Neural network classification

We are going to make a simple neural network which is able to classify to which of two clumps of data a data point belongs. We are not going to tell our network anything about the two clumps of data directly. We are going to generate two different clumps of data and show these to our network. The network will adjust the weights and bias of its neuron to 'learn' to classify the data points into the two groups. Once we have trained the network using the training data, we will try it out on some testing data to see if it can classify data it hasn't been trained on.

To start, let's import some commands we are going to need for this example.

```
import matplotlib.pyplot as plt
from math import exp
from random import random
```

Neuron

Now we will set up our neuron. First we define the activation function.

```
def activation_fn(w1,w2,b,x1,x2):
    return 1/(1+exp(-(x1 * w1 + x2 * w2 + b)))
```

We start with weights and bias randomised between -1 and 1.

```
w1 = random()*2-1
w2 = random()*2-1
b = random()*2-1
```

We also define η.

```
eta = 0.1
```

Training our network

We train our network using training data. These are data for which we know the expected output, so we can adjust our weights and biases in line with this.

Download the training data and save it in the same location as a new Python file.

First, let's have a look at what the data looks like. If you open the file (probably your computer will decide to import it into Microsoft Excel), you should see data like this:

```
3.4836027414296655,4.581374480301093,0
8.515179655197741,9.443408833084046,1
9.078970547884293,8.706353901036625,1
8.64784019022494,8.902076574540423,1
2.082886027283916,1.6121719135849424,0
```

Here the first two numbers are the inputs and the third is the expected output. The data are in two groups, group `0` and group `1`.

We can load the data from the file into our script using `file()`. Since this is Comma Separated Values (CSV) data from a file, we first `rstrip` the newline character off the end of the line and then `split` it into `parts` wherever we find a comma. Finally we convert the data from the file (automatically a string) into floats (i.e. decimal numbers) and store the input data in `data` and the label (which group it belongs to) in `labels`.

```
f = open("training-data.csv", "r")

data = []
labels = []

for line in f:
    line = line.rstrip('\r\n')
    parts = line.split(",")
    data.append((float(parts[0]),float(parts[1])))
    labels.append(float(parts[2]))

f.close()
```

We can plot our data to have a look at what it looks like. Depending whether the label is `0` or `1`, we load the data into lists representing x_1 and x_2 input data for the appropriate group. This is so we can plot the two groups as different colours.

```
group0_x1 = []
group0_x2 = []
group1_x1 = []
group1_x2 = []

for i in range(len(data)):
    if labels[i] == 0: # group 0
        group0_x1.append(data[i][0])
        group0_x2.append(data[i][1])
    else: # group 1
        group1_x1.append(data[i][0])
        group1_x2.append(data[i][1])

plt.scatter(group0_x1,group0_x2,c="blue",marker="+",label="Group 0")
```

```
plt.scatter(group1_x1,group1_x2,c="purple",marker="+",label="Group 1")
plt.xlabel("x1")
plt.ylabel("x2")
plt.title("Training data")
plt.legend()
plt.show()
```

You should be looking at a plot with two distinct sets of points.

We can run the same training data through our network multiple times, called *epochs*. The more epochs we run through, the closer our model will fit our training data. A closer fit may not be a good thing, because a model can get to the point where it can only classify the training data and the point is to train it to recognise real data!

Here we run our data through two epochs using a loop. Within our loop, we run each item of the training data through our neuron's activation function, then adjust the weights and bias according to the result. Recall each item in `data` contains three elements, corresponding to the two inputs and the expected output (which group the data point belongs to).

```
print(f"w1={w1}, w2={w2}, b={b}")
for i in range(0,2):
    print(f"Epoch {i}")

    for i in range(len(data)):
        output = activation_fn(w1,w2,b,data[i][0],data[i][1])

        w1 = w1 + eta*(labels[i]-output)*output*(1-output)*data[i][0]
        w2 = w2 + eta*(labels[i]-output)*output*(1-output)*data[i][1]
        b = b + eta*(labels[i]-output)*output*(1-output)

    print(f"w1={w1}, w2={w2}, b={b}")
```

Testing our network

Now we are ready to use our network to analyse some testing data. Testing data is also data where we know what the output should be, but crucially the neural network has never seen it before. We use this to test whether our model is making sensible predictions, in this case whether it can tell us which group the data belongs to. In this case, we will generate some data points and ask the network to classify these as either part of group 0 or part of group 1, then inspect visually to see whether it has done a good job.

Here we just generate data by moving randomly over the range (0,0) to (10,10).

```
testing_data = []
for i in range(0,10**4):
    testing_data.append((random()*10+1,random()*10+1))
```

Now we ask our network to analyse these data. For each pair of coordinates, we calculate the output using the activation function. If the output is < 0.5, the network is saying this is part of group 0; otherwise it is part of group 1.

```
group0_x1 = []
group0_x2 = []
group1_x1 = []
group1_x2 = []

for item in testing_data:
    output = activation_fn(w1,w2,b,item[0],item[1])

    if output < 0.5:
        group0_x1.append(item[0])
        group0_x2.append(item[1])
    else:
        group1_x1.append(item[0])
        group1_x2.append(item[1])
```

To see whether this has worked, we can plot these groups.

```
plt.scatter(group0_x1,group0_x2,c="blue",marker="+",label="Group 0")
plt.scatter(group1_x1,group1_x2,c="purple",marker="+",label="Group 1")
plt.xlabel("x1")
plt.ylabel("x2")
plt.title("Testing data")
plt.legend()
plt.show()
```

Hopefully you will see that our simple network with one artificial neuron has successfully learned to classify data into our two groups. You may notice that the network has made a classification for some data points that are very far from the original clumps of data. Whether this is good or not depends on our circumstances, and is in some ways analogous to whether a model can be applied beyond its range of validity.

Perhaps you would like to explore what other classification tasks you can perform using a neural network. I'd be interested to see what you come up with.