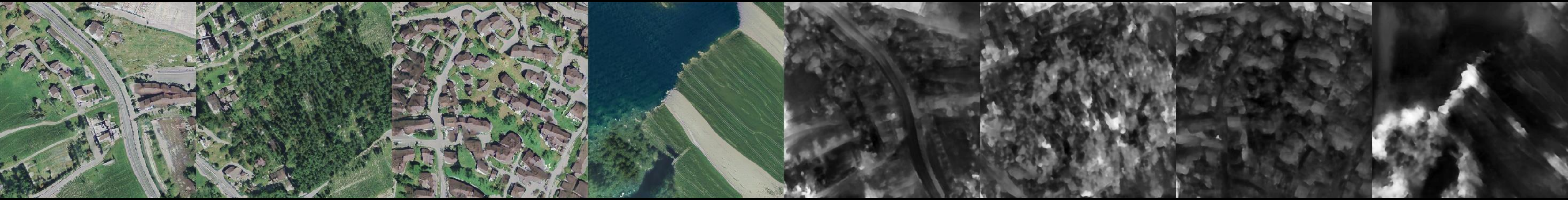
Exploring Machine Intelligence

Week 2, Basic building blocks

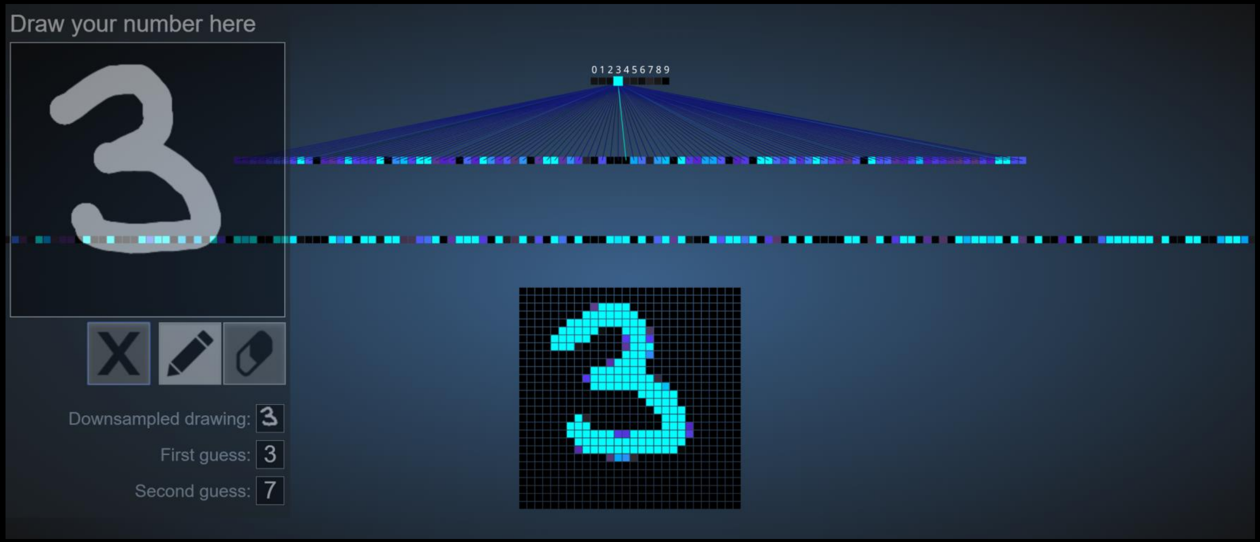


^ image credits: [gallery]

~ Vítek Růžička

Motivation for today

- Learn about the basic neural network building blocks to understand what's happening here:



Interactive fully connected neural network

>>> www.cs.cmu.edu/~aharley/vis/fc/flat.html <<<

Feedback

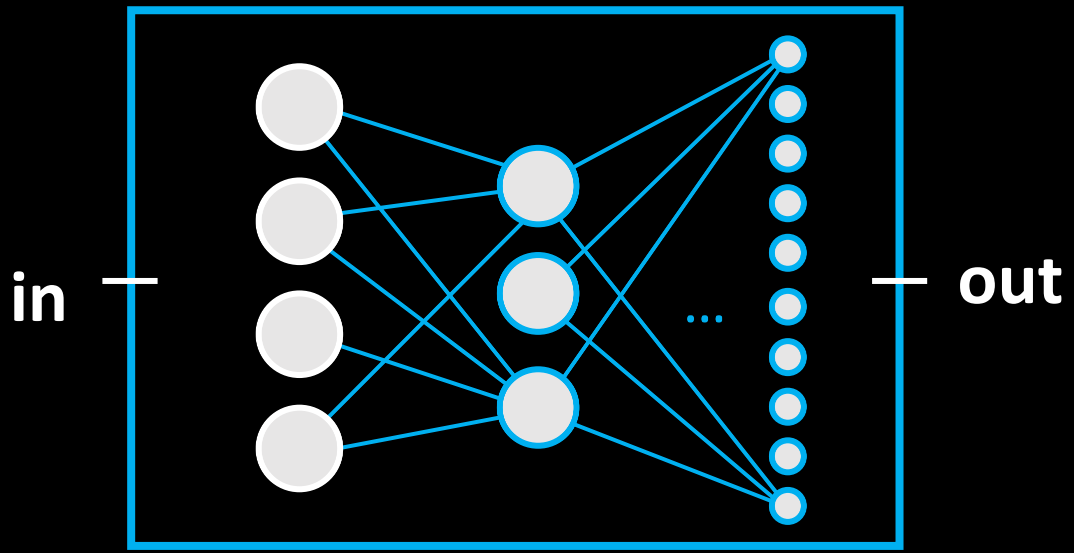
Last week's quiz feedback:

- **Mixed backgrounds:** Animation, Game Dev, Mathematics, Interaction design, Music production, Illustration, etc. etc.
- Mentioned **topics of interests**:
  - ML in **Animation**, ML in **Music Generation**, …
  - Training and creating **your own models** (and not just reusing pre-trained ones)
  - Manifesting **from digital into the real world** (3D printing, etc.)
  - **Interactivity** and physical interactivity

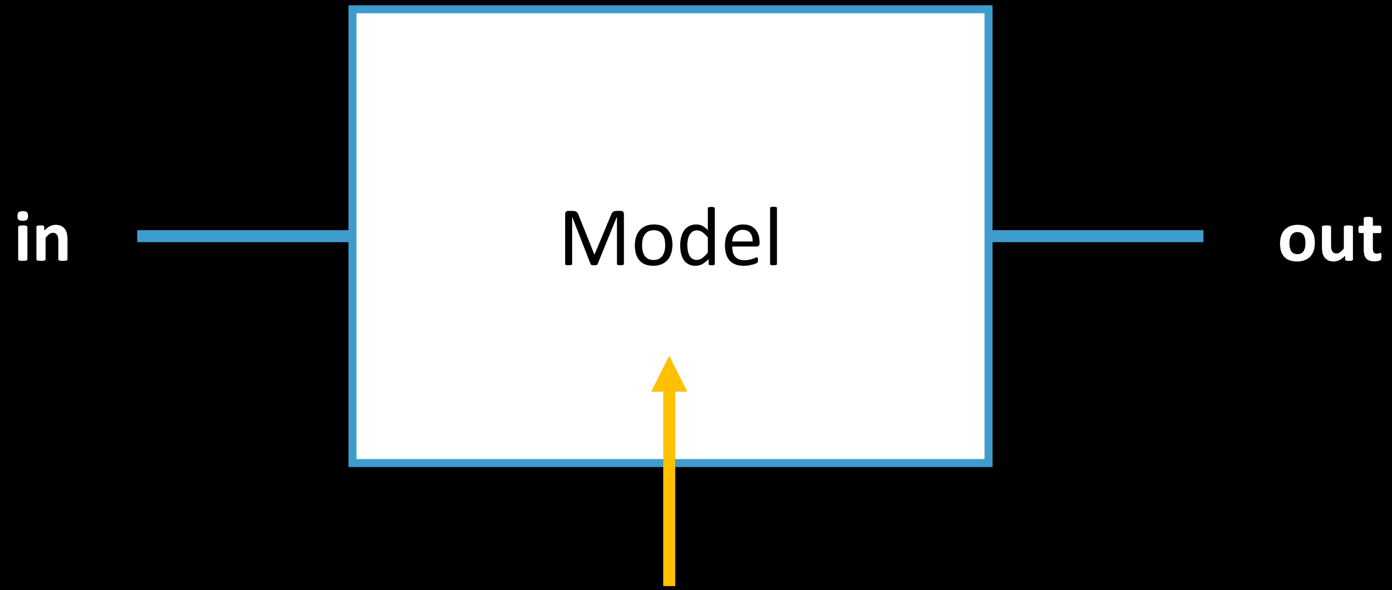
Today

Basic building blocks:

- Artificial Neurons
- Neural Networks
- Plugging in image data
- Training a NN model

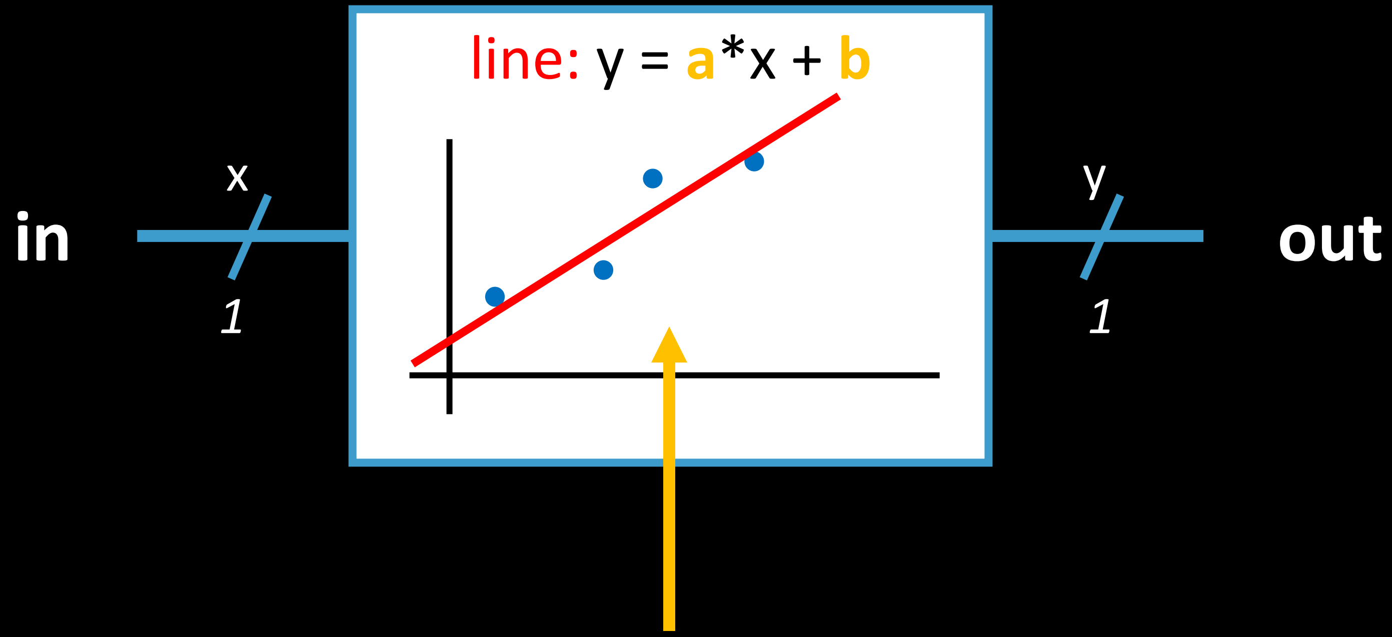


Model



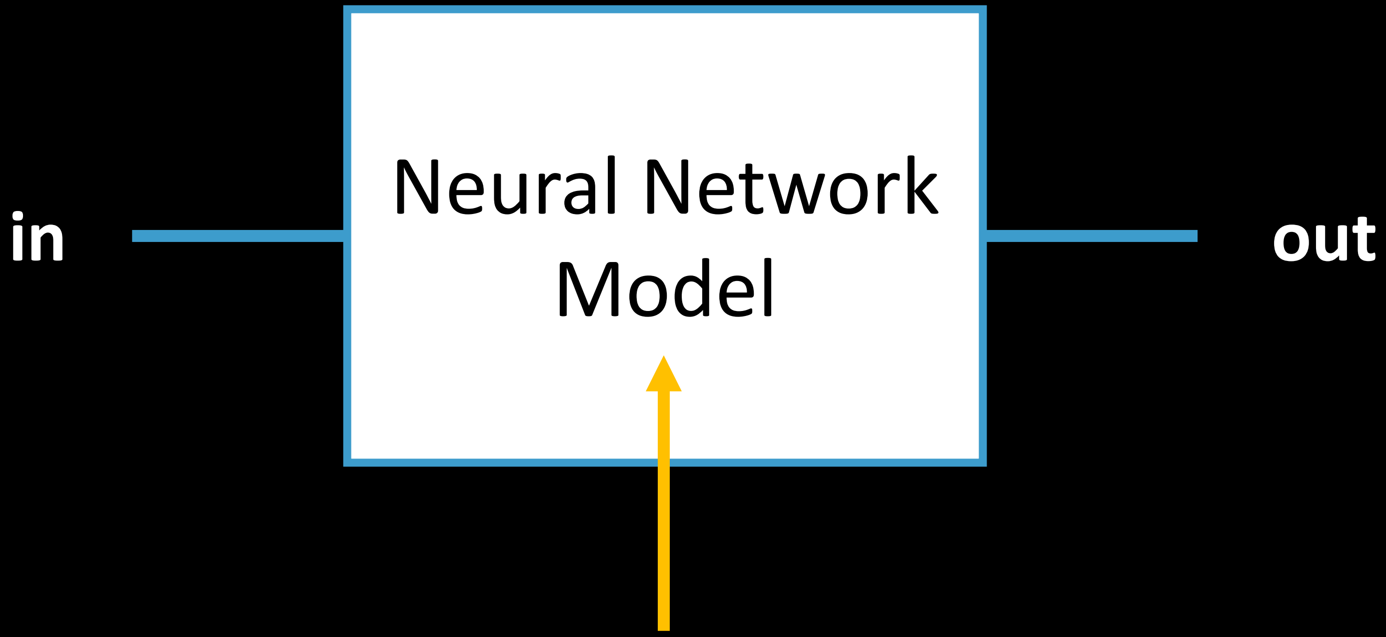
Task: find the best parameters so that they correspond to the translation of inputs to outputs

Line as a model (linear regression)



Task: find the best parameters **a** and **b**, so that the line corresponds the best to the data

Neural Network



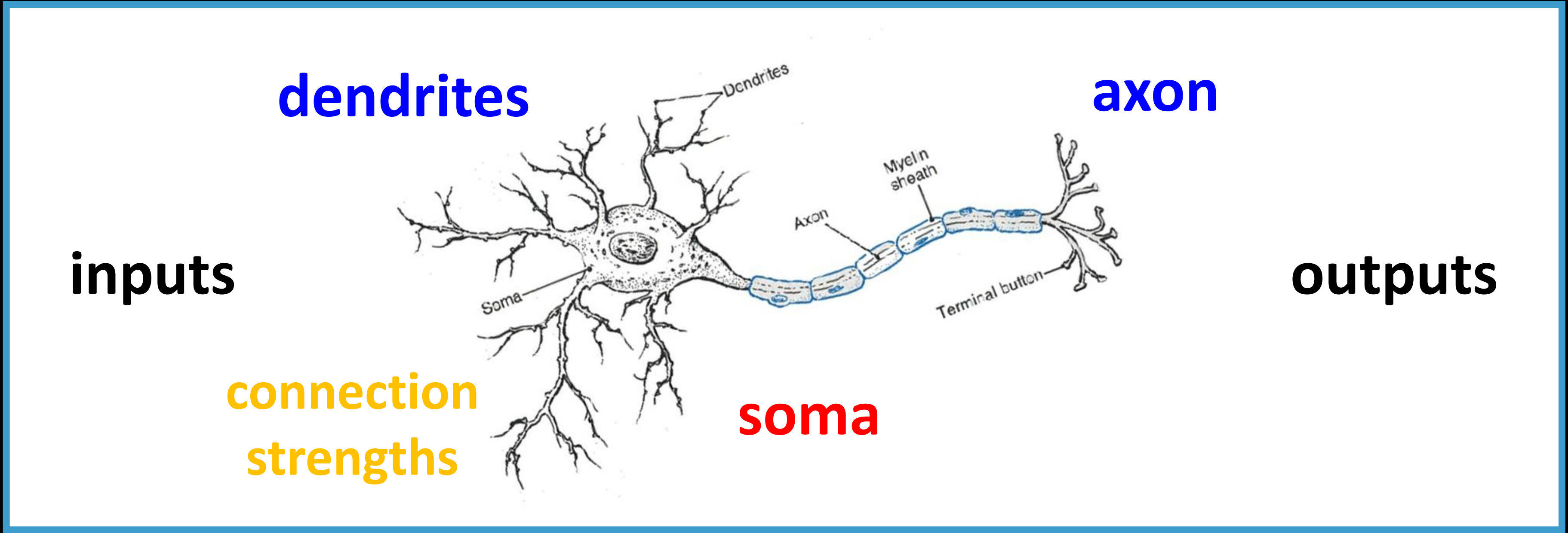
Task: find the best parameters
of the Neural Network

Neurons

- Before we start talking about more complex ML models, we should address the basic building block – the artificial neuron

Biological Neuron

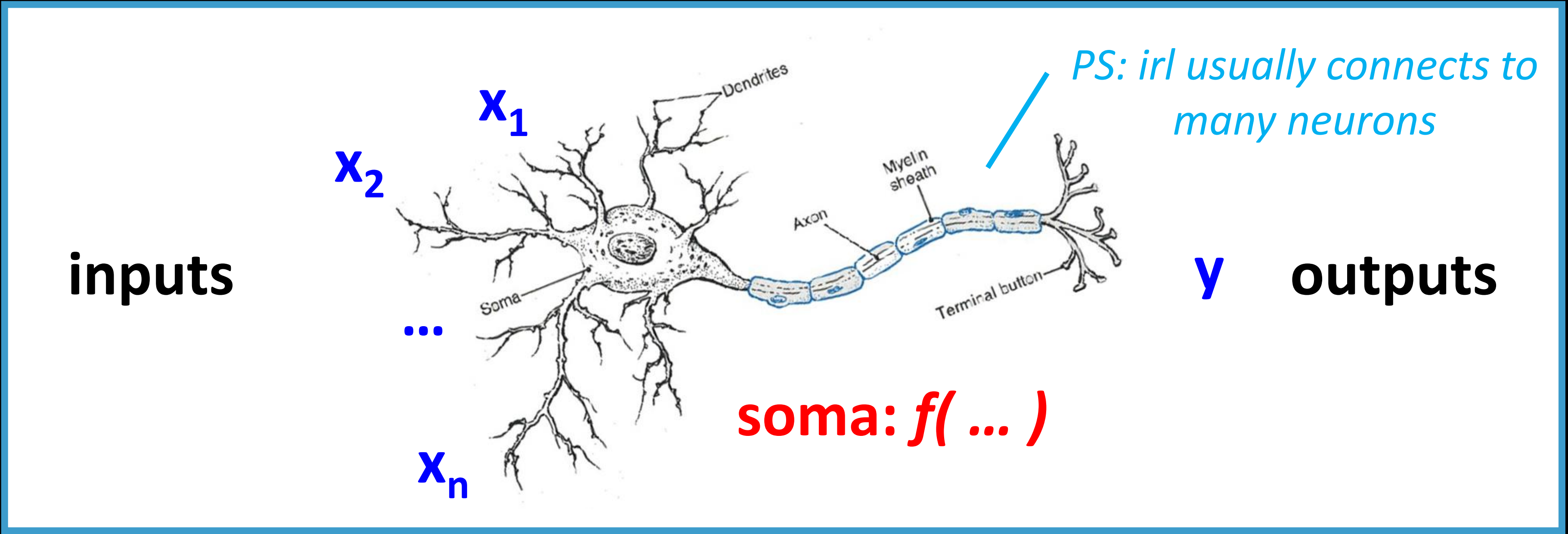
- How does a real biological neuron in brain work? (*Roughly*)



Neuron accepts some signals (from other neurons with **different connection strengths**), adjusts them in **soma** and then propagates the signal further.

Biological Neuron

- Can this be described in a mathematical way?



Neuron somehow combines the incoming signals, processes then using a function *f(…)* and outputs them.

Abstraction of neuron

biological

different connection strengths

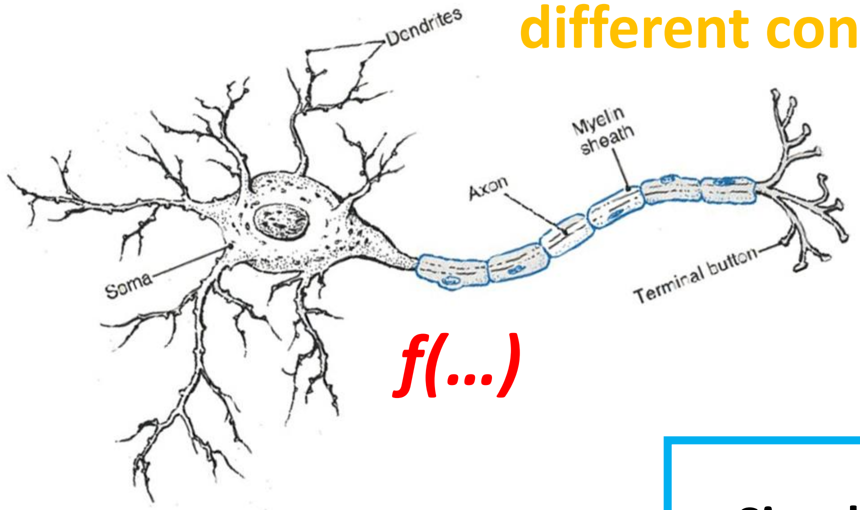
inputs

x_1

x_2

...

