

MNIST 필기체 구현

딥러닝 / 머신러닝

임 경 태

Week	Chapter	Contents
1	1, 2장	강의 소개, 파이썬 복습
2	1, 3장	파이썬 복습, Numpy, Pandas
3	1, 4장	딥러닝을 위한 미분
4	5장	회귀
5	5장	분류
6	6장	XOR문제
7	7장	딥러닝
8	1~7장	중간고사
9	8장	MNIST 필기체 구현 및 팀 프로젝트 소개
10	9장	오차역전파
11	11장	Jetbot 자율주행 (Collision Avoidance, Transfer Learning)
12	12장	특강 (인공지능 활용 연구) + 팀프로젝트 자율 실습
13	10장	Jetbot 자율주행 (Road Following)
14	11장	합성곱 신경망(CNN), 순환 신경망(RNN) + 팀프로젝트 자율 실습
15	8~12장	기말고사 (or 프로젝트 발표)

CONTENTS

- 1 MNIST 소개**
- 2 MNIST 분류 딥러닝 모델**
- 3 CODE**



목적 : 딥러닝 튜토리얼인 MNIST에 대한 이해 및 모델 구현



목표 : MNIST에 대해 이해하고 이를 딥러닝 모델로 구현한다.



내용 : MNIST구조, 다른 데이터셋 소개, 딥러닝 모델 구현

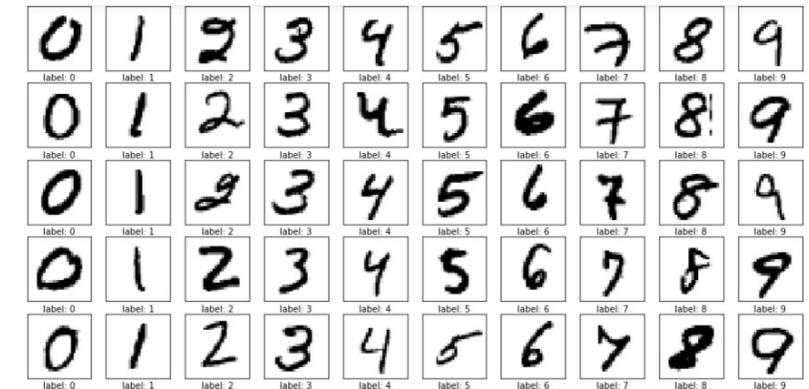
CONTENTS

- 1 MNIST 소개**
- 2 MNIST 분류 딥러닝 모델**
- 3 CODE**

MNIST

MNIST란?

- 0~9의 숫자의 손 글씨 데이터 집합
- 데이터 한 개당 785개의 숫자
→ 정답을 나타내는 숫자 1개, 28x28의 이미지를 나타내는 숫자 784개
- Train Data 60000개, Test Data 10000개



MNIST dataset 다운로드 링크

http://pjreddie.com/media/files/mnist_train.csv

http://pjreddie.com/media/files/mnist_test.csv

MNIST

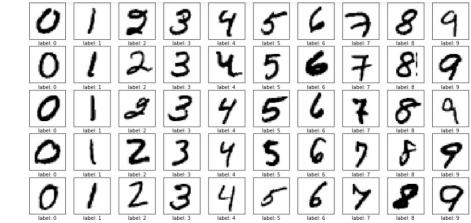
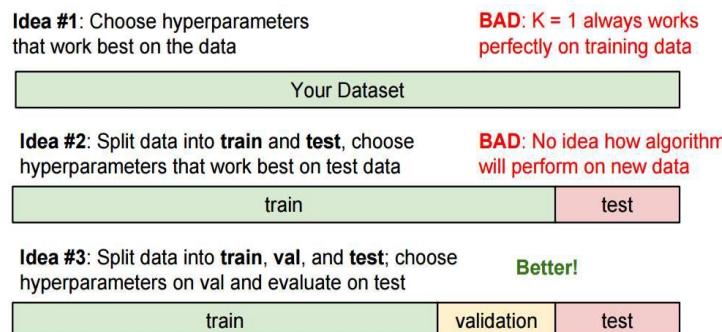
Train/Test data??

- Train Data 60000개, Test Data 10000개
 - 학습 데이터: 우리가 선택한 알고리즘을 위해 학습에 활용할 데이터. (수능 문제집)
 - 검증 데이터: 모델의 예측/분류 정확도를 계산할 수 있다. (수능 모의고사)

사실 모든 검증 세트에 대한 실제 레이블, 즉 정답을 알고 있지만 그렇지 않은 척 하는 셈이다

- 평가 데이터: 검증 데이터 와 비슷하지만, 모델을 구축하거나 튜닝할 때 포함된 적 없다는

점에서 차이가 있다 (수능)



MNIST

📝 MNIST란?

