

딥러닝 기본 예제

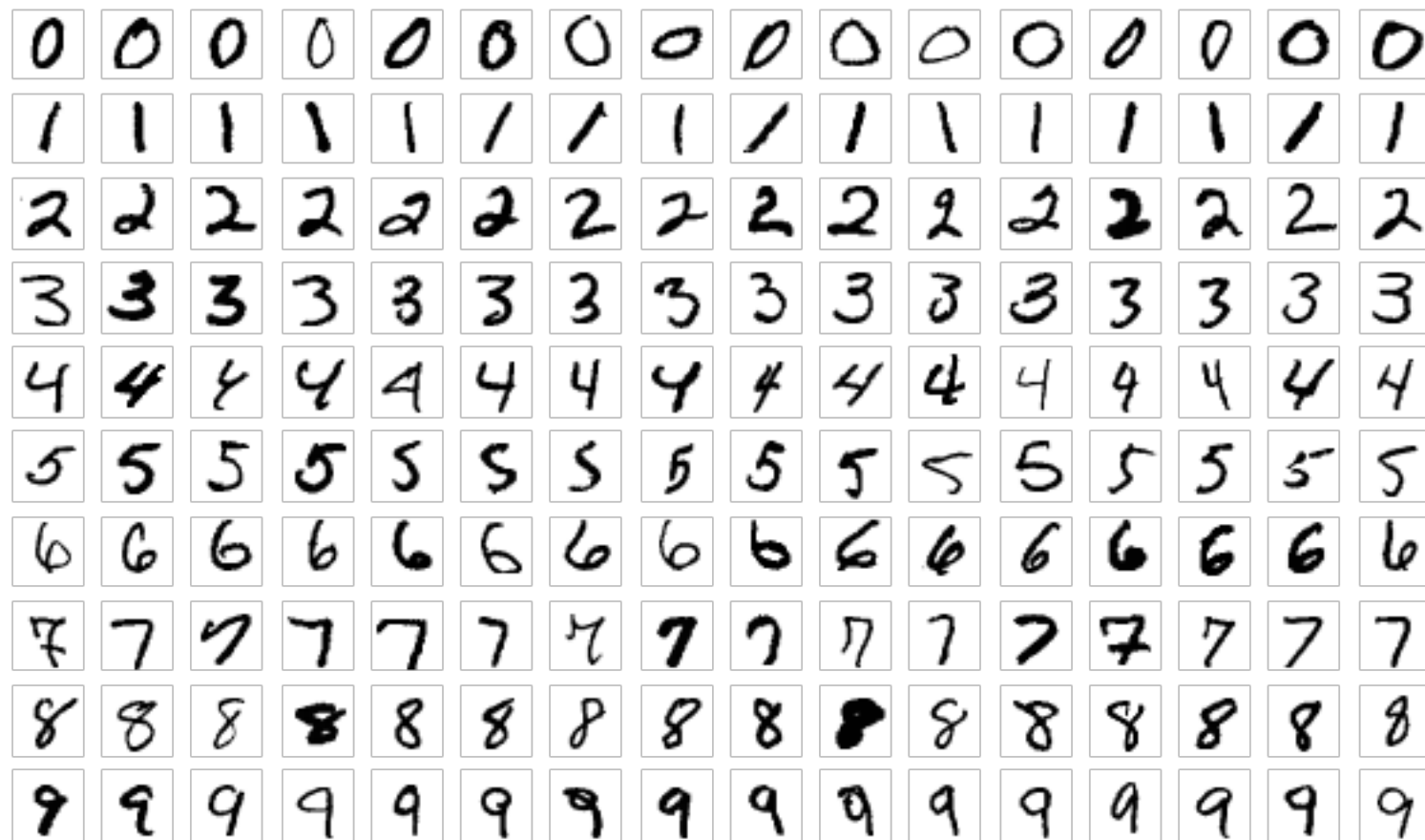
딥러닝 Hello World~

개요

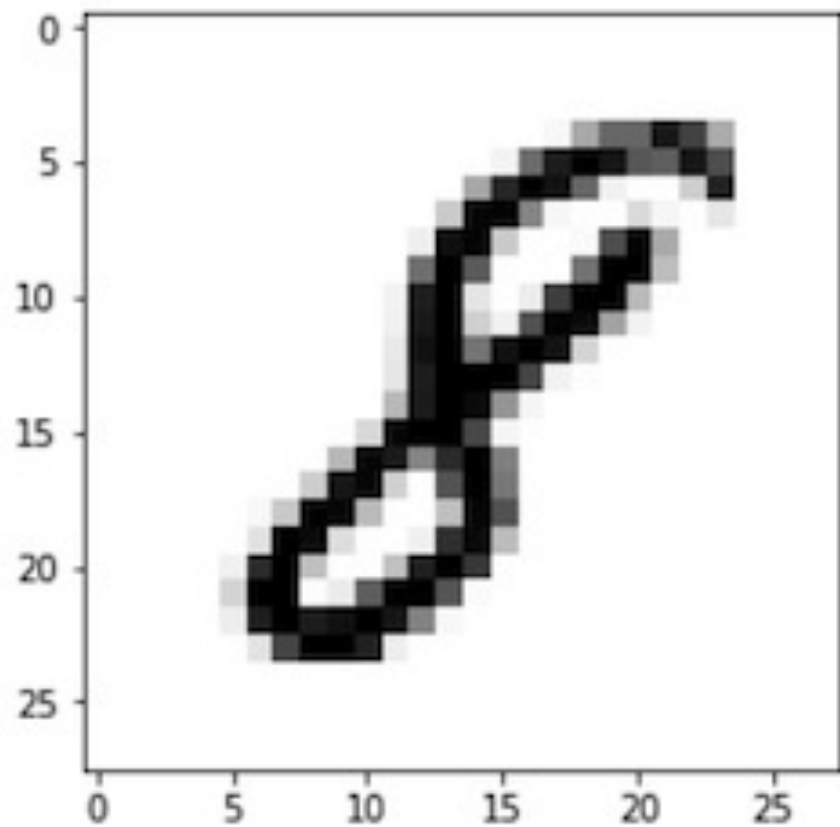
- MNIST데이터를 다루는 간단한 딥러닝 프로그램을 살펴보고, 딥러닝 프로그램에 대한 전체적인 개념을 습득한다.
- 참고 : <https://github.com/rickiepark/deep-learning-with-python-notebooks>
- (업데이트) : <https://github.com/rickiepark/deep-learning-with-python-2nd>

1. 개요 : MNIST

- MNIST(Modified National Institute of Standards and Technology database)의 약자로 사람이 쓴 손글씨와 관련된 데이터셋



1. 개요 : MNIST



- 28 x 28 픽셀 이미지
- 0에서 9까지 10개의 숫자
- 숫자 손글씨 이미지를 입력으로 받아 숫자를 인식하는 문제
- 딥러닝의 Hello World



rickiepark tf2 브랜치 소개 추가

c787090 6 days ago ⌚ 66 commits

📁	datasets	케라스 2.2.4 버전에서 재실행	2 years ago
📄	.gitignore	8장 노트북 번역	2 years ago
📄	2.1-a-first-look-at-a-neural-...	케라스 2.2.4 버전에서 재실행	2 years ago
📄	3.4-classifying-movie-review...	케라스 2.2.4 버전에서 재실행	2 years ago
📄	3.5-classifying-newswires.ip...	케라스 2.2.4 버전에서 재실행	2 years ago
📄	3.6-predicting-house-prices....	케라스 2.2.4 버전에서 재실행	2 years ago
📄	4.4-overfitting-and-underfitt...	케라스 2.2.4 버전에서 재실행	2 years ago
📄	5.1-introduction-to-convnet...	케라스 2.2.4 버전에서 재실행	2 years ago
