1 Week 2

Control and output spaces   The control space for robots is the wheels, and for neural nets the weights and biases. The output space for robots is the movement/travel in the 2D obstacle space, and for neural nets the activation of the output layer.

The cost function   Consider the output space

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \in \mathbb{R}^2.$$

The current weight configuration in the control space

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} \in \mathbb{R}^3$$

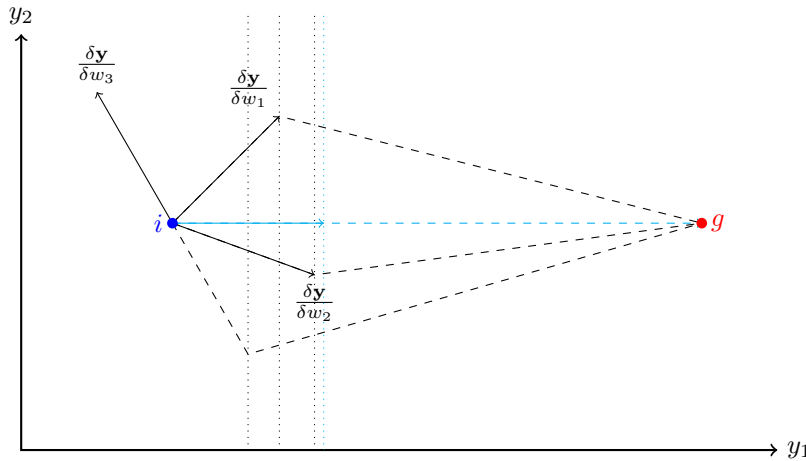produces the initial output $i \in \mathbf{y}$. Our goal is $g \in \mathbf{y}$.

Figure 1: Diagram of the output space.

We define the cost function as

$$C\left(\mathbf{w}\right) = \frac{\text{fractional progress}}{\text{total Euclidean distance}}. \tag{1}$$

The straight line goal-connecting path would minimize the Euclidean distance while maximizing the fractional progress.

2   Week 3

2.1   Research

Simulated annealing   This refers to a process that will temporarily accept sub-optimal (worse) values in hopes of overcoming local maxima, to reach better local minima.

Adam optimizer   The name means "ADAptive Moment estimation". Instead of using a global learning rate (as in SGD), Adam "computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients" [Kingma and Ba, 2014], i.e. the mean and variance.

The stripe problem in literature   To do

2.2   A problem that always achieves a point on the goal line (gradient of weights are at right angle)

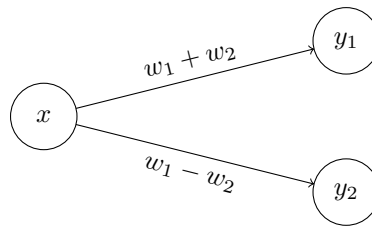Consider the neural network below. Note that it does not have an activation function.

Figure 2: The neural network.

Here we have

$$\mathbf{y}(x; \mathbf{w}) = \begin{bmatrix} w_1 + w_2 \\ w_1 - w_2 \end{bmatrix} x$$

with the derivative

$$\frac{\partial \mathbf{y}}{\partial \mathbf{w}} = \begin{bmatrix} \frac{\partial y_1}{\partial w_1} & \frac{\partial y_1}{\partial w_2} \\ \frac{\partial y_2}{\partial w_1} & \frac{\partial y_2}{\partial w_2} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} x.$$

For $x = 1$, let our goal be $\mathbf{g} = \begin{bmatrix} 10 \\ 2 \end{bmatrix}$ and the initial weight configuration be $\mathbf{w} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$. This leads to the current position $\mathbf{p} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$ in output space.

Say we want to achieve $\frac{1}{6}$ fractional progress, so we aim to achieve the point $\mathbf{q}$ at the intersection of the fractional progress line with the goal line in the diagram of Fig. 3.

