


# DNN\_01. 퍼셉트론, MLP

🕒 생성일	@2022년 6월 9일 오후 3:26
📁 유형	머신러닝/딥러닝
👤 작성자	 동훈 오

딥러닝 (DNN) 은 여러 뉴런의 연결을 통해 신경망을 구현한다. input, output이 존재하는 뉴런은 사실상 함수 필터(말 그대로)라고 생각할 수 있고, 의도하고자 하는 방향으로 정보 가공할 수 있는 함수를 선택하는 것이 중요하다. (‘의도하고자 하는 방향’ 이란 데이터에서 숨겨진 특성을 추적하는 것, 정보의 일부를 왜곡시키는 것, 정보의 일부를 끝까지 보존하는 것 등 다양하게 해석할 수 있다.)

신경망에서 하나의 정보가 주입되었을 때, 여러 뉴런을 거쳐서 도출되는 최종 결과는 완벽히 예측하기 어렵다. 신경망이 깊어질 수록 더욱 그런 경향이 있다. 매 훈련마다 모델은 업데이트 되기에 변동성이 있으며 학습의 흐름이 진행되는 것이 매번 달라진다. DNN의 이러한 특징 때문에 DNN 분야가경험적 (empirical) 성격이 강한 것도 사실이다.

딥러닝 챕터에서는 학습의 전반적인 흐름과 뉴런(=함수 필터) 에서 어떤 일이 일어나는지, 두 가지를 중점으로 설명할 계획이다.

## 혼공머신: 7장 딥러닝

혼공머신 7장 에서는 fashion MNIST 데이터셋을 이용한 다중 분류 문제를 해결하는 인공 신경망을 생성한다. 2개의 인공 신경망을 생성하고 Dropout callback 을 적용하는 것 까지 목표로 한다.

### 데이터셋

```
from tensorflow import keras
(train_input, train_target), (test_input, test_target) = \
    keras.datasets.fashion_mnist.load_data()

# 훈련 데이터의 크기 살펴보기
print(train_input.shape, train_target.shape)
>>> (60000, 28, 28) (60000,)

# 테스트 데이터
print(test_input.shape, test_target.shape)
>>> (10000, 28, 28) (10000,)

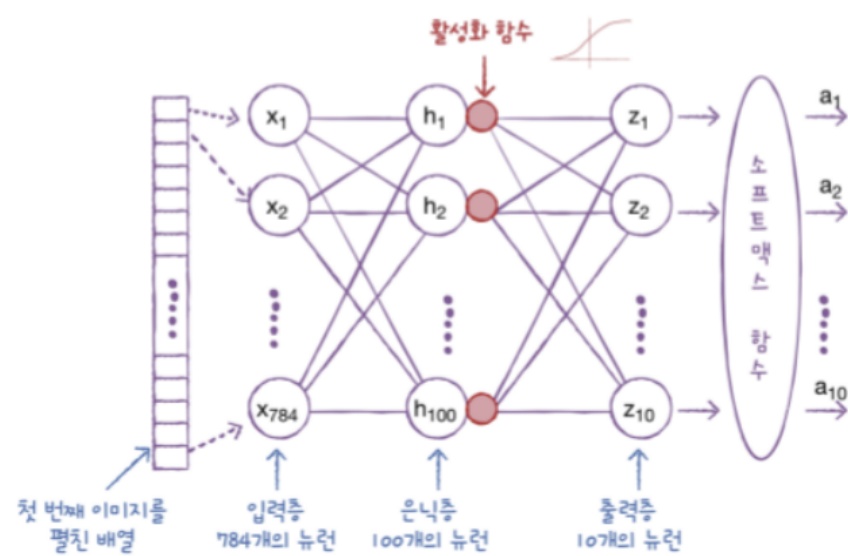
# 클래스 별 데이터 분포 살펴보기
# unique() 함수로 레이블 당 샘플 개수 확인
import numpy as np
print(np.unique(train_target, return_counts=True))

# 이미지 데이터 normalization
# 이미지의 경우 보통 255로 나누어 0-1 사이의 값으로 정규화.
train_scaled = train_input / 255.0
train_scaled = train_scaled.reshape(-1, 28*28)    # 첫번째 차원인 샘플의 개수는 변하지 않고 두 번째, 세번째 차원이 1차원으로 합쳐진다.

# validation set 구성
from sklearn.model_selection import train_test_split
train_scaled, val_scaled, train_target, val_target = train_test_split(
    train_scaled, train_target, test_size = 0.2, random_state=42)
```