x_n



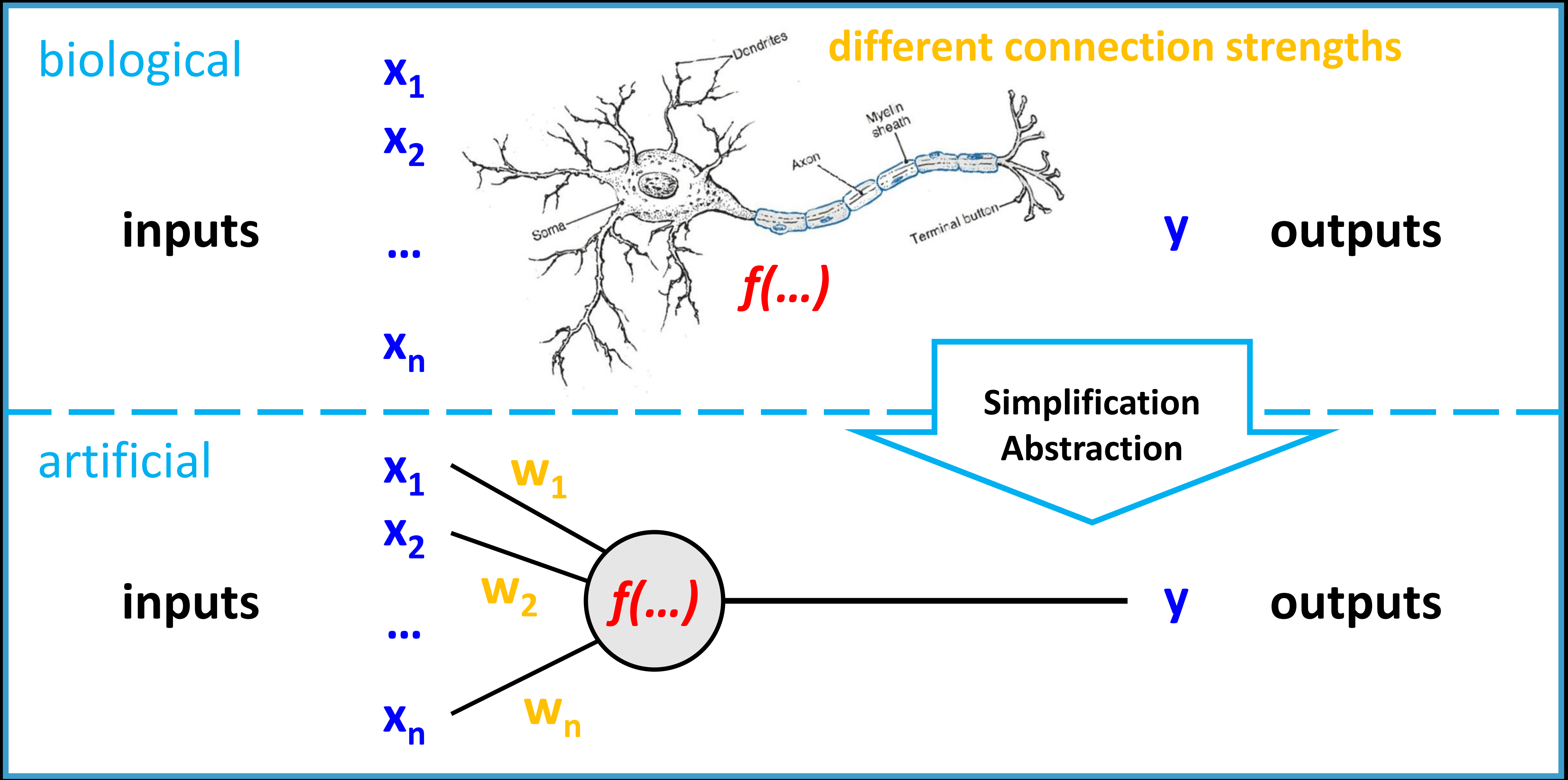
$f(\ldots)$

y

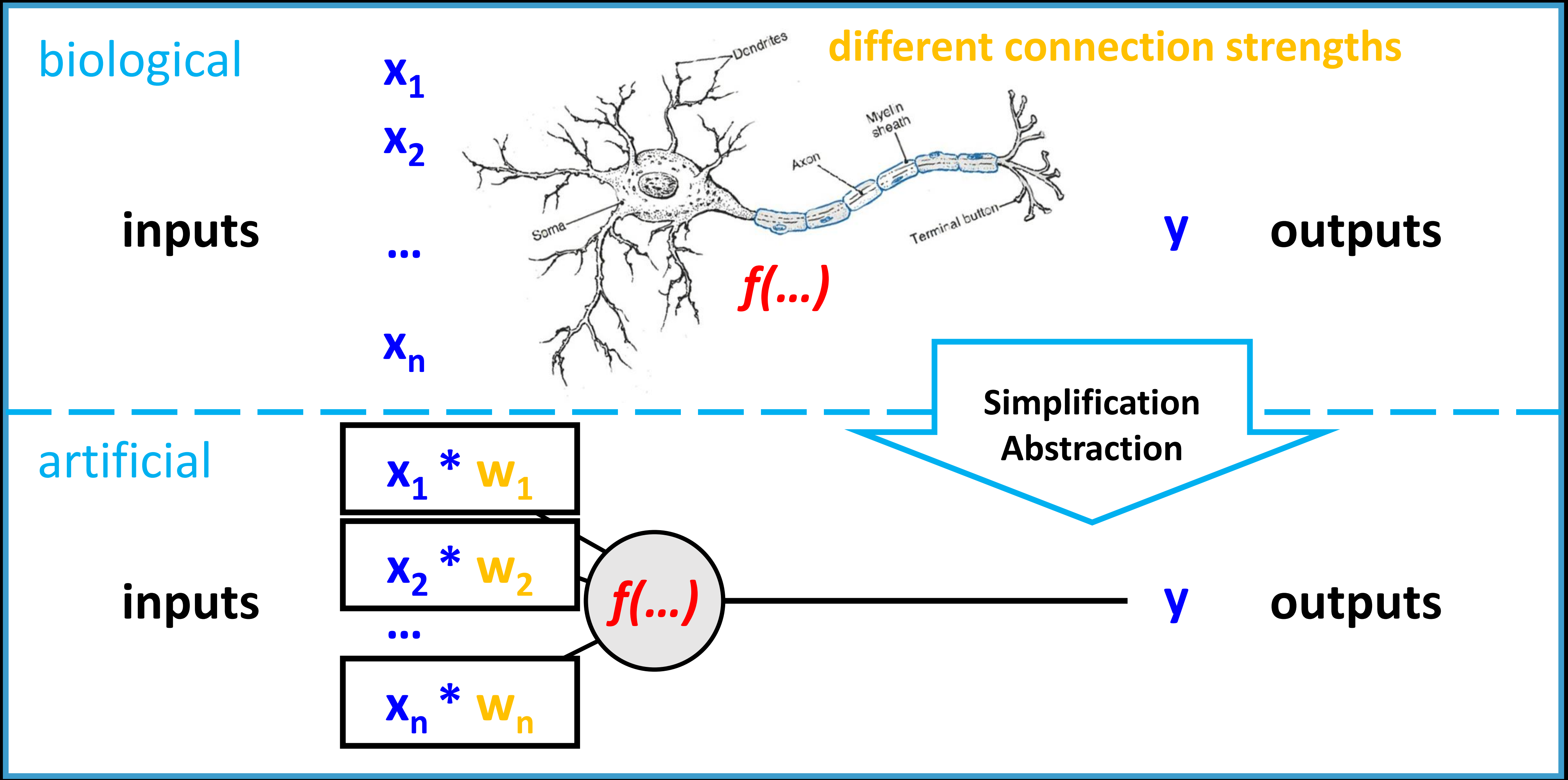
outputs

Simplification
Abstraction

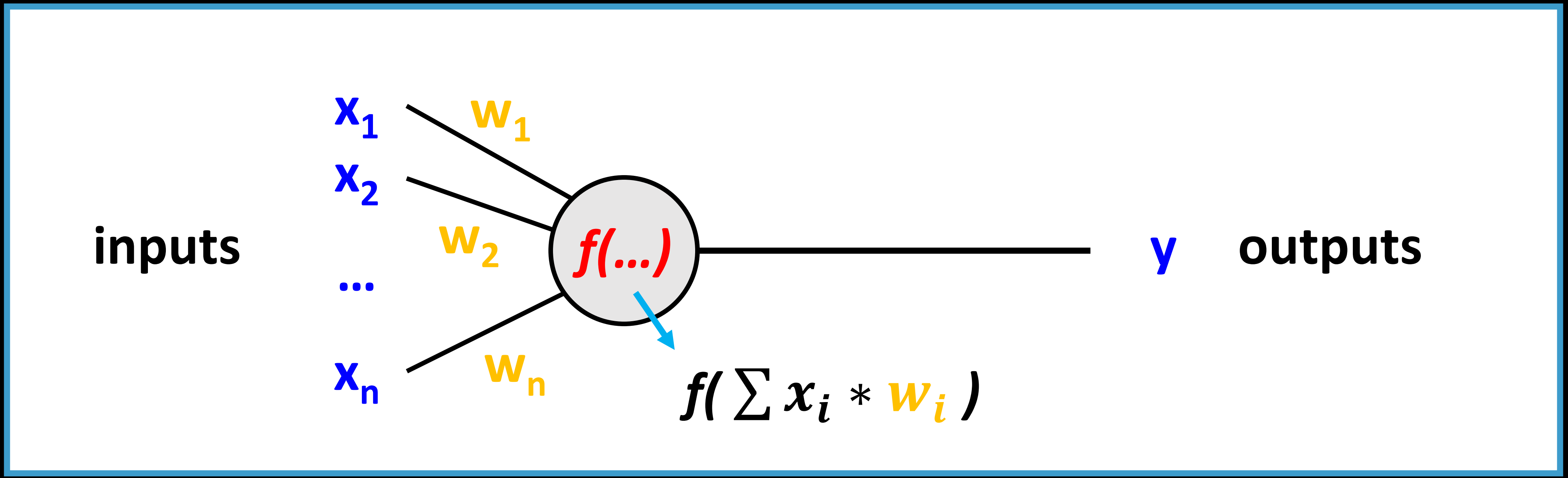
Abstraction of neuron



Abstraction of neuron

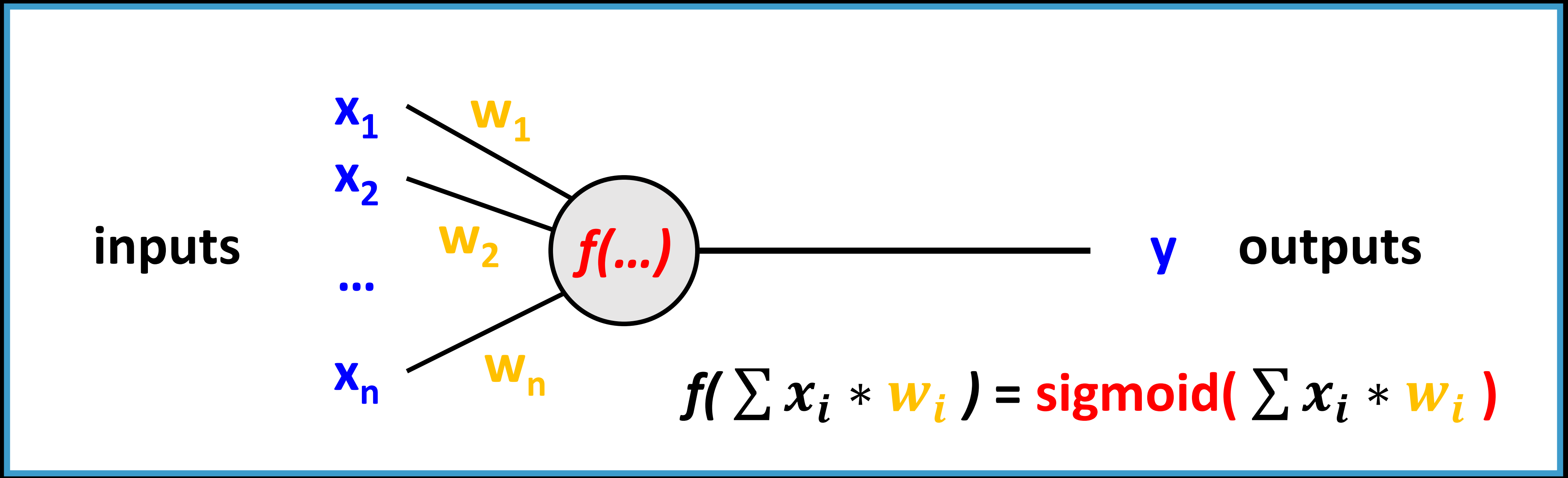


Artificial Neuron



- This model still has to **combine the incoming signals** (sum them) and then **process** them somehow.

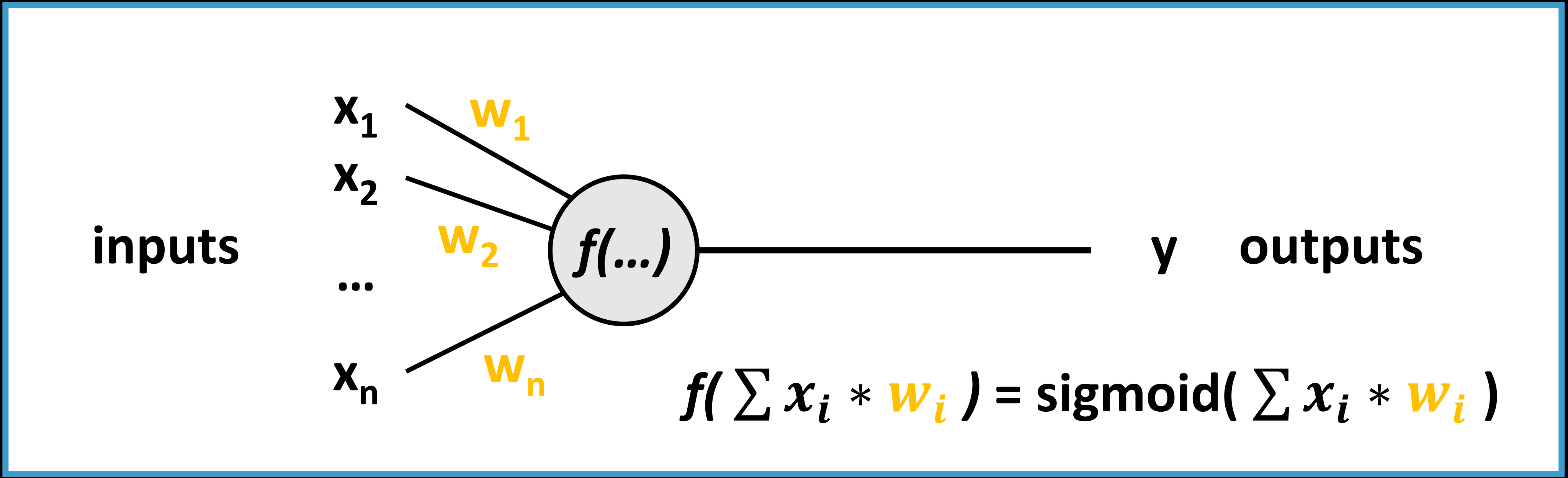
Artificial Neuron



$$f\left(\sum x_i * w_i\right) = \text{sigmoid}\left(\sum x_i * w_i\right)$$

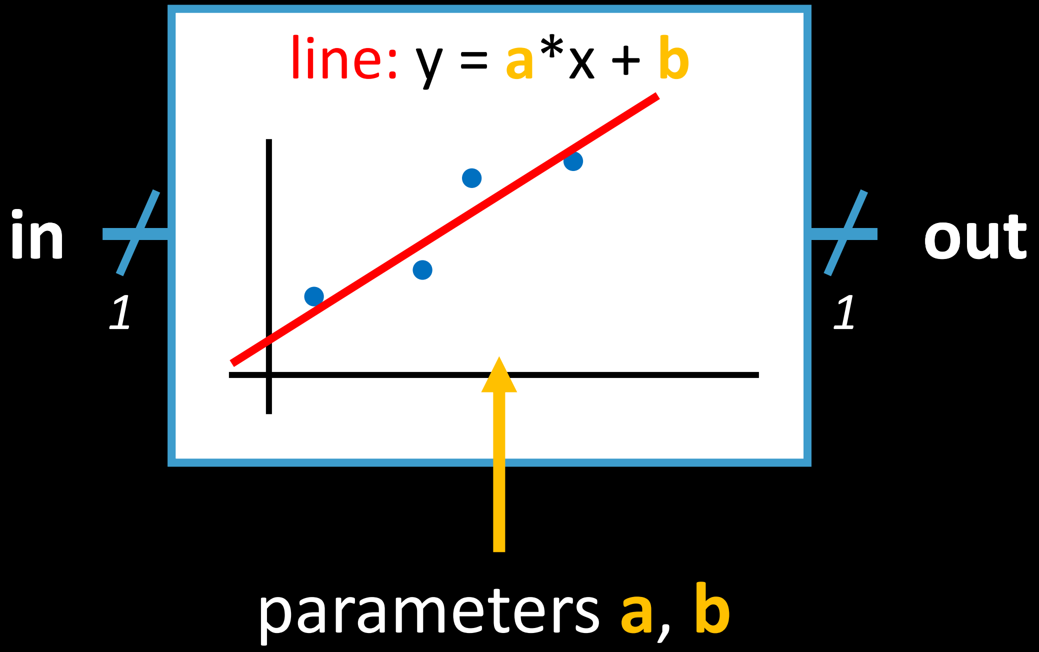
- This model still has to **combine the incoming signals** (sum them) and then **process** them somehow.
- Ideally so that everything goes in between 0 and 1 (**sigmoid**).

Parameters of Artificial Neuron

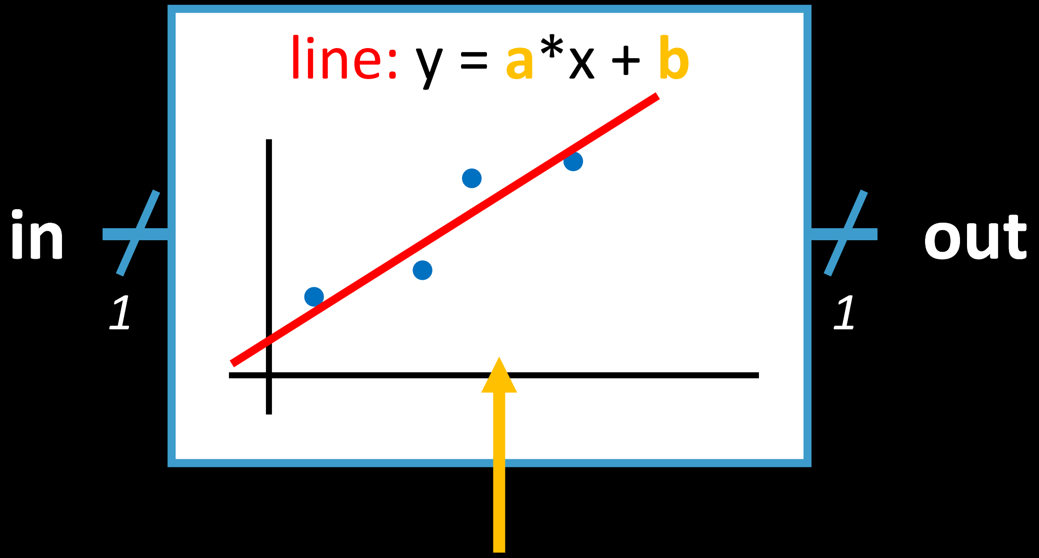


- With inputs of **size N**, we have **N parameters** (**weights**).
  - *PS: Detail: there is one more parameter, so N+1 but it's not too important right now*

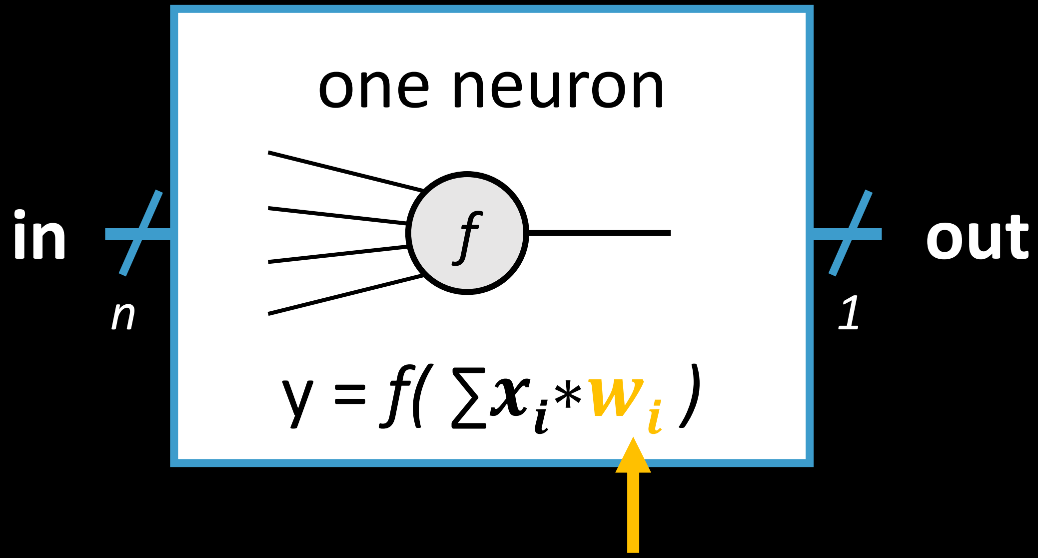
Bigger picture



Bigger picture

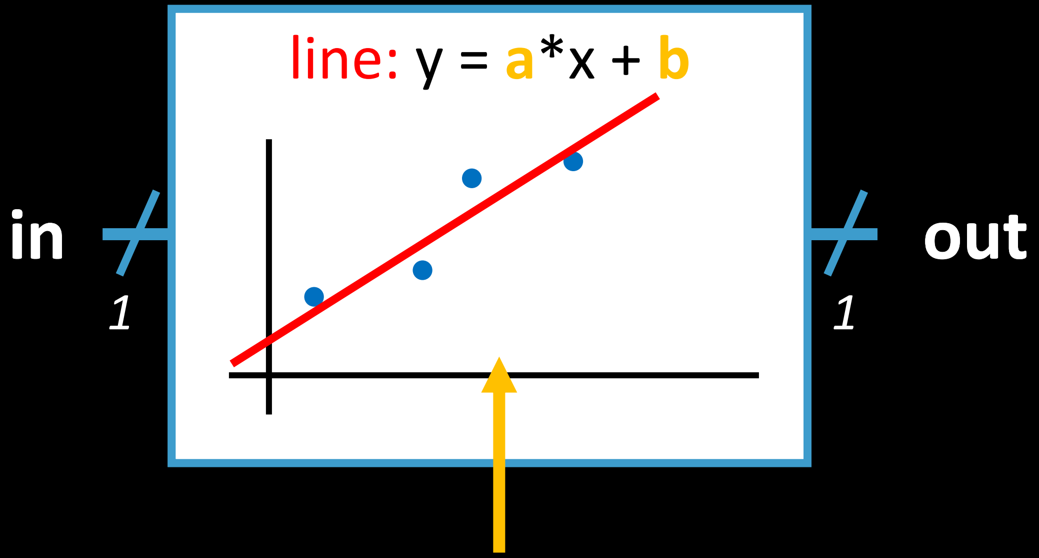


parameters **a**, **b**

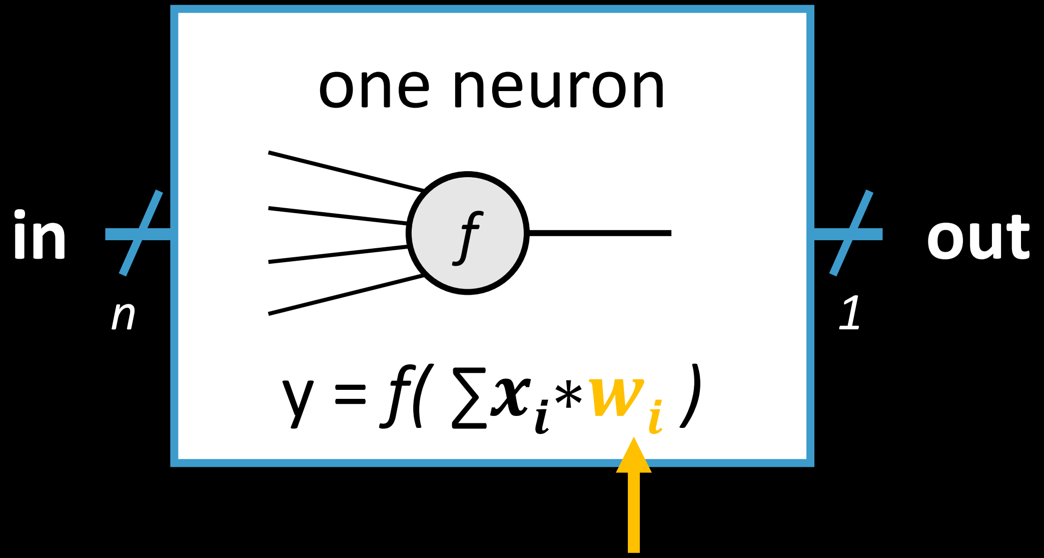


parameters $\mathbf{w_i}$ (weights)

Bigger picture



parameters **a**, **b**



parameters $\mathbf{w_i}$ (weights)

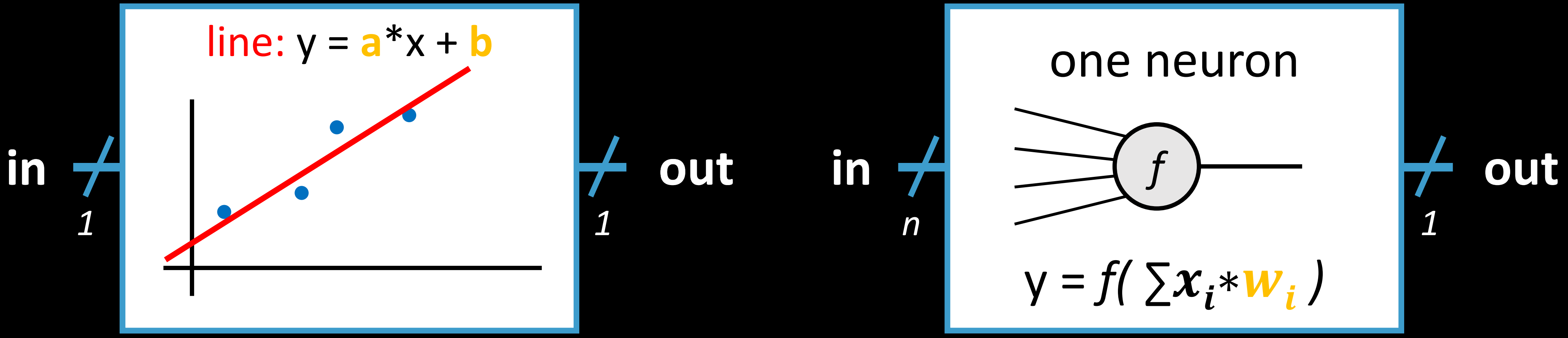
BTW:

$y = x*a + 1*b$

mathematically for n=1
these are the same models!

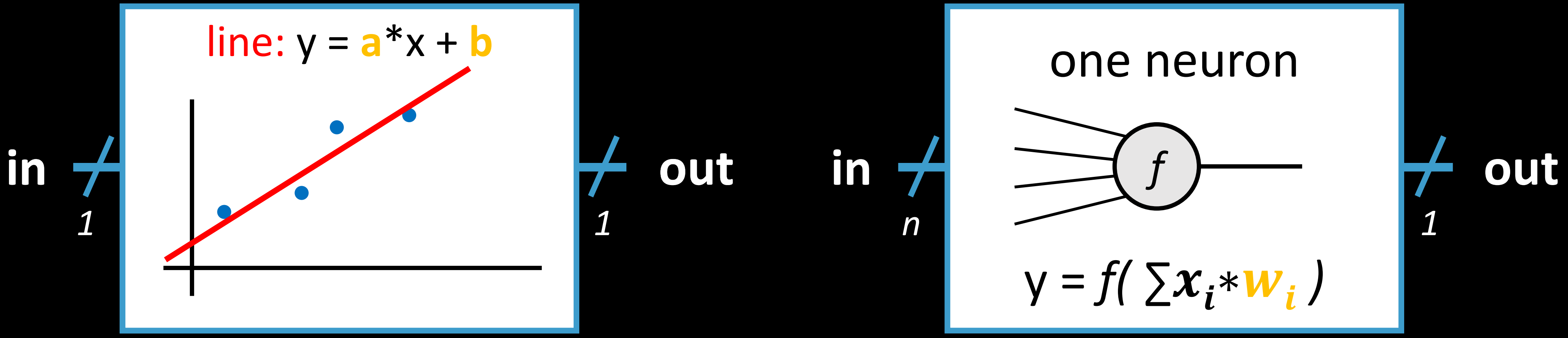
$y = x*w_1 + 1*w_0$

Bigger picture



- In order to **train the model** (= make it useful) we have to adjust parameters a, b or w_i so that the model fits the data.

Bigger picture

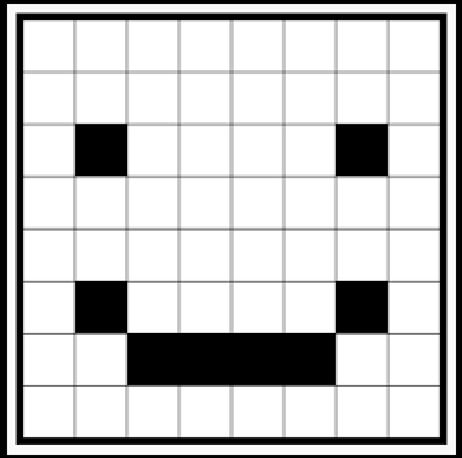


- As before with the line – we will need couple of things:
  - **data** and know how to *plop* them at the inputs and outputs
  - **measure of error**

Plugging in the data

- I will directly jump into data we care about – images:

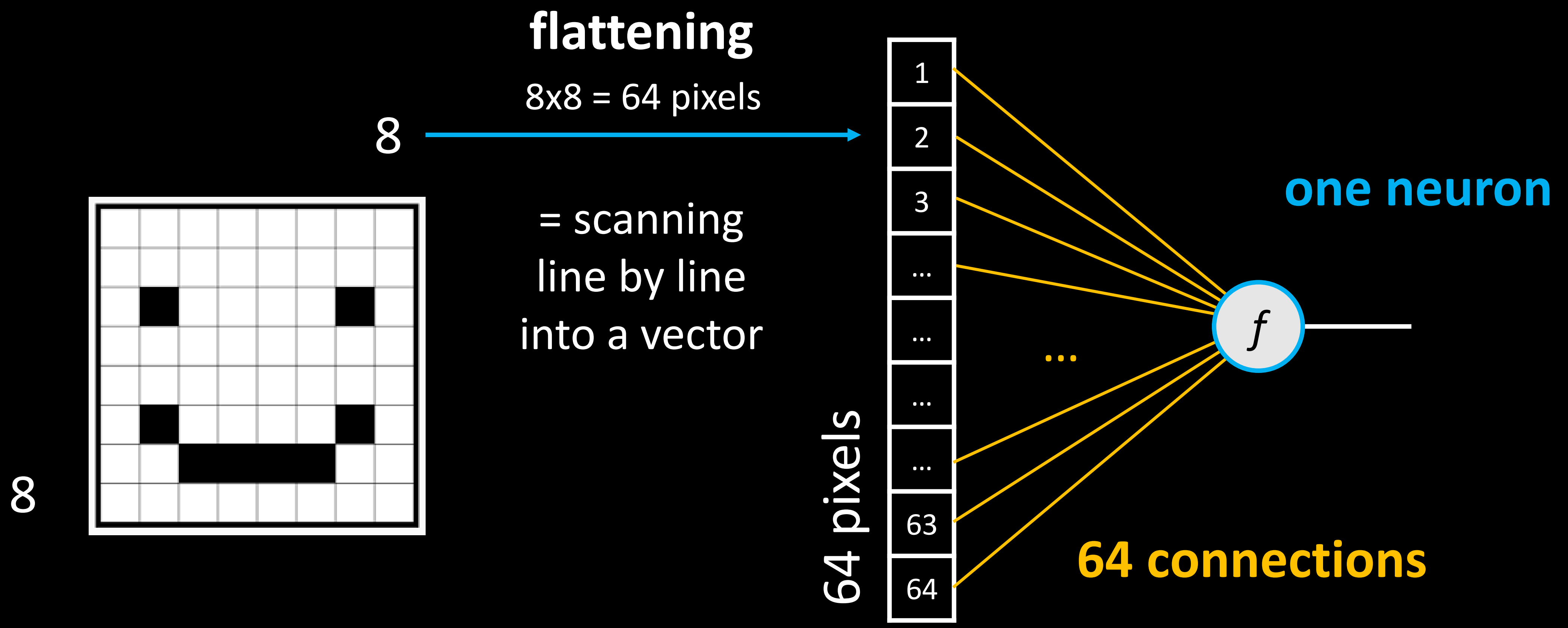
8



8

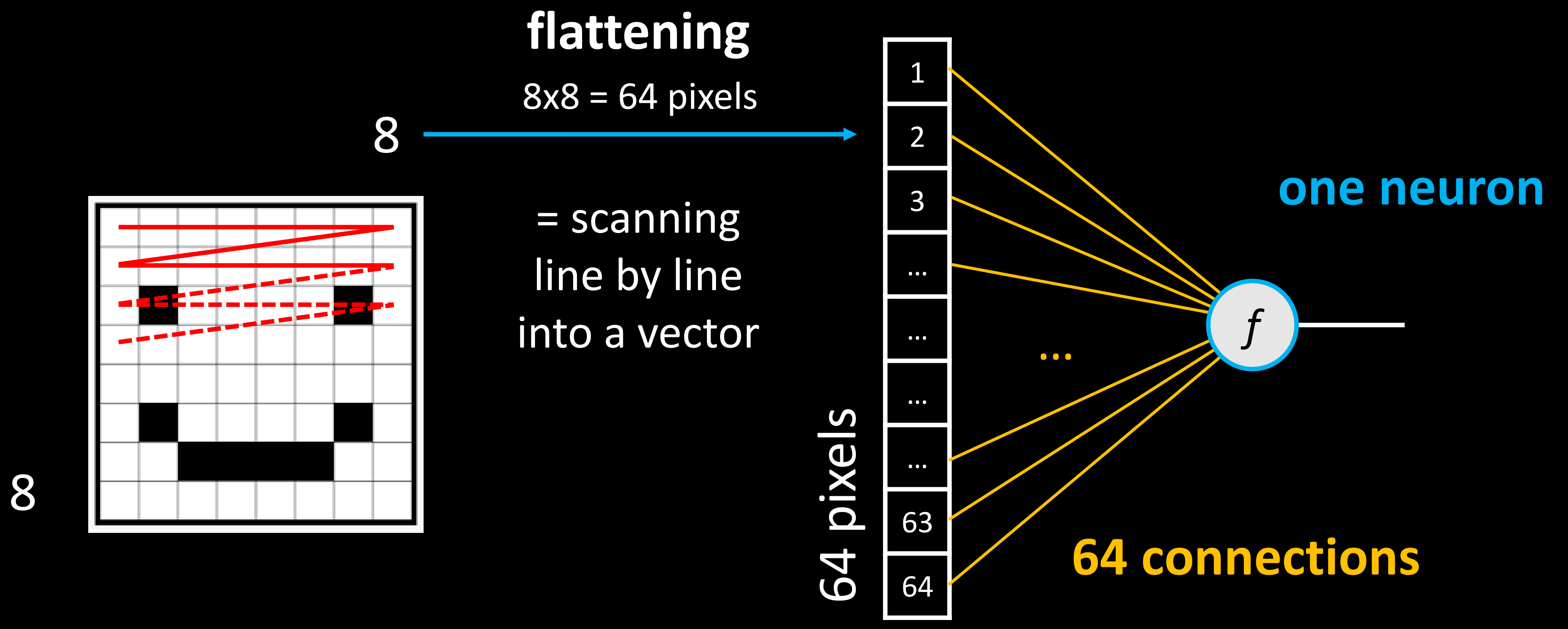
Plugging in the data

- I will directly jump into data we care about – images:

