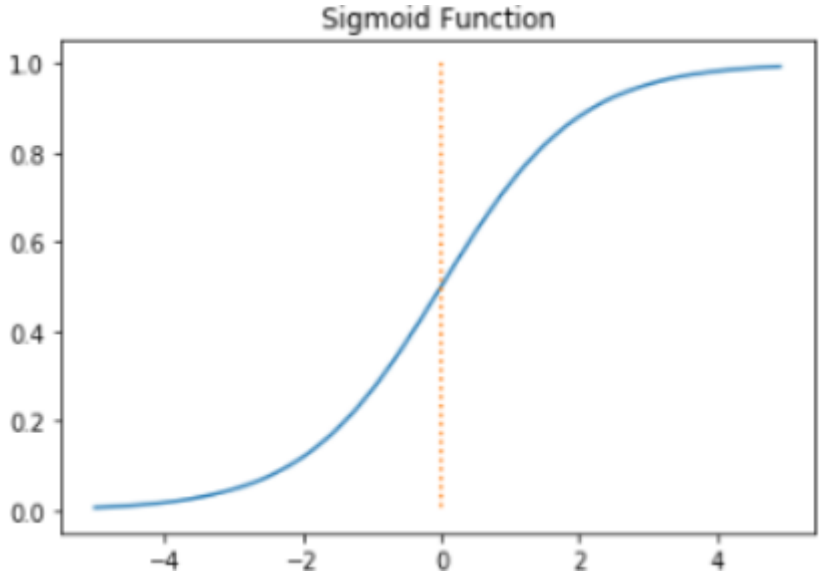
2. 인공 신경망 – 활성화 함수 종류




```
def step(x):
    return np.array(x > 0, dtype=np.int)
```

1) 계단 함수





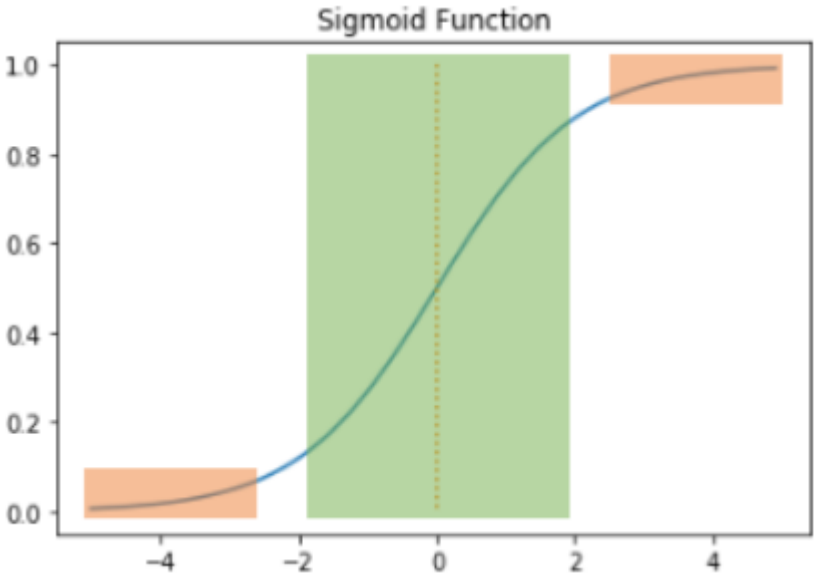

```
def sigmoid(x):
    return 1/(1+np.exp(-x))
```

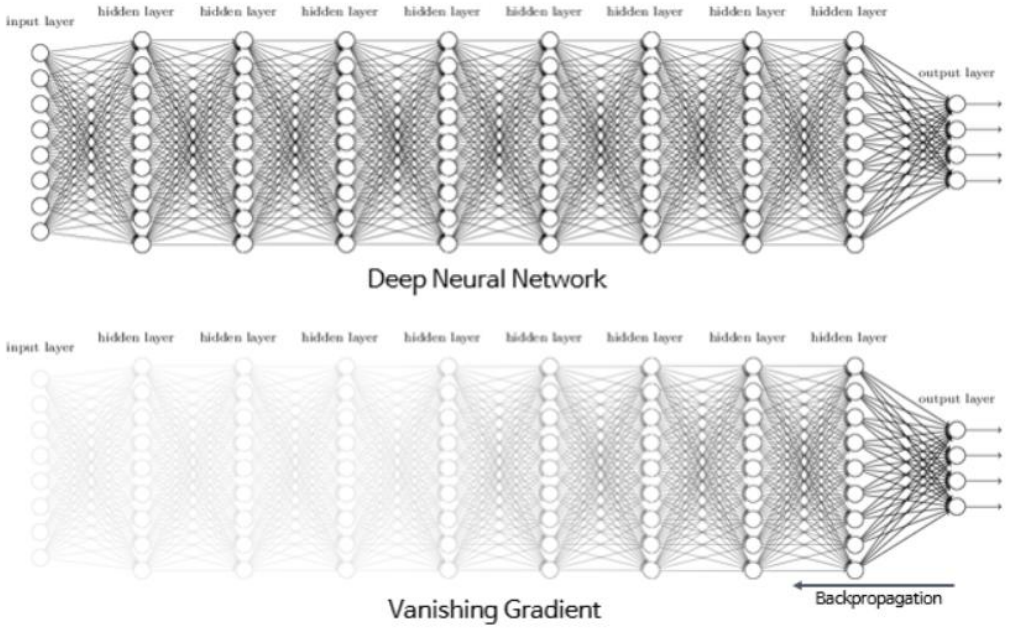
2) 시그모이드 함수

- 출력층에 주로 사용
- 이진 분류 문제에 사용
- 기울기 소실 문제

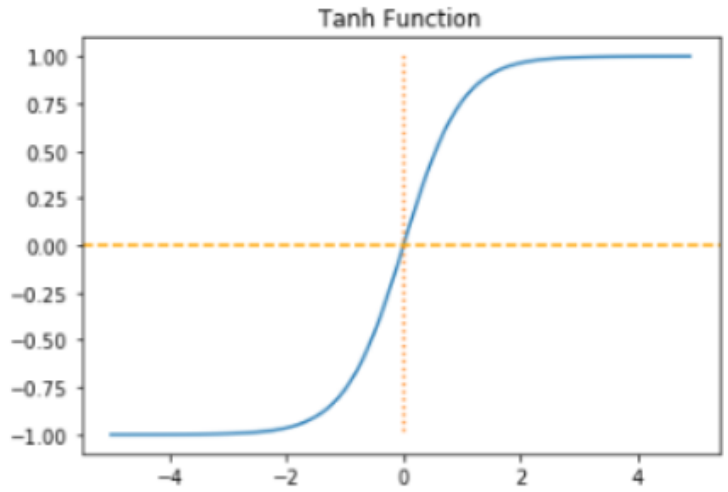
2. 인공 신경망 – 기울기 소실 문제

- 인공 신경망 학습 과정 : 순전파 -> 오차 계산 -> 기울기 계산 -> 역전파
- 주황색 구간의 미분은 0에 가까운 값
- 초록색 구간의 미분은 최대 0.25
- 역전파 과정에서 0에 가까운 값이 누적해서 곱해지게 되면서, 앞단에는 기울기가 잘 전달되지 않게 됨





