



인공지능 TensorFlow (Ch 4)

데이터사이언스 & A.I 정화민 교수



MNIST 숫자인식 2

- 숫자 인식 코딩 2 : Relu function
 - 모델 평가: 예측 정확도 확인 CH 3에서 sigmoid function 83%
 - Relu function 과 Softmax 사용 정확도 향상

```
x_train = x_train.reshape([60000,28*28])
x_test = x_test.reshape([10000,28*28])
```

```
print(x_train.shape, x_test.shape)
```

```
(60000, 784) (10000, 784)
```

```
# 학습모델 만들기 Dense 32 또는 128(레이어가 와이드 해짐) 비교
model = keras.Sequential()
model.add(keras.layers.Dense(128, activation="sigmoid",input_shape=
(28*28,)))
model.add(keras.layers.Dense(128, activation="sigmoid"))
model.add(keras.layers.Dense(128, activation="sigmoid"))
model.add(keras.layers.Dense(10, activation="sigmoid"))
```

```
optimizer = keras.optimizers.SGD(lr=0.1)
model.compile(optimizer=optimizer, loss="categorical_crossentropy",
metrics=["accuracy"])
```

```
model.summary()
```

```
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
dense_7 (Dense)	(None, 128)	100480
dense_8 (Dense)	(None, 128)	16512
dense_9 (Dense)	(None, 128)	16512
dense_10 (Dense)	(None, 10)	1290

```
Total params: 134,794
Trainable params: 134,794
Non-trainable params: 0
```

```
# 모델 학습시킴
model.fit(x_train, y_train, batch_size=32, epochs=10, validation_data=(x_test, y_test))
```

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

```
10000/10000 [=====] - 0s 33us/sample - loss: 0.5365 - accuracy: 0.8322
[0.5364658487319947, 0.8322]
```

```
1 # 학습모델 만들기 Dense 32 또는 128(레이어가 와이드 해짐), relu function + softmax 사용
2 model = keras.Sequential()
3 model.add(keras.layers.Dense(128, activation="relu",input_shape=(28*28,)))
4 model.add(keras.layers.Dense(128, activation="relu"))
5 model.add(keras.layers.Dense(128, activation="relu"))
6 model.add(keras.layers.Dense(128, activation="relu"))
7 model.add(keras.layers.Dense(128, activation="relu"))
8 model.add(keras.layers.Dense(10, activation="softmax"))
```

```
1 optimizer = keras.optimizers.SGD(lr=0.001)
2 model.compile(optimizer=optimizer, loss="categorical_crossentropy", metrics=["accuracy"])
```

```
1 model.summary()
```

```
Model: "sequential_11"
```

Layer (type)	Output Shape	Param #
dense_63 (Dense)	(None, 128)	100480
dense_64 (Dense)	(None, 128)	16512
dense_65 (Dense)	(None, 128)	16512
dense_66 (Dense)	(None, 128)	16512
dense_67 (Dense)	(None, 128)	16512
dense_68 (Dense)	(None, 10)	1290

```
Total params: 167,818
Trainable params: 167,818
Non-trainable params: 0
```

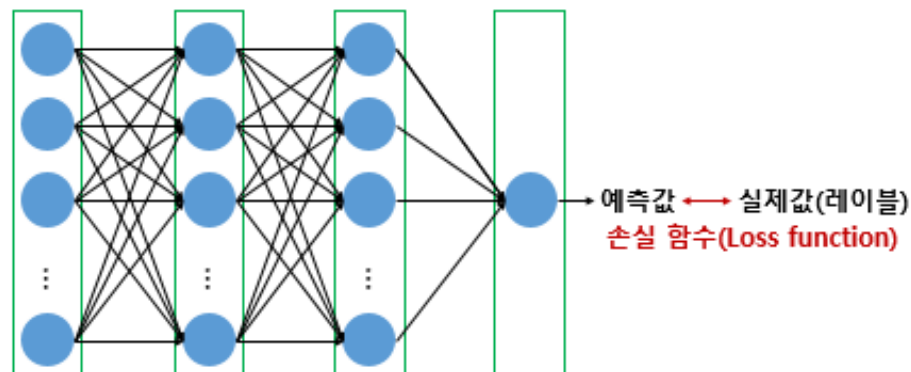
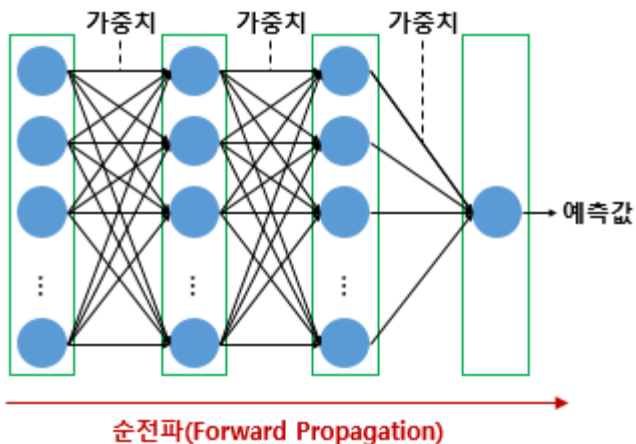
```
1 model.evaluate(x_test, y_test)
```

