We have a network of three neurons with activity (firing rate) given by x_1, x_2, x_3, respectively. The neurons are arranged in a sequence and the firing rate of neuron 3 is given by:

$$x_3 = w_2 x_2 = w_2 w_1 x_1$$

where, w_1 and w_2 are the synaptic weights connecting neuron 1 to neuron 2 and neuron 2 to neuron 3, respectively.

The goal is to train this network such that the firing rate of neuron 3 matches a target firing rate y. In order to do so, we use a quadratic loss function given by:

$$l(y, x_3) = \frac{1}{2}(y - x_3)^2$$

Our goal is to determine the set of weights for which the loss function is minimized.

a) Warm-up. If $x_1 > 0$, $w_1 < 0$, and $x_3 < y$, should w_2 increase or decrease to improve the loss? Should w_1 increase or decrease?

Since $w_1 < 0$ and $x_1 > 0$, w_2 should decrease and w_1 should decrease for the loss to improve. This is because when $w_1 > 0$, a negative w_1 will cause x_2 and x_3 to have a negative activity. Since $x_3 < y$, this means that the activity of x_3 is too low, so increasing w_2 (which will increase the activity of x_3) and decreasing w_1 (which will decrease the activity of x_3) will help to reduce the loss.

To be more precise than the above procedure, we use gradient descent, which computes derivatives of the loss function with respect to the synaptic weights, and then uses these derivatives to iteratively update the weights. Backpropagation is simply a procedure for computing these derivatives by the chain rule taught in any standard calculus course.

b) Compute the derivative of l with respect to x_3. Then, using the chain rule compute the derivatives with respect to w_2, x_2 and w_1

The loss function is given by:

$$l(y, x_3) = \frac{1}{2}(y - x_3)^2$$

Computing the derivative of l with respect to x_3:

$$\frac{dl}{dx_3} = \frac{d}{dx_3}\frac{1}{2}(y - x_3)^2$$

$$\therefore \frac{dl}{dx_3} = \frac{1}{2} \times 2(y - x_3) \times (-1)$$

$$\therefore \frac{dl}{dx_3} = x_3 - y$$

$\because x_3 = w_2 x_2$, we can rewrite the partial derivate wrt w_2 as:

$$\frac{\partial l}{\partial w_2} = \frac{\partial l}{\partial x_3} \times \frac{\partial x_3}{\partial w_2}$$

$$\therefore \frac{\partial l}{\partial w_2} = (x_3 - y) \times \frac{\partial w_2 x_2}{\partial w_2}$$

$$\therefore \frac{\partial l}{\partial w_2} = (x_3 - y)x_2$$

Similarly, the partial derivative wrt x_2 can be written as:

$$\frac{\partial l}{\partial x_2} = \frac{\partial l}{\partial x_3} \times \frac{\partial x_3}{\partial x_2}$$

$$\therefore \frac{\partial l}{\partial x_2} = (x_3 - y) \times \frac{\partial w_2 x_2}{\partial x_2}$$

$$\therefore \frac{\partial l}{\partial x_2} = (x_3 - y)w_2$$

We also have: $x_3 = w_2 w_1 x_1$, therefore we can rewrite the partial derivative wrt w_1 as:

$$\frac{\partial l}{\partial w_1} = \frac{\partial l}{\partial x_3} \times \frac{\partial x_3}{\partial w_1}$$

$$\therefore \frac{\partial l}{\partial w_1} = (x_3 - y) \times \frac{\partial w_2 w_1 x_1}{\partial w_1}$$

$$\therefore \frac{\partial l}{\partial w_1} = (x_3 - y)w_2 x_1$$

c) Using your results above, write down an update rule that is guaranteed to decrease the loss function for a sufficiently small step size (learning rate).

The update rule that is guaranteed to decrease the loss function for a sufficiently small step size (learning rate) is:

$$w_2 = w_2 - \alpha(x_3 - y)x_2$$

$$w_1 = w_1 - \alpha(x_3 - y)w_2 x_1$$

where, α is the learning rate.

This update rule uses the derivatives that we computed earlier to adjust the weights in a way that will reduce the loss. The learning rate determines the size of the update, and if it is sufficiently small then the updated weights will always result in a lower loss than the previous weights.

