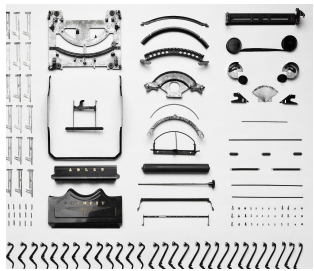
1장 텐스플로 2.0으로 신경망 구현



1장 목차

가급적 수학적인 이해는 간단히 하고, 개념적 이해를 위주로

텐서플로(TensorFlow)/케라스 (Keras)

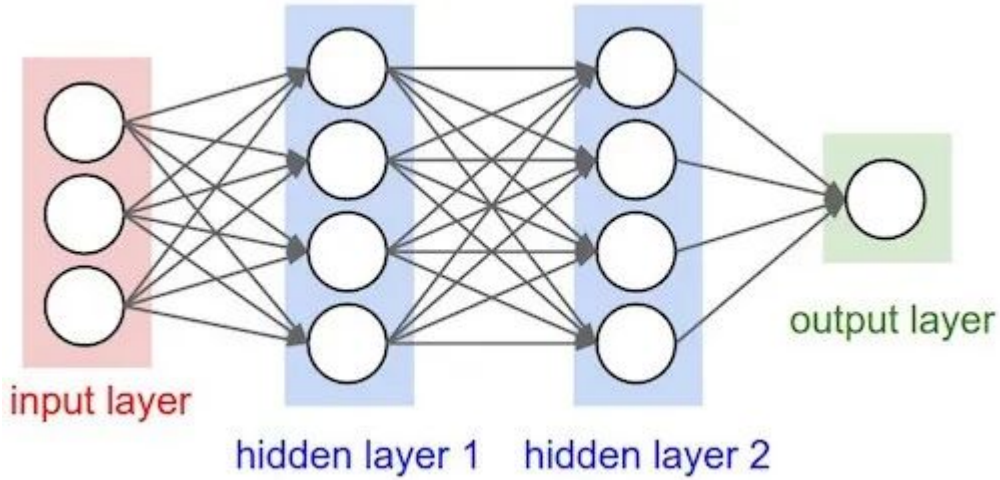
- TensorFlow: 구글 브레인팀이 개발, Deep neural networks을 위한 오픈소스 소프트웨어 프레임워크
- Keras : 거의 모든 종류의 딥러닝 모델을 간편하게 훈련시킬수 파이썬을 위한 API,
- TensorFlow2.0: Keras를 내부에 구현하여 외부 호출 없이 사용가능

```
import tensorflow.compat.v1 as tf

in_a = tf.placeholder(dtype=tf.float32, shape=(2))

def model(x):
   with tf.variable_scope("matmul"):
      W = tf.get_variable("W", initializer=tf.ones(shape=(2,2)))
      b = tf.get_variable("b", initializer=tf.zeros(shape=(2)))
      return x * W + b

out_a = model(in_a)

with tf.Session() as sess:
   sess.run(tf.global_variables_initializer())
   outs = sess.run([out_a],
                   feed_dict={in_a: [1, 0]})
```

```
import tensorflow as tf
W = tf.Variable(tf.ones(shape=(2,2)), name="W")
b = tf.Variable(tf.zeros(shape=(2)), name="b")

@tf.function
def model(x):
   return W * x + b
out_a = model([1,0])

print(out_a)
```

