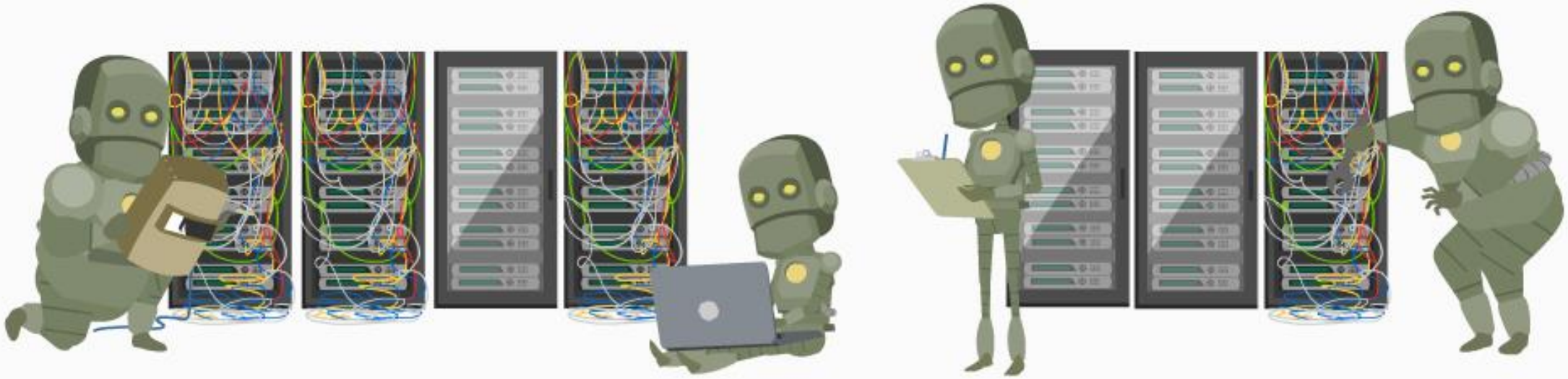
365 DataScience

MACHINE LEARNING

COURSE NOTES – SECTION 6

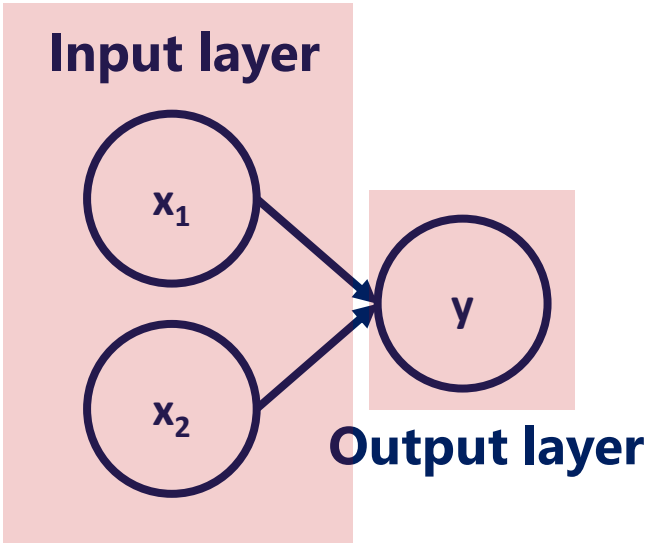


IT'S TIME TO DIG DEEPER

Layers

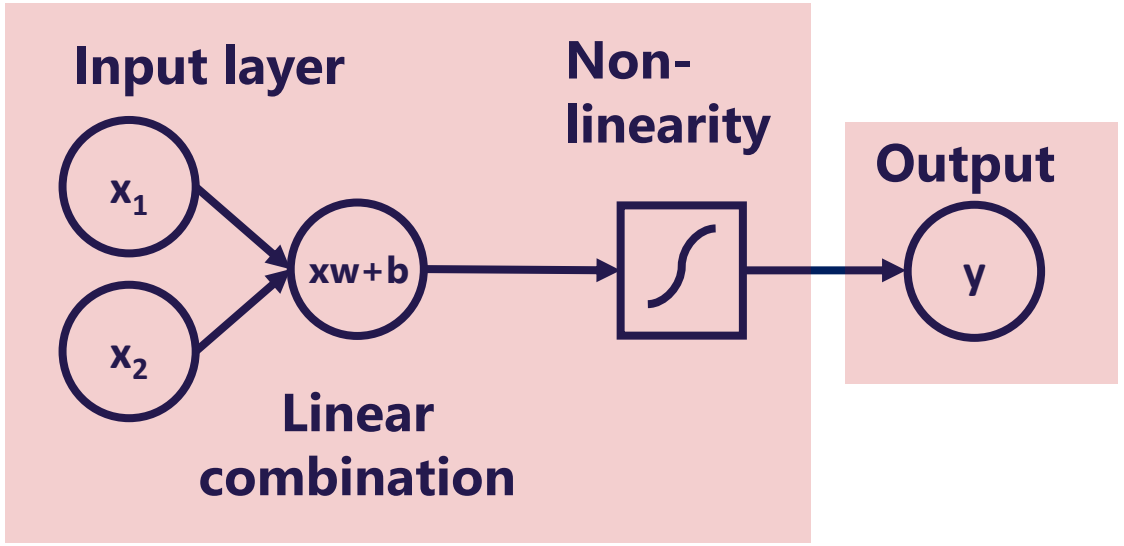
An initial linear combination and the added non-linearity form a **layer**. The layer is the building block of neural networks.

