인공 신경망 소개

Hands-On Machine Learning Part2
& Deep Learning from Scratch 3

Heeji Won

Contents

01. 퍼셉트론

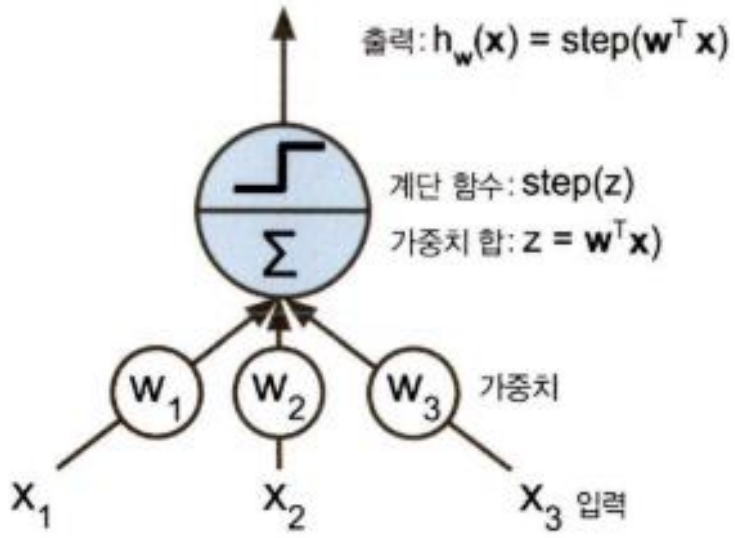
02. 다층 퍼셉트론

03. 구현하기

01. 퍼셉트론

- 기본 구조

– TLU(threshold logic unit 또는 LTU)이라는 인공 뉴런을 기반으로 함



"TLU를 훈련한다는 것은

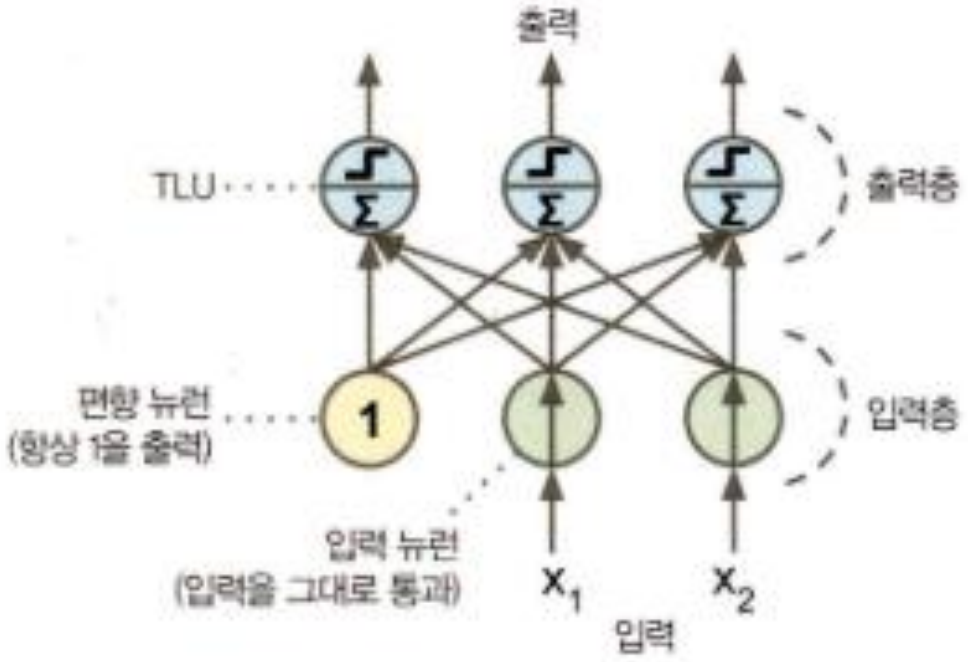
최적의 가중치를 찾는다는 것"

식 10-1 퍼셉트론에서 일반적으로 사용하는 계단 함수(임곗값을 0으로 가정)

$$\text{heaviside}(z) = \begin{cases} 0 & z < 0\text{일 때} \\ 1 & z \geq 0\text{일 때} \end{cases} \qquad \text{sgn}(z) = \begin{cases} -1 & z < 0\text{일 때} \\ 0 & z = 0\text{일 때} \\ +1 & z > 0\text{일 때} \end{cases}$$