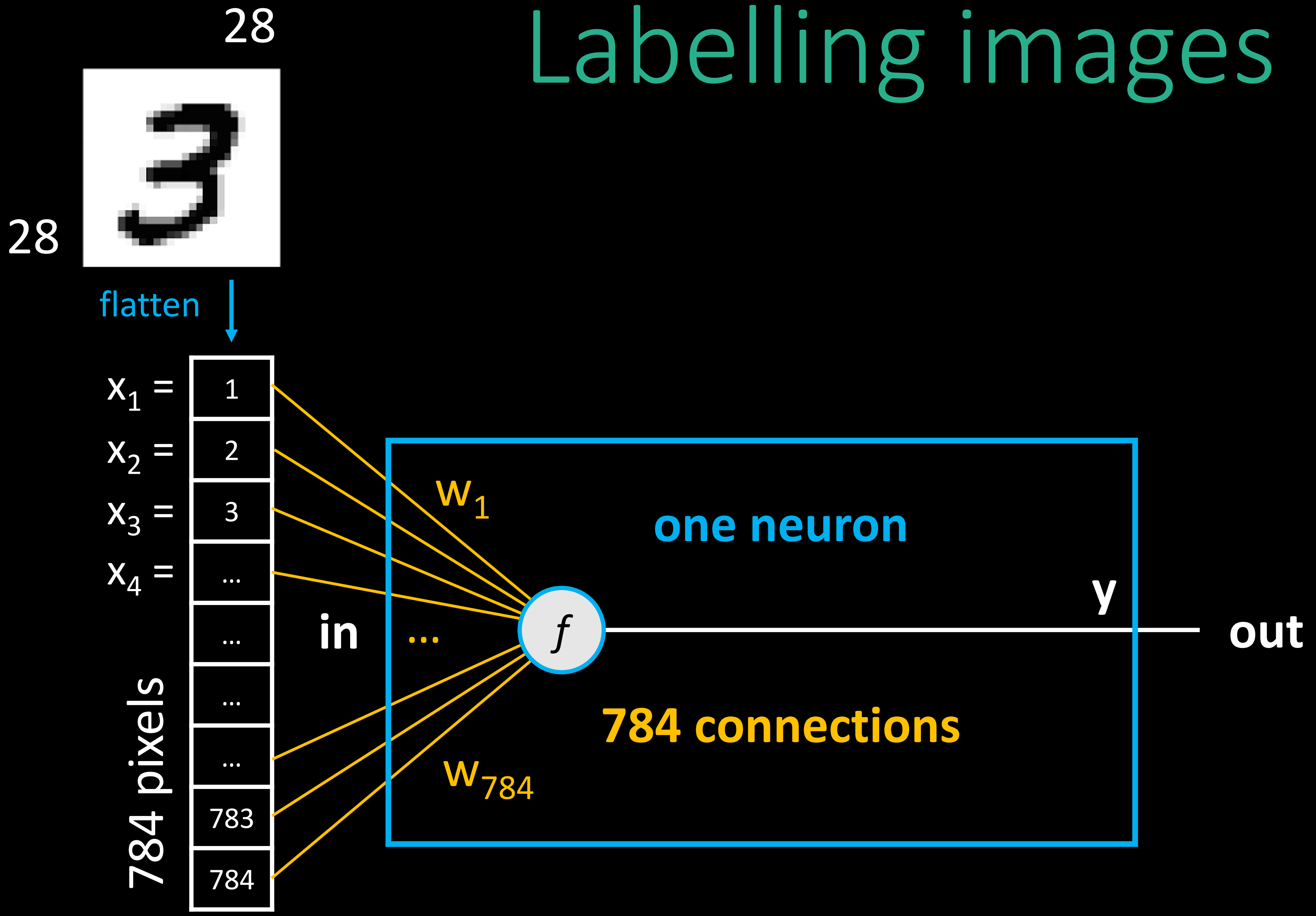


Plugging in the data

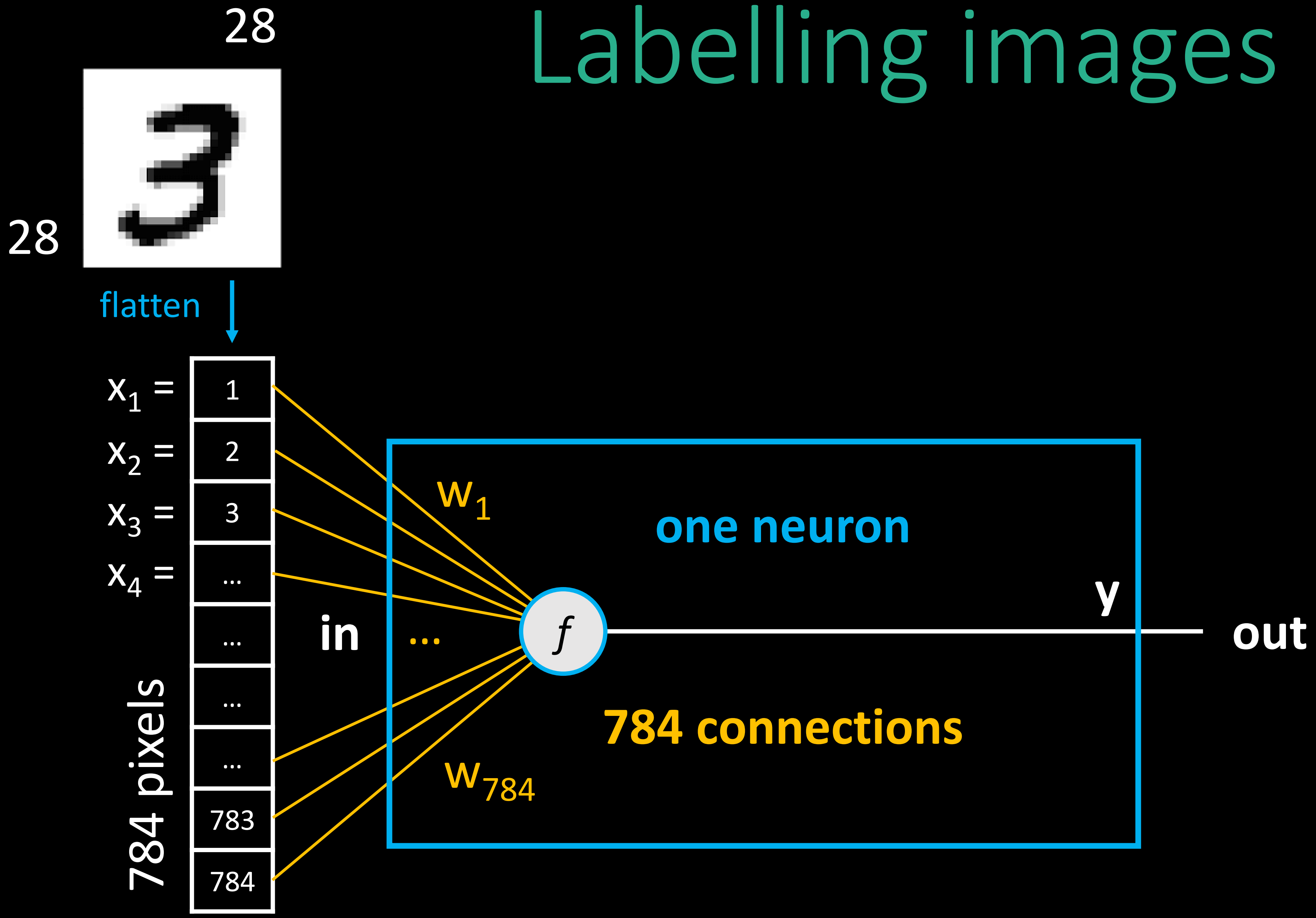
- I will directly jump into data we care about – images:



Labelling images

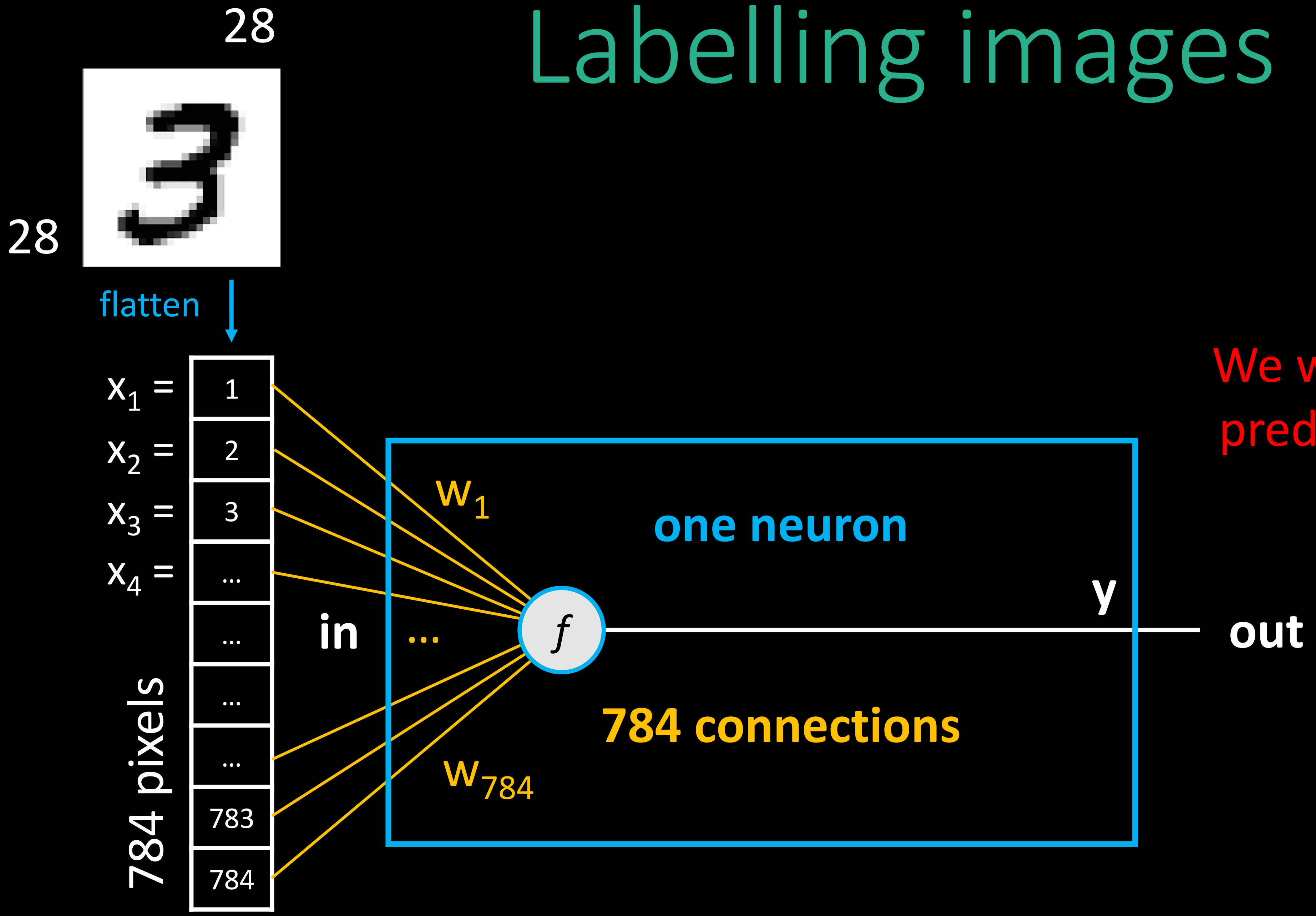


Labelling images



Question: What are we missing?

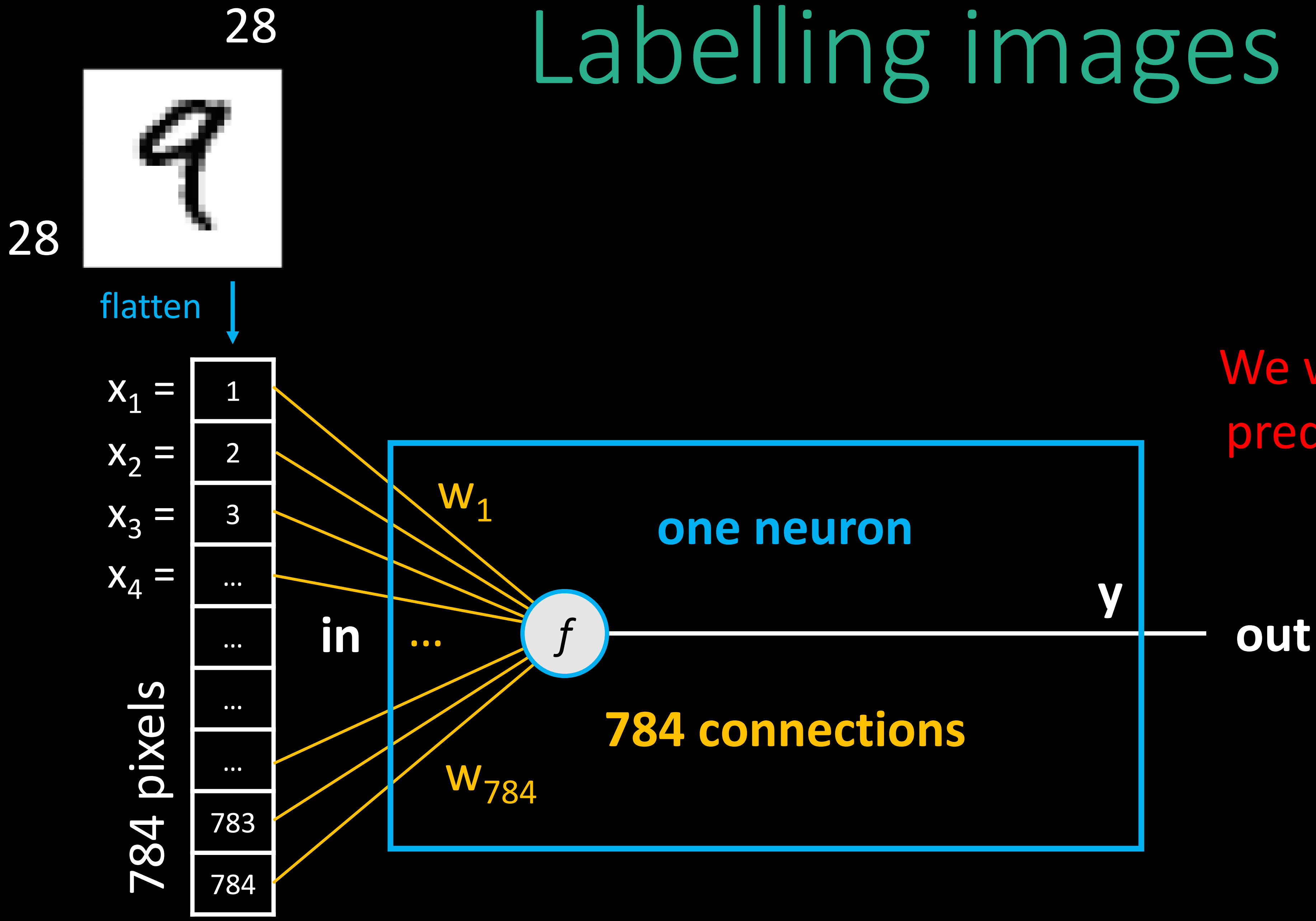
Labelling images



We want to have the prediction similar to the label:

label of “3”

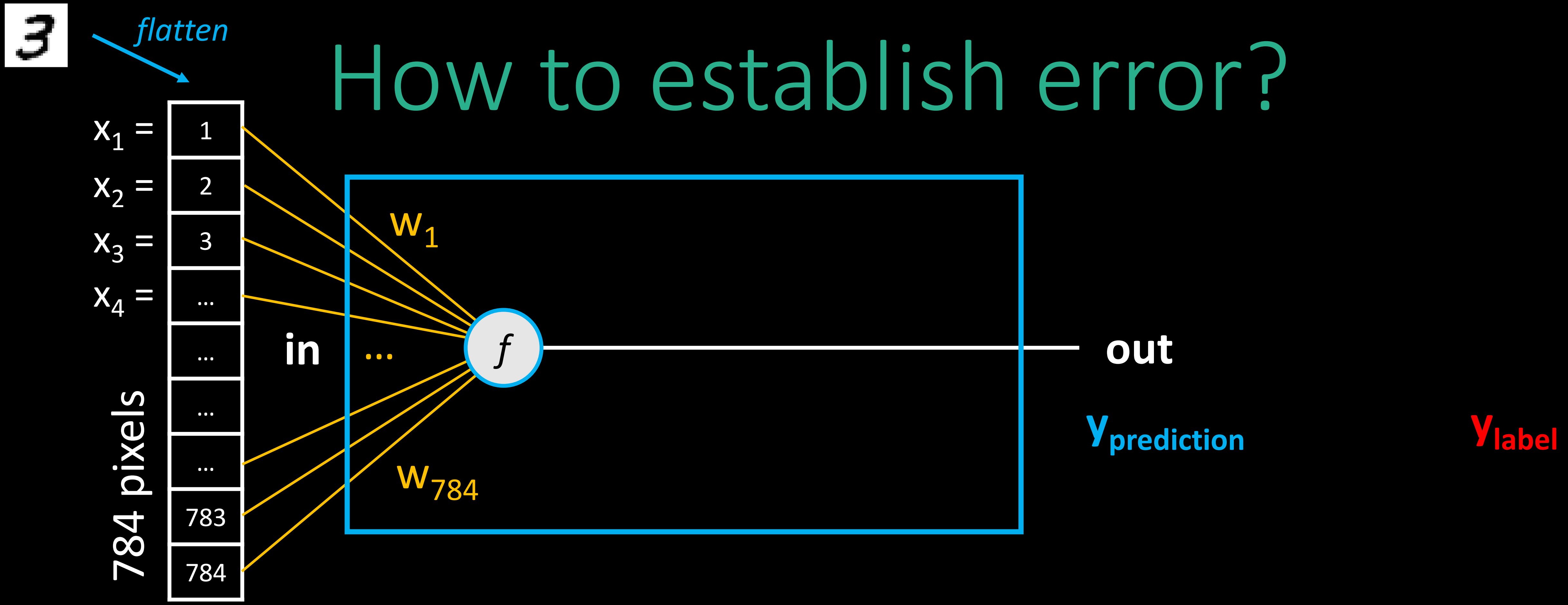
Labelling images



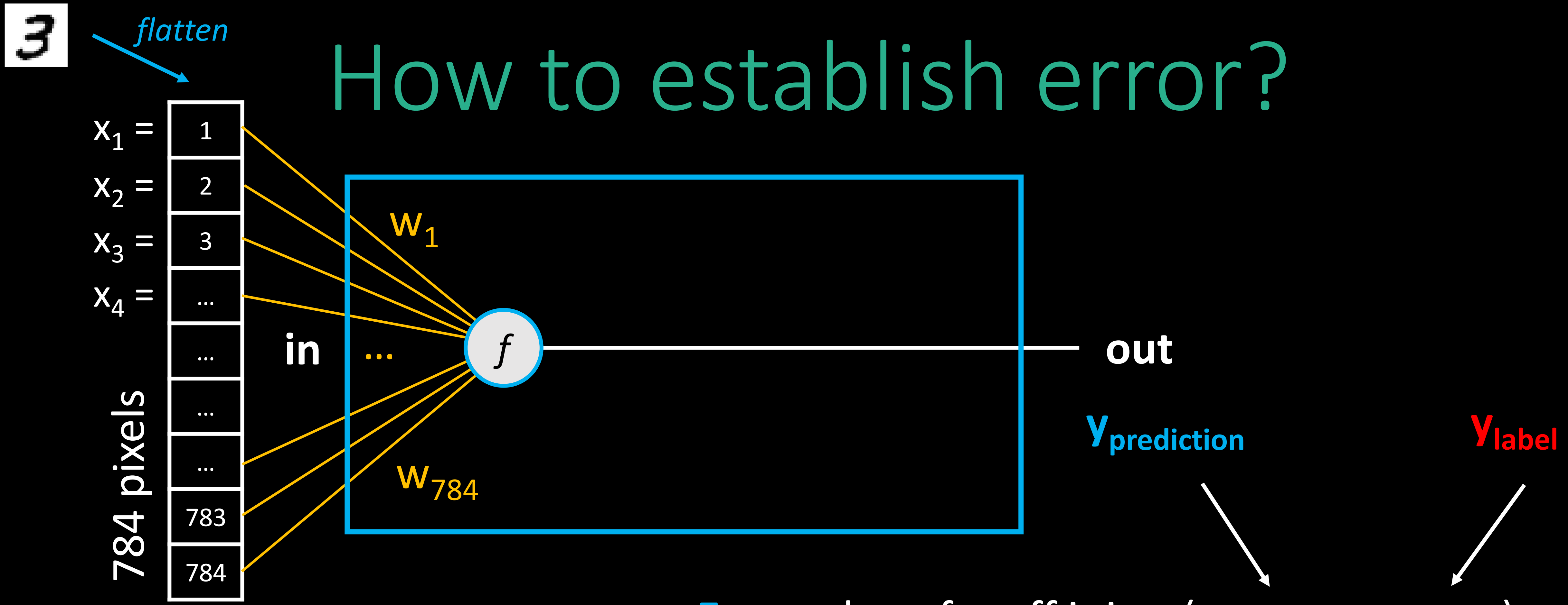
We want to have the prediction similar to the label:

label of "9"

How to establish error?

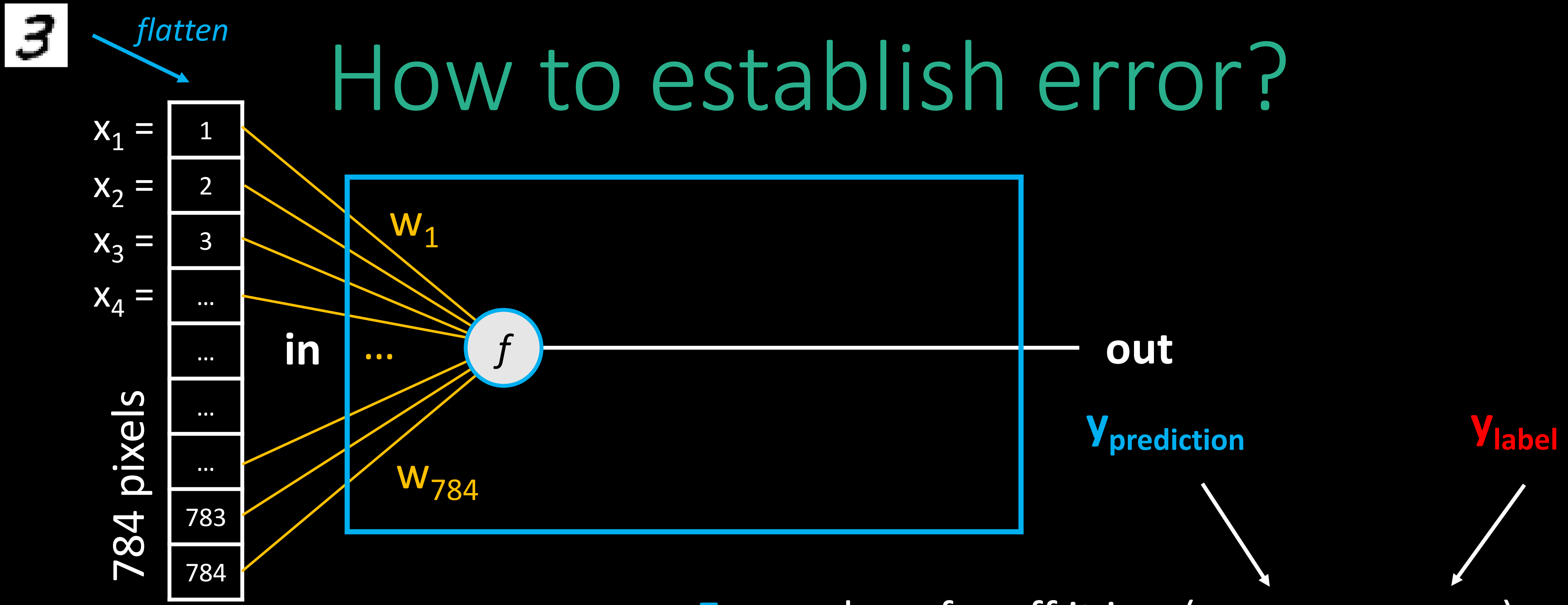


How to establish error?



Error = how far off it is = ($y_{prediction}$ - y_{label})

How to establish error?