### 모델

모델의 대략적인 구조는 다음과 같다.



- 활성화 함수는 신경망 층의 선형 방정식의 계산 값에 적용하는 함수이다.
- 출력층에 적용하는 활성 함수는 종류가 제한되어 있다. 이진 분류일 경우 시그모이드 함수를 사용하고 다중 분류일 경우 소프트맥스 함수를 사용한다.
- 이에 비해 은닉층의 활성화 함수는 비교적 자유롭다. 대표적으로 시그모이드 함수와 렐루 함수 등을 사용한다.
- 회귀의 출력은 임의의 어떤 숫자이므로 활성화 함수를 적용할 필요가 없다.

```
# Sequential 클래스의 생성자 안에서 바로 Dense 클래스의 객체를 만들 수 있다.
model = keras.Sequential([
    # 은닉층, output 크기 100
    keras.layers.Dense(100, activation='sigmoid',
                        input_shape=(784,)),
    # 출력층, output 크기는 클래스와 동일한 10
    keras.layers.Dense(10, activation='softmax')
])

# 모델정보 확인
model.summary()
>>> Model: Sequential_1
```

Layer (type)	Output Shape	Param #
Dense1	(None, 100)	78500
Dense2	(None, 10)	1010

```
Total params: 79,510
Trainable params: 79,510
Non-trainable params: 0
```

위의 모델은 간단한 다층 퍼셉트론 (multi-layers-perceptron) 이다. 인공 신경망의 기초적인 모델이라고 생각하면 된다. 위의 모델 생성 방식은 케라스의 Sequential API 를 사용했다. Sequential() 은 신경망을 구성하는 layer 들을 하나로 묶어 주는 역할을 한다.

다음에 나올 코드는 모델을 생성하는, 조금 더 편리한 방법(고수준 API 의 또다른 방식) 이고, 이미지 데이터를 처리한다는 점에서 Flatten layer 가 추가되고, 은닉층 활성화 함수가 변경되며, 최적화를 위해 optimizer 로 'adam' 이 사용된다.

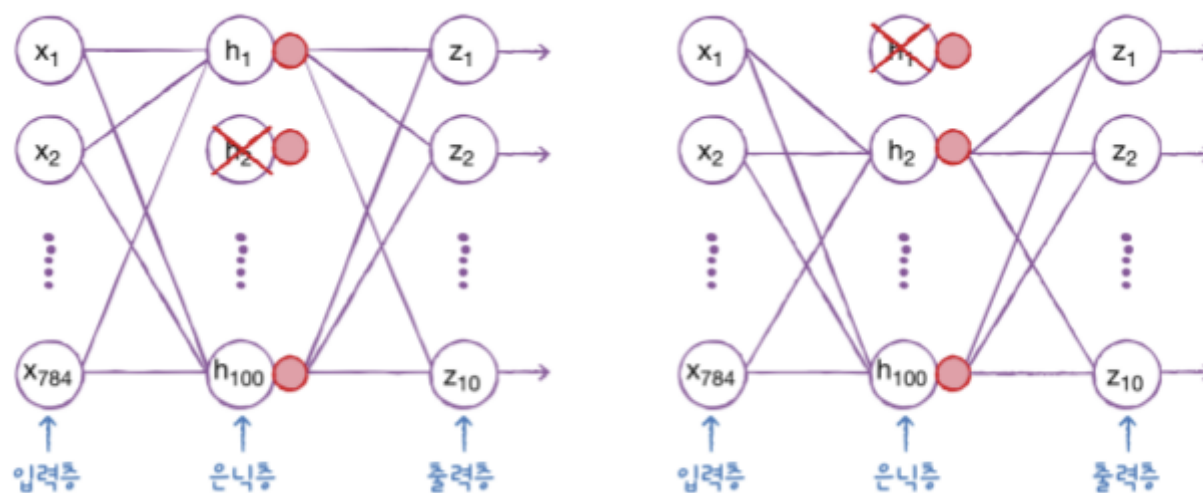
```
model = keras.Sequential()
model.add(keras.layers.Flatten(input_shape=(28, 28)))
model.add(keras.layers.Dense(100, activation='relu'))
model.add(keras.layers.Dense(10, activation='softmax'))
```

```
# 컴파일에서 optimizer, 손실함수, 성능지표 설정.
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics='accuracy')

# 훈련 진행(훈련 횟수(epoch) : 20 지정)
model.fit(train_scaled, train_target, epochs=20, verbose=0,
          validation_data=(val_scaled, val_target))
```

