Machine learning & deep learning basics

Practice Session 3

Changho Suh

January 22, 2024

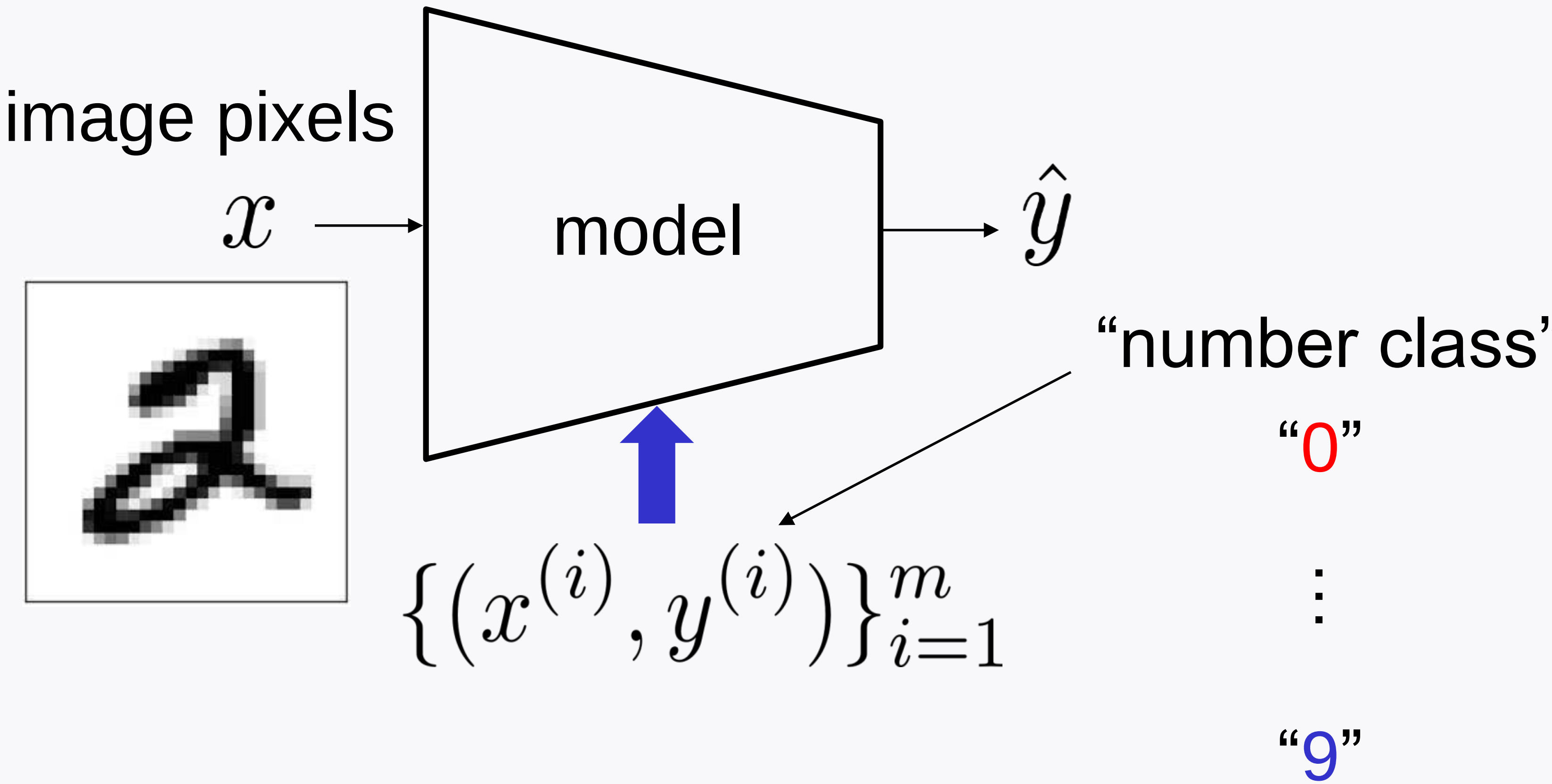
Outline

Will learn how to do **TensorFlow** implementation:

1. Least Squares
2. Logistic regression
3. **Deep learning**

Will do this in the context of **handwritten digit classification**.

