INTRODUCTION TO NEURAL NETS

Perceptron and Backpropagation

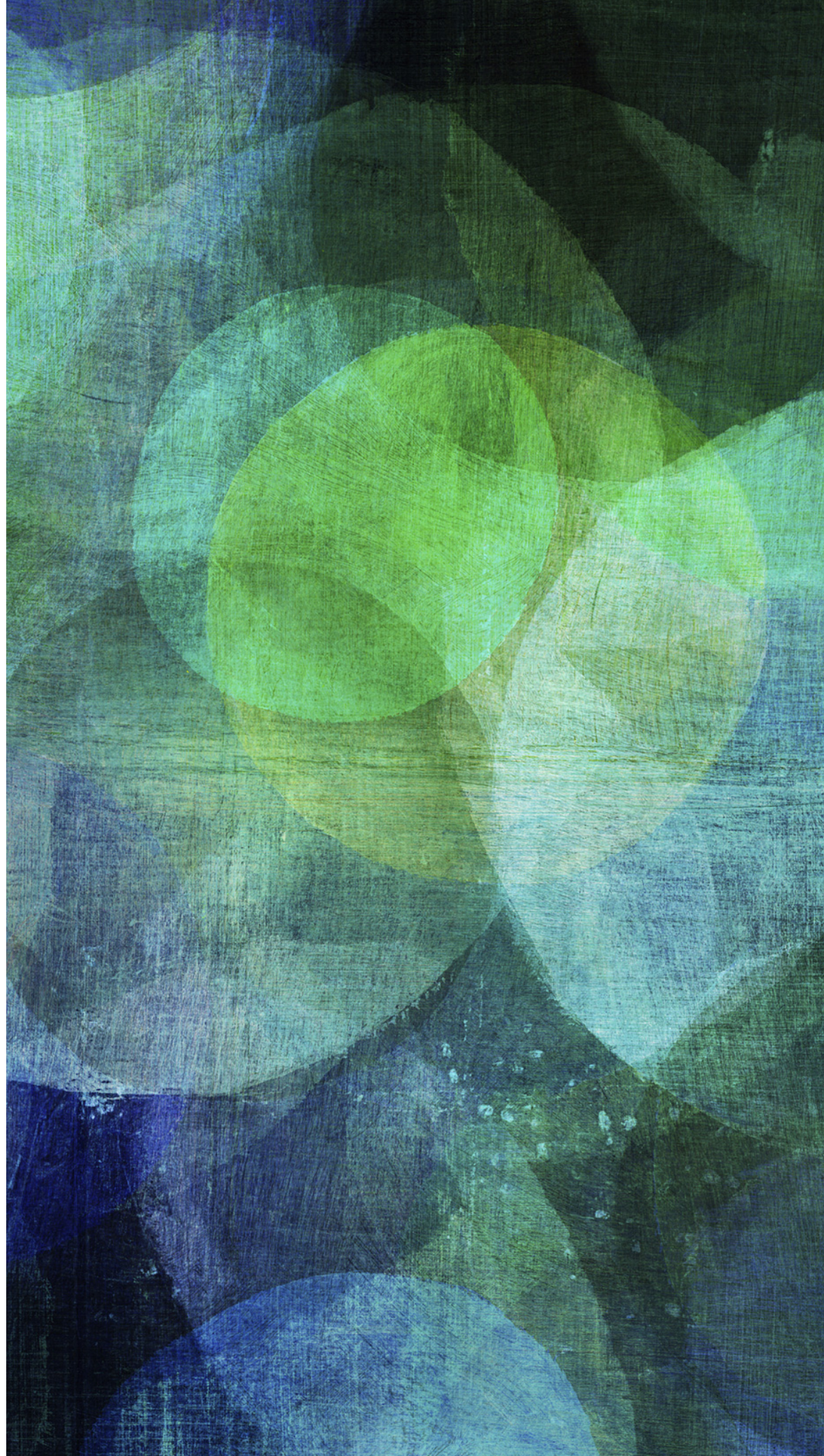
LAST TIME...

- We introduced Machine Learning
- We show an example of the simplest ML algorithm, namely the decision trees
- We discussed evaluation metrics

LEARNING OBJECTIVES

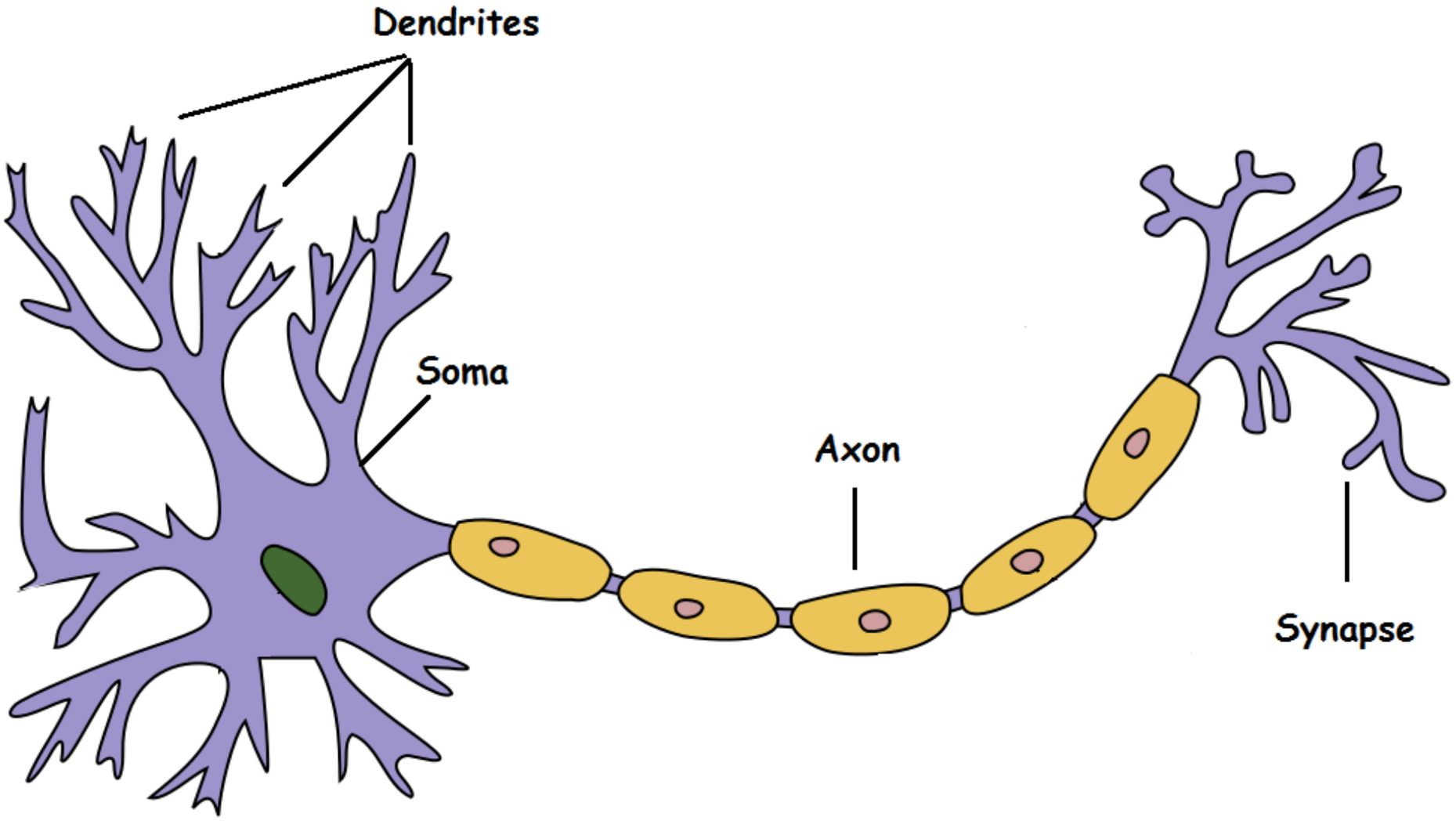
- To understand what neural networks are and how they work
- To learn the perceptron, the predecessor of neural networks
- To learn and understand how neural networks are trained with backpropagation
- *Why?* Neural Nets is the most powerful machine learning technique at the moment.