모델 생성부터 훈련 진행까지 코드가 복잡하지 않다. 2개의 층을 가진 비교적 단순한 MLP 이기 때문이며 모델 성능을 높이기 위해 Regularization 방법인 dropout 과 early-stopping을 적용하겠다.

## Dropout



- dropout 은 훈련 과정에서 일부 뉴런을 랜덤하게 off 시켜서 과대 적합을 막는다.
- 얼마나 많은 뉴런을 off 시킬지는 또 다른 하이퍼파라미터가 된다.
- 이전 층의 일부 뉴런이 랜덤하게 꺼지면 특정 뉴런에 과대하게 의존하는 것을 줄일 수 있고 모든 입력에 대해 주의를 기울여야 한다. 일부 뉴런의 출력이 없을 수 있다는 것을 감안하면 이 신경망은 더 안정적인 예측을 만들 수 있을 것이다.
- keras 에서는 어떤 층의 뒤에 dropout 을 두어 이 층의 출력을 랜덤하게 0으로 만든다. dropout 이 층처럼 사용되지만 훈련되는 모델 파라미터는 없다.

## Early-stopping

모델이 학습을 진행하면서 검증 점수가 상승하기 시작하면 그 이후에는 과대적합이 더 커지기 때문에 훈련을 계속할 필요가 없다. early-stopping은 과대적합이 일어나는 시점에서 훈련을 중단하는 것이다.

early-stopping 은 patience 매개변수로 검증 점수가 향상되지 않더라도 참을 epoch 횟수를 지정한다. 예를 들어 patience=2 인 경우 2번 연속 검증 점수가 향상되지 않으면 훈련을 종료한다.

```
model = keras.Sequential()
model.add(keras.layers.Flatten(input_shape=(28, 28)))
model.add(keras.layers.Dense(100, activation='relu'))
# dropout layer
model.add(keras.layers.Dropout(0.3))
model.add(keras.layers.Dense(10, activation='softmax'))

# 컴파일에서 optimizer, 손실함수, 성능지표 설정.
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics='accuracy')
```

```
# early-stopping callback
early_stopping_cb = keras.callbacks.EarlyStopping(patience=2)

# 훈련 진행(훈련 횟수(epoch) : 20 지정)
model.fit(train_scaled, train_target, epochs=20, verbose=0,
          validation_data=(val_scaled, val_target),
          callbacks=[early_stopping_cb])
```

## 핸즈온: 10. 케라스를 사용한 인공 신경망 소개

핸즈온 10장은 혼공머신의 7장과 내용이 유사하다. 그래서 추가적으로 의미있는 내용만 아래에서 설명한다.