📄	5.2-using-convnets-with-sm...	케라스 2.2.4 버전에서 재실행	2 years ago
📄	5.3-using-a-pretrained-conv...	케라스 2.2.4 버전에서 재실행	2 years ago
📄	5.4-visualizing-what-convne...	케라스 2.2.4 버전에서 재실행	2 years ago
📄	6.1-one-hot-encoding-of-w...	케라스 2.2.4 버전에서 재실행	2 years ago
📄	6.1-using-word-embedding...	저장소에 GloVe 데이터 포함되었다는 설명 삭제	2 years ago

2. 데이터

```
#from keras.datasets import mnist
from tensorflow.keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

- train_images와 train_labels : 모델이 학습할 훈련 셋 (training set)
- test_images와 test_labels : 모델을 검증하기 위한 테스트 셋 (test set)

2. 데이터

```
train_images.shape
```

```
(60000, 28, 28)
```

```
len(train_labels)
```

```
60000
```

```
train_labels
```

```
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

학습데이터의 개수 : **60,000개**
각 데이터의 이미지 사이즈 : **28 x 28**

label (레이블, 여기서는 정답지)
학습 데이터의 각각의 정답을 가지고 있음

label은 0에서 9까지의 정수로 구성

- 각 데이터 셋의 형태를 **shape**을 통해서 확인
- 데이터 셋에 몇개의 아이템이 있는지 **len**을 통해서 확인

2. 데이터

```
test_images.shape
```

```
(10000, 28, 28)
```

```
len(test_labels)
```

```
10000
```

```
test_labels
```

```
array([7, 2, 1, ..., 4, 5, 6], dtype=uint8)
```