01. 퍼셉트론

- 예시 – 다중 출력 분류기



* 층의 모든 뉴런들이 연결되어 있으므로 **완전 연결 층**(fully connected layer) 또는 **밀집 층**(dense layer)

- 출력 계산

$$h_{\mathbf{W},\mathbf{b}}(\mathbf{X}) = \phi(\mathbf{XW} + \mathbf{b})$$

- W : 가중치 행렬. 열은 각각 입력 뉴런과 출력 뉴런에 해당
- b : 편향 벡터
- ϕ : 활성화 함수(activation function)

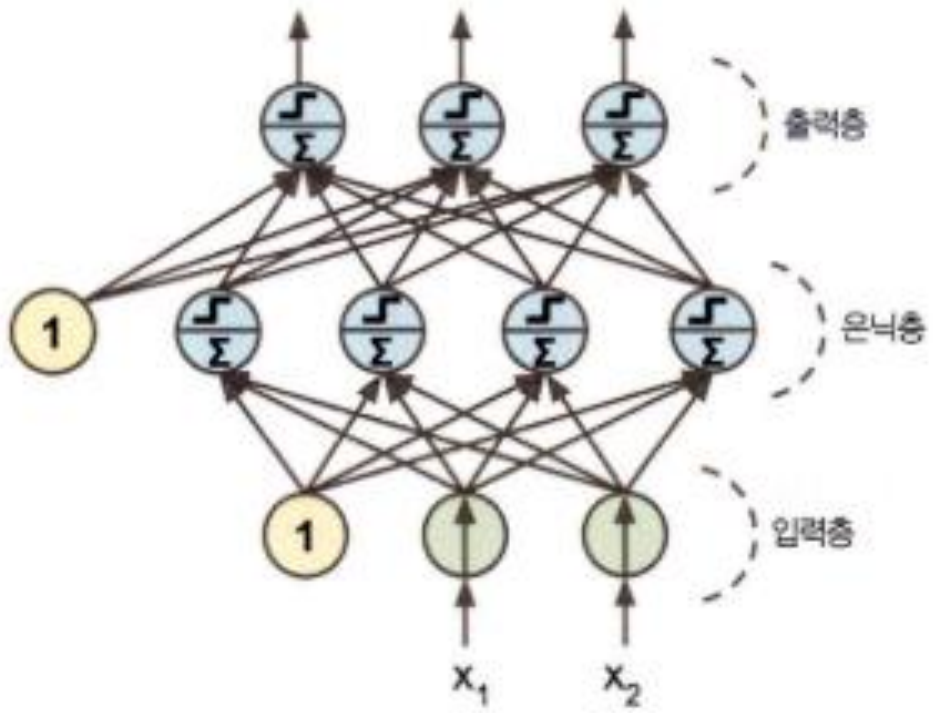
- 퍼센트론의 학습

$$w_{i,j}^{(\text{next step})} = w_{i,j} + \eta(y_j - \hat{y}_j)x_i$$

- $w_{i,j}$: i번째 입력 뉴런과 j번째 출력 뉴런 사이의 가중치
- η : 학습률

02. 다층 퍼셉트론(MLP, multi-layer perceptron)

- 기본 구조



- 입력층, 하나 이상의 은닉층, 출력층으로 구성
- **심층 신경망**(DNN, deep neural network) :

  은닉층을 쌓아 올린 인공신경망

- 한 방향으로만 흐르므로 **피드포워드 신경망**

  (feedforward neural network)

02. 다층 퍼셉트론(MLP, multi-layer perceptron)

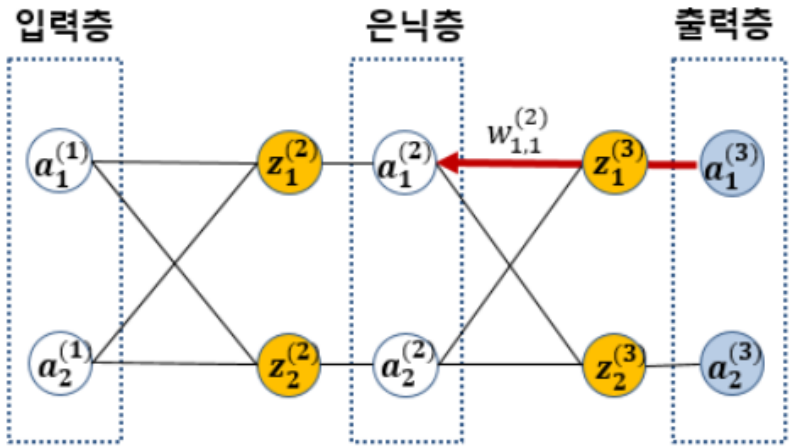
- MLP의 훈련

  역전파 알고리즘(backpropagation) – 후진 모드 자동 미분(reverse-mode autodiff)

