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 $g \in \mathbf{y}$.

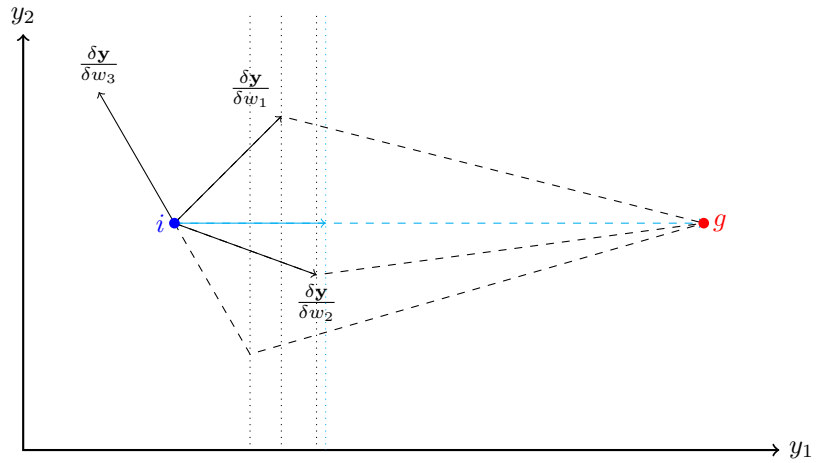

Figure 1: Diagram of the output space.

We define the cost function as

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

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.

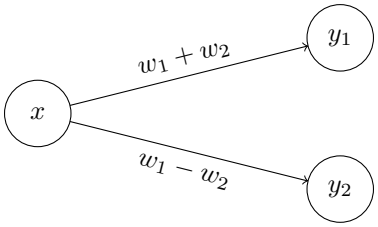

Figure 2: The neural network.

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

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

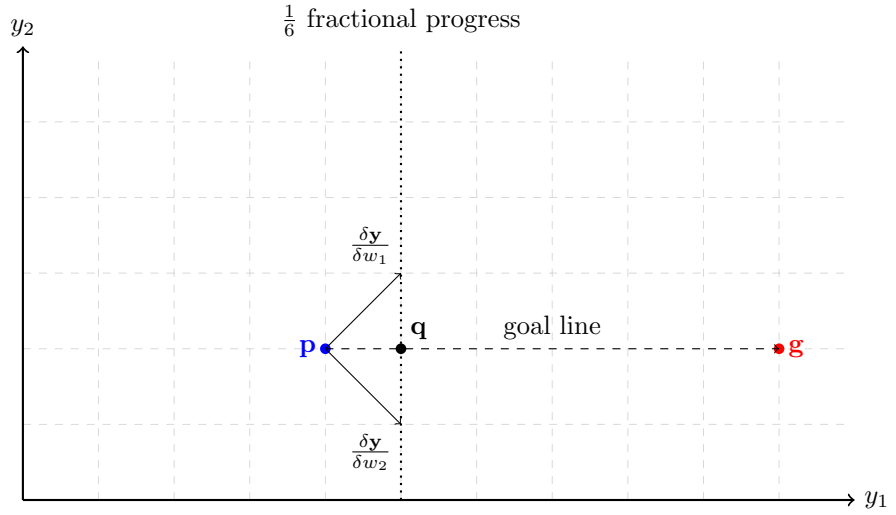

Figure 3: Diagram of the weights' partial derivatives in output space along with the sub-goal.

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

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

so

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

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.)

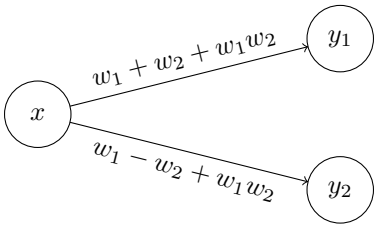

Figure 4: The neural network.

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.

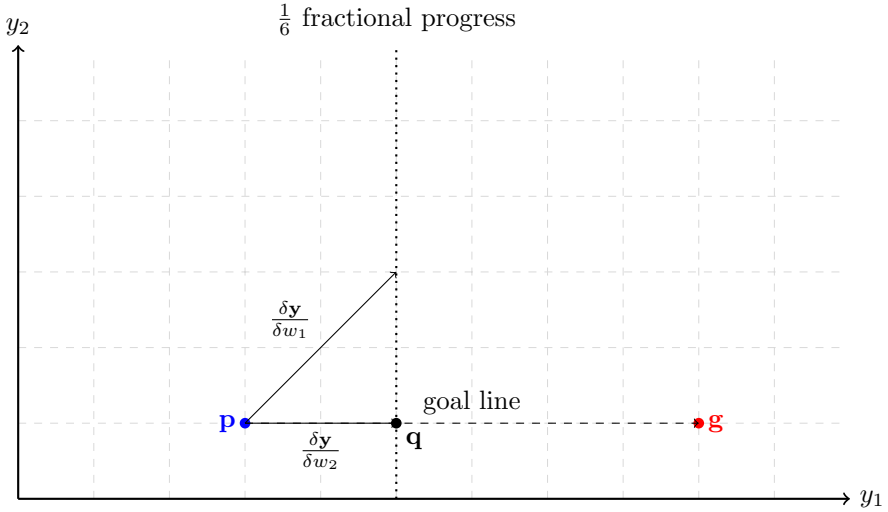

Figure 5: Diagram of the weights' partial derivatives in output space along with the sub-goal.