Intro to Neural Networks

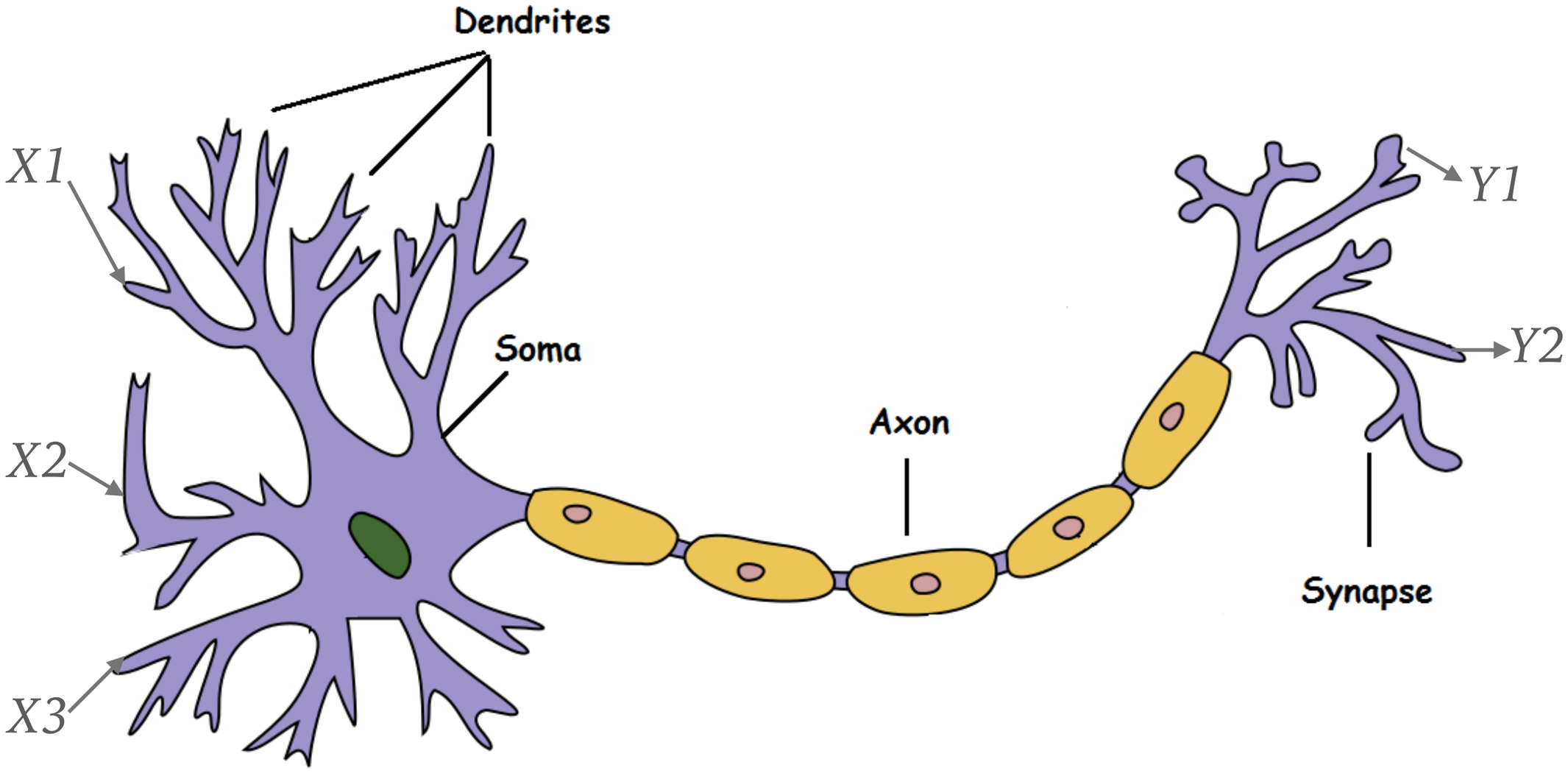


NEURAL NETWORKS ARE INSPIRED BY THE HUMAN BRAIN

THE BRAIN CONSISTS OF MANY NEURONS

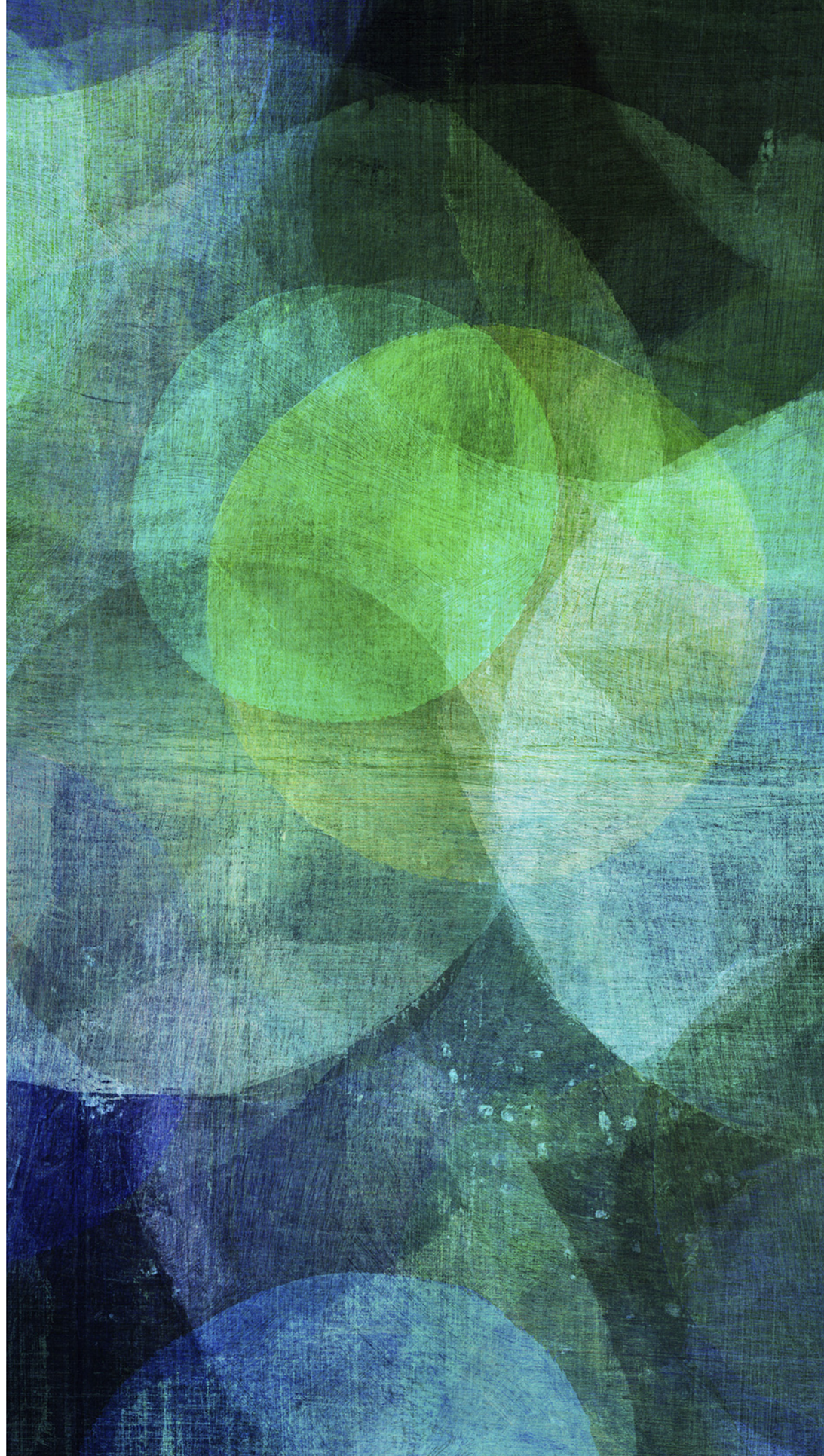


THE BRAIN CONSISTS OF MANY NEURONS



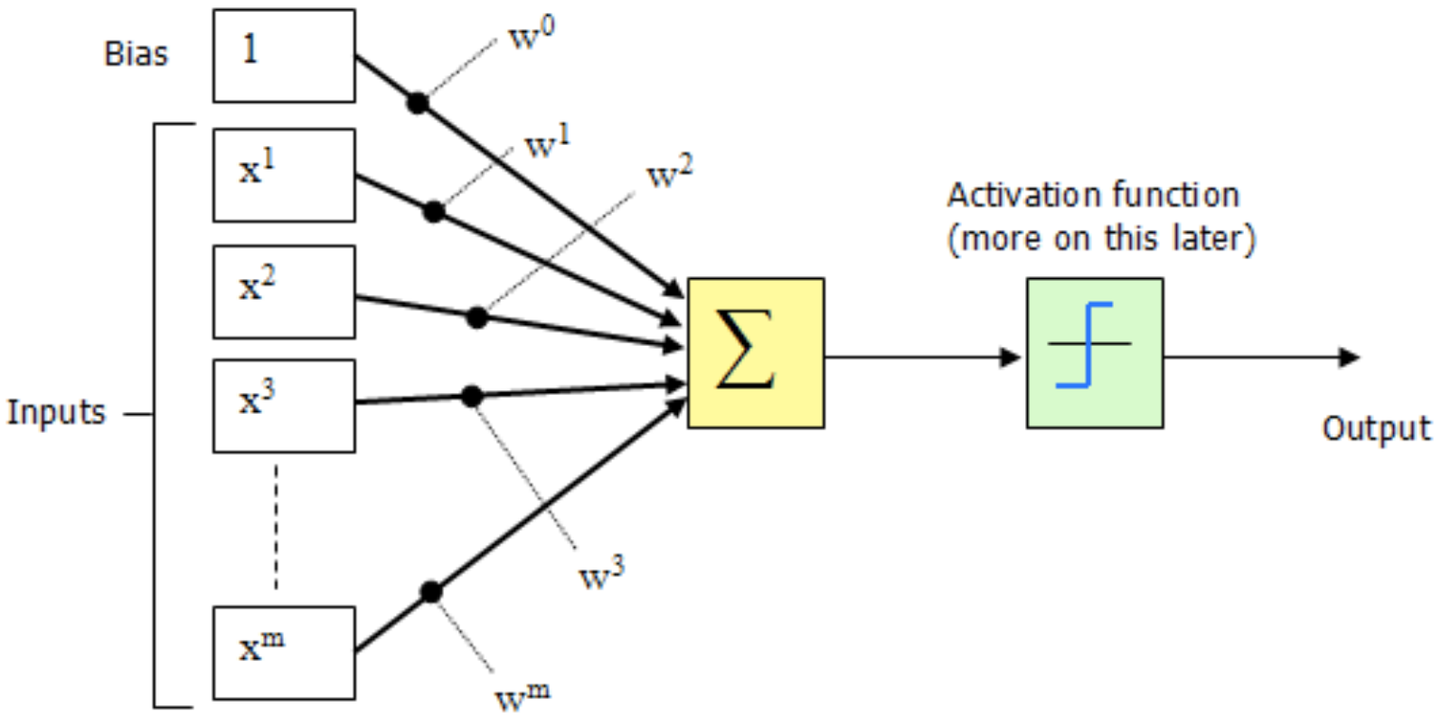
HISTORY OF NEURAL NETWORKS

The Perceptron

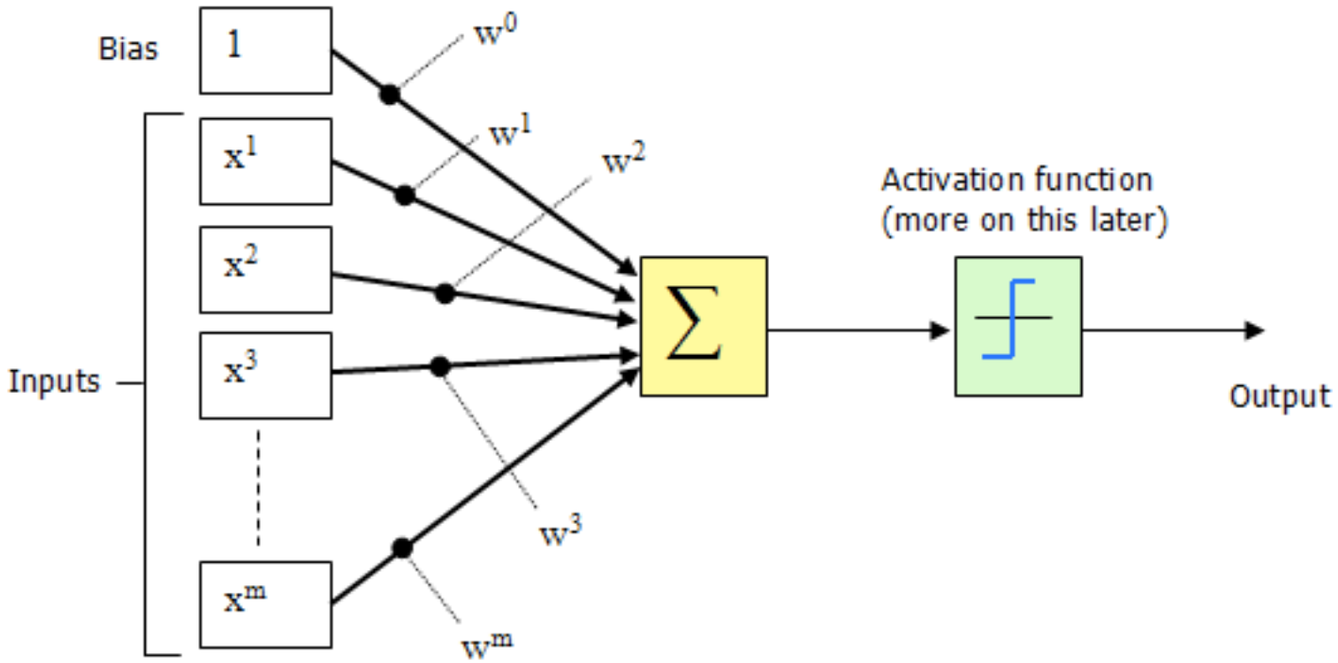


A PERCEPTRON: A (COMPUTATIONAL) NEURON

- Rosenblatt proposed the Perceptron for binary classifications
  - Takes some input x, does some maths and outputs y
  - Simplest Artificial Neural Net (ANN)



THE PERCEPTRON (HOW TO MAKE PREDICTIONS)



- One weight *wi* per feature *xi*
- 1st step: Multiply weights with respective features, *x*1*w1, x2*w2, etc.
- 2nd step: Add the weighted input and bias (w0 + x1*w1 + x2*w2 + …)
- 3rd step (activation function): If the result is larger than a threshold return 1, otherwise 0

ACTIVATION FUNCTION

- The **activation function** defines the output of given an input or set of inputs.
- For binary classification, the simplest activation function is the **sign** function.
- There are other activation functions, that we will discuss next time.

WEIGHTS

- The weights denote the relationship between an input **xi** and the output **y**.
- The greater the absolute value of the weight, the greater the impact on the output.

BIAS

- A bias unit is an "extra" neuron added to each pre-output layer that stores the value of 1.
- Bias units aren't connected to any previous layers and in this sense don't represent a true “activity”.
