# Part III

# Machine Learning

# Chapter 11

# Machine Learning

The last years have seen an incredible growth in interest in machine learning. The field is revolutionizing all aspects of our lives: health care, transport, security, entertainment. Obviously, it is also changing dramatically the fields of network and telecommunications.

This chapter provides a very small overview of one of the most important machine learning tools: neural networks and deep learning. You should not expect to have a comprehensive understanding of the field but we hope that it gives you an insight of the main ideas and sparks your interest in studying it a bit more.

## 11.1 Machine Learning techniques

Machine learning follows a completely different approach than traditional programming. The driver for this is that for many problems (character recognition, autonomic driving, etc) it has proven impossible to come up with algorithms that solve the problem. Humans are just not that smart.

In ML we use **data** to **train** a system to **learn**. We then than test the system with new/other data and hopefully the system knows how to handle the data. This is illustrated in Figure 11.1

Machine learning is seen as a field of Artificial Intelligence which is a wider field. However, in today's world when people mention A.I. they generally mean Machine Learning (ML). See ML as the AI that really works.

Machine learning can be divided in the following sub-fields

- Supervised Learning: In a supervised learning model, the algorithm learns on a labeled dataset, providing an answer key that the algorithm can use to evaluate its accuracy on training data. The most used example of SL is Neural Networks (Deep Learning is just neural networks with lots of layers)

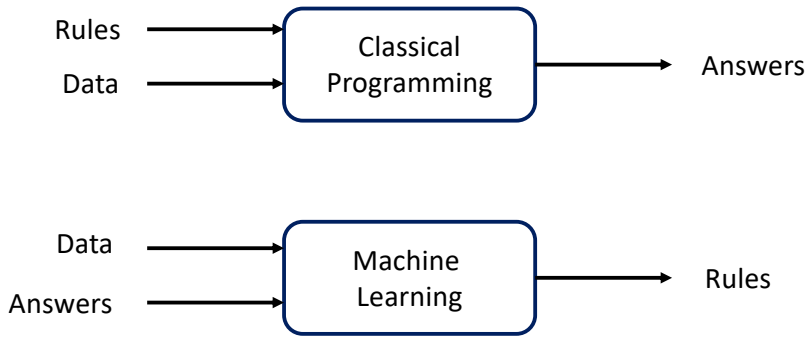


Figure 11.1: Machine Learning vs Traditional Programming

- Unsupervised Learning: in contrast, provides unlabeled data that the algorithm tries to make sense of by extracting features and patterns on its own.
- Reinforcement Learning: is an area of machine learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward.

## 11.2 Neural Networks

Artificial Neural Networks (or Neural Networks (NN) for short) is a field of machine learning with already an "old history". The intellectual roots are in the 1940s but the first breakthroughs were done in the 1970s. The field stayed relatively under the radar until the deep learning breakthroughs of the beginning of the 21st century.

### 11.2.1 The MNIST data set

Let us illustrate the use of NNs with a famous example. The MNIST dataset. One problem that is incredibly difficult (if not impossible) to solve with traditional programming is hand-written recognition. This is a clear example where Machine learning can be very advantageous. The MNIST dataset http://yann.lecun.com/exdb/mnist/ is illustrated in Figure 11.2.

A black box illustration of a neural network is illustrated in Figure 11.3. The data including a character (in this case, an array of 28x28 pixels with a grey value in each element) is inputted in the system which then identifies it as an 8.

In slightly more detail the system will output an array of values, ideally with a 1 for the digit 8 and 0 for the others. This can be seen in Figure 11.4.