Step 1. 정방향 계산 : 출력층까지 계산을 하여 오차를 측정

Step 2. 역방향 계산 : 모든 연결 가중치에 대한 오차 그레디언트(Gradient)를 계산

Step 3. 경사하강법 : 오차가 감소하도록 가중치를 조정



$$\frac{\partial J_{total}}{\partial w_{1,1}^{(2)}} = \frac{\partial J_1}{\partial a_1^{(3)}} \times \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \times \frac{\partial z_1^{(3)}}{\partial w_{1,1}^{(2)}}$$

연쇄 법칙을 이용한 후진 모드 자동 미분 (Step 2)

02. 다층 퍼셉트론(MLP, multi-layer perceptron)

- 다양한 활성화 함수

> Sigmoid 함수(로지스틱 함수)

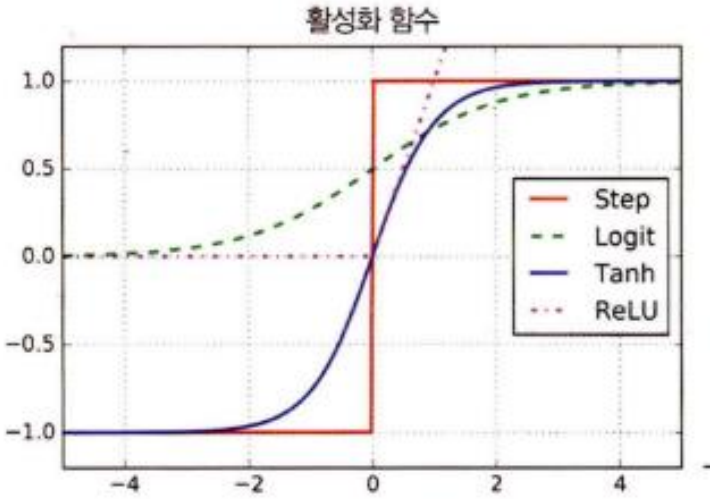
$$\sigma(z) = \frac{1}{1+e^{-z}}$$

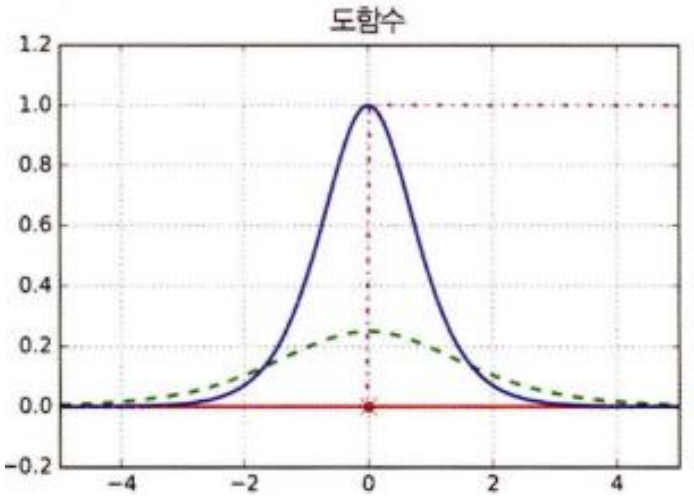
- 출력 범위 : (0, 1)

> Tanh 함수

$$tanh(z) = \frac{1-e^{-z}}{1+e^{-z}} = \frac{2}{1+e^{-2z}} - 1$$

- 출력 범위 : (−1, 1)
- 훈련 초기에 각 층의 출력을 원점 근처로 모아 빠르게 수렴됨





02. 다층 퍼셉트론(MLP, multi-layer perceptron)