Now we need to find out by how much we need to update $\mathbf{w}$. In other words, we need to find $\mathbf{a} \in \mathbb{R}^2$ so that after

$$\mathbf{w} \leftarrow \mathbf{w} + \mathbf{a} \tag{2}$$

we get

$$\mathbf{y}(1; \mathbf{w}) = \begin{bmatrix} 4 \\ 2 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$
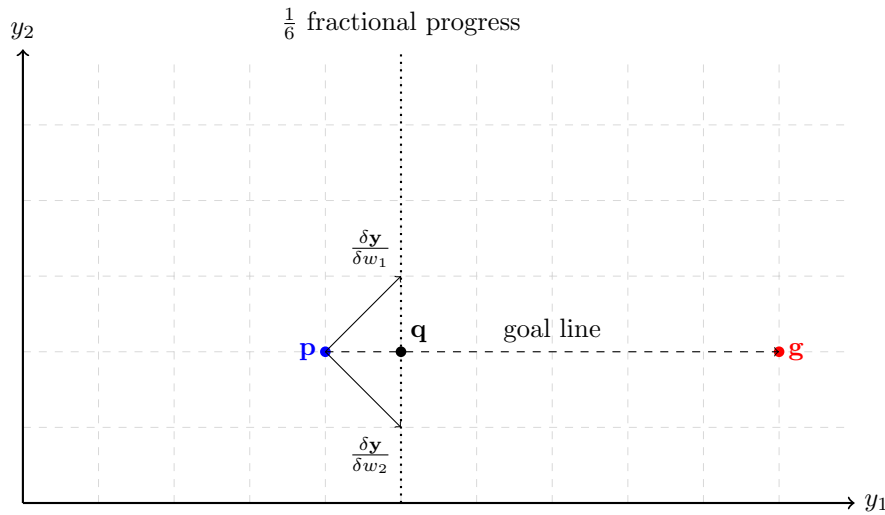
2

Figure 3: Diagram of the weights' partial derivatives in output space along with the sub-goal.

This is equivalent to finding the vector $\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$ that satisfies

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{\partial \mathbf{y}}{\partial \mathbf{w}} \cdot \mathbf{a}$$
$$= a_1 \frac{\partial \mathbf{y}}{\partial w_1} + a_2 \frac{\partial \mathbf{y}}{\partial w_2}$$
$$= a_1 \begin{bmatrix} 1 \\ 1 \end{bmatrix} + a_2 \begin{bmatrix} 1 \\ -1 \end{bmatrix},$$

so

$$a_1 = a_2 = \frac{1}{2}$$
$$\mathbf{a} = \frac{1}{2} \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Using (2), we update $\mathbf{w}$ to

$$\mathbf{w} \leftarrow \begin{bmatrix} 3 \\ 1 \end{bmatrix} + \frac{1}{2} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 7 \\ 3 \end{bmatrix}.$$

Now let's see what we get for the new $\mathbf{q}$ in this updated system:

$$\mathbf{q} = \mathbf{y}(1; \mathbf{w}) = \begin{bmatrix} w_1 + w_2 \\ w_1 - w_2 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 7 + 3 \\ 7 - 3 \end{bmatrix} = \begin{bmatrix} 5 \\ 2 \end{bmatrix}$$

which is correct (see diagram).

Remarks: In this very simple example, we can use linear algebra to deterministically reach our sub-goal. However, as soon as we have three weights, there will be more than one possible vector $\mathbf{a} \in \mathbb{R}^3$ that achieves the sub-goal. Are there any drawbacks for choosing a particular $\mathbf{a}$ over another, if there are multiple options?

Furthermore, this network is so simple that the partial derivatives of the weights with respect to the output space aren't dependent on the weights themselves. However, in a "real" neural network, the gradients will influence each other because they depend on the weights. What happens in that case? (This question is examined in the next section.)
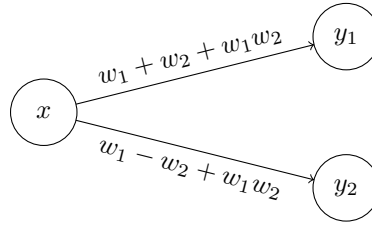
3

Figure 4: The neural network.

2.3   When gradients are dependent on other weights

Consider the neural network below. It uses the ReLU activation function.