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



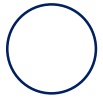
Neural networks step on linear combinations, but add a non-linearity to each one of them. Mixing linear combinations and non-linearities allows us to model arbitrary functions.

A deep net

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

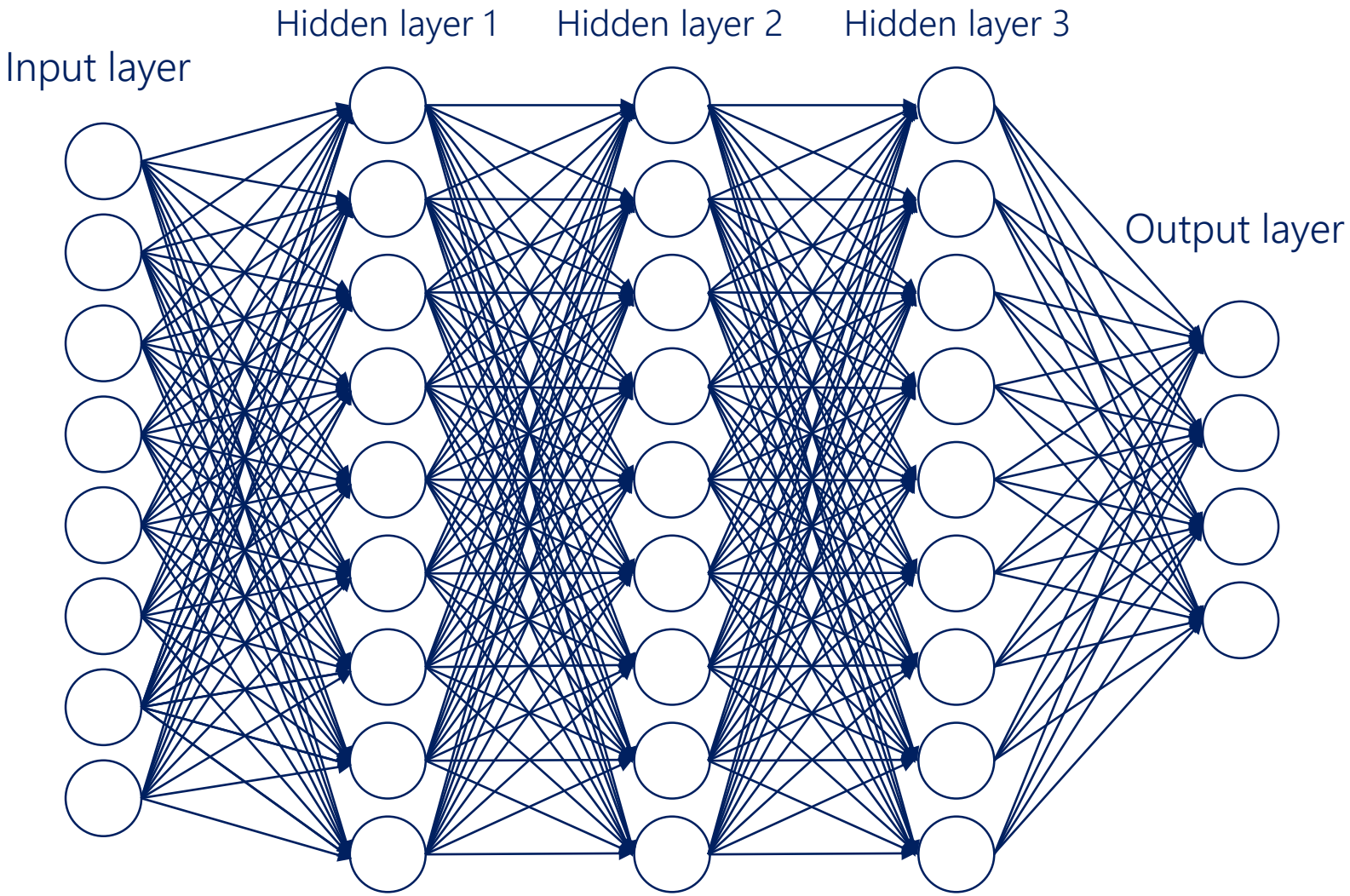
How to read this diagram:

A layer



A unit (a neuron)

Arrows represent
mathematical transformations



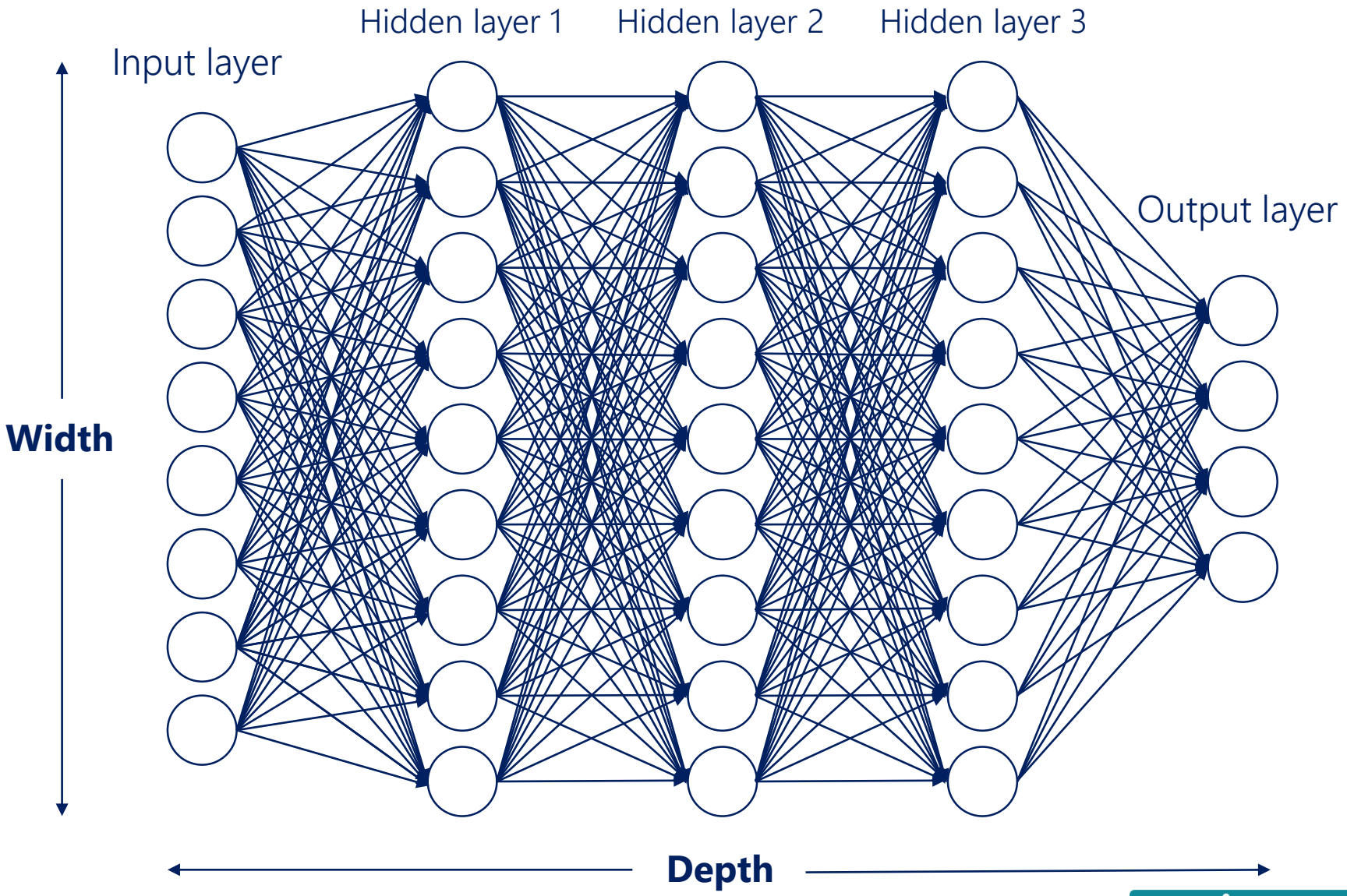
A deep net

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

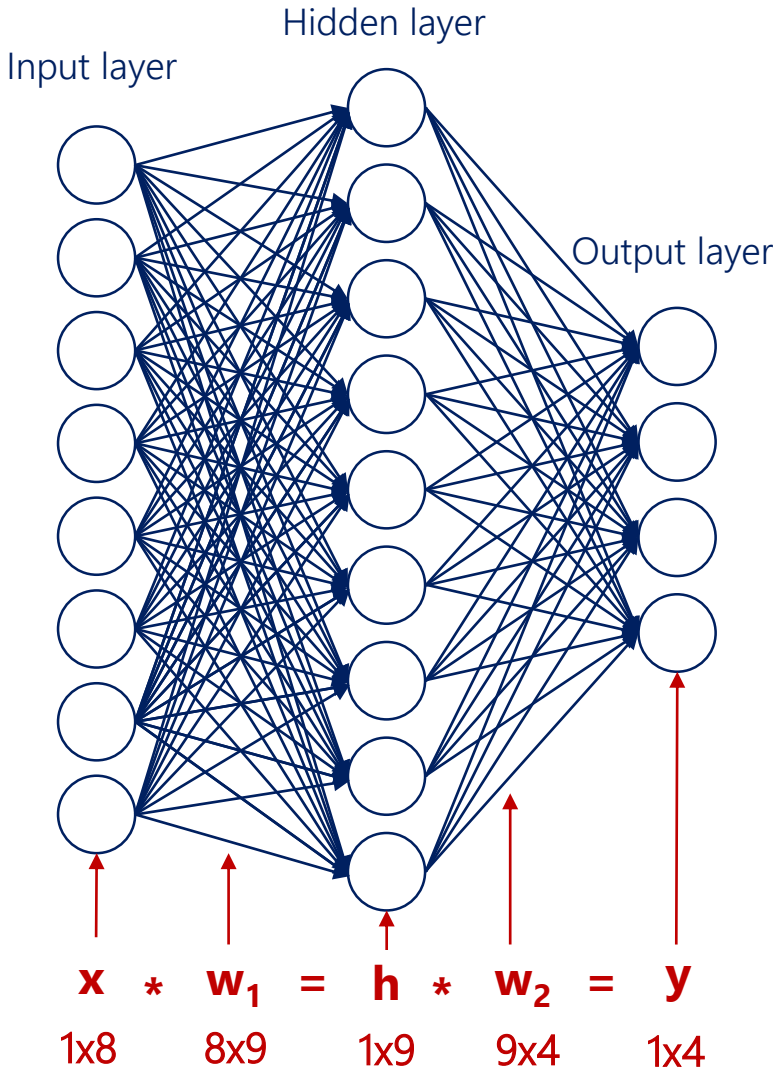
The **depth** of the net is equal to the number of layers or the number of hidden layers. The term has different definitions. More often than not, we are interested in the number of hidden layers (as there are always input and output layers).

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.



$$h = x * w_1$$

$$y = h * w_2$$

$$y = x * w_1 * w_2$$

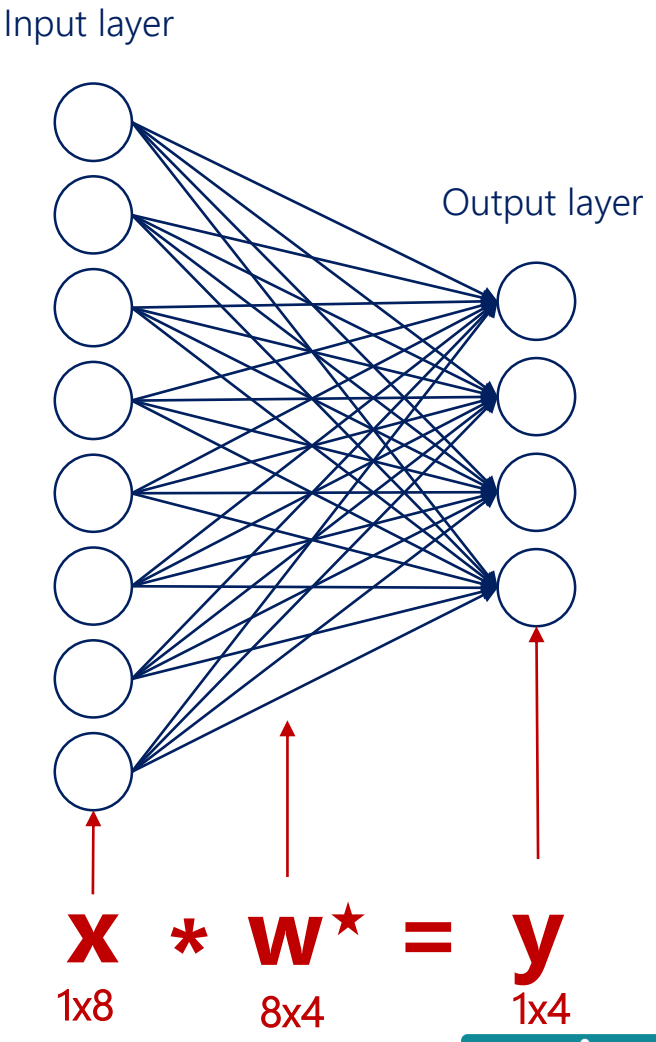
8x9 9x4

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

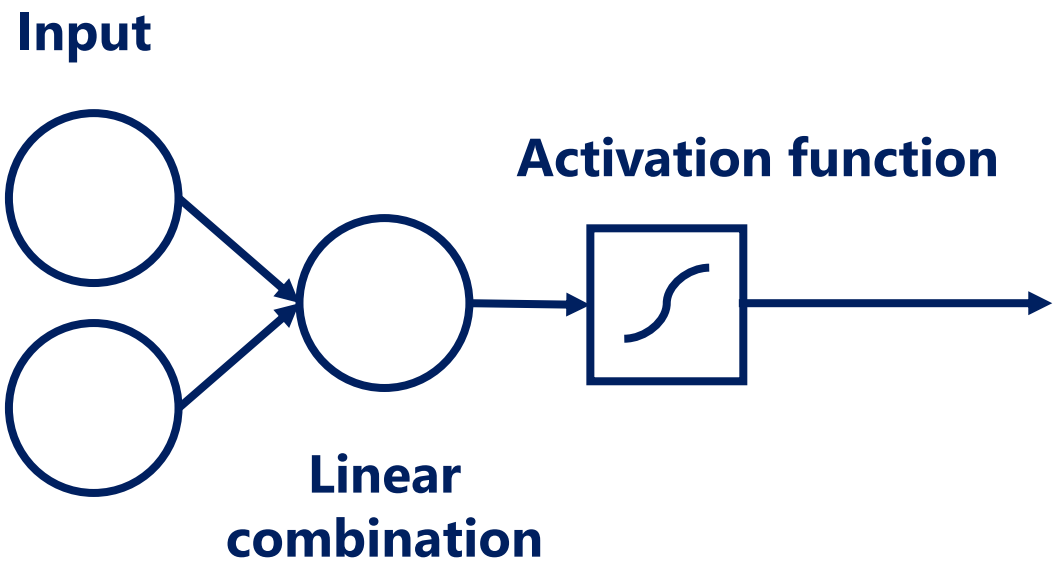
8x4

Two consecutive linear transformations are equivalent to a single one.

Two consecutive linear transformations are equivalent to a single one.



Activation functions



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.

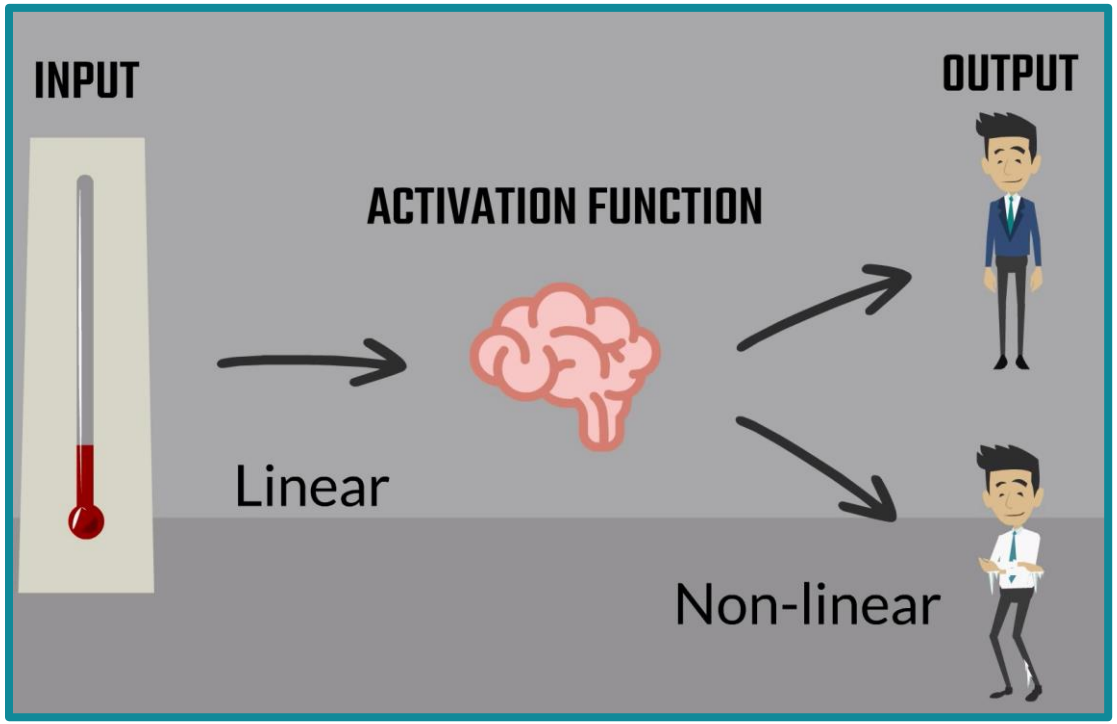
Putting on a jacket is a binary action: 0 (no jacket) or 1 (jacket).

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

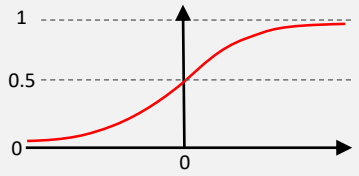
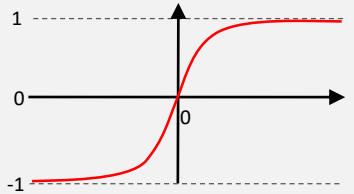
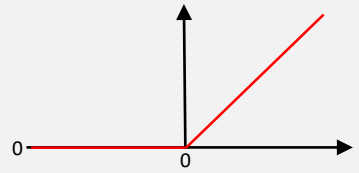
Activation functions (non-linearities) are needed so we can break the linearity and represent more complicated relationships.

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

Activation functions transform inputs into outputs of a different kind.

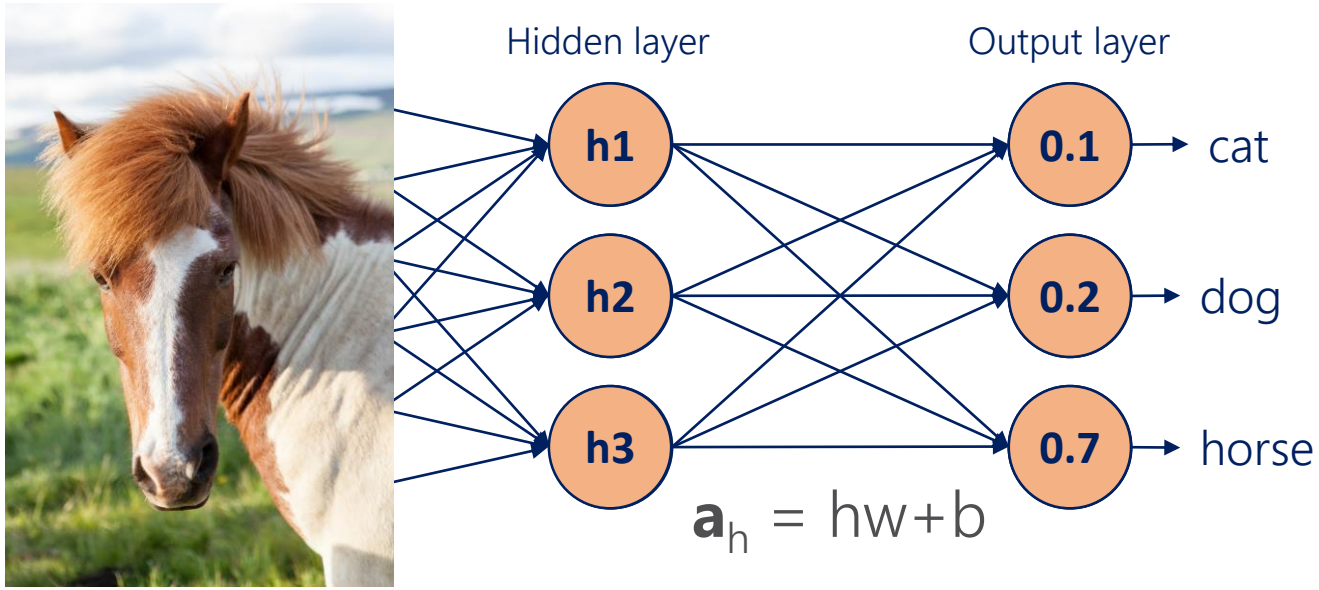


Common activation functions

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)	relu(a) = max(0,a)	$\frac{\partial \text{ relu}(a)}{\partial a} = \begin{cases} 0, if\ a \le 0 \\ 1, if\ a > 0 \end{cases}$		(0,∞)
softmax	$\sigma_i(\boldsymbol{a}) = \frac{e^{a_i}}{\sum_j e^{a_j}}$	$\frac{\partial \sigma_i(\boldsymbol{a})}{\partial a_j} = \sigma_i(\boldsymbol{a})\left(\delta_{ij} - \sigma_j(\boldsymbol{a})\right)$ 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



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]

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

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

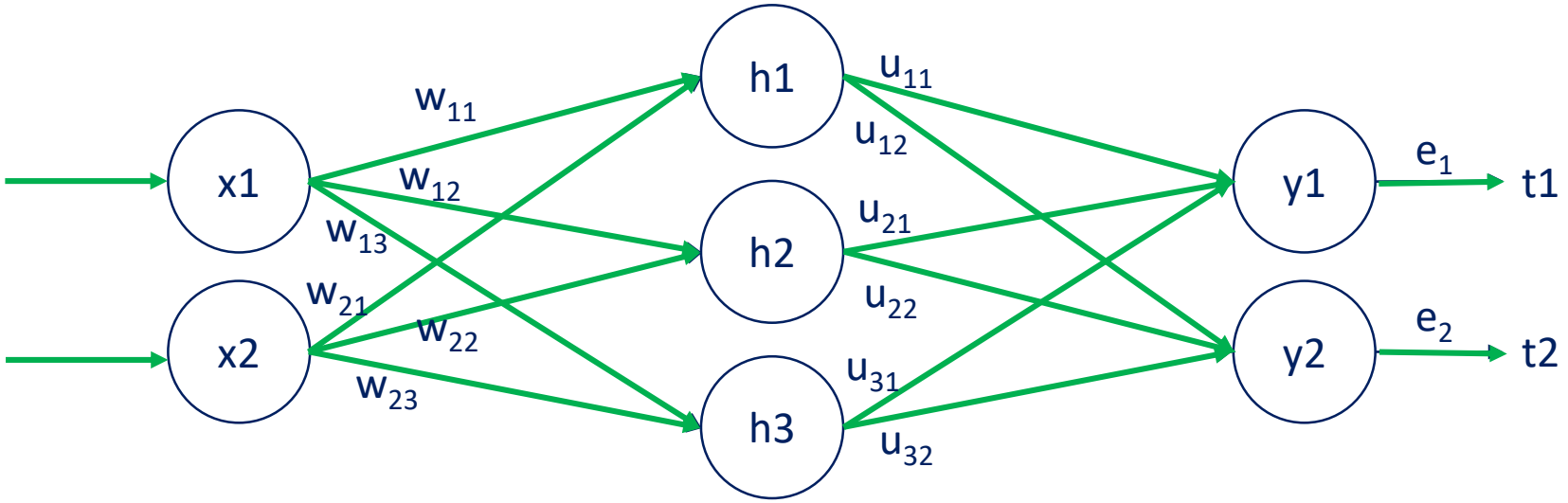
$$\text{softmax}(\mathbf{a}) = \left[\frac{e^{-0.21}}{8}, \frac{e^{0.47}}{8}, \frac{e^{1.72}}{8}\right]$$

y = [0.1,0.2,0.7] → probability distribution

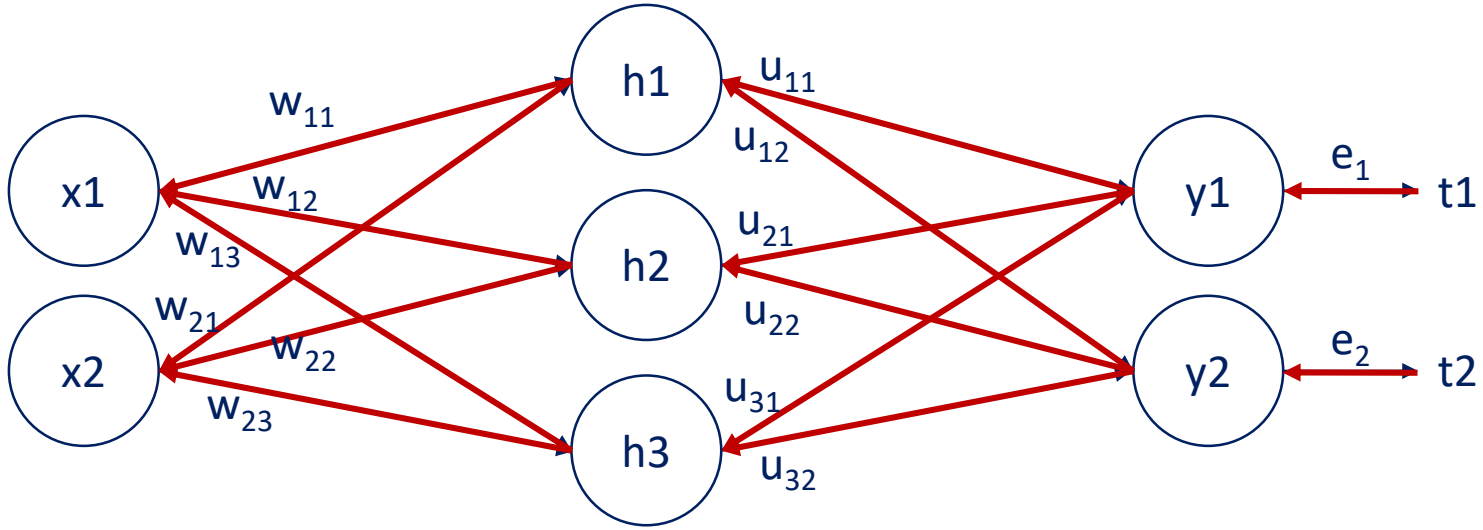
The property of the softmax to output probabilities is so useful and intuitive that it is often used as the activation function for the **final (output) layer.**

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

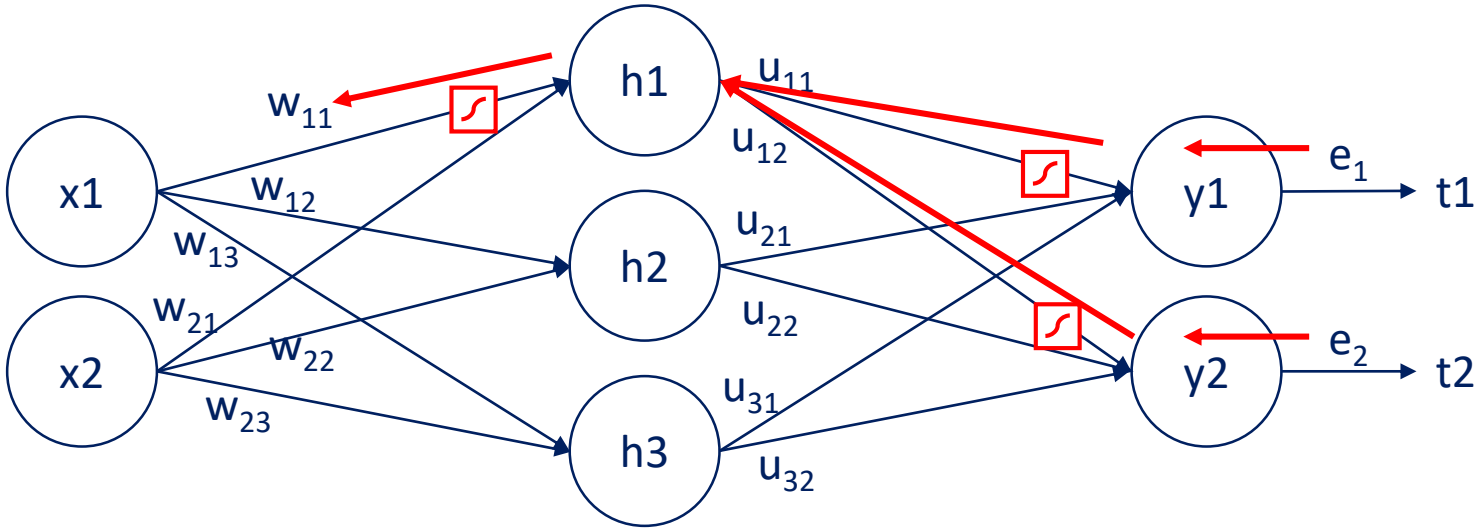


Forward propagation is the process of pushing inputs through the net. At the end of each epoch, the obtained outputs are compared to targets to form the errors.



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.**