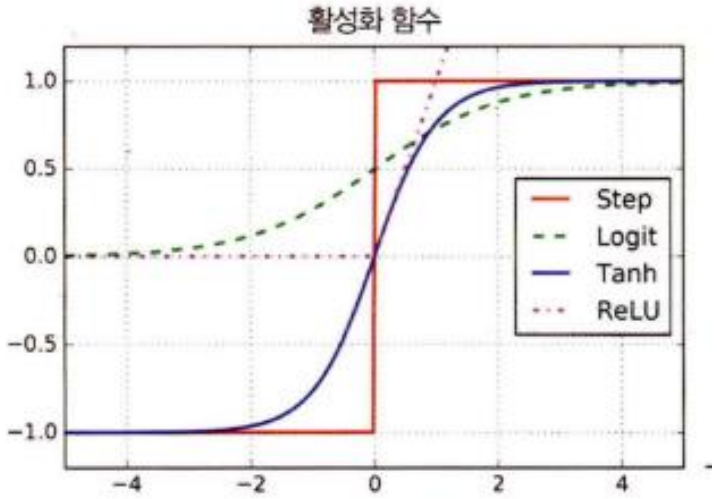
- 다양한 활성화 함수

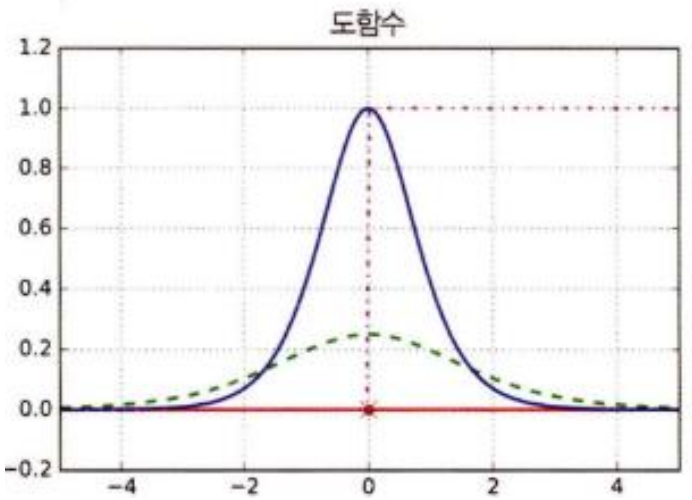
> ReLU 함수

$ReLU(z) = max(0, z)$

- z=0일 때, 미분 불가능이지만, 잘 작동하고 계산속도가 빠름
- 출력에 최댓값 X -> 일부 문제 완화
- 기울기가 항상 1이므로 오차 그레이언트를 그대로 역전파

cf) Sigmoid 함수나 Tanh 함수는 양극단에서 기울기 급감 -> 역전파 잘 못 함





02. 다층 퍼셉트론(MLP, multi-layer perceptron)

- MLP for Regression

하이퍼파라미터	일반적인 값
입력 뉴런 수	특성마다 하나(예를 들어 MNIST의 경우 28×28 = 784)
은닉층 수	문제에 따라 다름. 일반적으로 1에서 5 사이
은닉층의 뉴런 수	문제에 따라 다름. 일반적으로 10에서 100 사이
출력 뉴런 수	예측 차원마다 하나
은닉층의 활성화 함수	ReLU(또는 SELU, 11장 참조)
출력층의 활성화 함수	없음, 또는 (출력이 양수일 때) ReLU/softplus나 (출력을 특정 범위로 제한할 때) logistic/tanh를 사용
손실 함수	MSE나 (이상치가 있다면) MAE/Huber

cf) 후버(Huber) 손실

$$L_\delta(e) = \begin{cases} \frac{1}{2}e^2 & ,for\ |e| \le \delta \\ \delta\left(|e| - \frac{1}{2}\delta\right) & ,otherwise \end{cases}$$

- MSE와 MAE의 w절충안
- 오차가 δ보다 작으면 이차함수, 크면 일차함수(이상치에 덜 민감)

- MLP for Classification

하이퍼파라미터	이진 분류	다중 레이블 분류	다중 분류
입력층과 은닉층	회귀와 동일	회귀와 동일	회귀와 동일
출력 뉴런 수	1개	레이블마다 1개	클래스마다 1개
출력층의 활성화 함수	로지스틱 함수	로지스틱 함수	소프트맥스 함수
손실 함수	크로스 엔트로피	크로스 엔트로피	크로스 엔트로피

> Softmax 함수

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad ,j = 1, \dots, K$$

- 출력값의 합이 1 (클래스가 배타적인 경우)

