Exercise 11

Machine Learning I

11A-1.

Let us first look at the data:

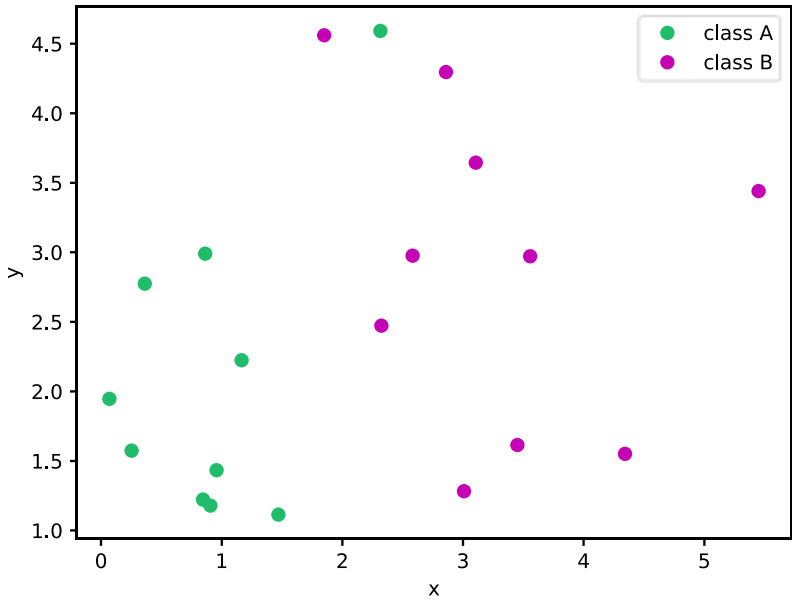

Figure 1 A sample of the data. The values are slightly interlapping and not completely linearly separable.

As we can see, the data is slightly interlapping. Nevertheless, this does not look entangled enough that I would consider neural networks to solve this problem. Logistic regression, as we have seen in previous exercises, seems more than up to this task (Question: Do you think it is possible to separate data that is interlapping through the use of basis functions?).

But let us start with the Neural Network. In the task, we want to have the following layout:

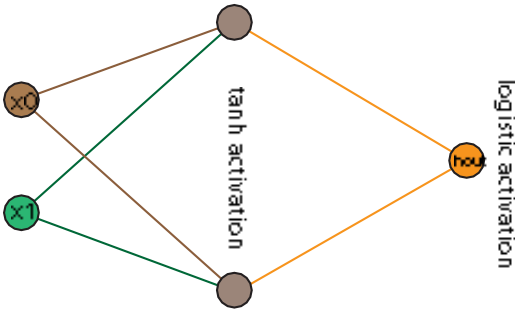

Figure 2 Layout of the Neural Network as given in the task.

For backpropagation to work, we will need the derivatives of our activation functions. The logistic one is obvious, as we found out its derivative in one of the previous exercises.

Let $\sigma(z) = \frac{1}{1+e^{-z}}$. Then:

$$\frac{d\sigma}{dz} = \sigma(1-\sigma)$$

The $\tanh(z)$function is a little more involved. Looking up its derivative gives us:

$$\frac{d}{dz}\tanh(z) = 1 - \tanh(z)^2$$

Lastly, we will have to pick a loss function. It appears that we are free to choose, that's why we will pick our standard squared loss:

$$Loss_i[\mathbf{x}_i] = \frac{1}{2}\big(t_i - f(\mathbf{x}_i)\big)^2$$

where f is the function defined by our neural network. For gradient descent to work, we will have to calculate $\nabla E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}\big(t_i - f(\mathbf{x}_i)\big)^2$ with respect to all the weights.
Because our total error function is constructed as a sum, we can also iteratively evaluate each term sequentially, i.e.

$$\begin{aligned}\nabla E(\mathbf{w}) &= \nabla\left[\frac{1}{2}\sum_{i=1}^{n}\big(t_i - f(\mathbf{x}_i)\big)^2\right] \\ &= \nabla Loss_1 + \nabla Loss_2 + \cdots + \nabla Loss_n\end{aligned}$$

For computational reasons, we don't always use all n terms in the sum but a strictly smaller subset. These $k < n$ terms are called minibatch. How you choose your minibatch is entirely up to you. Let w_{ij} be the weight connecting node i to node j.

In earlier exercises (for example in logistic regression), we could find a closed form solution in matrix form. That approach only worked, because all dimensions of $\nabla E(\mathbf{w})$ had similar terms (in fact the dimensions were not only similar but equivalent if we neglect the differences in term indices). However, because now each dimension of behaves differently it makes sense to look at each component $\nabla E(\mathbf{w})_k$ separately.

As calculation of $\frac{\partial E}{\partial w_{ij}}$ directly proves difficult because of the activation functions nested nature, the trick is to instead utilize the chain rule. Let $z_j = \sum_{i=1}^{k} w_{ij}\sigma_i + w_{0j}$, where σ_i now is an arbitrary differentiable activation function. We then have:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial \sigma_j}\frac{\partial \sigma_j}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}}$$

If you start from the output neuron h_{out} and iteratively apply this rule to all other neurons until you arrive at the input node, the above product can be calculated for all weights in the network. To see why this is the case, follow this link:

https://en.wikipedia.org/wiki/Backpropagation

If we follow the rules on that page, we will be able to calculate the following partial derivatives:

[This will be added later.]

11A-2.

Just plug in the identity $h(a) = a$ and calculate.

$$
\begin{aligned}
y_k &= \sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right) \\
&= \sum_{j=1}^{M} w_{kj}^{(2)} \sum_{i=0}^{D} w_{ji}^{(1)} x_i \\
&= \sum_{j=1}^{M} \sum_{i=0}^{D} w_{kj}^{(2)} w_{ji}^{(1)} x_i \\
&= \sum_{i=1}^{D} \sum_{j=0}^{M} w_{kj}^{(2)} w_{ji}^{(1)} x_i \\
&= \sum_{i=0}^{D} x_i \underbrace{\sum_{j=1}^{M} w_{kj}^{(2)} w_{ji}^{(1)}}_{\beta_i} \\
&= \sum_{i=0}^{D} x_i \beta_i \\
&= \sum_{i=1}^{D} x_i \beta_i + \beta_0
\end{aligned}
$$

11A-3.

Even though it is written in the task, it always helps to write a truth table:

X_0	X_1	$X_0 \oplus X_1$
-1	-1	0
-1	1	1
1	-1	1
1	1	0

In addition, let us define the weights as follows:

$$w_{1,1}^{(1)} = 1, \quad w_{1,2}^{(1)} = 1, \quad w_{1,0}^{(1)} = -1 \; w_{2,1}^{(1)} = -1, \quad w_{2,2}^{(1)} = -1, \quad w_{2,0}^{(1)} = -1$$

$$w_{1,1}^{(2)} = -2, \quad w_{2,1}^{(2)} = -2, \quad w_{2,0}^{(12)} = 1$$

Let h_1, h_2 be two hidden layer output functions and h_{out} be the last layer output function:

$$h_1 = RELU(1x_1 + 1x_2 - 1)$$

$$h_2 = RELU(-1x_1 - 1x_2 - 1)$$

$$h_{out} = -2h_1 - 2h_2 + 1$$

Drawn, the layout looks like this:

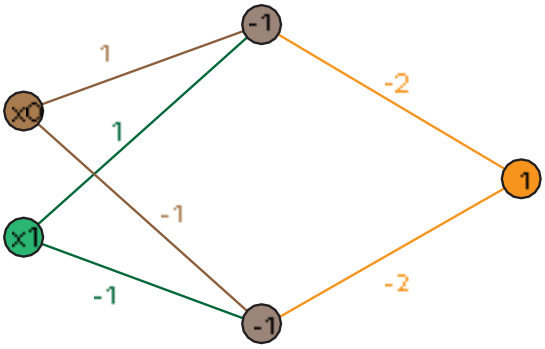

Figure 3 XOR Function realized with the wanted network. The values in the nodes represent the bias term $w_{i,0}^{l}$ of each node.

Inputting the values into f we notice:

$$f(-1,-1) = -2[RELU(-1-1-1)] - 2[RELU(1+1-1)] + 1 = -1$$

$$f(-1,1) = -2[RELU(-1+1-1)] - 2[RELU(1-1-1)] + 1 = 1$$

$$f(1,-1) = -2[RELU(1-1-1)] - 2[RELU(-1+1-1)] + 1 = 1$$

$$f(1,1) = -2[RELU(1 + 1 - 1)] - 2[RELU(-1 - 1 - 1)] + 1 = -1$$

Why does it work?

The two interior nodes act as AND gates, that output *true* if $x_0 = x_1$. We then negate their input and receive the corresponding output.

If $x_0 \neq x_1$, both AND gates are shut. Because the last bias $w^2_{1,0} = 1$ is constantly true, we will output *true* if both inputs are different.