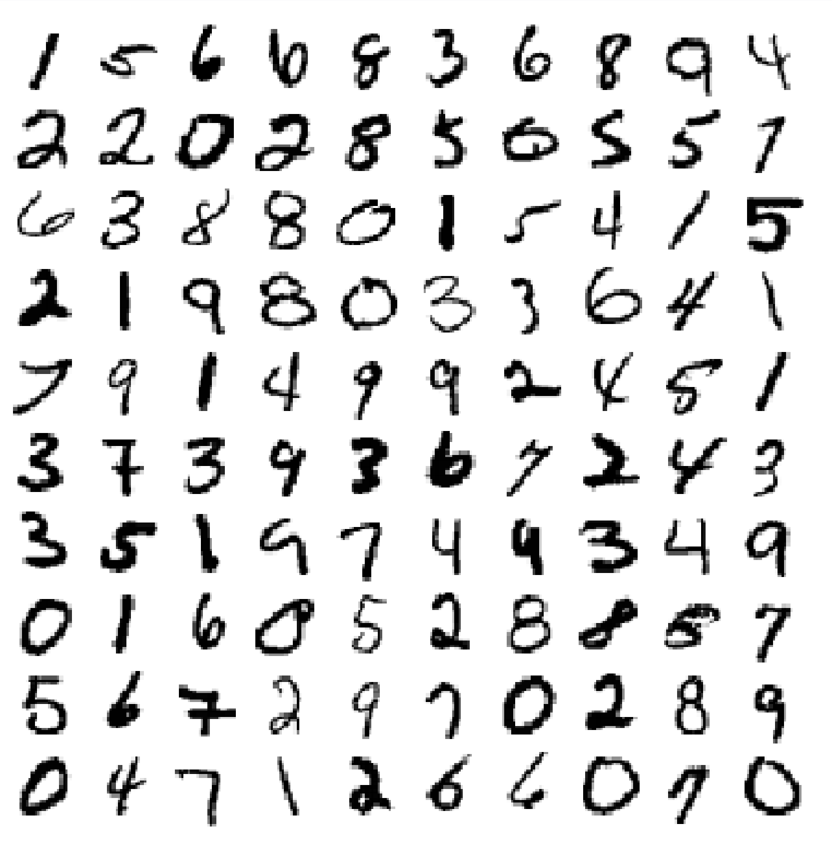
Handwritten digit classification



MNIST dataset

$$\{(x^{(i)}, y^{(i)})\}_{i=1}^{m} \qquad m = 60,000$$

Examples:

