Level 5: AI Game Programming

Week 4

Lab 4

RECORD ALL THAT YOU DO IN YOUR LAB LOGBOOK (HARDCOPY 'OR' WORD FILE).

It is strongly recommended that you implement your code in Google Colab using Python. You need to use Python and Colab in your assignment. (See Lab Week 2 for the introduction to Colab.)

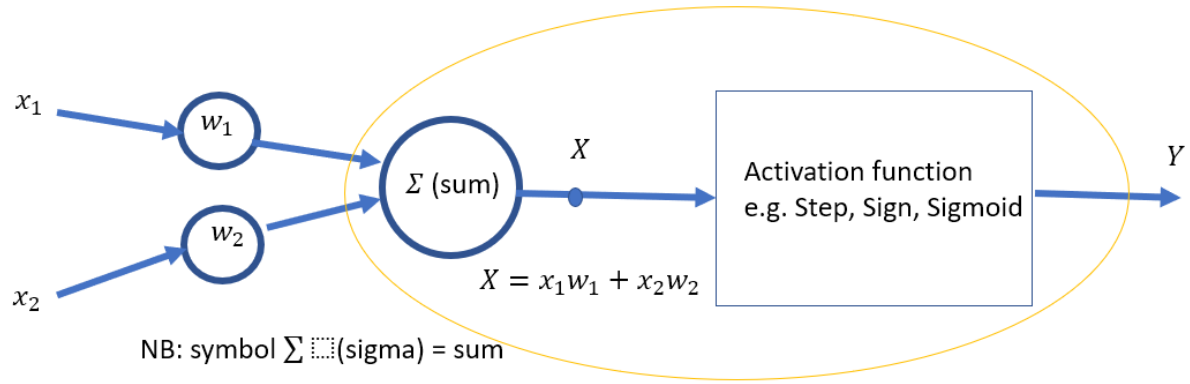
Training a single neuron (perceptron) to learn a simple task using the error

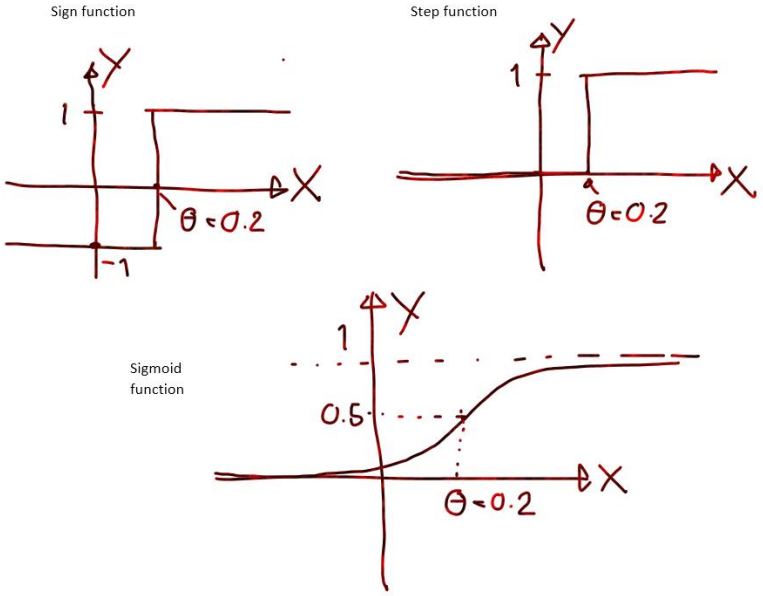
In the lecture last week, we learned the method to train a single neuron to do a decision task by using the error at the output of the neuron.

You already implemented the simple training algorithm in the lab last week. The puzzle occurred as most of your implementation won't give you the correct results ($w_1 = 0.1, w_2 = 0.1$). Since then, your brain has had some time to digest the idea. Today we will revisit the code again with some additional information. Can you figure out what goes wrong?

RECAP

From the lecture last week, as an example, a neuron called 'perceptron' (single neuron with two inputs and one output) is shown below:





$$X = \sum_{i=1}^{2} x_i w_i = x_1 w_1 + x_1 w_2$$

In this case, the output Y is defined by **Step function**

$$Y = \begin{cases} 1 \; if \; X \geq \theta \\ 0 \; if \; X \; < \theta \end{cases}$$

Where θ is the threshold (equal to 0.2 in this example).