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.

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.

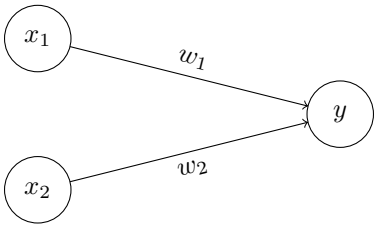

Figure 6: The neural network.

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.

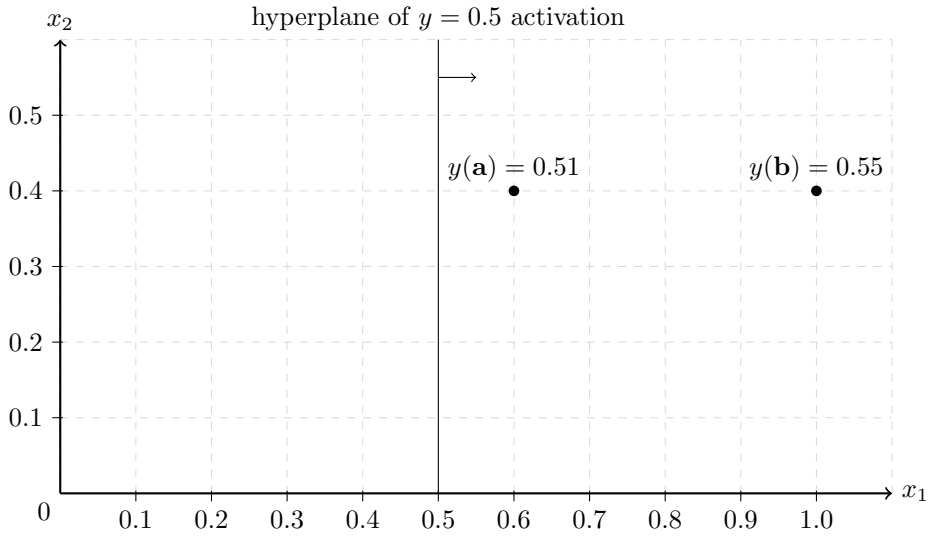

Figure 7: The initial configuration with the hyperplane. The arrow is pointing in the direction of increasing activation.

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.

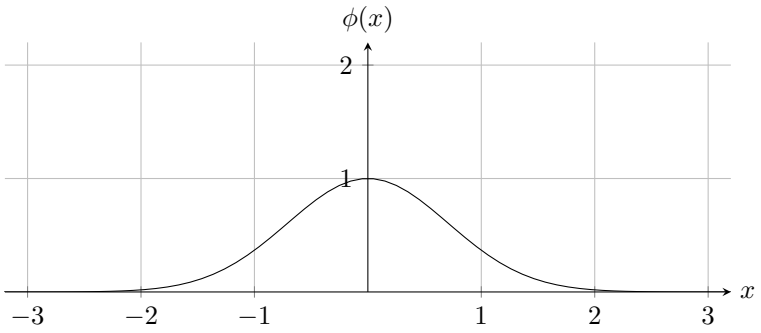

Figure 8: Plot of the radial basis activation function.

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

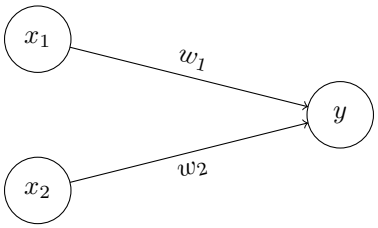

Figure 9: The neural network.