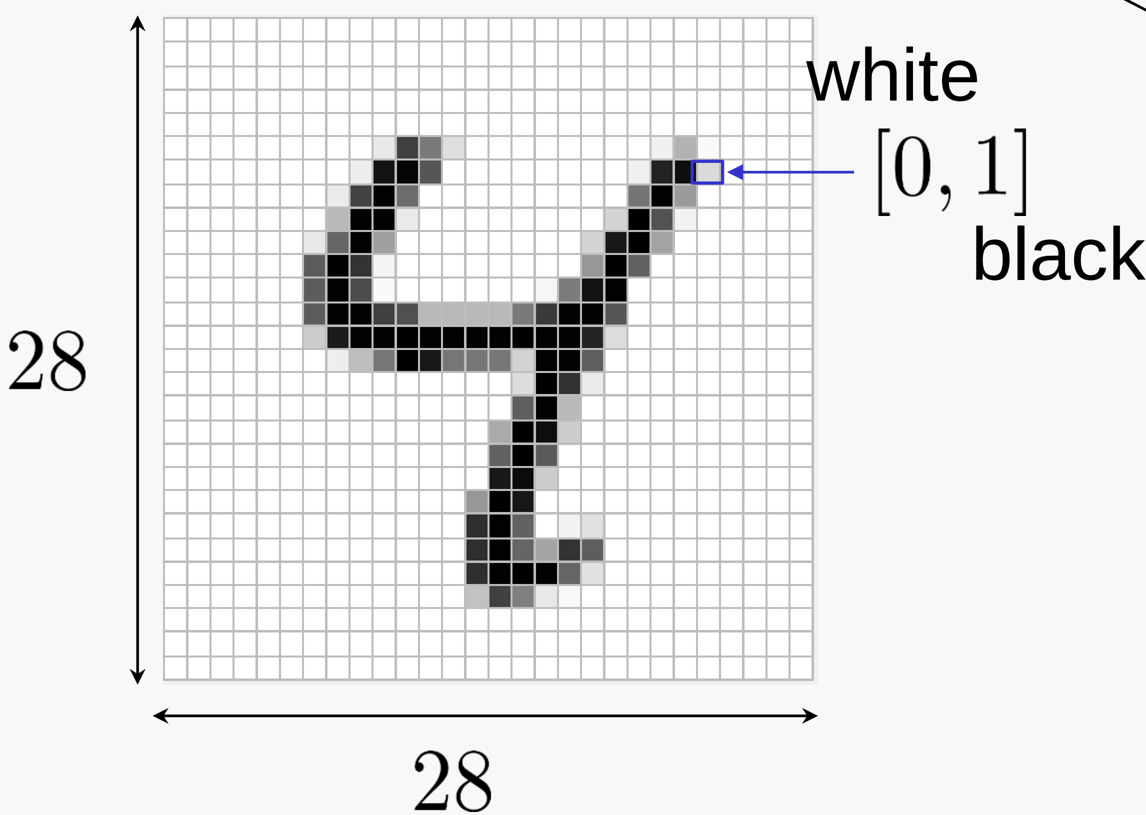


Yann LeCun 1998

Input image & label

$$\{(x^{(i)}, y^{(i)})\}_{i=1}^{m} \qquad m = 60,000$$

$$y^{(i)} \in \{0, 1, \ldots, 9\}$$



How to load MNIST dataset

```
from tensorflow.keras.datasets import mnist
```

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(60000, 28, 28)
(60000,)
(10000, 28, 28)
(10000,)
```

Pixel data

```
print(X_train[0])
```

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3  18  18  18 126 136
  175  26 166 255 247 127   0   0   0   0]
 [  0   0   0   0   0   0   0   0  30  36  94 154 170 253 253 253 253 253
  225 172 253 242 195  64   0   0   0   0]
 [  0   0   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251
   93  82  82  56  39   0   0   0   0   0]
 [  0   0   0   0   0   0   0  18 219 253 253 253 253 253 198 182 247 241
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
```

Range: 0 ~ 255

Normalization (preprocessing)

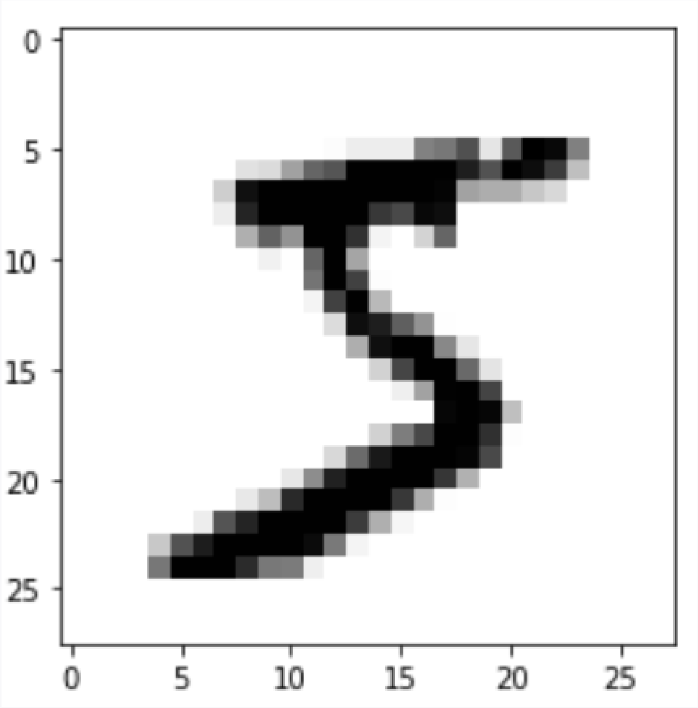
```
X_train, X_test = X_train/255., X_test/255
```