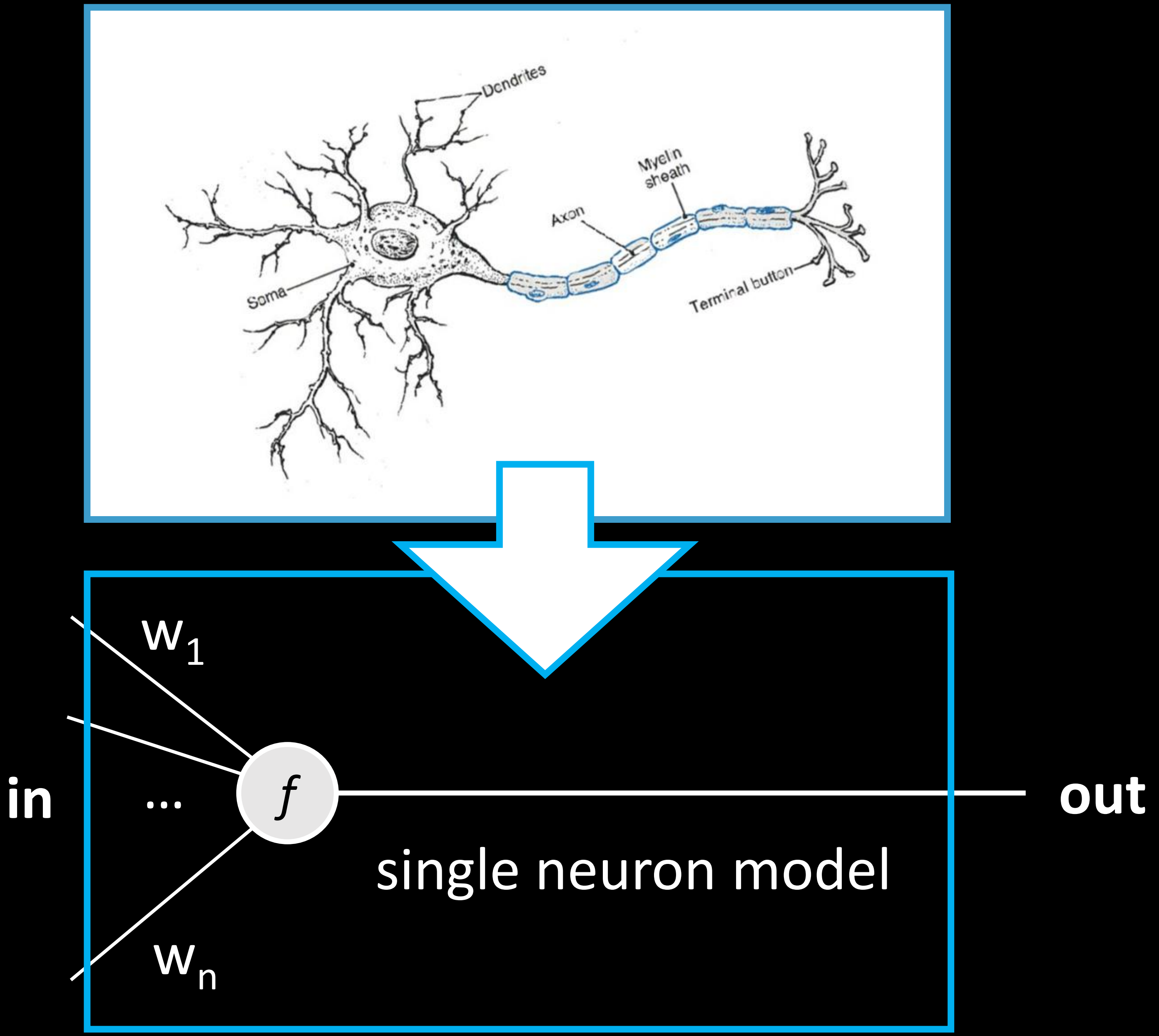
Error = how far off it is = ($y_{prediction}$ - y_{label})

- **<u>Task becomes:</u>** Change the parameters of $w_1, \dots, w_n$ in order so that when you give it the input image, the model calculates as answer which is the closest from what I labelled it with.

Pause 1

Pause 1

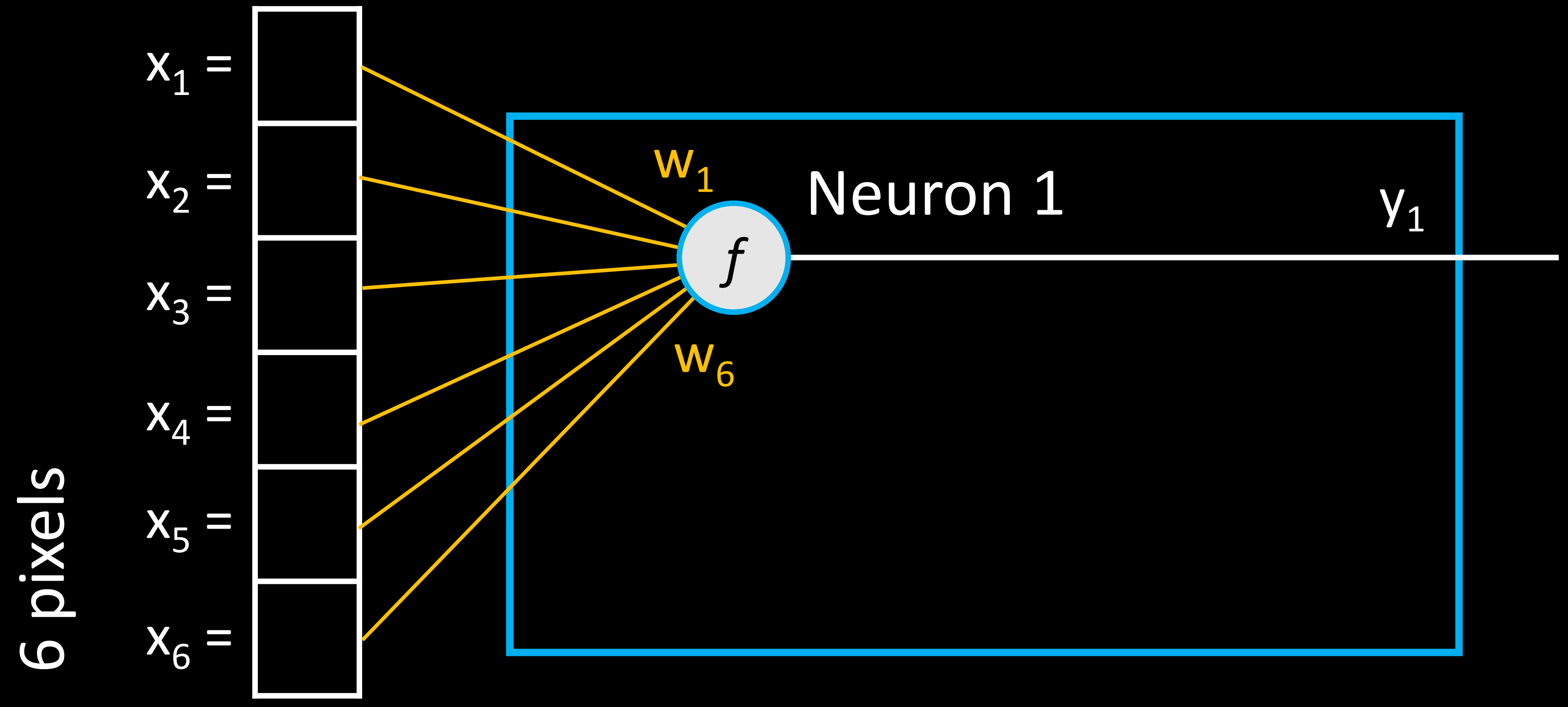
Is everything clear until now?



xkcd.com/1838/

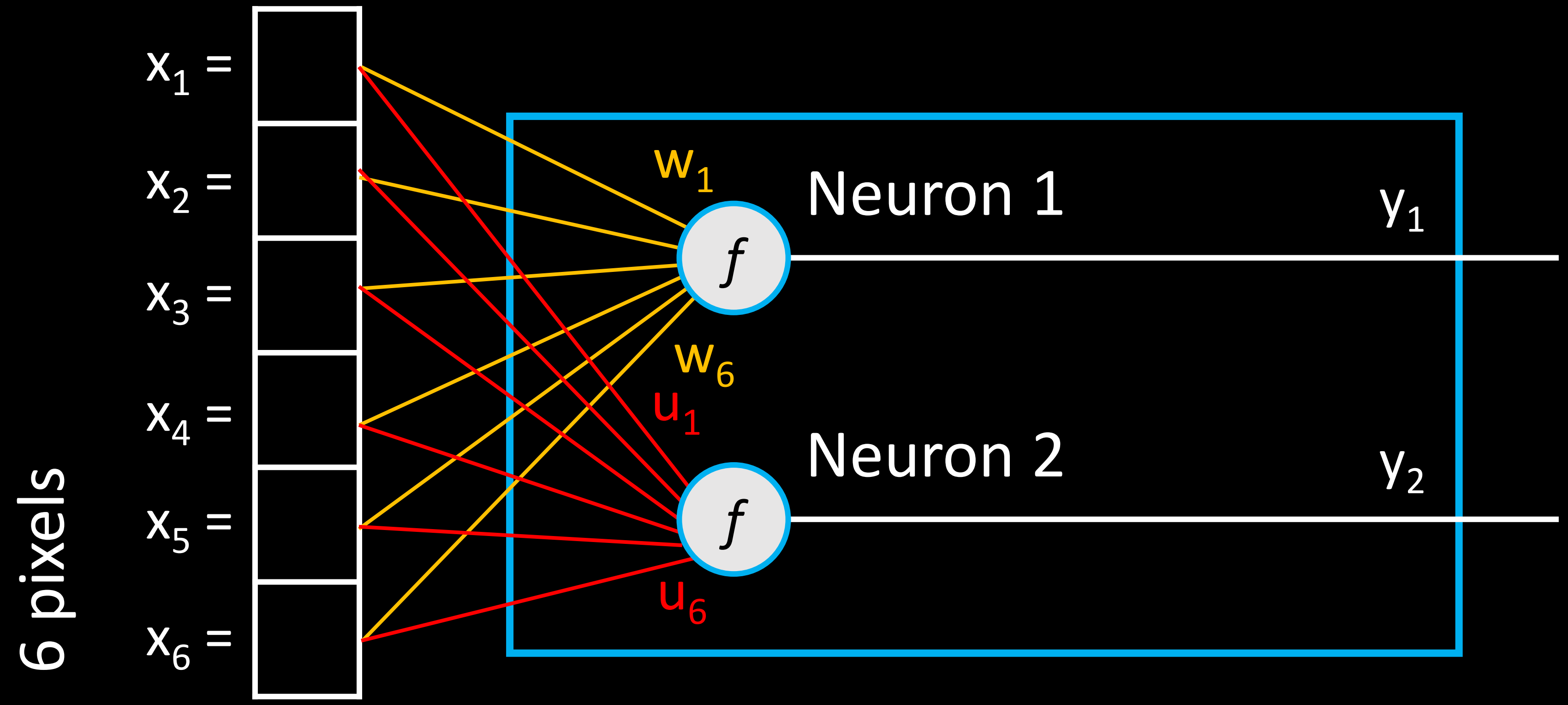
Multiple Neurons

- So far, we operated with just one Neuron – why not more?



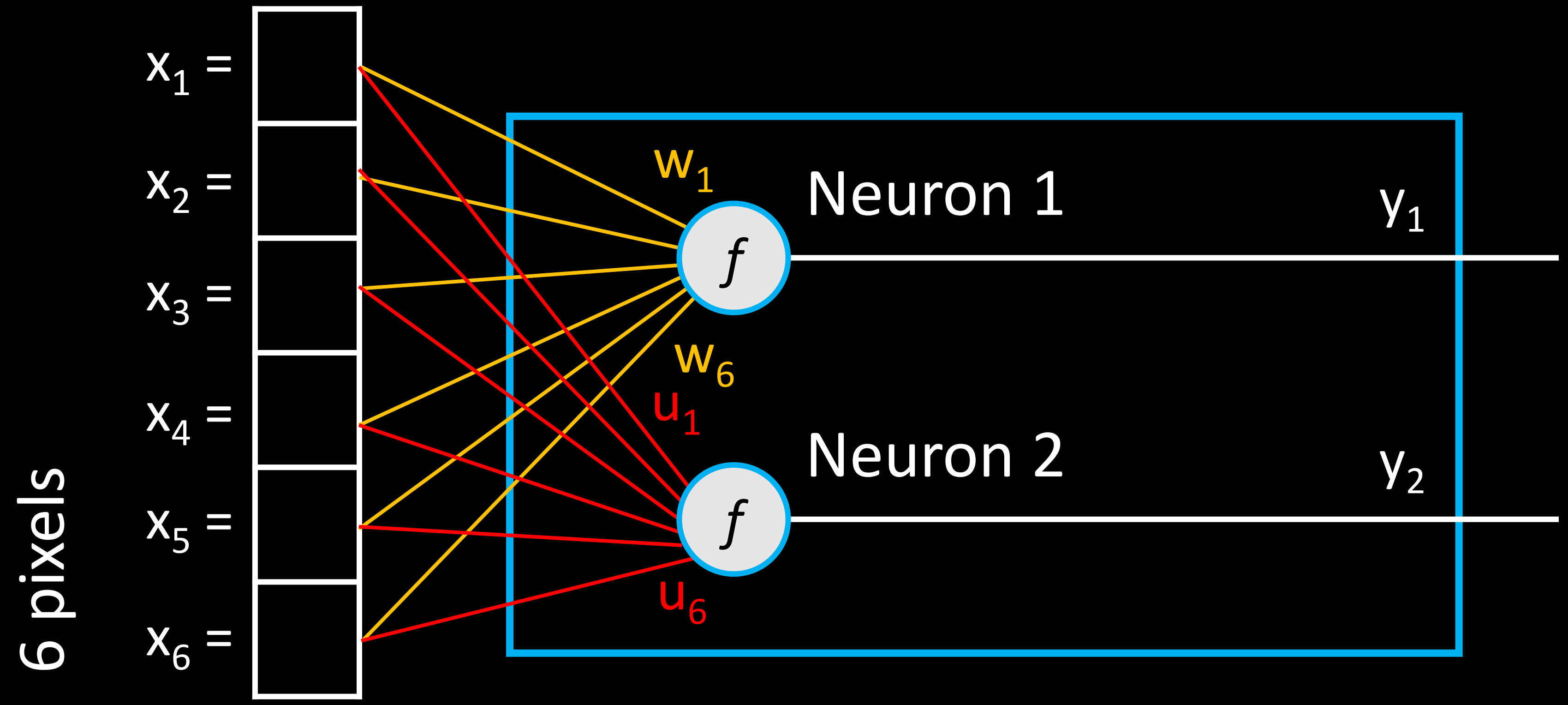
Multiple Neurons

- So far, we operated with just one Neuron – why not more?



Multiple Neurons

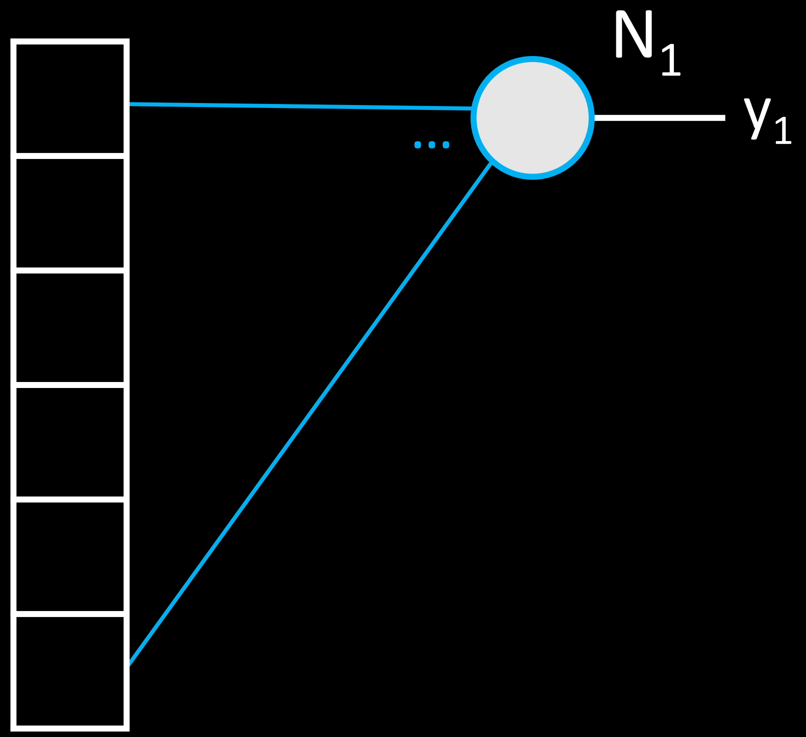
- So far, we operated with just one Neuron – why not more?



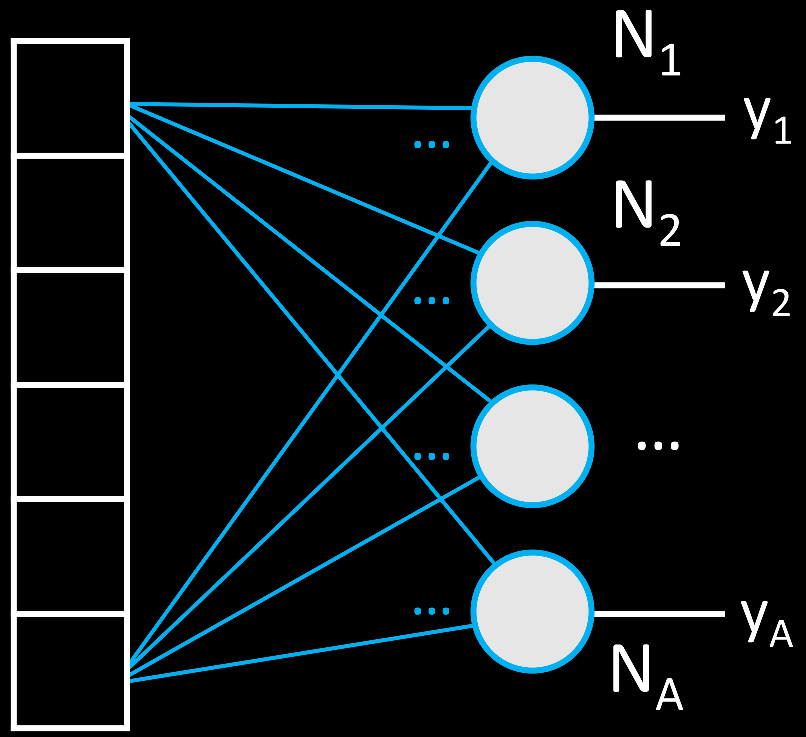
Also note that we are now outputting two numbers (**y_1, y_2**) instead of just one.

- New neuron will have its own unique weights.

Why stop there?

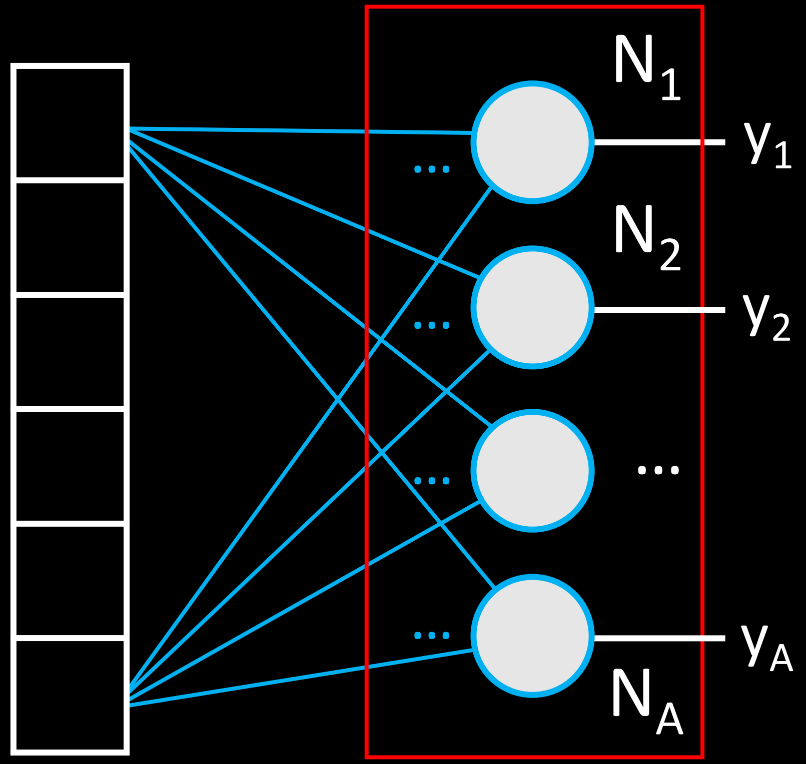


Why stop there?



- We can have many neurons next to each other.

Why stop there?

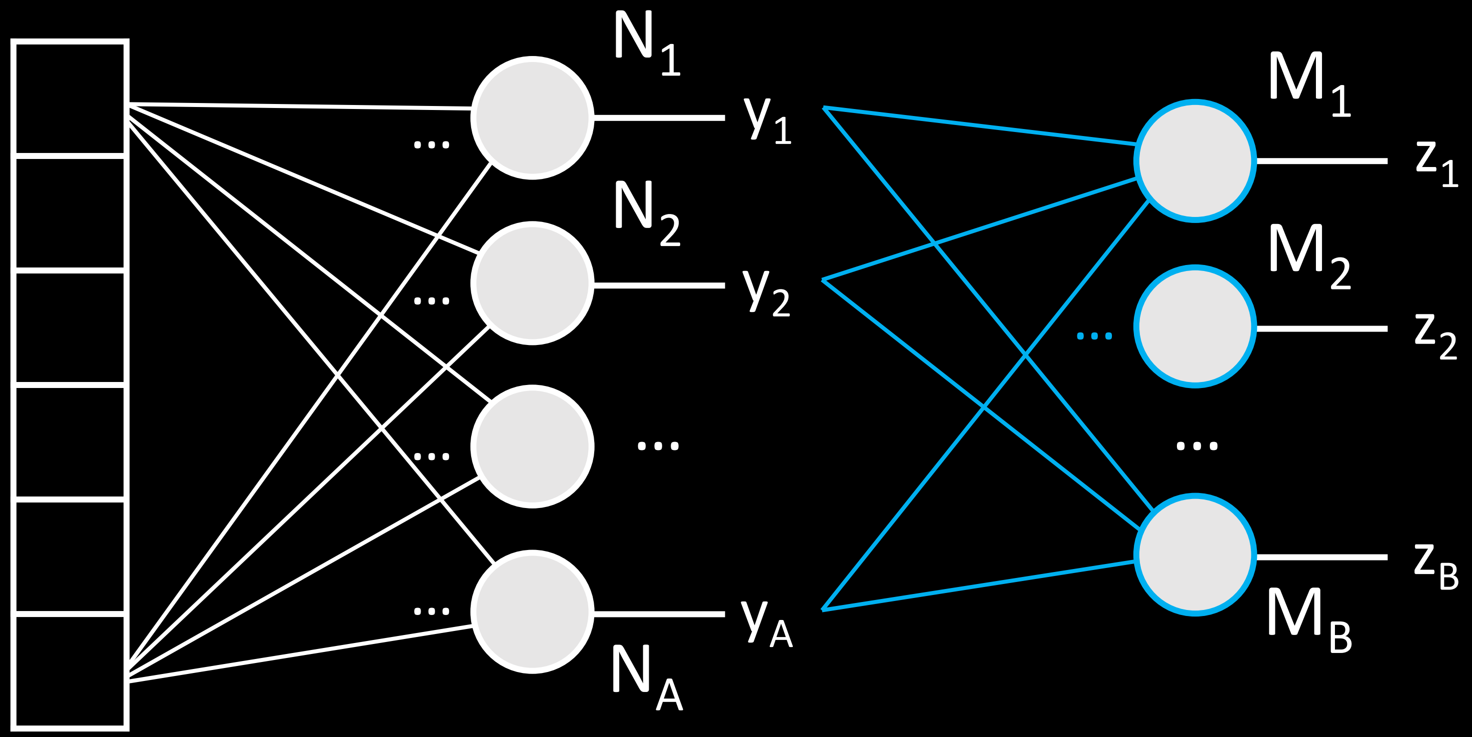


How many neurons, that many outputs:

Neurons N_1 … N_A => y_1 … y_A outputs

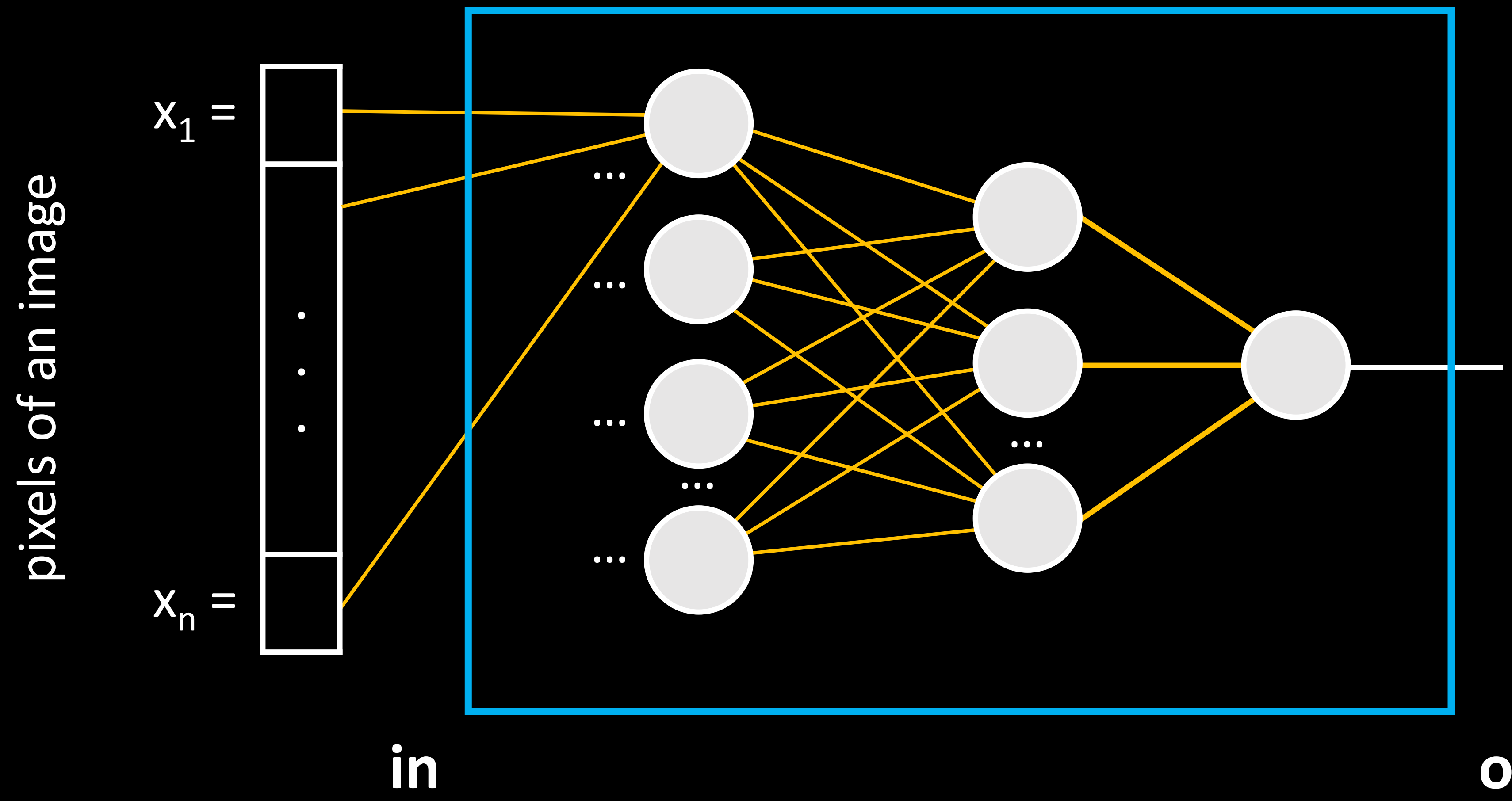
- We can have many neurons next to each other.
- We call this a single **layer of neurons**. That layer will have an output of that many numbers as the amount of neurons in it.