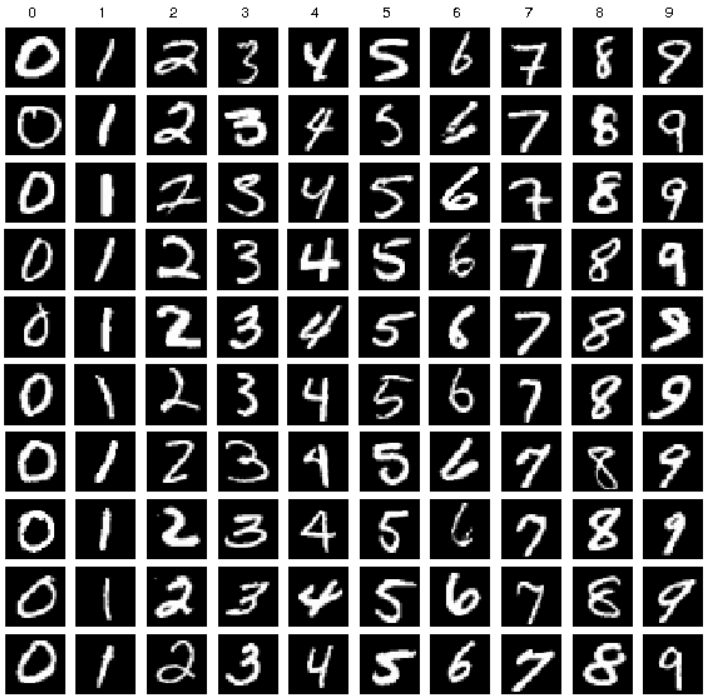


Figure 11.2: The MNIST dataset

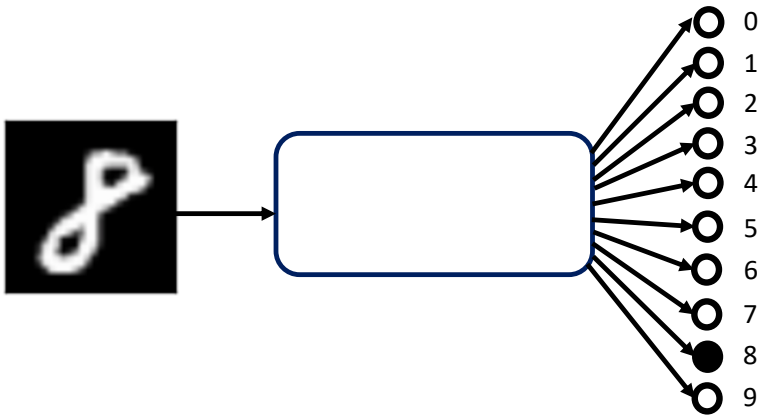


Figure 11.3: Neural Network black box

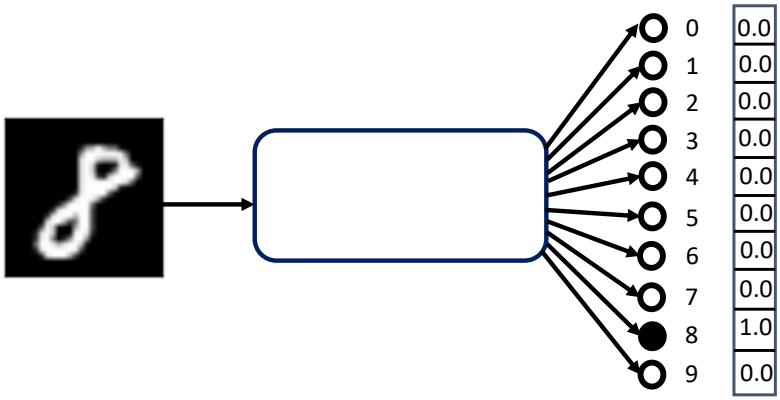


Figure 11.4: Neural Network structure

Let us see now, how a neural network is organized. One can see this in Figure 11.5. A Neural network is made up of **neurons** represented by the circles interconnected. They are organized in several layers. The first layer of neurons will have the inputs. The last layer will have the outputs. All joint layers are fully connected (this is not illustrated for clarity).

Let us now see what happens for each neuron. This is illustrated in Figure **??**.

- Each link will have an associated weight with it (w1, w2, etc)
- The value of the neuron will consist of the sum of products of the input times the weight as illustrated in the formula in the image
- The total sum is then passed through a function that transforms the value between 0 and 1. There are several alternatives for this.
- This applies to every subsequent layer until we have the final values for the output.