> 크로스 엔트로피(Cross-entropy loss, 또는 로그 손실)

$$D(\hat{Y}, Y) = -Y\ log\ \hat{Y}$$

03. 구현하기 – 시퀀셜 API로 모델 만들기

- 모델 만들기

```
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape = [28,28]))
model.add(keras.layers.Dense(300, activation = 'relu'))
model.add(keras.layers.Dense(100, activation = 'relu'))
model.add(keras.layers.Dense(10, activation = 'softmax'))
```

- Flatten 층 : 입력 이미지를 1D 배열로 변환. 파라미터를 가지지 않음 (keras.layers.InputLayer로도 가능)
- 다음은 뉴런 300개, 100개를 가진 Dense 은닉층
- 마지막으로 뉴런 10개를 가진 출력층(배타적인 클래스이므로 softmax 사용)

- 모델 확인 및 컴파일

```
hidden1 = model.layers[1]
weights, biases = hidden1.get_weights()
```

```
model.compile(loss = 'sparse_categorical_crossentropy',
              optimizer = 'sgd',
              metrics = ['accuracy'])
```

- 클래스가 배타적이므로 sparse_categorical_crossentropy 사용
- if 클래스별 확률을 가지고 있다면(원핫벡터) categorical_crossentropy 손실 사용
- if 이진분류,
  softmax -> sigmoid,
  binary_crossentropy 손실 사용
- for 회귀, mean_squared_error 손실 사용

03. 구현하기 – 시퀀셜 API로 모델 만들기

1. 모델 만들기

```
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape = [28,28]))
model.add(keras.layers.Dense(300, activation = 'relu'))
model.add(keras.layers.Dense(100, activation = 'relu'))
model.add(keras.layers.Dense(10, activation = 'softmax'))
```

- Flatten 층 : 입력 이미지를 1D 배열로 변환. 파라미터를 가지지 않음 (keras.layers.InputLayer로도 가능)
- 다음은 뉴런 300개, 100개를 가진 Dense 은닉층
- 마지막으로 뉴런 10개를 가진 출력층(배타적인 클래스이므로 softmax 사용)

2. 모델 확인 및 컴파일

```
hidden1 = model.layers[1]
weights, biases = hidden1.get_weights()
```

```
model.compile(loss = 'sparse_categorical_crossentropy',
              optimizer = 'sgd',
              metrics = ['accuracy'])
```

- 클래스가 배타적이므로 sparse_categorical_crossentropy 사용
- if 클래스별 확률을 가지고 있다면(원핫벡터) categorical_crossentropy 손실 사용
- if 이진분류,
  softmax -> sigmoid,
  binary_crossentropy 손실 사용
- for 회귀, mean_squared_error 손실 사용

03. 구현하기 – 시퀀셜 API로 모델 만들기

3. 모델 훈련과 평가

```
history = model.fit(X_train, y_train, epochs = 30,  # epoch 디폴트 : 1
                    validation_data = (X_valid, y_valid)) # valid는 선택
```

```
model.evaluate(X_test, y_test)
y_proba = model.predict(X_new)
```

- if 클래스가 불균형, class_weight 매개변수 지정하는게 좋음 (적게 등장하는 클래스는 높은 가중치를 많이 등장하는 클래스는 낮은 가중치를 부여)
- if 샘플별로 가중치를 부여하고 싶다면, sample_weight 매개변수 지정(class_weight와 sample_weight를 곱하여 사용됨)

4. 파라미터 튜닝하기

```
#GridSearchCV등을 사용하기위해 케라스 모델을 사이킷런 추정기처럼(RandomForest처럼)

def build_model(n_hidden = 1, n_neurons = 30, learning_rate = 3e-3,
input_shape = [8]):
    model = keras.models.Sequential()
    model.add(keras.layers.InputLayer(input_shape=input_shape))
    for layer in range(n_hidden):
        model.add(keras.layers.Dense(n_neurons, activation='relu'))
    model.add(keras.layers.Dense(1))
    optimizer = keras.optimizers.SGD(lr = learning_rate)
    model.compile(loss='mse', optimizer = optimizer)
    return model

keras_reg = keras.wrappers.scikit_learn.KerasRegressor(build_model)
```

```
rnd_search_cv = RandomizedSearchCV(keras_reg, param_distribs,
                                   n_iter = 10, cv = 3)
rnd_search_cv.fit(X_train, y_train, epochs = 100,
                  validation_data = (X_valid, y_valid),
                  callbacks = [keras.callbacks.EarlyStopping(patience =
10)])
```

Thank you