Train Dataset – 60000개 Data

```
1 df = pd.read_csv('./mnist_train (2).csv', names = a)
```

```
2 df
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
...
59995	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
59996	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
59997	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
59998	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
59999	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

60000 rows x 785 columns

Test Dataset - 10000개 Data

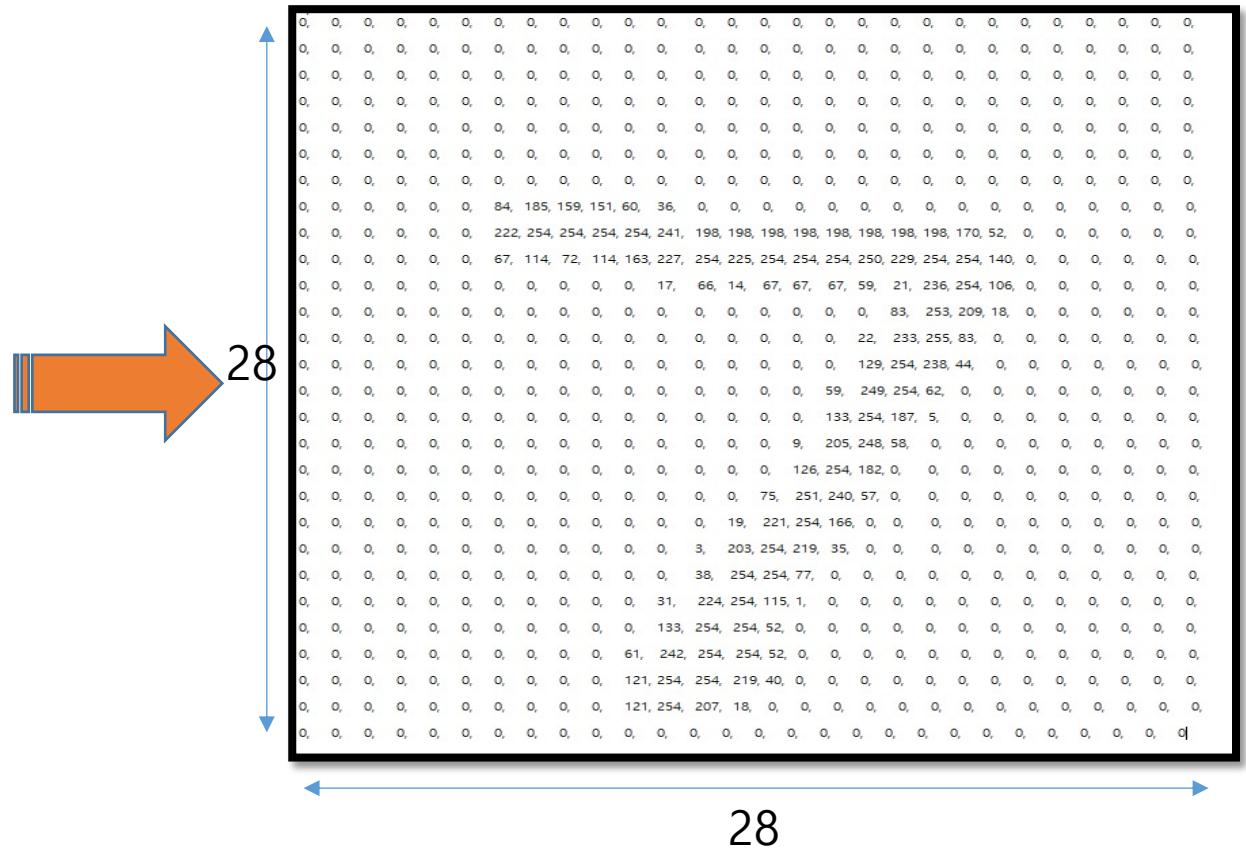
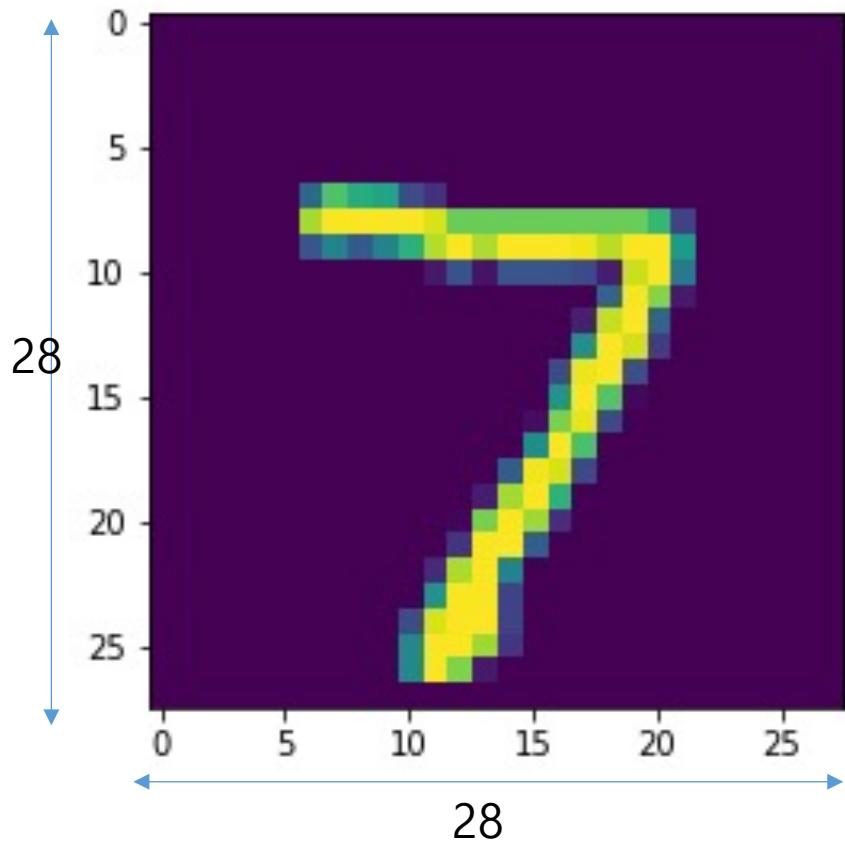
```
1 df = pd.read_csv('./mnist_test (2).csv', names = a)
```