### 11.2.2 Backpropagation

Now the big question is: how do we calculate the weights. This is what the **backpropagation** algorithm does. In a nutshell, the algorithm takes labeled data which in this case is a picture of a 8 with the label 8. It then inputs the data and adjusts the weights in order to make the output as close as possible to 8 (this is called minimizing the loss function as illustrated in Figure **??**). This is repeated many many times (this is the training or learning). When it is finished we should have weights that identify pictures you haven't used in the training part. The system has learned !

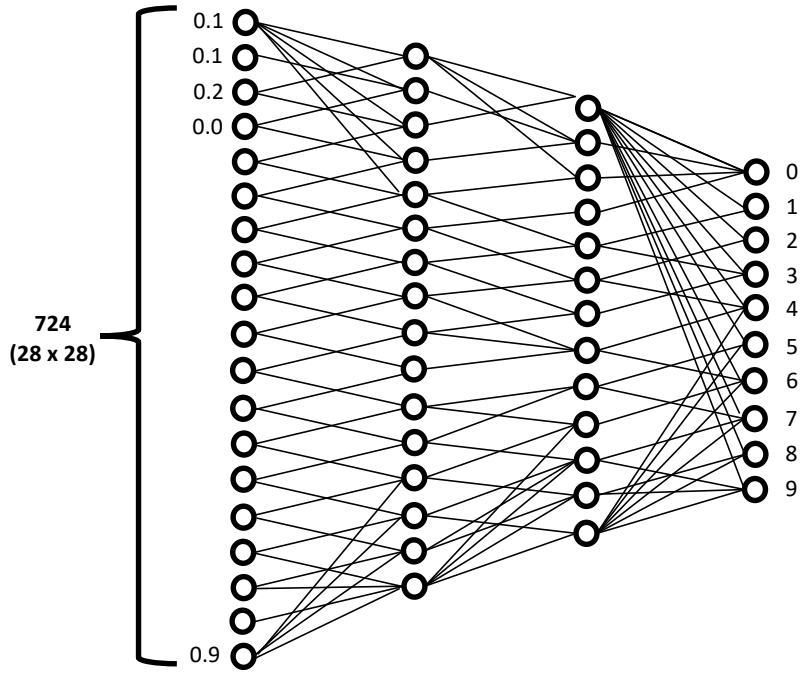


Figure 11.5: Neural Network

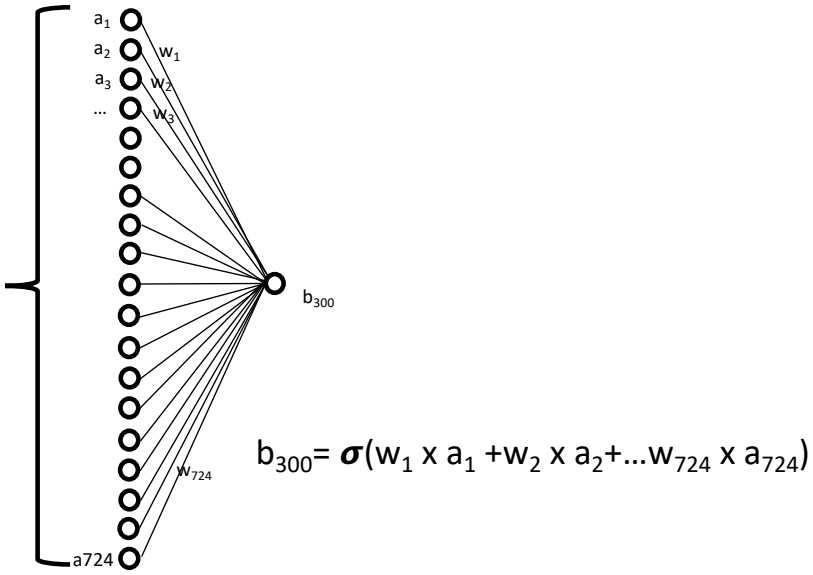


Figure 11.6: fig:NN4

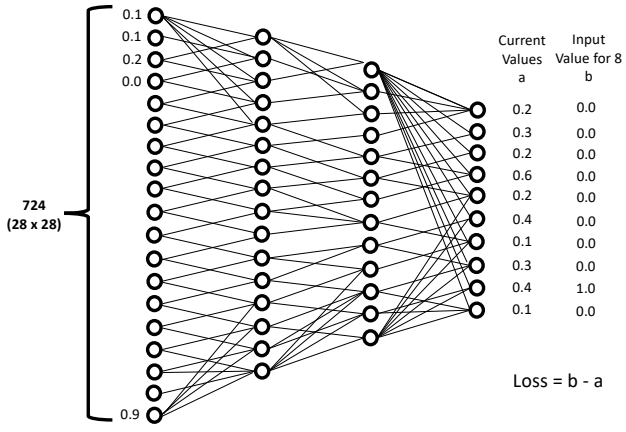


Figure 11.7: fig:NN5

## 11.3 Using Neural Networks in Keras

Until recently, to implement a neural networks was very cumbersome. Fortunately we have the Keras library. Figure **??** shows the code that does this.

### Exercises

1. Run the above code. Change some of the parameters and observe the changes
2. Use the UK home broadband data to predict what the speed of a house will be. https://data.gov.uk/dataset/dfe843da-06ca-4680-9ba0-fbb27319e402/uk-home-broadband-performance

```
1  import keras
2  from keras.datasets import mnist
3  from keras.models import Sequential
4  from keras.layers import Dense, Dropout
5  from keras.optimizers import RMSprop
6
7  # The batch size is the number of samples
8  batch_size = 128
9  # There are 10 possible digits
10 num_classes = 10
11 # Epochs is the number of times the training set is used
12 epochs = 20
13
14 # the data, split between train and test sets
15 (x_train, y_train), (x_test, y_test) = mnist.load_data()
16
17
18 # Each image in the MNIST dataset has 28*28 = 74 pixels.
19 # We reshape this 28x28 matrix into a 784 array
20 x_train = x_train.reshape(60000, 784)
21 x_test = x_test.reshape(10000, 784)
22 x_train = x_train.astype('float32')
