MSc in Data Science - Neural Computing
Model Answers for Selected Exercises
Artur Garcez

EXERCISE 1

(a) f_3 = nand, f_2 = or and f_3 = and will do (all three are linearly separable). For a proof you can, eg, verify that (p_1 nand p_2) and (p_1 or p_2) = p_1 xor p_2.

(b)$(w_{g_1i_1}, w_{g_1i_2}, w_{g_2i_1}, w_{g_2i_2}, w_{g_2g_1}, w_{g_21}, w_{g_11}) = (1, 1, 1, 1, -2, -0.5, -1.5)$, eg. The hidden neuron computes and. Without the weight $w_{g_2g_1}$ the output would compute or, but with $w_{g_2g_1} = -2$ the output changes in the case when both inputs are on, resulting in xor.

EXERCISE 3

(a) The line equation: $i_2 = m.i_1 + b$ where m=-2/3 and b=0.5 given the points (0,0.5) and (0.75,0).

We need to match $i_2 = m.i_1 + b$ with $w_1.i_1 + w_2.i_2 + \theta$

Therefore, a perceptron with inputs i1 and i2 and weights 2/3 and 1, respectively, and bias -0.5 would solve the classification problem exactly.

(b) See Figure 1.

(c) The trained perceptron implements straight line: 0.51 . i1 + 0.03 . i2 -0.2 = 0. This can be seen as an approximation of the intended classification in Q3(a) obtained by learning from the examples provided.

Comparing the above line with the original line (Figure 2), this tells us that in the unit square (as the problem was defined originally), in area A, there are false negatives, in area B, there are false positives. In the rest of the unit square the perceptrons agree. So, area A+B as a percentage of the unit square gives the generalization error.

The error is given by:

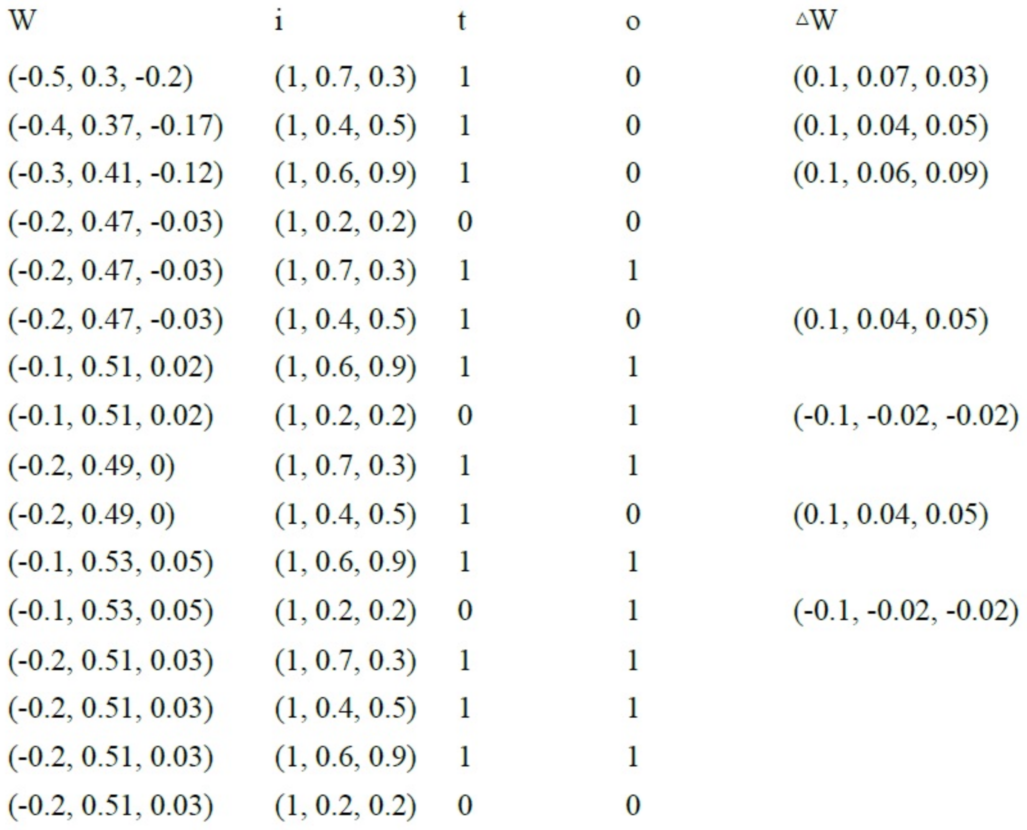
area A = (6.66 x 0.392)/2 - (5.66 x 0.333)/2 - (0.377 x 0.252)/2 - (0.248 x 0.015)/2 - (0.248 x 0.377) = 0.219