- The bias helps to control the behavior of the layer and contributes to better learning.

OVERSIMPLIFIED: IS IT A KITTY?

x_1 *has pointed ears* — w_1 *0.8*

x_2 *it's gray* — w_2 *0.01*

x_3 *has four legs* — w_3 *0.4*

x_4 *has windows* — w_4 *- 0.5*

$$\sum_{i=1}^{m} w_i x_i$$

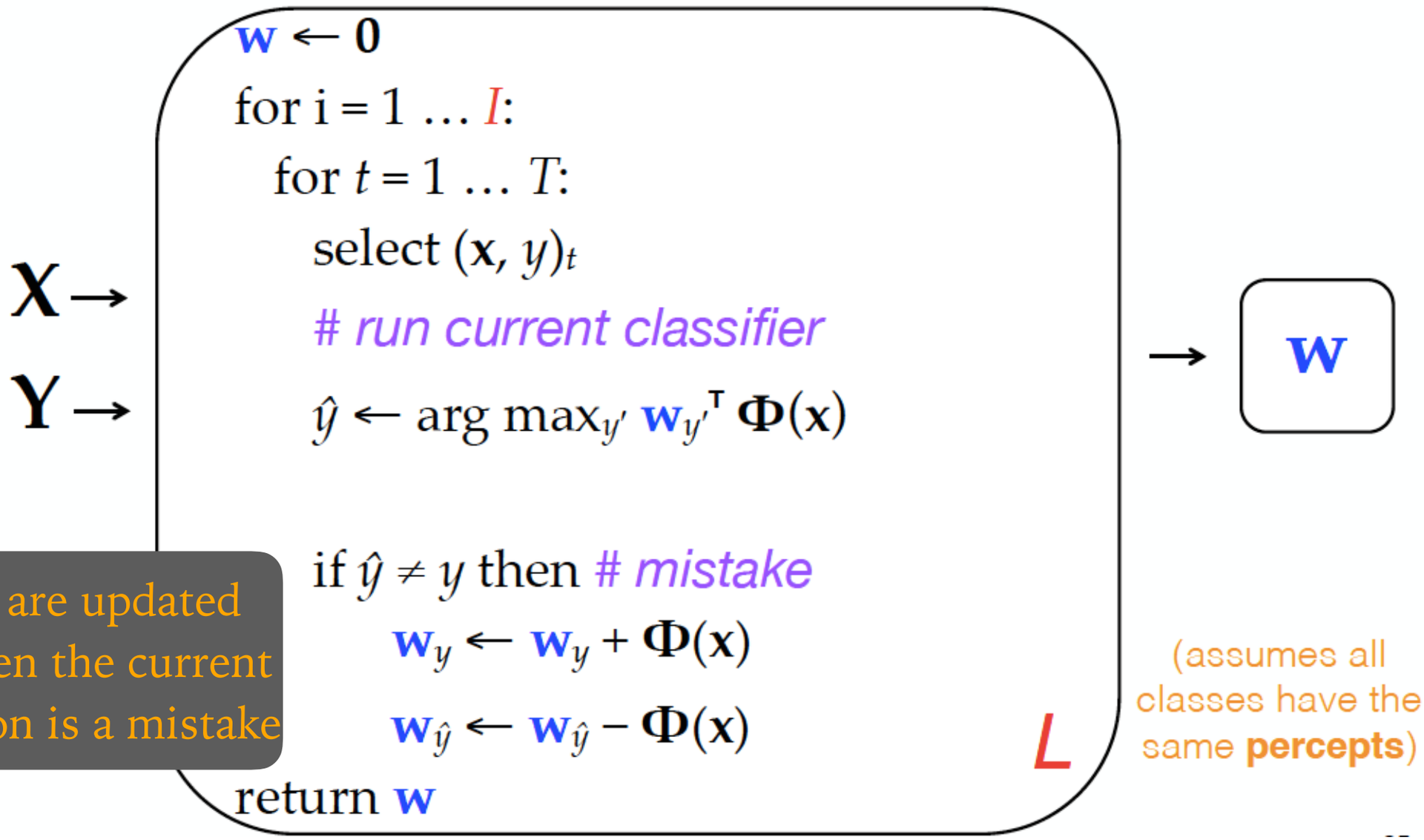
Σ — φ — *1*

$$f(x) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

CHOOSING FEATURES

- **Domain expertise helps in choosing** which kinds of features to include in the model (visual features such as body parts, building parts, words, sub-word units, metadata, …)
- The decision about what features to include in a model is called **feature engineering**.
  - (There are some algorithmic techniques, such as feature selection, that can assist in this process.)
  - More features = more flexibility, but also more expensive to train, more opportunity for **overfitting (more on this later)**.

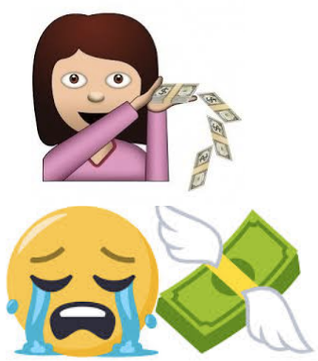
PERCEPTRON LEARNER



EXAMPLE: PREDICT WHETHER TO GIVE A LOAN

- Task: Predict whether to give someone a loan based on some characteristics such as age, work pattern and income.
- This is a binary classification problem with two potential outputs

1. Approve loan → *1*
2. Not approve → *0*



EXAMPLE: PREDICT WHETHER TO GIVE A LOAN

Age > 25	Work Pattern (Full-time = 1, 0 otherwise)	Income > 20,000	Loan?
1	1	1	1
0	1	1	1
1	0	1	1
1	0	0	0
0	0	1	1

- Input X:
  - X1 = Age, X2 = Work Pattern, X3 = Income
- Output Y:
  - Loan?