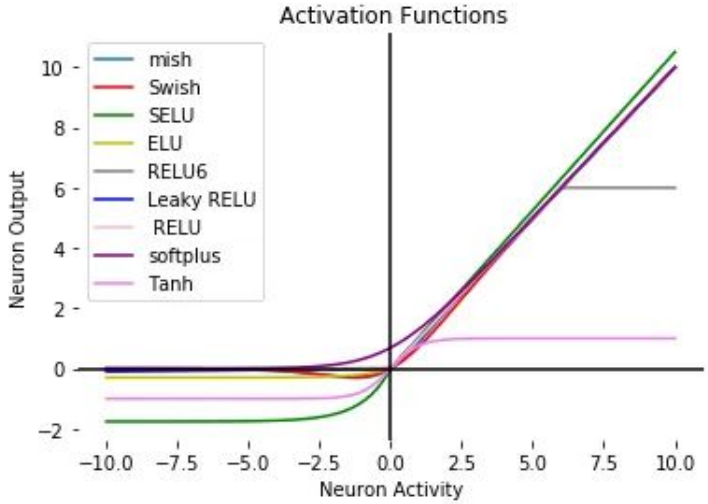
신경망(Neural Network) 개요

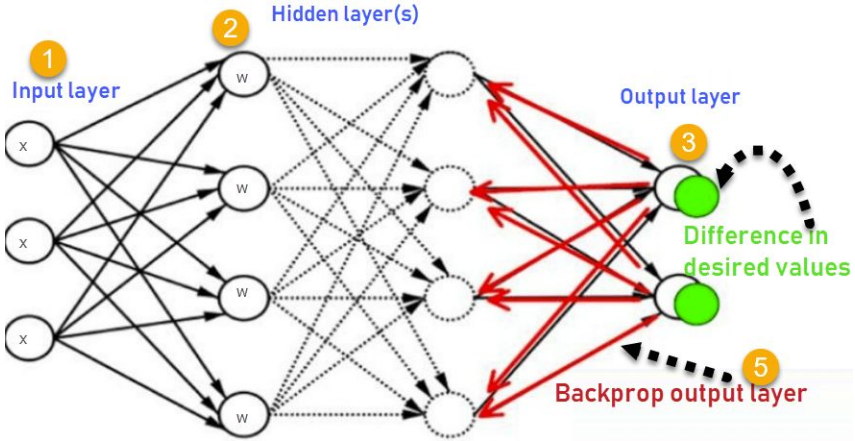
- 신경망 소개: 신경세포를 모방한 인공신경망을 연결한 머신러닝 모델
- 퍼셉트론: 단위 인공신경망세포, 입력층과 출력층만 있음, XOR 연산불가
- 다층퍼셉트론: 입력층과 출력층 사이에 은닉층(hidden layer), XOR 연산가능



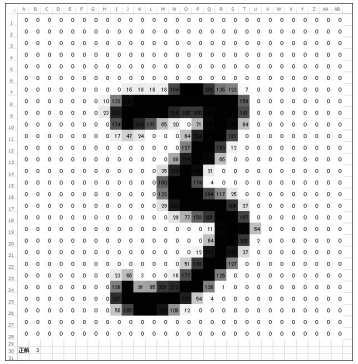
다중퍼셉트론의 학습

- 활성화 함수(Activation Function)
  - 시그모이드
  - tanh
  - ReLU
  - ELU와 LeakyReLU
  - softmax
- 역전파 알고리즘(Backpropagation): 먼저 계산 결과와 정답의 오차를 구해 이 오차에 관여하는 값들의 가중치를 수정하여 오차가 작아지는 방향으로 일정 횟수를 반복해 수정하는 방법

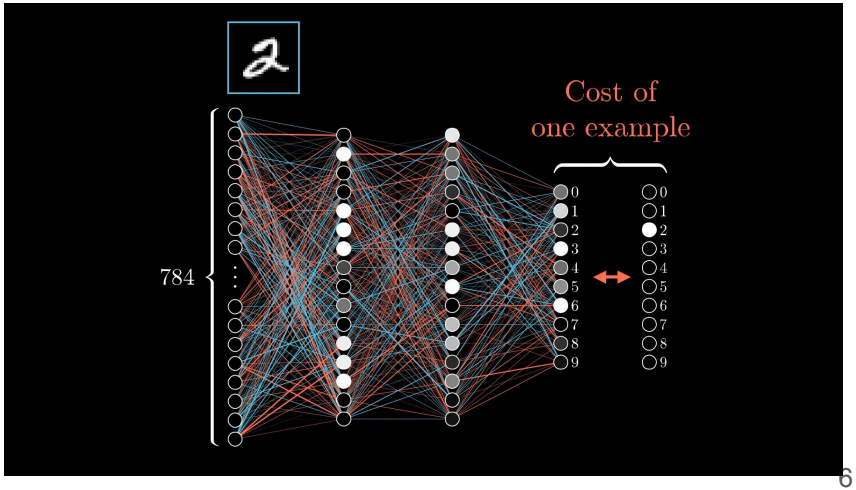




실제 예제: 필기체 숫자 인식



- MNIST 데이터베이스 (Modified National Institute of Standards and Technology database)는 손으로 쓴 숫자들로 이루어진 대형 데이터베이스이며, 다양한 화상 처리 시스템을 트레이닝하기 위해 일반적으로 사용
- MNIST 데이터베이스는 60,000개의 트레이닝 이미지와 10,000개의 테스트 이미지를 포함