area B = (0.373 x 0.248)/2 - (0.248 x 0.015)/2 = 0.044

Generalisation error = area A + area B = 0.263 = 26%

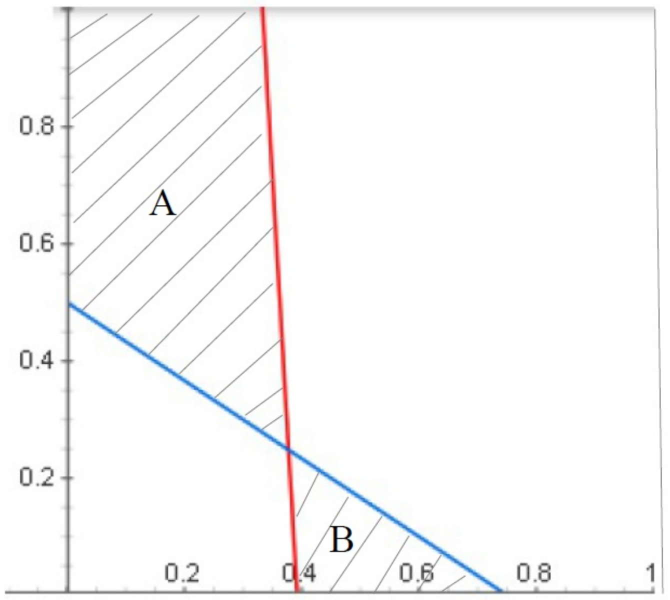
Notice that 6.66 is the point where the red line in Fig. 2 crosses the y axis.

Figure 1: Perceptron training algorithm



W	i	t	o	△W
(-0.5, 0.3, -0.2)	(1, 0.7, 0.3)	1	0	(0.1, 0.07, 0.03)
(-0.4, 0.37, -0.17)	(1, 0.4, 0.5)	1	0	(0.1, 0.04, 0.05)
(-0.3, 0.41, -0.12)	(1, 0.6, 0.9)	1	0	(0.1, 0.06, 0.09)
(-0.2, 0.47, -0.03)	(1, 0.2, 0.2)	0	0	
(-0.2, 0.47, -0.03)	(1, 0.7, 0.3)	1	1	
(-0.2, 0.47, -0.03)	(1, 0.4, 0.5)	1	0	(0.1, 0.04, 0.05)
(-0.1, 0.51, 0.02)	(1, 0.6, 0.9)	1	1	
(-0.1, 0.51, 0.02)	(1, 0.2, 0.2)	0	1	(-0.1, -0.02, -0.02)
(-0.2, 0.49, 0)	(1, 0.7, 0.3)	1	1	
(-0.2, 0.49, 0)	(1, 0.4, 0.5)	1	0	(0.1, 0.04, 0.05)
(-0.1, 0.53, 0.05)	(1, 0.6, 0.9)	1	1	
(-0.1, 0.53, 0.05)	(1, 0.2, 0.2)	0	1	(-0.1, -0.02, -0.02)
(-0.2, 0.51, 0.03)	(1, 0.7, 0.3)	1	1	
(-0.2, 0.51, 0.03)	(1, 0.4, 0.5)	1	1	
(-0.2, 0.51, 0.03)	(1, 0.6, 0.9)	1	1	
(-0.2, 0.51, 0.03)	(1, 0.2, 0.2)	0	0	

Figure 2: Error calculation



The training set error is the error of the network on the set of examples used for changing the weights; the test set error gives an estimate of the generalisation error.

Normally, the generalization error can't be calculated and has to be estimated using the test set. The test set can never be used for training the network.

EXERCISE 5

(a) Here is how backpropagation with momentum is expressed mathematically:

$$\Delta W^i = -\eta \nabla E_{W^{i-1}} + \alpha \Delta W^{i-1} \tag{1}$$

where $\alpha \Delta W^{i-1}$ is the term of momentum and $0 < \alpha < 1$ is the momentum constant.

The term of momentum allows a network to respond *not only to the local gradient*, but also to *recent trends in the error surface.* Acting like a low pass filter, momentum allows the network to ignore small features in the error surface. Momentum is added to backpropagation learning by making weight changes equal to the sum of a fraction of the last weight change and the new change suggested by the backpropagation rule.

(b) If *standard backprogation* is used, or if the momentum constant is set to 0, the network follows *pure gradient descent* upon training. The result is shown in Figure 3(a). If we set the momentum constant to 0.95 for a high degree of momentum, the network will not be stuck in a shallow minimum as easily. In Figure 3(b), the network rolls down into M1, but shoots up the opposite slope to fall into the deeper minimum. It then rolls back and forth across M2 until it comes to rest. Global minimisation is not guaranteed though (see Figure 4(a)). For the same reason above, due to the term of momentum, the network can get out of the global minimum of the error surface and get stuck in a local minimum. Figure 4(b) illustrates this.