```
10000/10000 [=====] - 0s 40us/sample - loss: 0.1802 - accuracy: 0.9519
[0.18021341592413373, 0.9519]
```

딥러닝의 학습방법

■ 딥러닝의 학습 방법

- 순전파(Foward Propagation) : 입력층에서 출력층 방향으로 예측값의 연산이 진행되는 과정
(활성화 함수, 은닉층의 수, 각 은닉층의 뉴런 수 등 딥 러닝 모델을 설계하고나면 입력값은 입력층, 은닉층을 지나면서 각 층에서의 가중치와 함께 연산되며 출력층으로 나옴)
- 손실함수(Loss function) : 실제값과 예측값의 차이를 수치화해주는 함수
(손실 함수의 값을 최소화하는 두 개의 매개변수인 가중치 W와 편향 b를 찾아가는 것이 딥 러닝의 학습 과정이므로 손실 함수의 선정은 매우 중요)
- MSE(Mean Squared Error, MSE) : 오차 제곱 평균을 의미. 연속형 변수를 예측할 때 사용
- 크로스 엔트로피(Cross-Entropy): 낮은 확률로 예측해서 맞추거나, 높은 확률로 예측해서 틀리는 경우 loss가 더 큼. 이진 분류 (Binary Classification)의 경우 `binary_crossentropy`를 사용하고, 다중 클래스 분류(Multi-Class Classification)일 경우 `categorical_crossentropy`를 사용.



딥러닝의 학습방법

- 딥러닝의 학습 방법

- 전결합층(Fully-connected layer, FC, Dense layer)

다층 퍼셉트론은 은닉층과 출력층에 있는 모든 뉴런은 바로 이전 층의 모든 뉴런과 연결돼 있음 .
그와 같이 어떤 층의 모든 뉴런이 이전 층의 모든 뉴런과 연결돼 있는 층을 전결합층이라고 하고
FC라고 부르기도 함. 밀집층을 구현할 때 Dense()를 사용

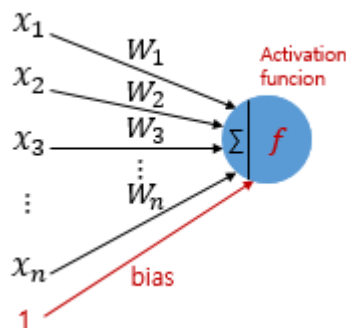
- 활성화 함수(Activation Function)

은닉층과 출력층의 뉴런에서 출력값을 결정하는 함수를 활성화 함수(Activation function)라고 함.

1) 활성화 함수의 특징 - 비선형 함수(Nonlinear function):

활성화 함수의 특징은 선형 함수가 아닌 비선형 함수여야 한다는 점입니다. 선형 함수란 입력의 상수배만큼 변하는 함수를 선형함수라고 합니다. 예를 들어 $f(x)=Wx+b$ 라는 함수가 있을 때, W 와 b 는 상수, 이 식은 그래프를 그리면 직선이 됨 .

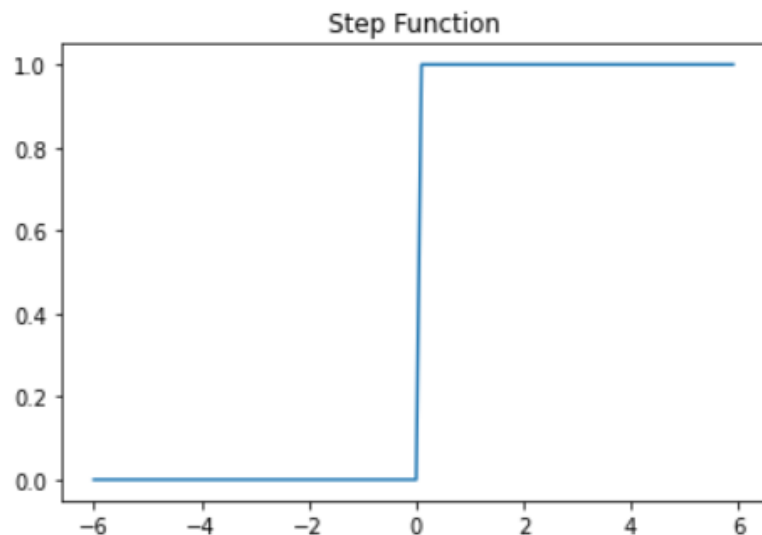
비선형 함수는 직선 1개로는 그릴 수 없는 함수를 말함.



딥러닝의 학습방법

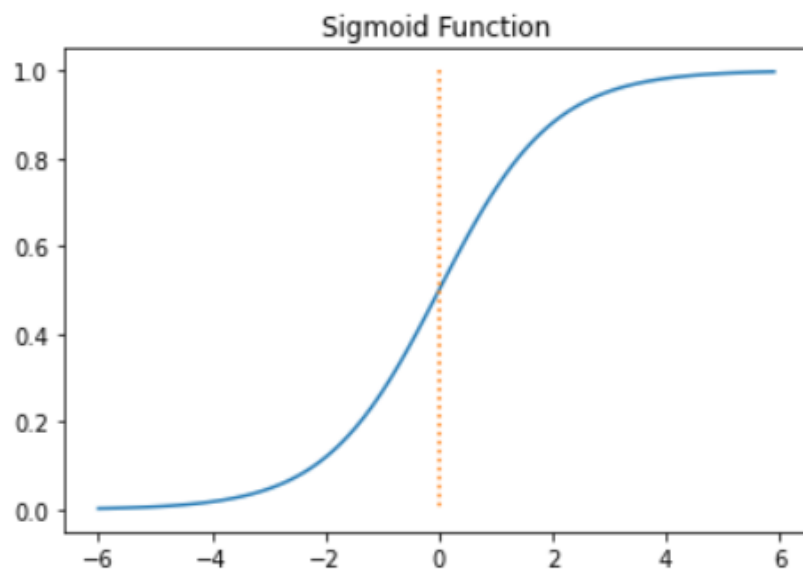
```
1 # 주피터 노트북에서 넘파이와 맷플로립 사용
2 import numpy as np
3 import matplotlib.pyplot as plt
```

