In [142]:

```
import numpy
import plotly.offline
import scipy
import pandas
import matplotlib.pyplot as plt
import sklearn
from keras.models import Sequential
from keras.layers import Dense
import tensorflow as tf
from sklearn.preprocessing import LabelEncoder
```

Neural Networks

This week, we'll predict the digits in an image using a neural network. Let's check our dataset:

In [143]:

```
from keras.datasets import mnist

(train_X, train_Y), (test_X, test_Y) = mnist.load_data()
train_X.shape
```

Out[143]:

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

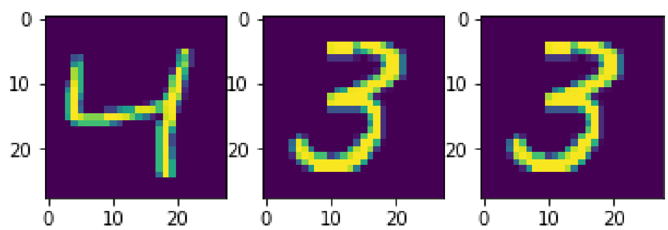
We can see that there are 60,000 square images in our training set, loading some images in the train set, and checking an example input:

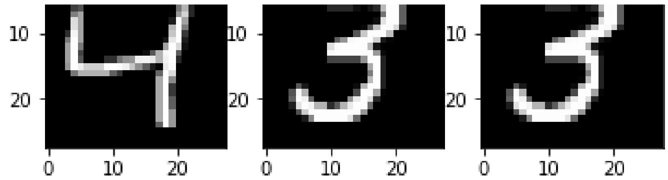
In [144]:

```
plot_inds = [2, 433, 433]
plt.figure()
# subplot(n_row n_col draw_index), e.g. 123: 3rd index on a 1x2 plot grid
[(plt.subplot(130 + ind + 1), plt.imshow(train_X[train_ind])) for ind, train_ind in enum
erate(plot_inds)]
# we need to specify color mapping to draw image correctly
plt.figure()
[(
    plt.subplot(130 + ind + 1),
    plt.imshow(train_X[train_ind], cmap=plt.get_cmap('gray'))
  ) for ind, train_ind in enumerate(plot_inds)]
# as an example, 14th row of an input
display(train_X[433, 13])
print(f'Max value: {train_X.max()}, Min value: {train_X.min()}')
# let's see what value Y takes for this X:
print(f'Y: {train_Y[433]}')
```

```
array([  0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 248, 253, 253,
       253, 253, 253, 253, 152,   9,   0,   0,   0,   0,   0,   0,   0,
         0,   0], dtype=uint8)

Max value: 255, Min value: 0
Y: 3
```





As we can see, inputs take values between 255 and 0, each corresponding to the level of whiteness in a pixel, and Y values are just the numbers, so we should divide X by 255 and convert Y to one-hot encoding.

In [145]:

```
train_X_scaled = train_X / 255
test_X_scaled = test_X / 255
train_Y01 = pandas.get_dummies(train_Y).to_numpy()
test_Y01 = pandas.get_dummies(test_Y).to_numpy()
```

Let's start by building a simple neural network:

In [146]:

```
model = Sequential()
# single layer, binary output corresponding to the one-hot encoding
model.add(Dense(train_Y01.shape[1], input_dim=numpy.prod(train_X.shape[1:]),
                activation='sigmoid'))
# we can include metrics to keep track of while training the model
model.compile(optimizer='adam', loss='binary_crossentropy',
              metrics=['accuracy'])
# training
model.fit(
    train_X_scaled.reshape(train_X.shape[0], numpy.prod(train_X.shape[1:])),
    train_Y01, epochs=5, batch_size=10)
```

```
Epoch 1/5
6000/6000 [==============================] - 11s 2ms/step - loss: 0.0959 - accuracy: 0.87
72
Epoch 2/5
6000/6000 [==============================] - 9s 1ms/step - loss: 0.0715 - accuracy: 0.909
4
Epoch 3/5
6000/6000 [==============================] - 9s 1ms/step - loss: 0.0680 - accuracy: 0.915
0
Epoch 4/5
6000/6000 [==============================] - 9s 1ms/step - loss: 0.0661 - accuracy: 0.917
9
Epoch 5/5
6000/6000 [==============================] - 9s 1ms/step - loss: 0.0650 - accuracy: 0.919
2
```

Out[146]:

```
<keras.callbacks.History at 0x7f75f1995110>
```

Making predictions on test set using the trained model:

In [147]:

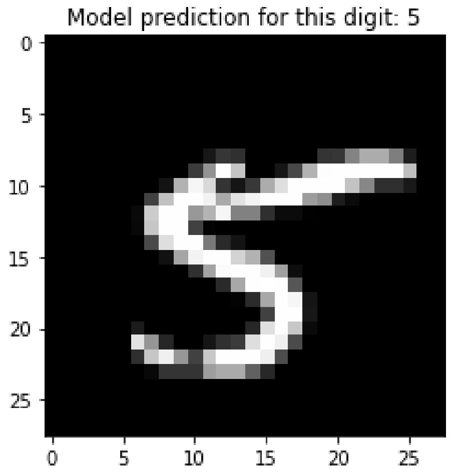
```
# accuracy over the test set
loss, accuracy = model.evaluate(
    test_X_scaled.reshape(test_X.shape[0], numpy.prod(train_X.shape[1:])),
    test_Y01)
print(f'Accuracy: {accuracy*100:.2f}%')
# predicting a single row:
test_row_id = 433
actual_value = test_Y[test_row_id]
X_values = test_X_scaled[test_row_id]
predicted_outcome = model.predict(
    X_values.reshape(1, numpy.prod(train_X.shape[1:]))).argmax()
# plotting
```

```python
plt.figure()
plt.title(f'Model prediction for this digit: {predicted_outcome}')
plt.imshow(X_values, cmap=plt.get_cmap('gray'))
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.0639 - accuracy: 0.9212
Accuracy: 92.12%
```

Out[147]:

```
<matplotlib.image.AxesImage at 0x7f75ea961590>
```



Accuracy isn't too good for this type of task, so let's add more layers, increase the number of iterations and see what happens:

In [148]:

```python
model = Sequential()
model.add(Dense(50, input_dim=numpy.prod(train_X.shape[1:]),
                activation='sigmoid'))
model.add(Dense(50, activation='relu'))
model.add(Dense(50, activation='sigmoid'))
model.add(Dense(50, activation='relu'))
model.add(Dense(50, activation='sigmoid'))
model.add(Dense(50, activation='relu'))
model.add(Dense(50, activation='sigmoid'))
# last layer
model.add(Dense(train_Y01.shape[1], activation='sigmoid'))
# we can include metrics to keep track of while training the model
model.compile(optimizer='adam', loss='binary_crossentropy',
              metrics=['accuracy'])
# training
model.fit(
    train_X_scaled.reshape(train_X.shape[0], numpy.prod(train_X.shape[1:])),
    train_Y01, epochs=30, batch_size=20)
```

```
Epoch 1/30
3000/3000 [==============================] - 8s 2ms/step - loss: 0.2465 - accuracy: 0.339
8
Epoch 2/30
3000/3000 [==============================] - 7s 2ms/step - loss: 0.1045 - accuracy: 0.807
7
Epoch 3/30
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0536 - accuracy: 0.922
6
Epoch 4/30
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0421 - accuracy: 0.937
6
Epoch 5/30
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0345 - accuracy: 0.948
8
Epoch 6/30
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0301 - accuracy: 0.954
5
Epoch 7/30
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0267 - accuracy: 0.959
```

```
6
Epoch 8/30
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0242 - accuracy: 0.963
4
Epoch 9/30
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0218 - accuracy: 0.966
6
Epoch 10/30
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0202 - accuracy: 0.969
3
Epoch 11/30
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0186 - accuracy: 0.971
2
Epoch 12/30
3000/3000 [==============================] - 8s 3ms/step - loss: 0.0170 - accuracy: 0.973
6
Epoch 13/30
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0156 - accuracy: 0.976
2
Epoch 14/30
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0150 - accuracy: 0.977
0
Epoch 15/30
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0137 - accuracy: 0.979
3
Epoch 16/30
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0133 - accuracy: 0.979
1
Epoch 17/30
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0123 - accuracy: 0.980
9
Epoch 18/30
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0119 - accuracy: 0.982
2
Epoch 19/30
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0112 - accuracy: 0.982
6
Epoch 20/30
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0103 - accuracy: 0.984
5
Epoch 21/30
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0099 - accuracy: 0.985
1
Epoch 22/30
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0094 - accuracy: 0.986
1
Epoch 23/30
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0094 - accuracy: 0.985
8
Epoch 24/30
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0088 - accuracy: 0.986
3
Epoch 25/30
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0085 - accuracy: 0.987
1
Epoch 26/30
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0081 - accuracy: 0.987
5
Epoch 27/30
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0075 - accuracy: 0.988
7
Epoch 28/30
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0077 - accuracy: 0.988
1
Epoch 29/30
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0073 - accuracy: 0.988
9
Epoch 30/30
3000/3000 [==============================] - 7s 2ms/step - loss: 0.0068 - accuracy: 0.989
8
```