Why stop there? Why stop there?



- We can add another layer of neurons!
- Each neuron in the new layer will be **connected to every output of the previous layer**.

Fully connected neural network (with 3 layers):



- Imagine all the connection between the neurons (each neuron with all neuron in the previous layer). **Each of** these **connections corresponds to a parameter** (weight).

Data

x_1 =

.
.
.

x_n =

in

out

Data x
eg: image

Data y
eg: label

X:

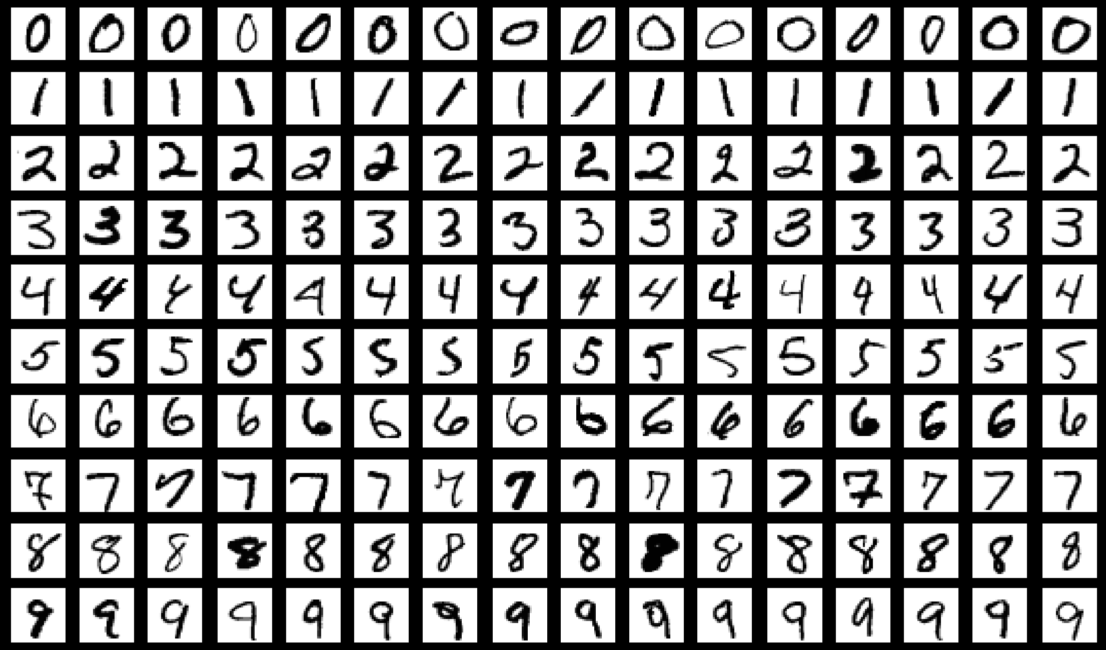
Y:

Label saying “3”

Label saying “9”

Datasets and data

- We were showing examples from a dataset of handwritten numbers which is called MNIST.
  - MNIST contains images in resolution of 28x28 pixels with numbers from “0” to “9” (each with many samples).

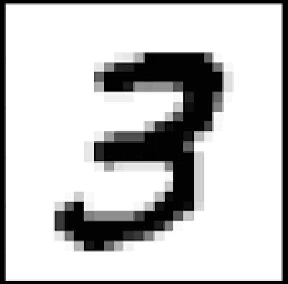


One hot vectors

- Similarly as we represent images in a special way (flattened):

28

28

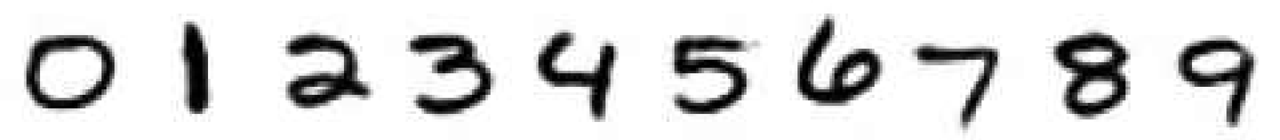


x_1 = 1, 2, 3, …, …, …, …, 783, x_{784} = 784

y_0 = 0.0
y_1 = 0.0
y_2 = 0.0
y_3 = **1.0** ← **Label saying "3"**
y_4 = 0.0
… 0.0
0.0
0.0
0.0
y_9 = 0.0

One hot vectors

X:

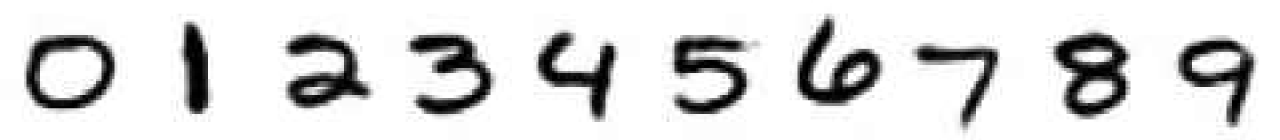


Y:

1	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1

One hot vectors

X:

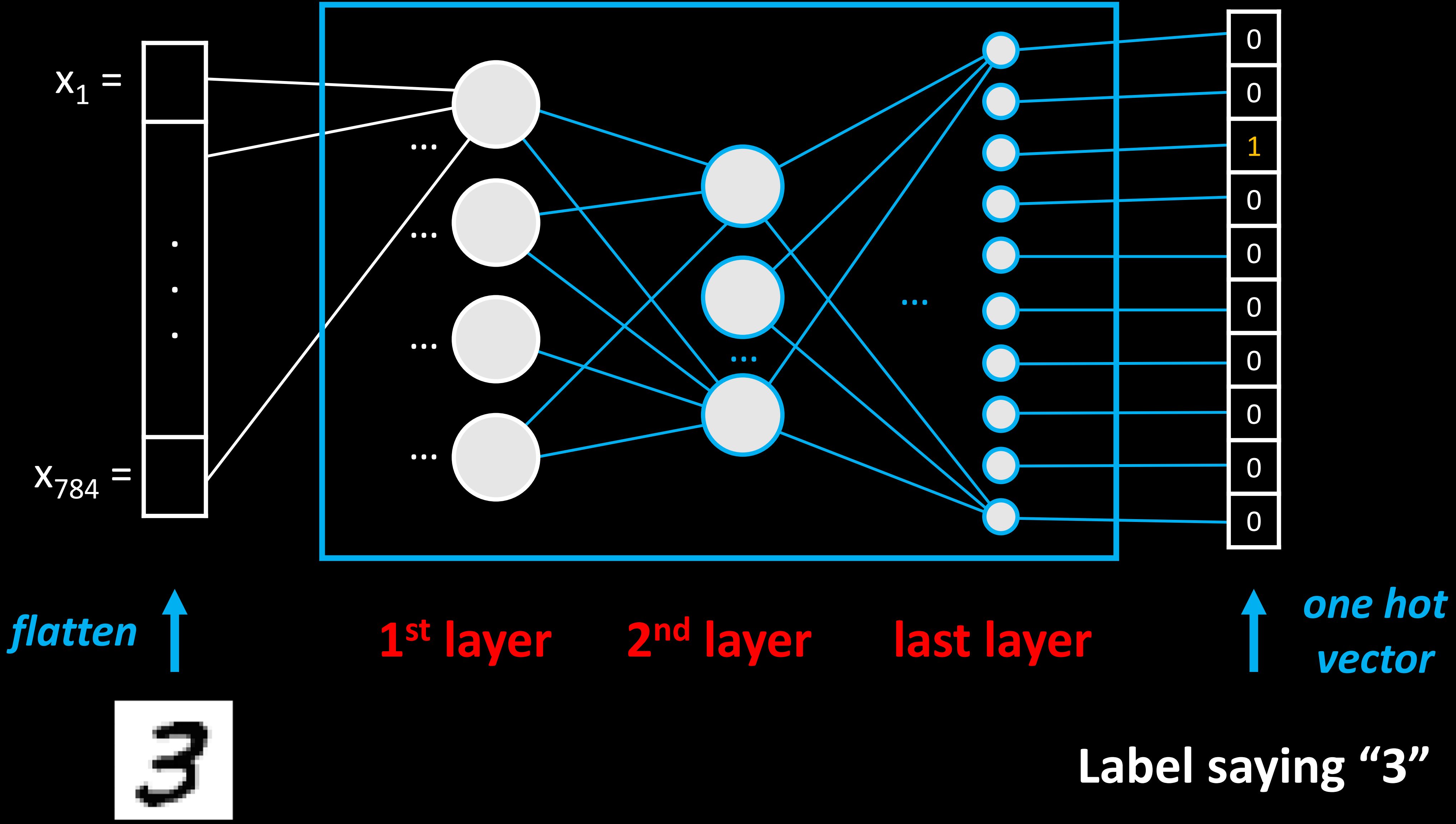


Y:

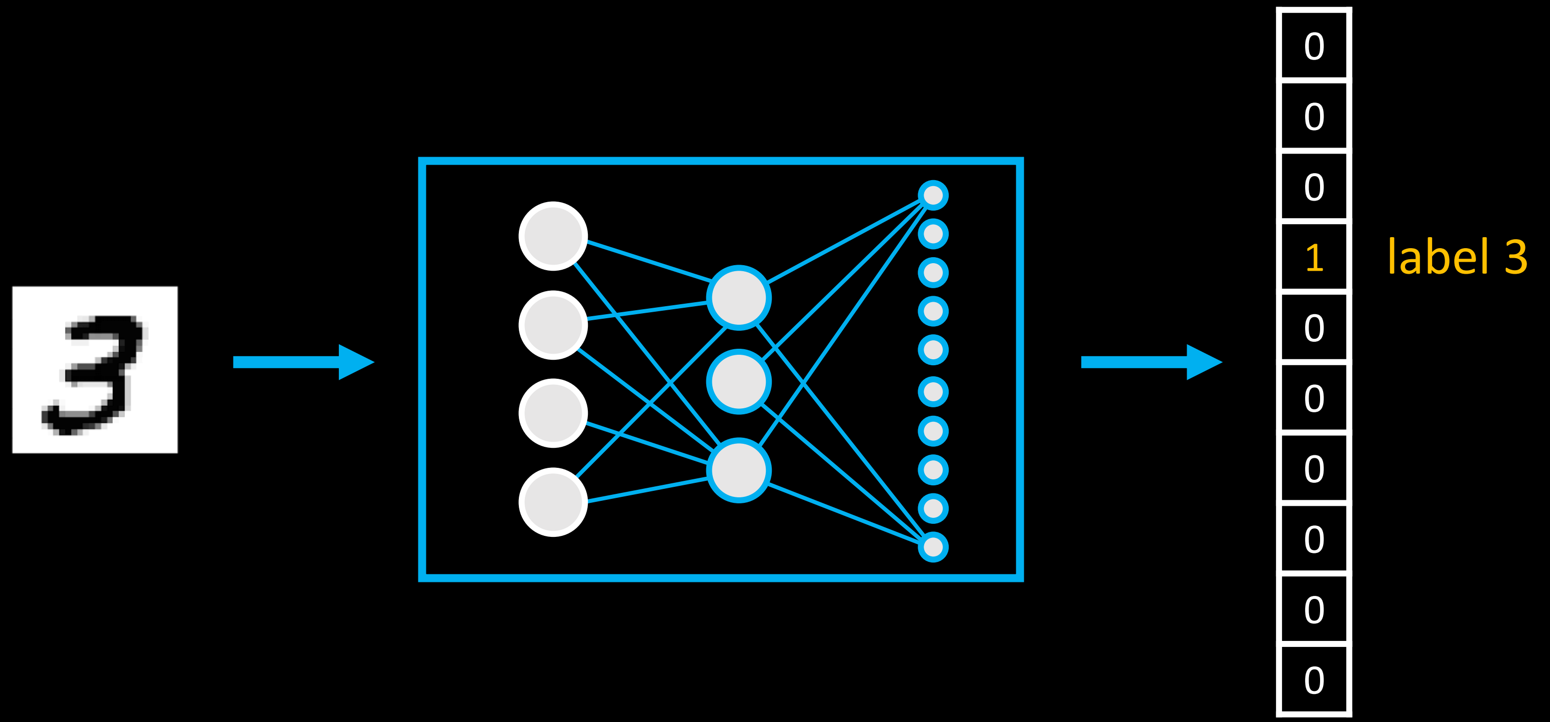
1	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1

We can say that we have **10 classes** in our dataset (numbers 0 to 9)

Fully connected neural network on MNIST



Fully connected neural network on MNIST



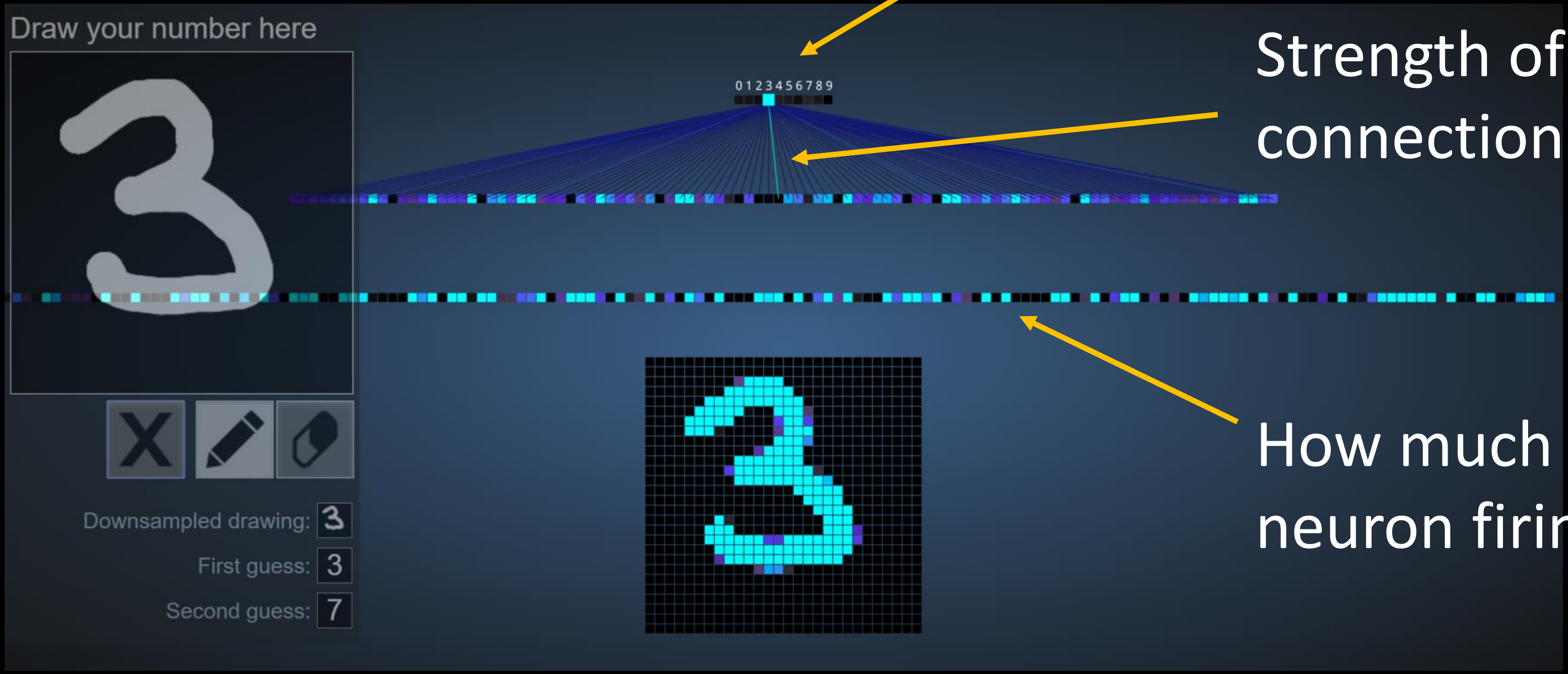
Let’s see an **interactive online visualization** of the same network trained on MNIST!

www.cs.cmu.edu/~aharley/vis/fc/flat.html

Fully connected neural network on MNIST

Model prediction

Strength of a connection



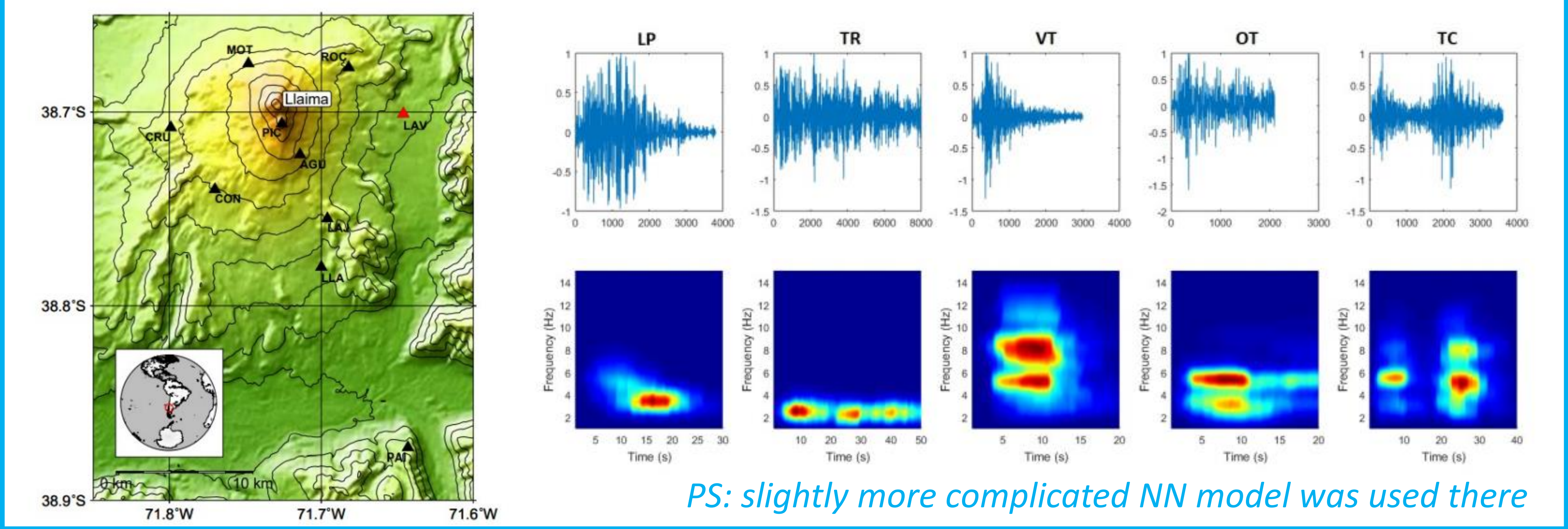
Drawn input image

How much is each neuron firing

>>> www.cs.cmu.edu/~aharley/vis/fc/flat.html <<<

Real-world example

- Classification is not just a synthetic task.
- For example: “Pattern recognition applied to seismic signals of Llaima volcano (in Chile) …”, *Journal of Volcanology and Geothermal Research* (2016)



PS: slightly more complicated NN model was used there

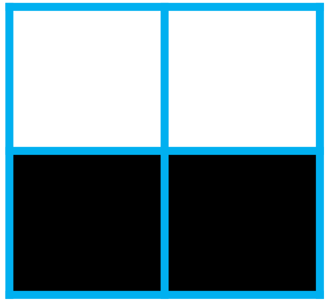
Pause 2

Homework:

- Let’s talk about today’s homework – it will consist of working with a simple **single neuron model** …
- You will be manually finding the best weights to identify an image!
- (Don’t worry it will be only 2x2 pixel image and you’ll only have 4 weights to figure out :))

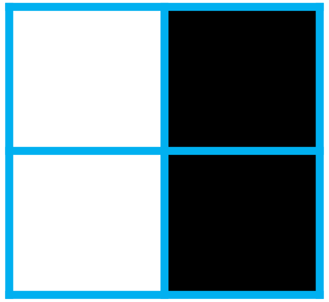
Homework:

- We have a 2 pixels times 2 pixels image as input – and we want to use a **single neuron** as a model to distinguish between **two classes**:



Horizontal line

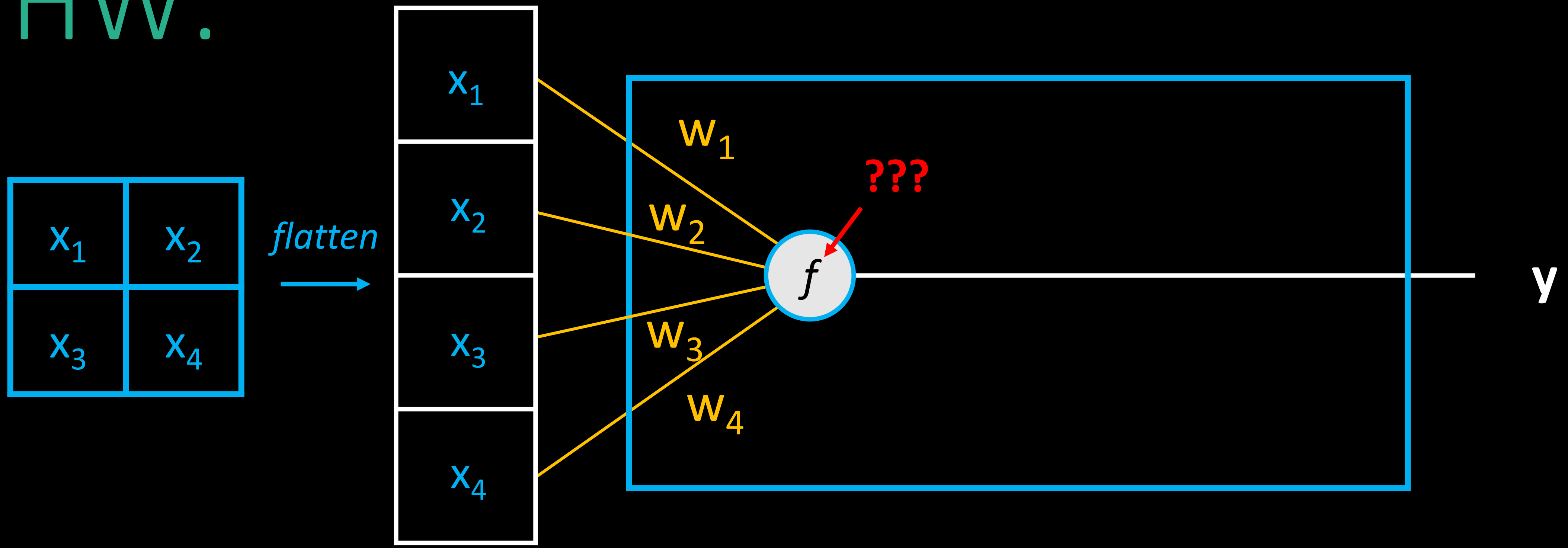
Label them as: 0.0



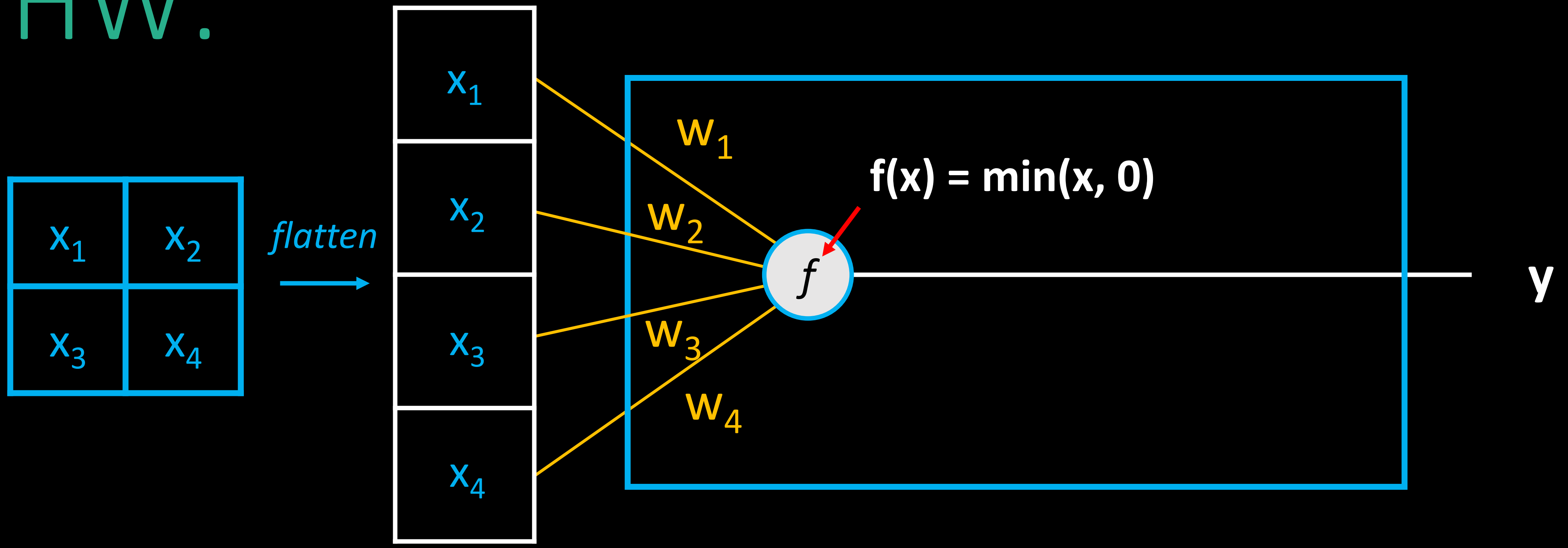
Vertical line

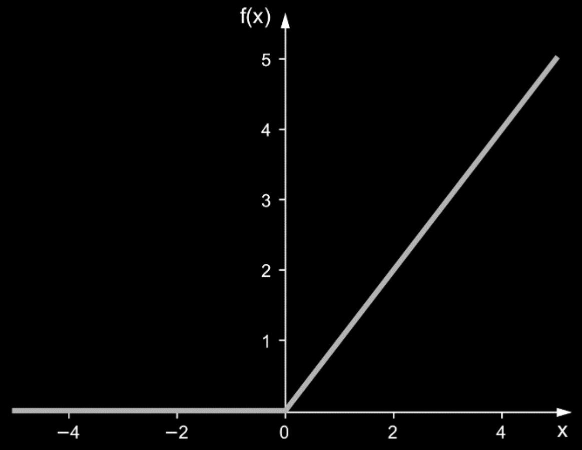
Label them as: 1.0

HW:



HW:



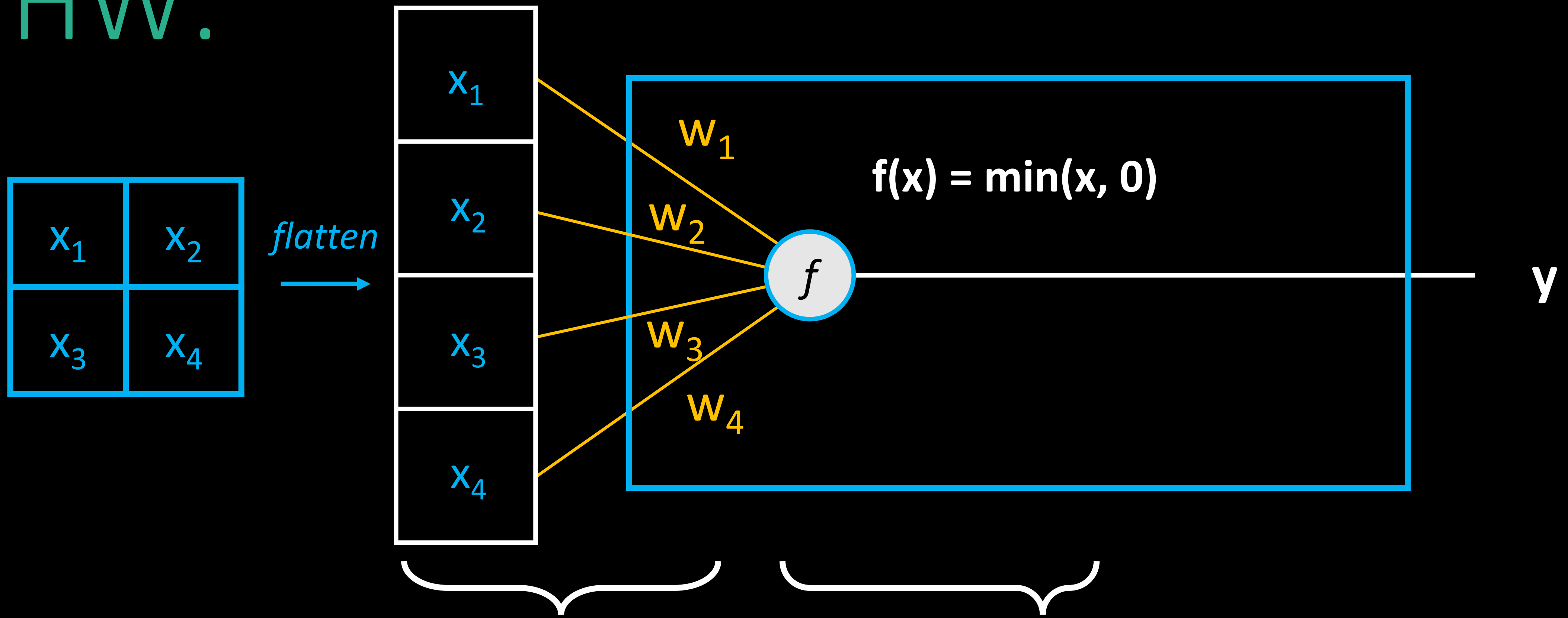


Simplification: Instead of a sigmoid function, we will have a simple function which lets through any positive number.