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

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

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

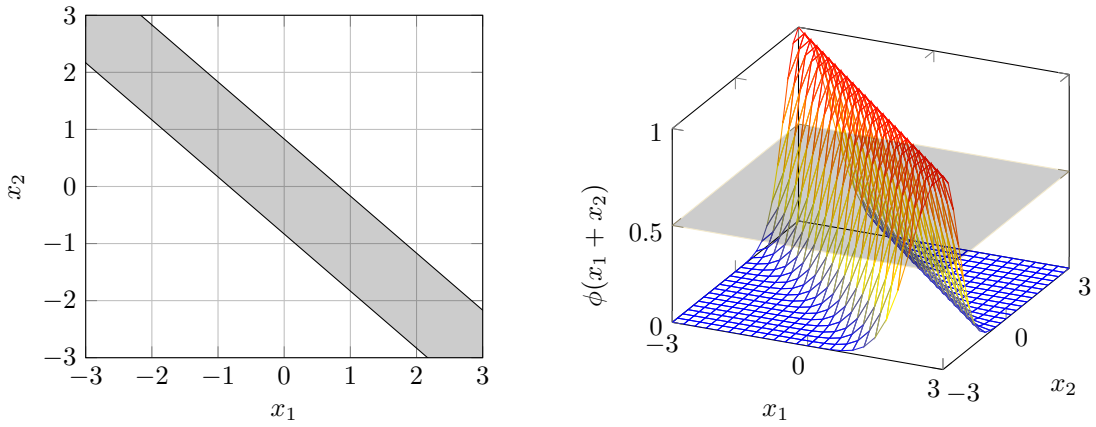

Figure 10: Plot of hyperplanes in input space of the radial basis activation function.

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.

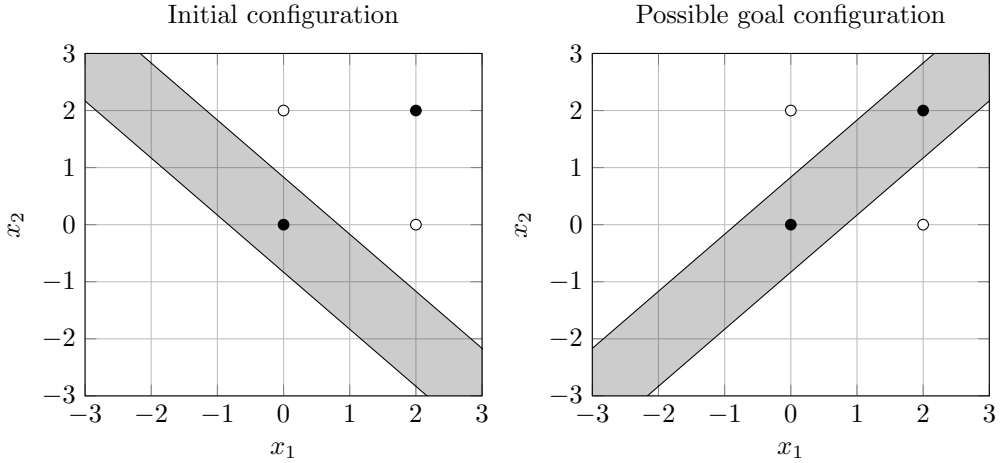

Figure 11: Initial and possible goal configuration for the stripe problem.

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.

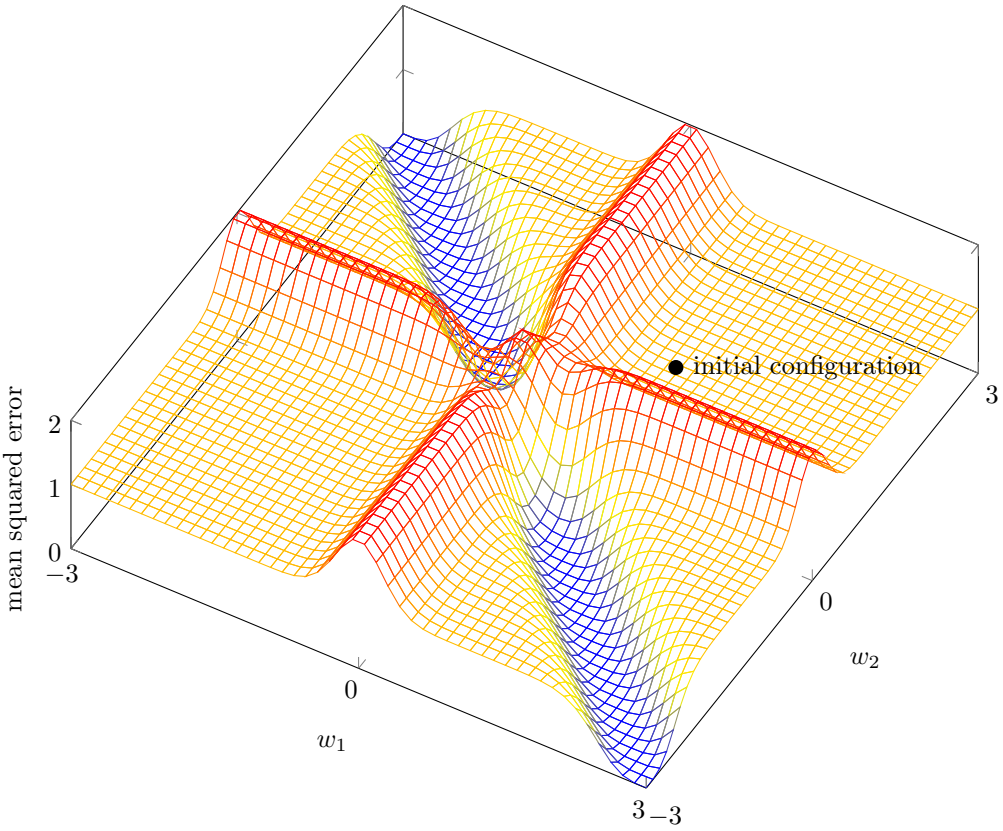

Figure 12: The error-weight surface of the stripe problem.

