


Branch: master ▾

Find file

Copy path

AI\_Spring\_2019 / README.md

 jeongsooyeon Update README.md  
a0ef571 6 minutes ago

1 contributor

RawBlameHistory

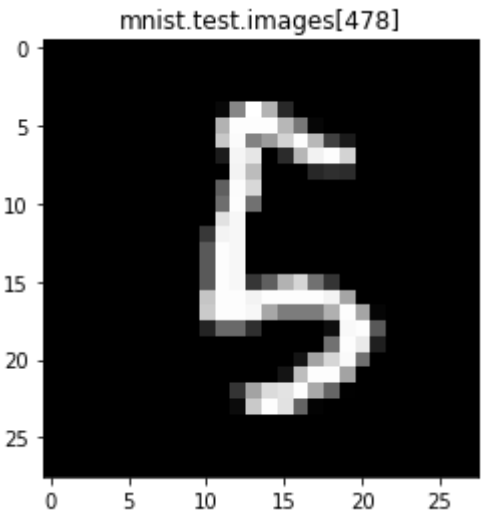
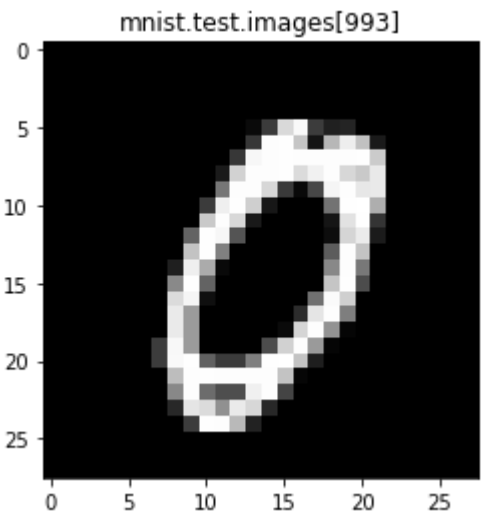
140 lines (70 sloc) 4.21 KB

# AI\_Spring\_2019

인공지능개론 2019 봄학기

- 03\_hello\_dataset1\_MNIST.ipynb



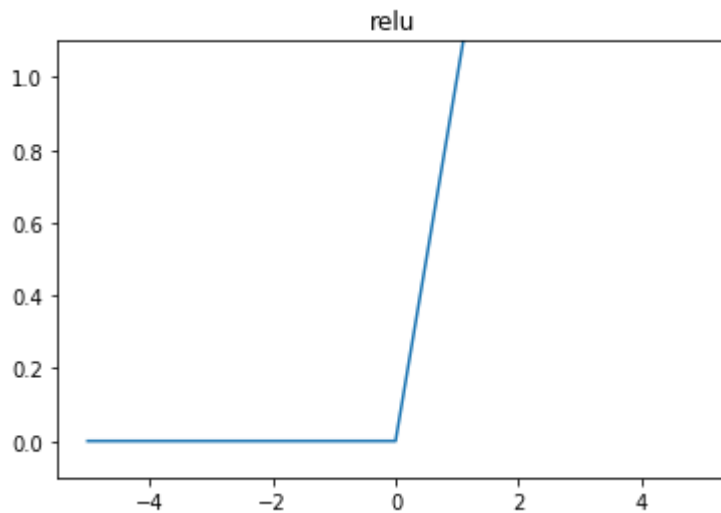
`pixels = img1.reshape((28, 28))` # shape을 바꾼다. #  $28 \times 28 = 784$ , 각각의 글자는 2828이다. 정보가 784개고 28개로 쪼갬다.

`plt.imshow(pixels, cmap='gray')` #plt에게 흑백으로 그리도록 한다.

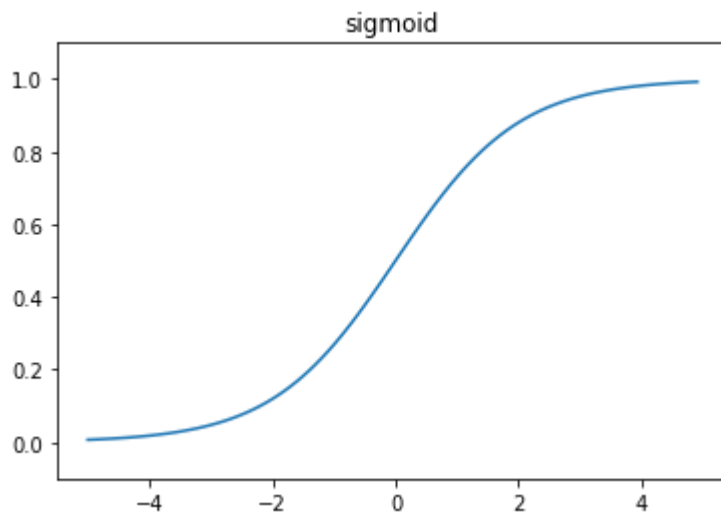
`plt.title('mnist.test.images[{}].format(idx))` # 이미지로 저장하게 해준다.

