

# Backpropagation in Feedforward Networks

Dr. Olivia Guest

October 16, 2015

## 1 Overview

Last week we ran and trained a 2-layer perceptron. For reasons that became obvious (i.e., if no linear separation of the target states can be accomplished) 2-layer perceptrons are not as useful as one might initially assume. For this reason, we must turn to adding a 3rd layer of units between the input and the output units. This as you might expect is called the hidden layer because they are neither input nor output units — their activations are not directly in contact with the environment the network is in. Adding hidden units produces a 3-layer perceptron. As we shall see it can now be taught classification problems like exclusive-or which are not linearly separable.

Backpropagation in a feedforward network works by running the network **forwards** (as we did with the 2-layer perceptron) and then running the network **backwards**. This second phase is required because we need to calculate targets not only for the output units but also for the hidden units. Their targets are calculated based on the output units' activations and the output targets. Using these hidden unit targets we can then calculate the error of the hidden units and subsequently update the weights from the input units to the hidden units, as well as being able to update the weights from the hidden units to the output units. In other words, running the network backwards allows us to calculate errors for every single unit (apart from those in the input layer, which have no targets). The errors for every unit are then used to calculate the connection weight updates, thus adjusting our network to conform to the required output states given an input.

## 2 Forwards Phase: Propagation of Activations

During the feedforward phase, each unit, apart from the input units, receives activations from units in the previous layer. Thus, the input to each unit,  $i$ , is:

$$\eta_i = b_i + \sum_j s_j w_{ji} \quad (1)$$

where  $b_i$  is the bias of unit  $i$  ( $b_i = -1$ ), and  $s_j$  is the state of a unit  $j$  that projects onto  $i$ . When unit  $i$  receives input  $\eta_i$ , its state changes to:

$$s_i = \frac{1}{1 + e^{-\eta_i}} \quad (2)$$

as usual by means of the **logistic function** (previously we called this simple  $f$ , when discussing the 2-layer perceptron). The logistic function is the activation function that is most commonly used with backpropagation, although  $\tanh(\eta_i)$  (the hyperbolic tangent function) is also common.

In Figure 1, we can see a graphical depiction of what the sigmoid activation function, in this case a logistic curve. The values on the  $x$  axis represent the values that  $\eta_i$  (the pre-synaptic input to unit  $i$ ) takes on, while the  $y$  axis represents the output of the activation function that will be the state of the unit:  $s_i$  (the post-synaptic state).

1. What does this function do to the pre-synaptic input  $\eta_i$  (as calculated in Equation 1)?
2. How is this different to how we defined our activation function  $f$  last week?
3. What is the constant  $e$  in Equation 2?
4. What are the differences and similarities between running the 3-layer and the 2-layer perceptrons?
5. What type of flow control statement is appropriate for calculating the  $\sum$  in Equation 1?
6. Fill in the `Run()` function to reflect any changes when compared to the 2-layer network. Use the inbuilt function `expit()` for the logistic function.

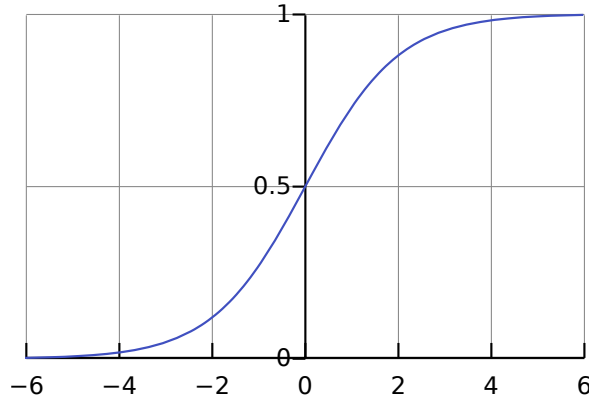


Figure 1: The shape of the logistic curve.

### 3 Backwards Phase: Propagation of Error Signal

After the feedforward phase, the network is run in reserve in order to gather the weight adjustments. Unit  $i$  sends the error signal:

$$\delta_i = \eta_i(1 - \eta_i)(t_i - s_i) \quad (3)$$

where  $t_i$  is the target state. If  $i$  is an output unit  $t_i$  is the pre-set target pattern. Notice how the pre- and post-synaptic states of the unit are used to measure the error — compare to how the output of the 2-layer perceptron was evaluated. This metric for calculating error is called cross-entropy error, other types (e.g., mean squared error) can also be used.

However, if  $i$  is a hidden unit, its target is:

$$t_i = \sum_j \delta_j w_{ij} \quad (4)$$

meaning that the error signal  $i$  emits is a function of the error signal emitted by the output units and the weights that connect them.

1. If the pre-synaptic activation of an output unit  $i$  is  $-6$  and its target  $t_i$  is  $1.0$ , what error signal  $d_i$  will it emit? (Use equations 1, 2, and 3.)
2. What type of flow control statement is appropriate for calculating the  $\sum$  in Equation 4?
3. What type of data is  $i$  and what type of data is  $s_i$ ?
4. What are the main differences and similarities between the backwards and the feedforward phases of running a network?
5. Why is the backwards phase required?
6. How many types of target  $t_i$  are there and why?
7. Fill in the `Backprop()` function to run the network backwards. Use the inbuilt function `logit()` for the inverse of the logistic function.

## 4 Weight Adjustments

Once the network has been run forwards and backwards for an epoch,  $\epsilon$ , weight updates are calculated using:

$$\Delta w_{ij}^\epsilon = -\mu \delta_j s_i + \nu \Delta w_{ij}^{\epsilon-1} \quad (5)$$

taking into account the momentum,  $\nu = 0.9$ , and the learning rate,  $\mu = 0.025$

## References

- Rumelhart, D. E., Hinton, G., & Williams, R. (1986). Learning internal representations by error propagation. *Parallel distributed processing: explorations in the microstructure of cognition, 1*, 318–362.