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.

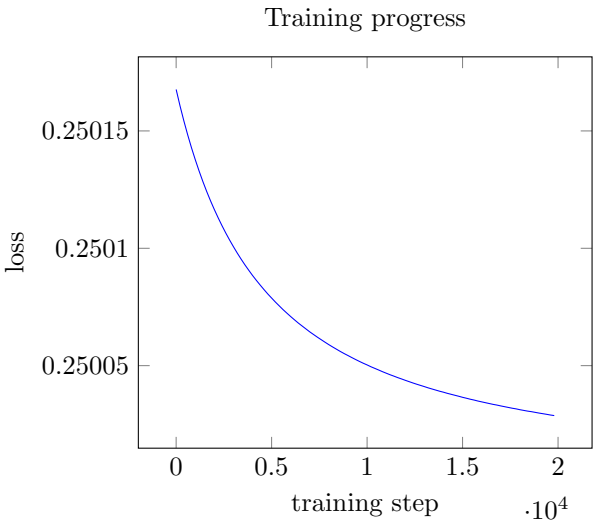

Figure 13: Loss over time during training.

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.

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 $\{\mathbf{a}, \mathbf{b}, \mathbf{b}, \mathbf{b}, \mathbf{b}\}$.

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**.

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.

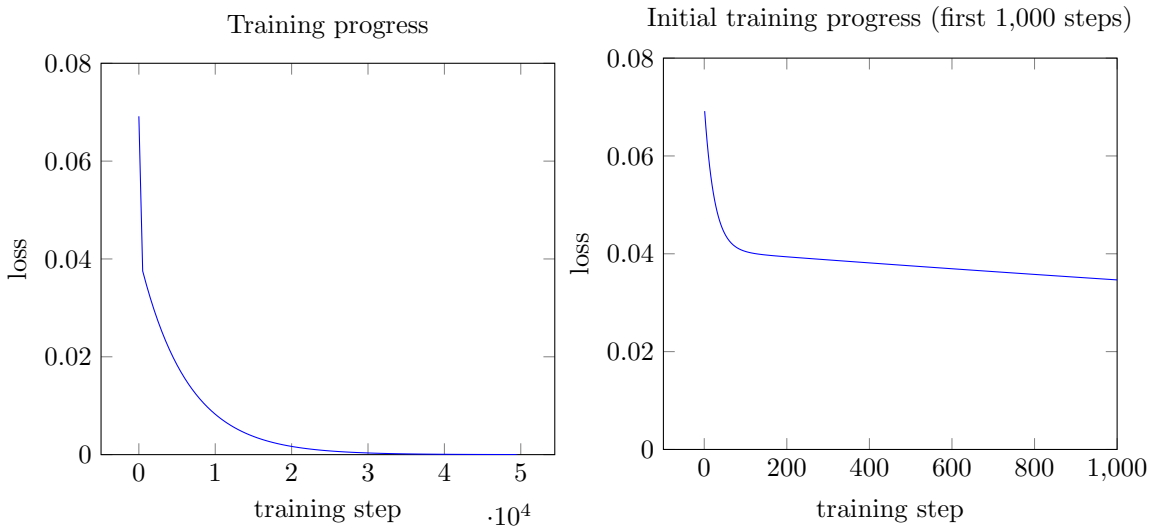

Figure 14: Loss over time during training. Notice the abrupt change in gradient which can be seen in more detail on the right.