```
1 # 2) 계단 함수(Step function): 계단 함수는 이제 거의 사용되지 않지만,
2 # 퍼셉트론을 통해 처음으로 인공 신경망을 배울 때 가장 처음 접하게 되는 활성화 함수.
3
4 def step(x):
5     return np.array(x > 0, dtype=np.int)
6 x = np.arange(-6.0, 6.0, 0.1) # -6.0부터 6.0까지 0.1 간격 생성
7 y = step(x)
8 plt.title('Step Function')
9 plt.plot(x,y)
10 plt.show()
```



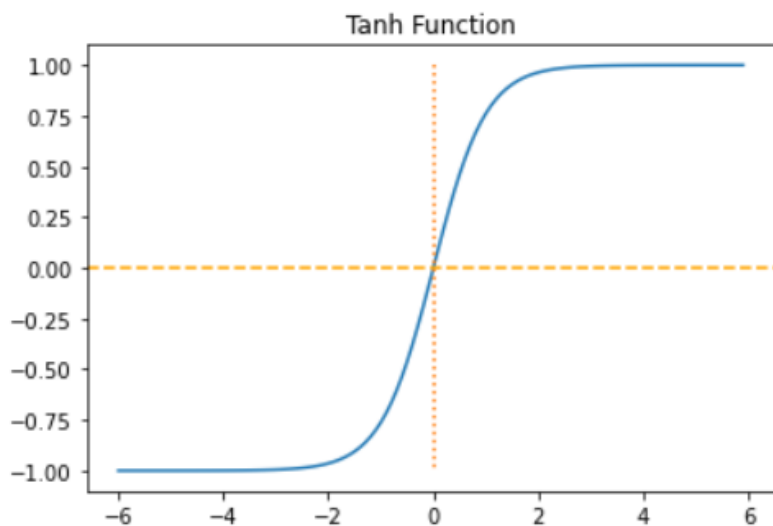
딥러닝의 학습방법

```
1 # 3) 시그모이드 함수(Sigmoid function)
2 #시그모이드 함수를 사용하는 은닉층의 개수가 다수가 될 경우에는 0에 가까운 기울기가
3 #계속 급해지면 앞단에서는 거의 기울기를 전파받을 수 없게 됨.
4 #즉, 매개변수 W가 업데이트 되지 않아 학습이 되지를 앎(기울기소실 Vanishing Gradient문제)
5 def sigmoid(x):
6     return 1/(1+np.exp(-x))
7 x = np.arange(-6.0, 6.0, 0.1)
8 y = sigmoid(x)
9
10 plt.plot(x, y)
11 plt.plot([0,0],[1.0,0.0], ':') # 가운데 점선 추가
12 plt.title('Sigmoid Function')
13 plt.show()
```



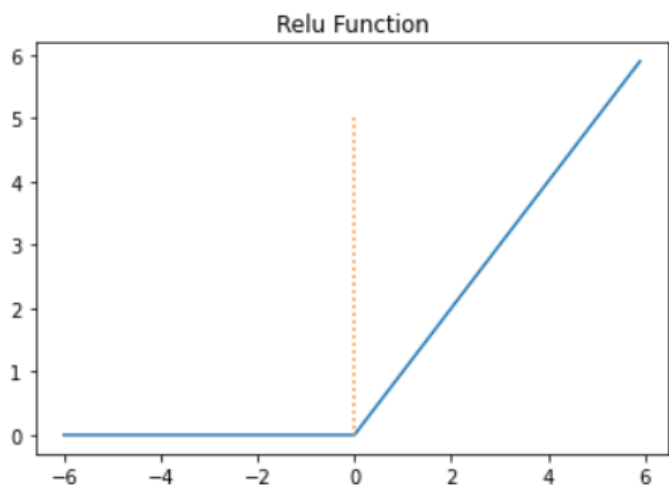
딥러닝의 학습방법