Data visualization

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

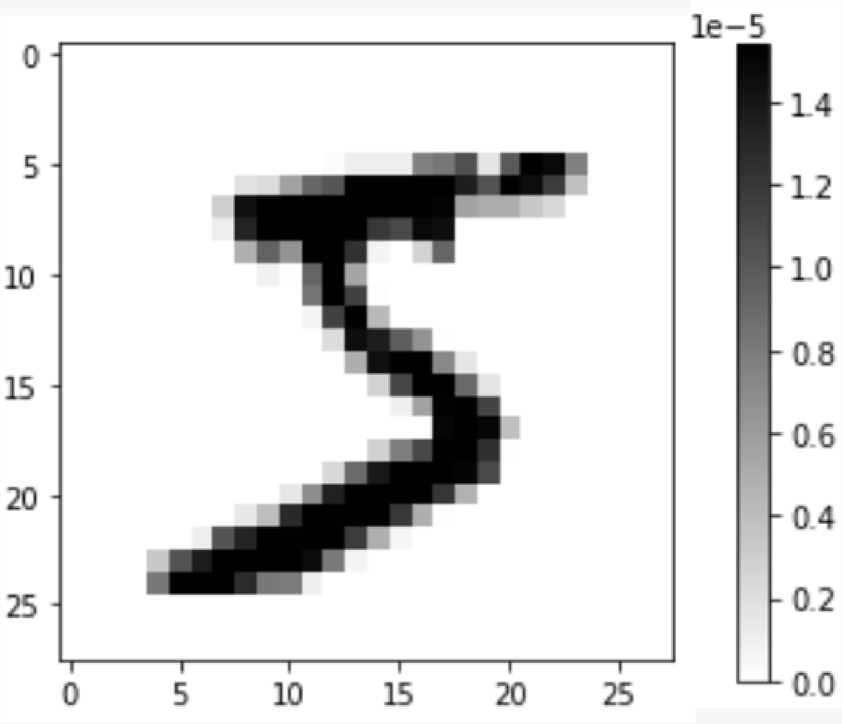
'gray': background is black

```python
plt.imshow(X_train[0], cmap='gray_r') # gray_r: background is white
```



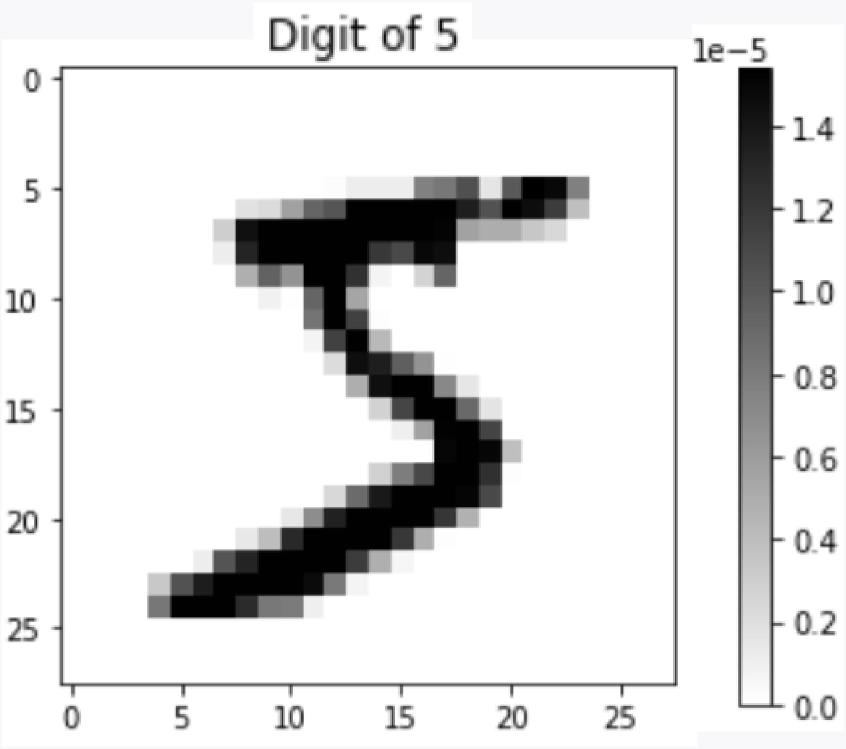
Data visualization

```
import matplotlib.pyplot as plt
plt.imshow(X_train[0], cmap='gray_r') # gray_r: background is white
plt.colorbar() # Display a colored bar right next to an image
```