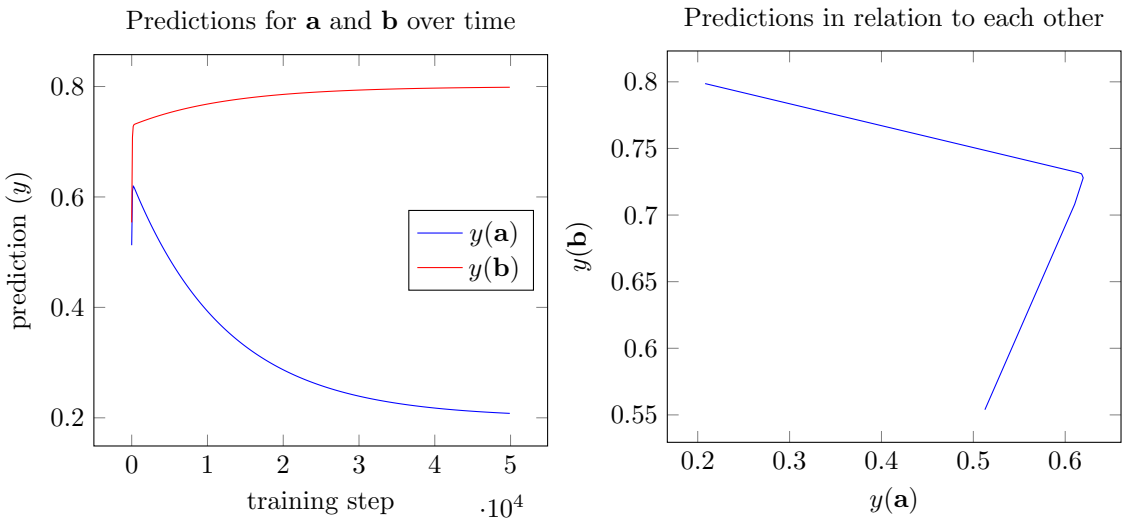

Figure 15: Changes in the predictions of **a** and **b** during the training process.

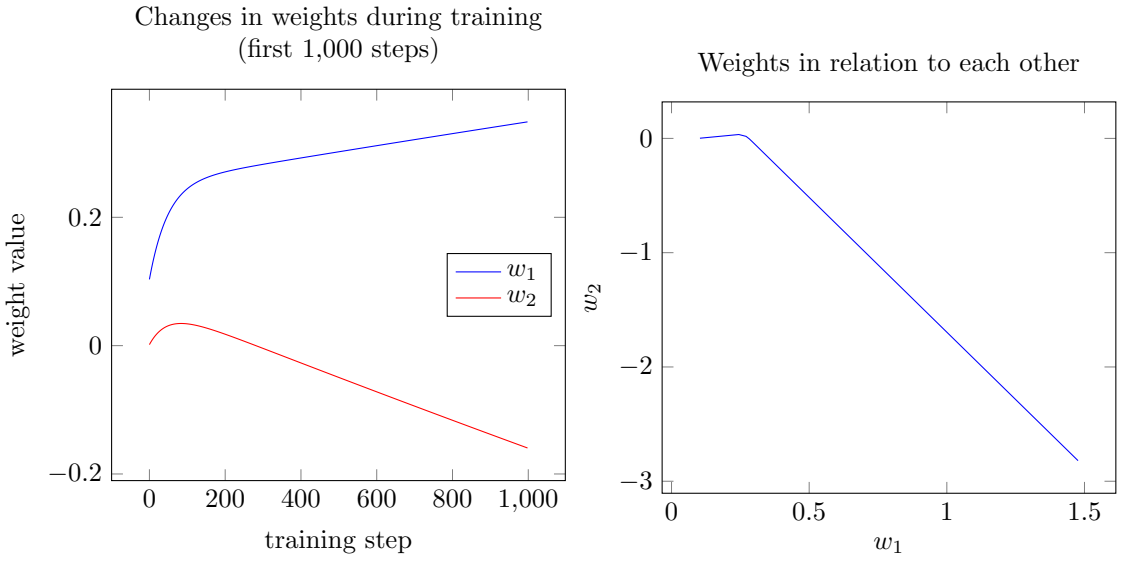

Figure 16: Changes in weights during the training process.

5 Week 11

5.1 Defining unrealizable regions

An *unrealizable region* is a subspace of the output space that cannot be attained. Consider a neural network architecture expressed as the function $\mathbf{f}(\mathbf{x}; \mathbf{W}_1, \ldots, \mathbf{W}_n)$ that takes the weight configurations $\mathbf{W}$ for each layer as a parameter. Given the sample $\langle \mathbf{x}, \mathbf{y} \rangle$ where $\mathbf{x}$ lies in input space and $\mathbf{y}$ in output space, we say that $\mathbf{y}$ is *strongly unrealizable* iff there exists no weight configuration $\mathbf{W}_1, \ldots, \mathbf{W}_n$ such that $\mathbf{f}(\mathbf{x}; \mathbf{W}_1, \ldots, \mathbf{W}_n) = \mathbf{y}$.

A point is *weakly unrealizable* for particular training algorithm if that algorithm cannot achieve a weight configuration such that $\mathbf{f}(\mathbf{x}; \mathbf{W}_1, \ldots, \mathbf{W}_n) = \mathbf{y}$.