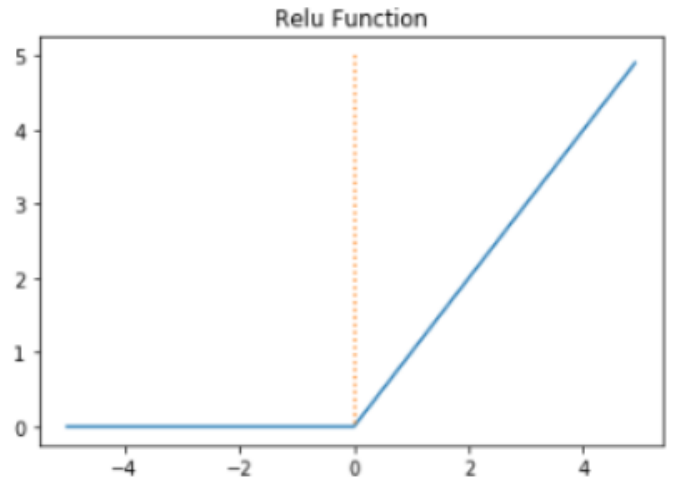
2. 인공 신경망 – 활성화 함수 종류




```
x = np.arange(-5.0, 5.0, 0.1) # -5.0부터 5.0까지 0.1 간격 생성
y = np.tanh(x)
```

3) 하이퍼볼릭탄젠트 함수

- 미분 최대값 1
- 기울기 소실 증상이 적음



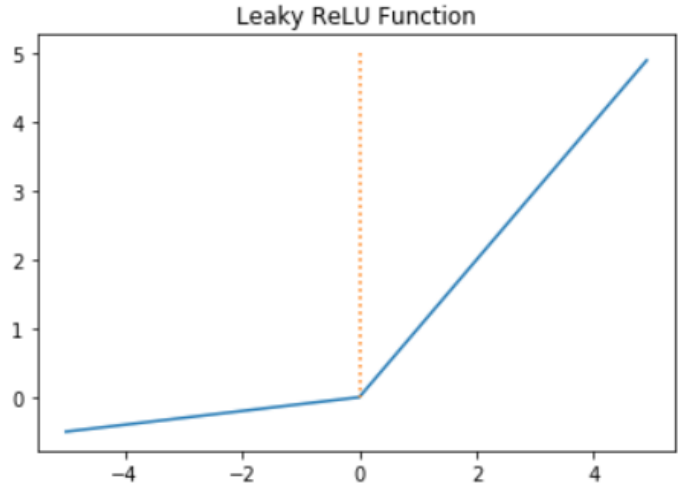

```
def relu(x):
    return np.maximum(0, x)

x = np.arange(-5.0, 5.0, 0.1)
y = relu(x)
```

4) 렐루 함수

- 연산 속도가 빠름
- 음수일 때 미분이 0이므로 '죽은 렐루'가 됨

2. 인공 신경망 – 활성화 함수 종류



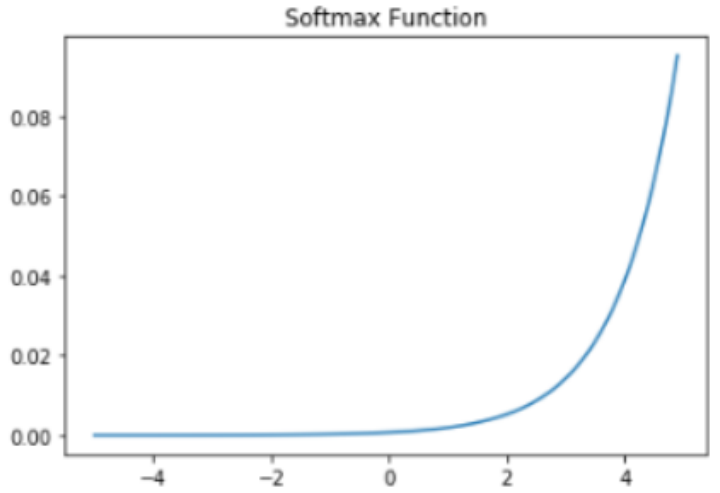

```
a = 0.1

def leaky_relu(x):
    return np.maximum(a*x, x)

x = np.arange(-5.0, 5.0, 0.1)
y = leaky_relu(x)
```

5) 리키 렐루

- 죽은 렐루 보완




```
x = np.arange(-5.0, 5.0, 0.1) # -5.0부터 5.0까지 0.1 간격 생성
y = np.exp(x) / np.sum(np.exp(x))
```

6) 소프트맥스 함수

- 출력층에 주로 사용
- 다중 클래스 분류 문제에 사용

2. 인공 신경망 – 기울기 소실 해결책

- ReLU와 ReLU의 변형을 사용
- 그래디언트 클리핑 : 임계치만큼 기울기 감소

2. 인공 신경망 – 기울기 소실 해결책

- 가중치 초기화

1) 세이비어 초기화 : 여러 층의 기울기 분산 사이에 균형을 맞춰서 특정 층이 너무 주목을 받거나 다른 층이 뒤쳐지는 것을 막음 (S자 활성화 함수와 주로 사용)

2) He 초기화 : 다음 층의 뉴런의 개수를 반영하지 않음. (ReLU 함수와 주로 사용)

<세이비어 초기화>

$$W \sim Uniform(-\sqrt{\frac{6}{n_{in}+n_{out}}}, +\sqrt{\frac{6}{n_{in}+n_{out}}})$$

<He 초기화>

$$W \sim Uniform(-\sqrt{\frac{6}{n_{in}}}, \quad +\sqrt{\frac{6}{n_{in}}})$$

균등 분포를 이용한 가중치 초기화 시, 균등 분포 범위

<He 초기화>

$$\sigma = \sqrt{\frac{2}{n_{in}+n_{out}}}$$

<He 초기화>

$$\sigma = \sqrt{\frac{2}{n_{in}}}$$

정규 분포를 이용한 가중치 초기화 시, 표준편차 값
(평균은 0)

*n(in) : 이전 층의 뉴런의 개수
**n(out) : 다음 층의 뉴런의 개수

2. 인공 신경망 – 기울기 소실 해결책

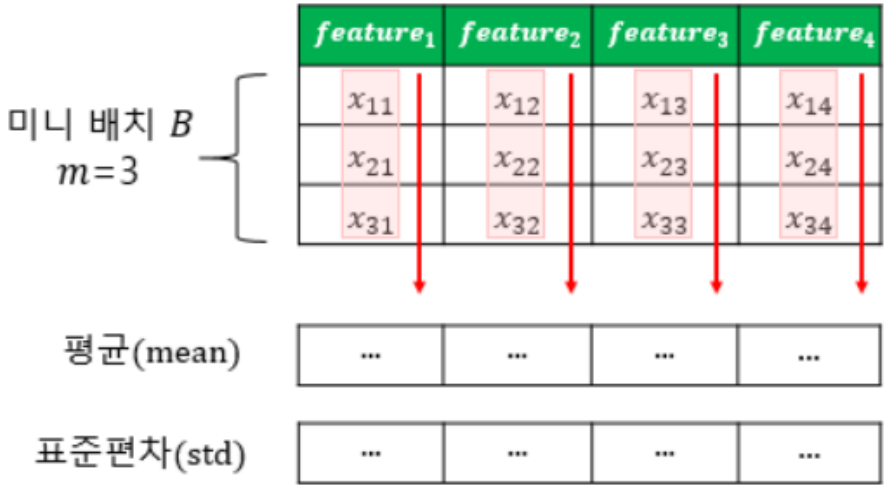
- 내부 공변량 변화 : 학습 과정에서 층 별로 입력 데이터 분포가 달라지는 현상 (∵가중치 변화)
- 배치 정규화 : 각 층에 들어가는 입력을 평균과 분산으로 정규화 (= 한 번에 들어오는 배치 단위로 정규화)