테스트 데이터의 개수 : **10,000개**

각 데이터의 이미지 사이즈 : **28 x 28**

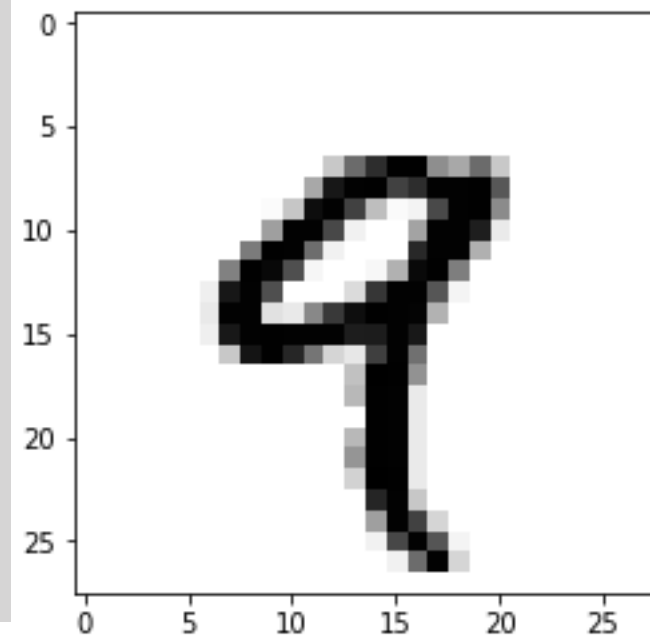
- 테스트 데이터셋은 학습에 사용되지 않으며, 검증용으로 사용
- 데이터의 모양은 학습 데이터와 동일

2. 데이터

```
import matplotlib.pyplot as plt
```

```
digit = train_images[4]
```

```
plt.imshow(digit, cmap=plt.cm.binary)  
plt.show()
```



- matplotlib (MATLAB like한 가시화(Plot) 라이브러리)
- 파이썬에서 그래프를 그리거나 가시화 할 때 가장 범용적으로 사용됨

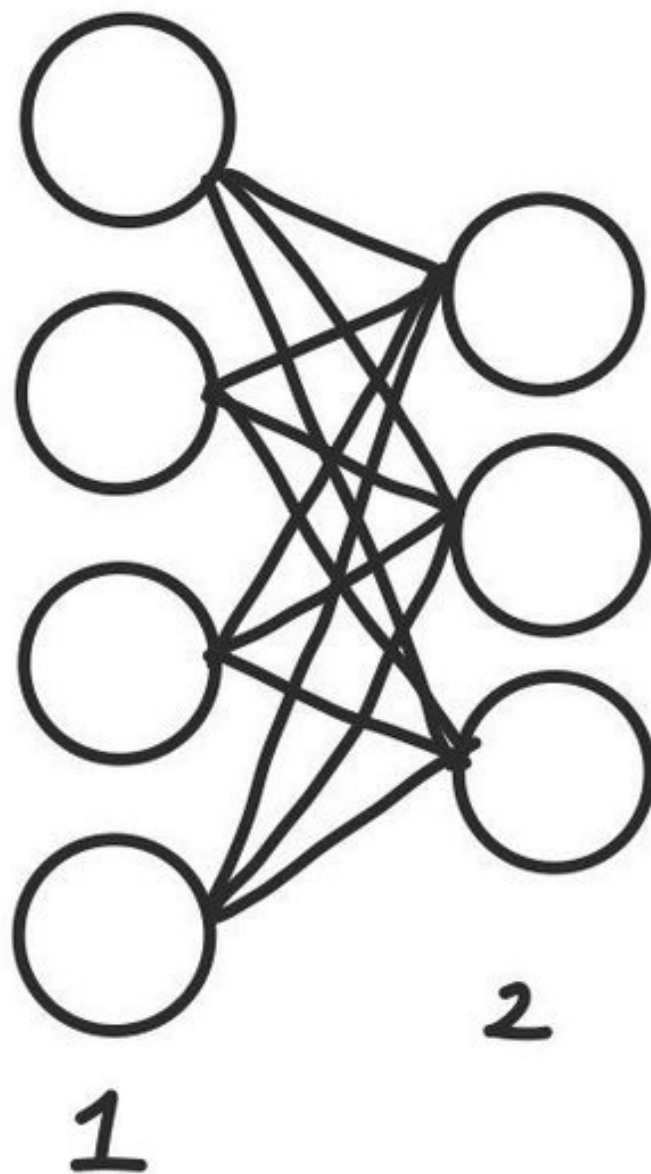
3. 모델

```
#from keras import models
#from keras import layers
#from tensorflow.keras import models
from tensorflow import keras
from tensorflow.keras import layers




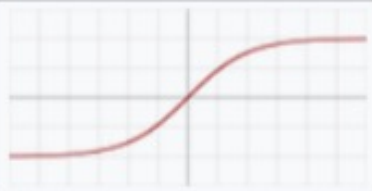
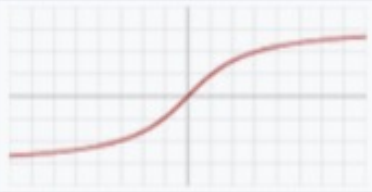
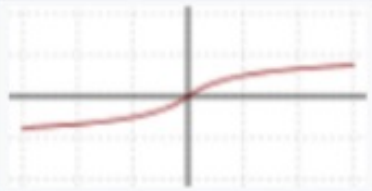

network = keras.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
```