5.2 The problem of non-orthogonal weight gradients

Consider a neural network with two weights w_1 and w_2, as well as two outputs y_1 and y_2. Figure 17 depicts a particular configuration in output space, showing an unrealizable region as well as how changes in the weights will affect the output[1]. Note that in this example, $\boldsymbol{\Delta}\mathbf{w}^+ \neq -\boldsymbol{\Delta}\mathbf{w}^-$, i.e. changing a weight in the negative direction does *not* result in a change in the opposite direction because the problem is inherently non-linear.

Furthermore, the vector $\boldsymbol{\Delta}\mathbf{w}^+$ is *not* the same as the partial derivative $\frac{\delta w}{\delta \mathbf{y}}$. Instead, the value of $\boldsymbol{\Delta}\mathbf{w}^+$ was obtained by calculating the neural network's output, substituting w for $w + \alpha$ for some small α. Similarly, $\boldsymbol{\Delta}\mathbf{w}^-$ was obtained by calculating the output where w is $w - \alpha$.

The current configuration produces point $\mathbf{p}$ in output space. Say we want to achieve the sub-goal $\mathbf{s}$ in the next step. What change in w_1 and w_2 will get us closest to $\mathbf{s}$?

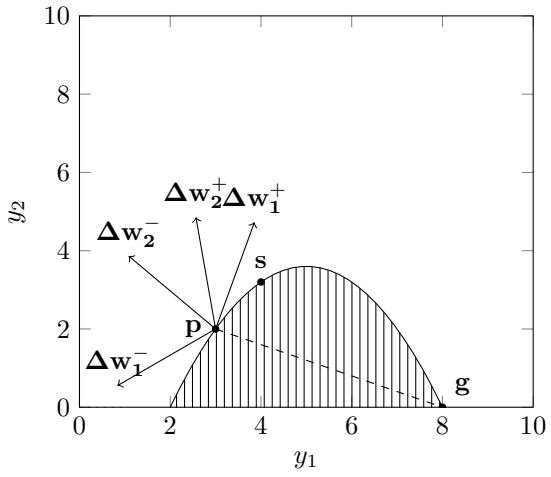

Figure 17: Weight changes in output space.

Treating the problem as being linear How do we define *linear*, and how would we solve this?

Brute-force all assignments We could calculate the output of the network for all possible assignments of the weights. Every weight w will be tested with the values $w - \alpha$, w, and $w + \alpha$. The limitation of this technique is that the time complexity will be $\mathcal{O}(3^n)$ where n is the number of weights, and this calculation has to be carried out at each step.

Random assignments Instead, we could test a subset of the 3^n possible assignments. For example, we could randomly generate assignments (each weight w being either $w - \alpha$, w, or $w + \alpha$) until we find one that is *good enough* according to some previously defined stopping condition.

[1] The diagram displays a made-up scenario that does not correspond to an actual experiment.

5.3 Controlling properties of the RBF stripe problem

5.3.1 Width of the basin

In order to control the width of the basin, we need to control the thickness of the RBF activation function. We introduce the parameter ϵ which will affect the width of the RBF function from (3), now given as

$$\phi(x;\epsilon) = e^{-\epsilon x^2}. \tag{4}$$

Figure 18 depicts a plot of the RBF for $\epsilon = 10$, showing clearly that the resulting function has a thinner graph as compared to Figure 8 where $\epsilon = 1$. Furthermore, Figure 19 shows that the error-weight surface for the stripe problem given in Section 4.2 has a narrower minimum basin using the thinner RBF as compared to Figure 12.

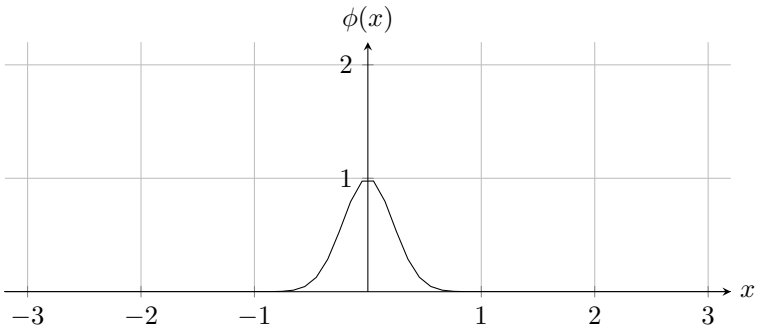

Figure 18: Plot of the radial basis activation function with $\epsilon = 10$.