`plt.show()` # 그림을 그린다.

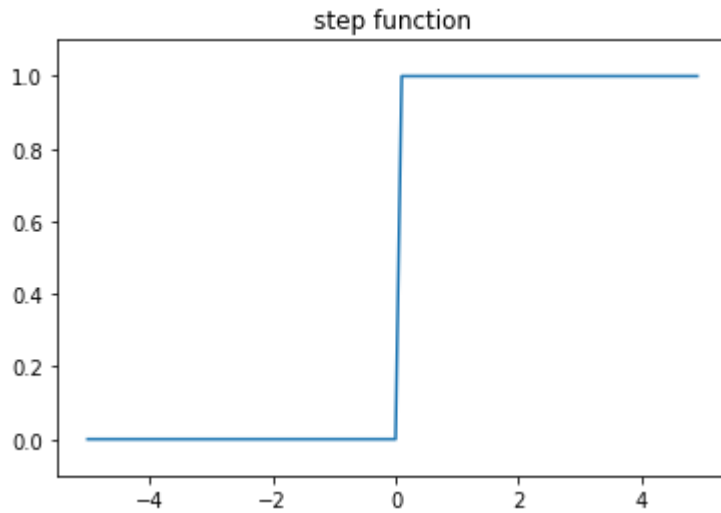
- 04\_2\_ActivationFunctions.ipynb



`plt.title('relu')`



`plt.title('sigmoid')`



```
plt.title('step function')
```

- 05\_Linear\_regression\_course.ipynb

```
cost = tf.reduce_mean(tf.square(hypothesis - y_train))
```

#가장 기초적인 cost #가설값과 y값의 거리의차이를 제곱을 통해 +, - 구분없이 절대적인 거리를 구하고 그값을 cost에 저장한다.

optimizer = tf.train.GradientDescentOptimizer(learning\_rate=0.01) #optimizer를 통해 에러를 보며 train을 풀어간다.

```
train = optimizer.minimize(cost) #cost를 최소화 한다.
```

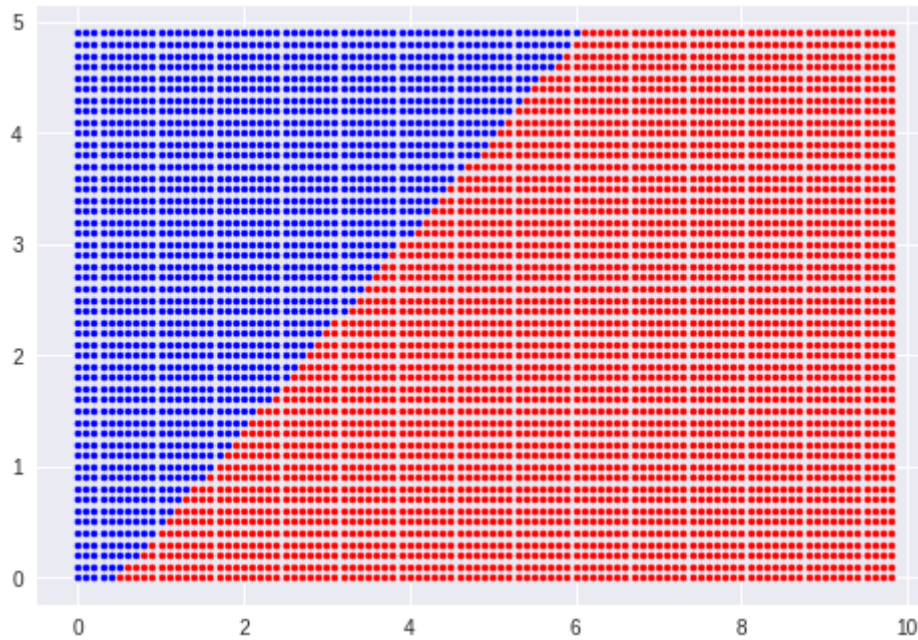
- 06\_1\_Perceptron\_prediction\_gates.ipynb

Perceptrons - Making Predictions

AND Gate, NAND Gate, OR Gate, XOR Gate

XOR cannot be expressed as a single layer Perceptron. #XOR은 단일 레이어 퍼셉트론으로 표현 될 수 없다.

- 06\_2\_Perceptrons\_prediction.ipynb



퍼세트론으로 공간을 나눈다. 점의 밀도를 높일수록 경계값이 명확해진다.