23 x_test = x_test.astype('float32')
24
25 # The RGB values are between 0 and 255,
26 # and we want to input values between 0 and 1.
27 x_train /= 255
28 x_test /= 255
29 print(x_train.shape[0], 'train samples')
30 print(x_test.shape[0], 'test samples')
31
32 # convert class vectors to binary class matrices
33 # e.g instead of "8" we want [0,0,0,0,0,0,0,0,1,0]
34 y_train = keras.utils.to_categorical(y_train, num_classes)
35 y_test = keras.utils.to_categorical(y_test, num_classes)
36
37 # We initialize an empty Sequential model
38 model = Sequential()
39
40 # And then sequentially add new layers.
41 # A Dense layer is the one we covered this chapter,
42 # where a neuron connects to all the neurons in the,
43 # following layer.
44 # For each layer, we have to specify the activation,
45 # function and the output size. In the first layer,
46 # we also have to specify the input shape.
47 model.add(Dense(512, activation='relu', input_shape=(784,)))
48
49 # Dropout is a regularization technique (to prevent) overfitting
50 model.add(Dropout(0.2))
51 model.add(Dense(512, activation='relu'))
52 model.add(Dropout(0.2))
```

```
50 model.add(Dense(num_classes, activation='softmax'))
51
52 model.summary()
53 # Once the neural network structure is set we compile it.
54 # That means associate a loss function and an optimzer with it.
55 model.compile(loss='categorical_crossentropy',
56              optimizer=RMSprop(),
57              metrics=['accuracy'])
58
59 # After the network is compiled we can train it, using our
60 # training set.
61 history = model.fit(x_train, y_train,
62                     batch_size=batch_size,
63                     epochs=epochs,
64                     verbose=1,
65                     validation_data=(x_test, y_test))
66
67 #Finally, we check the performance of the model
68 # in the test set
69 score = model.evaluate(x_test, y_test, verbose=0)
70 print('Test loss:', score[0])
71 print('Test accuracy:', score[1])
72
```

Figure 11.8: MNIST in Keras