```
1 # 4) 하이퍼볼릭탄젠트 함수(Hyperbolic tangent function)
2 # 하이퍼볼릭탄젠트 함수(tanh)는 입력값을 -1과 1사이의 값으로 변환.
3 # -1과 1에 가까운 출력값을 출력할 때, 시그모이드 함수와 같은 문제가 발생함. 그러나 하이퍼볼릭탄젠트 함수의 경우에는
4 # 시그모이드 함수와는 달리 0을 중심으로 하고 있는데, 이때문에 시그모이드 함수와 비교하면 반환값의 변화폭이 더 큼.
5 # 그래서 시그모이드 함수보다는 기울기 소실 증상이 적은 편임. 그래서 은닉층에서 시그모이드 함수보다는 많이 사용
6 x = np.arange(-6.0, 6.0, 0.1)
7 y = np.tanh(x)
8
9 plt.plot(x, y)
10 plt.plot([0,0],[1.0,-1.0], ':')
11 plt.axhline(y=0, color='orange', linestyle='--')
12 plt.title('Tanh Function')
13 plt.show()
```



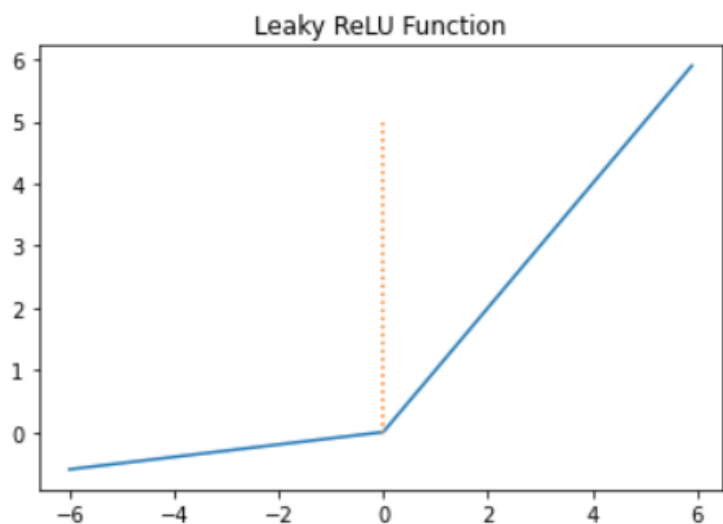
딥러닝의 학습방법

```
1 # 5) 렐루 함수(ReLU)
2 # 인공 신경망에서 가장 최고의 인기를 얻고 있는 함수. 수식은  $f(x)=\max(0,x)$ 로 아주 간단.
3 # 렐루 함수는 음수를 입력하면 0을 출력하고, 양수를 입력하면 입력값을 그대로 반환함.
4 # 렐루 함수는 특정 양수값에 수렴하지 않으므로 깊은 신경망에서 시그모이드 함수보다 훨씬 더 잘 작동함.
5 # 렐루 함수는 시그모이드 함수와 하이퍼볼릭탄젠트 함수와 같이 어떤 연산이 필요한 것이 아니라 단순 임계값이므로 연산 속도도 빠름.
6 # 문제점: 입력값이 음수면 기울기도 0이 됨. 이 뉴런은 다시 회생하는 것이 매우 어려워 죽은 렐루(dying ReLU)라고 함.
7
8 def relu(x):
9     return np.maximum(0, x)
10
11 x = np.arange(-6.0, 6.0, 0.1)
12 y = relu(x)
13
14 plt.plot(x, y)
15 plt.plot([0,0],[5.0,0.0], ':')
16 plt.title('Relu Function')
17 plt.show()
```



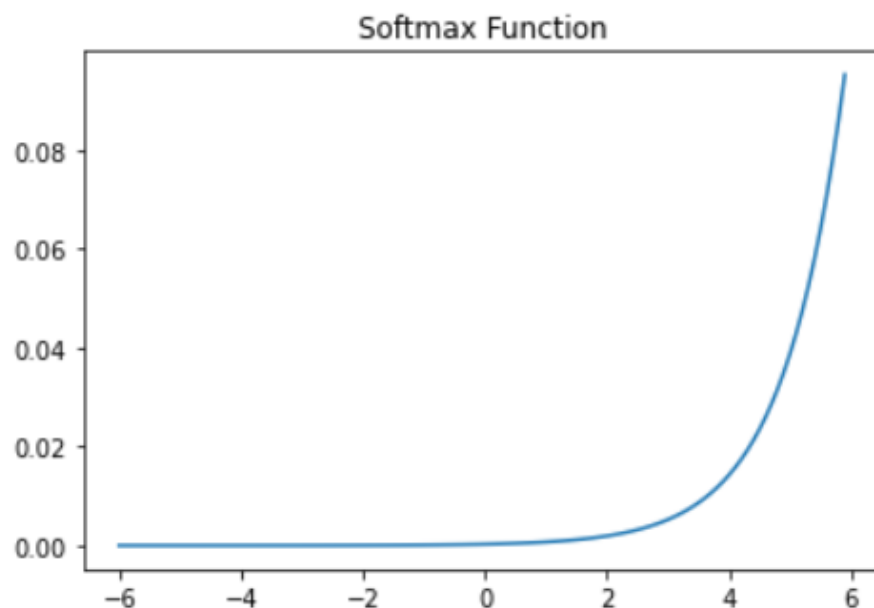
딥러닝의 학습방법