$$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x^{(i)} \quad \# \text{미니 배치에 대한 평균 계산}$$

$$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu_B)^2 \quad \# \text{미니 배치에 대한 분산 계산}$$

$$\hat{x}^{(i)} \leftarrow \frac{x^{(i)} - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}} \quad \# \text{정규화}$$

$$y^{(i)} \leftarrow \gamma\hat{x}^{(i)} + \beta = BN_{\gamma,\beta}(x^{(i)}) \quad \# \text{스케일 조정}(\gamma)\text{과 시프트}(\beta)\text{를 통한 선형 연산}$$



2. 인공 신경망 – 기울기 소실 해결책

- 배치 정규화 특징

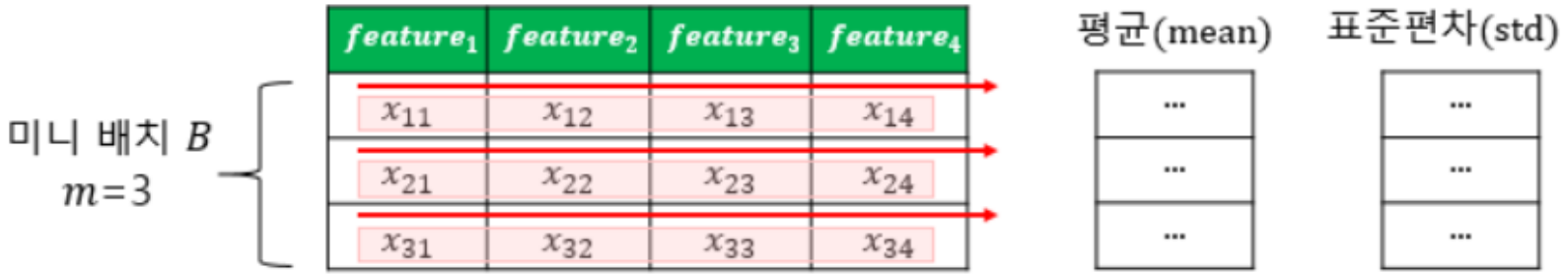
1) 시그모이드 함수나 하이퍼볼릭탄젠트 함수 사용 때의 기울기 소실 문제 크게 개선
2) 가중치 초기화에 훨씬 덜 민감
3) 훨씬 큰 학습률을 사용할 수 있어 학습 속도를 개선
4) 데이터에 잡음 주입의 부수 효과로 과적합을 방지 (∵미니 배치마다 평균과 표준편차를 계산하여 사용)
5) 테스트 데이터에 대한 예측 시에 실행 시간이 느려짐 (∵모델을 복잡성, 추가 계산)

- 배치 정규화 한계

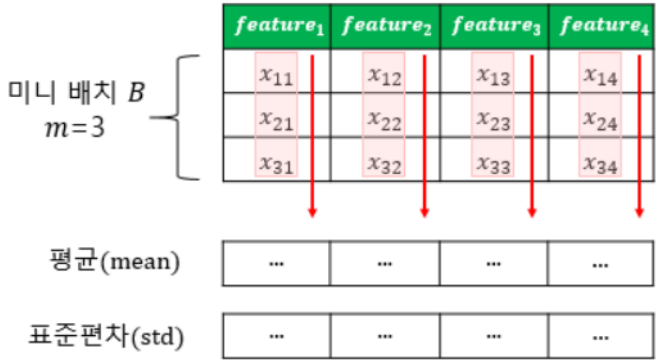
1) 미니 배치 크기에 의존적이다. (너무 작은 배치 크기에서는 잘 작동하지 않는다.)
2) RNN에 적용하기 힘들다. (각 시점마다 다른 통계치가 적용하기 어렵게 만든다.)