Here we have

$$\mathbf{y}(x; \mathbf{w}) = \begin{bmatrix} \max(0, w_1 + w_2 + w_1 w_2) \\ \max(0, w_1 - w_2 + w_1 w_2) \end{bmatrix} x$$

with the derivative

$$\frac{\partial \mathbf{y}}{\partial \mathbf{w}} = \begin{bmatrix} \max(0, w_2 + 1) & \max(0, w_1 + 1) \\ \max(0, w_2 + 1) & \max(0, w_1 - 1) \end{bmatrix} x.$$

For $x = 1$, let our goal be $\mathbf{g} = \begin{bmatrix} 9 \\ 1 \end{bmatrix}$ and the initial weight configuration be $\mathbf{w} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. This leads to the current position $\mathbf{p} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$ in output space.
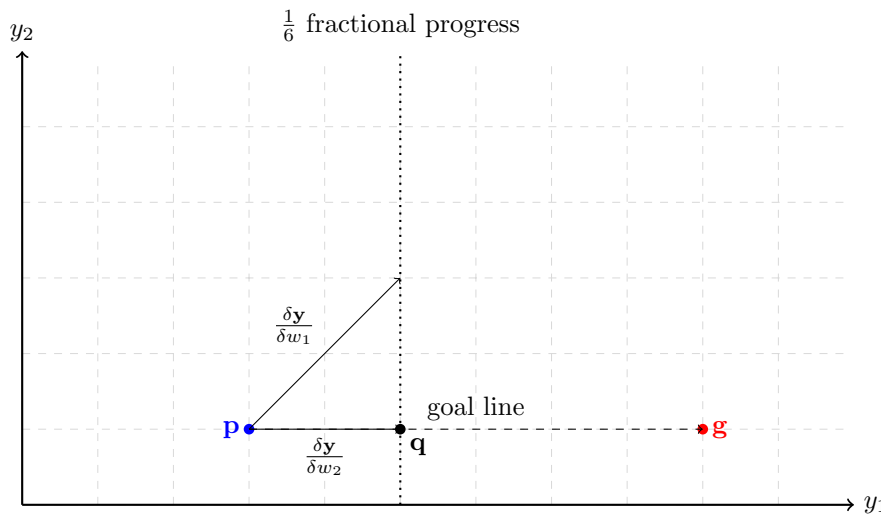

Figure 5: Diagram of the weights' partial derivatives in output space along with the sub-goal.

Just looking at the diagram, one could guess to update the weights as follows to achieve $\mathbf{q}$:

$$\mathbf{w} \leftarrow \mathbf{w} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Using $\mathbf{w} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, we calculate the new output

$$\mathbf{y}(1; \mathbf{w}) = \begin{bmatrix} \max(0, 1 + 2 + 2) \\ \max(0, 1 - 2 + 2) \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \end{bmatrix}$$

which is *not* our sub-goal $\mathbf{p}$! The reason for this is that unlike the previous network, the partial derivative $\frac{\partial \mathbf{y}}{\partial w_1}$ is dependent on the value of w_2 and vice-versa.

4

Remarks: When the partial derivatives are functions of each other (i.e. depend on each other), we can't use simple linear algebra in order to find the combination of the weights to achieve a specific subgoal. This makes intuitive sense, because if it were possible, we could simply choose the goal as our subgoal and expect to use linear algebra to find the optimum weight configuration which would eliminate the need for training neural networks.

2.4   A problem with an unrealizable goal line, but realizable goal

Did not find a good example yet.

3 Week 5

3.1 Research

Radial basis function   A radial basis function is a real-valued function f that satisfies the property $f(\mathbf{x}) = f(\|\mathbf{x}\|)$. RBFs are used as a kernel in SVMs. They are infinitely differentiable.

Two commonly used RBFs are the Gaussian function

$$f(x) = e^{-x^2}$$

and the bump function

$$f(x) = \begin{cases} e^{-\frac{1}{1-x^2}} & \text{for } -1 < x < 1 \\ 0 & \text{otherwise.} \end{cases}$$

3.2 Replicating the simple neural network with the shallow excitation gradient
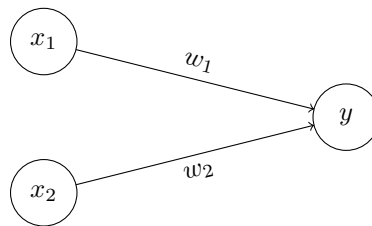
Consider the very simple neural network below.

Figure 6: The neural network.

The output neuron's excitation is given by

$$y(\mathbf{x}; \mathbf{w}, b) = \mathbf{w}\mathbf{x} + b.$$

Consider the two points in input space $\mathbf{a} = \begin{bmatrix} 0.6 \\ 0.4 \end{bmatrix}$ and $\mathbf{b} = \begin{bmatrix} 1 \\ 0.4 \end{bmatrix}$ with the targets $y(\mathbf{a}) = 0.2$ and $y(\mathbf{b}) = 0.8$.