- 06\_3\_Perceptrons\_training\_AND\_v1.ipynb

$$\begin{aligned}
 \frac{\partial J(m, b)}{\partial m} &= \frac{1}{n} \sum_{i=1}^n -2x^{(i)}(y_i - (mx^{(i)} + b)) \\
 &= \frac{2}{n} \sum_{i=1}^n x^{(i)}((mx^{(i)} + b) - y^{(i)}) \\
 &= \frac{2}{n} \sum_{i=1}^n x^{(i)}(\hat{y}^{(i)} - y^{(i)})
 \end{aligned}$$

partial derivative with respect to m #웨이트로 미분한다. 바이어스로 미분 할 때와는 다르게 x를 곱한다.

$$\begin{aligned}
 \frac{\partial J(m, b)}{\partial b} &= \frac{1}{n} \sum_{i=1}^n -2(y^{(i)} - (mx^{(i)} + b)) \\
 &= \frac{-2}{n} \sum_{i=1}^n (y^{(i)} - (mx^{(i)} + b)) \\
 &= \frac{2}{n} \sum_{i=1}^n (\hat{y}^{(i)} - y^{(i)})
 \end{aligned}$$

partial derivative with respect to b #바이어스로 미분한다.

- 07\_0\_TF\_reduce\_mean.ipynb

import numpy as np #수학을 다루기 쉽게 해준다. #보통 이걸로 평균내는데, 굳이 텐서플로우에서 평균내는거 또 만들었다.

import tensorflow as tf #텐서플로우로 평균내는 것

m2 = tf.reduce\_mean(c, axis = 1) #reduce\_mean은 평균을 구하는것

- 07\_1\_TF\_AND\_simplest.ipynb

AND학습하기 아까 같은 문제를 텐서플로우로 풀어보기

- 07\_2\_TF\_XOR.ipynb

```

X step=0 / cost=0.30544182658195496
step=1000 / cost=0.2510456442832947
step=2000 / cost=0.2501783072948456
step=3000 / cost=0.24968695640563965
step=4000 / cost=0.24883615970611572
step=5000 / cost=0.24662882089614868
step=6000 / cost=0.24077501893043518
step=7000 / cost=0.2259557843208313
step=8000 / cost=0.20062844455242157
step=9000 / cost=0.17635396122932434
step=10000 / cost=0.15907993912696838

```

```

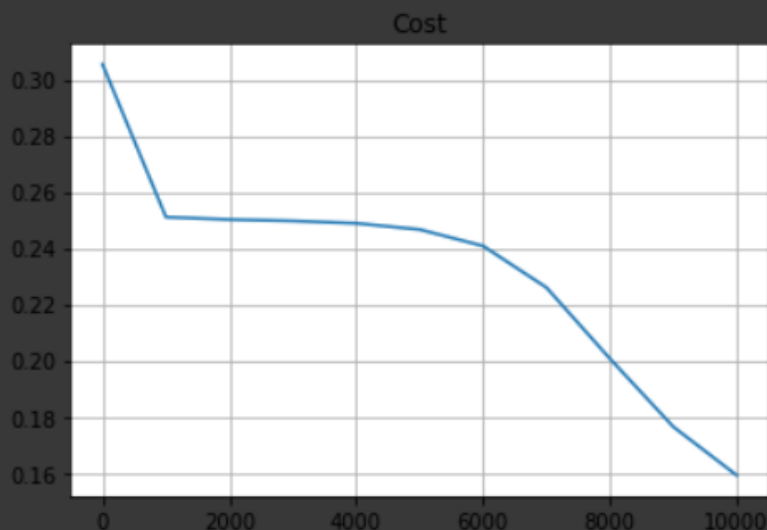
[ ] plt.plot(vstep, vcost)
    plt.grid()
    plt.title('Cost')

```

```

☞ Text(0.5, 1.0, 'Cost')

```



- HW : 위의 코드를 변형하여 XOR 학습시 얻어진 Cost 그래프를 그리시오. Hint : List 사용

코스트도 feed를 해주고 천번마다 출력한다. 결과보면, 코스트가 점점 줄어든다.

- 07\_3\_TF\_MNIST\_hello\_keras.ipynb

```
[ ] model.fit(x_train, y_train, epochs=5) #모델을 맞춰라. #에포크가 5번 돌리라는거.
```

```
Epoch 1/5
60000/60000 [=====] - 25s 422us/sample - loss: 0.1913 - acc: 0.9419
Epoch 2/5
60000/60000 [=====] - 25s 418us/sample - loss: 0.0876 - acc: 0.9725
Epoch 3/5
60000/60000 [=====] - 25s 418us/sample - loss: 0.0638 - acc: 0.9800
Epoch 4/5
60000/60000 [=====] - 25s 415us/sample - loss: 0.0511 - acc: 0.9844
Epoch 5/5
60000/60000 [=====] - 25s 420us/sample - loss: 0.0395 - acc: 0.9880
<tensorflow.python.keras.callbacks.History at 0x7f661a793518>
```

```
model.summary()
```