2. 인공 신경망 – 기울기 소실 해결책

- 층 정규화

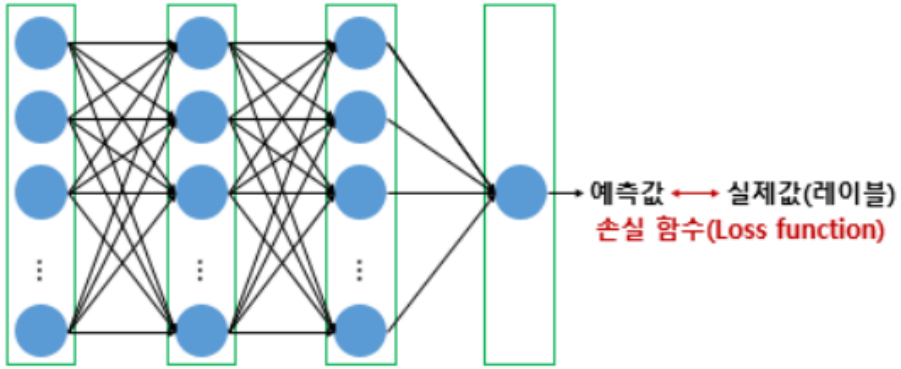


- 배치 정규화



3. 딥 러닝 학습 방법에 필요한 용어

- 손실 함수 : 실제값과 예측값의 차이를 수치화해주는 함수




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

예시 1) MSE

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
```

예시 2) 이진 크로스 엔트로피

예시 3) 카테고리칼 크로스 엔트로피

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

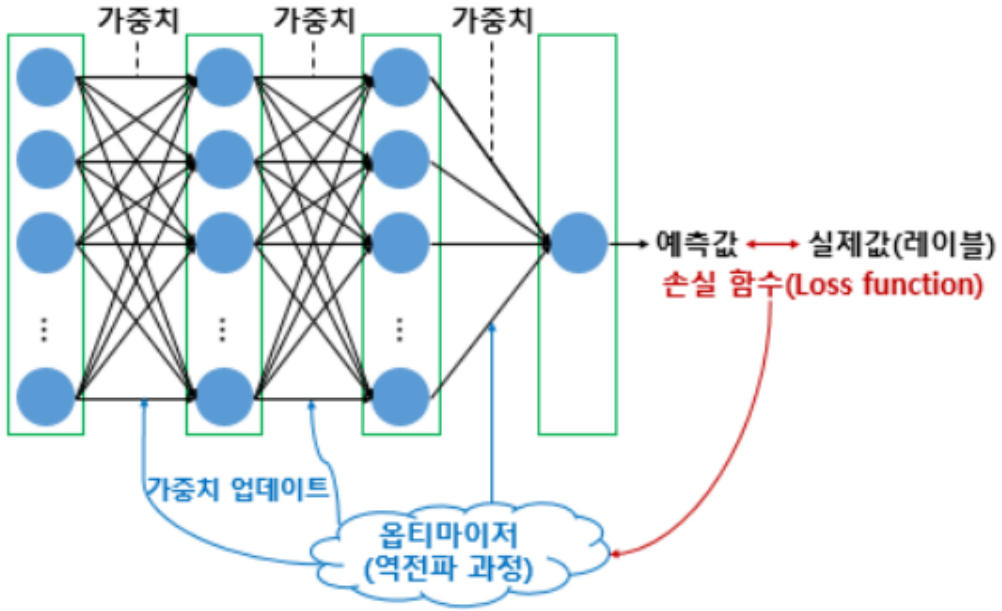
```
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['acc'])
```

(sparse는 원-핫 인코딩을 생략하고 진행하고 싶은 경우 사용)

3. 딥 러닝 학습 방법 – 배치크기&경사 하강법

1) 배치 경사 하강법

- 한 번의 에포크에 모든 매개변수 업데이트를 단 한 번 수행
- 옵티마이저 중 하나로 오차를 구할 때 전체 데이터를 고려
- 한 번의 매개 변수 업데이트에 시간이 오래 걸리며, 메모리를 크게 요구

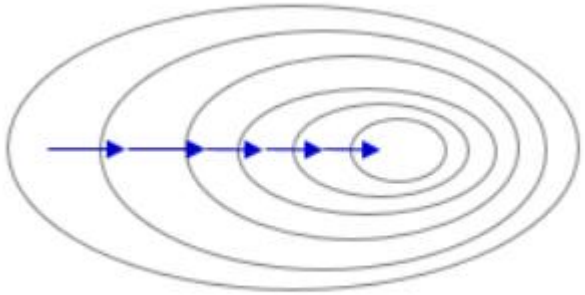


*배치 : 가중치 등의 매개 변수의 값을 조정하기 위해 사용하는 데이터의 양
**1 에포크 : 전체 데이터에 대한 한 번의 훈련 횟수

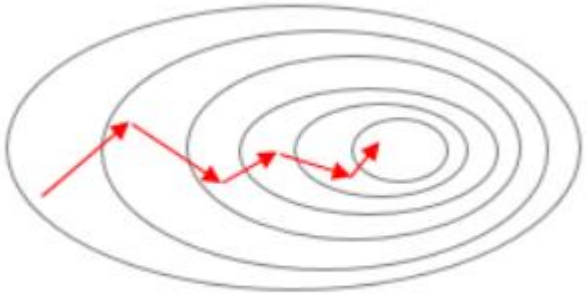
3. 딥 러닝 학습 방법 – 배치크기&경사 하강법

2) 배치 크기가 1인 확률적 경사 하강법(SGD)

- 매개변수 조정 시 전체 데이터가 아니라 랜덤으로 선택한 하나의 데이터에 대해서만 계산하는 방법
- 더 적은 데이터를 사용하므로 더 빠르게 계산
- 하나의 데이터에 대해서만 메모리에 저장하면 되므로 자원이 적은 컴퓨터에서도 쉽게 사용가능
- 매개변수의 변경폭이 불안정하고, 때로는 배치 경사 하강법보다 정확도가 낮음



경사 하강법



SGD

3. 딥 러닝 학습 방법 – 배치크기&경사 하강법

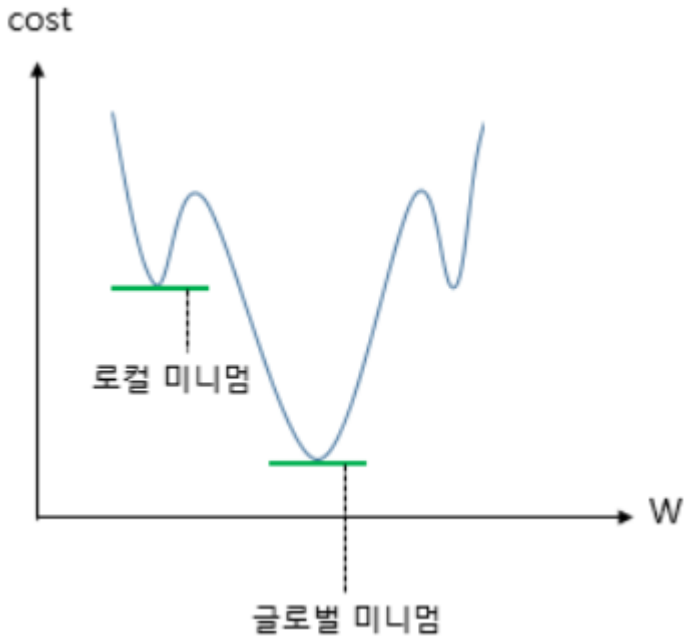
3) 미니 배치 경사 하강법

- 전체 데이터도, 1개의 데이터도 아닐 때, 배치 크기를 지정하여 해당 데이터 개수만큼에 대해서 계산하여 매개 변수의 값을 조정하는 경사 하강법
- 경사 하강법보다 빠르고, SGD보다 안정적

3. 딥 러닝 학습 방법 – 옵티마이저

1) 모멘텀(Momentum)

- 관성을 응용한 방법
- 작은 웅덩이(로컬 미니엄)에 빠지더라도 관성으로 넘어서는 효과



3. 딥 러닝 학습 방법 – 옵티마이저

2) 아다그라드(Adagrad)

- 각 매개변수에 서로 다른 학습률을 적용
- 변화가 많은 매개변수는 학습률이 낮게, 변화가 적은 매개변수는 학습률이 높게

3. 딥 러닝 학습 방법 – 옵티마이저

3) 알엠에스프롭(RMSprop)

- 아다그라드의 학습을 계속 진행할 경우, 학습률이 지나치게 떨어진다는 단점을 개선

3. 딥 러닝 학습 방법 – 옵티마이저

4) 아담(Adam)

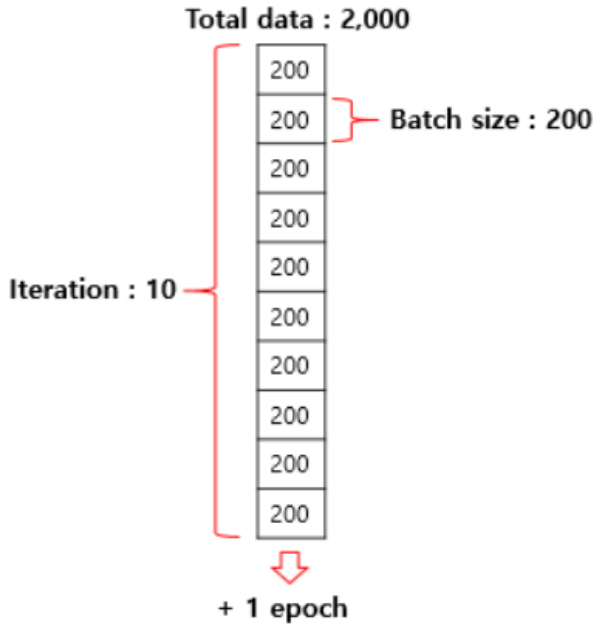
- 알엠에스프롭 + 모멘텀
- 방향과 학습률을 모두 잡음

3. 딥 러닝 학습 방법 – 에포크와 배치 크기와 이터레이션

- 에포크 : 인공 신경망에서 전체 데이터에 대해서 순전파와 역전파가 끝난 상태
- 예) 문제지의 모든 문제를 끝까지 다 풀고, 채점을 하여 한 번 끝낸 상태

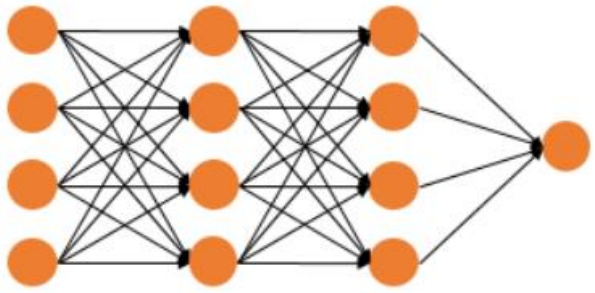
- 배치 크기 : 몇 개의 데이터 단위로 매개변수 업데이트 하는지
- 예) 문제지에서 몇 개씩 문제를 풀고 나서 정답지를 확인하느냐

- 이터레이션 : 한 번의 에포크를 끝내기 위해서 필요한 배치의 수
- 예) 전체 데이터가 2000이고 배치 크기가 200이라면 이터레이션은 10

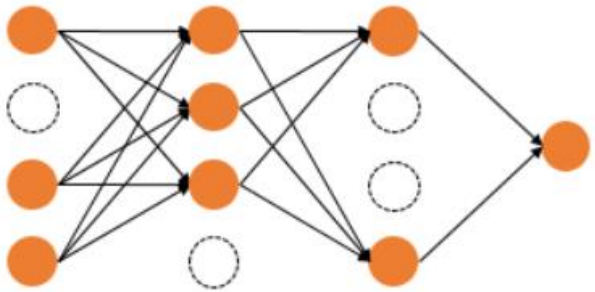


4. 과적합을 막는 방법

1) 데이터의 양을 늘리기 - 일반적인 패턴의 데이터를 학습하여 과적합 방지
2) 모델의 복잡도 줄이기 - 은닉층과 매개변수 수를 조정하여 과적합 방지