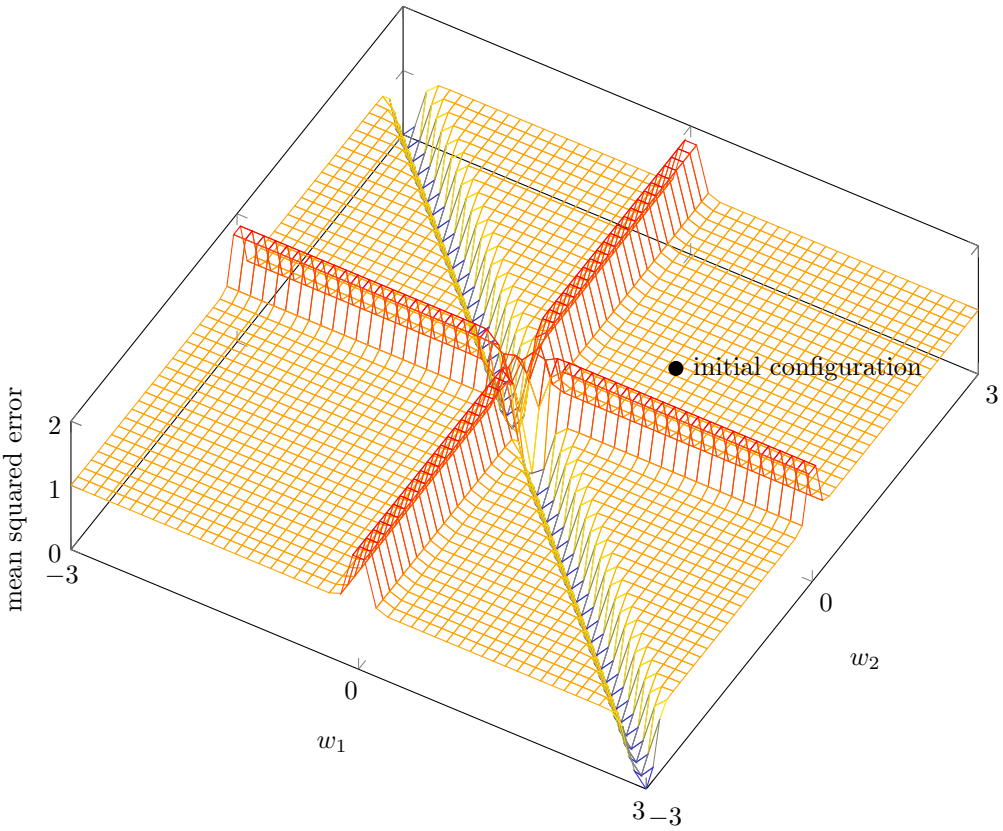

Figure 19: The error-weight surface of the stripe problem with a narrower basin.

5.3.2 Guaranteeing a finite number of solutions with MSE of zero

For the stripe problem as defined in Section 4.2, there exists no real-valued weight configuration that leads to a mean squared error of 0. This is because the activation function is always greater than 0. In fact, the range of ϕ is the interval $(0, 1]$.

We will set the training dataset as shown in Table 2. Essentially, the data is the same as shown in Figure 11, except that the targets of 0 have been adapted to slightly higher values.

input ($\mathbf{x}$)	target output ($\mathbf{y}$)	initial output
$\begin{bmatrix}0\\0\end{bmatrix}$	1	1
$\begin{bmatrix}2\\2\end{bmatrix}$	1	$\phi(4)$
$\begin{bmatrix}0\\2\end{bmatrix}$	$\phi(2)$	$\phi(2)$
$\begin{bmatrix}2\\0\end{bmatrix}$	$\phi(2)$	$\phi(2)$

Table 2: Initial and target values for **a** and **b**.

In the initial configuration, $w_1 = w_2 = 1$, so the mean squared error is

$$\begin{aligned} MSE &= \sum_i \left(f(\mathbf{x}_i) - \mathbf{y}_i\right)^2 \\ &= (1-1)^2 + \left(e^{-8\epsilon} - 1\right)^2 + \left(\phi\left(\sqrt{2}\right) - \phi\left(\sqrt{2}\right)\right)^2 + \left(\phi\left(\sqrt{2}\right) - \phi\left(\sqrt{2}\right)\right)^2. \\ &= \left(e^{-8\epsilon} - 1\right)^2. \end{aligned}$$

If we consider the weight configuration $w_1 = 1$ and $w_2 = -1$, we obtain

$$\begin{aligned} MSE &= \left(f\left(\begin{bmatrix}0\\0\end{bmatrix}\right) - 1\right)^2 + \left(f\left(\begin{bmatrix}2\\2\end{bmatrix}\right) - 1\right)^2 + \left(f\left(\begin{bmatrix}0\\2\end{bmatrix}\right) - \phi\left(\sqrt{2}\right)\right)^2 + \left(f\left(\begin{bmatrix}2\\0\end{bmatrix}\right) - \phi\left(\sqrt{2}\right)\right)^2 \\ &= \left(\phi(0+0) - 1\right)^2 + \left(\phi(2-2) - 1\right)^2 + \left(\phi(0-2) - \phi(2)\right)^2 + \left(\phi(2-0) - \phi(2)\right)^2 \\ &= \left(\phi(0) - 1\right)^2 + \left(\phi(0) - 1\right)^2 + \left(\phi(-2) - \phi(2)\right)^2 + \left(\phi(2) - \phi(2)\right)^2. \end{aligned}$$

Noticing that $\phi(0) = 1$ and $\phi(x) = \phi(-x)$ for any x, we obtain $MSE = 0$. In fact, there are only two unique weight configurations that lead to $MSE = 0$; these are $\mathbf{w} = \begin{bmatrix}1\\-1\end{bmatrix}$ and $\mathbf{w} = \begin{bmatrix}-1\\1\end{bmatrix}$, so these are the only global minima in the error-weight surface. Incidentally, the solutions do *not* depend on the parameter ϵ for the activation function $\phi(x)$ provided $\epsilon \neq 0$.