In general we call these f(…) **activation functions**.

This activation function is called **ReLU** (*rectified linear unit*).

HW:



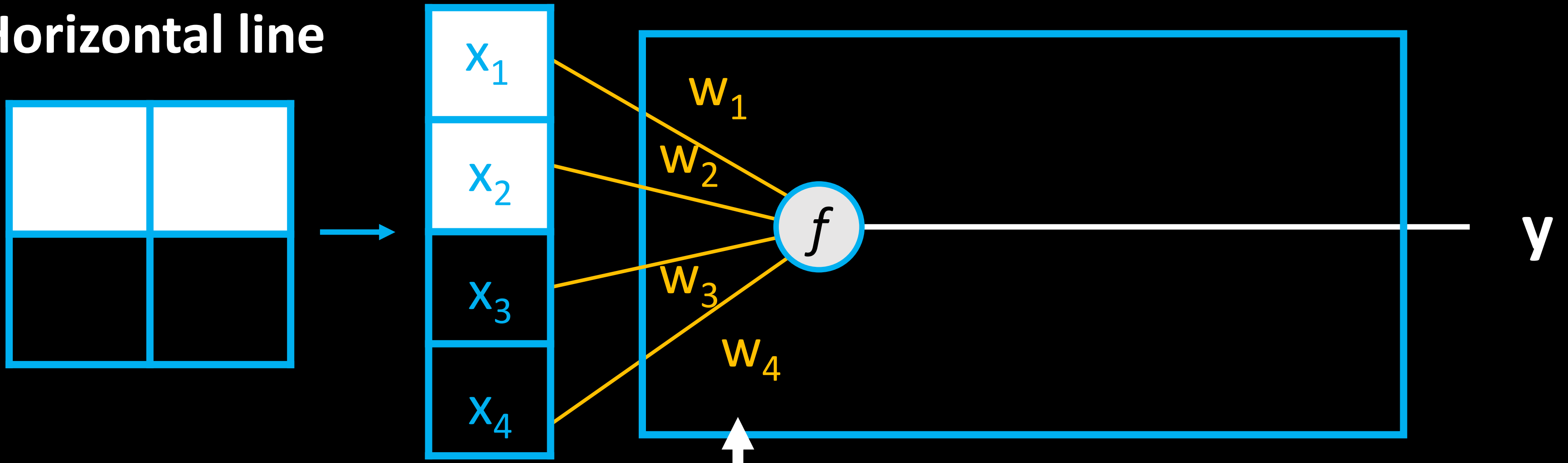
Here we first weigh all the signals ($x_i * w_i$) Then we accumulate them (= sum them).

$$\sum x_i * w_i = x_1 * w_1 + x_2 * w_2 + x_3 * w_3 + x_4 * w_4$$

Note that: all the x_i are fixed by whatever the input image is. But w_i can be set.

$f(x) = \min(x, 0)$

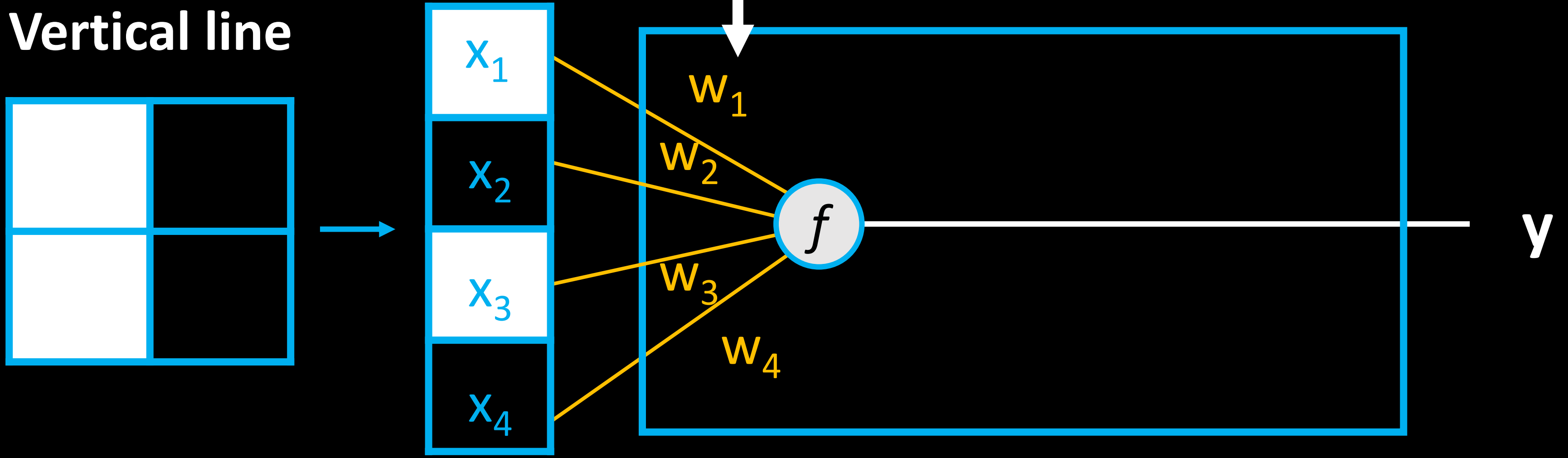
Horizontal line



Label: 0.0

PS: These are the **same weights** w_1 to w_4 (but the **inputs** x_1 to x_4 **are different** in each case)

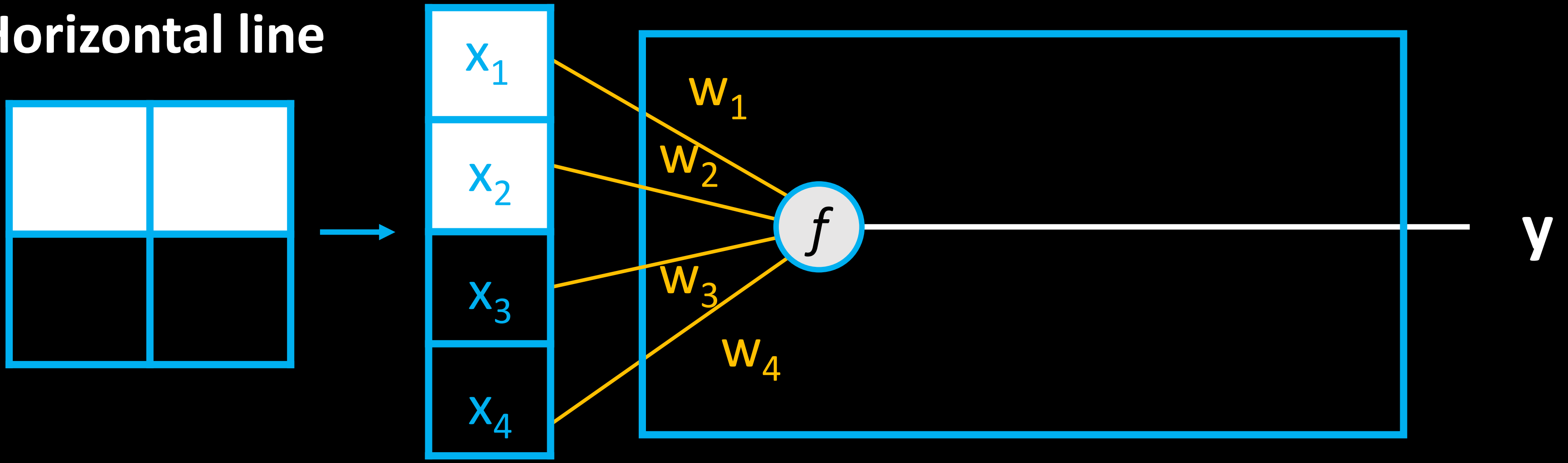
Vertical line



Label: 1.0

$f(x) = \min(x, 0)$

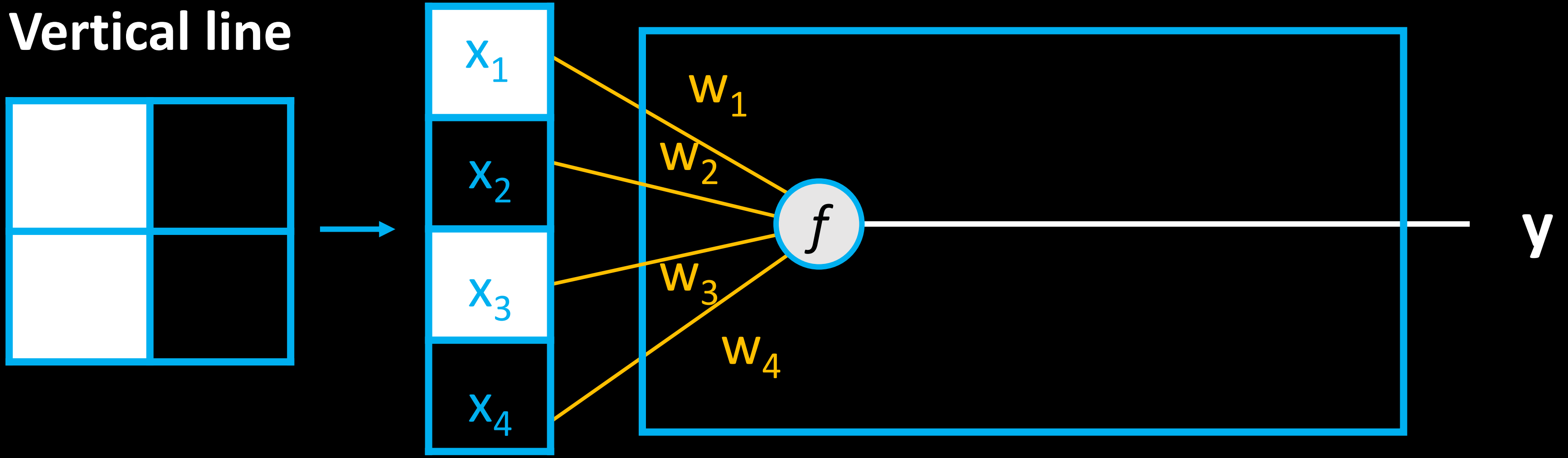
Horizontal line



Label: 0.0

$$f(x_1 * w_1 + x_2 * w_2 + x_3 * w_3 + x_4 * w_4) = \text{number which I want to be } 0.0$$

Vertical line

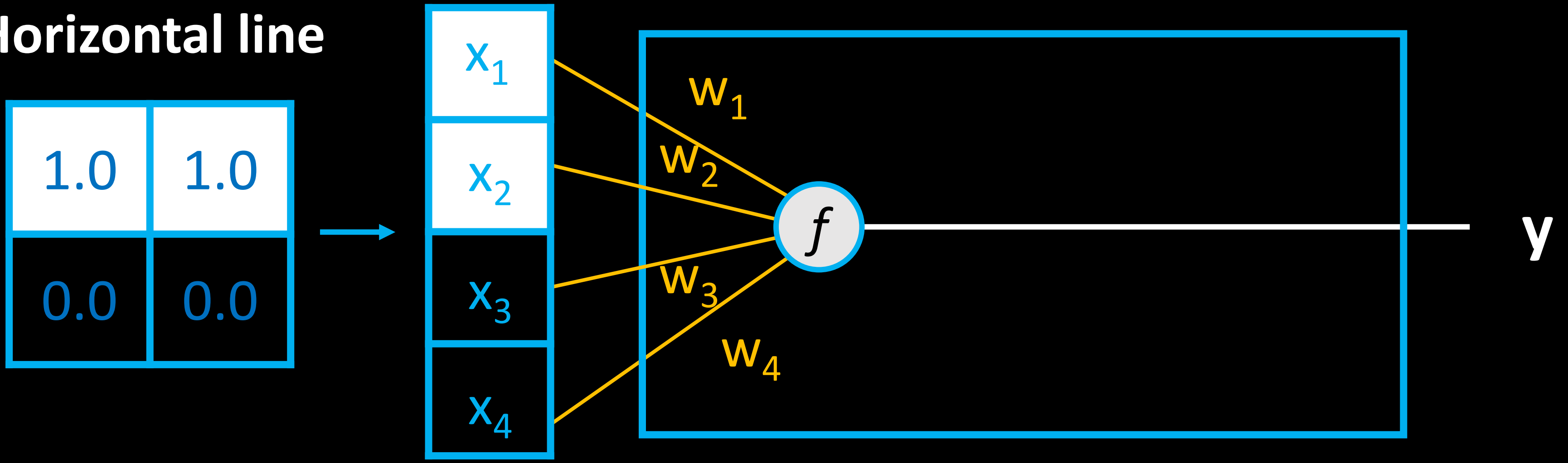


Label: 1.0

$$f(x_1 * w_1 + x_2 * w_2 + x_3 * w_3 + x_4 * w_4) = \text{number which I want to be } 1.0$$

$f(x) = \min(x, 0)$

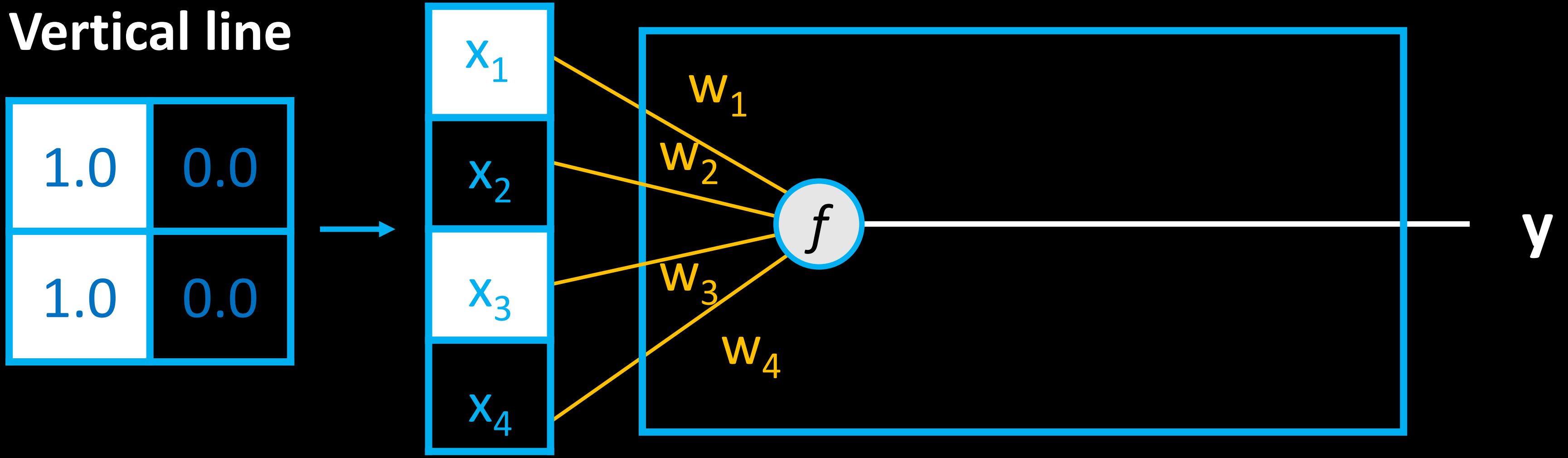
Horizontal line



Label: 0.0

$$f(\,1.0 * w_1 + 1.0 * w_2 + 0.0 * w_3 + 0.0 * w_4\,)$$ = number which I want to be **0.0**

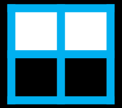
Vertical line



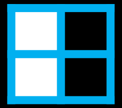
Label: 1.0

$$f(\,1.0 * w_1 + 0.0 * w_2 + 1.0 * w_3 + 0.0 * w_4\,)$$ = number which I want to be **1.0**

x: y:



Label: 0.0



Label: 1.0

Homework:

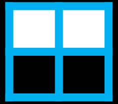
f(x) = min(x, 0)

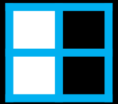
f($1.0 * w_1 + 1.0 * w_2 + 0.0 * w_3 + 0.0 * w_4$) = number which I want to be **0.0**

f($1.0 * w_1 + 0.0 * w_2 + 1.0 * w_3 + 0.0 * w_4$) = number which I want to be **1.0**

Homework:

x: **y:**

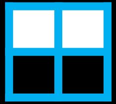
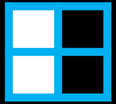
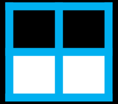
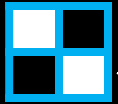
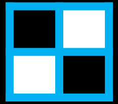
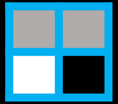
 **Label: 0.0**

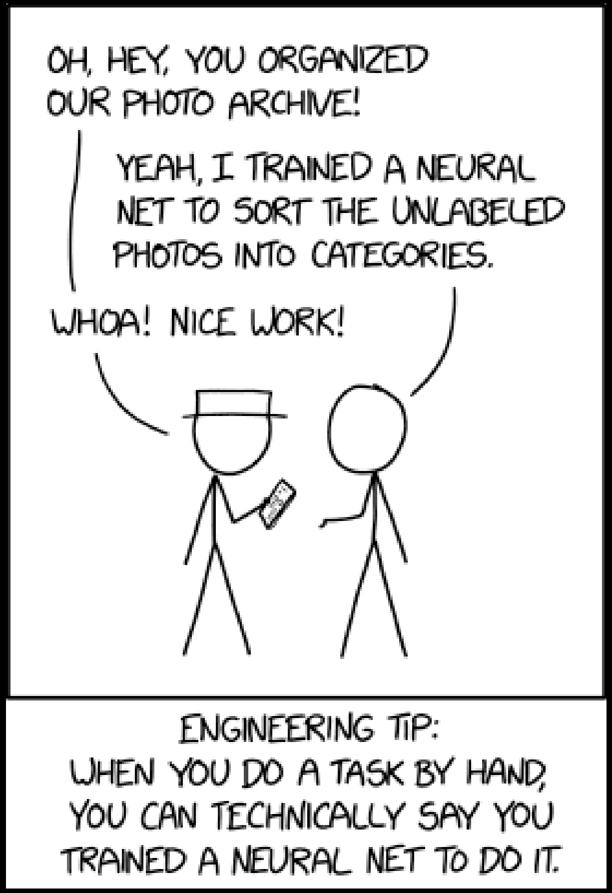
 **Label: 1.0**

f(x) = min(x, 0)

$f(\,1.0 * w_1 + 1.0 * w_2 + 0.0 * w_3 + 0.0 * w_4\,)$ = number which I want to be **0.0**

$f(\,1.0 * w_1 + 0.0 * w_2 + 1.0 * w_3 + 0.0 * w_4\,)$ = number which I want to be **1.0**

- **Task:** Manually find values for w_1, w_2, w_3, w_4, which would do the classification between  and  images. Horizontal/Vertical classifier!
- **Bonus question:** *Would this work well with some new types of images? What would happen for*  *,*  *,*  *or*  *?*



xkcd.com/2173/

Training a neural network model

How do we find the best parameters?

- Manually!

Training a neural network model

How do we find the best parameters?

- Manually! It's tedious! Your homework is easy because it has just 4 weights to figure out. What if we have more (*way more*)?

Training a neural network model

How do we find the best parameters?

- Manually! It's tedious! Your homework is easy because it has just 4 weights to figure out. What if we have more (*way more*)? ❌
- Automatically! Brute force? Guess all the w_i randomly and save the ones which work the best? Sure … would work … but it would take ages! ❌

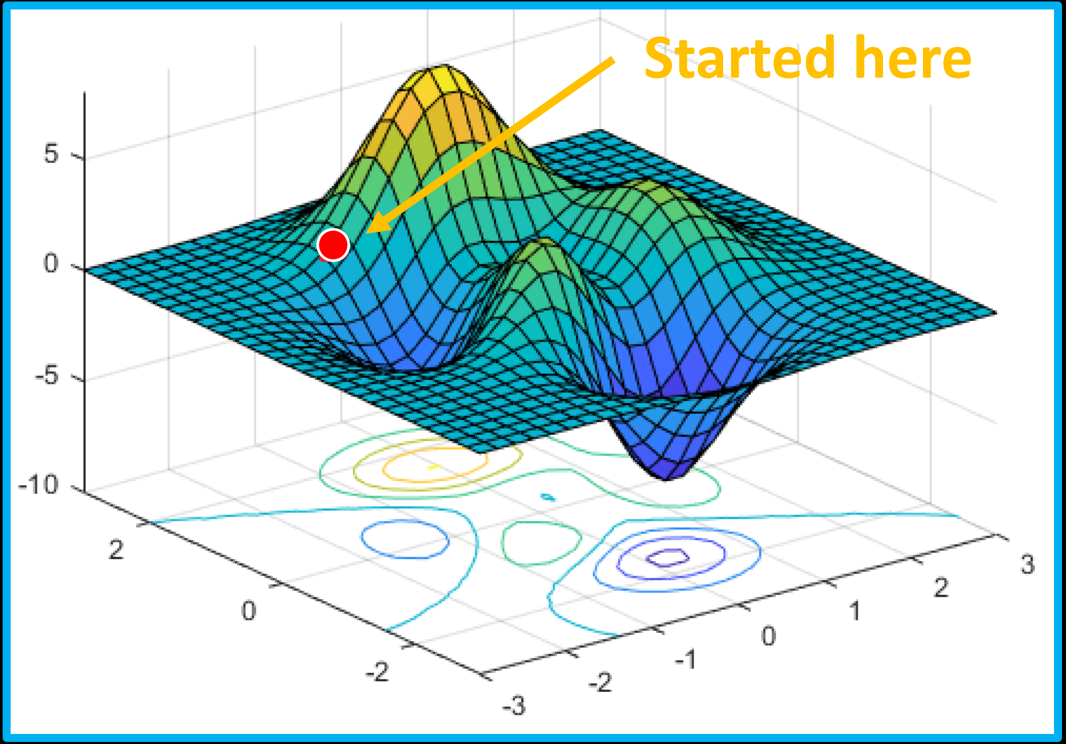
Training a neural network model

How do we find the best parameters?

- Manually! It's tedious! Your homework is easy because it has just 4 weights to figure out. What if we have more (*way more*)?
- Automatically! Brute force? Guess all the w_i randomly and save the ones which work the best? Sure … would work … but it would take ages!
- Automatically with a smart algorithm!