3) 가중치 규제 적용하기 – L1 규제 : 가중치 ω들의 절대값의 합계($\lambda|\omega|$)를 비용함수에 추가 L1은 어떤 특성이 모델에 영향을 주는지 정확한 판단을 할 때 유용
4) 가중치 규제 적용하기 – L2 규제 : 모든 가중치 ω들의 제곱합($\frac{1}{2}\lambda\omega^2$)을 비용함수에 추가 *L2는 ω값이 완전히 0이 되기 보다는 0에 가까워지므로 0이 될 수 있는 L1보다 잘 작동
5) 드롭아웃 - 학습 과정에서 일부 신경망을 사용하지 않는 것



드롭아웃 적용 전

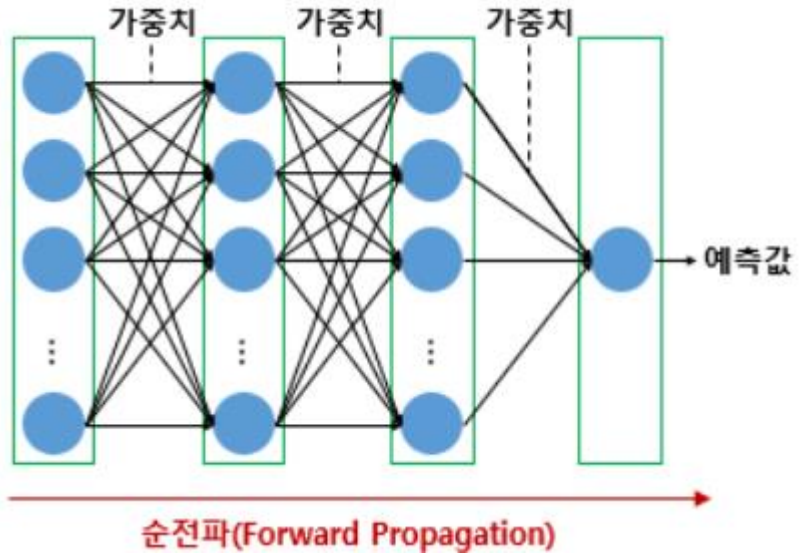


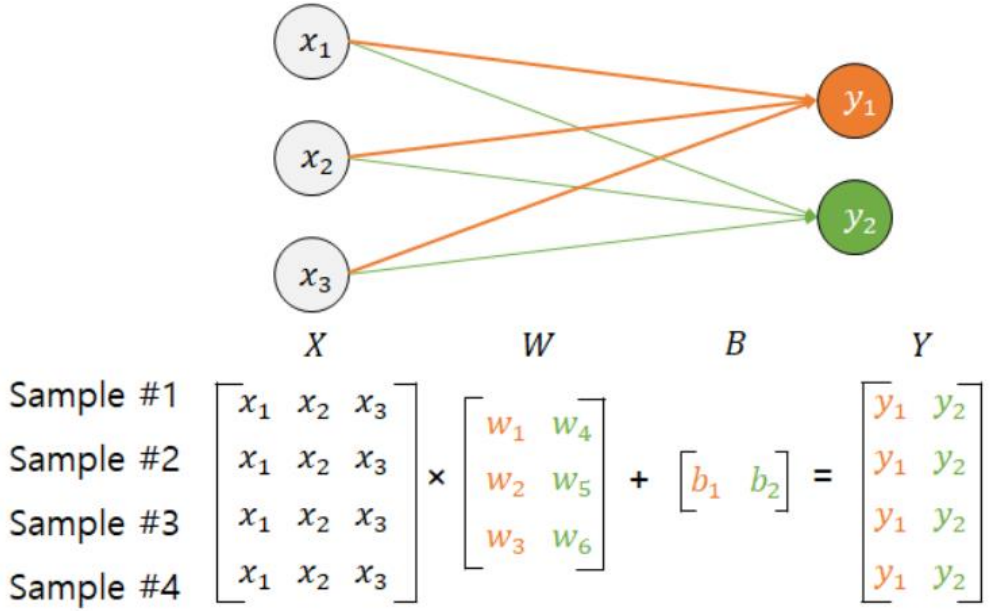
드롭아웃 적용 후

*비용함수가 최소화 되기 위해서는 가중치도 최소화 되어야 하므로

5. 순전파

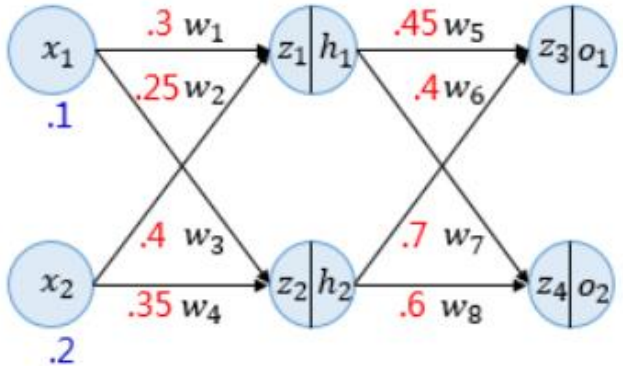
- 입력층에서 출력층 방향으로 예측값의 연산이 진행되는 과정





6. 역전파 – 1단계 '순전파'

- 학습 과정에서 기울기 계산 후 가중치를 업데이트하는 과정



실제값 : 0.4

실제값 : 0.6

$$z_1 = w_1x_1 + w_2x_2 = 0.3\times0.1 + 0.25\times0.2 = 0.08$$

$$z_2 = w_3x_1 + w_4x_2 = 0.4\times0.1 + 0.35\times0.2 = 0.11$$

$$h_1 = sigmoid(z_1) = 0.51998934$$

$$h_2 = sigmoid(z_2) = 0.52747230$$

$$z_3 = w_5h_1 + w_6h_2 = 0.45\times h_1 + 0.4\times h_2 = 0.44498412$$

$$z_4 = w_7h_1 + w_8h_2 = 0.7\times h_1 + 0.6\times h_2 = 0.68047592$$

$$o_1 = sigmoid(z_3) = 0.60944600$$

$$o_2 = sigmoid(z_4) = 0.66384491$$

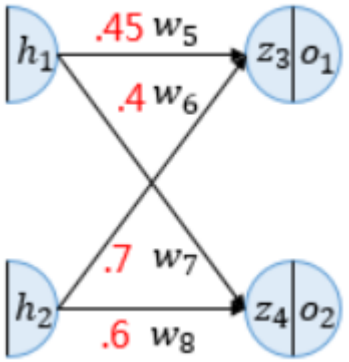
$$E_{o1} = \frac{1}{2}(target_{o1} - output_{o1})^2 = 0.02193381$$

$$E_{o2} = \frac{1}{2}(target_{o2} - output_{o2})^2 = 0.00203809$$

$$E_{total} = E_{o1} + E_{o2} = 0.02397190$$

6. 역전파 – 2단계 '역전파 1단계'

- 출력층과 N층 사이의 가중치를 업데이트하는 단계



$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial o_1} \times \frac{\partial o_1}{\partial z_3} \times \frac{\partial z_3}{\partial w_5}$$

w5 업데이트를 위한 계산 식

$$E_{total} = \frac{1}{2}(target_{o1} - output_{o1})^2 + \frac{1}{2}(target_{o2} - output_{o2})^2$$

$$\frac{\partial E_{total}}{\partial o_1} = -(target_{o1} - output_{o1}) = -(0.4 - 0.60944600) = 0.20944600$$

$$\frac{\partial o_1}{\partial z_3} = o_1 \times (1 - o_1) = 0.60944600(1 - 0.60944600) = 0.23802157$$

$$\frac{\partial z_3}{\partial w_5} = h_1 = 0.51998934$$

$$w_5^+ = w_5 - \alpha \frac{\partial E_{total}}{\partial w_5} = 0.45 - 0.5 \times 0.02592286 = 0.43703857$$

α : 학습률(하이퍼파라미터)

6. 역전파 – 3단계 '역전파 2단계'

- N층과 N층 이전층 사이의 가중치를 업데이트하는 단계

