10장 케라스를 사용한 인공신경망 소개

- 인공신경망
    - 딥러닝의 핵심
    - 강력하며 확장성이 좋음

- 인공신경망의 응용: 대규모 머신러닝 문제 다루기에 적합
    - 구글 이미지: 수백만 개의 이미지 분류
    - 애플의 시리: 음성인식 서비스
    - 유튜브: 가장 좋은 비디오 추천
    - 딥마인드의 알파고: 스스로 기보를 익히면서 학습하는 바둑 프로그램

- 퍼셉트론
    - 가장 단순한 인공신경망 구조 중 하나
    - 프랑크 로젠블라트가 1957년에 발표.

주요 내용

- 퍼셉트론 소개
- 다층 퍼셉트론과 역전파
- 케라스(keras)를 이용한 다층 퍼셉트론 구현
- 텐서보드를 활용한 시각화
- 신경망 하이퍼파라미터 튜닝

퍼셉트론 소개

- TLU(threshold logic unit) 또는 LTU(linear threshold unit) 라 불리는 인공 뉴런 활용
- 입력/출력: 숫자
- 모든 입력은 가중치와 연결됨.

TLU

- 입력값과 가중치의 곱한 값들의 합에 계단함수(step function) 적용.

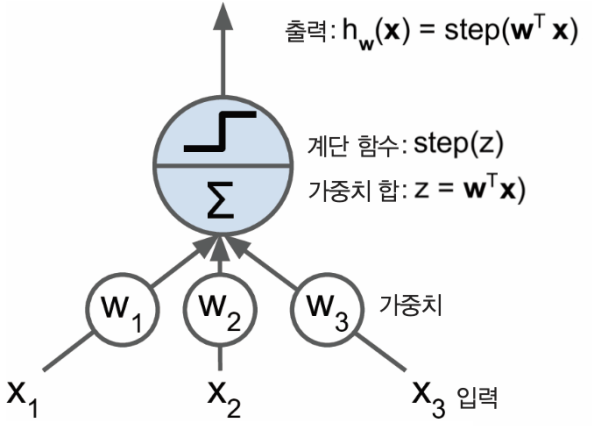
$$
\begin{aligned}
z &= \mathbf{x}^T \mathbf{w} \\
&= x_1 w_1 + x_2 w_2 + \cdots + x_n w_n
\end{aligned}
$$

$$
h_{\mathbf{w}}(\mathbf{x}) = step(z)
$$

- n: 특성 수

예제

- 3개의 특성을 갖는 샘플 하나가 입력되면 가중치와 곱한 후 합을 계산
- 이후 계단함수를 통과시킨 결과를 출력
- 보통은 편향에 대한 가중치고 함께 고려. (잠시 뒤에 설명)



계단함수

- 가장 많이 사용되는 계단함수: Heaviside 계단함수와 sign 함수

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

TLU와 선형 이진분류

- 하나의 TLU를 간단한 이진분류기로 활용 가능
- 모든 입력값(특성)의 선형 조합을 계산한 후에 임곗값을 기준으로 양성/음성 분류
- 작동은 로지스틱 회귀 또는 선형 SVM 분류기와 비슷.
- TLU 모델 학습 = 최적의 가중치 w_i를 찾기

퍼셉트론 정의

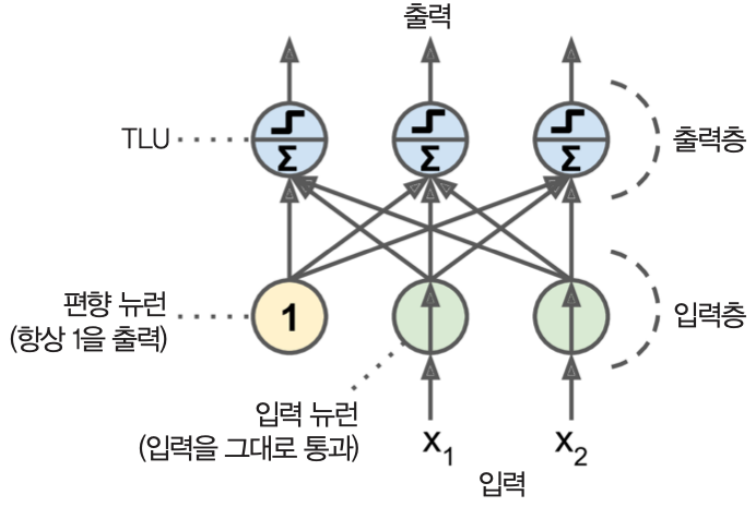
- 하나의 층에 여러 개의 TLU로 구성됨.
- TLU 각각은 모든 입력과 연결됨.