- keras에 모델을 정의하는 방법 중 하나
- Sequential 모델은 모델의 각 layer가 순차적으로 연결된 형태를 지닌 가장 간단한 모델
- Dense 레이어는 모든 입력과 모든 출력이 연결 (총 weight: 입력 x 출력)
- Activation은 출력값에 적용하는 함수로, 일반적으로 sigmoid, relu, softmax 등이 사용됨

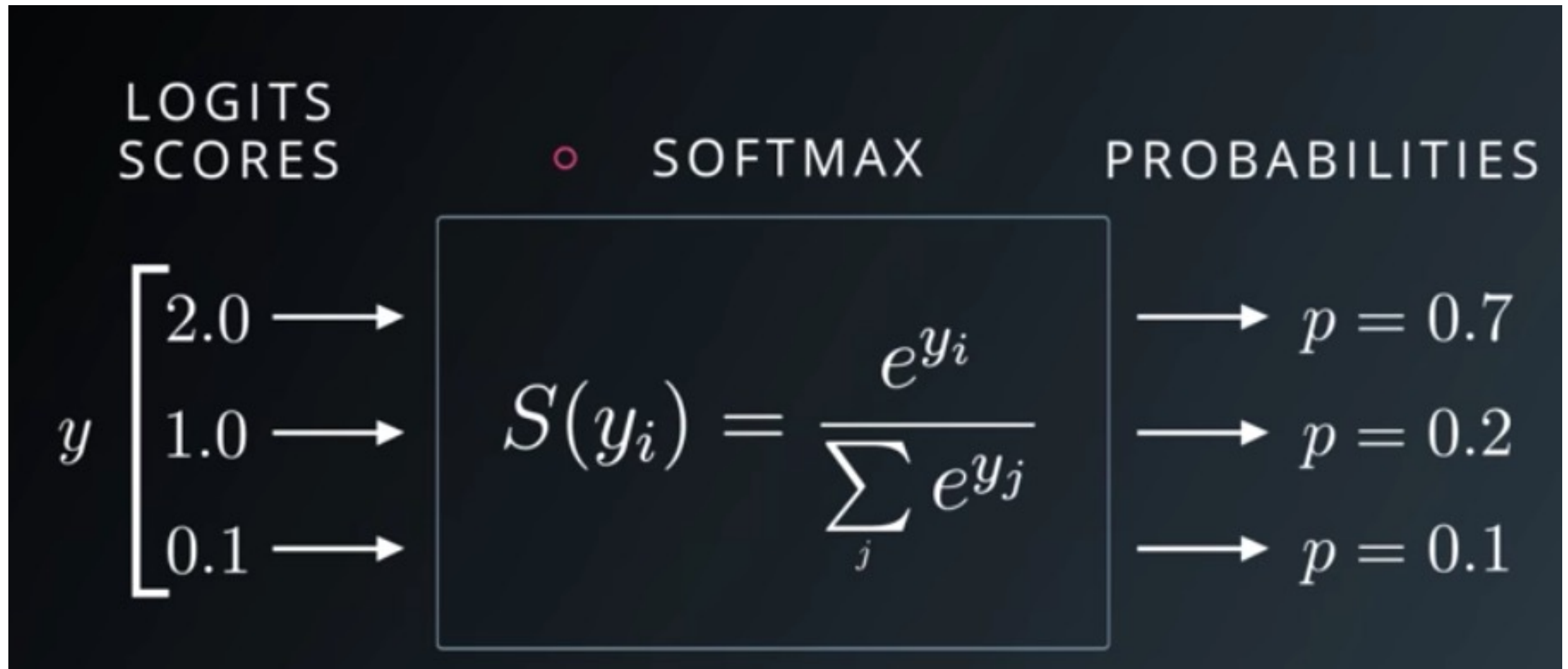
3. 모델 : dense layer



3.1 activation function

Identity		$f(x) = x$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$
ArcTan		$f(x) = \tan^{-1}(x)$
Softsign ^{[7][8]}		$f(x) = \frac{x}{1 + x }$
Rectified linear unit (ReLU) ^[9]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$

3.1 act. func. : softmax



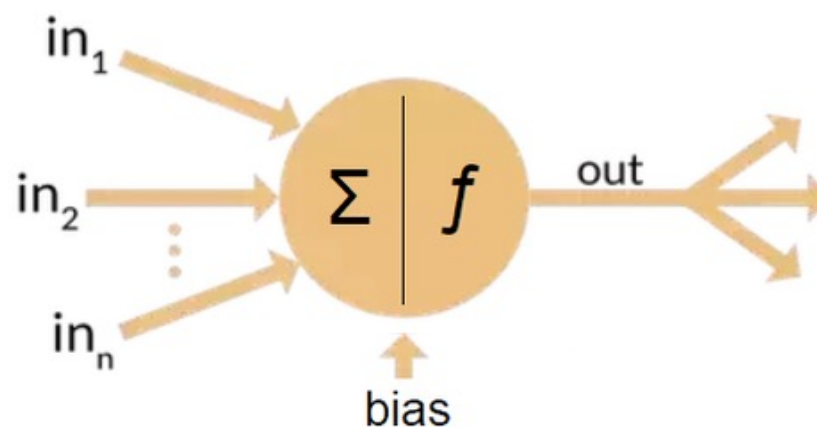
3.1 softmax : logit

In Math, [Logit](#) is a function that maps probabilities ($[0, 1]$) to \mathbb{R} ($(-\infty, \infty)$)