### 은닉층 개수

이론적으로 은닉층이 하나인 다층 퍼셉트론이더라도 뉴런 개수가 충분하면 아주 복잡한 함수도 모델링할 수 있다. 하지만 복잡한 문제에서는 심층 신경망이 얇은 신경망보다 파라미터 효율성이 훨씬 좋다. 실제 데이터는 계층 구조를 가진 경우가 많으므로 심층 신경망은 유리하다.

아래쪽 은닉층은 저수준의 구조를 모델링하고, 중간 은닉층은 저수준의 구조를 연결해 중간 수준의 구조를 모델링한다. 그리고 가장 위쪽 은닉층과 출력층은 중간 수준의 구조를 연결해 고수준의 구조를 모델링한다.

계층 구조는 심층 신경망이 좋은 솔루션으로 빨리 수렴하게끔 도와줄 뿐만 아니라 새로운 데이터에 일반화되는 능력도 향상시켜준다. 이미 훈련한 모델을 새로운 데이터에서 새로 훈련하려면 하위 층을 재사용하여 훈련을 시작할 수 있다. 저수준 구조를 학습할 필요가 없게 되며 이러한 방식을 전이 학습(transfer learning) 이라고 한다.

핸즈온 10 장에서 설명하는 내용은 혼공머신 7장 딥러닝 파트와 크게 다르지 않다. 다만, 지금까지 모델을 작성한 방식은 고수준 API 라면, 저수준 api 에서 서브클래싱 방식으로 모델을 생성하는 방법이 핸드온에서 더 잘 설명되어 있다. 아래는 서브클래싱 방식으로 모델을 생성하는 법에 관한 내용이다.

### 10.3 신경망 하이퍼파라미터 튜닝하기

조정할 하이퍼파라미터가 많기 때문에 신경망의 유연성은 단점이기도 하다.

### 은닉층의 뉴런 개수

입력층과 출력층의 뉴런 개수는 해당 작업에 필요한 입력과 출력의 형태에 따라 결정된다. 은닉층의 구성 방식은 일반적으로 각 층의 뉴런을 점점 줄여서 깔때기처럼 구성한다. 저수준의 많은 특성이 고수준의 적은 특성으로 합쳐질 수 있기 때문이다.

층의 개수와 마찬가지로 네트워크가 과대적합이 시작되기 전까지 점진적으로 뉴런 수를 늘릴 수 있다. 하지만 실전에서는 필요한 것보다 더 많은 층과 뉴런을 가진 모델을 선택하고, 그런 다음 과대적합되지 않도록 early-stopping 등의 regularizaion 기법을 사용하는 것이 효과적이다.

### 학습률

일반적으로 최적의 학습률은 최대학습률의 절반 정도이다. 좋은 학습률을 찾는 한 가지 방법은 매우 낮은 학습률에서 시작해 점진적으로 매우 큰 학습률까지 반복하여 모델을 훈련하는 것이다. 학습률에 대한 손실을 그래프로 그리면 처음에는 손실이 줄어드는 것이 보이지만 학습률이

커지게 되면서 특정 시점, 손실이 다시 커진다. 최적은 학습률은 이러한 시점에서보다 조금 이전 시점의 학습률이다.

## 배치 크기

배치 크기는 모델 성능과 훈련 시간에 큰 영향을 미칠 수 있다. 큰 배치 크기를 사용하는 것의 주요 장점은 GPU 와 같은 하드웨어 가속기를 효율적으로 활용할 수 있다는 점이다. 따라서 훈련 알고리즘이 초당 더 많은 샘플을 처리할 수 있다. 하지만 실전에서 큰 배치를 사용시 훈련 초기 불안정하게 훈련될 수 있다.

배치 크기에 대한 관점은 대표적으로 두 가지가 있다. 우선 딥러닝의 아버지 안 르쿤은 2에서 32 사이의 미니배치를 사용하는 방식을 권장한다. 반대로, 작은 학습률로 훈련을 시작해 점점 커지는 warm up 방식을 사용하면 매우 큰 배치 크기를 사용할 수 있다고 주장한다.

---