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial h_1} \times \frac{\partial h_1}{\partial z_1} \times \frac{\partial z_1}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial h_1} = \frac{\partial E_{o1}}{\partial h_1} + \frac{\partial E_{o2}}{\partial h_1}$$

$$\frac{\partial E_{o1}}{\partial h_1} = \frac{\partial E_{o1}}{\partial z_3} \times \frac{\partial z_3}{\partial h_1} = \frac{\partial E_{o1}}{\partial o_1} \times \frac{\partial o_1}{\partial z_3} \times \frac{\partial z_3}{\partial h_1}$$

$$= -(target_{o1} - output_{o1}) \times o_1 \times (1 - o_1) \times w_5$$

$$= 0.20944600 \times 0.23802157 \times 0.45 = 0.02243370$$

$$\frac{\partial E_{o2}}{\partial h_1} = \frac{\partial E_{o2}}{\partial z_4} \times \frac{\partial z_4}{\partial h_1} = \frac{\partial E_{o2}}{\partial o_2} \times \frac{\partial o_2}{\partial z_4} \times \frac{\partial z_4}{\partial h_1} = 0.00997311$$

$$\frac{\partial E_{total}}{\partial h_1} = 0.02243370 + 0.00997311 = 0.03240681$$

$$\frac{\partial h_1}{\partial z_1} = h_1 \times (1 - h_1) = 0.51998934(1 - 0.51998934) = 0.24960043$$

$$\frac{\partial z_1}{\partial w_1} = x_1 = 0.1$$

$$w_1^+ = w_1 - \alpha \frac{\partial E_{total}}{\partial w_1} = 0.3 - 0.5 \times 0.00080888 = 0.29959556$$

6. 역전파 – 4단계 '결과 확인'

- 역전파를 통해 오차가 감소했는지 확인

$$z_1 = w_1 x_1 + w_2 x_2 = 0.29959556 \times 0.1 + 0.24919112 \times 0.2 = 0.07979778$$

$$z_2 = w_3 x_1 + w_4 x_2 = 0.39964496 \times 0.1 + 0.34928991 \times 0.2 = 0.10982248$$

$$h_1 = sigmoid(z_1) = 0.51993887$$

$$h_2 = sigmoid(z_2) = 0.52742806$$

$$z_3 = w_5 h_1 + w_6 h_2 = 0.43703857 \times h_1 + 0.38685205 \times h_2 = 0.43126996$$

$$z_4 = w_7 h_1 + w_8 h_2 = 0.69629578 \times h_1 + 0.59624247 \times h_2 = 0.67650625$$

$$o_1 = sigmoid(z_3) = 0.60617688$$

$$o_2 = sigmoid(z_4) = 0.66295848$$

$$E_{o1} = \frac{1}{2}(target_{o1} - output_{o1})^2 = 0.02125445$$

$$E_{o2} = \frac{1}{2}(target_{o2} - output_{o2})^2 = 0.00198189$$

$$E_{total} = E_{o1} + E_{o2} = 0.02323634 \quad \Leftarrow \quad E_{total} = E_{o1} + E_{o2} = 0.02397190$$

7. 케라스 훑어보기

-----	원-핫 벡터	임베딩 벡터
차원	고차원(단어 집합의 크기)	저차원
다른 표현	대부분의 값이 0이 대부분인 희소 벡터	모든 값이 실수인 밀집 벡터
표현 방법	수동	훈련 데이터로부터 학습함
값의 타입	1과 0	실수

- 딥 러닝 과정

1) 전처리 : Tokenizer(), pad_sequence()

2) 워드 임베딩 : 단어들을 밀집 벡터로 만드는 과정, Embedding()