$$L = \ln \frac{p}{1-p} \qquad p = \frac{1}{1+e^{-L}}$$

Probability of 0.5 corresponds to a logit of 0. Negative logit correspond to probabilities less than 0.5, positive to > 0.5 .

Logits also [sometimes](#) refer to the element-wise inverse of the sigmoid function.

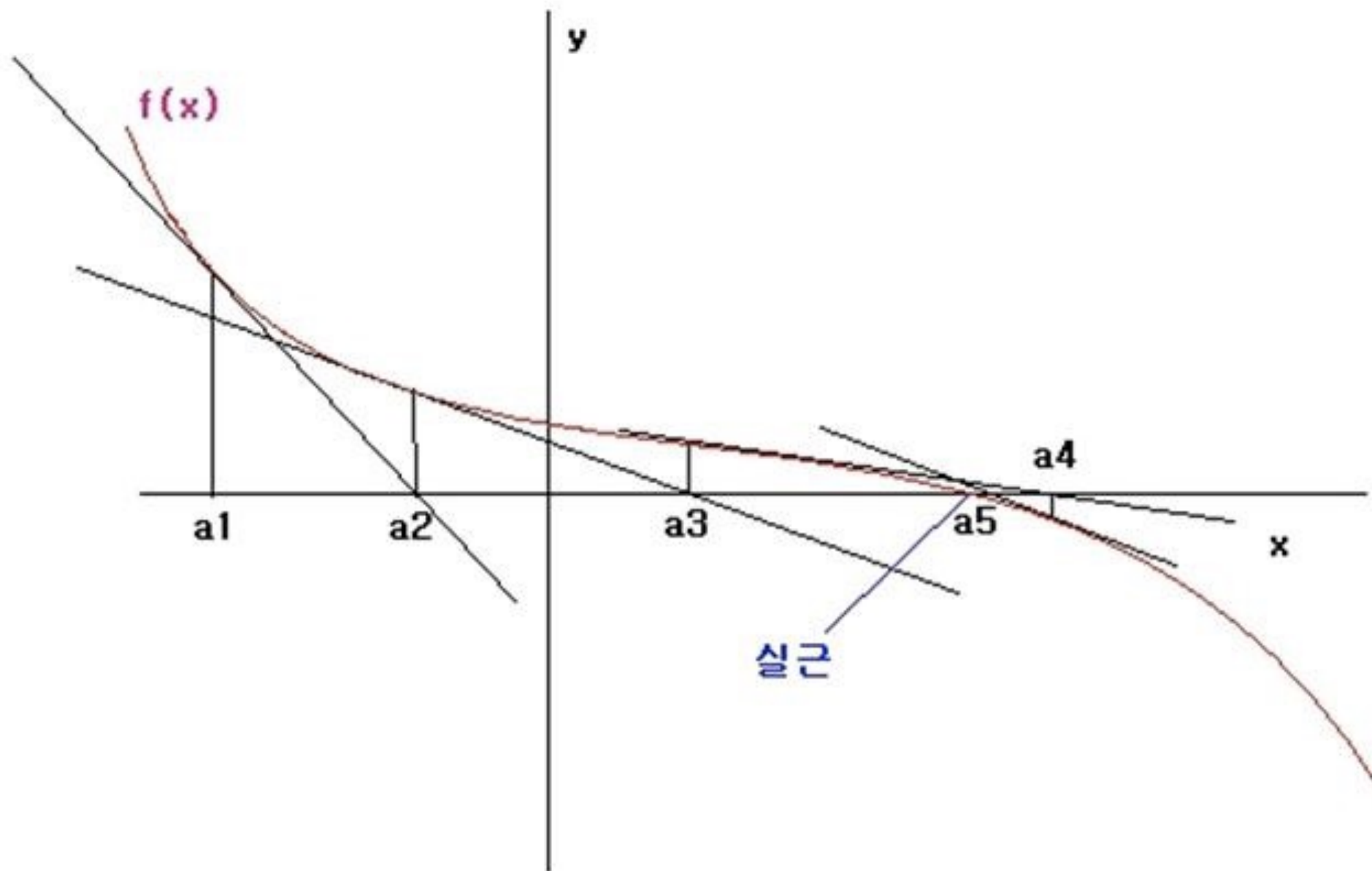


4. 최적화

```
network.compile(optimizer='rmsprop',  
                loss='categorical_crossentropy',  
                metrics=['accuracy'])
```