입력층

- 입력 뉴런으로 구성된 층
- 입력 뉴런: 입력을 받아 그대로 통과시켜 출력하는 뉴런
- 편향값 1을 항상 출력하는 편향 뉴런과 여러 개의 입력 뉴런이 함께 사용됨.

예제

- 입력 두 개와 출력 세 개로 구성된 퍼셉트론

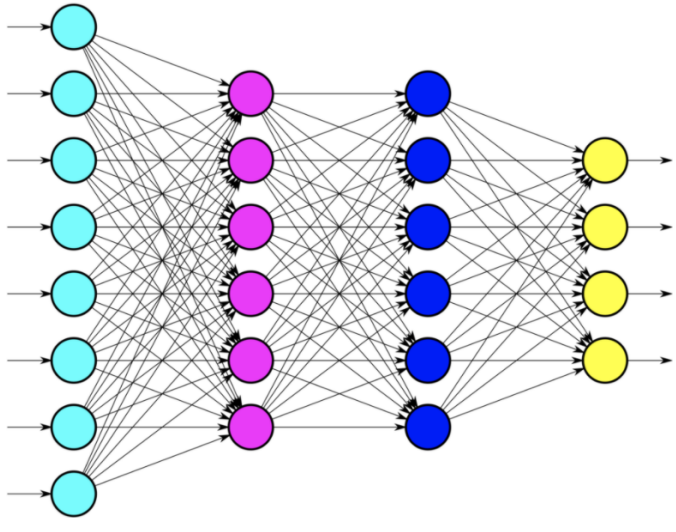


완전연결 층(밀집 층)

- 층에 속한 각각의 뉴런이 이전 층의 모든 뉴런과 연결되어 있을 때를 가리킴.
- 예제: 퍼셉트론의 출력층

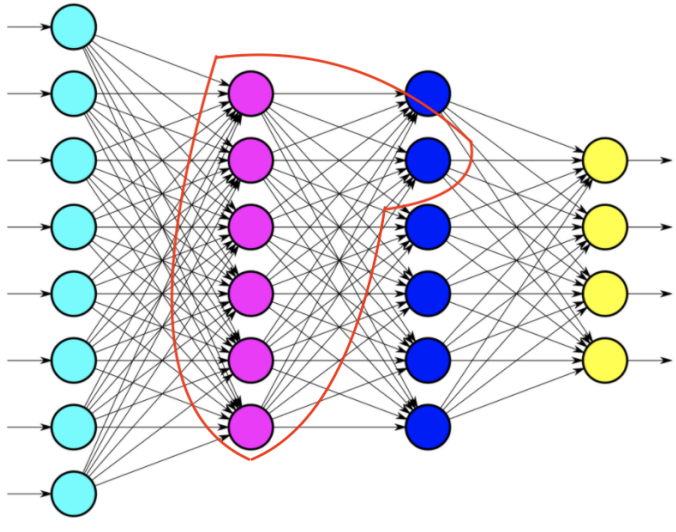
완전연결 층의 출력 계산

- 여러 개의 완전연결 층으로 구성된 다층 퍼셉트론 모델의 일반적인 형태

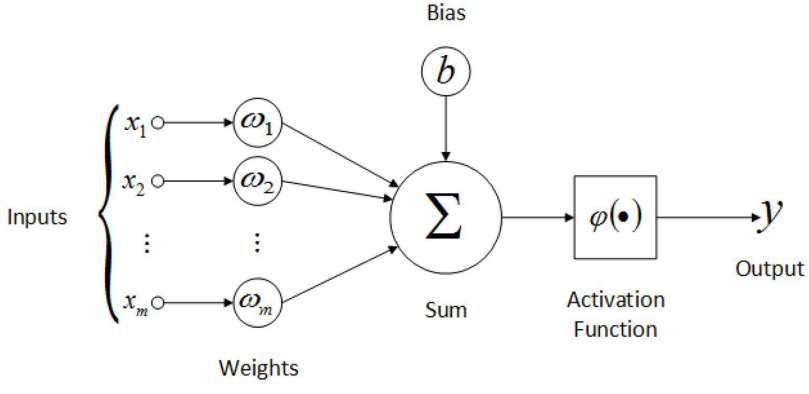


<그림 출처: netclipart (https://www.netclipart.com/down/ihwJhJh_angle-symmetry-area-neural-networks-transparent-backgrounds/)>

- 이 중에서 아래 빨간색으로 이루어진 부분에 대한 계산은 어떻게?

