



# Νευρο-Ασαφής Υπολογιστική Neuro-Fuzzy Computing

Διδάσκων –  
Δημήτριος Κατσαρός

@ Τμ. ΗΜΜΥ  
Πανεπιστήμιο Θεσσαλίας



# Practice on multi-layer neural networks and backpropagation I



## Exercise-07

Show that backpropagation reduces to the LMS algorithm for a single-layer linear network (ADALINE).

### **SOLUTION**

The sensitivity calculation for a single-layer linear network:

$$\mathbf{s}^1 = -2\mathbf{F}'^1(\mathbf{n}^1)(\mathbf{t} - \mathbf{a}) = -2\mathbf{I}(\mathbf{t} - \mathbf{a}) = -2\mathbf{e}$$

The weight update is:

$$\mathbf{W}^1(k+1) = \mathbf{W}^1(k) - \alpha \mathbf{s}^1 (\mathbf{a}^0)^T = \mathbf{W}^1(k) - \alpha (-2\mathbf{e}) \mathbf{p}^T = \mathbf{W}^1(k) + 2\alpha \mathbf{e} \mathbf{p}^T$$

$$\mathbf{b}^1(k+1) = \mathbf{b}^1(k) - \alpha \mathbf{s}^1 = \mathbf{b}^1(k) - \alpha (-2\mathbf{e}) = \mathbf{b}^1(k) + 2\alpha \mathbf{e}$$

This is identical to the LMS algorithm.

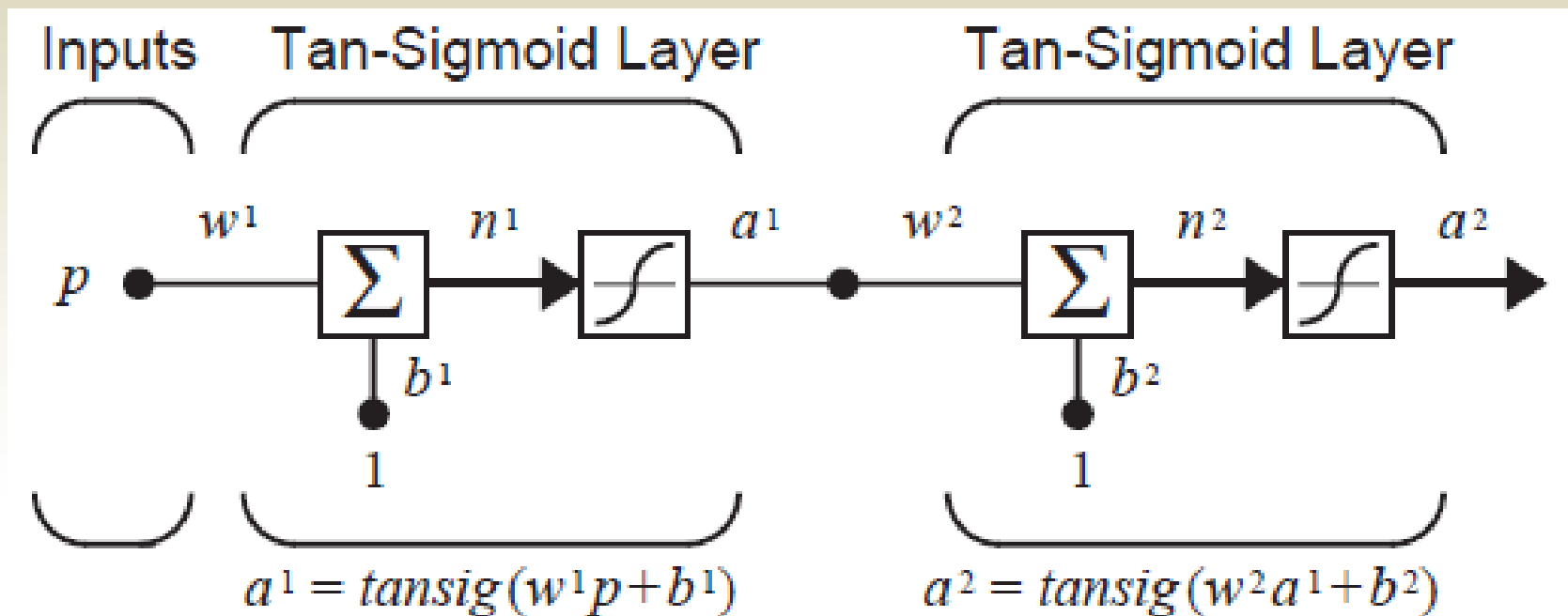
## Exercise-08

For the network shown below, the initial weights and biases are the following:

$$w^1(0) = -1, \quad b^1(0) = 1, \quad w^2(0) = -2, \quad b^2(0) = 1$$

An input/target pair is given:  $(p = -1, t = 1)$

Learning rate:  $\alpha = 1$ . Perform one iteration of backpropagation





## Exercise-08 Solution

The output of the first layer is then:

$$n^1 = w^1 p + b^1 = (-1)(-1) + 1 = 2$$

$$a^1 = \text{tansig}(n^1) = \frac{\exp(n^1) - \exp(-n^1)}{\exp(n^1) + \exp(-n^1)} = \frac{\exp(2) - \exp(-2)}{\exp(2) + \exp(-2)} = 0.964$$

The output of the second layer is then:

$$n^2 = w^2 a^1 + b^2 = (-2)(0.964) + 1 = -0.928$$

$$a^2 = \text{tansig}(n^2) = \frac{\exp(n^2) - \exp(-n^2)}{\exp(n^2) + \exp(-n^2)} = \frac{\exp(-0.928) - \exp(0.928)}{\exp(-0.928) + \exp(0.928)} = -0.7297$$

The error is:

$$e = t - a^2 = (1 - (-0.7297)) = 1.7297$$



## Exercise-08 Solution

We can now perform the backpropagation. The starting point is found at the second layer

$$s^2 = -2\mathbf{F}'^2(\mathbf{n}^2)(\mathbf{t} - \mathbf{a}) = -2[1 - (a^2)^2](e) = -2[1 - (-0.7297)^2]1.7297 = -1.6175$$

The first layer sensitivity is then computed by backpropagating the sensitivity from the second layer

$$s^1 = \mathbf{F}'^1(\mathbf{n}^1)(\mathbf{W}^2)^T s^2 = [1 - (a^1)^2]w^2 s^2 = [1 - (0.964)^2](-2)(-1.6175) = 0.2285$$

Finally, weight update takes place as follows:

$$w^2(1) = w^2(0) - \alpha s^2 (a^1)^T = (-2) - 1(-1.6175)(0.964) = -0.4407$$

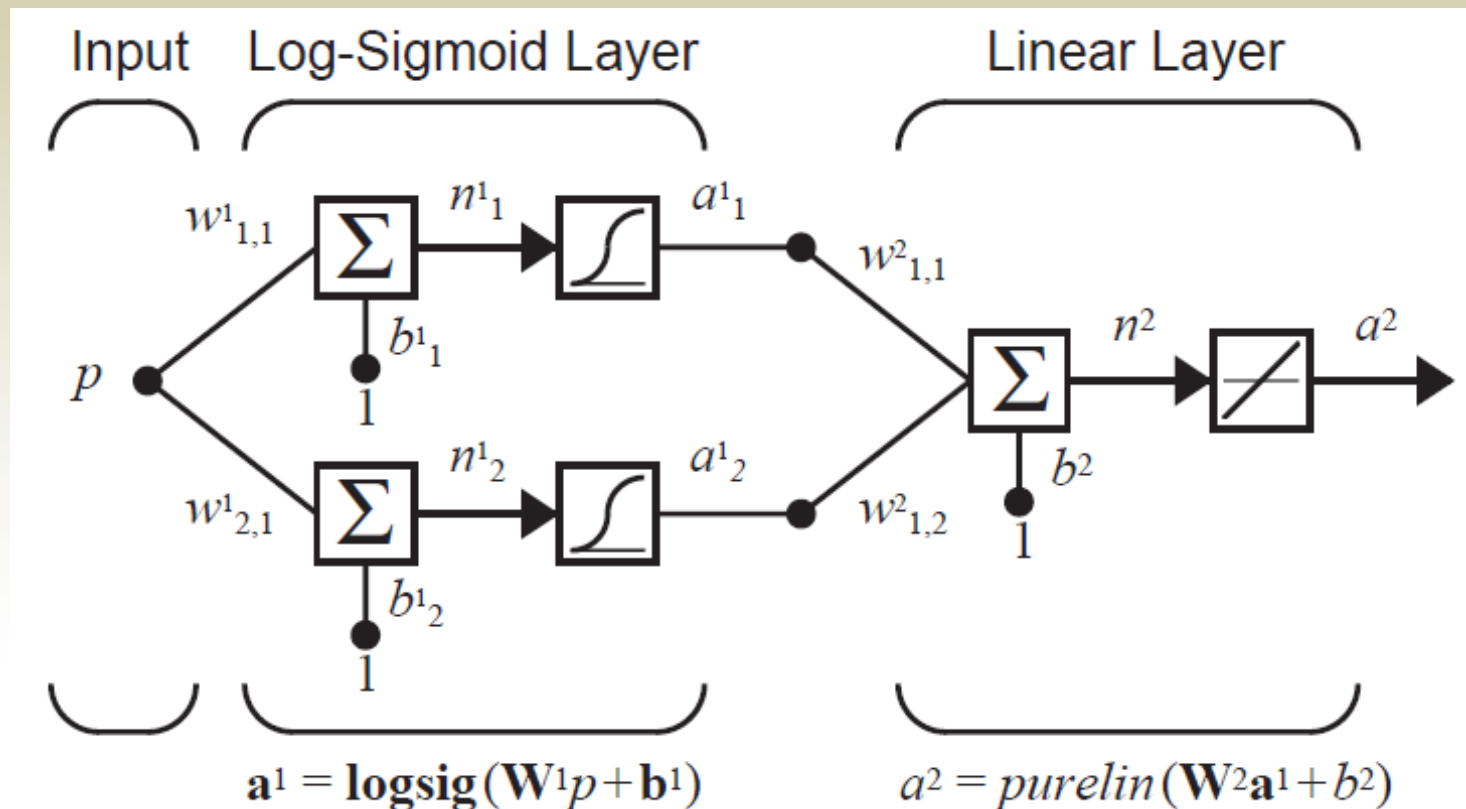
$$b^2(1) = b^2(0) - \alpha s^2 = 1 - 1(-1.6175) = 2.6175$$

$$w^1(1) = w^1(0) - \alpha s^1 (a^0)^T = (-1) - 1(0.2285)(-1) = -0.7715$$

$$b^1(1) = b^1(0) - \alpha s^1 = 1 - 1(0.2285) = 0.7715$$

## Exercise-09

Apply (one step of) backpropagation in the following 1-2-1 neural network in order to approximate the following function:

$$g(p) = 1 + \sin\left(\frac{\pi}{4}p\right) \quad \text{for } -2 \leq p \leq 2$$


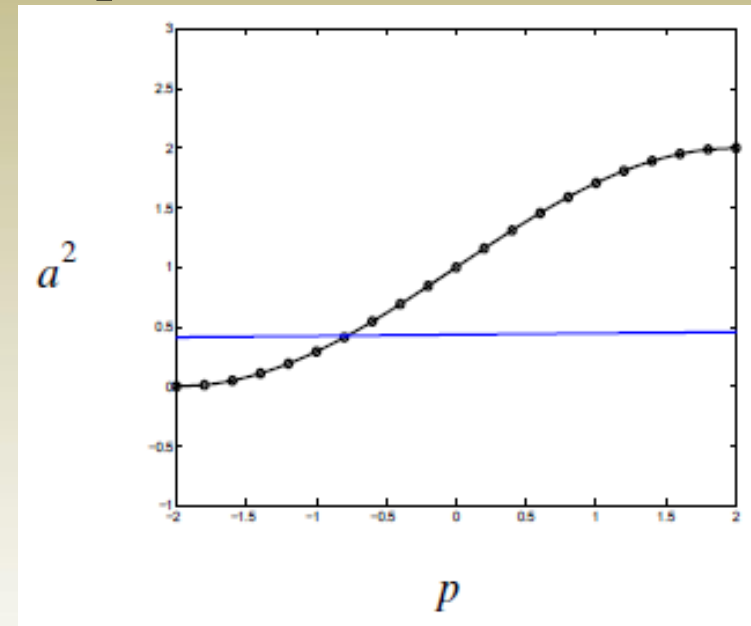


## Exercise-09

Apparently we need several training points (input-target), but for our single step let us work with  $p=1$  and initial weights and biases the following:

$$\mathbf{W}^1(0) = \begin{bmatrix} -0.27 \\ -0.41 \end{bmatrix} \quad \mathbf{W}^2(0) = \begin{bmatrix} 0.09 & -0.17 \end{bmatrix}$$

$$\mathbf{b}^1(0) = \begin{bmatrix} -0.48 \\ -0.13 \end{bmatrix} \quad \mathbf{b}^2(0) = \begin{bmatrix} 0.48 \end{bmatrix}$$



Generally initial weights and biases are chosen to be small random values. We will explain why in subsequent lectures.

Learning rate=  $\alpha=0.1$





## Exercise-09 Solution

The output of the first layer is then:

$$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{W}^1 \mathbf{a}^0 + \mathbf{b}^1) = \text{logsig}\left(\begin{bmatrix} -0.27 \\ -0.41 \end{bmatrix} [1] + \begin{bmatrix} -0.48 \\ -0.13 \end{bmatrix}\right) =$$


$$\text{logsig}\left(\begin{bmatrix} -0.75 \\ -0.54 \end{bmatrix}\right) = \begin{bmatrix} \frac{1}{1+e^{0.75}} \\ \frac{1}{1+e^{0.54}} \end{bmatrix} = \begin{bmatrix} 0.321 \\ 0.368 \end{bmatrix}$$

The output of the second layer is then:

$$a = f^2(\mathbf{W}^2 \mathbf{a}^1 + \mathbf{b}^2) = \text{purelin}([0.09 \quad -0.17] \begin{bmatrix} 0.321 \\ 0.368 \end{bmatrix} + [0.48]) = [0.446]$$

The error is:

$$e = t - a = \left\{ 1 + \sin\left(\frac{\pi}{4}p\right) \right\} - a^2 = 1 + \sin\left(\frac{\pi}{4}1\right) - 0.446 = 1.261$$



## Exercise-09 Solution

The next stage of the algorithm is to backpropagate the sensitivities. But before of that, we will need the derivatives of the transfer functions.

For the first layer:

$$f'^1(n) = \frac{d}{dn} \left( \frac{1}{1 + e^{-n}} \right) = \frac{e^{-n}}{(1 + e^{-n})^2} = \left( 1 - \frac{1}{1 + e^{-n}} \right) \left( \frac{1}{1 + e^{-n}} \right) = (1 - a^1)(a^1)$$

For the second layer:

$$f'^2(n) = \frac{d}{dn}(n) = 1$$



## Exercise-09 Solution

We can now perform the backpropagation. The starting point is found at the second layer

$$s^2 = -2\mathbf{F}'^2(\mathbf{n}^2)(\mathbf{t} - \mathbf{a}) = -2[f'^2(n^2)](1.261) = -2[1](1.261) = -2.522$$

The first layer sensitivity is then computed by backpropagating the sensitivity from the second layer

$$\begin{aligned} \mathbf{s}^1 &= \mathbf{F}'^1(\mathbf{n}^1)(\mathbf{W}^2)^T \mathbf{s}^2 = \begin{bmatrix} (1 - a_1^1)(a_1^1) & 0 \\ 0 & (1 - a_2^1)(a_2^1) \end{bmatrix} \begin{bmatrix} 0.09 \\ -0.17 \end{bmatrix} [-2.522] = \\ & \begin{bmatrix} (1 - 0.321)(0.321) & 0 \\ 0 & (1 - 0.368)(0.368) \end{bmatrix} \begin{bmatrix} 0.09 \\ -0.17 \end{bmatrix} [-2.522] = \\ & \begin{bmatrix} 0.218 & 0 \\ 0 & 0.233 \end{bmatrix} \begin{bmatrix} -0.227 \\ 0.429 \end{bmatrix} = \begin{bmatrix} -0.0495 \\ 0.0997 \end{bmatrix} \end{aligned}$$



## Exercise-09 Solution

The final stage of the algorithm is to update the weights:

$$\mathbf{W}^2(1) = \mathbf{W}^2(0) - \alpha \mathbf{s}^2 (\mathbf{a}^1)^T = [0.09 \quad -0.17] - 0.1[-2.522][0.321 \quad 0.368] = [0.171 \quad -0.0772]$$

$$\mathbf{b}^2(1) = \mathbf{b}^2(0) - \alpha \mathbf{s}^2 = [0.48] - 0.1[-2.522] = [0.732]$$

$$\mathbf{W}^1(1) = \mathbf{W}^1(0) - \alpha \mathbf{s}^1 (\mathbf{a}^0)^T = \begin{bmatrix} -0.27 \\ -0.41 \end{bmatrix} - 0.1 \begin{bmatrix} -0.0495 \\ -0.0997 \end{bmatrix} [1] = \begin{bmatrix} -0.265 \\ -0.420 \end{bmatrix}$$

$$\mathbf{b}^1(1) = \mathbf{b}^1(0) - \alpha \mathbf{s}^1 = \begin{bmatrix} -0.48 \\ -0.13 \end{bmatrix} - 0.1 \begin{bmatrix} -0.0495 \\ -0.0997 \end{bmatrix} = \begin{bmatrix} -0.475 \\ -0.140 \end{bmatrix}$$



Some issues related to the  
practical implementation of  
backpropagation



# Choice of network architecture

We want to approximate the following functions:

$$g(p) = 1 + \sin\left(\frac{i\pi}{4}p\right) \quad \text{for } -2 \leq p \leq 2$$

where  $i$  takes on the values 1, 2, 4 and 8.

As  $i$  is increased, the function becomes more complex, because we will have more periods of the sine wave over the interval.

For this we will use a 1-3-1 network, where the transfer function for the first layer is log-sigmoid and the transfer function for the second layer is linear.

Thus, this type of two-layer network can produce a response that is a sum of three log-sigmoid functions

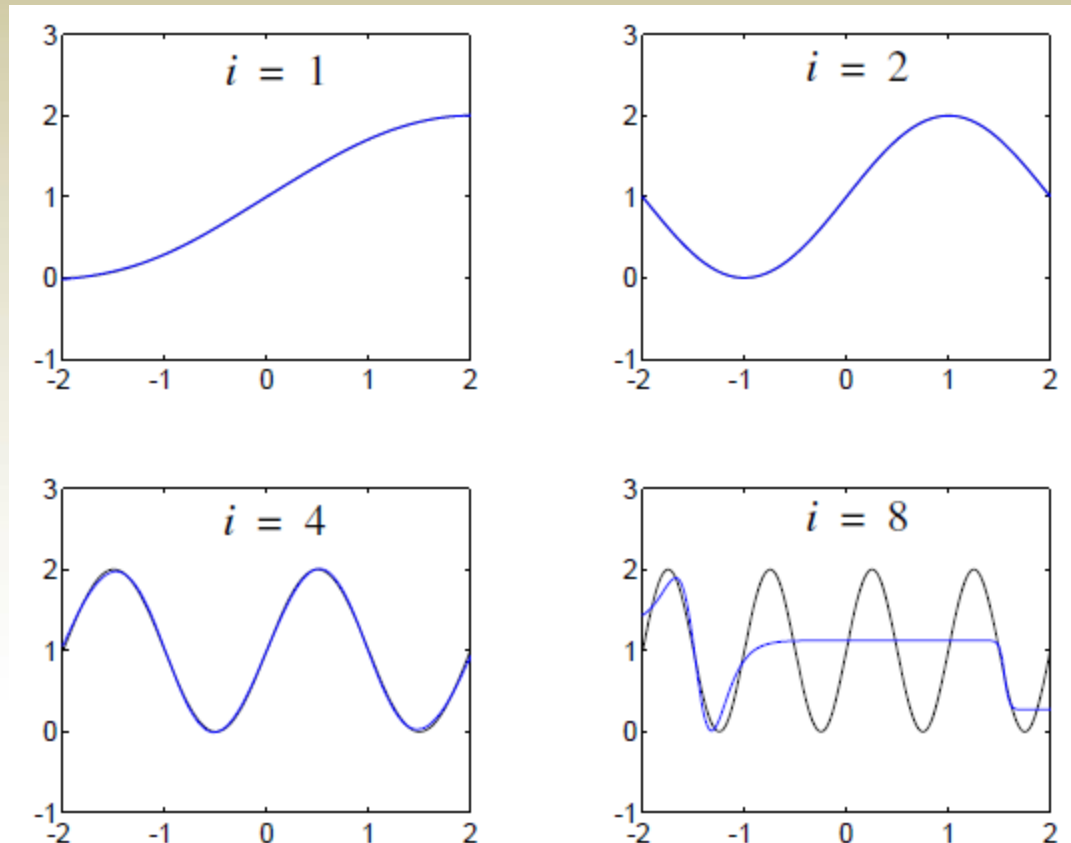
# Choice of network architecture

The response of the network after it has been trained to approximate  $g(p)$  for  $i=1,2,4,8$ . The final network responses are shown by the blue lines

We can see that for  $i=4$  the 1-3-1 network reaches its maximum capability.

When  $i>4$  the network is not capable of producing an accurate approximation of  $g(p)$ .

In the bottom right graph of Figure, we can see how the 1-3-1 network attempts to approximate  $g(p)$  for  $i=8$ . The mean square error between the network response and  $g(p)$  is minimized, but the network response is only able to match a small part of the function







# Choice of network architecture

Now, we will approach the problem from a slightly different perspective: we will pick one function  $g(p)$  and then use larger and larger networks until we are able to accurately represent the function. For  $g(p)$  we will use

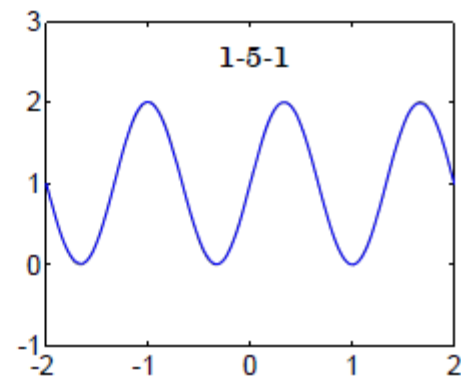
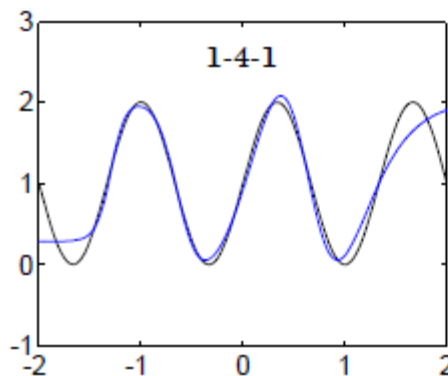
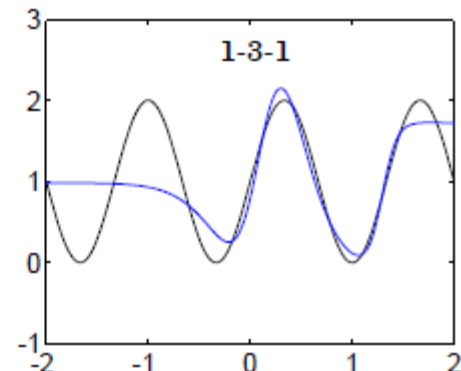
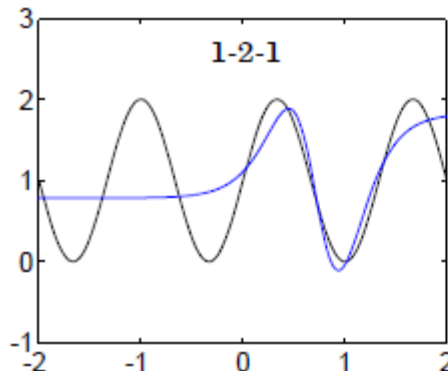
$$g(p) = 1 + \sin\left(\frac{6\pi}{4}p\right) \quad \text{for } -2 \leq p \leq 2$$

To approximate this function we will use two-layer networks, where the transfer function for the first layer is log-sigmoid and the transfer function for the second layer is linear

Again, the response of this network is a superposition of sigmoid functions.

# Choice of network architecture

Here we see network response as the number of neurons in the first layer (hidden layer) is increased. Unless there are at least five neurons in the hidden layer the network cannot accurately represent  $g(p)$ .





# Generalization

In most cases the multilayer network is trained with a finite number of examples of proper network behavior.

This training set is normally representative of a much larger class of possible input/output pairs. It is important that the network successfully generalize what it has learned to the total population.

Suppose that the training set is obtained by sampling the following function:

$$g(p) = 1 + \sin\left(\frac{\pi}{4}p\right)$$

at the points  $p = -2, -1.6, -1.2, \dots, 1.6, 2$ . (There are a total of 11 input/target pairs.)

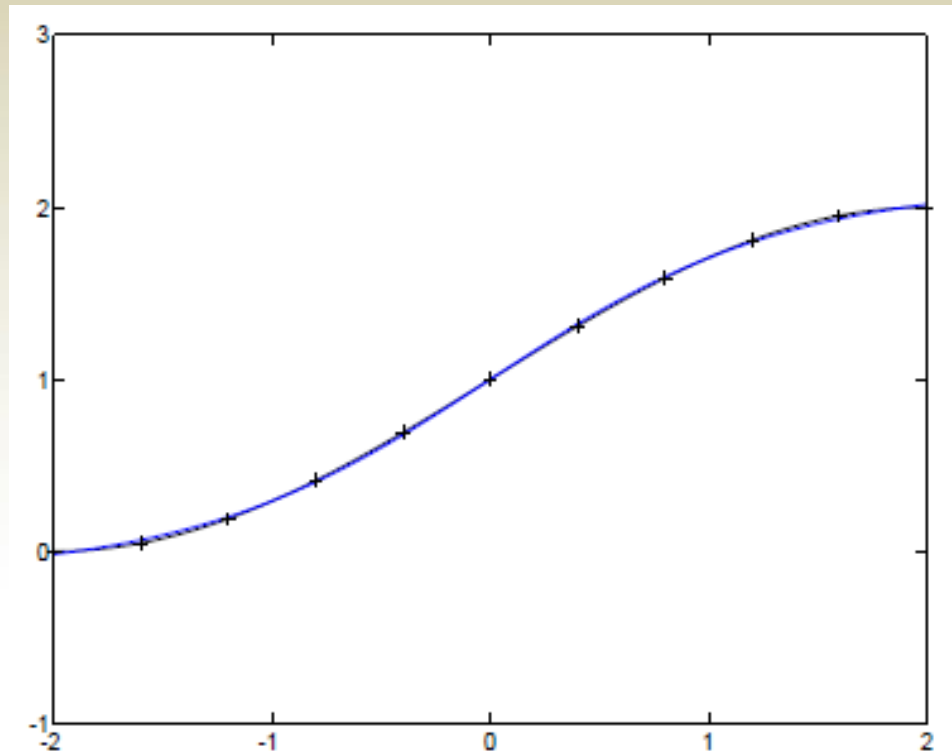
# Generalization

We see the response of a 1-2-1 network that has been trained on this data. The black line represents  $g(p)$ , the blue line represents the network response, and the '+' symbols indicate the training set.

We can see that the network response is an accurate representation of  $g(p)$

If we were to find the response of the network at a value of  $p$  that was not contained in the training set (e.g.,  $p=-0.2$ ), the network would still produce an output close to  $g(p)$ .

This network generalizes well.



# Generalization

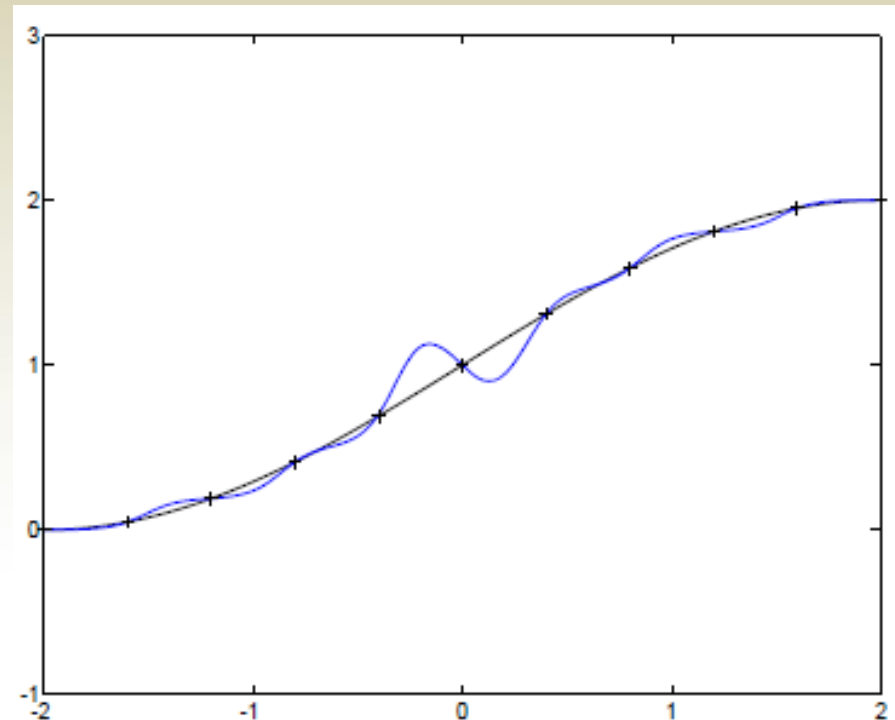
Now consider this figure, which shows the response of a 1-9-1 network that has been trained on the same data set.

Note that the network response accurately models  $g(p)$  at all of the training points.

However, if we compute the network response at a value of  $p$  not contained in the training set (e.g.,  $p=-0.2$ ) the network might produce an output far from the true response.

This network does not generalize well.

The 1-9-1 network has too much flexibility for this problem; it has a total of 28 adjustable parameters (18 weights and 10 biases), and yet there are only 11 data points in the training set. The 1-2-1 network has only 7 parameters.





# Generalization

*For a network to be able to generalize, it should have fewer parameters than there are data points in the training set.*

In neural networks, as in all modeling problems, we want to use the simplest network that can adequately represent the training set. Don't use a bigger network when a smaller network will work (a concept often referred to as Ockham's Razor).

An alternative to using the simplest network is to stop the training before the network overfits (see subsequent lectures)



# Choice of hyperparameter

Hyper-parameter	Increases capacity when ...	Reason	Caveats
Number of hidden units	increased	Increasing the number of hidden units increases the representational capacity of the model	Increasing the number of hidden units increases both the time and memory cost of essentially every operation on the model
Learning rate	tuned optimally	An improper learning rate, whether too high or too low, results in a model with low effective capacity due to optimization failure	





# Choice of hyperparameter

Hyper-parameter	Increases capacity when ...	Reason	Caveats
Convolution kernel width	increased	Increasing the kernel width increases the number of parameters in the model	A wider kernel results in a narrower output dimension, reducing model capacity unless you use implicit zero padding to reduce this effect. Wider kernels require more memory for parameter storage and increase runtime, but a narrower output reduces memory cost



# Choice of hyperparameter

Hyper-parameter	Increases capacity when ...	Reason	Caveats
Implicit zero padding	increased	Adding implicit zeros before convolution keeps the representation size large	Increased time and memory cost of most operations
Weight decay coefficient	decreased	Decreasing the weight decay coefficient frees the model parameters to become larger	
Dropout rate	decreased	Dropping units less often gives the units more opportunities to “conspire” with each other to fit the training set	