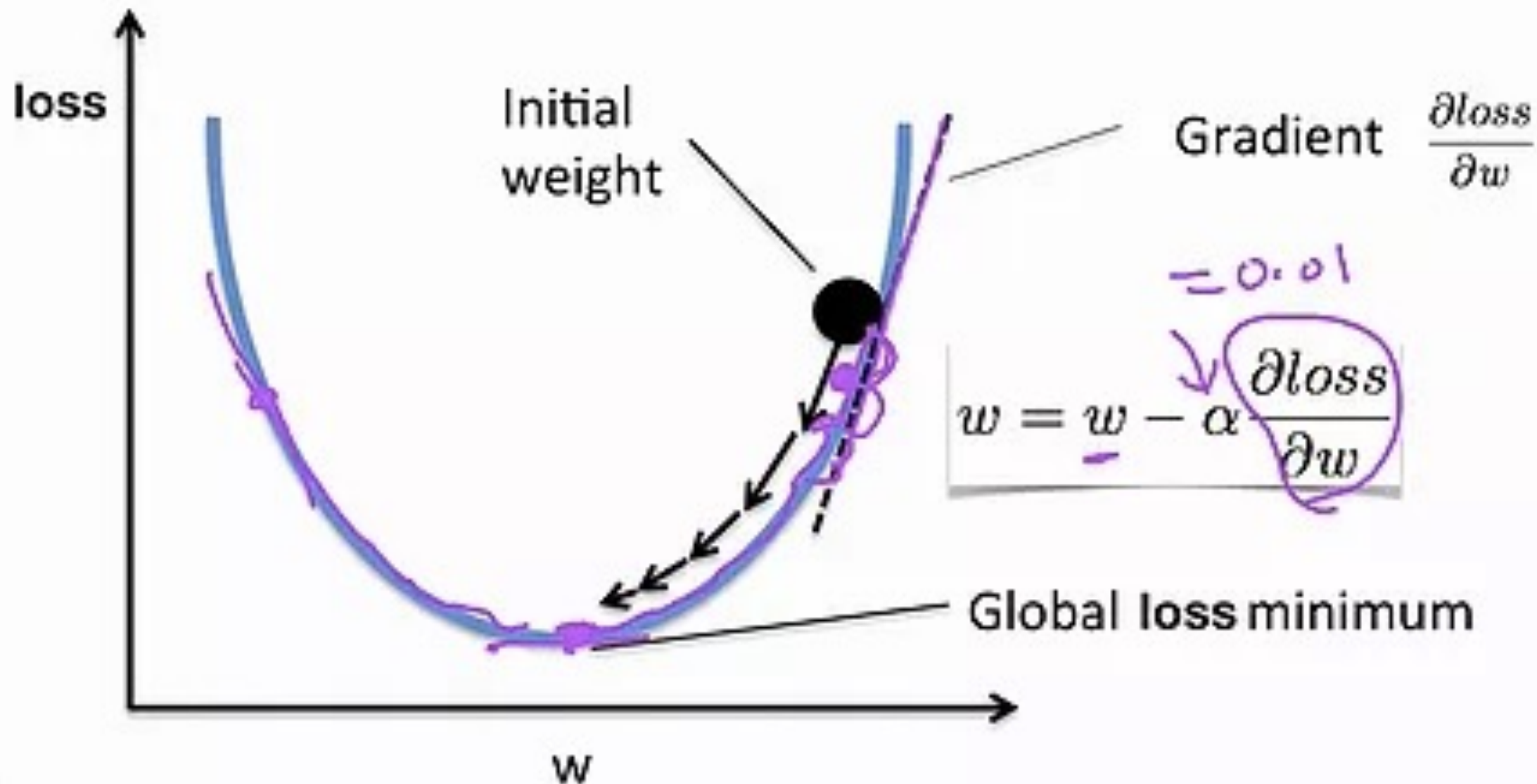
- 학습 : 데이터를 기반으로 모델의 weight 를 찾아나가는 과정
- 최적화(optimizer) 함수 : 최적화 함수 (수학하는 사람들이 이미 만들어 놓음)
- 예) 컴퓨터에서 방정식의 근을 구할 경우, Newton의 방정식을 사용 하기도 함
- 손실(loss) 함수 : 최적화에 사용되는 함수로, 문제 성격에 따라 손실을 정의 (binary, category, regression)
- 기준(metric) : 모델 성능을 측정 지표(예: 손실함수 그 자체, accuracy(정확도) ..)