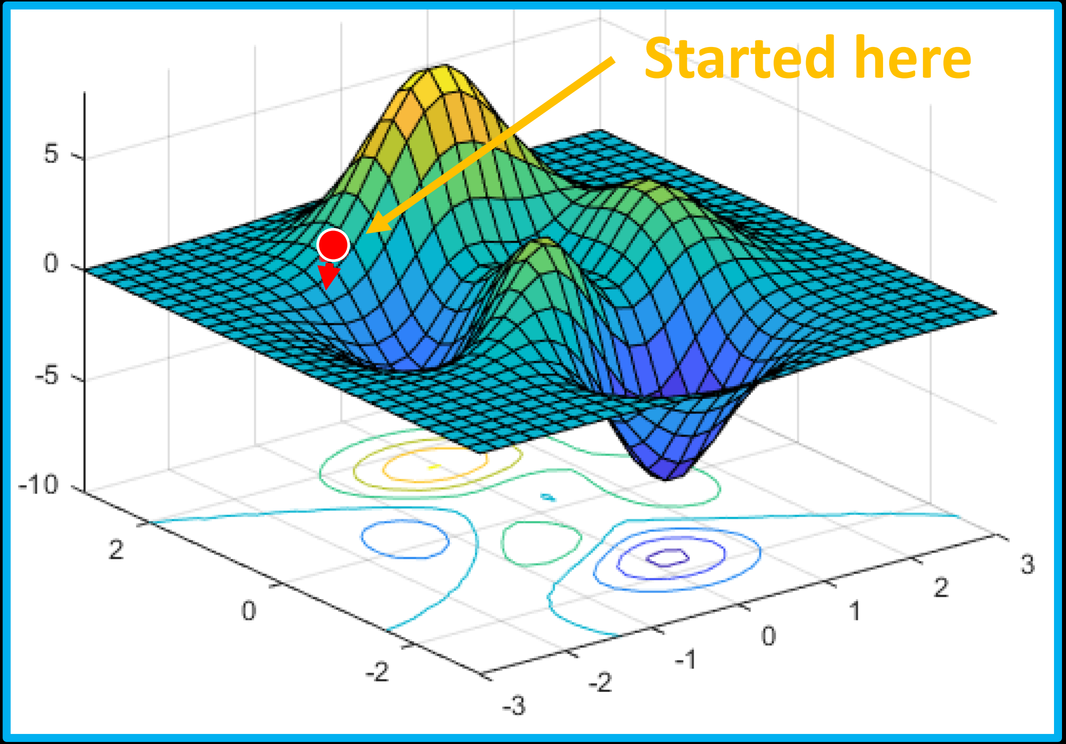
Side step: Gradient descent

Imagine you are in a landscape and want to ~~climb a mountain~~ find the deepest hole. Also it’s *really foggy* and you can only check your near surroundings …



Side step: Gradient descent

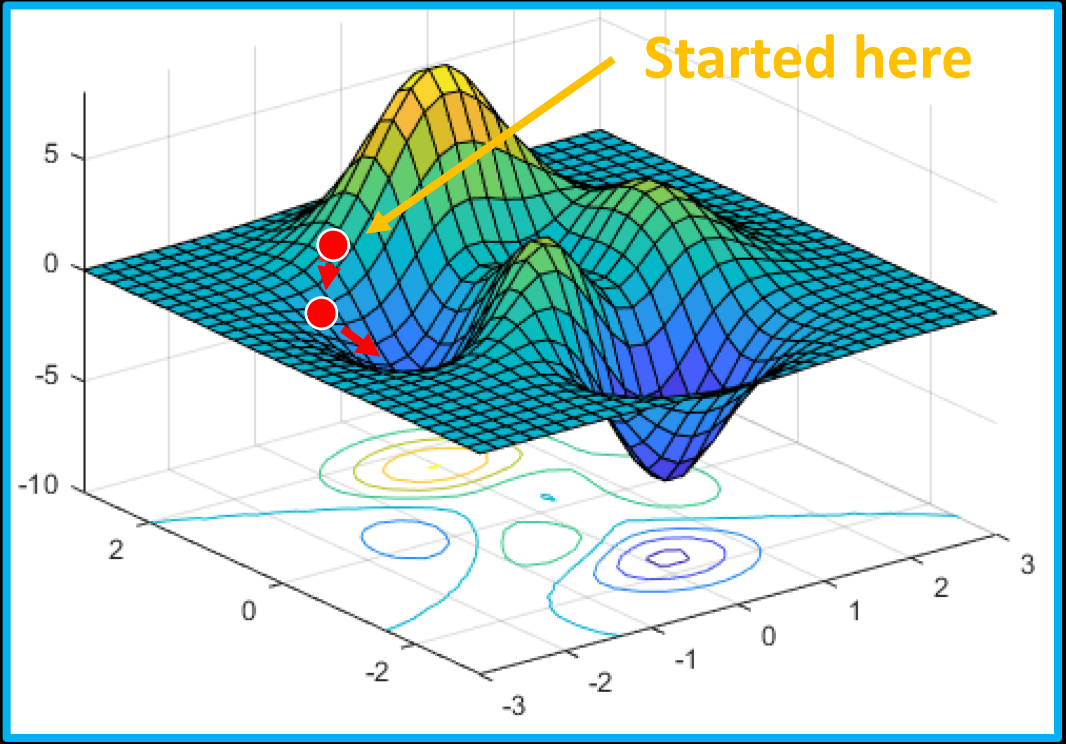
Imagine you are in a landscape and want to ~~climb a mountain~~ find the deepest hole. Also it's *really foggy* and you can only check your near surroundings …



- **We start somewhere** and we check around – we find the direction with the **most descending slope**. (We can call this a gradient)

Side step: Gradient descent

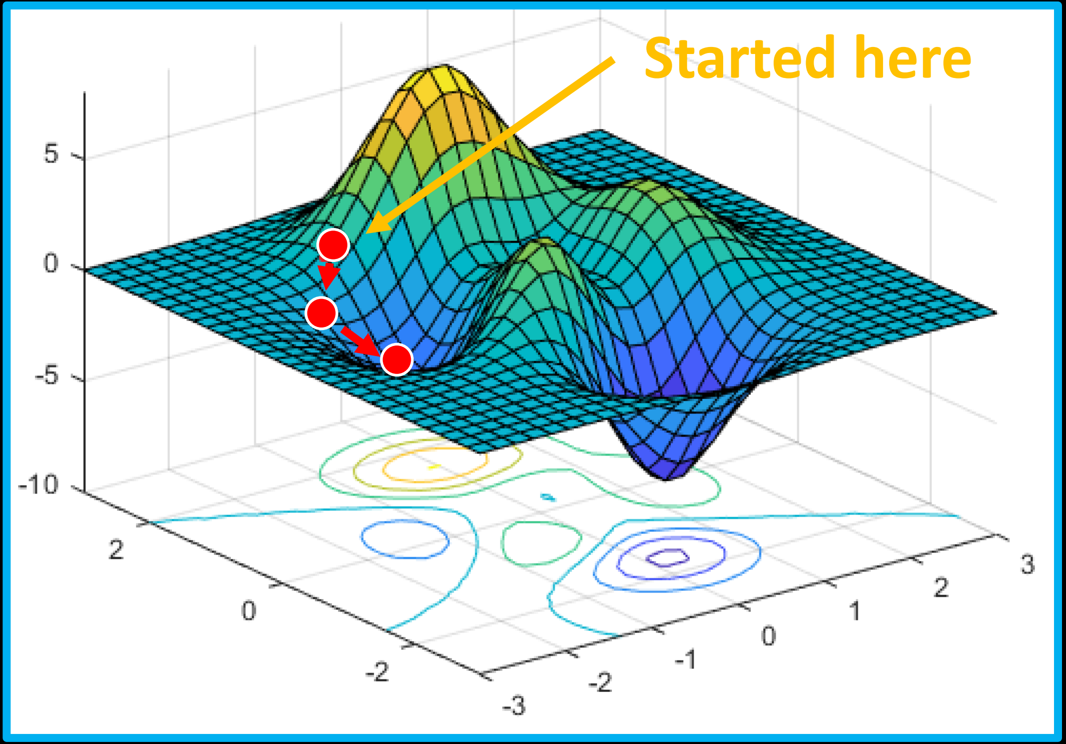
Imagine you are in a landscape and want to ~~climb a mountain~~ find the deepest hole. Also it's *really foggy* and you can only check your near surroundings …



- **We start somewhere** and we check around – we find the direction with the **most descending slope**. (We can call this a gradient)
- We **take a step** in that direction …
- … and check again!

Side step: Gradient descent

Imagine you are in a landscape and want to ~~climb a mountain~~ find the deepest hole. Also it's *really foggy* and you can only check your near surroundings …

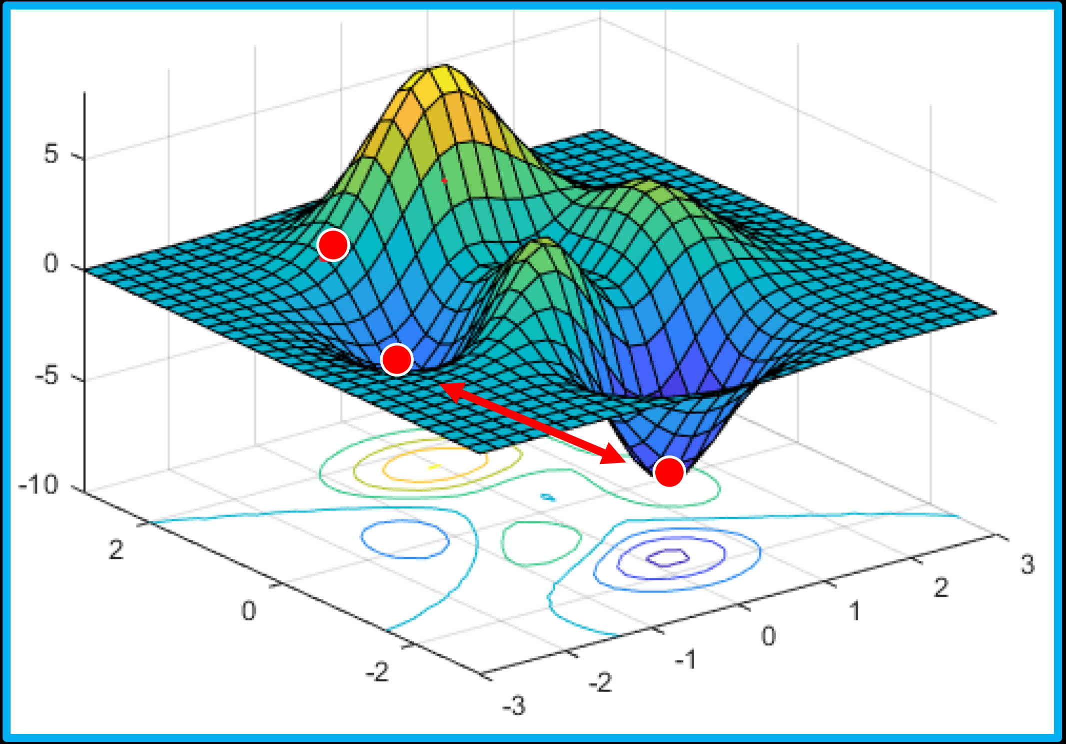


- **We start somewhere** and we check around – we find the direction with the **most descending slope**. (We can call this a gradient)
- We **take a step** in that direction …
- … and check again!

- Over time we will **end up in a minimum** (*also known as* a hole *in this metaphor*)!

Side step: Gradient descent

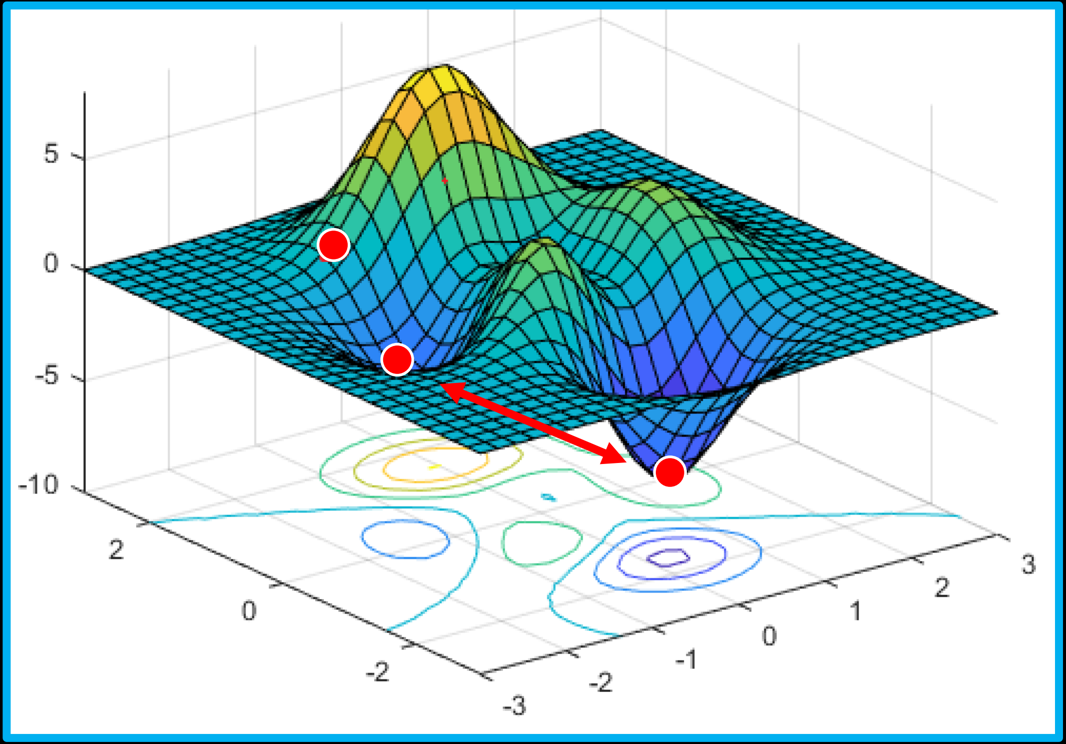
Imagine you are in a landscape and want to ~~climb a mountain~~ find the deepest hole. Also it's *really foggy* and you can only check your near surroundings …



- With this approach we can miss some obvious better solutions just behind a hill …

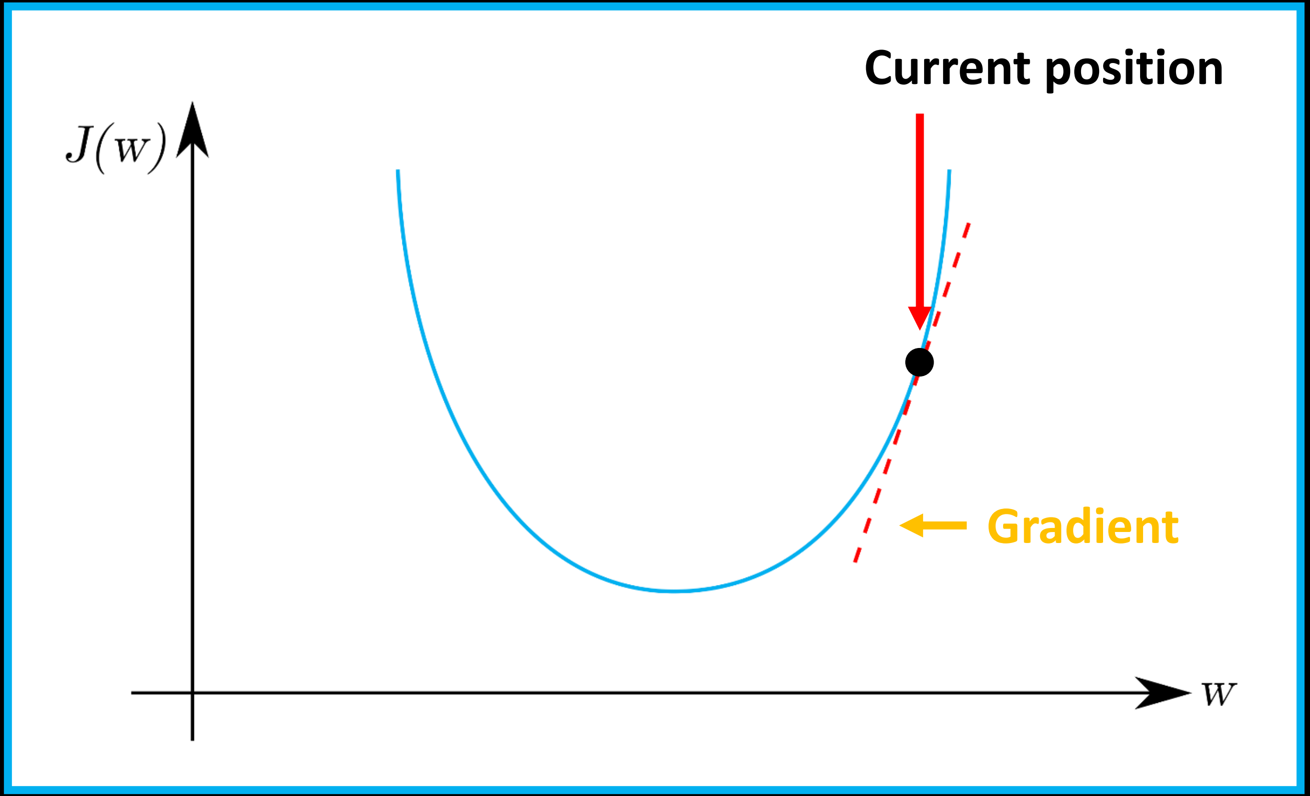
Side step: Gradient descent

Imagine you are in a landscape and want to ~~climb a mountain~~ find the deepest hole. Also it's *really foggy* and you can only check your near surroundings …



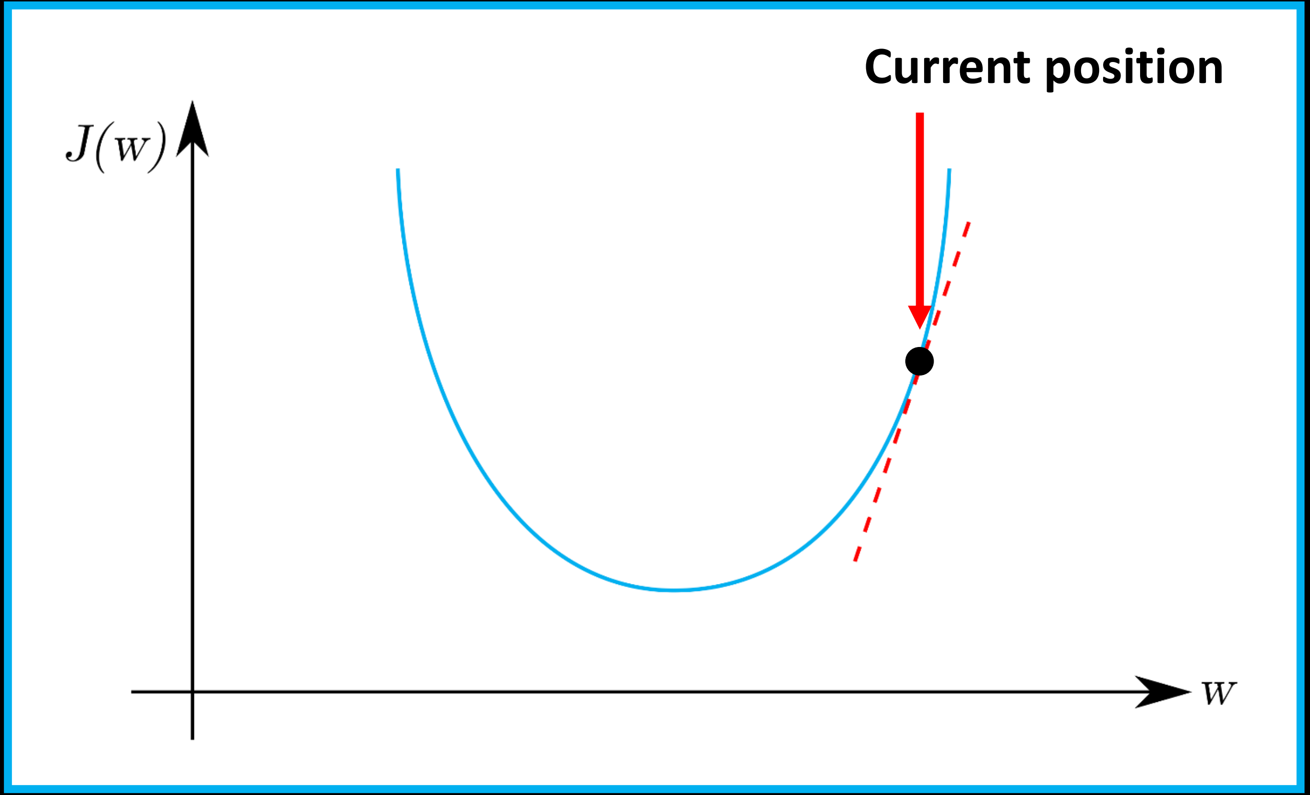
- With this approach we can miss some obvious better solutions just behind a hill …
- This is called finding a **local minimum** and missing the **global minimum**.
- *(Beyond current lesson) There are some approaches how to get around this, for example trying out many starting locations.*

Gradient descent



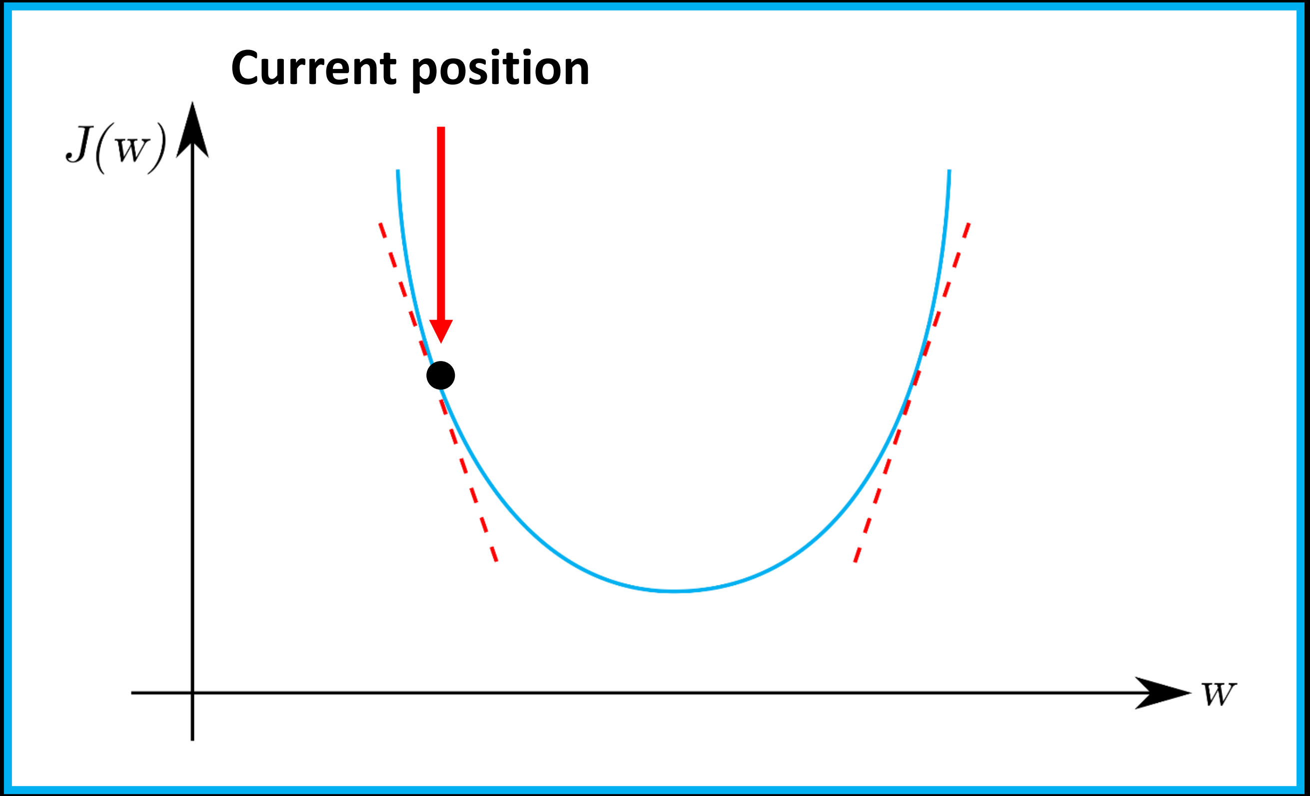
- **Gradient** – we can derive a function at a position to discover what is the slope of the function at that location

Gradient descent



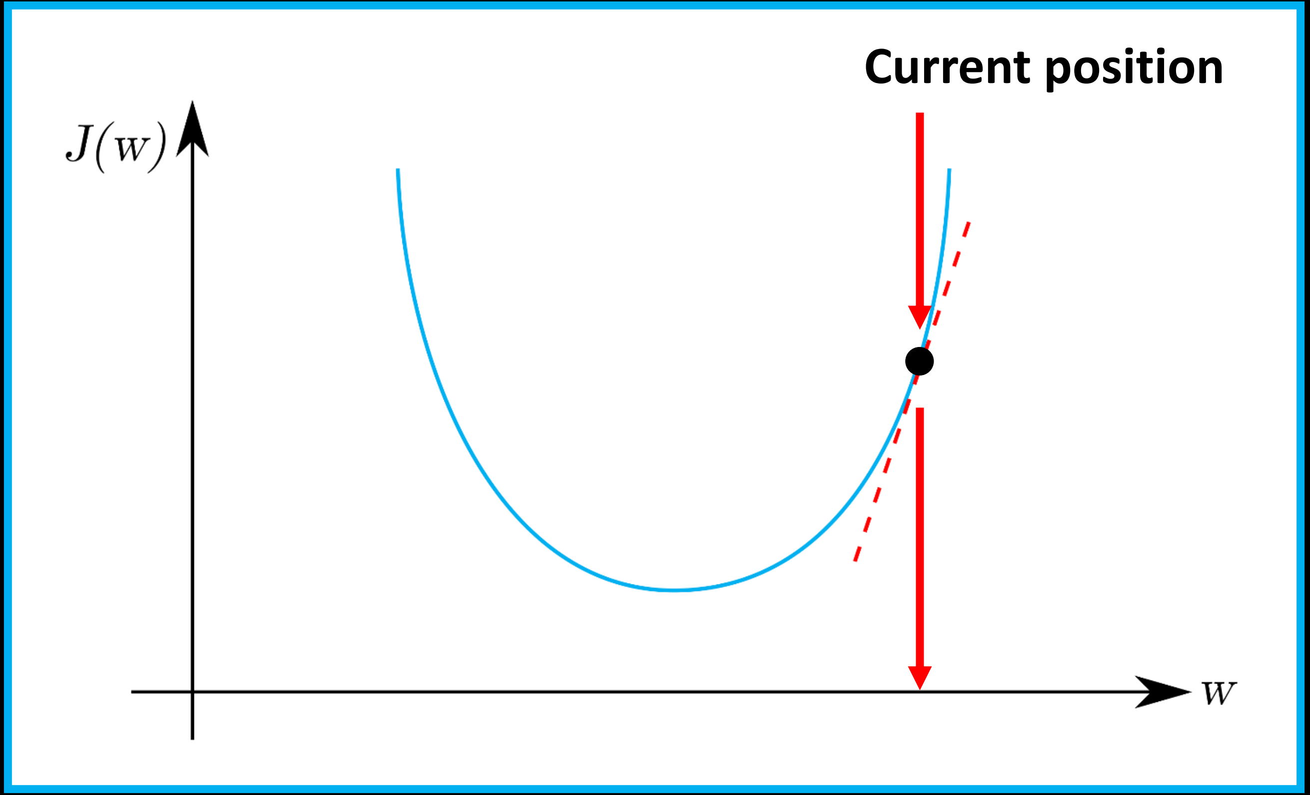
- **Gradient** – we can derive a function at a position to discover what is the slope of the function at that location
- One tangent has a **positive slope** – this means we would go to the **left** from the current position
  - **direction** = left

Gradient descent



- **Gradient** – we can derive a function at a position to discover what is the slope of the function at that location
- One tangent has a **positive slope** – this means we would go to the **left** from the current position
  - **direction** = left
- Another tangent has a **negative slope** – this would mean to go to the **right**
  - **direction** = right

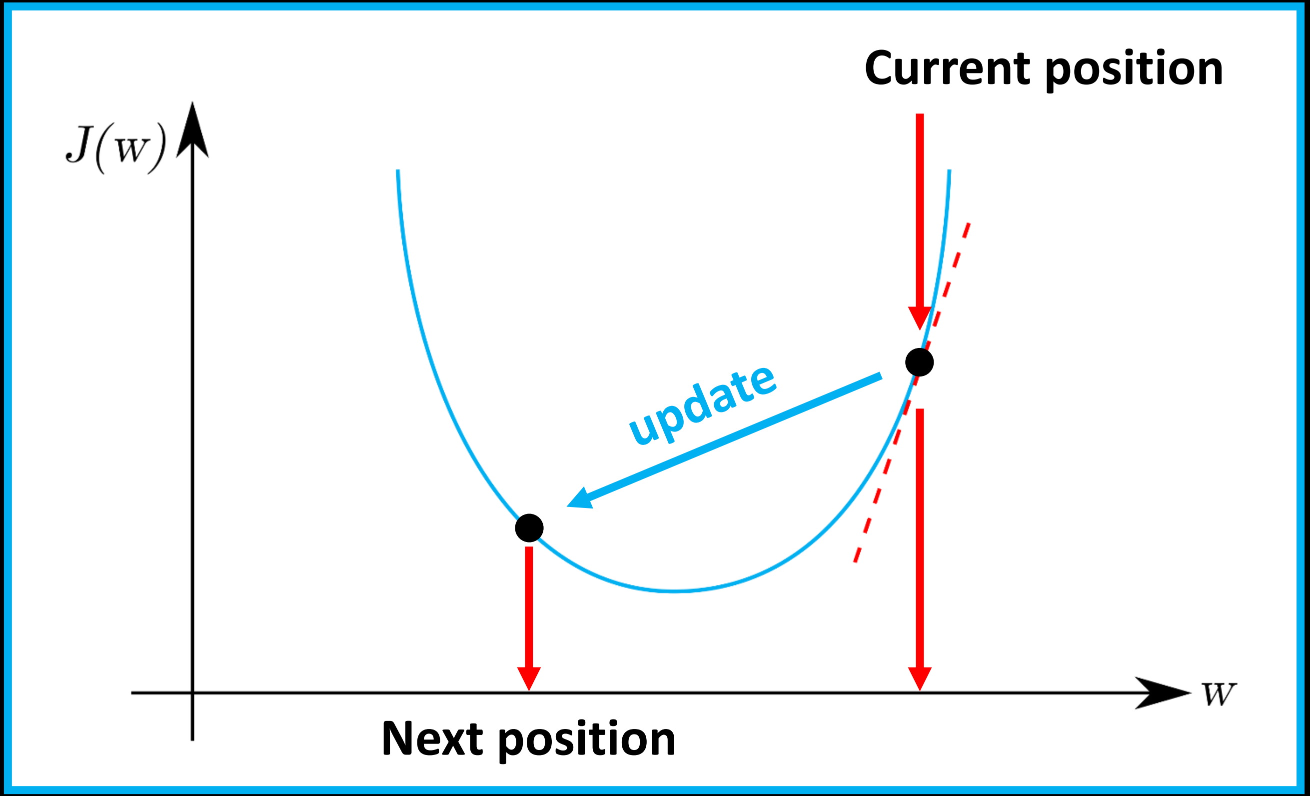
Gradient descent



- **Gradient** – we can derive a function at a position to discover what is the slope of the function at that location
- One tangent has a **positive slope** – this means we would go to the **left** from the current position
  - **direction** = -1
- Another tangent has a **negative slope** – this would mean to go to the **right**
  - **direction** = +1