```
1 #6) 리키 렐루(Leaky ReLU)죽은 렐루를 보완하기 위해 ReLU의 변형 함수들이 등장하기 시작함.
2 # Leaky ReLU는 입력값이 음수일 경우에 0이 아니라 0.001과 같은 매우 작은 수를 반환하도록 되어있음.
3 # 수식은  $f(x)=\max(ax,x)$ 로 아주 간단.  $a$ 는 하이퍼파라미터로 Leaky('새는') 정도를 결정하며 일반적으로는 0.01의 값을 가짐.
4 # '새는 정도'라는 것은 입력값의 음수일 때의 기울기를 비유하고 있음.
5
6 a = 0.1
7
8 def leaky_relu(x):
9     return np.maximum(a*x, x)
10
11 x = np.arange(-6.0, 6.0, 0.1)
12 y = leaky_relu(x)
13
14 plt.plot(x, y)
15 plt.plot([0,0],[5.0,0.0], ':')
16 plt.title('Leaky ReLU Function')
17 plt.show()
```



딥러닝의 학습방법

```
1 #7) 소프트맥스 함수(Softmax function)
2 # 은닉층에서 ReLU(또는 ReLU 변형) 함수들을 사용하는 것이 일반적이지만 분류 문제를 로지스틱 회귀와
3 # 소프트맥스 회귀를 출력층에 적용하여 사용함.
4
5 x = np.arange(-6.0, 6.0, 0.1)
6 y = np.exp(x) / np.sum(np.exp(x))
7
8 plt.plot(x, y)
9 plt.title('Softmax Function')
10 plt.show()
```



딥러닝의 학습방법

활성화 함수	원리	활용 사례	베스트 프랙티스
시그모이드	$S(x) = 1 / (1 + e^{-x})$, 입력값을 0과 1 사이의 값으로 압축	이진 분류 문제 (예: 이메일 스팸 분류, 의료 이미징에서 종양 판별)	출력층에서 주로 사용, 중간층에서는 렐루 계열 함수 사용 권장
렐루	$R(x) = \max(0, x)$, 입력이 0을 넘으면 그 입력을 그대로 출력, 0 이하면 0을 출력	이미지 인식, 음성 인식 등의 컨볼루션 신경망(CNN)과 깊은 신경망	렐루는 학습 속도가 빠르지만 죽은 렐루 문제가 있어, 리키 렐루나 파라미터화된 렐루(PReLU) 사용 고려
리키 렐루	$L(x) = \max(\alpha x, x)$, 음수 입력에 대해 작은 기울기 α 를 허용	GAN(Generative Adversarial Networks) 등에서 죽은 렐루 문제 해결	α 값은 실험을 통해 최적의 값을 찾아야 하며, 네트워크에 따라 다른 α 값이 더 나은 결과를 줄 수 있음
탄젠트 하이퍼볼릭	$T(x) = (e^x - e^{-x}) / (e^x + e^{-x})$, 입력값을 -1과 1 사이로 변환	감정 분석, 언어 모델링 등 자연 언어 처리	중간층에서 좋은 성능을 발휘하나, 깊은 네트워크에서는 그라디언트 소실 문제로 인해 렐루 계열 함수 선호

딥러닝의 학습방법

- **옵티마이저(Optimizer)**: 손실 함수의 값을 줄여 나가면서 학습하는 방법에서 어떤 옵티마이저를 사용하느냐에 따라 달라짐.

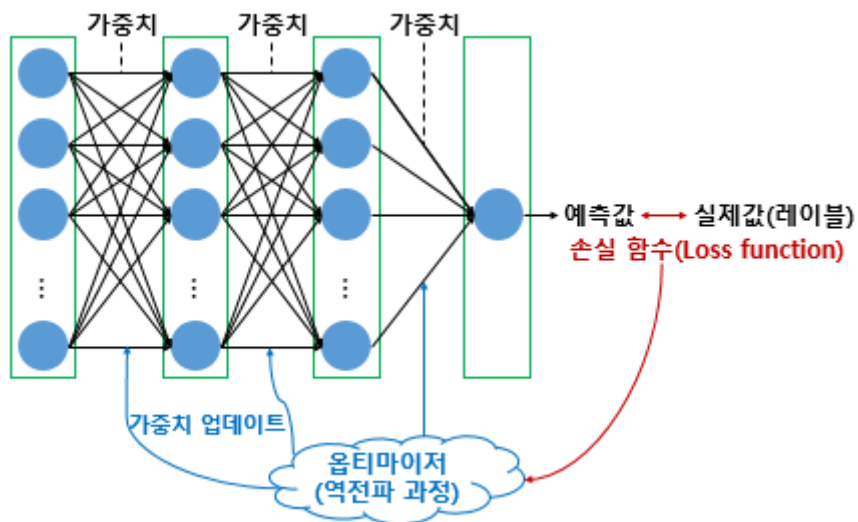
배치(Batch): 가중치 등의 매개 변수의 값을 조정하기 위해 사용하는 데이터의 양을 말하고, 전체 데이터를 가지고 매개 변수의 값을 조정할 수도 있고, 정해진 양의 데이터만 가지고도 매개 변수의 값을 조정할 수 있음.

1) 배치 경사 하강법(Batch Gradient Descent)

가장 기본적인 경사 하강법으로 오차(loss)를 구할 때 전체 데이터를 고려함.

머신 러닝에서는 1번의 훈련 횟수를 1 에포크라고 하는데, 배치 경사 하강법은 한 번의 에포크에 모든 매개 변수 업데이트를 단 한 번 수행함.

배치 경사 하강법은 전체 데이터를 고려해서 학습하므로 에포크당 시간이 오래 걸리며, 메모리를 크게 요구한다는 단점이 있으나 글로벌 미니멈을 찾을 수 있다는 장점이 있음.



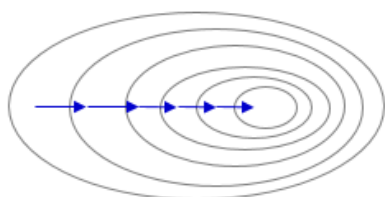
배치 경사하강법(Batch Gradient Descent) 코드 예

```
model.fit(X_train, y_train,  
         batch_size=len(trainX))
```

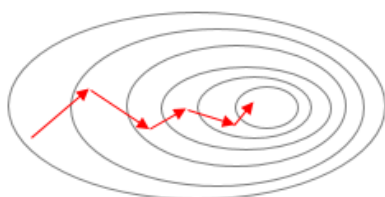
딥러닝의 학습방법

2) 확률적 경사 하강법(Stochastic Gradient Descent, SGD)

- 기존 배치 경사 하강법은 전체 데이터에 대해서 계산을 하다보니 시간이 너무 오래걸린다는 단점이 있는데 반해 확률적 경사 하강법은 매개변수 값을 조정 시 전체 데이터가 아니라 랜덤으로 선택한 하나의 데이터에 대해서만 계산하는 방법임.
- 더 적은 데이터를 사용하므로 더 빠르게 계산할 수 있음.
- 매개변수의 변경폭이 불안정하고, 때로는 배치 경사 하강법보다 정확도가 낮을 수도 있지만 속도만큼은 배치 경사 하강법보다 빠르다는 장점이 있음.



경사 하강법



SGD

[Source : wikidocs.net/](http://wikidocs.net/)

확률적 경사 하강법(Stochastic Gradient Descent, SGD) 코드 예
`model.fit(X_train, y_train, batch_size=1)`

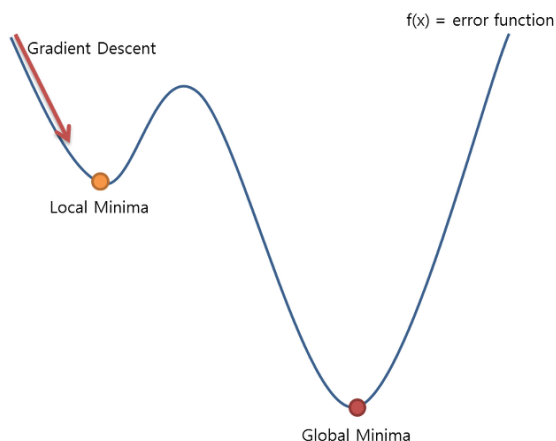
딥러닝의 학습방법

3) 미니 배치 경사 하강법(Mini-Batch Gradient Descent)

- 전체 데이터도 아니고, 1개의 데이터도 아니고 정해진 양에 대해서만 계산하여 매개 변수의 값을 조정하는 경사 하강법을 미니 배치 경사 하강법이라고 함.
- 미니 배치 경사 하강법은 전체 데이터를 계산하는 것보다 빠르며, SGD보다 안정적이라는 장점이 있습니다. 실제로 가장 많이 사용되는 경사 하강법임.

4) 모멘텀(Momentum)

- 모멘텀(Momentum)은 관성이라는 물리학의 법칙을 응용한 방법임.
- 모멘텀 SGD는 경사 하강법에 관성을 더 해줌.
- 모멘텀은 SGD에서 계산된 접선의 기울기에 한 시점(step) 전의 접선의 기울기값을 일정한 비율만큼 반영함. 이렇게 하면 마치 언덕에서 공이 내려올 때, 중간에 작은 웅덩이에 빠지더라도 관성의 힘으로 넘어서는 효과를 줄 수 있음.
- 로컬 미니마에 도달하였을 때, 기울기가 0이라서 기존의 경사 하강법이라면 이를 글로벌 미니마로 잘못 인식하여 계산이 끝났을 상황이라도 모멘텀. 즉, 관성의 힘을 빌리면 값이 조절되면서 로컬 미니멈에서 탈출하는 효과를 얻을 수도 있음.



미니 배치 경사 하강법(Mini-Batch Gradient Descent) 코드 예
`model.fit(X_train, y_train, batch_size=32) #배치크기 32`

모멘텀(Momentum)코드 예
`keras.optimizers.SGD(lr = 0.01, momentum= 0.9)`

딥러닝의 학습방법

5) 아다그라드(Adagrad)

- 매개변수들은 각자 의미하는 바가 다른데, 모든 매개변수에 동일한 학습률(learning rate)을 적용하는 것은 비효율적 임.
- 아다그라드는 각 매개변수에 서로 다른 학습률을 적용시킴
이 때, 변화가 많은 매개변수는 학습률이 작게 설정되고 변화가 적은 매개변수는 학습률을 높게 설정시킴

아다그라드(Adagrad) 코드 예

```
keras.optimizers.Adagrad(lr=0.01, epsilon=1e-6)
```

6) 알엠에스프롭(RMSprop)

- 아다그라드는 학습을 계속 진행한 경우에는, 나중에 가서는 학습률이 지나치게 떨어진다는 단점이 있는데 이를 다른 수식으로 대체하여 이러한 단점을 개선하였음

알엠에스프롭(RMSprop) 코드 예

```
keras.optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=1e-06)
```

7) 아담(Adam)

- 아담은 알엠에스프롭과 모멘텀 두 가지를 합친 듯한 방법으로, 방향과 학습률 두 가지를 모두 잡기 위한 방법

아담(Adam) 코드 예

```
keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)
```

source: <https://keras.io/optimizers/>

딥러닝의 학습방법

■ 에포크, 배치크기, 이터레이션 (Epochs and Batch size and Iteration)

1) 에포크 (Epochs)

- 에포크란 인공 신경망에서 전체 데이터에 대해서 순전파와 역전파가 끝난 상태를 말함.
전체 데이터를 하나의 문제지에 비유한다면 문제지의 모든 문제를 끝까지 다 풀고, 정답지로 채점을 하여 문제지에 대한 공부를 한 번 끝낸 상태를 말함
- 만약 이 에포크 횟수가 지나치거나 너무 적으면 과적합과 과소적합이 발생할 수 있음.

2) 배치 크기(Batch size)

- 배치 크기는 몇 개의 데이터 단위로 매개변수를 업데이트 하는지를 말함.
- 현실에 비유하면 문제지에서 몇 개씩 문제를 풀고나서 정답지를 확인하느냐의 문제임.
- 기계 입장에서는 실제값과 예측값으로부터 오차를 계산하고 옵티마이저가 매개변수를 업데이트하는데 ,
여기서 중요한 포인트는 업데이트가 시작되는 시점이 정답지/실제값을 확인하는 시점임.

ex) 사람이 2,000 문제가 수록되어있는 문제지의 문제를 200개 단위로 풀고 채점한다고 하면 이때 배치크기는 200임. 기계는 배치 크기가 200이면 200개의 샘플 단위로 가중치를 업데이트 함, 단 여기서 주의할 점은 배치 크기와 배치의 수는 다른 개념이라는 점임

- 전체 데이터가 2,000일때 배치 크기를 200으로 준다면 배치의 수는 10입니다.

이는 에포크에서 배치 크기를 나눠준 값($2,000/200 = 10$)이기도 합니다.

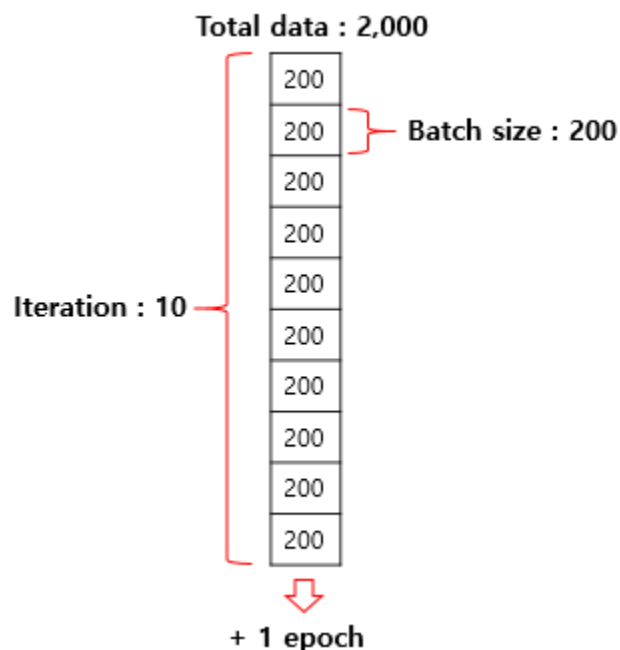
이때 배치의 수를 이터레이션이라고 함.

딥러닝의 학습방법

- 에포크, 배치크기, 이터레이션 (Epochs and Batch size and Iteration)

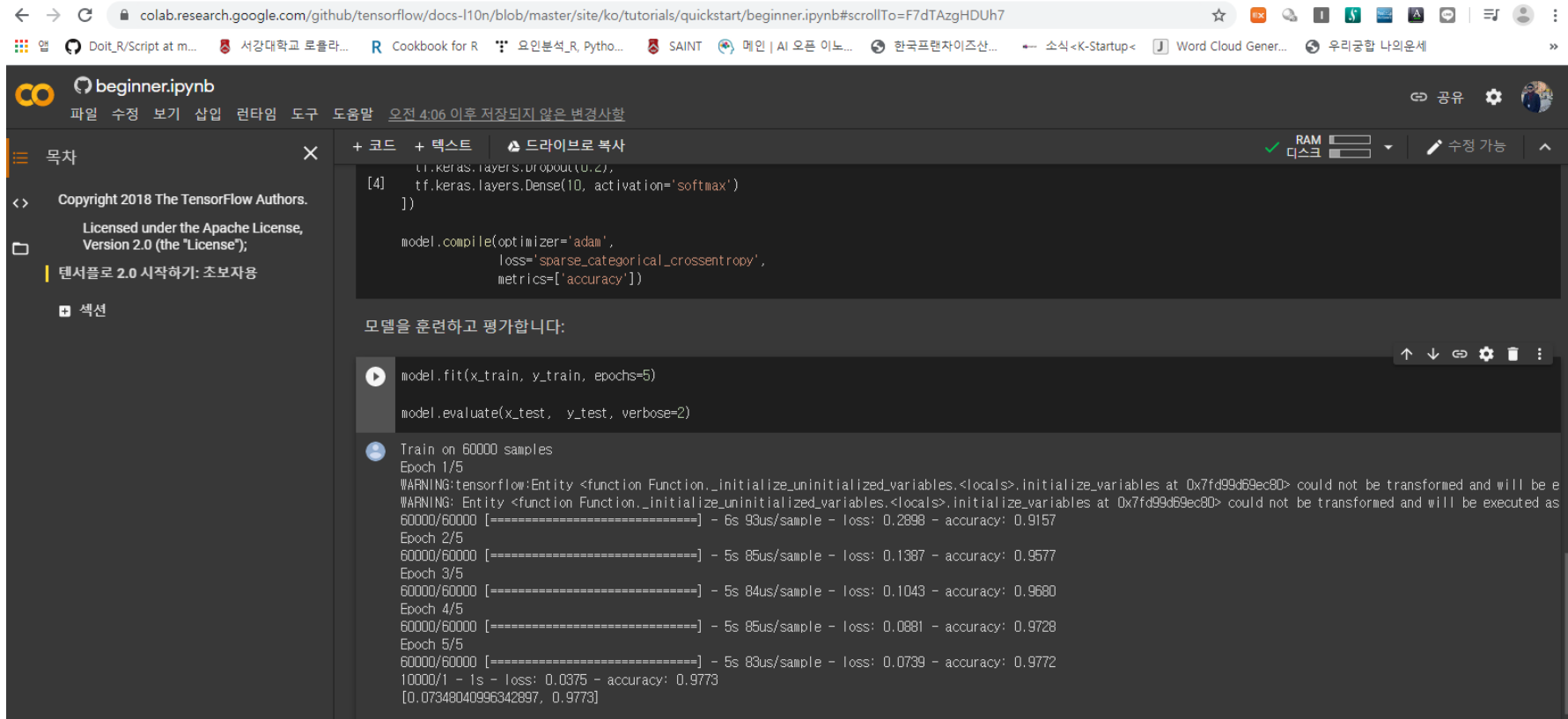
3) 이터레이션(Iteration)

- 이터레이션이란 한 번의 에포크를 끝내기 위해서 필요한 배치의 수를 말함. 또는 한 번의 에포크 내에서 이루어지는 매개변수의 업데이트 횟수이기도 함
- 전체 데이터가 2,000일 때 배치 크기를 200으로 한다면 이터레이션의 수는 총 10개입니다. 이는 한 번의 에포크 당 매개변수 업데이트가 10번 이루어진다는 것을 의미함
- SGD의 경우, SGD는 배치 크기가 1이므로 모든 이터레이션마다 하나의 데이터를 선택하여 경사 하강법을 수행함



MNIST 숫자인식 3

- 숫자 인식 코딩 3 : Tensorflow 공식 문서
- 정확도 97% 이상



The screenshot shows a Jupyter Notebook interface with the following content:

```
[4]: tf.keras.layers.Dropout(0.2),
      tf.keras.layers.Dense(10, activation='softmax')
      )

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

모델을 훈련하고 평가합니다:

```
model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test, verbose=2)
```

Train on 60000 samples

```
Epoch 1/5
WARNING:tensorflow:Entity <function Function._initialize_uninitialized_variables.<locals>.initialize_variables at 0x7fd99d69ec80> could not be transformed and will be e
WARNING: Entity <function Function._initialize_uninitialized_variables.<locals>.initialize_variables at 0x7fd99d69ec80> could not be transformed and will be executed as
60000/60000 [=====] - 6s 90us/sample - loss: 0.2898 - accuracy: 0.9157
Epoch 2/5
60000/60000 [=====] - 5s 85us/sample - loss: 0.1387 - accuracy: 0.9577
Epoch 3/5
60000/60000 [=====] - 5s 84us/sample - loss: 0.1043 - accuracy: 0.9680
Epoch 4/5
60000/60000 [=====] - 5s 85us/sample - loss: 0.0881 - accuracy: 0.9728
Epoch 5/5
60000/60000 [=====] - 5s 83us/sample - loss: 0.0739 - accuracy: 0.9772
10000/1 - 1s - loss: 0.0375 - accuracy: 0.9773
[0.07348040996342897, 0.9773]
```

Source <https://www.tensorflow.org/tutorials/quickstart/beginner>