4. 최적화 : Newton's method

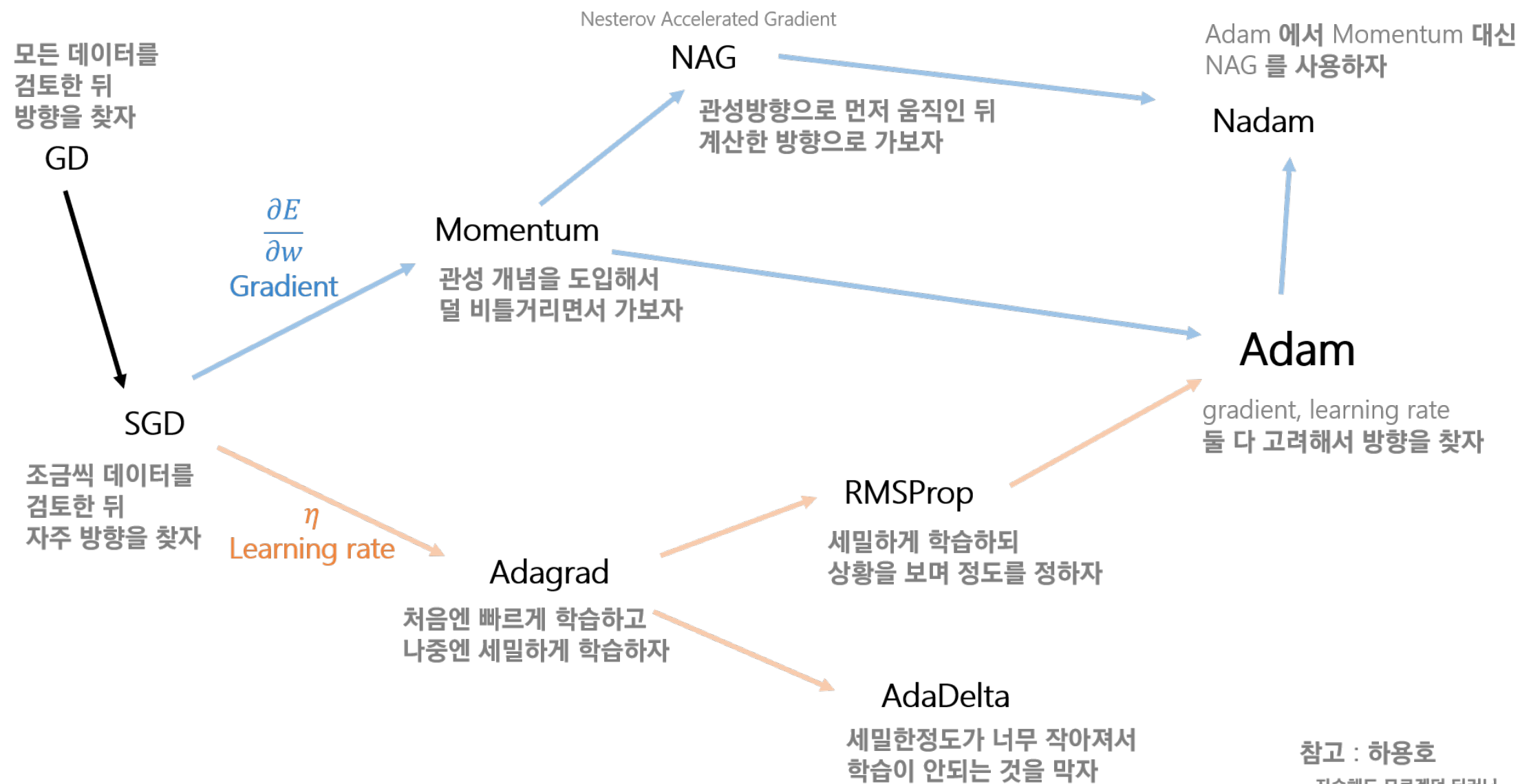


4. 최적화 : Gradient Descent

Gradient descent algorithm



4. 최적화 : optimizer의 종류



참고 : 하용호

- 자습해도 모르겠던 딥러닝, 머리속에 인스를 시켜드립니다.

4. 최적화 : 손실(loss) 함수

Value

이게 얼마가 될거같니?

**output을
그냥 받는다.**

MSE: Mean Squared Error

O/X

기냐? 아니냐?

**output에
sigmoid를 먹인다.**

Binary CrossEntropy

Category

종류중에 요건 뭐냐?

**output에
softmax를 먹인다.**

Categorical CrossEntropy

- Loss함수는 미분 가능
- 해당 함수를 기준으로 네트워크를 최적화
- (추후 자세히 살펴볼 예정)

4. 최적화 : 측정(metric)

```
network.compile(optimizer='rmsprop',  
                loss='categorical_crossentropy',  
                metrics=['accuracy'])
```

- 원하는 지표로 모델 성능을 측정하고 싶을 때 사용
- Loss 함수로 학습이 이루어지는데, 굳이 다른 성능 지표가 필요한가?
 - 미분 불가능한 성능지표인 경우
(예: 정확도(accuracy))
 - loss 함수가 여러 개의 성능지표의 합일 때, 각각의 성능지표가 어떻게 변하는지 보고 싶을때
(loss함수 = loss + regularization)

5. 데이터 전처리

```
train_images = train_images.reshape((60000, 28 * 28))  
train_images = train_images.astype('float32') / 255  
  
test_images = test_images.reshape((10000, 28 * 28))  
test_images = test_images.astype('float32') / 255
```

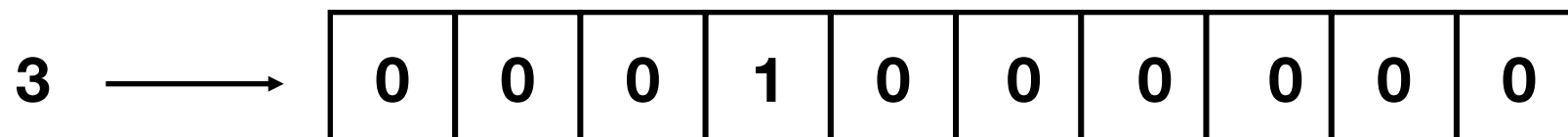
- 데이터를 학습모델 입력에 적합하도록 변경하는 작업
- 예제 설명 :
 - (28, 28) 2차원 이미지를 (784,) 1차원 벡터로 변환
 - 이미지의 intensity(0~255)를 0과 1사이의 값으로 정규화

5. 데이터 전처리

```
# from keras.utils import to_categorical
from tensorflow.keras.utils import to_categorical

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

- 결과값은 0~9까지의 숫자로 이루어져 있음
- 모델의 출력은 총 10개 (0~9)로 각 숫자에 대한 확률로 출력
- 따라서, 각 숫자를 dimension 10을 가진 벡터 확률로 변환 필요
- to_categorical : 출력 값을 one-hot encoding 형태로 변환



6. 학습 (training)

```
In [15]: network.fit(train_images, train_labels, epochs=5, batch_size=128)

Epoch 1/5
60000/60000 [=====] - 1s 22us/step - loss: 0.2534 - acc: 0.9264
Epoch 2/5
60000/60000 [=====] - 1s 13us/step - loss: 0.1015 - acc: 0.9700
Epoch 3/5
60000/60000 [=====] - 1s 12us/step - loss: 0.0670 - acc: 0.9800
Epoch 4/5
60000/60000 [=====] - 1s 12us/step - loss: 0.0490 - acc: 0.9855
Epoch 5/5
60000/60000 [=====] - 1s 12us/step - loss: 0.0372 - acc: 0.9891
Out[15]: <keras.callbacks.History at 0x7f2a4e8ceb00>
```

- fit : 데이터를 모델에 맞춤(fit)
- epoch (에폭) : 전체 데이터 셋을 학습하는 횟수 (60,000개)
- batch_size : 한번 학습때 사용되는 데이터의 개수(128개, 1epoch = 469번의 배치)

7. 평가 (evaluation)

```
In [16]: test_loss, test_acc = network.evaluate(test_images, test_labels)
          10000/10000 [=====] - 0s 16us/step

In [17]: print('test_acc:', test_acc)
          test_acc: 0.9797
```

- `evaluate` 함수에 Test 데이터(학습에 사용되지 않은 데이터)를 이용, 성능평가
- 일반적으로 학습 성능 보다는 낮아짐


```
[ ] import keras  
    keras.__version__
```

Using TensorFlow backend.
'2.2.4'

▼ 신경망과의 첫 만남

이 노트북은 [케라스 창시자에게 배우는 딥러닝](#) 책의 2장 1절의 코드 예제입니다. 책에는 더 많은 내용과 그림이 있습니다. 이 노트북에는 관련된 설명만 포함합니다. 이 노트북의 설명은 케라스 버전 2.2.2에 맞추어져 있습니다. 케라스 최신 버전이 릴리스되면 스트하기 때문에 설명과 코드의 결과가 조금 다를 수 있습니다.

케라스 파이썬 라이브러리를 사용하여 손글씨 숫자 분류를 학습하는 구체적인 신경망 예제를 살펴보겠습니다. 케라스나 비슷 사용한 경험이 없다면 당장은 이 첫 번째 예제를 모두 이해하지 못할 것입니다. 아직 케라스를 설치하지 않았을지도 모릅니다. 장에서 이 예제를 하나하나 자세히 설명합니다. 코드가 좀 이상하거나 요술처럼 보이더라도 너무 걱정하지 마세요. 일단 시작!

여기에서 풀려고 하는 문제는 흑백 손글씨 숫자 이미지(28x28 픽셀)를 10개의 범주(0에서 9까지)로 분류하는 것입니다. 머신 고전으로 취급받는 데이터셋인 MNIST를 사용하겠습니다. 이 데이터셋은 머신 러닝의 역사만큼 오래되었고 많은 연구에 사용 이터셋은 1980년대에 미국 국립표준기술연구소에서 수집한 6만 개의 훈련 이미지와 1만 개의 테스트 이미지로 구성되어 있습 를 알고리즘이 제대로 작동하는지 확인하기 위한 딥러닝계의 'Hello World'라고 생각해도 됩니다. 머신 러닝 기술자가 되기까지 블로그 포스트 등에서 MNIST를 보고 또 보게 될 것입니다.