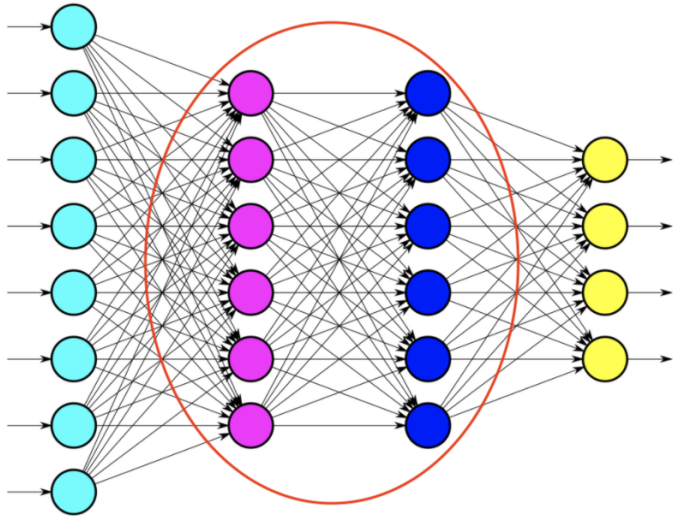


- 다음과 같이 이루어짐.



<그림 출처: medium (https://medium.com/@jayeshbahire/the-artificial-neural-networks-handbook-part-4-d2087d1f583e)>

- 하나의 층에서 이루어지는 입력과 출력을 행렬 수식으로 표현 가능



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

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

- $\mathbf{X}$: 입력 샘플들의 특성행렬.
    - 각 행은 샘플을 가리킴
    - 각 열은 특성을 가리킴

- $\mathbf{W}$: 편향 뉴런을 제외한 모든 뉴런에 대한 가중치.
    - 각 행은 하나의 입력과 연관됨
    - 각 열은 하나의 출력과 연관됨.

- $\mathbf{b}$: 편향벡터
    - 편향 뉴런과 연결된 각각의 출력에 대한 편향값으로 구성된 벡터

- ϕ: 활성화 함수(activation function)
    - 퍼셉트론 모델처럼 각 인공뉴런이 TLU인 경우, 계단함수가 사용됨.

퍼셉트론 학습 알고리즘

- 오차가 감소되도록 가중치를 조절하며 뉴런 사이의 관계를 강화시킴.

- 하나의 샘플이 입력될 때 마다 예측한 후에 오차를 계산하여 오차가 줄어드는 방향으로 가중치 조절.

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

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

- $w_{i,j}$: 가중치 행렬 W의 i행 j열의 값.
- x_i: 입력 샘플의 i번째 속성
- $\hat{y}_j$: j번째 출력값
- y_j: j번째 출력값에 대한 타깃
- η: 학습률

퍼셉트론과 선형성

- 각 출력 뉴런의 결정경계가 선형
- 따라서 퍼셉트론도 복잡한 패턴 학습 못함. 하지만 ...
- 퍼셉트론 수렴 이론: 선형적으로 구분될 수 있는 모델은 언제나 학습 가능