Example: two input signals and x1 = 0.2, x2 = 0.1, w1 = 1.0, w2 = 0.2, θ = 0.2

X = x1*w1 +x2*w2 = 0.22

X >= θ, therefore Y (output) = 1

TRAINING

In this lab, again, we will train our single neuron to learn to handle a game creature's decision-making process – that is, whether the creature will attack or flee, depending on the creature's power and enemy's power.

Input		Output
Creature's power (x_1)	Enemy's power (x_2)	Creature's decision/action (Y)
Weak (0)	Weak (0)	Flee (0)
Weak (0)	Strong (1)	Flee (0)
Strong (1)	Weak (0)	Flee (0) NB: the creature has high moral standards and does not want to attack a weaker enemy.
Strong (1)	Strong (1)	Attack (1)

To train or to make your neuron learn to make decision is to **adjust the weights** to correct values so that your neuron can produce correct output or decision when given a set of inputs. In this example, that means finding the fixed correct values for w_1 and w_2 which produce the correct outputs for 'all' combinations of inputs.

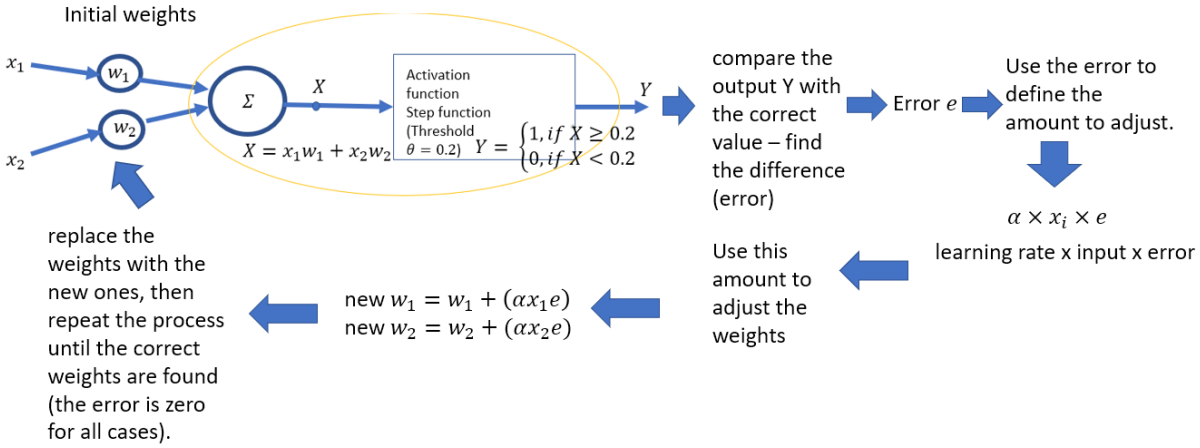
Note that our neuron model cannot accept 'weak', 'strong', 'flee' or 'attack'. Everything must be 'encoded' or represented in a numerical form. Therefore, as shown in the table, weak = 0, strong = 1, flee = 0 and attack = 1.

TASKS

Task 1: training one single neuron using the method explained in the lecture

We will have a look at the training method described in the lecture. This is the 'Mysterious gift from heaven' mentioned in the last lab.

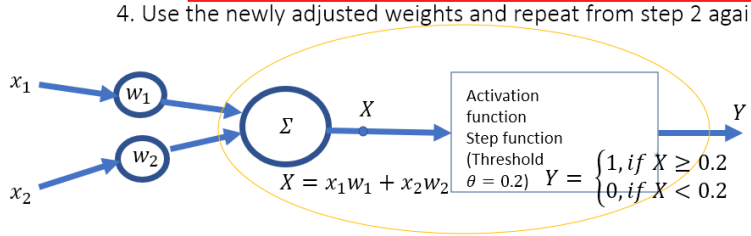
We use the error (the difference between the correct output Y and the output produced by your neuron when using the current weights) to guide the changes of the weight's values to move closer to the correct values.



Or, in a more formal presentation:

The main concept:

1. start with random weights.
2. produce outputs and compare to the desired (correct) outputs – the difference between the produced output and the desired outputs is call an error. $e = Y - Y_{current}$
3. Use the error value to incrementally adjust the weights
   new weight = current weight + (learning rate x input x error). $w_{new} = w_{current} + (\alpha \times x \times e)$
4. Use the newly adjusted weights and repeat from step 2 again until the correct weights are found.



INPUT		OUTPUT
Creature's power (x1)	Enemy's power (x2)	Creature's decision/action (Y)
Weak (0)	Weak (0)	Flee (0)
Weak (0)	Strong (1)	Flee (0)
Strong (1)	Weak (0)	Flee (0) (the creature has high moral standards and does not want to attack a weaker enemy.
Strong (1)	Strong (1)	Attack (1)

Example pseudocode from last week

Note that an ***epoch*** means training your neuron with all the training data (4 cases in this example) for one cycle of training. This is a parameter you have to set. You increase the number of epoch if the training does not convert to produce correct results.

For simplicity, let the step function threshold θ = 0.2 and set the learning rate α = 0.1.

In practice, you would initialise the weights by using random numbers. But it's a nightmare to debug. Therefore, you can manually set the initial values first. (In this case, w1 = 0.3, w2 = -0.1.) Once your code works properly, then you can randomly initialise the weight values.

```
NEpoch = 5; % Number of epoch
w1 = 0.3; w2 = -0.1; % Set initial set initial weights.
% Normally they are random numbers, but for simplicity
% in this experiment we set to those values.

Theta = 0.2; % Activation function threshold
alpha = 0.1; % Learning rate

% Training data - the decision table we want the neuron to learn
% with the correct output Y
x1(1) = 0; x2(1) = 0; Y(1) = 0;
x1(2) = 0; x2(2) = 1; Y(2) = 0;
x1(3) = 1; x2(3) = 0; Y(3) = 0;
x1(4) = 1; x2(4) = 1; Y(4) = 1;

for Epochno = 1:NEpoch % Epoch number
    for i = 1:4 % Each case of the four cases

        X = x1(i)*w1 + x2(i)*w2 % Sum the weighted inputs

        % Activation function
        if X >= Theta
            Ycurrent = 1
        else
            Ycurrent = 0
        end

        e = Y(i)-Ycurrent % calculate the error

        w1 = w1 + alpha*x1(i)*e % update weight w1
        w2 = w2 + alpha*x2(i)*e % update weight w2
    end
end
```

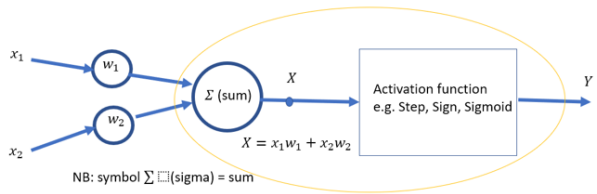
If your code works correctly, the final correct weights should be $w_1 = 0.1, w_2 = 0.1$.

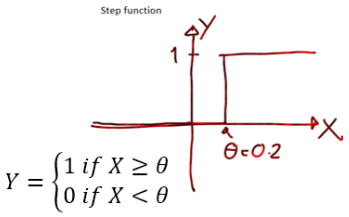
Hint: If your program does not produce the correct weights, for each step, check the results against the following table and adjust your code accordingly. This is a good way to practice implementing and debugging your code. When you find the error, think about what is the best way to solve this problem.

Epoch	Case number	Inputs		Correct output	Current weights		Current output (from the current weights)	Error (correct output minus current output)	New updated weights in this iteration (to be used as the current weights in the next iteration)	
		x_1	x_2	Y	w_1	w_2	$Y_{current}$	$e = Y - Y_{current}$	w_1	w_2
1	1	0	0	0	0.3	-0.1	0	0	0.3	-0.1
	2	0	1	0	0.3	-0.1	0	0	0.3	-0.1
	3	1	0	0	0.3	-0.1	1	-1	0.2	-0.1
	4	1	1	1	0.2	-0.1	0	1	0.3	0.0
2	1	0	0	0	0.3	0.0	0	0	0.3	0.0
	2	0	1	0	0.3	0.0	0	0	0.3	0.0
	3	1	0	0	0.3	0.0	1	-1	0.2	0.0
	4	1	1	1	0.2	0.0	1	0	0.2	0.0
3	1	0	0	0	0.2	0.0	0	0	0.2	0.0
	2	0	1	0	0.2	0.0	0	0	0.2	0.0
	3	1	0	0	0.2	0.0	1	-1	0.1	0.0
	4	1	1	1	0.1	0.0	0	1	0.2	0.1
4	1	0	0	0	0.2	0.1	0	0	0.2	0.1
	2	0	1	0	0.2	0.1	0	0	0.2	0.1
	3	1	0	0	0.2	0.1	1	-1	0.1	0.1
	4	1	1	1	0.1	0.1	1	0	0.1	0.1
5	1	0	0	0	0.1	0.1	0	0	0.1	0.1
	2	0	1	0	0.1	0.1	0	0	0.1	0.1
	3	1	0	0	0.1	0.1	0	0	0.1	0.1
	4	1	1	1	0.1	0.1	1	0	0.1	0.1

You could see that the error for 'all cases' in Epoch 5 are equal to zero when $w_1 = 0.1$ *and* $w_2 = 0.1$, there for they are correct weights.

You could also manually double check if the weights are really correct:





$w_1 = 0.1, \quad w_2 = 0.1$

Case 1

$X = 0 \times 0.1 + 0 \times 0.1 = 0$

therefore $Y = 0$

Case 2

$X = 0 \times 0.1 + 1 \times 0.1 = 0.1$

therefore $Y = 0$

Case 3

$X = 1 \times 0.1 + 0 \times 0.1 = 0.1$

therefore $Y = 0$

Case 4

$X = 1 \times 0.1 + 1 \times 0.1 = 0.2$

therefore $Y = 1$

Input		Output
Creature's power (x_1)	Enemy's power (x_2)	Creature's decision/action (Y)
Weak (0)	Weak (0)	Flee (0)
Weak (0)	Strong (1)	Flee (0)
Strong (1)	Weak (0)	Flee (0) NB: the creature has high moral standards and does not want to attack a weaker enemy.
Strong (1)	Strong (1)	Attack (1)

Task 2: Training parameters and activation functions

There are many factors and many choices in designing and training your neural model. In this experiment, you will be asked to find out what ranges of parameters and what activation functions work for this particular case.

Task 2.1: Range of the number of epoch

Find the minimum number of epochs. Is 5 (in the example code) the minimum number of epoch for this training?

Task 2.2: Range of learning rate α

From the previous task, the learning rate α was set to 0.1. Systematically conduct an experiment to find out the 'range' of α that works for this case.

Task 2.3: Activation functions and thresholds

So far, we have used only the Step function with threshold θ = 0.2. What about other activation functions and thresholds?

Recap: activation functions

$$Y^{step} = \begin{cases} +1 \; if \; X \geq \theta \\ 0 \; if \; X < \theta \end{cases}$$

$$Y^{sign} = \begin{cases} +1 \; if \; X \geq \theta \\ -1 \; if \; X < \theta \end{cases}$$

$$Y^{Sigmoid} = \frac{1}{1 + e^{-(X-\theta)}}$$

$$Y^{liniear} = X - \theta$$

a) Step function

Systematically conduct an experiment to find out the range of theta θ that works or does not work, in this case. Note that you need to also consider learning rate α.

b) Sign function

Systematically conduct an experiment to find out the range of theta θ that works or does not work, in this case. Note that you need to also consider learning rate α.

c) Sigmoid function

Systematically conduct an experiment to find out the range of theta θ that works or does not work, in this case. Note that you need to also consider learning rate α.

d) Linear function

Systematically conduct an experiment to find out the range of theta θ that works, or does not work, in this case. Note that you need to also consider learning rate α.

Task 3: Different inputs/outputs table (dataset) for training

Train your neuron with a different decision-making table for your NPC:

INPUT		OUTPUT
Creature's power	Enemy's power	Creature's decision/action
Strong (1)	Strong (1)	Attack (0)
Weak (0)	Strong (1)	Flee (1)
Strong (1)	Weak (0)	Flee (1)
Weak (0)	Weak (0)	Attack (0)

You need to change the training data part of the example code to:

```
x1(1) = 0; x2(1) = 0; Yd(1) = 0;
x1(2) = 0; x2(2) = 1; Yd(2) = 1;
x1(3) = 1; x2(3) = 0; Yd(3) = 1;
x1(4) = 1; x2(4) = 1; Yd(4) = 0;
```

Can you train your neuron to make the decision stated in the table? If not, why?

END