1ST ITERATION

	1st instance	Initial Weights
X1 (Age)	1	0
X2 (Work Pattern)	1	0
X3 (Income)	1	0

- The weights are initially 0 (no training has taken place yet)
- Output = 1*0 + 1*0 + 1*0 = 0 
- However, we know that the first example should have resulted in 1, so we need to update the weights by adding 1. 

WEIGHTS UPDATE

	1st instance	Weights Update
X1 (Age)	1	1
X2 (Work Pattern)	1	1
X3 (Income)	1	1

- All weights have been updated.
- Now if we multiply each input with the new weights, the output is 3, i.e. >=0, therefore the prediction will be 1.

2ND ITERATION

	1st instance	Weights Update	2nd instance
X1 (Age)	1	1	0
X2 (Work Pattern)	1	1	1
X3 (Income)	1	1	1

- The weighted sum will be 1x0 + 1x1 + 1x1 = 2, which is greater than 0, so the prediction is correct.
- The weights are therefore **not** updated! 

3RD ITERATION

	1st instance	Weights Update	2nd instance	3rd instance
X1 (Age)	1	1	0	1
X2 (Work Pattern)	1	1	1	0
X3 (Income)	1	1	1	1

- Weighted sum: 1x0 + 1x0 +1x1 = 1 >0.
- No update at this instance, because the prediction is correct.
- If for a new instance the prediction is wrong and the output is negative, we deduct 1 instead of adding one.
- The process continues until all instances have been considered for a number of times.
- Note that only the weights that correspond to a variable which is 1 are updated. If the variable is 0, the corresponding weight does not get updated.

STEPS

1. create features
2. initialise weights
3. update weights
4. predict

MODELLING

- In real-world examples, the inputs might include more information than we need and can be binary or real-valued.
- Formally, we denote with $\boldsymbol{\varphi(x)}$ a function that extracts a **vector** of features for an input x.

- Each feature receives a real-valued weight parameter w and the **weighted sum** is denoted by the following:

$$\mathbf{w}_{y'}{}^{\mathsf{T}}\boldsymbol{\phi}(\mathbf{x}) = \sum_j w_{y',j} \cdot \phi_j(\mathbf{x})$$

- For binary classification, this is equivalent to: $\text{sign}(w_T\varphi(x))$- $+1$ or -1

- In literature, weights are sometimes denoted with the greek letter theta θ

PERCEPTRON LEARNER

- The perceptron **adjusts weights** up or down until it classifies the training data correctly.
- Mistakes lead to weight updates.
- It uses only one hyperparameter: I, the number of iterations (passes through the training data).

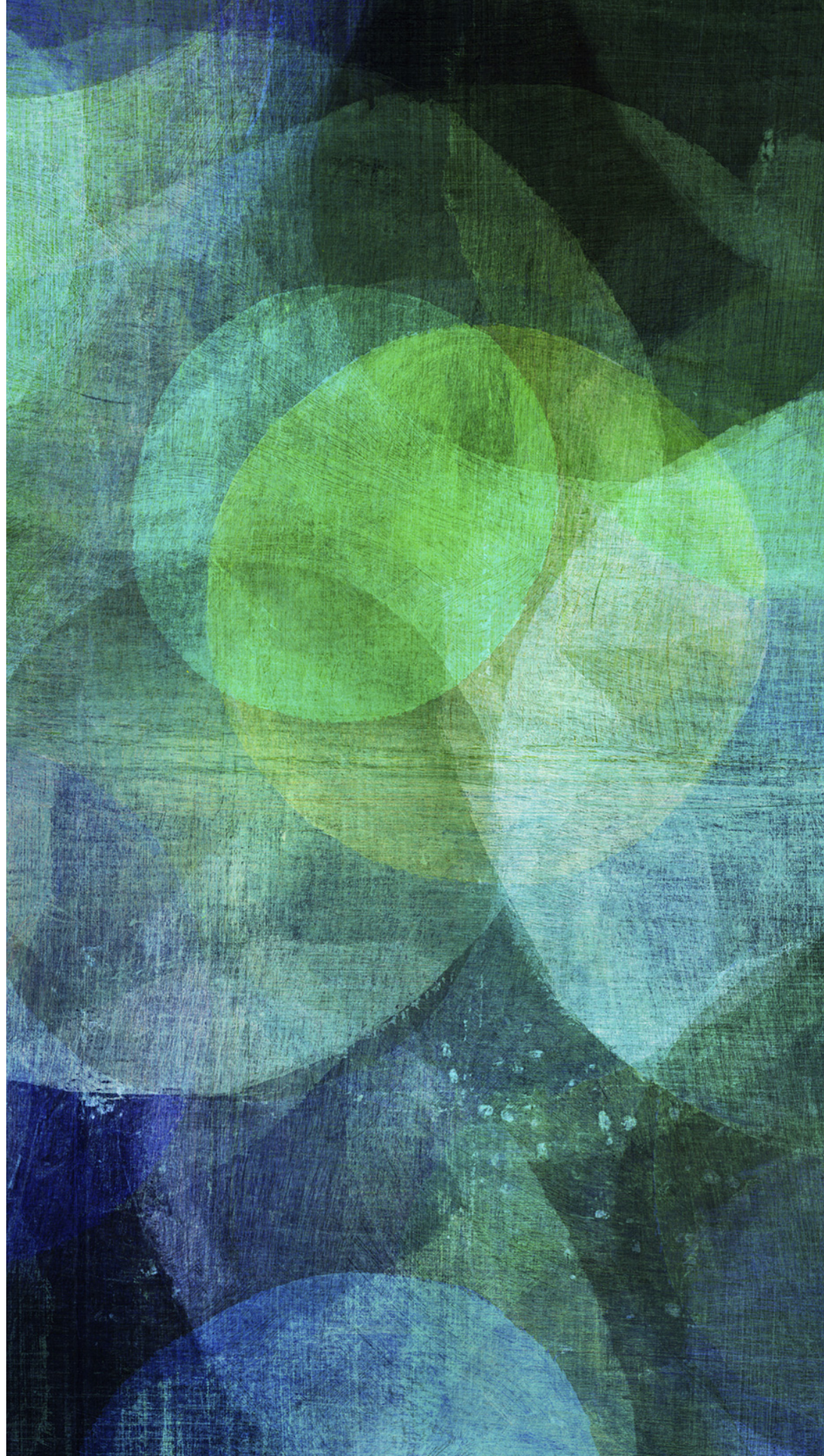
PERCEPTRON LEARNER

- Like any learning algorithm, the perceptron risks overfitting (memorising rather than learning) the training data.

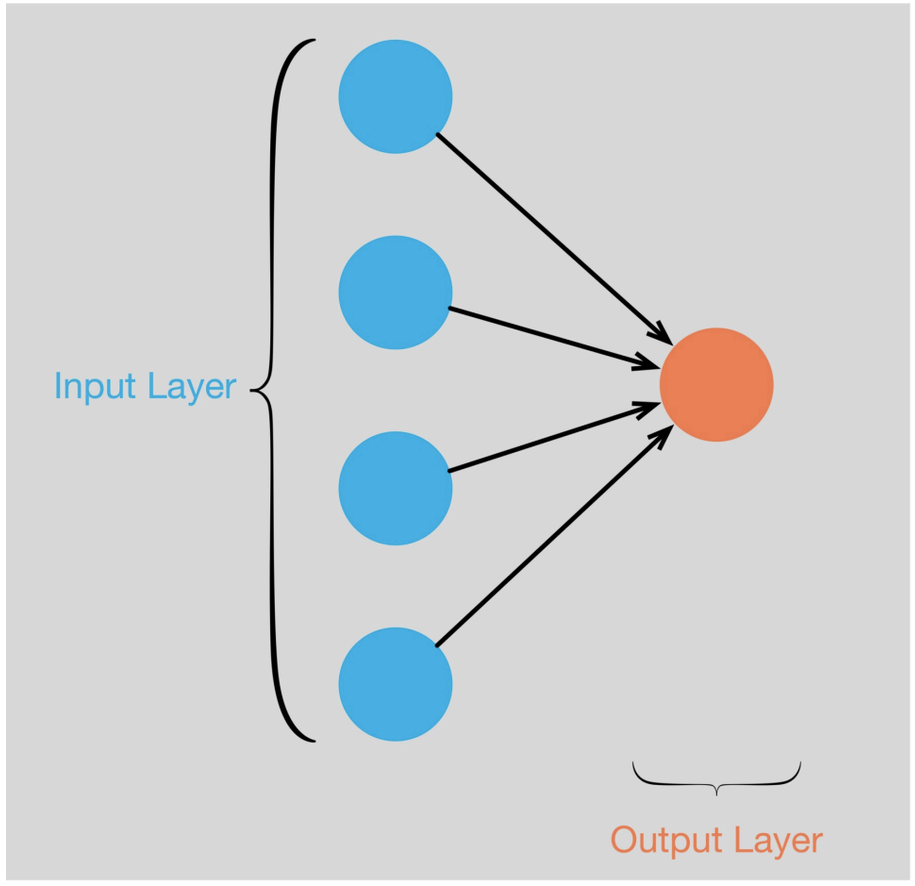
Two main techniques to improve generalization:

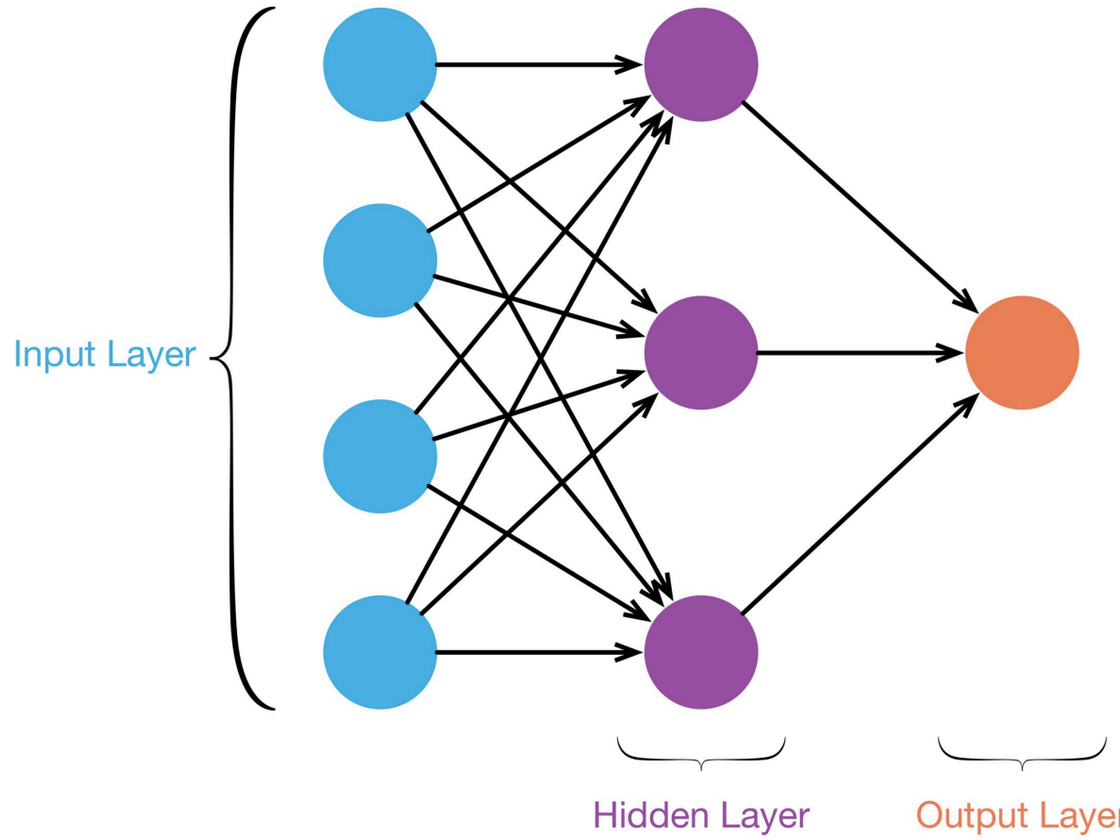
- **Averaging**: Keep a copy of each weight vector as it changes, then average all of them to produce the final weight vector.
- **Early stopping**: Tune I by checking held-out accuracy on dev data (or cross-val on train data) after each iteration. If accuracy has ceased to improve, stop training and use the model from iteration $I - 1$.

From Perceptron to Neural Nets



PERCEPTRON VS MULTI-LAYER NEURAL NETWORK

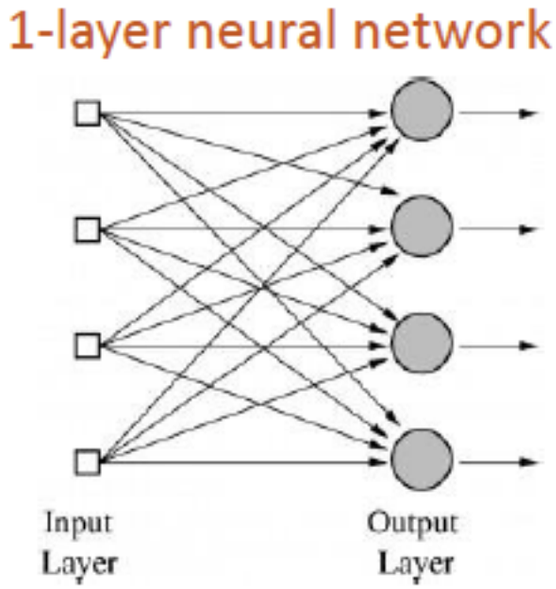


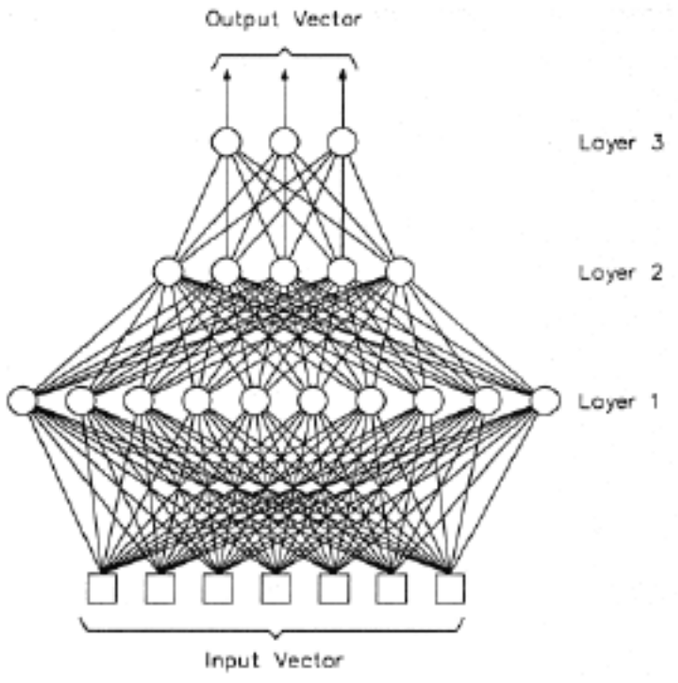


The main difference between the neurons of a Perceptron and the neurons of a multi-layer neural network is that the neurons at the hidden layer also use ***activation*** *functions.*

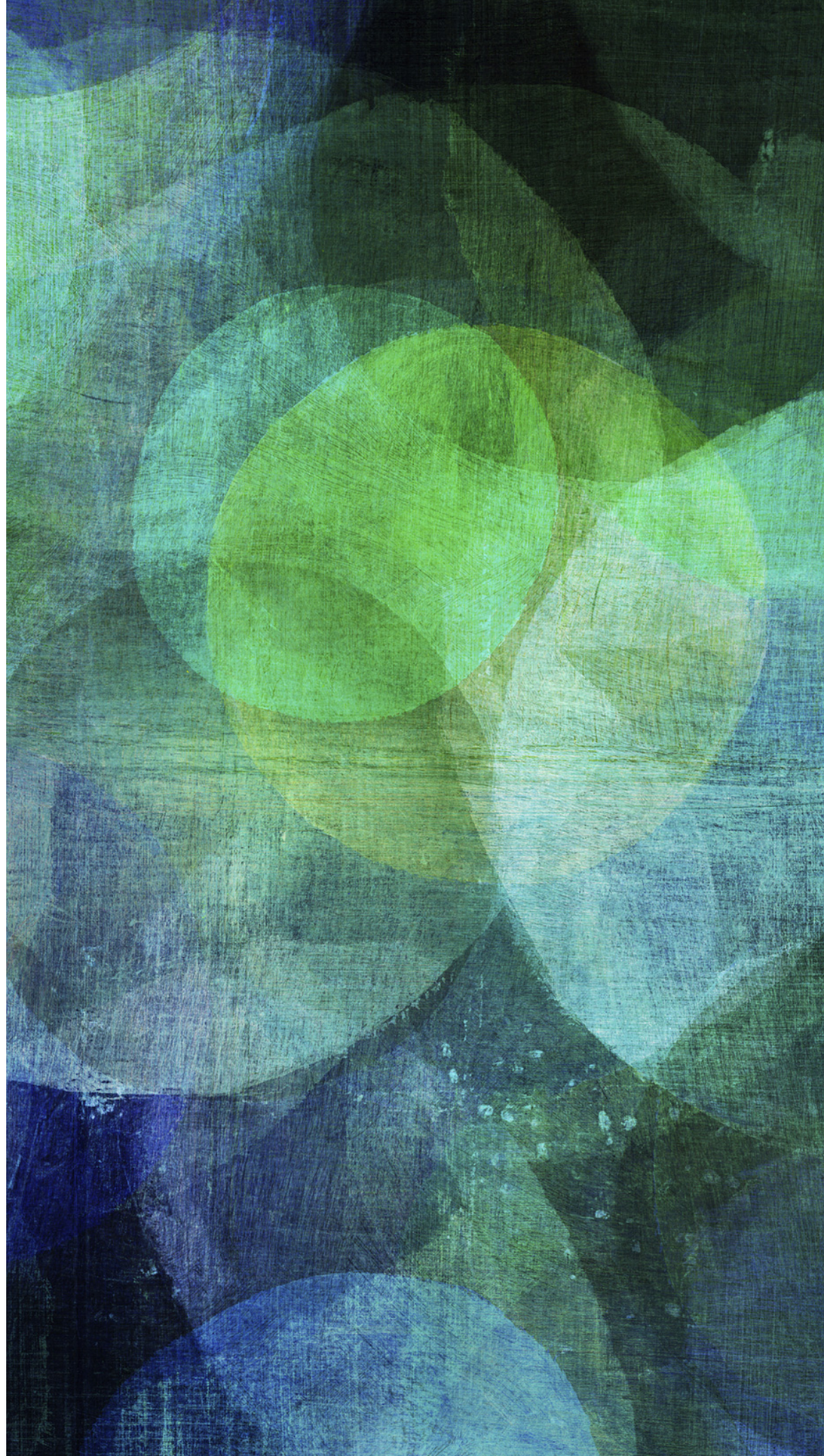
WHY MULTIPLE LAYERS?