사이킷런의 퍼셉트론 모델

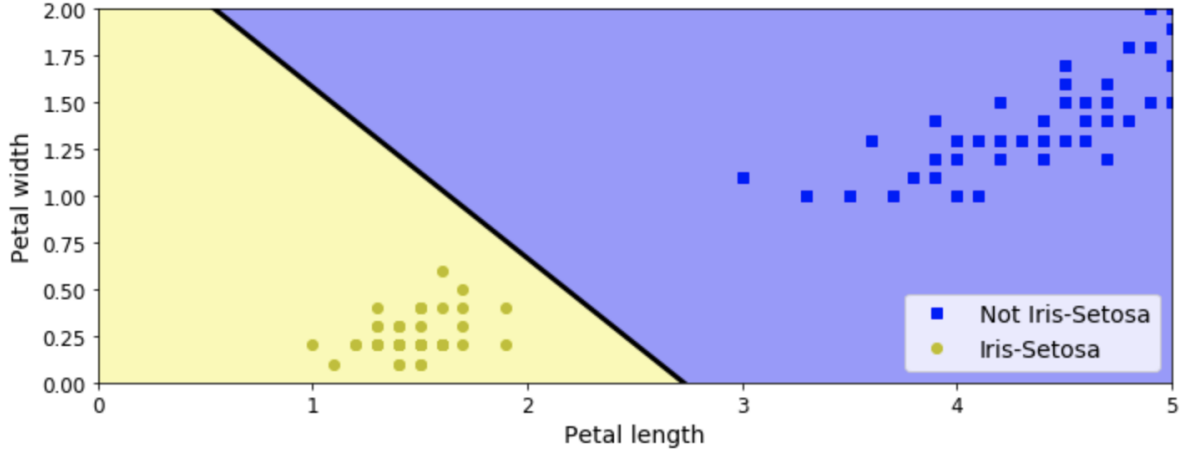
- Perceptron 클래스 활용

예제: 붓꽃 데이터셋 분류

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import Perceptron

iris = load_iris()
X = iris.data[:, (2, 3)]  # petal length, petal width
y = (iris.target == 0).astype(np.int)