The current weight configuration should be such that the hyperplane in output space of the 0.5 excitation line should be given by $x_1 = 0.5$. Furthermore, the current activations should be $y(\mathbf{a}) = 0.51$ and $y(\mathbf{b}) = 0.55$ such that we get the configuration outlined in Fig. 7.

After some experimentation, it was established that this configuration can be achieved using $\mathbf{w} = \begin{bmatrix} 0.1 \\ 0.0005 \end{bmatrix}$ and $b = 0.45$ thus leading to the function

$$y(\mathbf{x}) = 0.1x_1 + 0.0005x_2 + 0.45.$$

Remarks: To get a gradient this shallow, it was necessary to introduce a third parameter b. We need to investigate whether we can call it a "non-trainable variable", so we effectively only have two weights.
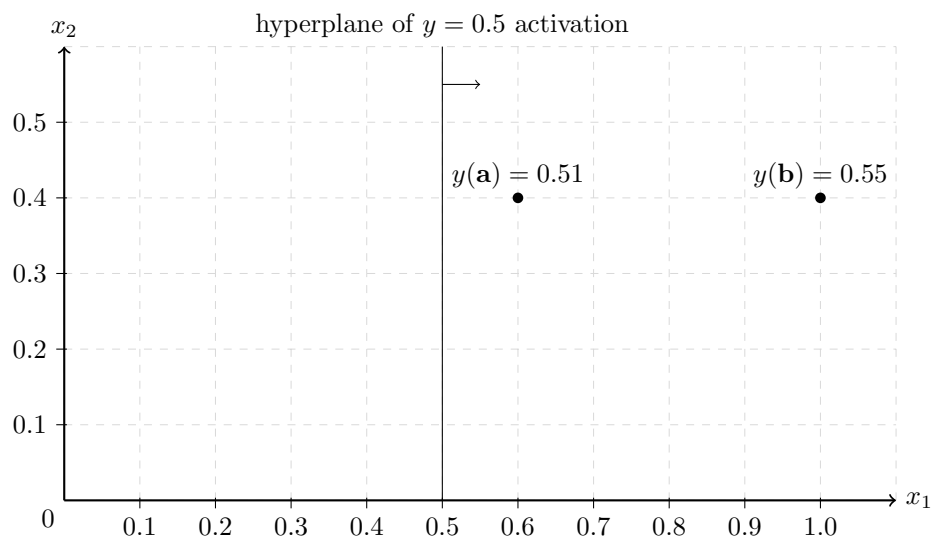
Figure 7: The initial configuration with the hyperplane. The arrow is pointing in the direction of increasing activation.

4   Week 6

4.1   The hyperplanes of radial basis activation functions

Consider the radial basis function
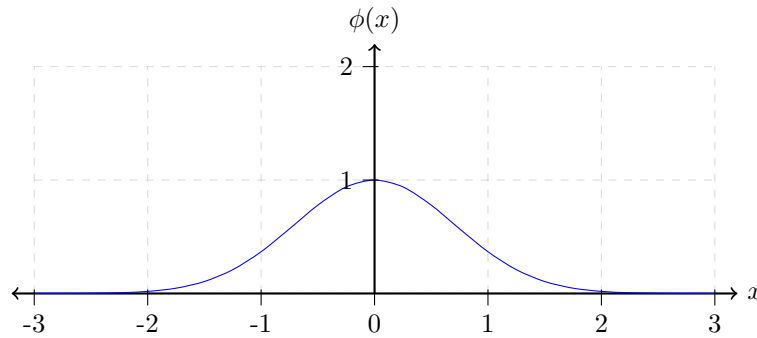$$\phi(x) = e^{-x^2} \tag{3}$$
with the graph pictured in Fig. 8.

Figure 8: Plot of the radial basis activation function.

To examine the hyperplanes corresponding to this activation function, consider the very simple neural network given in Fig. 9.
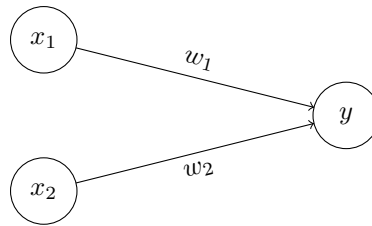

Figure 9: The neural network.