```
# 1. 토큰화
tokenized_text = [['Hope', 'to', 'see', 'you', 'soon'], ['Nice', 'to', 'see', 'you', 'again']]

# 2. 각 단어에 대한 정수 인코딩
encoded_text = [[0, 1, 2, 3, 4],[5, 1, 2, 3, 6]]

# 3. 위 정수 인코딩 데이터가 아래의 임베딩 층의 입력이 된다.
vocab_size = 7
embedding_dim = 2
Embedding(vocab_size, embedding_dim, input_length=5)
```

```
# 각 정수는 아래의 테이블의 인덱스로 사용되며 Embedding()은 각 단어마다 임베딩 벡터를 리턴한다.
+------------+------------+
|   index    | embedding  |
+------------+------------+
|     0      | [1.2, 3.1] |
|     1      | [0.1, 4.2] |
|     2      | [1.0, 3.1] |
|     3      | [0.3, 2.1] |
|     4      | [2.2, 1.4] |
|     5      | [0.7, 1.7] |
|     6      | [4.1, 2.0] |
+------------+------------+
```

7. 케라스 훑어보기 – Sequential API

3) 모델링 : Sequential() – 층 구성하기

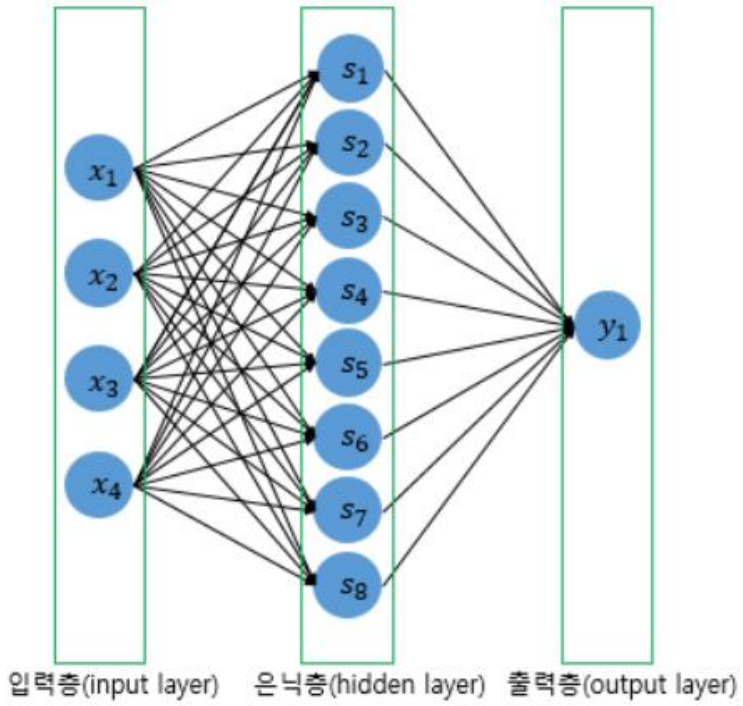
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(...) # 층 추가
model.add(...) # 층 추가
model.add(...) # 층 추가
```

```
model = Sequential()
model.add(Embedding(vocab_size, output_dim, input_length))
```

```
model = Sequential()
model.add(Dense(1, input_dim=3, activation='relu'))
```

```
model = Sequential()
model.add(Dense(8, input_dim=4, activation='relu'))
model.add(Dense(1, activation='sigmoid')) # 출력층
```



7. 케라스 훑어보기 – Sequential API

4) 컴파일과 훈련 : compile() – 손실함수와 옵티마이저, 메트릭 함수 결정

```
from tensorflow.keras.layers import SimpleRNN, Embedding, Dense
from tensorflow.keras.models import Sequential

vocab_size = 10000
embedding_dim = 32
hidden_units = 32

model = Sequential()
model.add(Embedding(vocab_size, embedding_dim))
model.add(SimpleRNN(hidden_units))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
```

7. 케라스 훑어보기 – Sequential API

참고) 문제 유형에 따른 **손실 함수**와 **활성화 함수**

문제 유형	**손실 함수명**	**출력층의 활성화 함수명**	**참고 실습**
회귀 문제	mean_squared_error	-	선형 회귀 실습
다중 클래스 분류	categorical_crossentropy	소프트맥스	로이터 뉴스 분류하기
다중 클래스 분류	sparse_categorical_crossentropy	소프트맥스	양방향 LSTM를 이용한 품사 태깅
이진 분류	binary_crossentropy	시그모이드	IMDB 리뷰 감성 분류하기

7. 케라스 훑어보기 – Sequential API

4) 컴파일과 훈련 : fit() – 모델 학습

```
model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=0, validation_data(X_val, y_val))
```

```
# 훈련 데이터의 20%를 검증 데이터로 사용.
model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=0, validation_split=0.2))
```

```
# verbose = 1일 경우.
Epoch 88/100
7/7 [==============================] - 0s 143us/step - loss: 0.1029 - acc: 1.0000
```

```
# verbose = 2일 경우.
Epoch 88/100
 - 0s - loss: 0.1475 - acc: 1.0000
```

7. 케라스 훑어보기 – Sequential API

5) 평가와 예측 : evaluate() – 테스트 데이터로 정확도 평가, predict() – 임의의 입력으로 출력 확인

6) 모델의 저장과 로드 : save() – 모델을 hdf5 파일로 저장, load_model() – 저장한 모델 불러오기

```
# 위의 fit() 코드의 연장선상인 코드
model.evaluate(X_test, y_test, batch_size=32)
```

```
# 위의 fit() 코드의 연장선상인 코드
model.predict(X_input, batch_size=32)
```

```
model.save("model_name.h5")
```

```
from tensorflow.keras.models import load_model
model = load_model("model_name.h5")
```

7. 케라스 훑어보기 – 함수형 API

- 각 층을 일종의 함수로서 정의

1) 전결합 피드 포워드 신경망

- Sequential API와는 다르게 입력 데이터의 크기(shape)를 인자로 입력층을 정의

```
inputs = Input(shape=(10,))
hidden1 = Dense(64, activation='relu')(inputs)
hidden2 = Dense(64, activation='relu')(hidden1)
output = Dense(1, activation='sigmoid')(hidden2)
model = Model(inputs=inputs, outputs=output) # <- 새로 추가
```

```
inputs = Input(shape=(10,))
x = Dense(8, activation="relu")(inputs)
x = Dense(4, activation="relu")(x)
x = Dense(1, activation="linear")(x)
model = Model(inputs, x)
```

7. 케라스 훑어보기 – 함수형 API

2) 선형 회귀

```
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras import optimizers
from tensorflow.keras.models import Model

X = [1, 2, 3, 4, 5, 6, 7, 8, 9] # 공부하는 시간
y = [11, 22, 33, 44, 53, 66, 77, 87, 95] # 각 공부하는 시간에 맵핑되는 성적

inputs = Input(shape=(1,))
output = Dense(1, activation='linear')(inputs)
linear_model = Model(inputs, output)

sgd = optimizers.SGD(lr=0.01)

linear_model.compile(optimizer=sgd, loss='mse', metrics=['mse'])
linear_model.fit(X, y, epochs=300)
```

7. 케라스 훑어보기 – 함수형 API

3) 로지스틱 회귀

```
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model

inputs = Input(shape=(3,))
output = Dense(1, activation='sigmoid')(inputs)
logistic_model = Model(inputs, output)
```

7. 케라스 훑어보기 – 함수형 API

4) 다중 입력을 받는 모델

```
from tensorflow.keras.layers import Input, Dense, concatenate
from tensorflow.keras.models import Model

# 두 개의 입력층을 정의
inputA = Input(shape=(64,))
inputB = Input(shape=(128,))

# 첫번째 입력층으로부터 분기되어 진행되는 인공 신경망을 정의
x = Dense(16, activation="relu")(inputA)
x = Dense(8, activation="relu")(x)
x = Model(inputs=inputA, outputs=x)

# 두번째 입력층으로부터 분기되어 진행되는 인공 신경망을 정의
y = Dense(64, activation="relu")(inputB)
y = Dense(32, activation="relu")(y)
y = Dense(8, activation="relu")(y)
y = Model(inputs=inputB, outputs=y)
```

```
# 두개의 인공 신경망의 출력을 연결(concatenate)
result = concatenate([x.output, y.output])

z = Dense(2, activation="relu")(result)
z = Dense(1, activation="linear")(z)

model = Model(inputs=[x.input, y.input], outputs=z)
```