per_clf = Perceptron(max_iter=1000, tol=1e-3, random_state=42)
per_clf.fit(X, y)
```


2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Petal width
0
1
2
3
4
5
Petal length
Not Iris-Setosa
Iris-Setosa

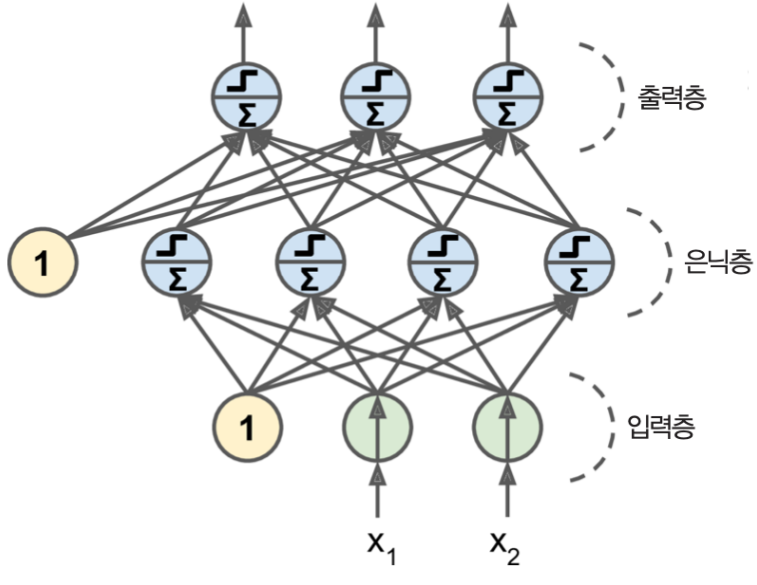
퍼셉트론의 특징

- 아래 옵션을 사용할 경우 SGDClassifier 와 동일하게 작동
    - `loss="perceptron"`
    - `learning_rate="constant"`
    - `eta0=1, penalty=None`
- 클래스 확률 지원 없음. 따라서 로지스틱 회귀가 보다 선호됨.
- 퍼셉트론은 매우 단순한 경우만 해결 가능.
- 하지만 퍼셉트론을 여러 개 쌓아올리면 꽤 강력한 인공신경망 구성함.

다층 퍼셉트론(MLP)과 역전파

- 다층 퍼셉트론(multilayer perceptron, MLP): 퍼셉트론을 여러 개 쌓아올린 인공신경망

- 구성은 다음과 같음.
    - 한 개의 입력층
    - 여러 개의 은닉층
    - 한 개의 출력층



심층신경망(DNN)

- 여러 개의 은닉층을 쌓아올린 인공신경망

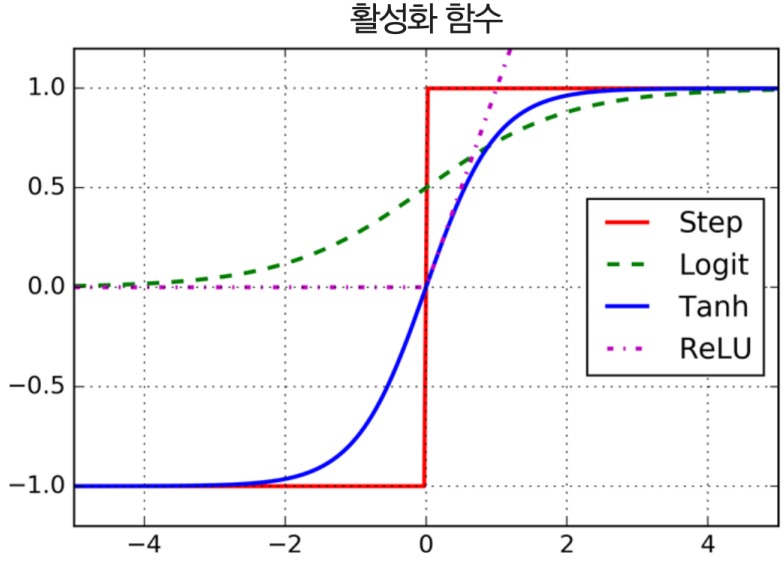
역전파 훈련 알고리즘

- 다층 퍼셉트론은 층이 많을 수록 훈련시키는 과정이 점점 더 어려워짐.
- 1986년에 소개된 역전파(backpropagation) 훈련 알고리즘이 발표된 이후로 실용성 갖춤.
- 1단계(정방향): 각 훈련 샘플에 대해 먼저 예측을 만든 후 오차 측정
- 2단계(역방향): 역방향으로 각 층을 거치면서 각 연결이 오차에 기여한 정도 측정
- 3단계: 오차가 감소하도록 모든 가중치 조정

MLP 특징

- 랜덤하게 설정함. 그렇지 않으면 층의 모든 뉴런이 동일하게 움직임.

- 활성화 함수: 보통 계단함수 대신에 다른 함수 사용.
    - 로지스틱(시그모이드)
    - 하이퍼볼릭 탄젠트 함수(쌍곡 탄젠트 함수)
    - ReLU 함수



활성화 함수 대체 필요성

- 선형성을 벗어나기 위해.
    - 선형 변환을 여러 개 연결 해도 선형 변환에 머무름.
    - 따라서 복잡한 문제 해결 불가능.

- 비선형 활성화 함수를 충분히 많은 층에서 사용하면 매우 강력한 모델 학습 가능

회귀를 위한 MLP

출력 뉴런 수

- 예측해야 하는 값의 수에 따라 출력 뉴런 설정

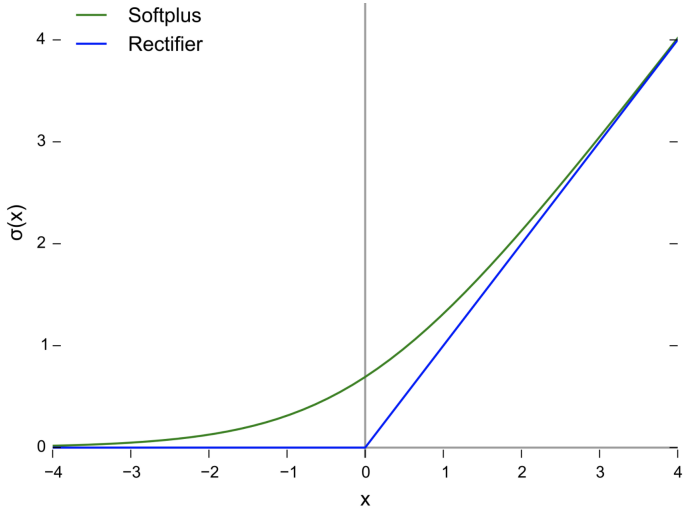
- 예제 1: 주택 가격 예측
    - 출력 뉴런 1개

- 예제 2: 다변량 회귀(동시에 여러값 예측하기)
    - 출력 차원마다 출력 뉴런 1개
    - 예를 들어, 물체의 중심 위치를 알아내려면 좌표 2개 각각에 해당하는 출력 뉴런 2개 필요.

활성화 함수 지정

- 출력값에 특별한 제한이 없다면 활성화 함수 사용하지 않음.

- 출력이 양수인 경우
    - ReLU 또는 softplus 사용 가능



<그림 출처: 위키피디아 (https://en.wikipedia.org/wiki/Rectifier_(neural_networks))>

- 출력이 특정 범위 안에 포함되어야 할 경우
    - 로지스틱 함수 또는 하이퍼볼릭 탄젠트 함수와 적절한 스케일 조정 활용

손실함수

- 일반적으로 평균제곱오차(MSE) 활용
- 이상치가 많을 경우: 평균절댓값오차(MAE) 사용 가능
- 후버(Huber) 손실 사용 가능
    - MSE와 MAE의 조합