For simplicity, we set $w_1 = w_2 = 1$. The output neuron's activation is then given by

$$y(\mathbf{x}) = \phi(x_1 + x_2).$$

To find the equations of the hyperplanes, we set the output to 0.5, so

$$\frac{1}{2} = \phi(x_1 + x_2)$$
$$= e^{-(x_1 + x_2)^2}$$
$$-\ln\frac{1}{2} = (x_1 + x_2)^2$$
$$\pm\sqrt{-\ln\frac{1}{2}} = x_1 + x_2$$
$$x_2 = -x_1 \pm \sqrt{-\ln\frac{1}{2}}$$

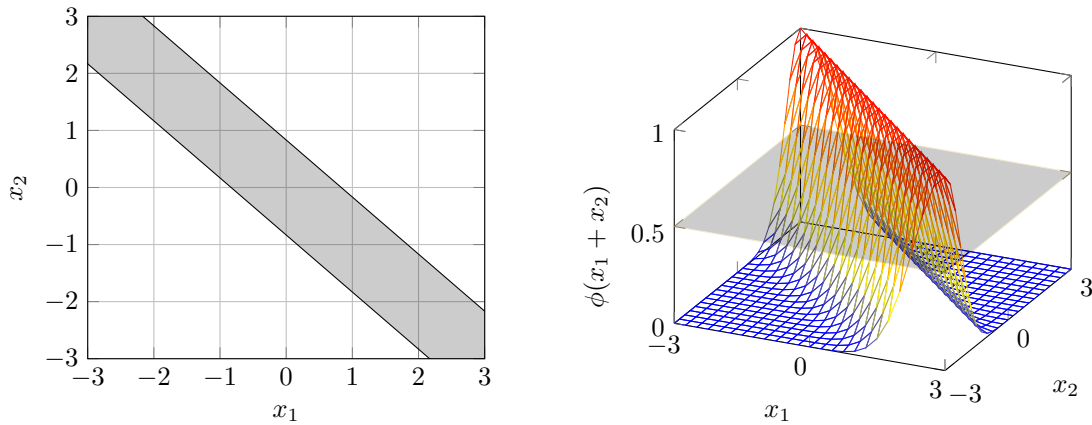which means that there are two lines. They are graphed in Fig. 10.

Figure 10: Plot of hyperplanes in input space of the radial basis activation function.

Remarks: Notice that the radial basis function given in (3) is differentiable and hence a suitable activation function for neural networks. This means that if we use this activation function, we do not require hidden units in order to create a stripe configuration.

4.2 The stripe problem without hidden units

From the findings of the previous section, it follows that we could create the stripe problem with the neural network architecture from Fig. 9 and the hyperplanes that are a result of the radial basis activation function. Fig. 10 shows the hyperplanes for the initial weight configuration $w_1 = w_2 = 1$.

We require four sample points to produce the problem, and they must be arranged in a way where the mean squared error is required to temporarily increase before it decreases. The initial and possible goal configuration are shown in Fig. 11.

The depicted goal configuration can be achieved when either w_1 or w_2 changes to -1 whilst the other weight remains at 1. In gradually changing the weight, some points will necessarily be misclassified.
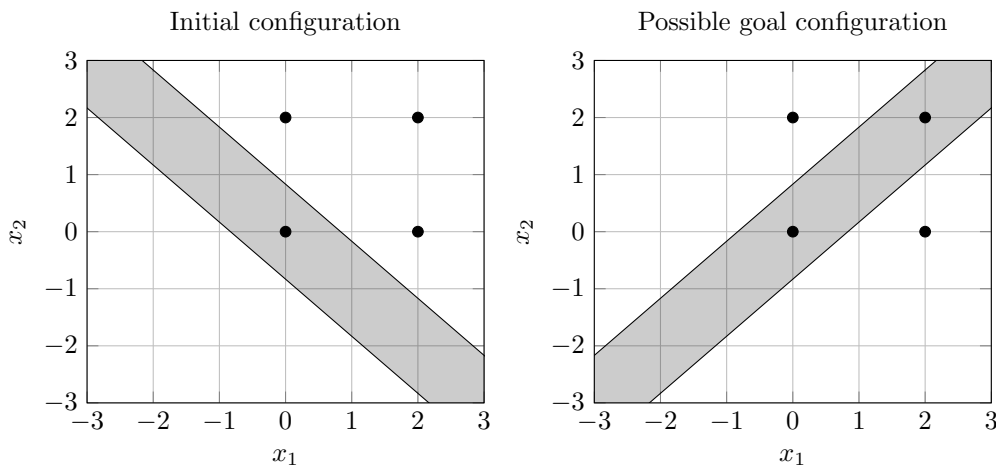

Figure 11: Initial and possible goal configuration for the stripe problem.