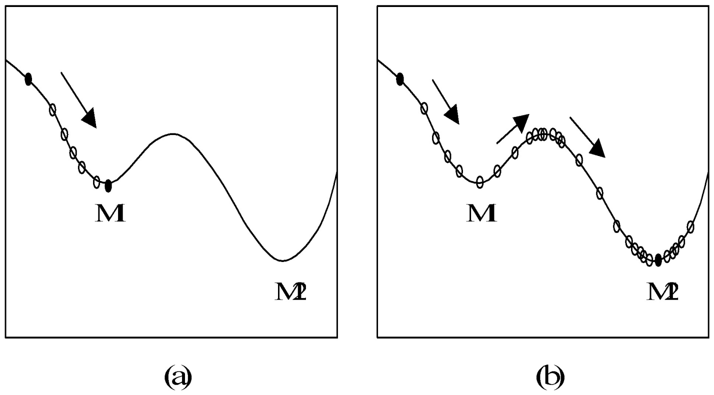


Figure 3: Local minima problem 1

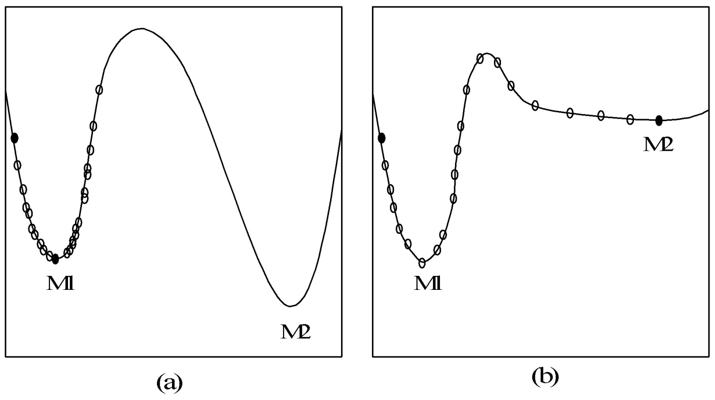


Figure 4: Local minima problem 2

EXERCISE 6

By looking at the two graphs, we realise that the network is not doing what we want. The network is not generating reasonable outputs between the original inputs, i.e., even though the training phase was sucessfull, the network's testing phase is a failure (a smoother function is desired). This is called *overfitting*. If a network appears to be trained fairly easily but test vectors do not generate reasonable results (the network does not generalise), the network probably has too many hidden neurons or a very small training set.

EXERCISE 7

10 training phases, with 900 training examples and 100 validation examples each. Note that the initial set of (random) weights, as well as the choice of the training and test sets, may influence the accuracy obtained in a given application. When cross-validation is used, the accuracy reported is the mean of many training phases, performed by taking different initial sets of weights and training sets. The accuracy obtained is thus more reliable as an approximate measure of the true generalization accuracy (i.e. accuracy on unseen examples). Cross-validation is particularly useful when a small set of examples is available.

Notice that in the pole balancing exercise (exercise 4), Matlab divided the set of examples randomly into a training set (with 70% of the examples), validation set (15%) and test set (15%). Cross-validation was not used. The test set is the measure that should be used for comparing different models.

The validation set can be used either to monitor a model for overfitting e.g. by implementing early stopping, or for model selection, e.g. if considering a choice of a model with 10, 20 or 30 hidden neurons. In addition, one wants the training set accuracy to be high, but not 100% (we assume that noise exists on the set of examples).

If cross-validation was to be used in the pole balancing example, the 15% of examples selected for testing should be taken out of the set of examples at the start. The remaining examples would be used in the cross-validation, e.g. training 10 networks with different sets of hidden neurons to choose the best performing network. At the end, the test set can be used to compare, e.g., the performance of the network chosen with that of another model, such as an SVM.

EXERCISE 11

1. *State Transition Diagram (see Figure 5)*

states	x_1	x_2	x_3
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

2. *Energy Function*

The energy of the network is given by the following equation:

$$E(W,S) = -\frac{1}{2}\sum_{a,b} W_{ab} S_a S_b$$

In this case the energy is:

$$E(W,S) = -\frac{1}{2}(2x_1x_2 + 2x_2x_3 - 4x_1x_3)$$

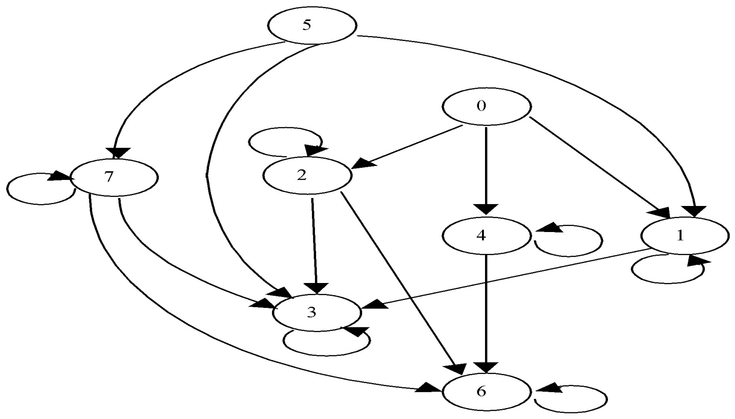


Figure 5: Energy transition diagram

3. *Memories*

   From the state transition diagram, it is clear that the memories are $(0, 1, 1)$ and $(1, 1, 0)$. Using the energy function instead, $(0, 1, 1)$ and $(1, 1, 0)$ should be local minima. In fact, $E(W, (0, 1, 1)) = E(W, (1, 1, 0)) = -1$, and $E(W, S) > -1$ for any state S in the neighbourhood of (0,1,1) and (1,1,0).