Next Position = **Current Position** + **direction** * step size

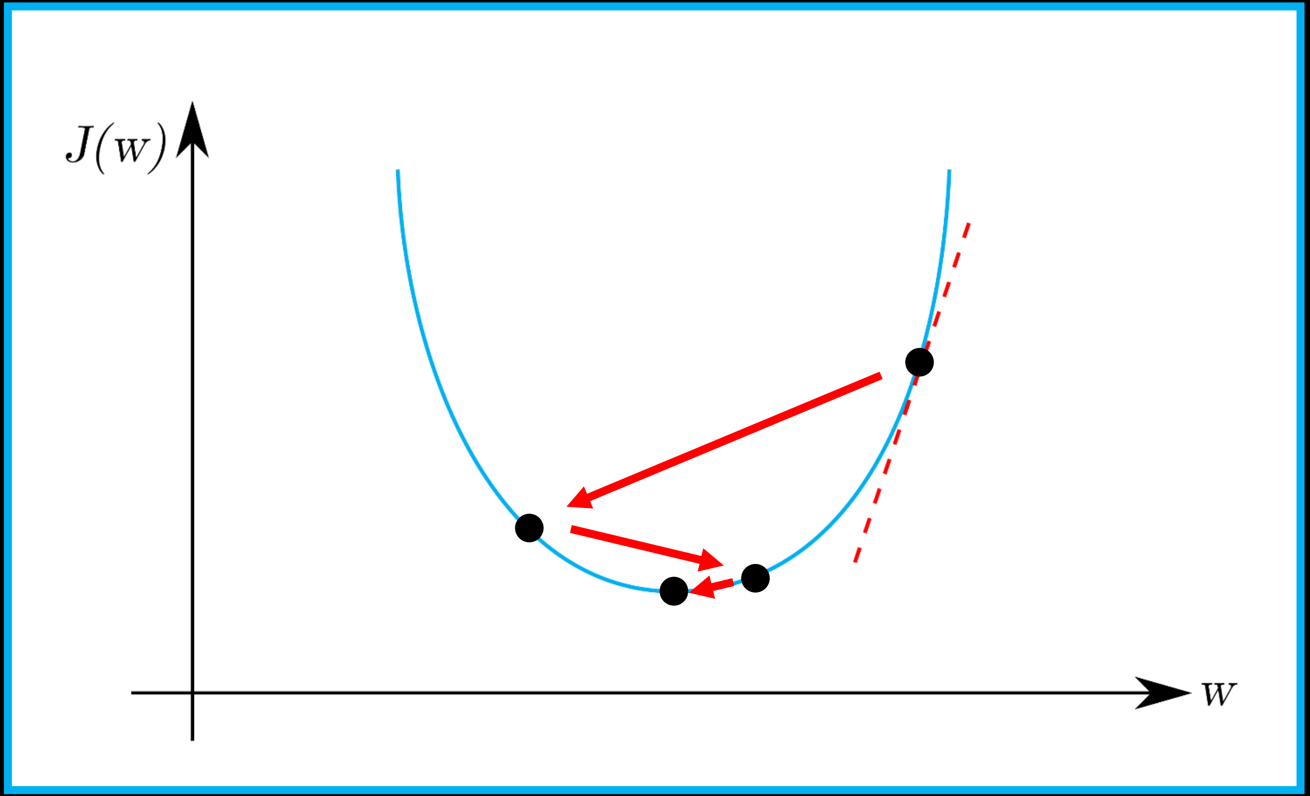
Gradient descent



- **Gradient** – we can derive a function at a position to discover what is the slope of the function at that location
- One tangent has a **positive slope** – this means we would go to the **left** from the current position
  - **direction** = -1
- Another tangent has a **negative slope** – this would mean to go to the **right**
  - **direction** = +1

Next Position = **Current Position** + **direction** * step size

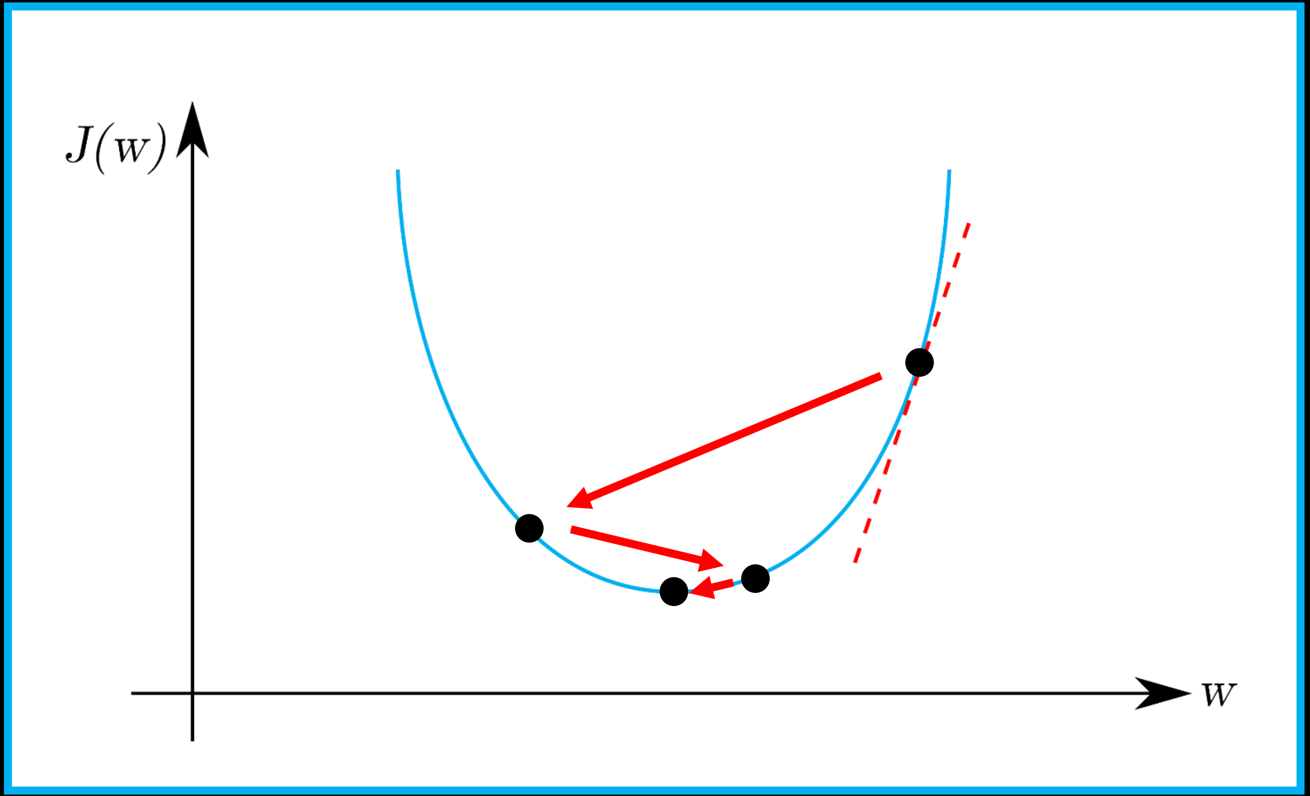
Gradient descent



- Iterative algorithm with which we will eventually find a local minimum

Next Position = **Current Position** + **direction** * step size

Gradient descent



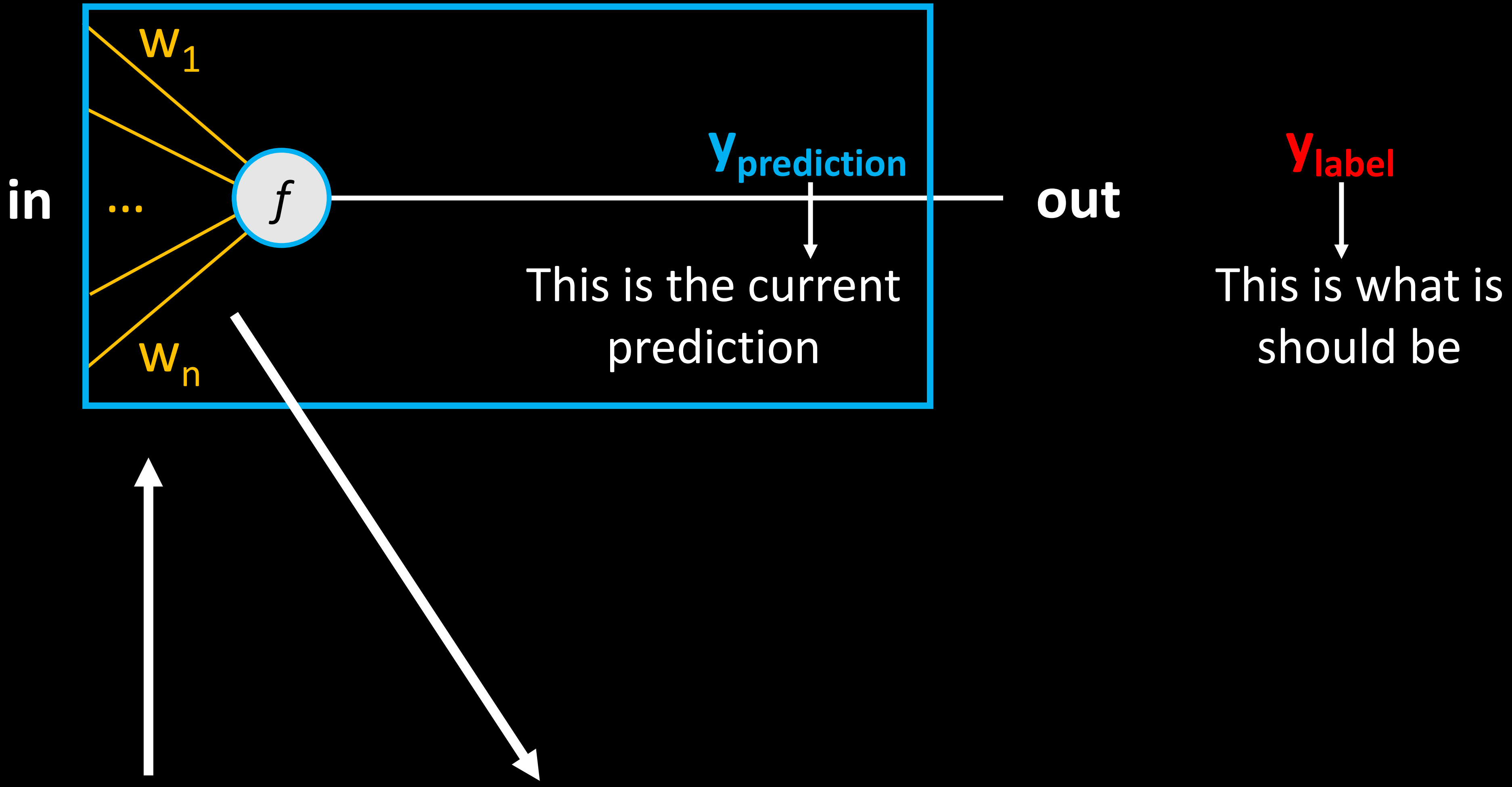
- Iterative algorithm with which we will eventually find a local minimum

- **J(w)** is the error function
- And changes to **w** are changes to the parameter

- We will end up with a *(locally)* **optimal value for the parameter**

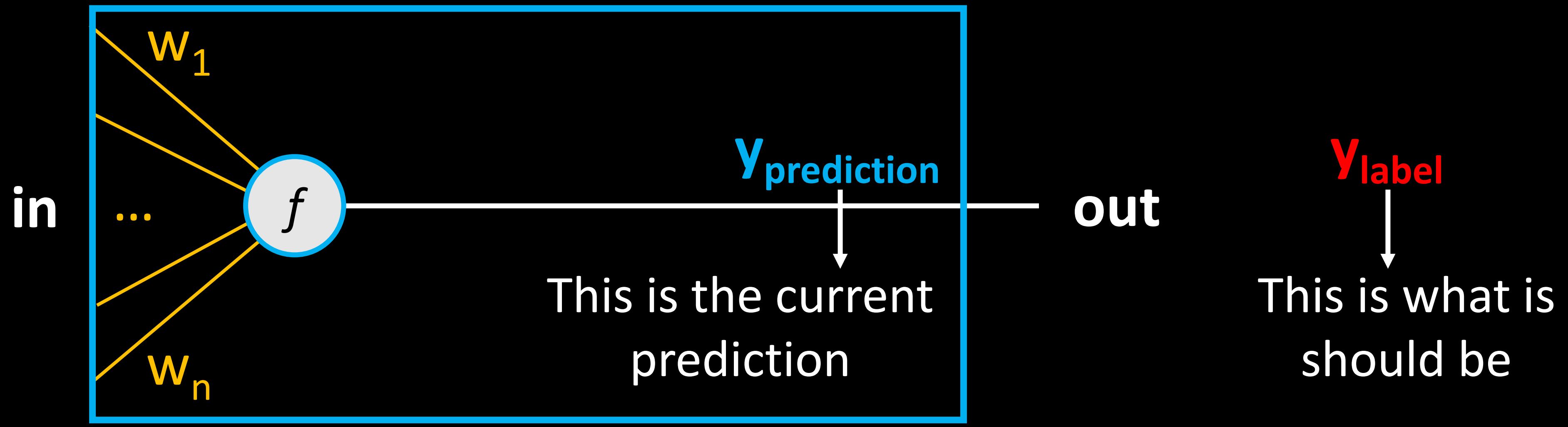
Next Position = **Current Position** + **direction** * step size

What about Neural Networks?



Next Weights = Current Weights + ???

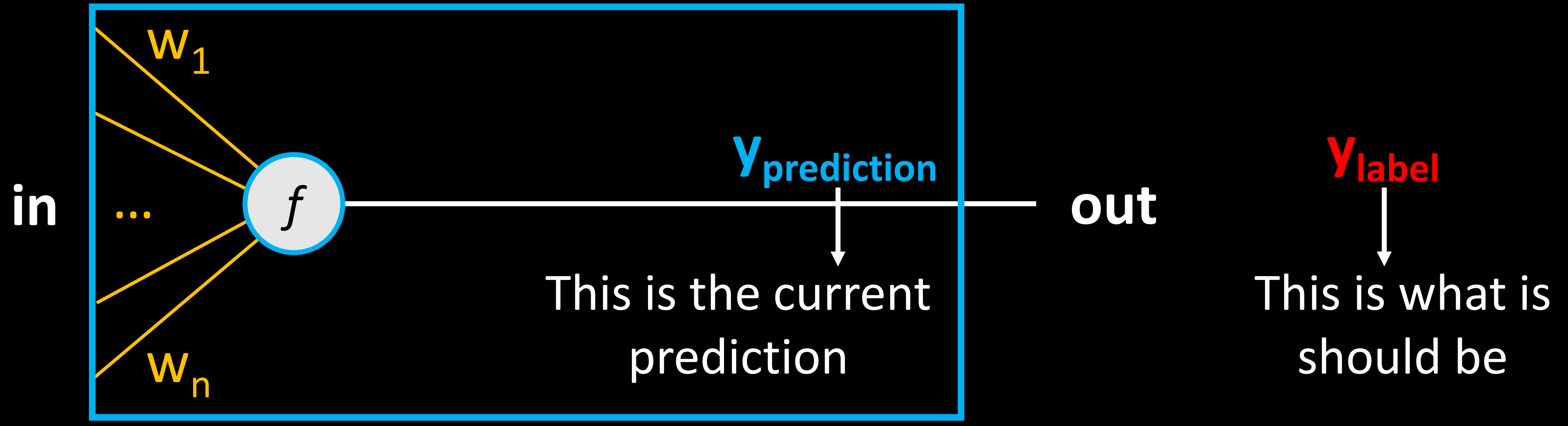
What about Neural Networks?



- We can see how far we are off: error = ($y_{prediction}$ - y_{label})
- All of these calculations are differentiable, which means that we can derive them and get a gradient which will give us a direction to minimize the error: (*simplified*)

Next Weights = **Current Weights** + **direction** * step size

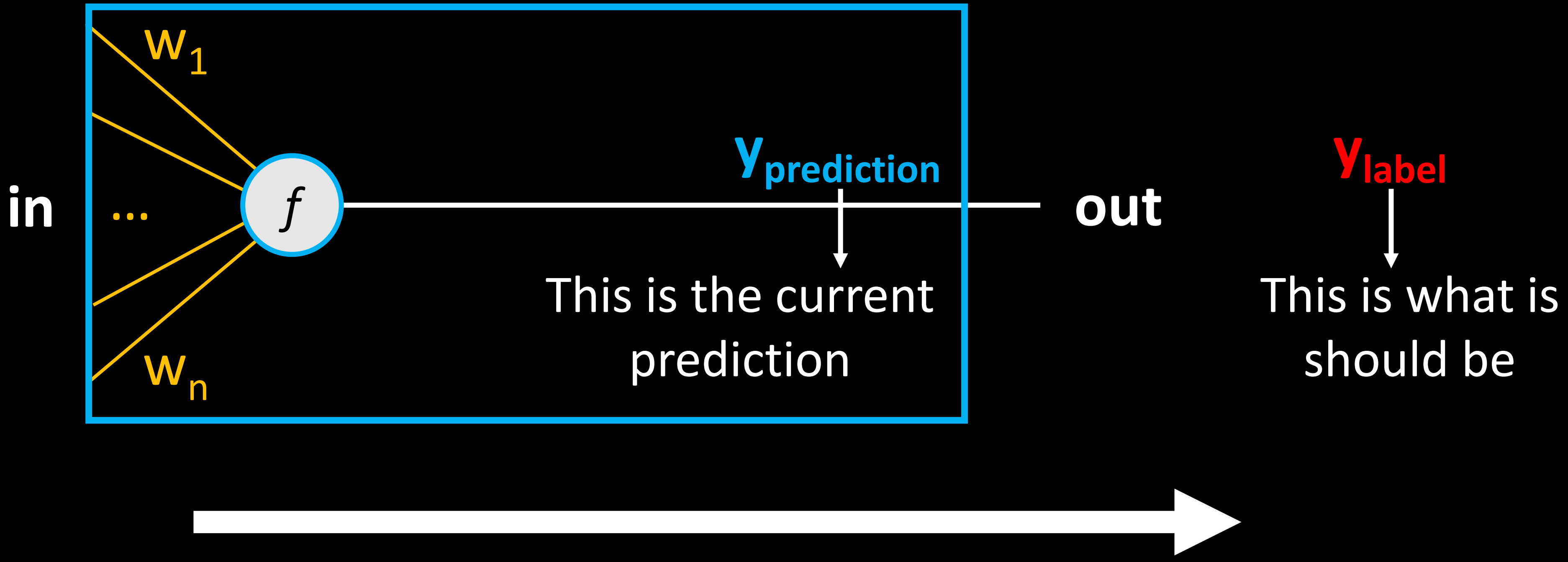
What about Neural Networks?



- This means we can iteratively try some values for **weights**, and get closer and closer to a solution which has a better error (predictions are similar to our labels)

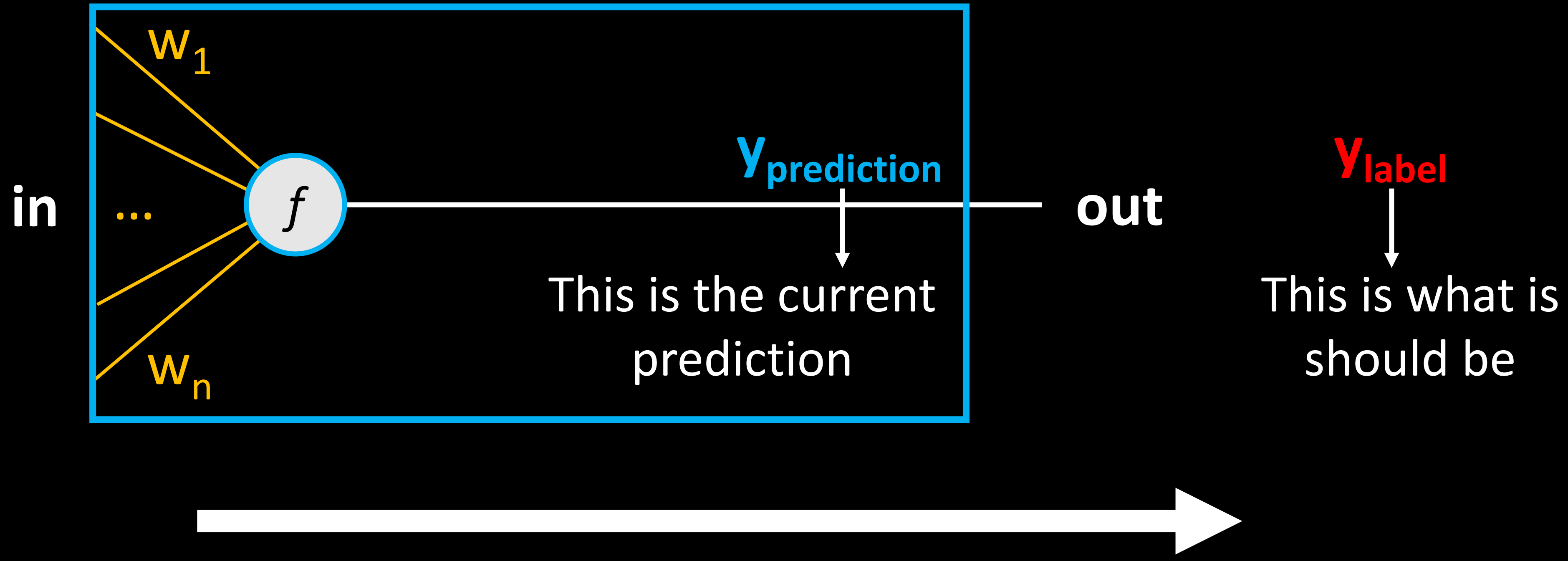
Next Weights = **Current Weights** + **direction** * step size

What about Neural Networks?



Calculating the predictions is called **forward pass**.

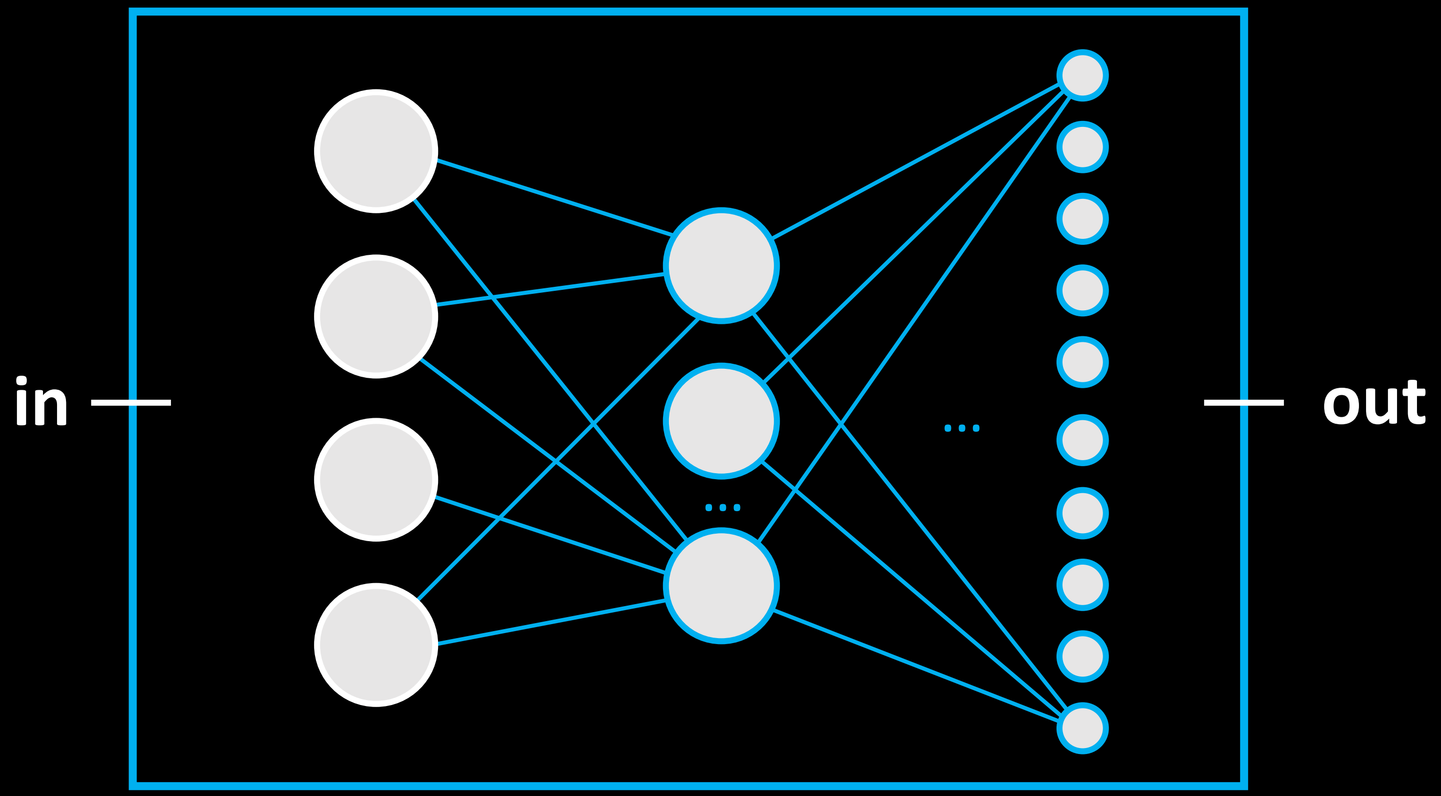
What about Neural Networks?



Calculating the predictions is called **forward pass**.

Calculating the update for the weights is called **backwards pass**.

Big picture



- With larger Neural Networks we repeat this process over and over during **training**. We call these iterations **epochs**.

End of the lecture …

- … In the practical section we will learn how to write some of these models in code!

Practicum

Fully connected neural networks with Keras

Motivation:

- Create the models we just discussed in code
- Show that while the theory was not trivial … the code is actually very short and simple to understand!

Practicum: Fully connected NNs

Continue with code on our Github:

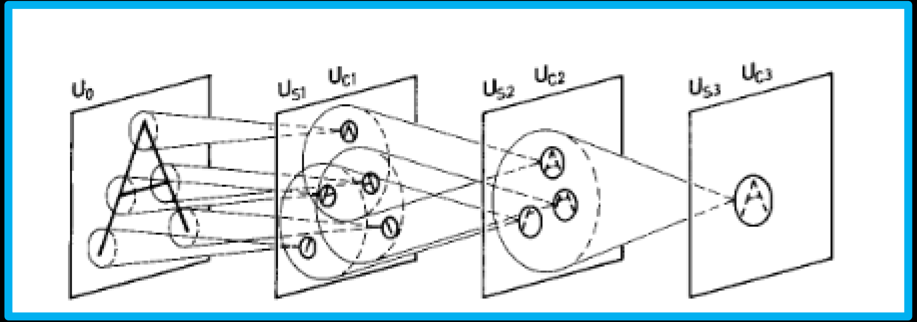
- github repo: github.com/previtus/cci_exploring_machine_intelligence
- notebook: week02_basic-building-blocks/ml02_fully_connected_nn.ipynb

Next class

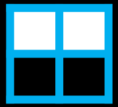
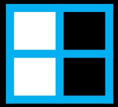
Convolutional neural networks

- Modelling artificial visual system
- ImageNet dataset and AlexNet model

- Practicum: Using trained models



Homework:

- **<u>Task:</u>** Manually find values for w_1, w_2, w_3, w_4, which would do the classification between  and  images. Horizontal/Vertical classifier!

Submission via weekly quiz:

- Answer the values for w_1, w_2, w_3, w_4 there.

The end