- 1 perceptron == 1 decision
- What about multiple decisions?
  - E.g. beyond binary classification
- Stack as many outputs as the possible outcomes into a layer, i.e. a Neural network
- Multiple layers help learn non-linear relationships in the data
- Use one layer as input to the next layer
- Multi-layer perceptron (MLP)



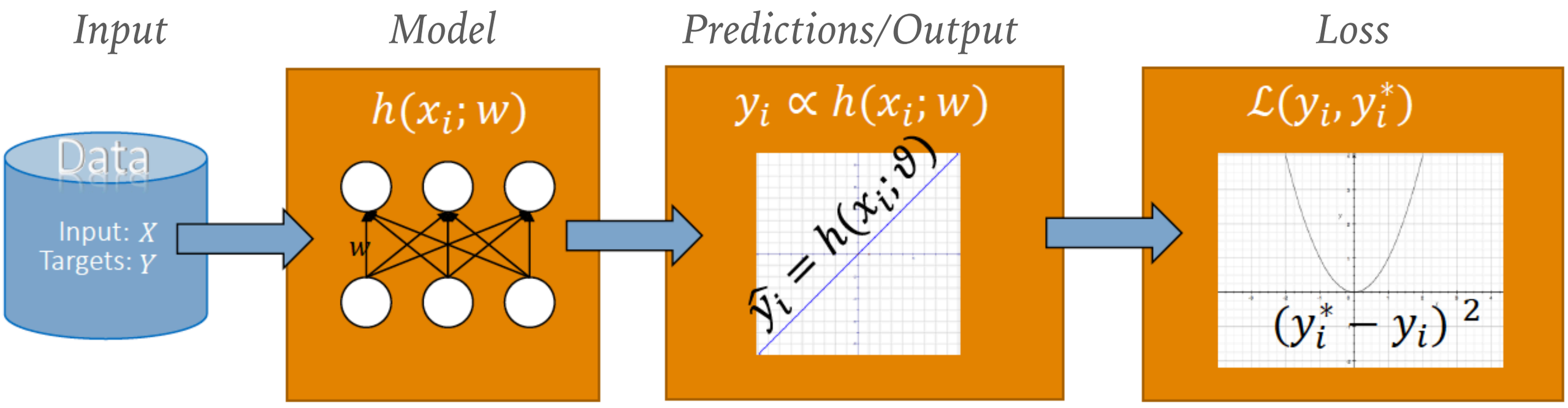


Backpropagation



FORWARD COMPUTATION

- Annotated data
- Define model (i.e. features) and initialise (more on this next time)
- Make predictions based on current model (i.e. forward propagation")
- Evaluate predictions



LOSS

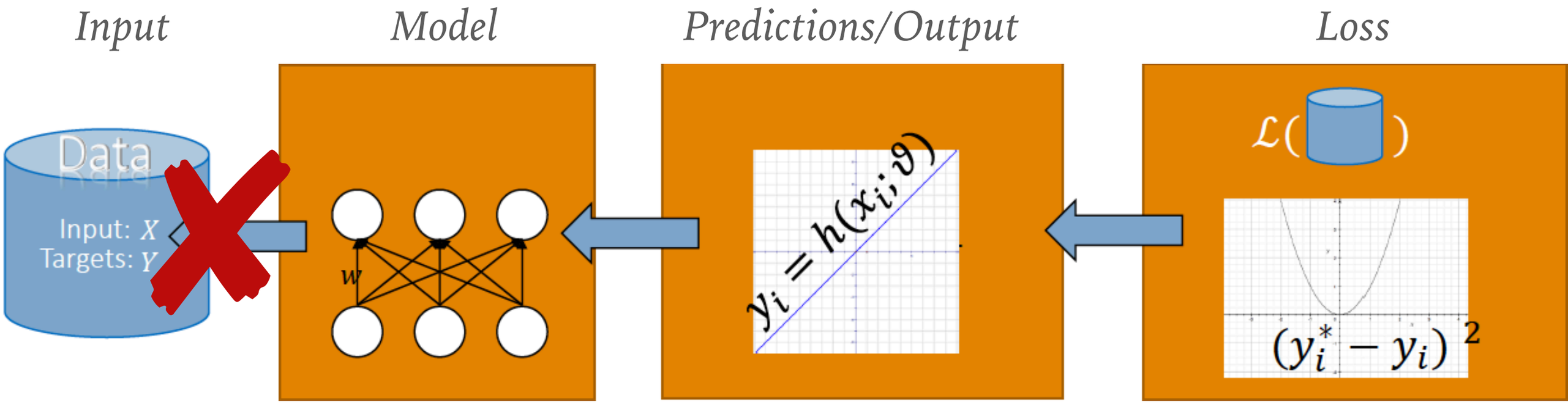
- Previously, we checked whether the prediction is accurate, if not, we updated the weights.
- In reality, neural networks use a loss function to evaluate how good the predictions are during training.
- One of the most popular loss functions (also called cost function) is the Mean Square Error:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

- In other words, the loss or cost function estimates how "wrong" the model is when making predictions.

BACKPROPAGATION

- Make predictions based on current model (i.e. **backpropagation**")
- Evaluate predictions
- Update weights to match the outputs (if wrongly predicted)

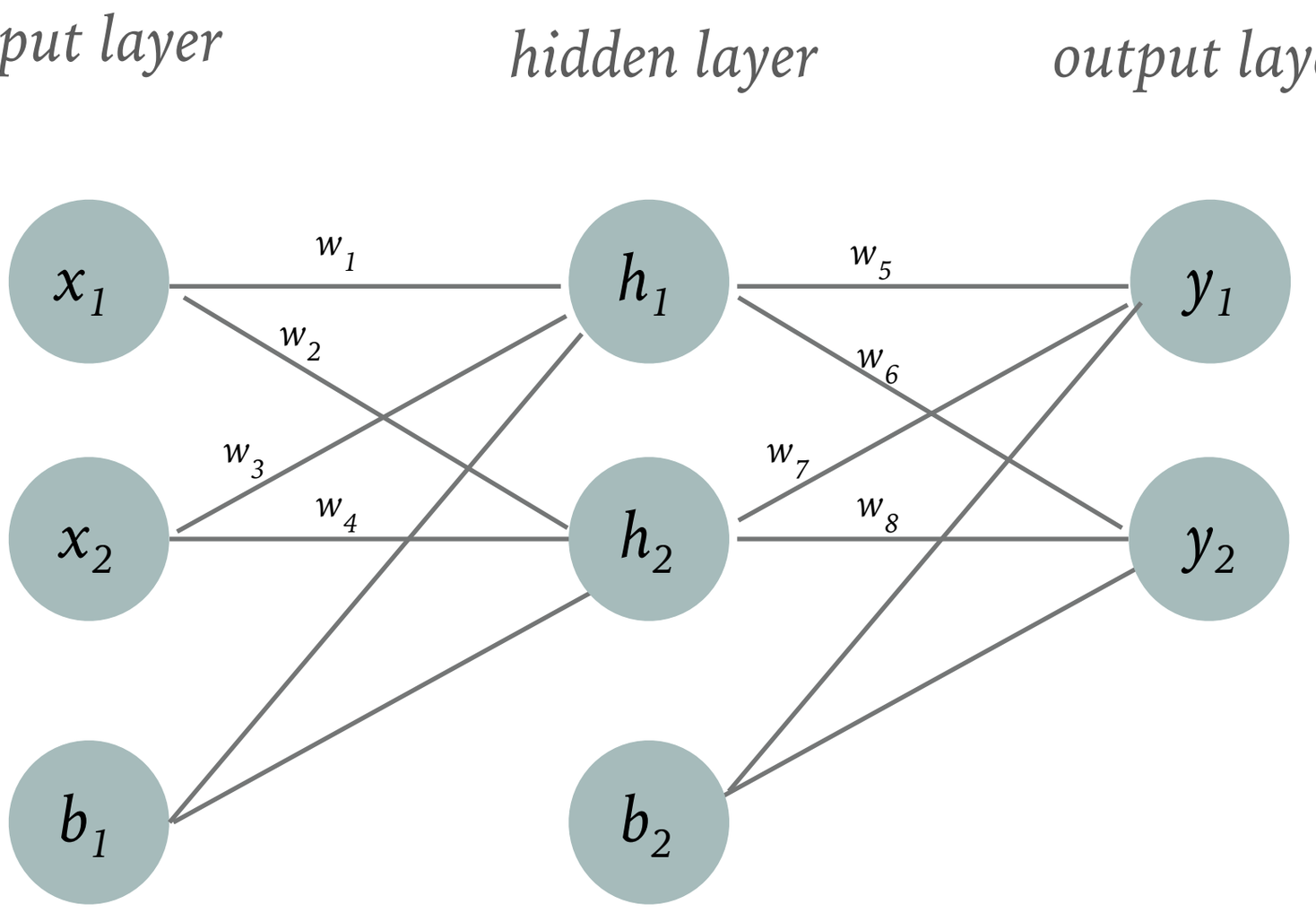


BACKPROPAGATION

- So far, we updated the weights adding/substracting 1. **In reality, it is more sophisticated.**
- Firstly, we initialise the weights (more on this later).
- At every time step we calculate the error (or cost), e.g. Mean Square Error:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