This becomes evident when graphing the error-weight surface, as shown in Fig. 12. There is no path from the initial configuration to the global minimum with a strictly decreasing error; the error must temporarily be increased (by overcoming a hill) until reaching the global minimum trough.
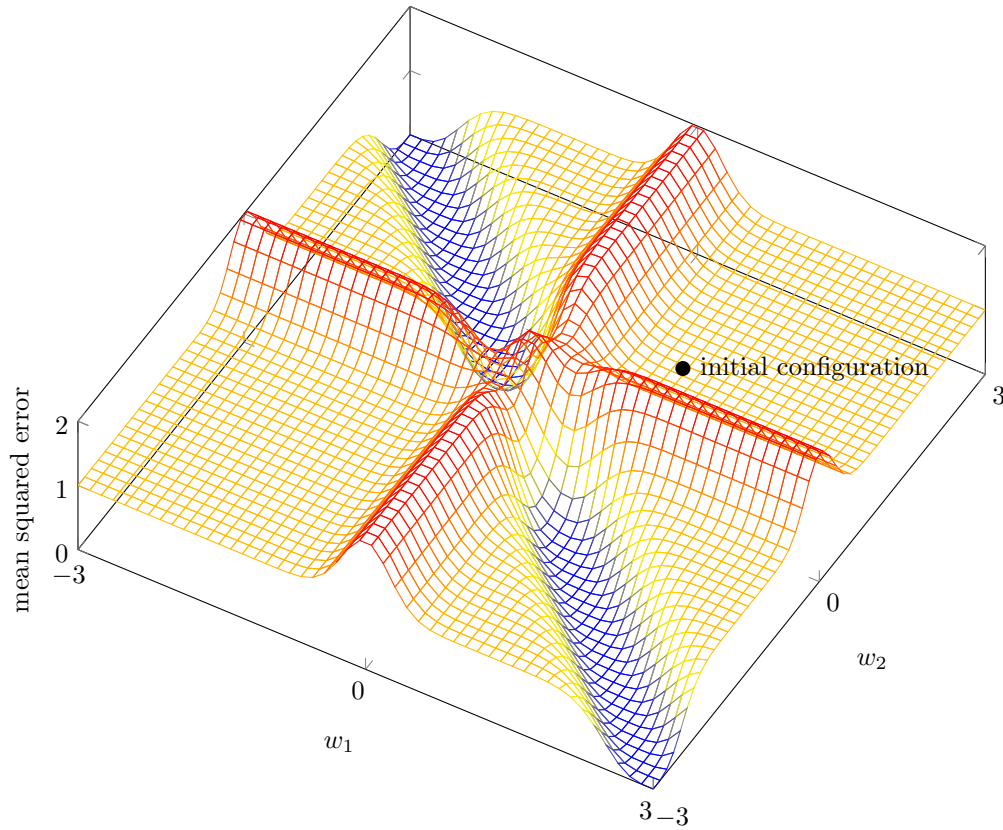
9

Figure 12: The error-weight surface of the stripe problem.

Training using gradient descent To test the above findings, this simple neural network was trained using the `keras` framework. As expected, the Stochastic Gradient Descent (SGD) optimizer did not achieve any noticable progress. After 20,000 iterations, the weights were at $w_1 = w_2 = 1.105$, i.e. they did not move to a configuration where $w_1 = -w_2$ which would constitute the global minimum.

Fig. 13 depicts the training accuracy across the 20,000 iterations. The only progress that was achieved was in the order of 10^{-3}, and that was not in the direction of the global minimum. Furthermore, the graph seems to converge to a value around 0.25.