Layer (type)	Output Shape	Param #
flatten_2 (Flatten)	(None, 784)	0
dense_3 (Dense)	(None, 512)	401920
dense_4 (Dense)	(None, 256)	131328
dense_5 (Dense)	(None, 256)	65792
dense_6 (Dense)	(None, 10)	2570
Total params: 601,610		
Trainable params: 601,610		
Non-trainable params: 0		

$$(784 * 512) + 512 = 401920$$

$$(512 * 10) + 10 = 5130$$

$$((784 * 512) + 512) + ((512 * 10) + 10) = 407050$$

Test accuracy, Training accuracy

- 07\_4\_Keras\_MNIST\_CNN\_keras.ipynb

```
[ ] model.summary() #타이핑한 모델의 구조를 출력해 준다.
```



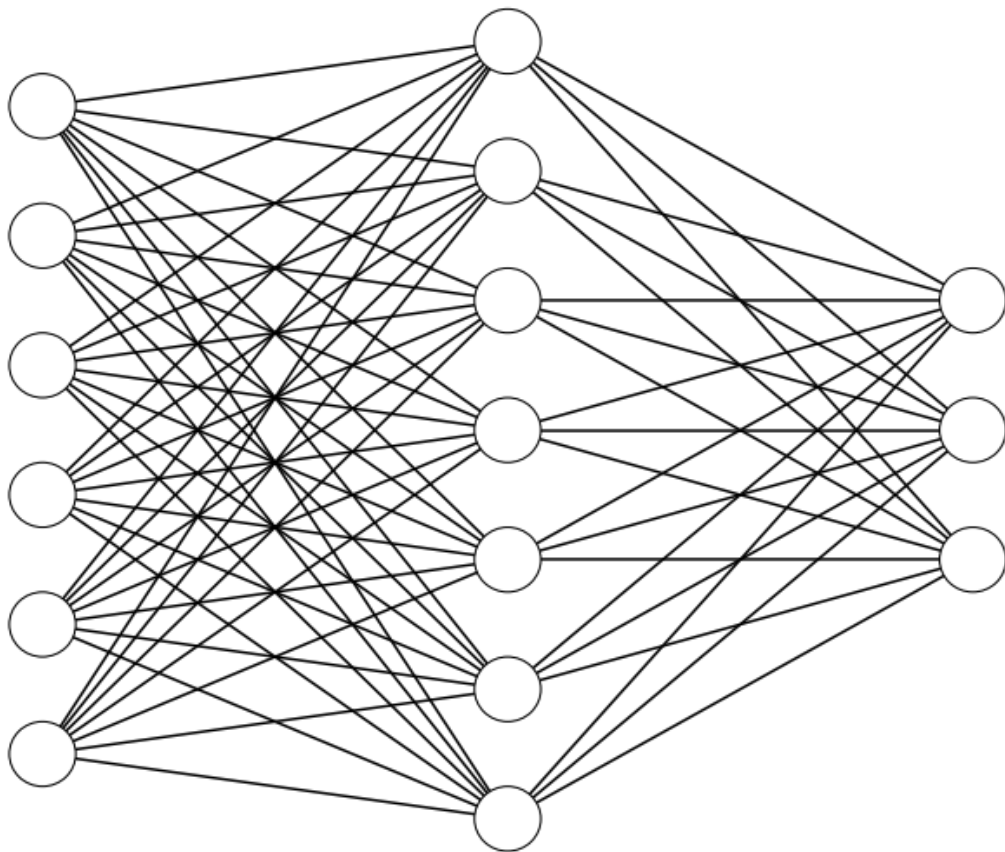
Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 26, 26, 32)	320
conv2d_2 (Conv2D)	(None, 24, 24, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 12, 12, 64)	0
dropout_1 (Dropout)	(None, 12, 12, 64)	0
flatten_1 (Flatten)	(None, 9216)	0
dense_1 (Dense)	(None, 128)	1179776
dropout_2 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 10)	1290
Total params: 1,199,882		
Trainable params: 1,199,882		
Non-trainable params: 0		

MLNN의 문제를 해결하고자 만들어진 것이 합성곱 계층(CNN)이다.

MNIST 데이터 같은 이미지 데이터는 일반적으로 채널, 세로, 가로 이렇게 3차원으로 구성된 데이터이다.

합성곱에서는 3차원 데이터를 입력하고 3차원의 데이터로 출력하므로 형상을 유지할 수 있다

- **Neural\_network.ipynb**



뉴럴 네트워크

`draw_neural_net(ax, .1, .9, .1, .9, [6, 7, 3])` #뉴런의 수를 늘릴 수 있다.