Hands-on 1

- 단순신경망
  - Layer 1개 → 정확도 92.2%

```
def build_model(self):
    reshaped = 28 * 28                                      #--- 행열(28 * 28)를 벡터(784 * 1)로 변환
    nb_classes = 10                                         #--- 분류 갯수

    model = tf.keras.models.Sequential()                    #--- 모델 : Sequential
    model.add(keras.layers.Dense(nb_classes, input_shape=(reshaped,),   #--- 출력 갯수, 입력 갯수
            name="dense_layer_1", activation=self.activation_function   #--- Activation Function
    ))

    model.summary()
    model.compile(
        optimizer=self.optimizer,                           #--- Optimizer
        loss=self.loss_function,                            #--- Loss Function
        metrics=[ self.metrics ],                           #--- Matric
    )
    return model
```

```
3012      1/313 [..............................
3013     31/313 [=>............................
3014     66/313 [=====>........................
3015     94/313 [========>.....................
3016    124/313 [==========>...................
3017    151/313 [=============>................
3018    175/313 [===============>..............
3019    203/313 [==================>...........
3020    235/313 [=====================>........
3021    261/313 [========================>.....
3022    296/313 [============================>..
3023    313/313 [==============================
3024    Test accuracy: 0.921999990940094
```

Hands-on 2

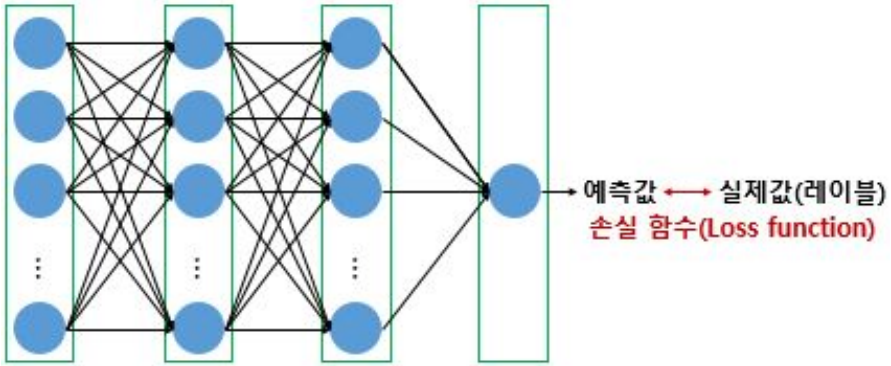
- 단순 신경망을 은닉층으로 개선 (은닉층2개 추가)
  - Layer 3개 → 정확도 96.5%

```
def build_model(self):
    reshaped = 28 * 28                                          #--- 행열(28 * 28)을 벡터(784 * 1)로 변환
    n_hidden = 128
    nb_classes = 10                                             #--- 분류 갯수

    model = tf.keras.models.Sequential()                        #--- 모델 : Sequential
    model.add(keras.layers.Dense(n_hidden, input_shape=(reshaped,),     #--- 출력 갯수, 입력 갯수
            name="dense_layer_1", activation='relu'             #--- Activation Function
    ))
    model.add(keras.layers.Dense(n_hidden,                      #--- 출력 갯수
            name="dense_layer_2", activation='relu'             #--- Activation Function
    ))
    model.add(keras.layers.Dense(nb_classes,                    #--- 출력 갯수
            name="dense_layer_3", activation=self.activation_function   #--- Activation Function
    ))

    model.summary()
    model.compile(
        optimizer=self.optimizer,                               #--- Optimizer
        loss=self.loss_function,                                #--- Loss Function
        metrics=[ self.metrics ],                               #--- Matric
    )
    return model
```

```
 1/313 [..............................] -
26/313 [=>............................] -
54/313 [====>.........................] -
82/313 [======>.......................] -
113/313 [=========>....................] -
144/313 [============>.................] -
173/313 [===============>..............] -
204/313 [==================>...........] -
234/313 [=====================>........] -
261/313 [========================>.....] -
292/313 [===========================>...] -
313/313 [==============================] -
Test accuracy: 0.9650999903678894
```

Optimizer

손실 함수의 값을 줄여나가면서 학습하는 방법은 어떤 옵티마이저를 사용하느냐에 따라 달라집니다. 여기서 배치(Batch)라는 개념에 대한 이해가 필요합니다. 배치는 가중치 등의 매개 변수의 값을 조정하기 위해 사용하는 데이터의 양을 말합니다. 전체 데이터를 가지고 매개 변수의 값을 조정할 수도 있고, 정해준 양의 데이터만 가지고도 매개 변수의 값을 조정할 수 있습니다.