회귀 MLP의 전형적인 구조

하이퍼파라미터	일반적으로 사용되는 값
입력뉴런 수	샘플의 특성마다 하나
은닉층 수	문제에 따라 다름. 보통 1-5개
은닉층의 뉴런 수	문제에 따라 다름. 보통 10-100개
출력뉴런 수	예측 차원마다 하나
은닉층의 활성화 함수	ReLU 또는 SELU(11장 참조)
출력층의 활성화 함수	보통 없음. 상황에 따라 ReLU/softplus 또는 logistic/tanh 사용
손실함수	MSE 또는 MAE/Huber(이상치 많은 경우)

분류를 위한 다층 퍼셉트론

이진분류

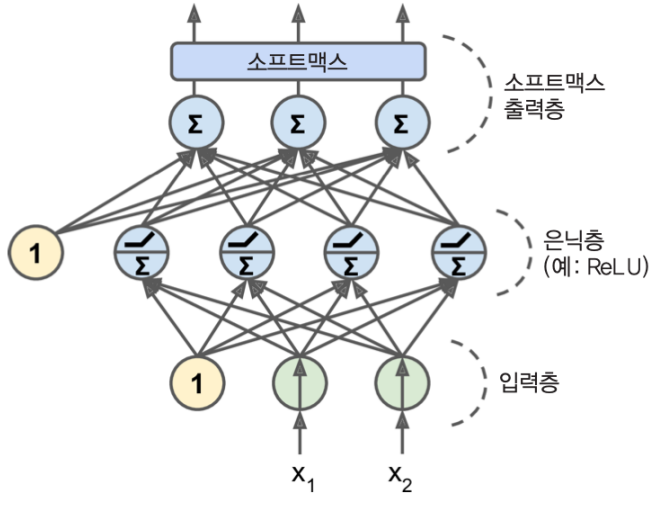
- 하나의 출력 뉴런 사용
- 활성화 함수: 로지스틱 함수

다중레이블 이진분류

- 다층 퍼셉트론 활용

- 예제 1: 이메일의 스팸 여부와 함께 긴급메일 여부 확인 가능
    - 활성화 함수: 두 뉴런 모두 로지스틱 함수 사용

- 예제 2: 다중 클래스 분류
    - 3개 이상의 클래스 중 하나의 클래스에만 속해야 하는 경우
    - 예를 들어, MNIST 숫자 이미지. 0부터 9까지.
    - 클래스마다 하나의 뉴런 사용
    - 출력층 활성화 함수: 소프트맥스 함수



손실함수

- 크로스 엔트로피(로그 손실). 4장 참조.

분류 MLP의 전형적인 구조

하이퍼파라미터	이진 분류	다중레이블 분류	다중클래스 분류
입력층과 은닉층	회귀와 동일	회귀와 동일	회귀와 동일
출력뉴런 수	1개	레이블 당 1개	클래스 당 1개
출력층의 활성화 함수	로지스틱 함수	로지스틱 함수	소프트맥스 함수
손실함수	크로스 엔트로피	크로스 엔트로피	크로스 엔트로피

텐서플로 플레이그라운드 활용

- 텐서플로 플레이그라운드 (https://playground.tensorflow.org) 를 이용한 이진 분류기 훈련해 볼 것.

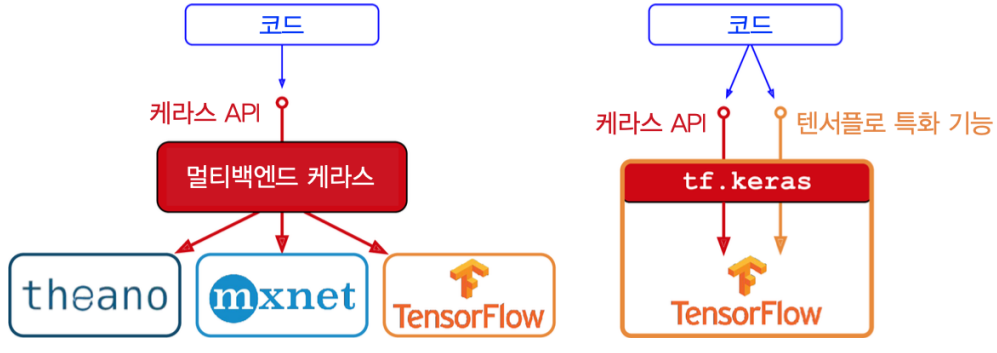
케라스(keras)를 이용한 MLP 구현

- 케라스 (https://keras.io)는 모든 종류의 신경망을 손쉽게 만들어 주는 최상위 API 제공.

- 멀티 백엔드 케라스:
    - 구글의 텐서플로(Tensorflow),
    - MS의 Cognitive Toolkit(CNTK),
    - Theano(시애노)
    - 아파치 MXNet
    - 애플 Core ML
    - 자바스크립트, 타입스크립트: 웹브라우저에서 케라스 실행 가능
    - PlaidML: 모든 종류의 GPU에서 실행 가능

tensorflow.keras

- 텐서플로만 지원하는 keras 백엔드
- 책에서 사용



파이토치(PyTorch)

- 파이토치 (https://pytorch.org)는 keras와 비슷한 API 제공하며, 쉽게 배울 수 있음.
- keras만큼 인기 좋음.

케라스 시퀀셜 API 활용: 분류

패션 MNIST 활용

- 10개의 클래스로 이루어짐.
- 데이터 셋: 28x28 픽셀 크기의 흑백 패션 이미지 샘플 70,000개

모델 선언

- 은닉층: 2개

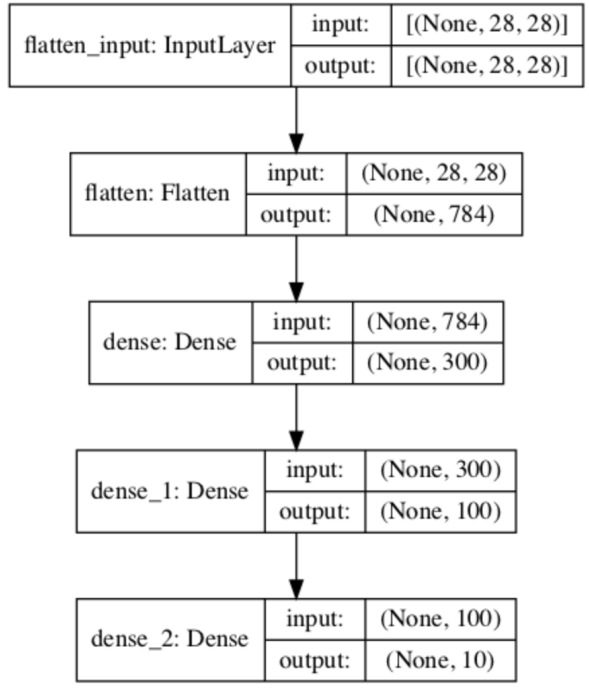
```
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28, 28]))
model.add(keras.layers.Dense(300, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(10, activation="softmax"))
```

- 아래 방식도 가능

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

- 아래 명령어를 사용하면 모델 선언된 모델 확인 가능