Out[148]:

```
<keras.callbacks.History at 0x7f75ea8de250>
```

Notice how accuracy improves at a slower rate initially, compared to the simple network. Calculating the accuracy on test set:

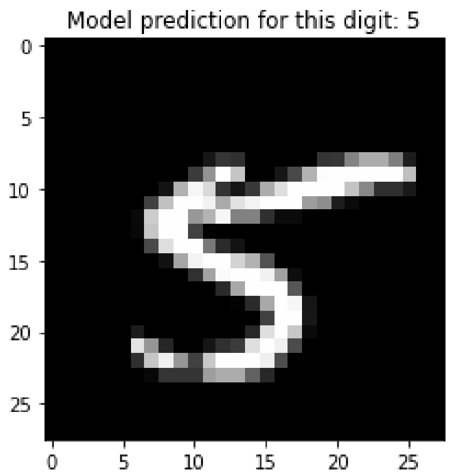
In [149]:

```
# accuracy over the test set
loss, accuracy = model.evaluate(
    test_X_scaled.reshape(test_X.shape[0], numpy.prod(train_X.shape[1:])),
    test_Y01)
print(f'Accuracy: {accuracy*100:.2f}%')
# predicting a single row:
test_row_id = 433
actual_value = test_Y[test_row_id]
X_values = test_X_scaled[test_row_id]
predicted_outcome = model.predict(
    X_values.reshape(1, numpy.prod(train_X.shape[1:]))).argmax()
# plotting
plt.figure()
plt.title(f'Model prediction for this digit: {predicted_outcome}')
plt.imshow(X_values, cmap=plt.get_cmap('gray'))
```

```
313/313 [==============================] - 1s 1ms/step - loss: 0.0255 - accuracy: 0.9675
Accuracy: 96.75%
```

Out[149]:

```
<matplotlib.image.AxesImage at 0x7f75ea6ff810>
```



We can see that accuracy has improved! Now let's train a neural network on the titanic dataset.

In [150]:

```
from sklearn.model_selection import train_test_split

titanic_df = pandas.read_csv('titanic_train.csv')
columns_to_drop = ['PassengerId', 'Name', 'Ticket', 'Fare', 'Cabin']
titanic_df = titanic_df.drop(columns=columns_to_drop).dropna()
titanic_df.head()
# X and y values:
X_, y = titanic_df.drop(columns=['Survived']), titanic_df['Survived'].to_numpy()
X = pandas.get_dummies(X_).to_numpy()
train_X, test_X, train_Y01, test_Y01 = train_test_split(X, y, test_size=0.33,
                                                         random_state=433)
# scaling X
max_values = train_X.max(axis=0)
train_X_scaled = train_X / max_values
test_X_scaled = test_X / max_values
```

Using a simple network:

In [151]:

```
model = Sequential()
model.add(Dense(1, input_dim=numpy.prod(train_X.shape[1:]),
                activation='sigmoid'))
# we can include metrics to keep track of while training the model
model.compile(optimizer='adam', loss='binary_crossentropy',
              metrics=['accuracy'])
# training
model.fit(
    train_X_scaled.reshape(train_X.shape[0], numpy.prod(train_X.shape[1:])),
    train_Y01, epochs=30, batch_size=20)
```

```
Epoch 1/30
24/24 [==============================] - 0s 1ms/step - loss: 0.7588 - accuracy: 0.2998
Epoch 2/30
24/24 [==============================] - 0s 1ms/step - loss: 0.7462 - accuracy: 0.3564
Epoch 3/30
24/24 [==============================] - 0s 1ms/step - loss: 0.7349 - accuracy: 0.4633
Epoch 4/30
24/24 [==============================] - 0s 2ms/step - loss: 0.7245 - accuracy: 0.5367
Epoch 5/30
24/24 [==============================] - 0s 1ms/step - loss: 0.7149 - accuracy: 0.5891
Epoch 6/30
24/24 [==============================] - 0s 1ms/step - loss: 0.7061 - accuracy: 0.5996
Epoch 7/30
24/24 [==============================] - 0s 2ms/step - loss: 0.6979 - accuracy: 0.5996
Epoch 8/30
24/24 [==============================] - 0s 1ms/step - loss: 0.6900 - accuracy: 0.6017
Epoch 9/30
24/24 [==============================] - 0s 1ms/step - loss: 0.6829 - accuracy: 0.6017
Epoch 10/30
24/24 [==============================] - 0s 1ms/step - loss: 0.6759 - accuracy: 0.5975
Epoch 11/30
24/24 [==============================] - 0s 1ms/step - loss: 0.6691 - accuracy: 0.5954
Epoch 12/30
24/24 [==============================] - 0s 2ms/step - loss: 0.6628 - accuracy: 0.5996
Epoch 13/30
24/24 [==============================] - 0s 2ms/step - loss: 0.6566 - accuracy: 0.6017
Epoch 14/30
24/24 [==============================] - 0s 1ms/step - loss: 0.6509 - accuracy: 0.6080
Epoch 15/30
24/24 [==============================] - 0s 2ms/step - loss: 0.6450 - accuracy: 0.6101
Epoch 16/30
24/24 [==============================] - 0s 2ms/step - loss: 0.6396 - accuracy: 0.6122
Epoch 17/30
24/24 [==============================] - 0s 1ms/step - loss: 0.6342 - accuracy: 0.6143
Epoch 18/30
24/24 [==============================] - 0s 1ms/step - loss: 0.6292 - accuracy: 0.6205
Epoch 19/30
24/24 [==============================] - 0s 1ms/step - loss: 0.6241 - accuracy: 0.6205
Epoch 20/30
24/24 [==============================] - 0s 2ms/step - loss: 0.6192 - accuracy: 0.6226
Epoch 21/30
24/24 [==============================] - 0s 2ms/step - loss: 0.6145 - accuracy: 0.6268
Epoch 22/30
24/24 [==============================] - 0s 1ms/step - loss: 0.6099 - accuracy: 0.6268
Epoch 23/30
24/24 [==============================] - 0s 1ms/step - loss: 0.6055 - accuracy: 0.6331
Epoch 24/30
24/24 [==============================] - 0s 1ms/step - loss: 0.6011 - accuracy: 0.6478
Epoch 25/30
24/24 [==============================] - 0s 2ms/step - loss: 0.5970 - accuracy: 0.6709
Epoch 26/30
24/24 [==============================] - 0s 1ms/step - loss: 0.5930 - accuracy: 0.6876
Epoch 27/30
24/24 [==============================] - 0s 1ms/step - loss: 0.5890 - accuracy: 0.6876
Epoch 28/30
24/24 [==============================] - 0s 1ms/step - loss: 0.5853 - accuracy: 0.6960
Epoch 29/30
24/24 [==============================] - 0s 1ms/step - loss: 0.5814 - accuracy: 0.6981
Epoch 30/30
24/24 [==============================] - 0s 1ms/step - loss: 0.5780 - accuracy: 0.7065
```

Out[151]:

```
<keras.callbacks.History at 0x7f75ea01bbd0>
```

Accuracy on test:

In [152]:

```python
# accuracy over the test set
loss, accuracy = model.evaluate(
    test_X_scaled.reshape(test_X.shape[0], numpy.prod(train_X.shape[1:])),
    test_Y01)
print(f'Accuracy: {accuracy*100:.2f}%')
# predicting a single row:
test_row_id = 200
actual_value = test_Y[test_row_id]
X_values = test_X_scaled[test_row_id]
predicted_outcome = model.predict(
    X_values.reshape(1, numpy.prod(train_X.shape[1:])))[0, 0]
print(
    f'Row: {test_row_id}, '
    f'Actual: {"Survived" if test_Y[test_row_id] else "Did not survive"}'
    f', Predicted: {"Survived" if predicted_outcome > 0.5 else "Did not survive"}')
# predictions over the whole test set:
X_values = test_X_scaled
actual_values = test_Y
data = [
        ('Yes' if survived else 'No',
         'Yes' if pred > 0.5 else 'No')
        for pred, survived in zip(
            model.predict(X_values.reshape(
                X_values.shape[0], numpy.prod(train_X.shape[1:])))[:, 0],
            actual_values
            )
        ]
results_df = pandas.DataFrame(data, columns=['Actual', 'Predicted'])
results_df
```

```
8/8 [==============================] - 0s 2ms/step - loss: 0.5876 - accuracy: 0.7064
Accuracy: 70.64%
Row: 200, Actual: Survived, Predicted: Did not survive
```

Out[152]:

	Actual	Predicted
0	Yes	No
1	Yes	No
2	Yes	Yes
3	No	No
4	Yes	No
...	...	...
230	Yes	No
231	Yes	No
232	Yes	Yes
233	Yes	Yes
234	Yes	No

235 rows × 2 columns