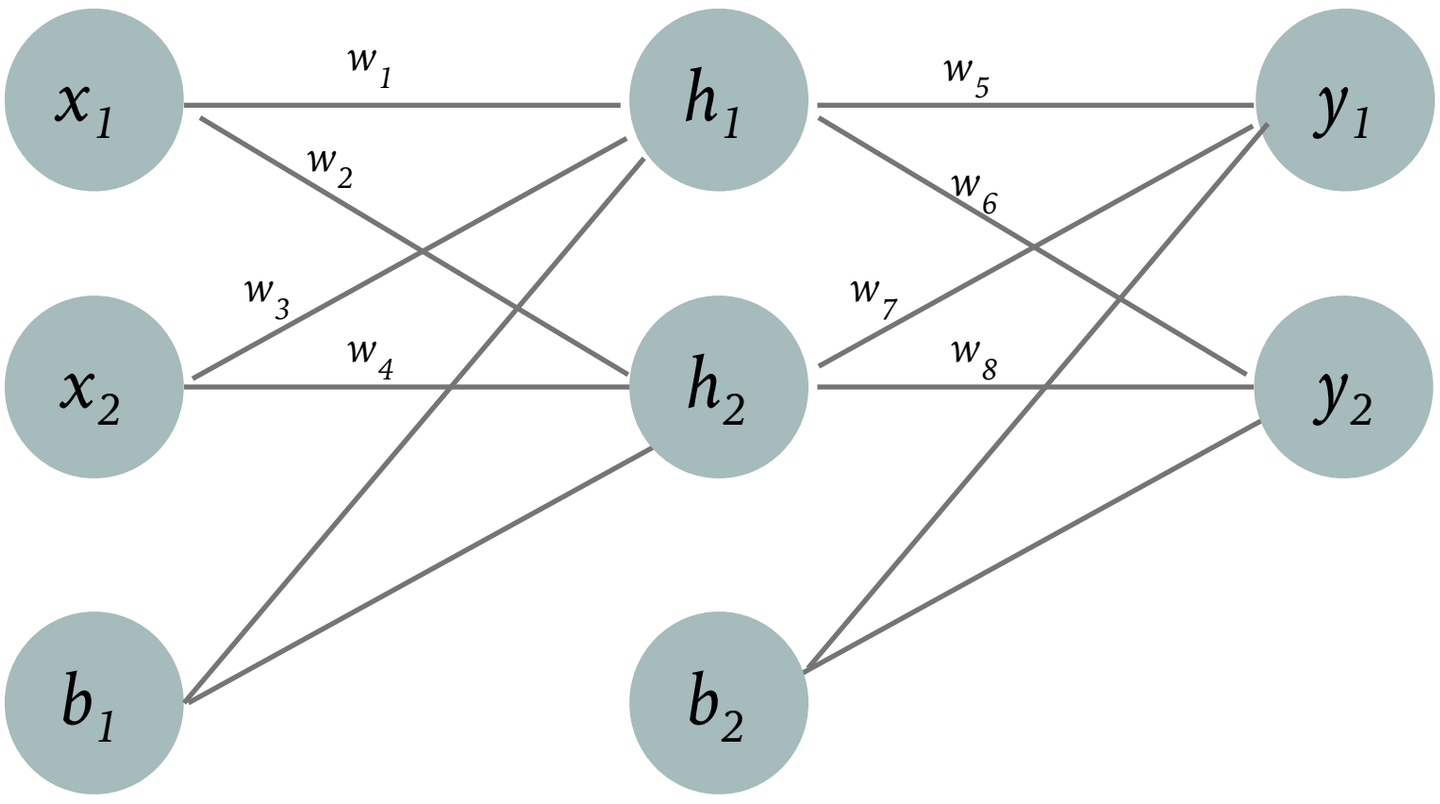
BACKPROPAGATION



➤ Consider the above neural net with all the weights and biases.

BACKPROPAGATION

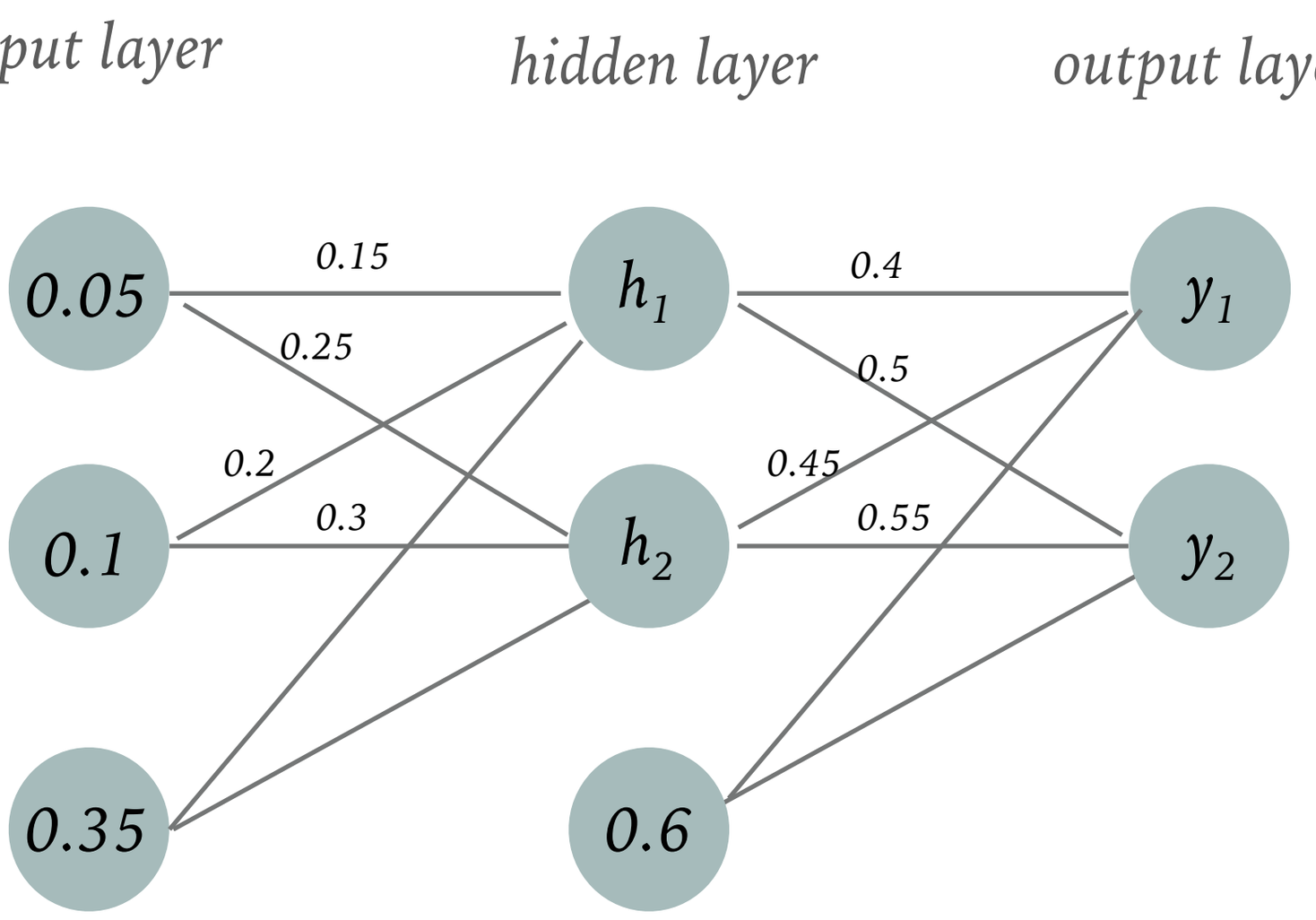
input layer *hidden layer* *output layer*



-How would you calculate h_1?