This means that we have found a simple example where SGD suffers the local minimum problem.
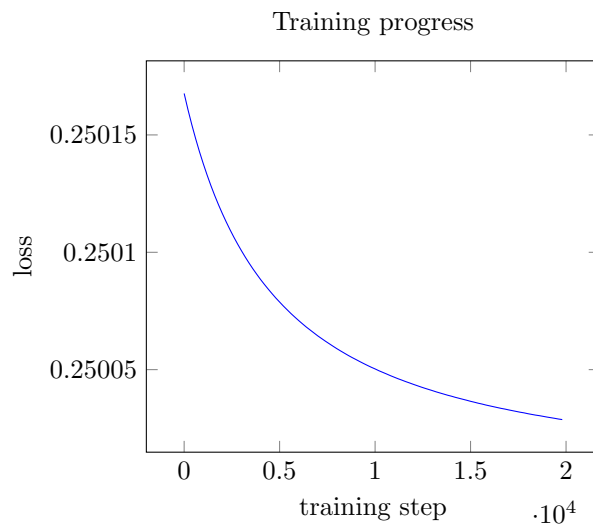

Figure 13: Loss over time during training.

4.3   Training the neural network with the shallow excitation gradient

Let's come back to the neural network from Sec. 3.2 given in Fig. 6. Recall the two points in input space **a** and **b** with initial and target values given in Table 1.

point in input space	initial value	target value
$\mathbf{a} = \begin{bmatrix} 0.6 \\ 0.4 \end{bmatrix}$	0.51	0.2
$\mathbf{b} = \begin{bmatrix} 0.6 \\ 1.0 \end{bmatrix}$	0.55	0.8

Table 1: Initial and target values for **a** and **b**.

In order to test the behaviour of gradient descent, we will give point **b** a weighting four times higher than **a** by simply duplicating **b** four times in the training set which is then given by $\{\mathbf{a}, \mathbf{b}, \mathbf{b}, \mathbf{b}, \mathbf{b}\}$.

The neural network's weights w_1 and w_2 were trained for 5,000 steps using `keras`. Note that the bias was kept constant at $b = 0.45$, and the network has no activation function. Fig. 14 shows the training progress over time. It can be seen that the gradient descent optimizer initially reduces the loss quite quickly (hence the steep slope), but around step 100, the slope becomes much more shallow. The reason for this is that until this step, the network prioritized getting a better prediction result for **b**, but the initial rapid improvements to the four points at **b** caused significant deteriorations in the predictions for point **a**. Fig. 15 illustrates this point, as the prediction for **a** first moves away from the target (it increases from 0.51, instead of decreasing to the target 0.2) before performing a sharp turn around iteration 100 after which it decreased until eventually meeting 0.2.

Finally, Fig. 16 shows how the weights w_1 and w_2 were updated during the training process. It can clearly be seen that SGD does not choose the optimal way of updating weights which would be a straight line from the initial weight configuration to the final configuration.
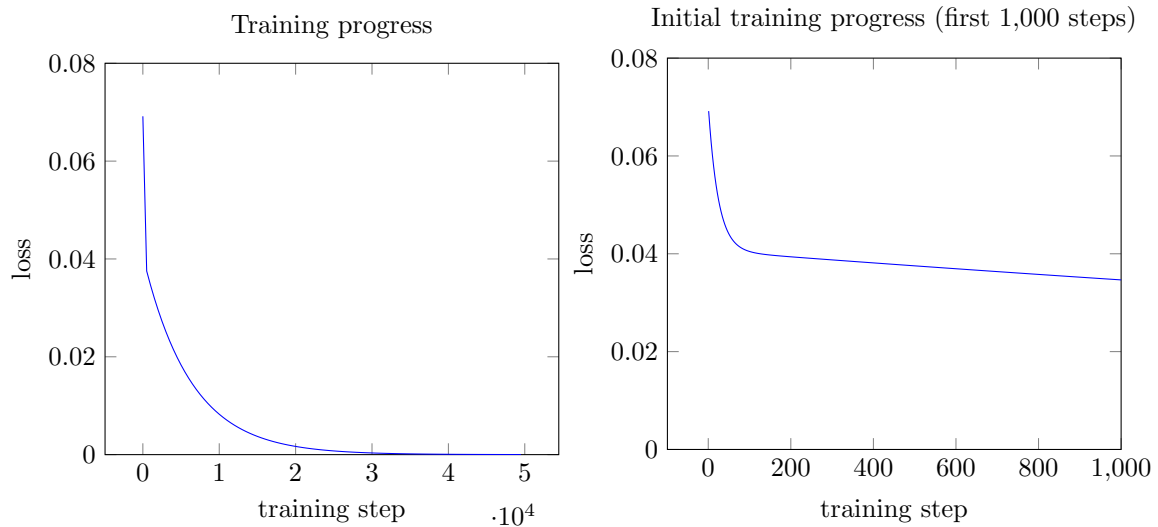