```
keras.utils.plot_model(model, "my_fashion_mnist_model.png", show_shapes=True)
```


flatten_input: InputLayer
input: [(None, 28, 28)]
output: [(None, 28, 28)]
flatten: Flatten
input: (None, 28, 28)
output: (None, 784)
dense: Dense
input: (None, 784)
output: (None, 300)
dense_1: Dense
input: (None, 300)
output: (None, 100)
dense_2: Dense
input: (None, 100)
output: (None, 10)

모델 컴파일하기

- 모델을 생성하려면 선언된 모델을 컴파일 해야함.
- 손실함수, 옵티마이저, 평가기준 등을 지정해야 함.

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

모델 훈련과 평가

- 훈련: fit 메서드 호출
- 평가 동시진행 가능

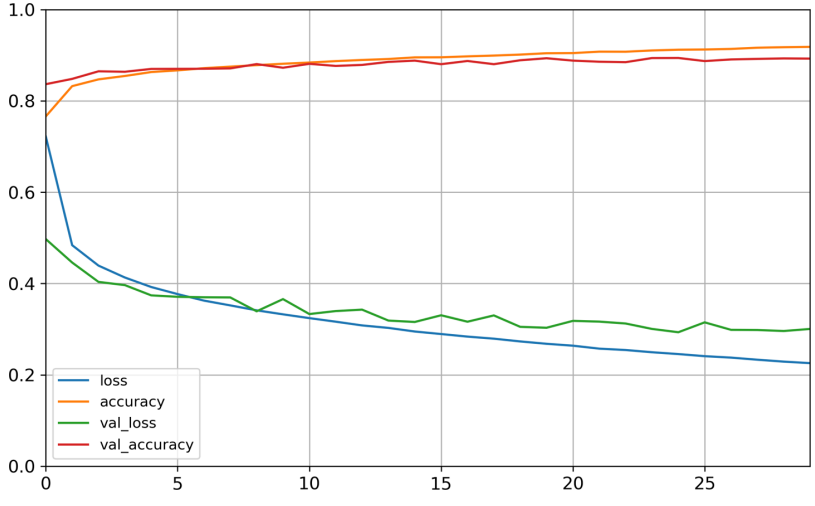
```
history = model.fit(X_train, y_train, epochs=30,
                    validation_data=(X_valid, y_valid))
```

모델 학습 곡선

- 훈련된 모델이 반환하는 History 객체의 history 속성에 학습된 에포크의 손실과 정확도 기록됨.