```
2 df
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
...
9995	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9996	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9997	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9998	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9999	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

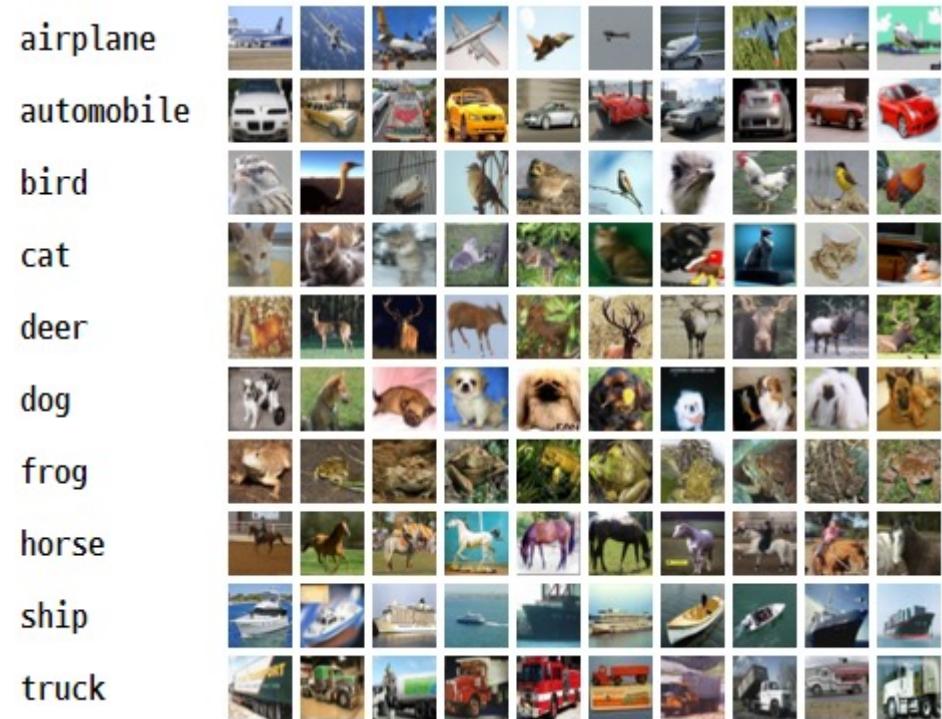
10000 rows x 785 columns

MNIST



✎ MNIST 이외의 Dataset

- CIFAR-10



- Fashion- MNIST



CONTENTS

- 1 MNIST 소개**
- 2 MNIST 분류 딥러닝 모델**
- 3 CODE**

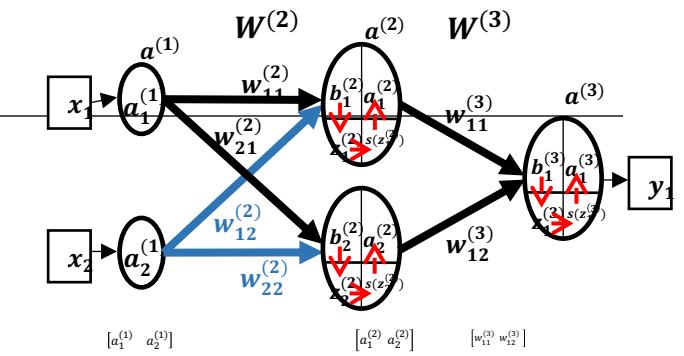
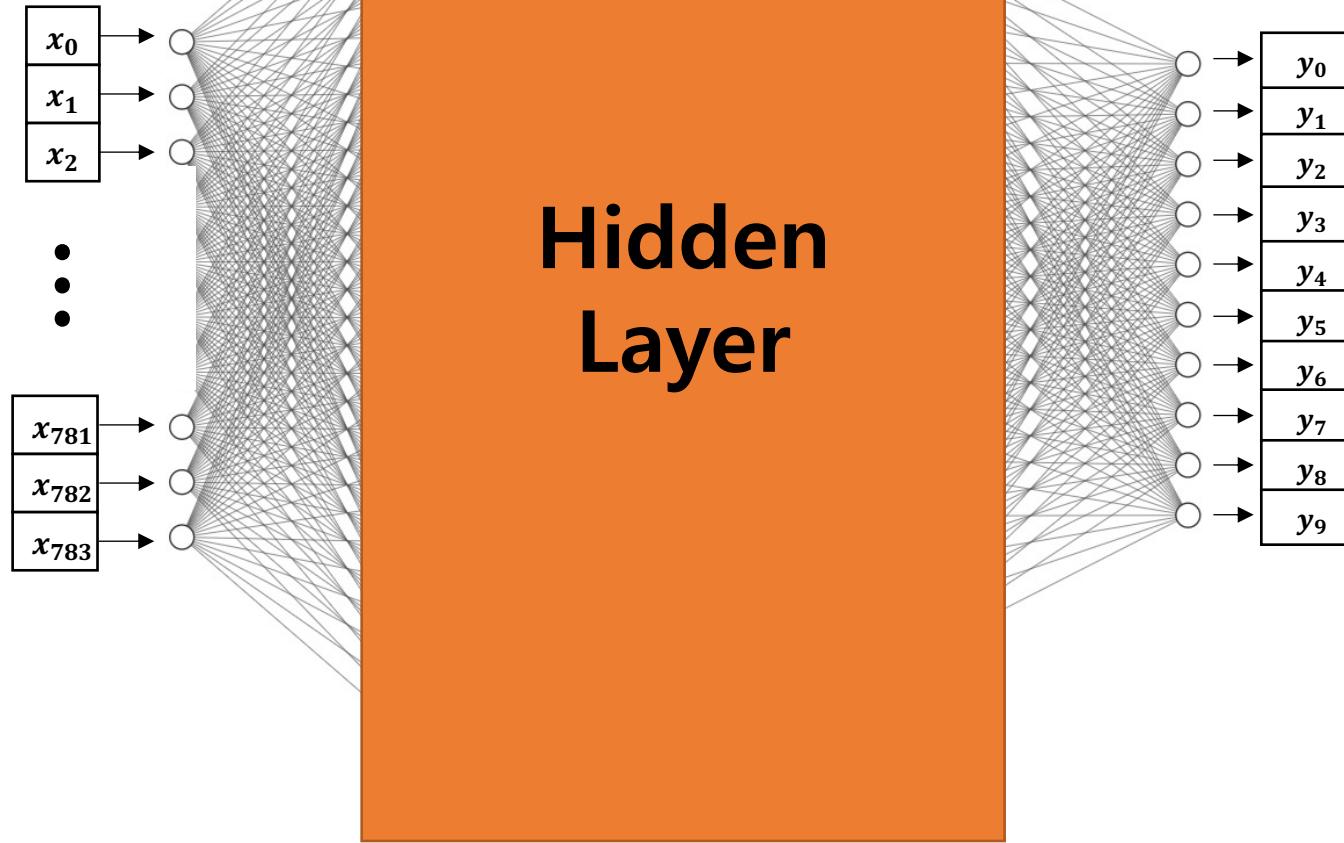
MNIST 분류 딥러닝 모델

모델 개요

- 이미지를 한 개의 벡터 (size = 784)로 표현
→ 784개의 input이 필요
- 0~9 사이의 값을 예측
→ 10개의 output이 필요

MNIST 분류 딥러닝 모델

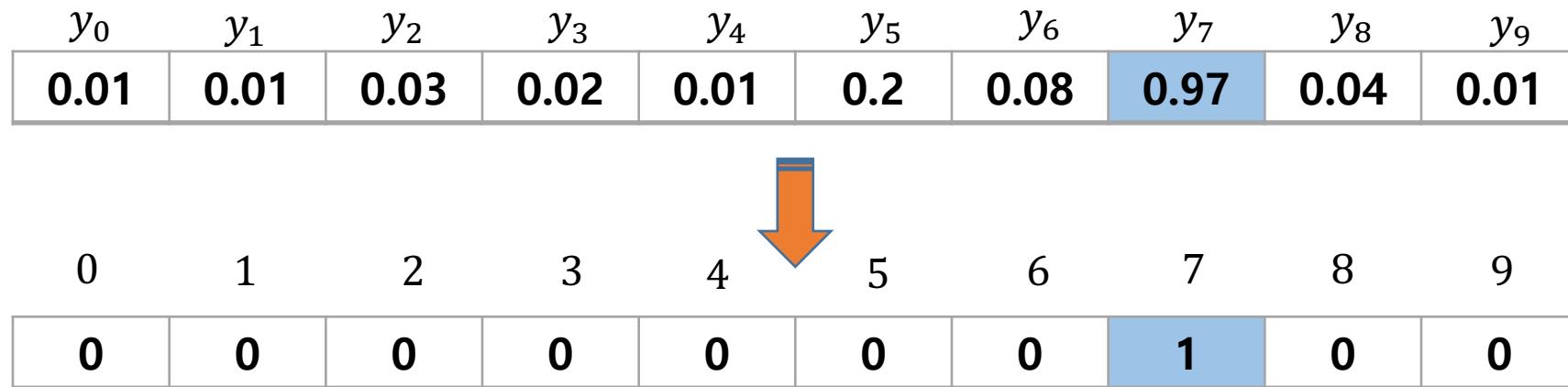
모델 개요



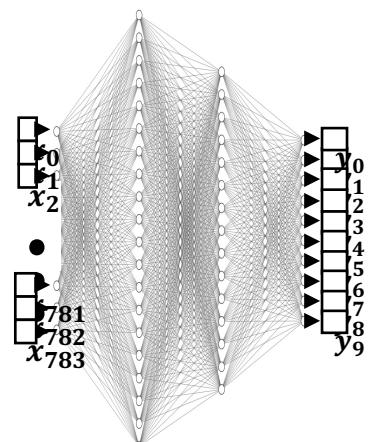
MNIST 분류 딥러닝 모델

One-Hot encoding?

- 출력층 노드에서 최댓값을 가지는 노드의 인덱스를 정답으로 판단하는 기법



→ y_7 노드의 값이 0.97로 가장 큼
→ 원핫 인코딩 기법에 의해 인덱스 값 7을 정답으로 판단
(argmax)



MNIST 분류 딥러닝 모델

📝 30개의 노드를 갖는 hidden layer 한 층으로 구현해보자!

- Input : 784
- Hidden layer1 : 30
- Output : 10

CONTENTS

- 1 MNIST 소개
- 2 MNIST 분류 딥러닝 모델
- 3 CODE

MNIST 분류 딥러닝 모델 – CODE

📝 Numpy 를 이용한 Data load

```
● ● ●  
1 training_data = np.loadtxt('./mnist_train.csv', delimiter=',', dtype=np.float32)  
2 test_data = np.loadtxt('./mnist_test.csv', delimiter=',', dtype=np.float32)  
3 print(f"training_data.shape = {training_data.shape} test_data.shape = {test_data.shape}")  
4  
5 #training_data.shape = (60000, 785) test_data.shape = (10000, 785)
```

```
np.loadtxt( '파일 경로', delimiter = '구분문자', dtype = data type)
```

MNIST 분류 딥러닝 모델 – CODE

수치 미분, Sigmoid

```
● ● ●

1 def numerical_derivative(f, x):
2     delta_x = 1e-4 # 0.0001
3     grad = np.zeros_like(x)
4
5     it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])
6
7     while not it.finished:
8         idx = it.multi_index
9         tmp_val = x[idx]
10        x[idx] = float(tmp_val) + delta_x
11        fx1 = f(x) # f(x+delta_x)
12
13        x[idx] = tmp_val - delta_x
14        fx2 = f(x) # f(x-delta_x)
15        grad[idx] = (fx1 - fx2) / (2*delta_x)
16
17        x[idx] = tmp_val
18        it.iternext()
19
20    return grad
21
22 # sigmoid 함수
23
24 def sigmoid(x):
25     return 1 / (1+np.exp(-x))
```

MNIST_Test 클래스 구성

```
1 class MNIST_Test:
2     # 생성자
3     def __init__(self, input_nodes, hidden_nodes, output_nodes, learning_rate):
4         # 손실함수
5         def feed_forward(self):
6             # 손실 값 계산
7             def loss_val(self):
8                 # 수치미분을 이용하여 손실함수가 최소가 될때 까지 학습하는 함수
9                 def train(self, input_data, target_data):
10                    # query, 즉 미래 값 예측 함수
11                    def predict(self, input_data):
12                        # 정확도 측정함수
13                        def accuracy(self, input_data, target_data):
14
```

MNIST 분류 딥러닝 모델 – CODE



생성자 `__init__`

```
● ● ●  
1 def __init__(self, input_nodes, hidden_nodes, output_nodes, learning_rate):  
2  
3     self.input_nodes = input_nodes  
4     self.hidden_nodes = hidden_nodes  
5     self.output_nodes = output_nodes  
6  
7     # 은닉층 가중치 W2 Xavier 방법으로 self.W2 가중치 초기화  
8     self.W2 = np.random.randn(self.input_nodes, self.hidden_nodes) / np.sqrt(self.input_nodes)  
9     self.b2 = np.random.rand(self.hidden_nodes)  
10  
11    # 출력층 가중치는 W3 Xavier 방법으로 self.W3 가중치 초기화  
12    self.W3 = np.random.randn(self.hidden_nodes, self.output_nodes) / np.sqrt(self.hidden_nodes)  
13    self.b3 = np.random.rand(self.output_nodes)  
14  
15    # 학습률 learning rate 초기화  
16    self.learning_rate = learning_rate
```

$$W = \frac{\text{np.random.randn}(\text{pre_node_num}, \text{next_node_num})}{\text{np.sqrt}(\text{pre_node_num})}$$

Xavier initialization ?

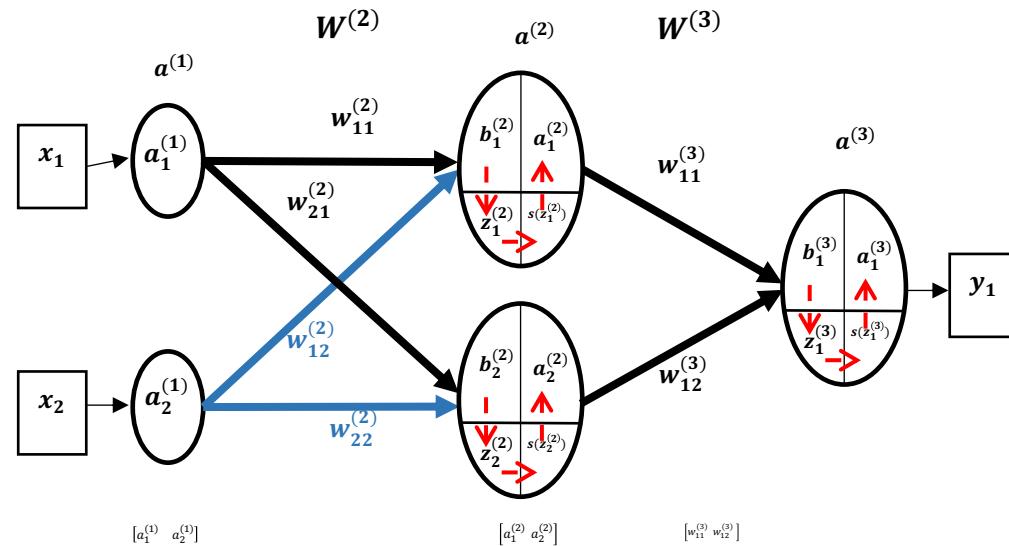
정규분포에서 random하게 뽑은 값을 앞 계층의 노드 수 제곱근으로 나눠주는 가중치 초기값
관련 기술, Sigmoid function을 사용할 때 효율적이라고 알려져 있다.

MNIST 분류 딥러닝 모델 – CODE

✎ 생성자 `_init_`

- Xavier initialization이 왜 필요할까?

- $W = \text{np.zeros}()$ 로 초기화해보자: 가중치가 0이라서 모든 뉴런은 모두 다 같은 연산을 하게 되어 결국 gradient가 같을 것이고 모든 뉴런이 똑같이 업데이트 되어 학습이 안됨.



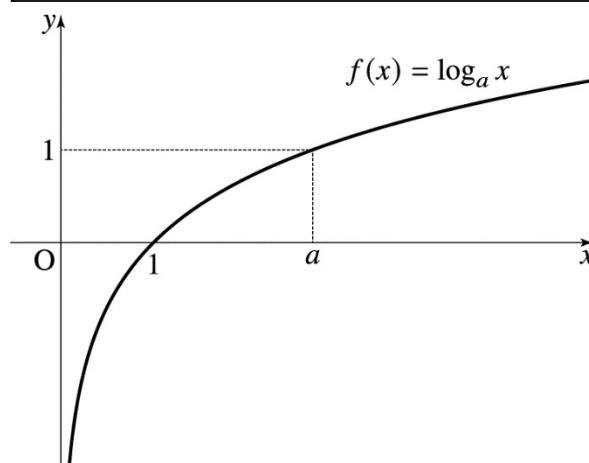
따라서, Standard gaussian으로 뽑은 값을 '입력의 수'로 스케일링 해줘서 입력 개수에 따라 W값의 분포를 변경!

- 결과적으로 Xavier Initialization는 입/출력의 분산을 맞춰줌

MNIST 분류 딥러닝 모델 – CODE

✎ 손실함수 feed_forward

```
● ● ●  
1 def feed_forward(self):  
2  
3     delta = 1e-7    # log 무한대 발산 방지  
4  
5     z1 = np.dot(self.input_data, self.W2) + self.b2  
6     y1 = sigmoid(z1)  
7  
8     z2 = np.dot(y1, self.W3) + self.b3  
9     y = sigmoid(z2)  
10  
11    # cross-entropy  
12    return -np.sum( self.target_data*np.log(y + delta) + (1-self.target_data)*np.log((1 - y)+delta) )
```



$f(x) = \log_a x$
Delta가 없으면?
 $-\infty$ 로 발산!

MNIST 분류 딥러닝 모델 – CODE

✎ 손실 값 계산 loss_val

```
● ● ●  
1 def loss_val(self):  
2  
3     delta = 1e-7      # log 무한대 발산 방지  
4  
5     z1 = np.dot(self.input_data, self.W2) + self.b2  
6     y1 = sigmoid(z1)  
7  
8     z2 = np.dot(y1, self.W3) + self.b3  
9     y = sigmoid(z2)  
10  
11    # cross-entropy  
12    return -np.sum( self.target_data*np.log(y + delta) + (1-self.target_data)*np.log((1 - y)+delta) )
```

MNIST 분류 딥러닝 모델 – CODE

📝 학습 train

```
● ● ●  
1 def train(self, input_data, target_data):  
2  
3     self.input_data = input_data  
4     self.target_data = target_data  
5  
6     f = lambda x : self.feed_forward()  
7  
8     self.W2 -= self.learning_rate * numerical_derivative(f, self.W2)  
9  
10    self.b2 -= self.learning_rate * numerical_derivative(f, self.b2)  
11  
12    self.W3 -= self.learning_rate * numerical_derivative(f, self.W3)  
13  
14    self.b3 -= self.learning_rate * numerical_derivative(f, self.b3)
```

MNIST 분류 딥러닝 모델 – CODE

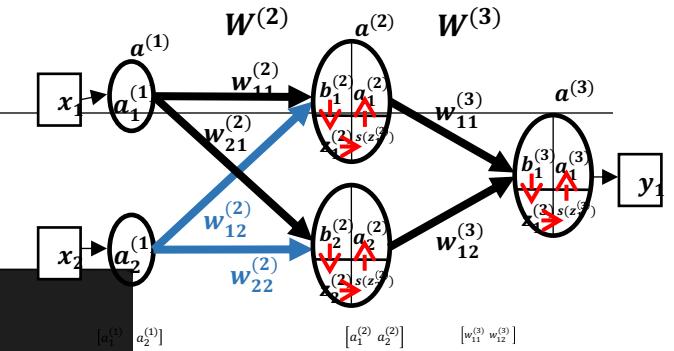
✎ 예측 predict

```
1 def predict(self, input_data):
2
3     z2 = np.dot(input_data, self.W2) + self.b2
4     a2 = sigmoid(z2)
5
6     z3 = np.dot(a2, self.W3) + self.b3
7     y = a3 = sigmoid(z3)
8
9     # MNIST 경우는 one-hot encoding 을 적용하기 때문에
10    # 0 또는 1 이 아닌 argmax() 를 통해 최대 인덱스를 넘겨주어야 함
11    predicted_num = np.argmax(y)
12
13    return predicted_num
```

MNIST 분류 딥러닝 모델 – CODE

정확도 측정 accuracy

```
 1 def accuracy(self, input_data, target_data):
 2     matched_list = []
 3     not_matched_list = []
 4     # list which contains (index, label, prediction) value
 5     index_label_prediction_list = []
 6     # temp list which contains label and prediction in sequence
 7     temp_list = []
 8     for index in range(len(input_data)):
 9         label = int(target_data[index])
10         # normalize
11         data = (input_data[index, :] / 255.0 * 0.99) + 0.01 ←
12         predicted_num = self.predict(data)
13         if label == predicted_num:
14             matched_list.append(index)
15         else:
16             not_matched_list.append(index)
17             temp_list.append(index)
18             temp_list.append(label)
19             temp_list.append(predicted_num)
20             index_label_prediction_list.append(temp_list)
21             temp_list = []
22     print("Current Accuracy = ", len(matched_list)/(len(input_data)) )
23
24     return matched_list, not_matched_list, index_label_prediction_list
```



입력값이 0일 경우
를 생각해보자! 해당
노드의 W값이 전혀
결과에 영향을 미치
지 않음.
따라서, 입력값을
0.01~1 사이 값으로
정규화

MNIST 분류 딥러닝 모델 – CODE

📝 Hyperparameters

```
● ● ●  
1 i_nodes = training_data.shape[1] - 1    # input nodes 개수  
2 h1_nodes = 30   # hidden nodes 개수. Test 8->30  
3 o_nodes = 10    # output nodes 개수  
4 lr = 1e-2      # learning rate  
5 epochs = 1     # 반복횟수
```

Training_data.shape[1] = 785
정답을 나타내는 요소 1개 빼야함.

Hyperparameters?

모델링 시 사용자가 직접 세팅해주는 매개변수로
딥러닝 모델이 데이터를 통해 구하는(학습하는) 매개변수와 구분된다.

MNIST 분류 딥러닝 모델 – CODE

학습 (많은 시간 소요)

```
●●●

1 # 손실함수 값을 저장할 list 생성
2 loss_val_list = []
3 # MNIST_Test 객체 생성
4 obj = MNIST_Test(i_nodes, h1_nodes, o_nodes, lr)
5 print("Neural Network Learning using Numerical Derivative...")
6 start_time = datetime.now()
7
8 for step in range(epochs):
9     for index in range(len(training_data)):
10         # input_data, target_data normalize
11         input_data = ((training_data[index, 1:] / 255.0) * 0.99) + 0.01
12         target_data = np.zeros(o_nodes) + 0.01
13         target_data[int(training_data[index, 0])] = 0.99 ←
14         obj.train(input_data, target_data)
15
16         if (index % 200 == 0):
17             print("epochs = ", step, ", index = ", index, ", loss value = ", obj.loss_val())
18         # 손실함수 값 저장
19         loss_val_list.append(obj.loss_val())
20 end_time = datetime.now()
21 print("")
22 print("Elapsed Time => ", end_time - start_time)
```

Target값도 0.01~1.0
사이 값으로 정규화

학습	epochs = 0 , index = 0 , loss value = 12.881948352151234 epochs = 0 , index = 200 , loss value = 3.2624176452836373
결과	epochs = 0 , index = 59400 , loss value = 3.007410588066169 epochs = 0 , index = 59600 , loss value = 0.6170549447664146 epochs = 0 , index = 59800 , loss value = 0.6065179201274236

MNIST 분류 딥러닝 모델 – CODE

검증 코드

```
1 test_data = np.loadtxt('./mnist_test.csv', delimiter=',', dtype=np.float32)
2 print("test_data.shape = ", test_data.shape)
3
4 test_input_data = test_data[ :, 1: ]
5 test_target_data = test_data[ :, 0 ]
6
7 (true_list_1, false_list_1, index_label_prediction_list) = obj.accuracy(test_input_data, test_target_data)
```

검증
결과

Current Accuracy = 0.924

결론

- 30개의 노드를 갖는 1개의 Hidden layer만으로 92.4%의 정확도
- Hidden layer를 더 추가하면?



감사합니다.