Figure 14: Loss over time during training. Notice the abrupt change in gradient which can be seen in more detail on the right.
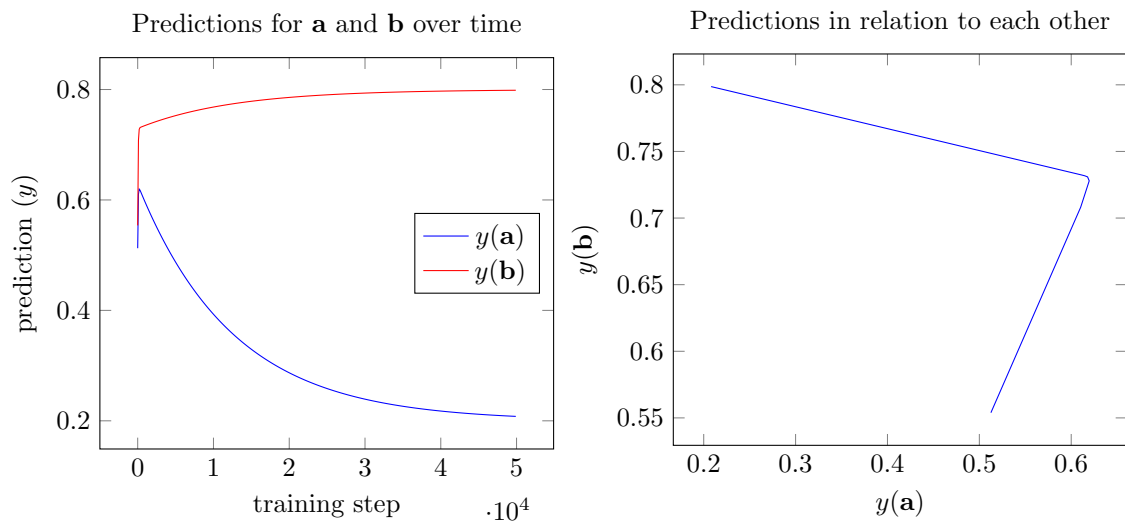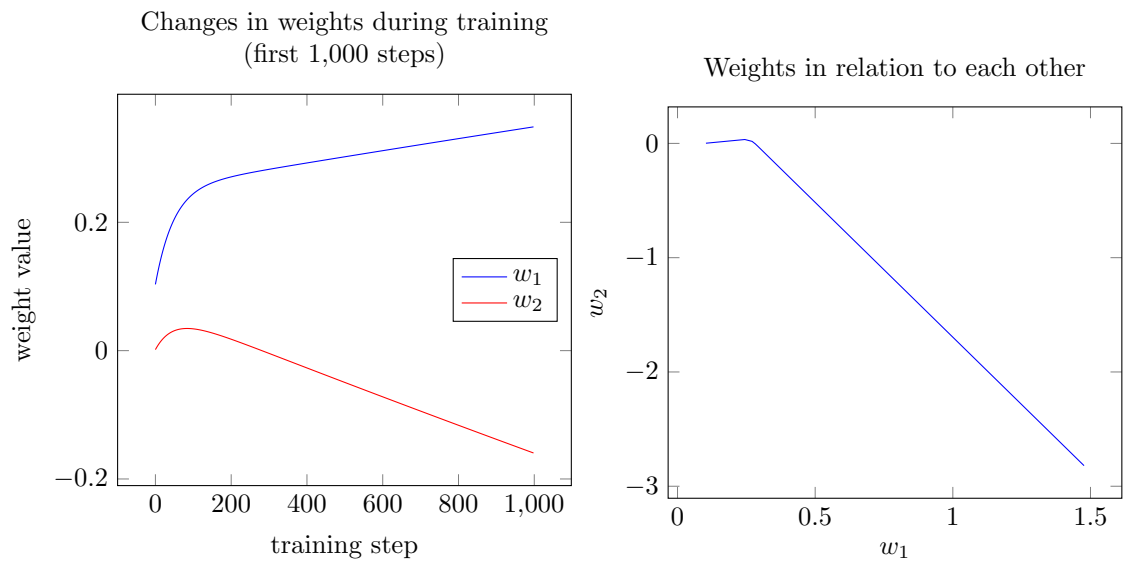
Figure 15: Changes in the predictions of **a** and **b** during the training process.

Figure 16: Changes in weights during the training process.

References

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization.