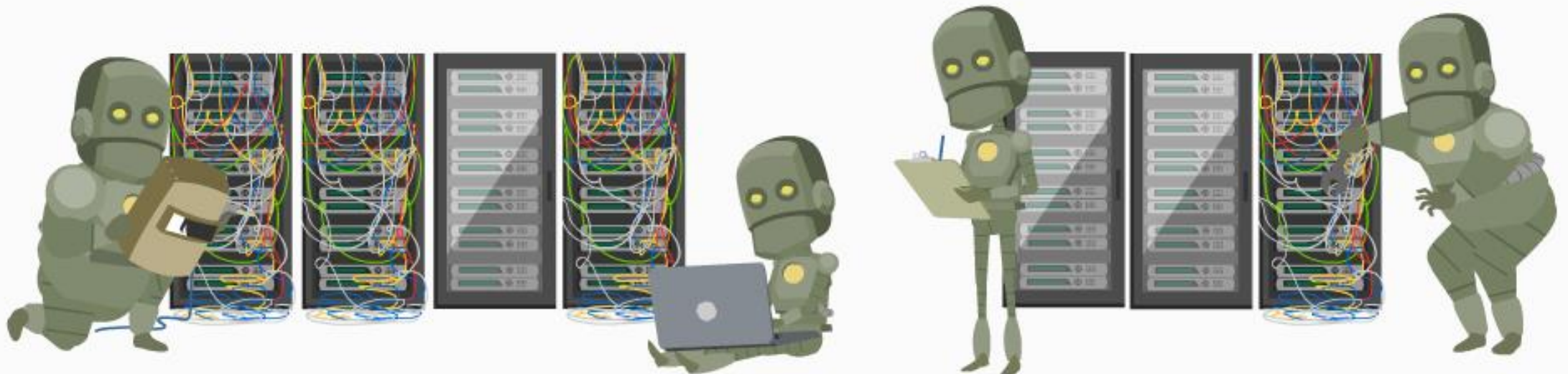


MACHINE LEARNING

COURSE NOTES – SECTION 6

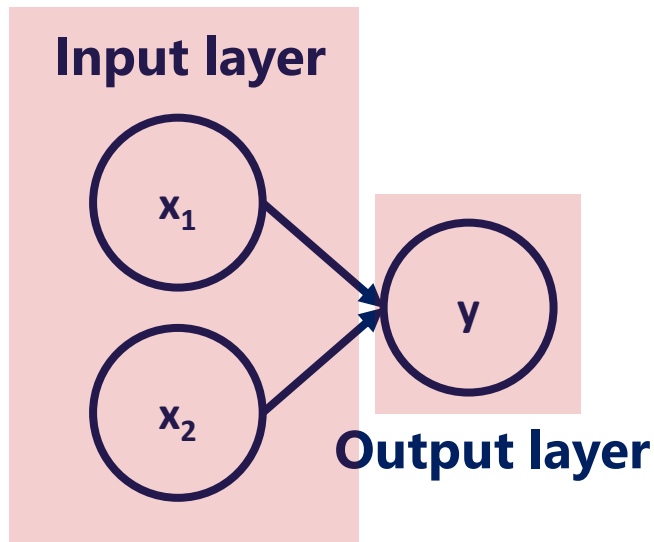


IT'S TIME TO DIG DEEPER

Layers

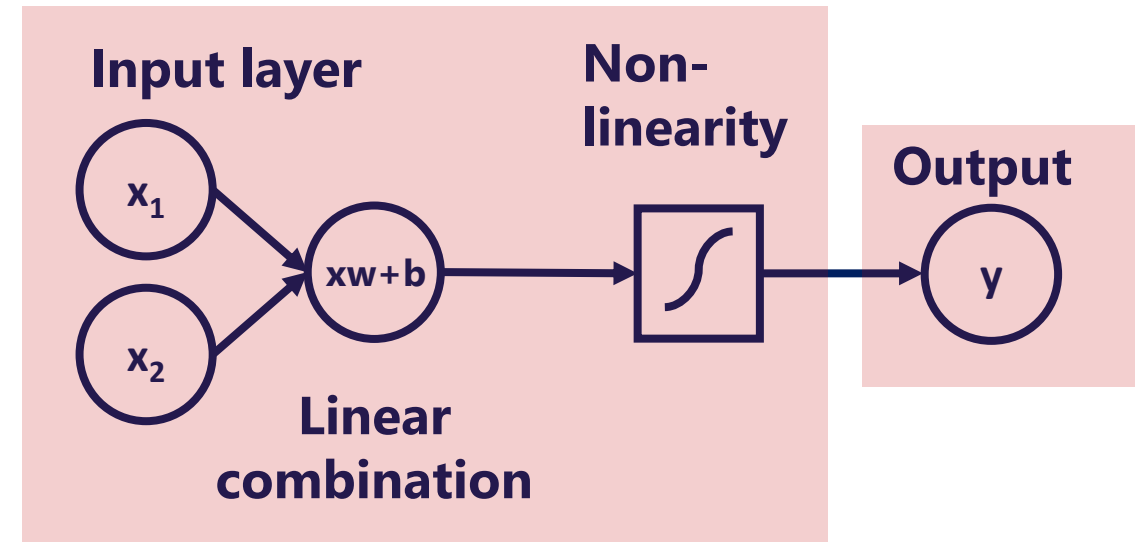
An initial linear combination and the added non-linearity form a **layer**. The layer is the building block of neural networks.

Minimal example (a simple neural network)



In the minimal example we trained a *neural network* which had no depth. There were solely an input layer and an output layer. Moreover, the output was simply a **linear combination** of the input.

Neural networks



Neural networks step on linear combinations, but add a non-linearity to each one of them. Mixing linear combinations and non-linearities allows us to model arbitrary functions.

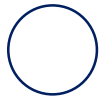
A deep net

This is a deep neural network (deep net) with 5 layers.

How to read this diagram:



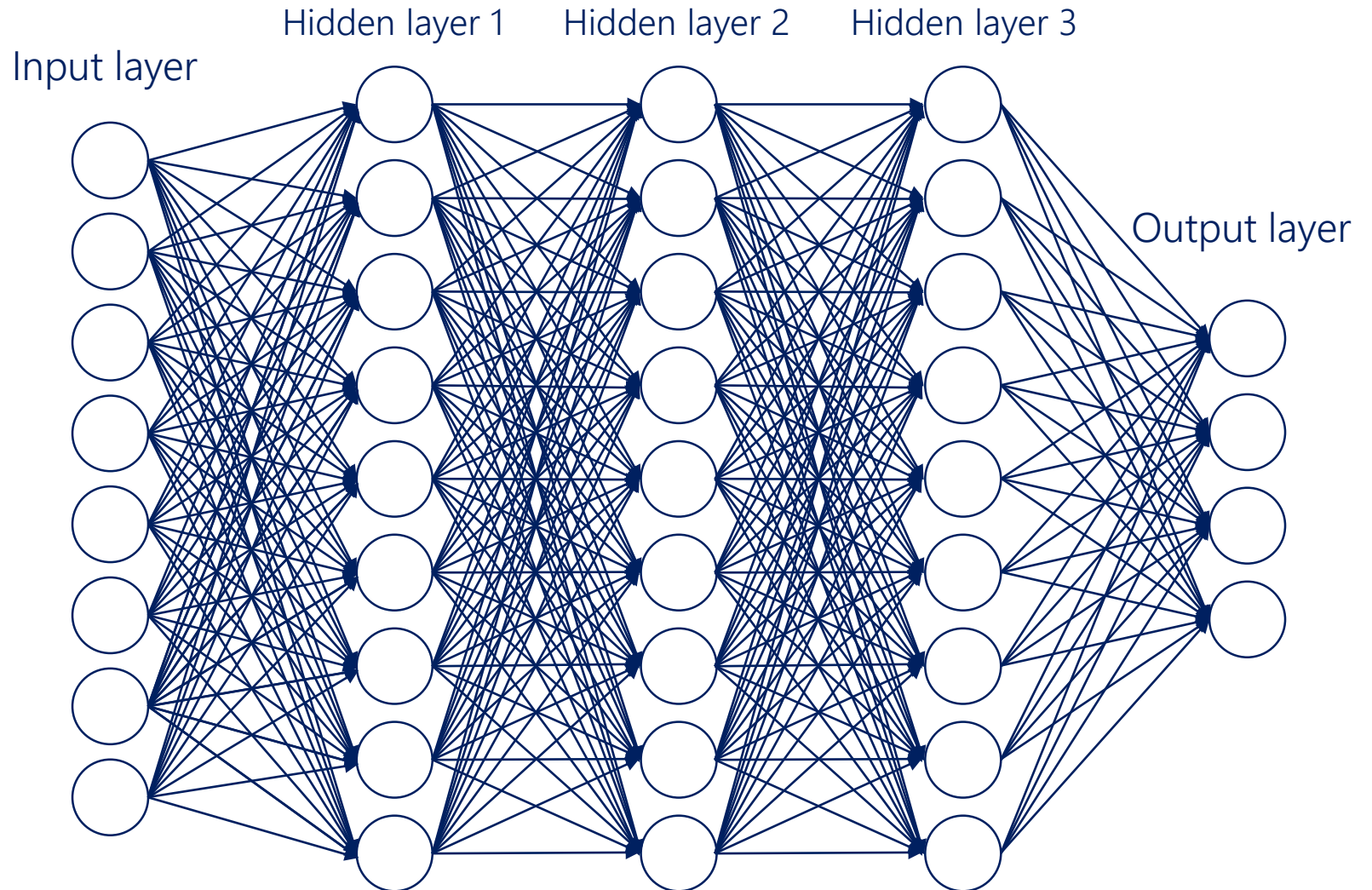
A layer



A unit (a neuron)



Arrows represent
mathematical transformations



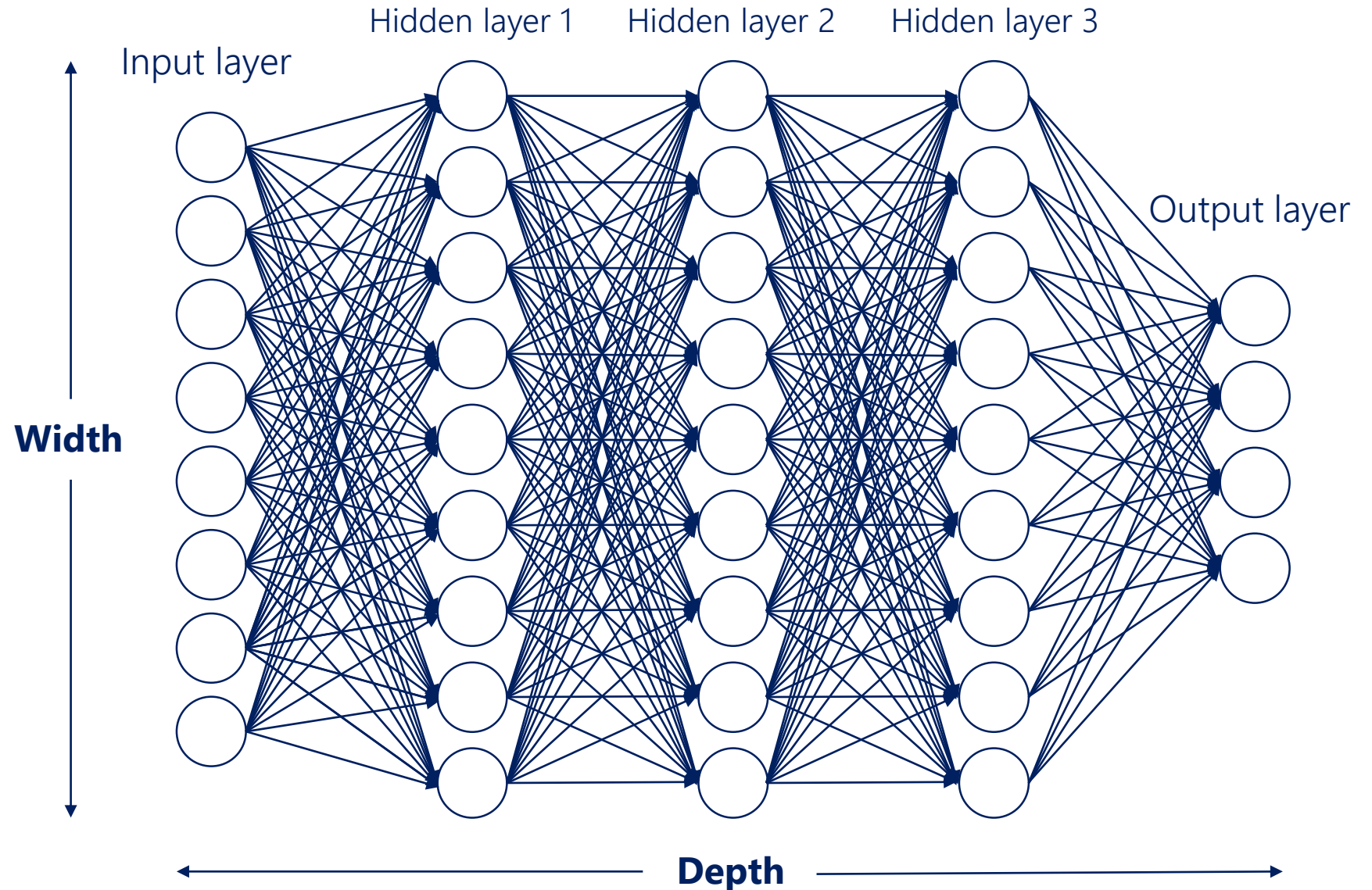
A deep net

The **width** of a layer is the number of units in that layer

The **width** of the net is the number of units of the biggest layer

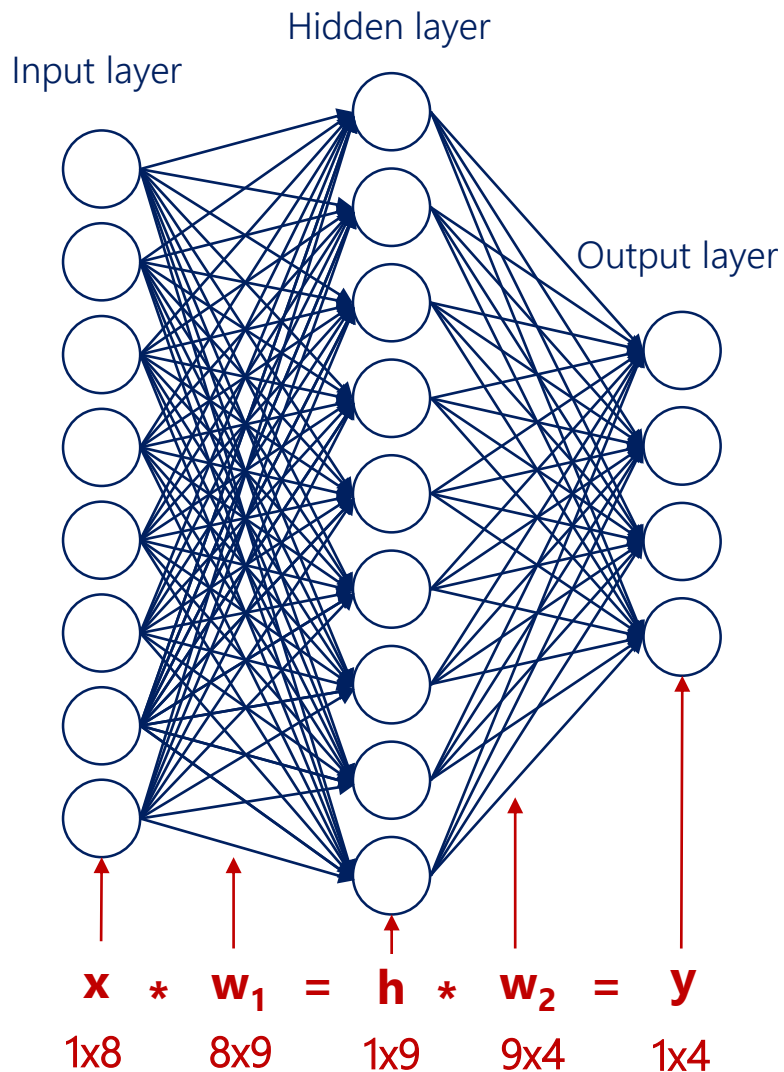
The **depth** of the net is equal to the number of layers or the number of hidden layers. The term has different definitions. More often than not, we are interested in the number of hidden layers (as there are always input and output layers).

The width and the depth of the net are called **hyperparameters**. They are values we manually chose when creating the net.



Why we need non-linearities to stack layers

You can see a net with no non-linearities: just linear combinations.



$$\mathbf{h} = \mathbf{x} * \mathbf{w}_1$$

$$\mathbf{y} = \mathbf{h} * \mathbf{w}_2$$

$$\mathbf{y} = \mathbf{x} * \mathbf{w}_1 * \mathbf{w}_2$$

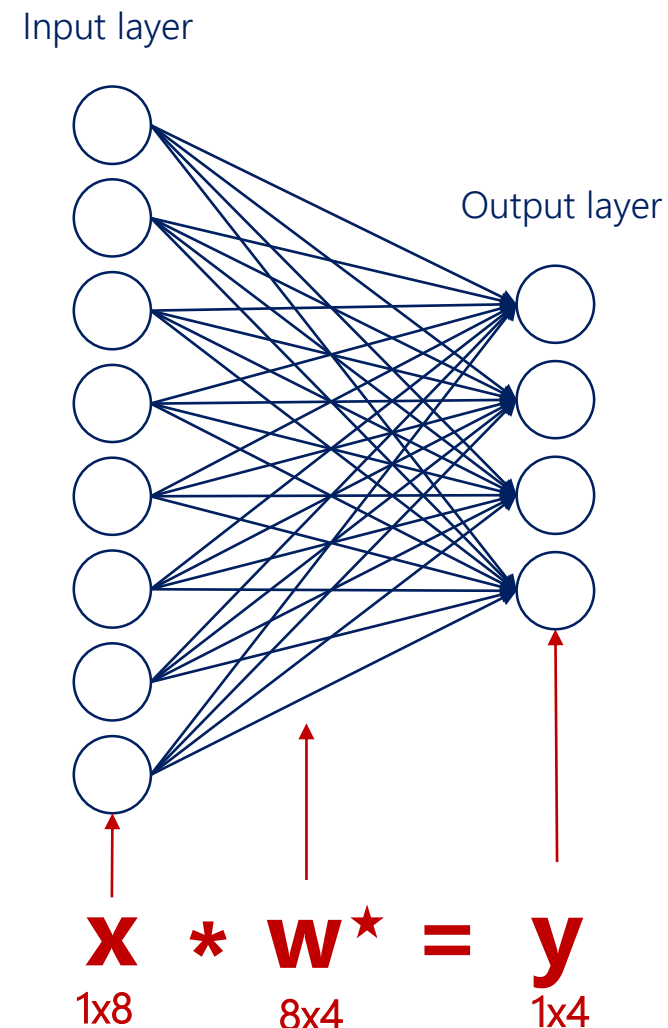
8x9 9x4

$$\mathbf{y} = \mathbf{x} * \mathbf{w}^*$$

8x4

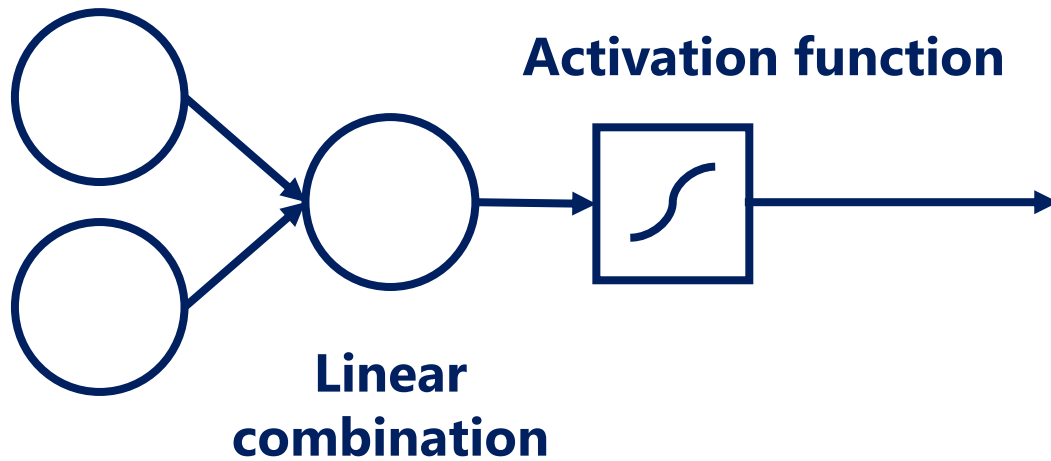
Two consecutive linear transformations are equivalent to a single one.

Two consecutive linear transformations are equivalent to a single one.



Activation functions

Input



Activation functions (non-linearities) are needed so we can break the linearity and represent more complicated relationships.

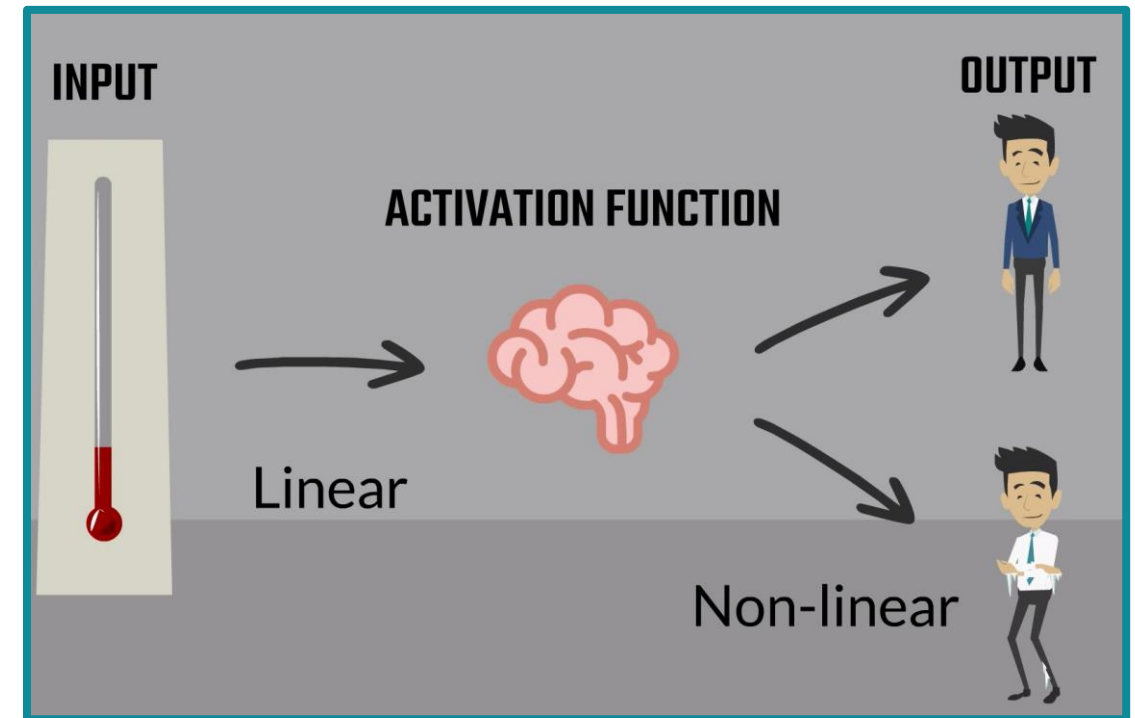
Moreover, activation functions are required in order to **stack layers**.

Activation functions transform inputs into outputs of a different kind.

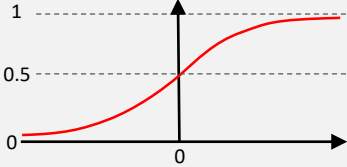
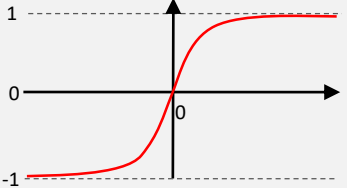
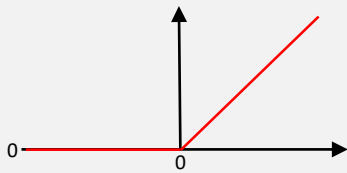
In the respective lesson, we gave an example of temperature change. The temperature starts decreasing (which is a numerical change). Our brain is a kind of an 'activation function'. It tells us whether it is **cold enough** for us to put on a jacket.

Putting on a jacket is a binary action: 0 (no jacket) or 1 (jacket).

This is a very intuitive and visual (yet not so practical) example of how activation functions work.



Common activation functions

Name	Formula	Derivative	Graph	Range
sigmoid (logistic function)	$\sigma(a) = \frac{1}{1+e^{-a}}$	$\frac{\partial \sigma(a)}{\partial a} = \sigma(a)(1 - \sigma(a))$		(0,1)
TanH (hyperbolic tangent)	$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$	$\frac{\partial \tanh(a)}{\partial a} = \frac{4}{(e^a + e^{-a})^2}$		(-1,1)
ReLu (rectified linear unit)	$\text{relu}(a) = \max(0, a)$	$\frac{\partial \text{relu}(a)}{\partial a} = \begin{cases} 0, & \text{if } a \leq 0 \\ 1, & \text{if } a > 0 \end{cases}$		(0,∞)
softmax	$\sigma_i(a) = \frac{e^{a_i}}{\sum_j e^{a_j}}$	$\frac{\partial \sigma_i(a)}{\partial a_j} = \sigma_i(a) (\delta_{ij} - \sigma_j(a))$ Where δ_{ij} is 1 if $i=j$, 0 otherwise		(0,1)

All common activation functions are: **monotonic**, **continuous**, and **differentiable**. These are important properties needed for the optimization.

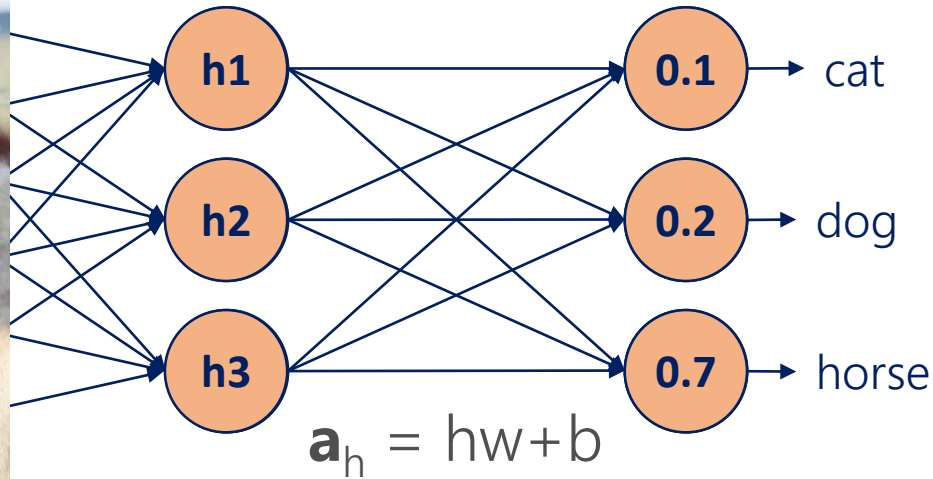
Softmax activation

Input layer



Hidden layer

Output layer



The softmax activation transforms a bunch of arbitrarily large or small numbers into a valid probability distribution.

While other activation functions get an input value and transform it, regardless of the other elements, the softmax considers the information about the **whole set of numbers** we have.

The values that softmax outputs are in the range from 0 to 1 and their sum is exactly 1 (like probabilities).

Example:

$$\mathbf{a} = [-0.21, 0.47, 1.72]$$

$$\text{softmax}(\mathbf{a}) = \frac{e^{a_i}}{\sum_j e^{a_j}}$$

$$\sum_j e^{a_j} = e^{-0.21} + e^{0.47} + e^{1.72} = 8$$

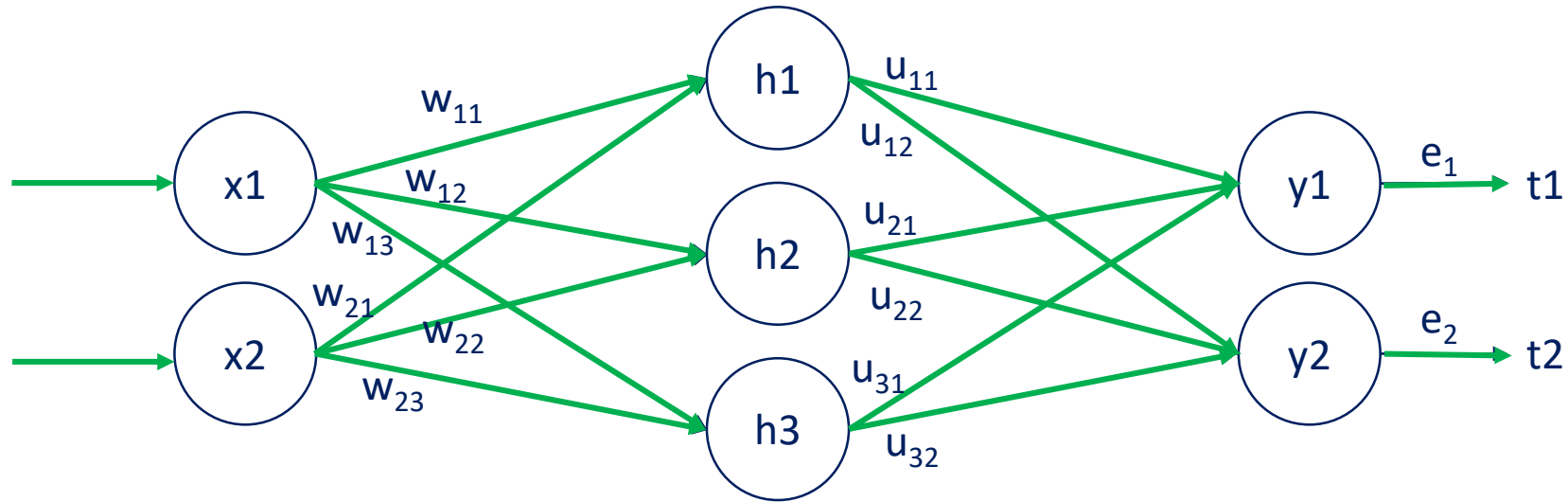
$$\text{softmax}(\mathbf{a}) = \left[\frac{e^{-0.21}}{8}, \frac{e^{0.47}}{8}, \frac{e^{1.72}}{8} \right]$$

$$\mathbf{y} = [0.1, 0.2, 0.7] \rightarrow \text{probability distribution}$$

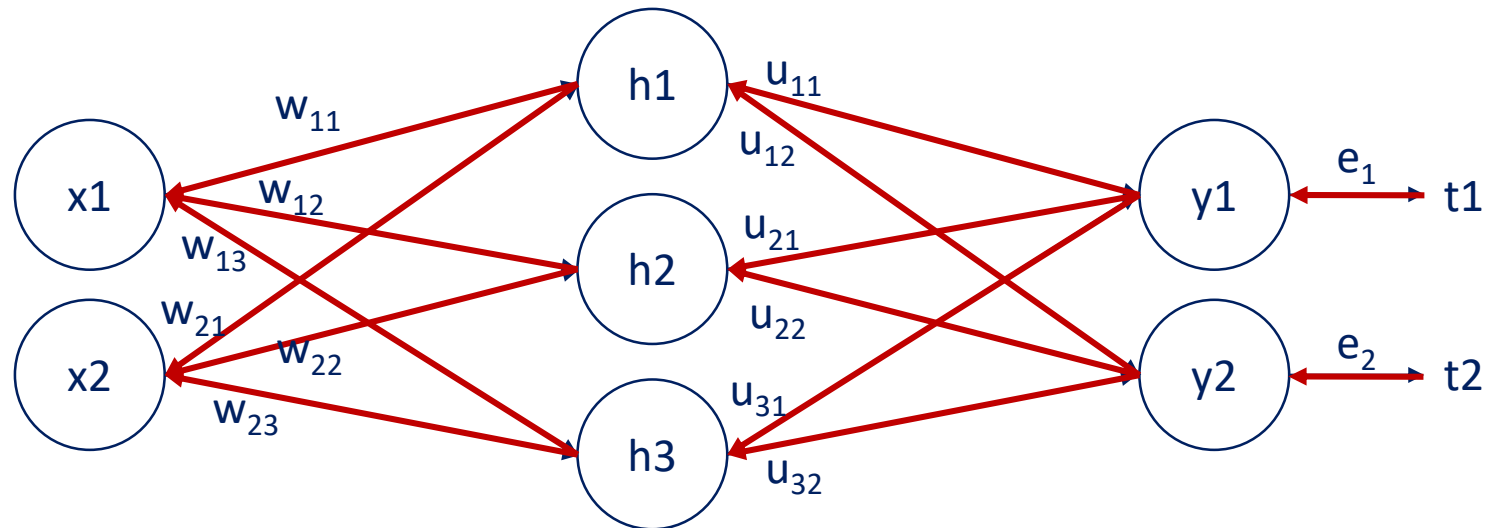
The property of the softmax to output probabilities is so useful and intuitive that it is often used as the activation function for the **final (output) layer**.

However, when the softmax is used prior to that (as the activation of a hidden layer), the results are not as satisfactory. That's because a lot of the information about the variability of the data is lost.

Backpropagation

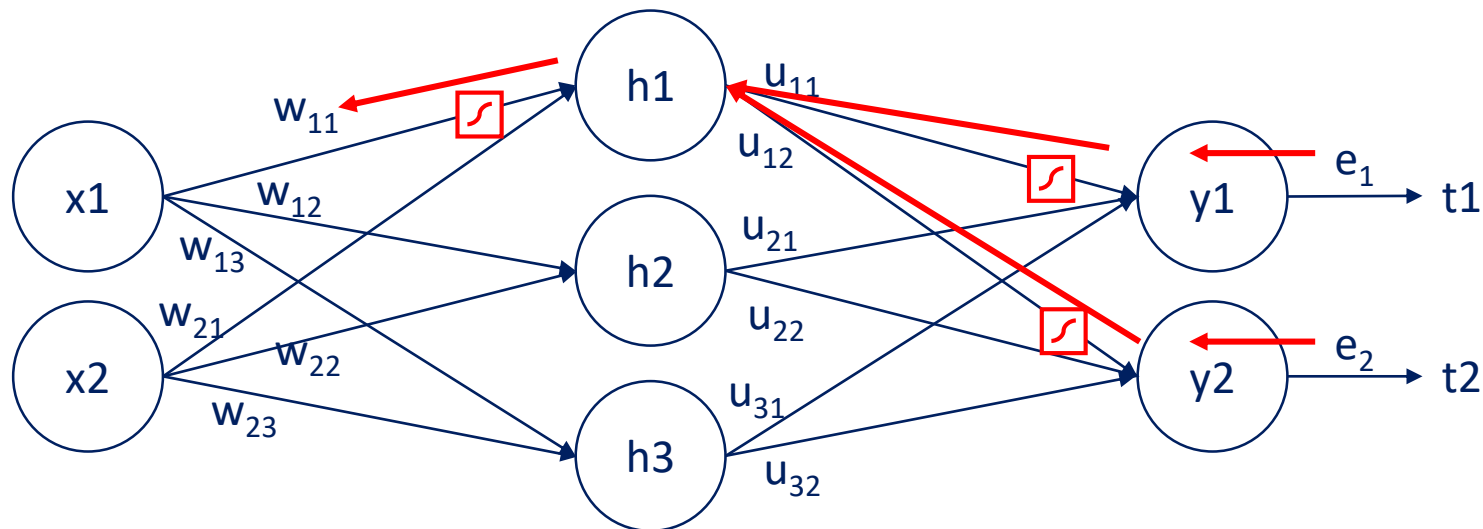


Forward propagation is the process of pushing inputs through the net. At the end of each epoch, the obtained outputs are compared to targets to form the errors.



Backpropagation of errors is an **algorithm** for neural networks using gradient descent. It consists of calculating the contribution of each **parameter** to the errors. We backpropagate the **errors** through the net and **update** the parameters (weights and biases) accordingly.

Backpropagation formula



$$\frac{\partial L}{\partial w_{ij}} = \delta_j x_i, \text{ where } \delta_j = \sum_k \delta_k w_{jk} y_j (1 - y_j)$$

If you want to examine the full derivation, please make use of the PDF we made available in the section: **Backpropagation. A peek into the Mathematics of Optimization.**

Backpropagation. A Peek into the Mathematics of Optimization



1 Motivation

In order to get a truly deep understanding of deep neural networks, one must look at the mathematics of it. As backpropagation is at the core of the optimization process, we wanted to introduce you to it. This is definitely not a necessary part of the course, as in TensorFlow, sk-learn, or any other machine learning package (as opposed to simply NumPy), will have backpropagation methods incorporated.

2 The specific net and notation we will examine

Here's our simple network:

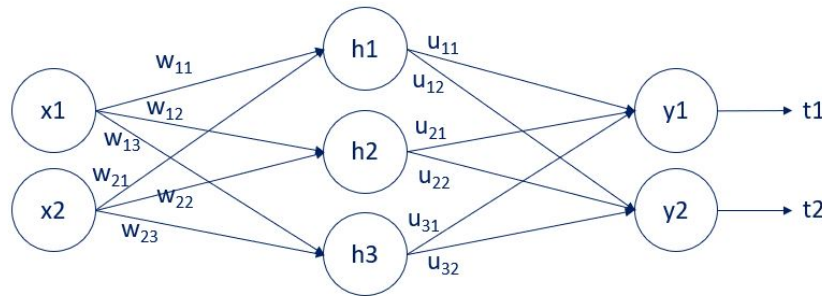


Figure 1: Backpropagation

We have two inputs: x_1 and x_2 . There is a single hidden layer with 3 units (nodes): h_1 , h_2 , and h_3 . Finally, there are two outputs: y_1 and y_2 . The arrows that connect them are the weights. There are two weights matrices: \mathbf{w} , and \mathbf{u} . The \mathbf{w} weights connect the input layer and the hidden layer. The \mathbf{u} weights connect the hidden layer and the output layer. We have employed the letters \mathbf{w} , and \mathbf{u} , so it is easier to follow the computation to follow.

You can also see that we compare the outputs y_1 and y_2 with the targets t_1 and t_2 .

There is one last letter we need to introduce before we can get to the computations. Let a be the linear combination prior to activation. Thus, we have: $\mathbf{a}^{(1)} = \mathbf{x}\mathbf{w} + \mathbf{b}^{(1)}$ and $\mathbf{a}^{(2)} = \mathbf{h}\mathbf{u} + \mathbf{b}^{(2)}$.

Since we cannot exhaust all activation functions and all loss functions, we will focus on two of the most common. A **sigmoid** activation and an **L2-norm loss**.

With this new information and the new notation, the output y is equal to the activated linear combination. Therefore, for the output layer, we have $\mathbf{y} = \sigma(\mathbf{a}^{(2)})$, while for the hidden layer: $\mathbf{h} = \sigma(\mathbf{a}^{(1)})$.

We will examine backpropagation for the output layer and the hidden layer separately, as the methodologies differ.

3 Useful formulas

I would like to remind you that:

$$\text{L2-norm loss: } L = \frac{1}{2} \sum_i (y_i - t_i)^2$$

The sigmoid function is:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

and its derivative is:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

4 Backpropagation for the output layer

In order to obtain the update rule:

$$\mathbf{u} \leftarrow \mathbf{u} - \eta \nabla_{\mathbf{u}} L(\mathbf{u})$$

we must calculate

$$\nabla_{\mathbf{u}} L(\mathbf{u})$$

Let's take a single weight u_{ij} . The partial derivative of the loss w.r.t. u_{ij} equals:

$$\frac{\partial L}{\partial u_{ij}} = \frac{\partial L}{\partial y_j} \frac{\partial y_j}{\partial a_j^{(2)}} \frac{\partial a_j^{(2)}}{\partial u_{ij}}$$

where i corresponds to the previous layer (input layer for this transformation) and j corresponds to the next layer (output layer of the transformation). The partial derivatives were computed simply following the chain rule.

$$\frac{\partial L}{\partial y_j} = (y_j - t_j)$$

following the L2-norm loss derivative.

$$\frac{\partial y_j}{\partial a_j^{(2)}} = \sigma(a_j^{(2)})(1 - \sigma(a_j^{(2)})) = y_j(1 - y_j)$$

following the sigmoid derivative.

Finally, the third partial derivative is simply the derivative of $\mathbf{a}^{(2)} = \mathbf{h}\mathbf{u} + \mathbf{b}^{(2)}$. So,

$$\frac{\partial a_j^{(2)}}{\partial u_{ij}} = h_i$$

Replacing the partial derivatives in the expression above, we get:

$$\frac{\partial L}{\partial u_{ij}} = \frac{\partial L}{\partial y_j} \frac{\partial y_j}{\partial a_j^{(2)}} \frac{\partial a_j^{(2)}}{\partial u_{ij}} = (y_j - t_j) y_j (1 - y_j) h_i = \delta_j h_i$$

Therefore, the update rule for a single weight for the output layer is given by:

$$u_{ij} \leftarrow u_{ij} - \eta \delta_j h_i$$

5 Backpropagation of a hidden layer

Similarly to the backpropagation of the output layer, the update rule for a single weight, w_{ij} would depend on:

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial h_j} \frac{\partial h_j}{\partial a_j^{(1)}} \frac{\partial a_j^{(1)}}{\partial w_{ij}}$$

following the chain rule.

Taking advantage of the results we have so far for transformation using the sigmoid activation and the linear model, we get:

$$\frac{\partial h_j}{\partial a_j^{(1)}} = \sigma(a_j^{(1)})(1 - \sigma(a_j^{(1)})) = h_j(1 - h_j)$$

and

$$\frac{\partial a_j^{(1)}}{\partial w_{ij}} = x_i$$

The actual problem for backpropagation comes from the term $\frac{\partial L}{\partial h_j}$. That's due to the fact that there is no "hidden" target. You can follow the solution for weight w_{11} below. It is advisable to also check Figure 1, while going through the computations.

$$\begin{aligned}\frac{\partial L}{\partial h_1} &= \frac{\partial L}{\partial y_1} \frac{\partial y_1}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial h_1} + \frac{\partial L}{\partial y_2} \frac{\partial y_2}{\partial a_2^{(2)}} \frac{\partial a_2^{(2)}}{\partial h_1} = \\ &= (y_1 - t_1)y_1(1 - y_1)u_{11} + (y_2 - t_2)y_2(1 - y_2)u_{12}\end{aligned}$$

From here, we can calculate $\frac{\partial L}{\partial w_{11}}$, which was what we wanted. The final expression is:

$$\frac{\partial L}{\partial w_{11}} = [(y_1 - t_1)y_1(1 - y_1)u_{11} + (y_2 - t_2)y_2(1 - y_2)u_{12}] h_1(1 - h_1)x_1$$

The generalized form of this equation is:

$$\frac{\partial L}{\partial w_{ij}} = \sum_k (y_k - t_k)y_k(1 - y_k)u_{jk}h_j(1 - h_j)x_i$$

6 Backpropagation generalization

Using the results for backpropagation for the output layer and the hidden layer, we can put them together in one formula, summarizing backpropagation, in the presence of L2-norm loss and sigmoid activations.

$$\frac{\partial L}{\partial w_{ij}} = \delta_j x_i$$

where for a hidden layer

$$\delta_j = \sum_k \delta_k w_{jk} y_j (1 - y_j)$$

Kudos to those of you who got to the end.

Thanks for reading.