Feedforward backpropagation (nonlinear case)

Real biological networks are nonlinear, as are artificial networks used in practice in machine learning. Consider a slight modification to the above network where x_2 has a nonlinear activation function (usually, the final layer of the network is linearly read out in machine learning applications). Specifically, consider the model:

$$x_3 = w_2\sigma(w_1x_1)$$

where, σ is a nonlinear function. Unless otherwise specified, simply assume that σ is an arbitrary monotonically increasing nonlinear function with derivative σ'

d) Recompute your answers to question (b) with the nonlinearity. How do your answers change after the introduction of this nonlinearity?

The derivative of l with respect to x_3 is still:

$$\frac{dl}{dx_3} = (x_3 - y)$$

However, since x_3 is now a nonlinear function of x_2 and x_1, the derivatives with respect to w_2, x_2 and w_1 will change. We can use the chain rule to compute these derivatives as follows:

$$\frac{\partial l}{\partial w_2} = \frac{\partial l}{\partial x_3} \times \frac{\partial x_3}{\partial w_2}$$

$$\therefore \frac{\partial l}{\partial w_2} = (x_3 - y) \times \sigma(w_1x_1)$$

$$\frac{\partial l}{\partial x_2} = \frac{\partial l}{\partial x_3} \times \frac{\partial x_3}{\partial x_2}$$

$$\therefore \frac{\partial l}{\partial x_2} = (x_3 - y) \times w_2 \times \sigma'(w_1x_1) \times w_1$$

$$\frac{\partial l}{\partial w_1} = \frac{\partial l}{\partial x_3} \times \frac{\partial x_3}{\partial w_1}$$

$$\therefore \frac{\partial l}{\partial w_1} = (x_3 - y) \times w_2 \times \sigma^{'}(w_1 x_1) \times x_1$$

These derivatives are different from the ones computed in the linear case because the nonlinearity introduces additional terms that need to be accounted for.

The update rule that is guaranteed to decrease the loss function for a sufficiently small step size (learning rate) is:

$$w_2 = w_2 - \alpha(x_3 - y) \times \sigma(w_1 x_1)$$

$$w_1 = w_1 - \alpha(x_3 - y) \times w_2 \times \sigma^{'}(w_1 x_1) \times x_1$$

This update rule is similar to the one in the linear case, but it incorporates the nonlinearity by using the derivatives that we computed above. Again, the learning rate determines the size of the update, and if it is sufficiently small then the updated weights will always result in a lower loss than the previous weights.

A typical choice for σ in machine learning applications is the rectified linear unit (ReLU), $\sigma(x) = max(0, x)$. Compute the gradient of the weights for this choice of σ. Can you foresee any problems for the backpropagation learning rule in this scenario? (Hint: think about what happens when x_1 and w_1 have opposite signs.)

If we choose the ReLU as the nonlinearity, then the gradient of the weights is given by:

$$\frac{\partial l}{\partial w_2} = \frac{\partial l}{\partial x_3} \times \frac{\partial x_3}{\partial w_2}$$

$$\therefore \frac{\partial l}{\partial w_2} = (x_3 - y) \times max(0, w_1 x_1)$$

$$\frac{\partial l}{\partial x_2} = \frac{\partial l}{\partial x_3} \times \frac{\partial x_3}{\partial x_2}$$

$$\therefore \frac{\partial l}{\partial x_2} = (x_3 - y) \times w_2 \times f(w_1 x_1) \times w_1$$

$$\frac{\partial l}{\partial w_1} = \frac{\partial l}{\partial x_3} \times \frac{\partial x_3}{\partial w_1}$$

$$\therefore \frac{\partial l}{\partial w_1} = (x_3 - y) \times w_2 \times f(w_1 x_1) \times w_1$$

where, $f(w_1 x_1) = 1$, if $w_1 x_1 > 0$ and is 0 otherwise.

One potential problem with this backpropagation learning rule is that if x_1 and w_1 have opposite signs, then the gradient w_1 will be 0 because the ReLU will always be 0 in this case. This means that the weight w_1 will not be updated, which can slow down or even prevent the learning process. One way to

address this problem is to use a different nonlinearity that does not have this issues, such as the sigmoid function or the hyperbolic tangent. These nonlinearities have a non-zero gradient for all input values, which allows the weights to be updated even when x_1 and w_1 have opposite signs.