```
import pandas as pd

pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1)
save_fig("keras_learning_curves_plot")
plt.show()
```


1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
loss
accuracy
val_loss
val_accuracy

모델 성능 테스트

- 훈련된 모델의 `evaluate` 메서드 활용
- 손실과 정확도 계산해줌.
- 아래 코드 실행결과: 손실은 0.3339, 정확도는 0.8851

```
model.evaluate(X_test, y_test)
```

모델 활용

- 훈련된 모델을 이용한 예측은 `predict` 또는 `predict_class` 메서드 활용.
- `predict` 메서드: 각 클래스에 속할 확률 계산
- `predict_class`: 가장 높은 확률의 클래스 지정

케라스 시퀀셜 API 활용: 회귀

캘리포니아 주택가격 예측

- 시퀀셜 API를 이용한 회귀용 MLP 구축은 분류용과 기본적으로 동일.

- 차이점:
    - 출력층에 활성화함수 사용하지 않는 하나의 뉴런만 사용.
        - 일반적으로는 예측값으 수 만큼 출력뉴런 사용
    - 손실함수: 평균제곱오차(MSE)

캘리포니아 데이터셋과 관련된 주의할 점

- 잡음이 많음.
- 따라서 과대적합을 줄이기 위해 뉴런 수가 적은 하나의 은닉층만 사용.
- 이유: 은닉층과 뉴런이 많이 사용될 수록 가중치 파라미터의 수가 증가하여 과대적합 위험도 커짐.

모델 생성, 훈련 및 평가

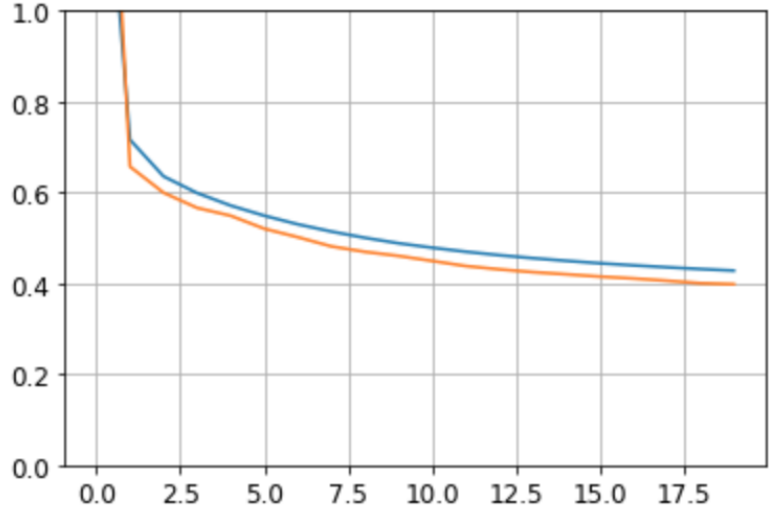
```
model = keras.models.Sequential([
    keras.layers.Dense(30, activation="relu", input_shape=X_train.shape[1:]),
    keras.layers.Dense(1)
    ])

model.compile(loss="mean_squared_error", optimizer=keras.optimizers.SGD(lr=1e-3))

history = model.fit(X_train, y_train, epochs=20, validation_data=(X_valid, y_vali
d))
```

학습 곡선

```
mse_test = model.evaluate(X_test, y_test)
```



케라스 시퀀셜 API의 장단점

- 사용하기 매우 쉬우며 성능 우수함.
- 입출력이 여러 개 이거나 더 복잡한 네트워크를 구성하기 어려움.
- 시퀀셜 API 대신 **함수형** API, **서브클래싱(subclassing)** API 등을 사용하여 보다 복잡하며, 보다 강력한 딥러닝 모델 구축 가능.