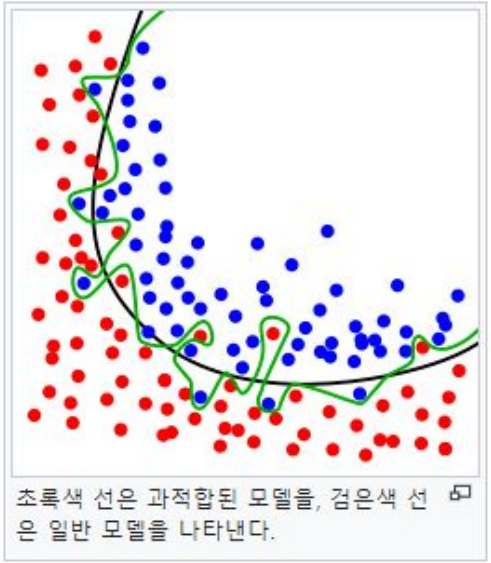


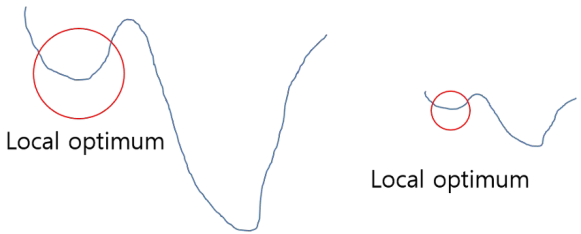
출저 : 딥 러닝을 이용한 자연어 처리 입문 / 07. 딥 러닝(Deep Learning) / 07-04 딥 러닝의 학습 방법

이해를 위한 참고 : 코딩뚠뚠 / [머신러닝 공부]딥러닝/Optimizer정리
Truman Show | 손실 함수(Loss Function)와 최적화(Optimizer)

훈련의 개선

- 손실함수(Loss function)
  - 지도학습(Supervised Learning) 시 알고리즘이 예측한 값과 실제 정답의 차이를 비교하기 위한 함수
  - 예측값과 실제값(레이블)의 차이를 구하는 기준을 의미하는 것으로 머신러닝 모델 학습에서 필수 구성,
  - 손실함수로 인해 모델의 성능이 달라지고 , 머신러닝 모델을 구현 시 손실함수 선택이 중요
- 과적합(overfitting)
  - machine learning에서 학습 데이터를 과하게 학습(overfitting)하는 것을 뜻함
  - 학습 데이터는 실제 데이터의 부분 집합
  - 학습데이터에 대해서는 오차가 감소하지만 실제 데이터에 대해서는 오차가 증가


초록색 선은 과적합된 모델을, 검은색 선은 일반 모델을 나타낸다.



과적합의 방지: Drop-out

- 정규화 Regularization
  - weight를 조정하는데 규제(제약)를 거는 기법
  - Overfitting을 막기위해 사용함(= 과적합을 완화해 일반화 성능을 높여주기 위한 기법)
    - L1정규화: LASSO(라쏘)
    - L2정규화: Lidge(릿지)
    - 일래스틱 정규화: L1과 L2의 조합
- 배치 정규화 Batch Normalization
  - 활성화함수의 활성화값 또는 출력값을 정규화(정규분포로 만든다)하는 작업
    - 학습 속도 상승
    - 초기 파라미터 의존도 약화
    - 모델 성능 향상

이해를 위한 참고 : [딥러닝] Drop-out(드롭아웃)은 무엇이고 왜 사용할까?
[고군분투 머신러닝] Dropout개념, 설명, Dropout과 앙상블, 텐서플로우 Dropout

Google Colab

- Colaboratory(줄여서 'Colab'이라고 함)을 통해 브라우저 내에서 Python 스크립트를 작성하고 실행
  - 구성이 필요하지 않음
  - 무료로 GPU 사용
  - 간편한 공유
- 구글 코랩 주소: https://colab.research.google.com/