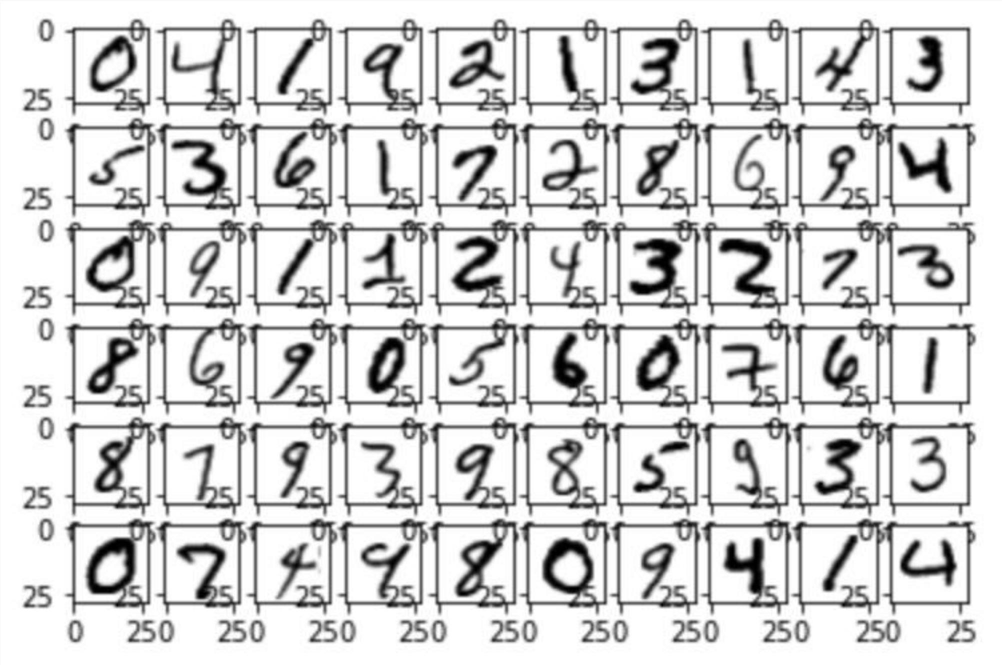
Data visualization

```
import matplotlib.pyplot as plt
plt.imshow(X_train[0], cmap='gray_r') # gray_r: background is white
plt.colorbar() # Display a colored bar right next to an image
plt.title('Digit of {}'.format(y_train[0]))
```



Plotting multiple figures

```python
import matplotlib.pyplot as plt

num_of_images = 60
for index in range(1,num_of_images+1):
    plt.subplot(6,10,index)
    plt.imshow(X_train[index], cmap = 'gray_r')
```



Plotting multiple figures

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

```
num_of_images = 60
for index in range(1,num_of_images+1):
    plt.subplot(6,10,index)
    plt.axis('off')
    plt.imshow(X_train[index], cmap = 'gray_r')
```

Least Squares

```
from sklearn.linear_model import RidgeClassifier
```

```
Model_LS = RidgeClassifier()
```

```
print(X_train.shape)
```

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

```
print(X_train.reshape(-1,28*28).shape)
```

```
(60000, 784)
```

Least Squares

```python
from sklearn.linear_model import RidgeClassifier
```

```python
Model_LS = RidgeClassifier()
```

```python
# training
Model_LS.fit(X_train.reshape(-1, 28*28), y_train)
```

```python
# prediction on test data
Model_LS.predict(X_test[0].reshape(1, -1))
```

```python
# evaluate test accuracy
Model_LS.score(X_train.reshape(-1, 28*28), y_train)
```

```
0.8574
```

Logistic regression

```
from sklearn.linear_model import LogisticRegression
```

```
Model_LR = LogisticRegression()
```

```
# training
Model_LR.fit(X_train.reshape(-1, 28*28), y_train)
```

```
# prediction on test data
Model_LR.predict(X_test[0].reshape(1,-1))
```

```
# evaluate test accuracy
Model_LR.score(X_test.reshape(-1, 28*28), y_test)
```

```
0.9258
```

Deep learning

```
from tensorflow.keras.models import Sequential
Model_NN = Sequential()

from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense

Model_NN.add(Flatten(input_shape=(28,28)))
Model_NN.add(Dense(128, activation='relu'))
Model_NN.add(Dense(10, activation='softmax'))
Model_NN.summary()
```

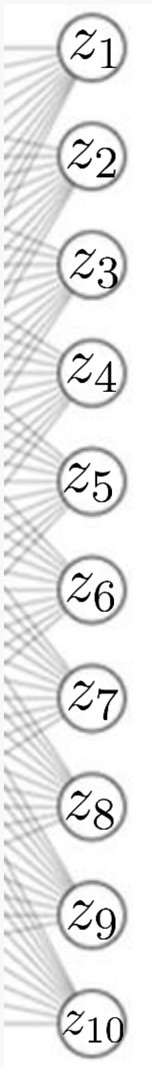
Deep learning

```
from tensorflow.keras.models import Sequential

Model_

from t
from t

Model_
Model_
Model_
Model_
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 128)               100480

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

Softmax activation at output layer

output layer



softmax

$$\hat{y}_j = \frac{e^{z_j}}{\sum_{k=1}^{10} e^{z_k}}$$

$\hat{y}_1$ $\mathbb{P}(x = \text{class } 0)$

$\hat{y}_2$ $\mathbb{P}(x = \text{class } 1)$

$\hat{y}_3$

$\hat{y}_4$

$\hat{y}_5$

$\hat{y}_6$

$\hat{y}_7$

$\hat{y}_8$

$\hat{y}_9$

$\hat{y}_{10}$ $\mathbb{P}(x = \text{class } 9)$

Optimal loss function

Turns out: The optimal loss function is again cross entropy loss.

$$\ell_{\mathsf{CE}}(y, \hat{y}) = \sum_{j=1}^{10} -y_j \log \hat{y}_j$$

y

one-hot vector (label=2)

$$\begin{array}{cc} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} & \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} \end{array}$$

Compile

```
from tensorflow.keras.optimizers import Adam

opt = Adam(learning_rate=1e-2,
           beta_1 = 0.9,
           beta_2 = 0.999)
```

```
Model_NN.compile(optimizer=opt,
                 loss='sparse_categorical_crossentropy',
                 metrics=['acc'])
```

Training

```
hist = Model_NN.fit(X_train, y_train, epochs=20)
```

```
Epoch 1/20
1875/1875 [==============================] - 1s 688us/step - loss: 0.2419 - acc:
0.9285
Epoch 2/20
1875/1875 [==============================] - 1s 715us/step - loss: 0.1621 - acc:
0.9545
```

:

```
Epoch 17/20
1875/1875 [==============================] - 1s 678us/step - loss: 0.0892 - acc:
0.9822
Epoch 18/20
1875/1875 [==============================] - 1s 707us/step - loss: 0.0875 - acc:
0.9825
Epoch 19/20
1875/1875 [==============================] - 1s 673us/step - loss: 0.0807 - acc:
0.9834
Epoch 20/20
1875/1875 [==============================] - 1s 659us/step - loss: 0.0931 - acc:
0.9834
```

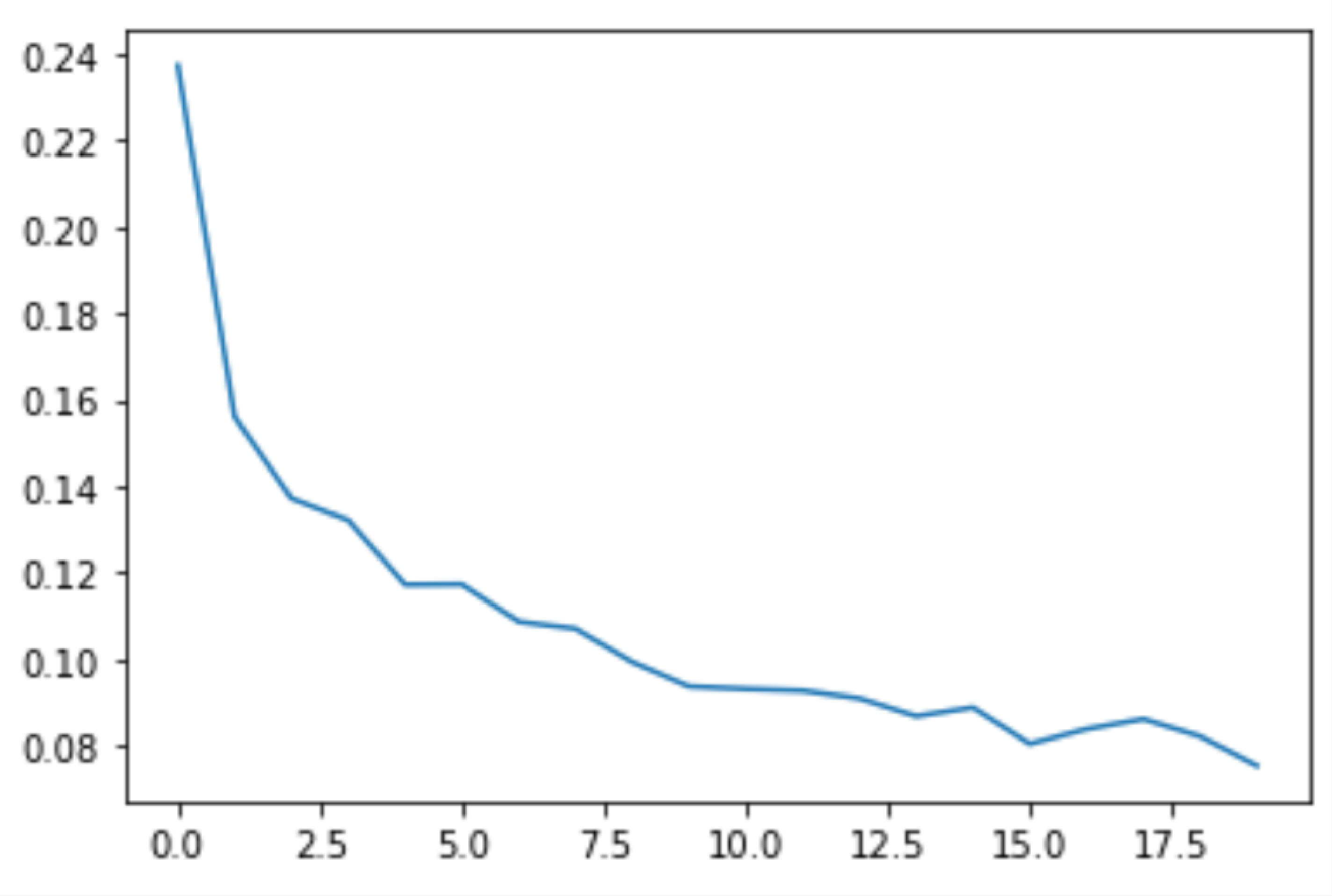
History

```
print(hist.history)
```


```
{'loss': [0.23721636831760406, 0.15599527955055237, 0.13709723949432373, 0.1321131
2890052795, 0.11719996482133865, 0.11729438602924347, 0.10861502587795258, 0.10701
850801706314, 0.09931265562772751, 0.09363316744565964, 0.0930742397904396, 0.0926
9481897354126, 0.09091321378946304, 0.08679905533790588, 0.0888138934969902, 0.080
38856834173203, 0.0838022530078888, 0.08611498773097992, 0.08213432878255844, 0.07
53093659877771], 'acc': [0.9310500025749207, 0.956166684627533, 0.96240001916885 3
8, 0.964733362197876, 0.9691833257675171, 0.9696999788284302, 0.971916675567627,
0.9732000231742859, 0.9745166897773743, 0.9763166904449463, 0.9783333539962769, 0.
9781666398048401, 0.9784166812896729, 0.9793166518211365, 0.9800833463668823, 0.98
04999828338623, 0.9807833433151245, 0.9817333221435547, 0.9823166728019714, 0.9836
166501045227]}
```

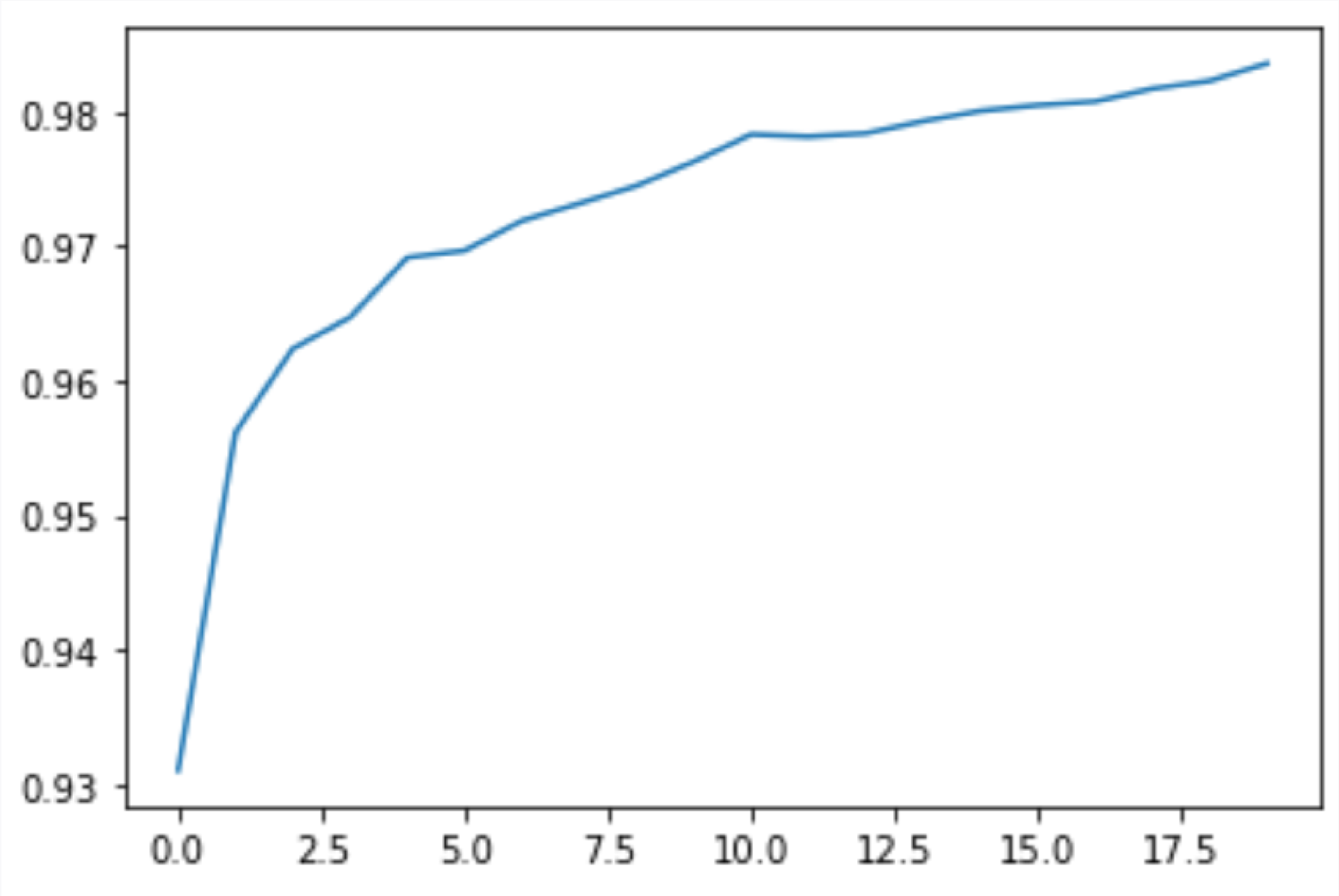
Plotting loss as a function of epoch

```
plt.plot(hist.history['loss'])
```



Plotting accuracy as a function of epoch

```
plt.plot(hist.history['acc'])
```



Testing

```
Model_NN.evaluate(X_test, y_test)
```

```
313/313 [==============================] - 0s 559us/step - loss: 0.3107 - acc: 0.9
661
```