7. 케라스 훑어보기 – 함수형 API

5) RNN 은닉층 사용하기

```
from tensorflow.keras.layers import Input, Dense, LSTM
from tensorflow.keras.models import Model

inputs = Input(shape=(50,1))
lstm_layer = LSTM(10)(inputs)
x = Dense(10, activation='relu')(lstm_layer)
output = Dense(1, activation='sigmoid')(x)

model = Model(inputs=inputs, outputs=output)
```

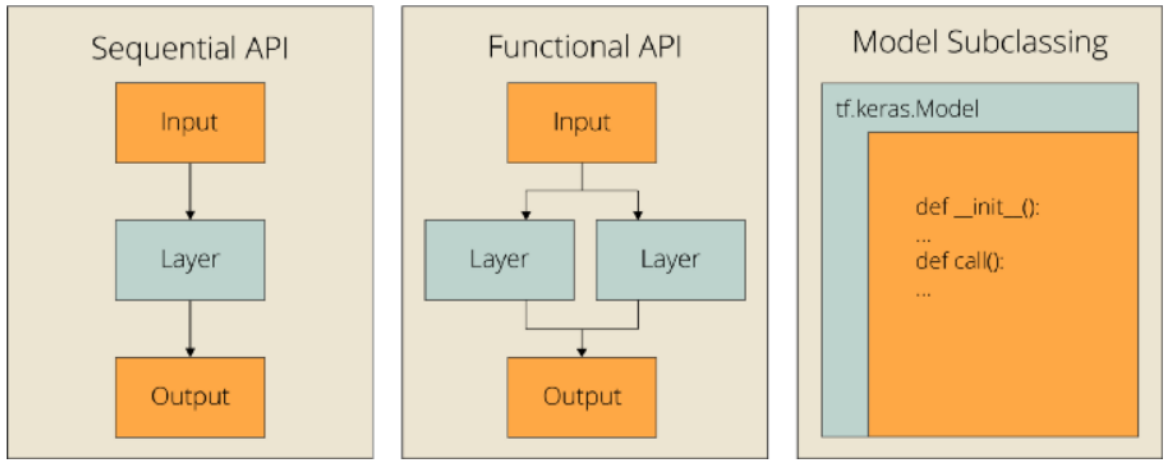
7. 케라스 훑어보기 – 함수형 API

6) 다르지만 같은 의미

```
result = Dense(128)(input)
```

```
dense = Dense(128)
result = dense(input)
```

7. 케라스 훑어보기 – 서브클래싱 API




```python
import tensorflow as tf
```

```python
class LinearRegression(tf.keras.Model):
  def __init__(self):
    super(LinearRegression, self).__init__()
    self.linear_layer = tf.keras.layers.Dense(1, input_dim=1, activation='linear')

  def call(self, x):
    y_pred = self.linear_layer(x)

    return y_pred
```

```python
model = LinearRegression()
```

```python
X = [1, 2, 3, 4, 5, 6, 7, 8, 9] # 공부하는 시간
y = [11, 22, 33, 44, 53, 66, 77, 87, 95] # 각 공부하는 시간에 맵핑되는 성적
```

```python
sgd = tf.keras.optimizers.SGD(lr=0.01)
model.compile(optimizer=sgd, loss='mse', metrics=['mse'])
model.fit(X, y, epochs=300)
```

7. 케라스 훑어보기 – Subclassing vs Sequential vs Functional

- Sequential

1) 장점 : 단순하게 층을 쌓는 방식으로 쉽고 사용하기가 간단함 (= 간단한 모델에 적합)

2) 단점 : 다수의 입력, 다수의 출력을 가진 모델 또는 층 간의 연결이나 덧셈과 같은 연산을 하는 모델을 구현하기는 적합하지 않음 (= 복잡한 모델에는 적합하지 않음)

- Functional

1) 장점 : Sequential API로는 구현하기 어려운 복잡한 모델들을 구현

2) 단점 : 입력의 크기(shape)를 명시한 입력층(Input layer)을 모델의 앞단에 정의

- Subclassing

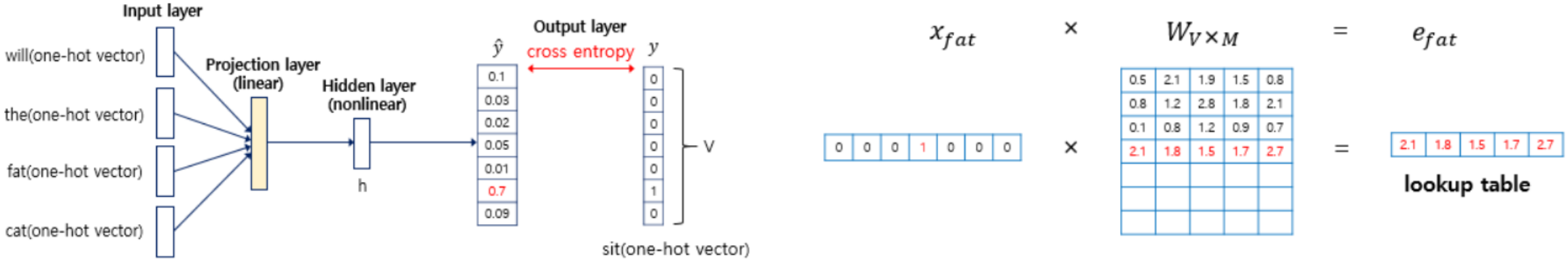
1) 장점 : Functional API로도 구현할 수 없는 모델들조차 구현이 가능 (= 밑바닥부터 새로운 수준의 아키텍처를 구현 가능)

2) 단점 : 객체 지향 프로그래밍에 익숙해야 하므로 코드 사용이 가장 까다로움

7. 케라스 훑어보기 – NNLM 실습

- N-gram 모델의 한계 : 충분한 데이터를 관측하지 못하면 언어를 정확히 모델링하지 못하는 희소 문제 (= 단어의 의미적 유사성 파악하지 못함)
- NNLM : 앞의 n개만큼(window)만 참고

예문 : "what will the fat cat sit on"



투사층 : $p^{layer} = (lookup(x_{t-n}); \ldots; lookup(x_{t-2}); lookup(x_{t-1})) = (e_{t-n}; \ldots; e_{t-2}; e_{t-1})$

은닉층 : $h^{layer} = tanh(W_h p^{layer} + b_h)$

출력층 : $\hat{y} = softmax(W_y h^{layer} + b_y)$

7. 케라스 훑어보기 – NNLM 실습

- NNLM의 이점 : 단어 유사도 계산 (= 희소 문제 해결)
- NNLM의 문제점 : 고정된 길이의 입력 (= 다음 단어를 예측하기 위해서 이전의 모든 단어를 참고하지 못함)