Remarks: To improve, it might be better to instead use a so-called *non-analytic smooth function* that is similar to the RBF activation function, except that its range includes zero. One example of such a function is the *bump function*[2]. This will be investigated in the next semester.

[2] `https://en.wikipedia.org/wiki/Bump_function`

6 Week 12

6.1 Assumptions on linearity

Using the notation from Section 5.2, we make the following assumptions:

- For a given weight w, the change that $w+\alpha$ for a small α will elicit in output space, denoted $\mathbf{\Delta w}^+$, is *linearly independent* of $\mathbf{\Delta w}^-$ in the local context of α.

  It follows that $\mathbf{\Delta w}^+$ and $\mathbf{\Delta w}^-$ are usually not collinear.

- Given two weights w_1 and w_2, we assume that the pairs $\langle \mathbf{\Delta w_1^+}, \mathbf{\Delta w_2^+} \rangle$, $\langle \mathbf{\Delta w_1^+}, \mathbf{\Delta w_2^-} \rangle$, $\langle \mathbf{\Delta w_1^-}, \mathbf{\Delta w_2^+} \rangle$, and $\langle \mathbf{\Delta w_1^-}, \mathbf{\Delta w_2^-} \rangle$ obey the rules of vector sums in the context of α.

- For our purposes, a linear combination $\mathcal{L}$ of two vectors $\mathbf{\Delta w_1}$ and $\mathbf{\Delta w_2}$ is defined as

  $$\mathcal{L}(\mathbf{\Delta w_1}, \mathbf{\Delta w_2}) = c_1 \mathbf{\Delta w_1} + c_2 \mathbf{\Delta w_2}$$

  where c_1 and c_2 are real-valued scalars such that $0 \leq c_i \leq 1$ for both i.

- The weight update corresponding to a linear combination $\mathcal{L}(\mathbf{\Delta w_1^+}, \mathbf{\Delta w_2^+})$ is $w_1 \leftarrow w_1 + c_1\alpha$ and $w_2 \leftarrow w_2 + c_2\alpha$.

  When considering weight changes in the negative direction, i.e. $\mathbf{\Delta w_1^-}$, the corresponding weight update will be $w_1 \leftarrow w_1 - c_1\alpha$.

  Note that the update $w \leftarrow w + c\alpha$ is based on the assumption that the mapping between weight and output space is *linear* in the context of α. This might not be the case and yield inaccurate results, so instead we might need to modify the algorithm to carry out a linear search instead to find what change in w will yield a vector of length $c\,\|\mathbf{\Delta w}\|$.

- It does not make sense to compute a linear combination of $\mathbf{\Delta w}^+$ and $\mathbf{\Delta w}^-$ for the same weight w because this does not correspond to a meaningful weight update.

Question 1 I understand how we aim to achieve a subgoal point using orthogonalization (Gram-Schmidt) and vector sums, *but how do we select the subgoal point?*

Do we just choose the subgoal to be, for example, one tenth of the way to the goal, i.e. the subgoal is lying on the goal line? Then if we deviate from the goal line this is because our local assumptions of linearity aren't 100 percent accurate (but that might not be too bad, as long as we don't deviate too much). The deviation from the goal line would then be described as a weakly unrealizable region.

If it is impossible to achieve a point on the goal line (because all gradients point away from it, like in Figure 17), we could just go for the weight change that makes the maximum fractional progress, and only change that one weight. Otherwise we could devise some sort of informed search that will yield the weight combination with the maximum fractional progress[3].

Question 2 To clarify, what is the definition of output space?

- The output space is defined by the activation of the neurons in the output layer, i.e. each dimension of the output space is the activation of a neuron. In this case, how do we deal with multiple training samples?

- The output space is only defined for networks with one output neuron. Each dimension corresponds to the activation of the output neuron for a training sample, so the number of dimensions of the output space is the number of training samples.

[3] It is important to notice that while the change in weight α is the same for each weight, the vector $\mathbf{\Delta w}$ in output space might not have the same magnitude as the others. This means that potentially a combination of weights might lead to a better fractional progress than just changing only one weight.

References

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