$$h_1 = x_1 * w_1 + x_2 * w_3 + b_1$$

EXAMPLE



target values

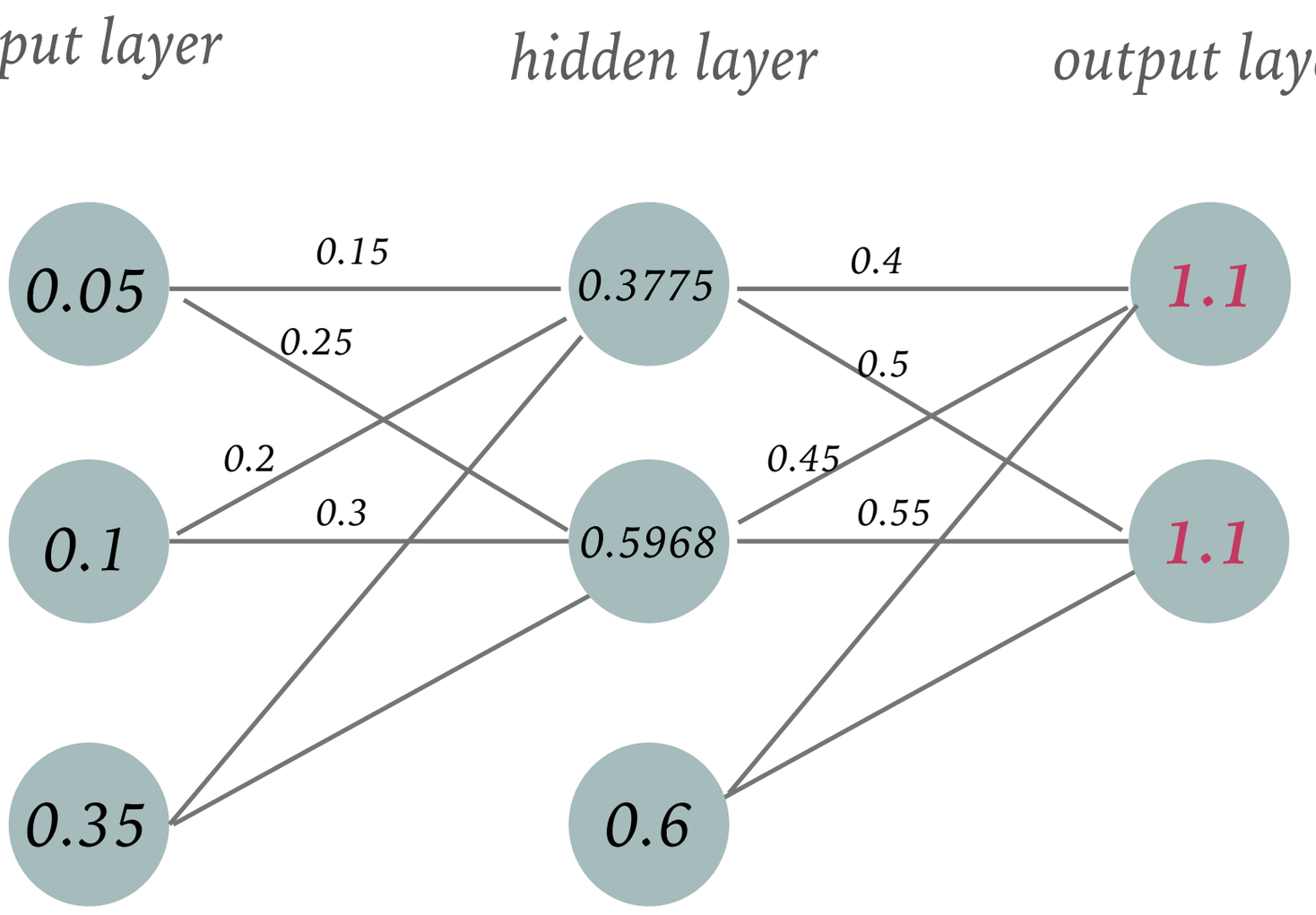
$y_1=0.01$

$y_2=0.99$

-What would be the value of y_1?

-What would be the value of y_2?

EXAMPLE: FORWARD PASS



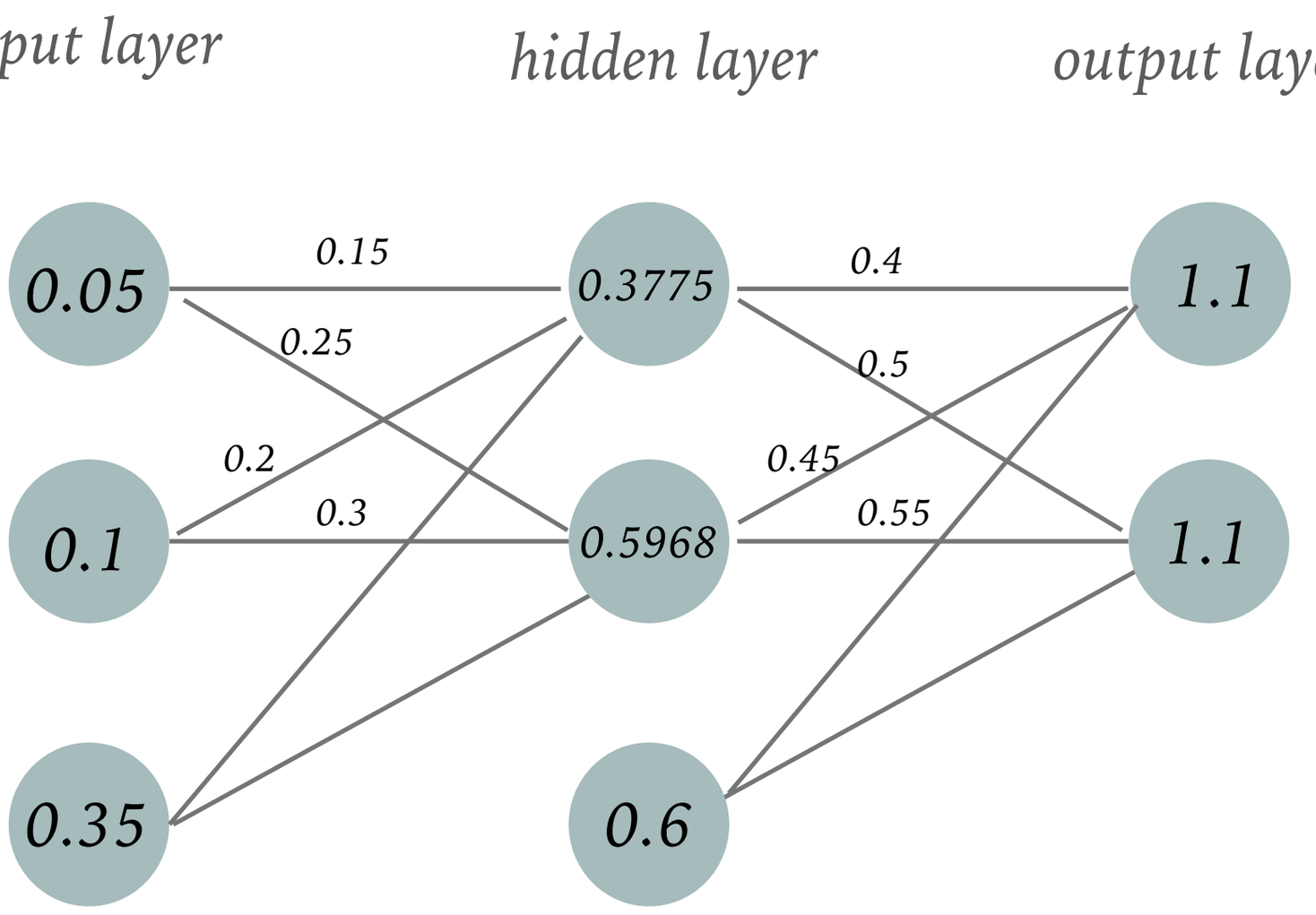
target values

$y_1 = 0.01$

$y_2 = 0.99$

-What would be the total error? Hint: $\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$

EXAMPLE: CALCULATE TOTAL ERROR



target values

$y_1 = 0.01$

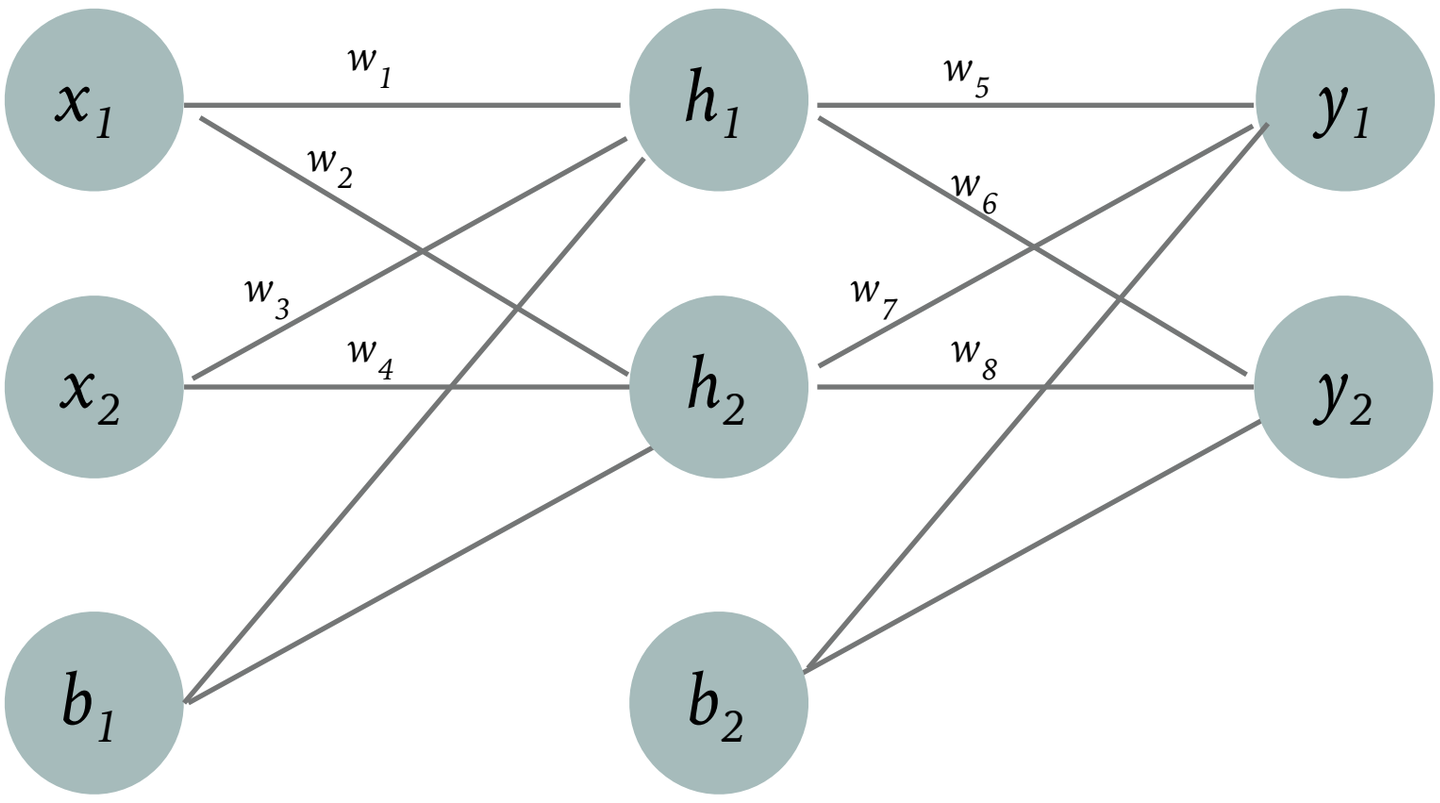
$y_2 = 0.99$

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

total error: 1/2 (0.01 - 1.1)^2 +1/2 (0.99 - 1.1)^2

BACKPROPAGATION

input layer *hidden layer* *output layer*

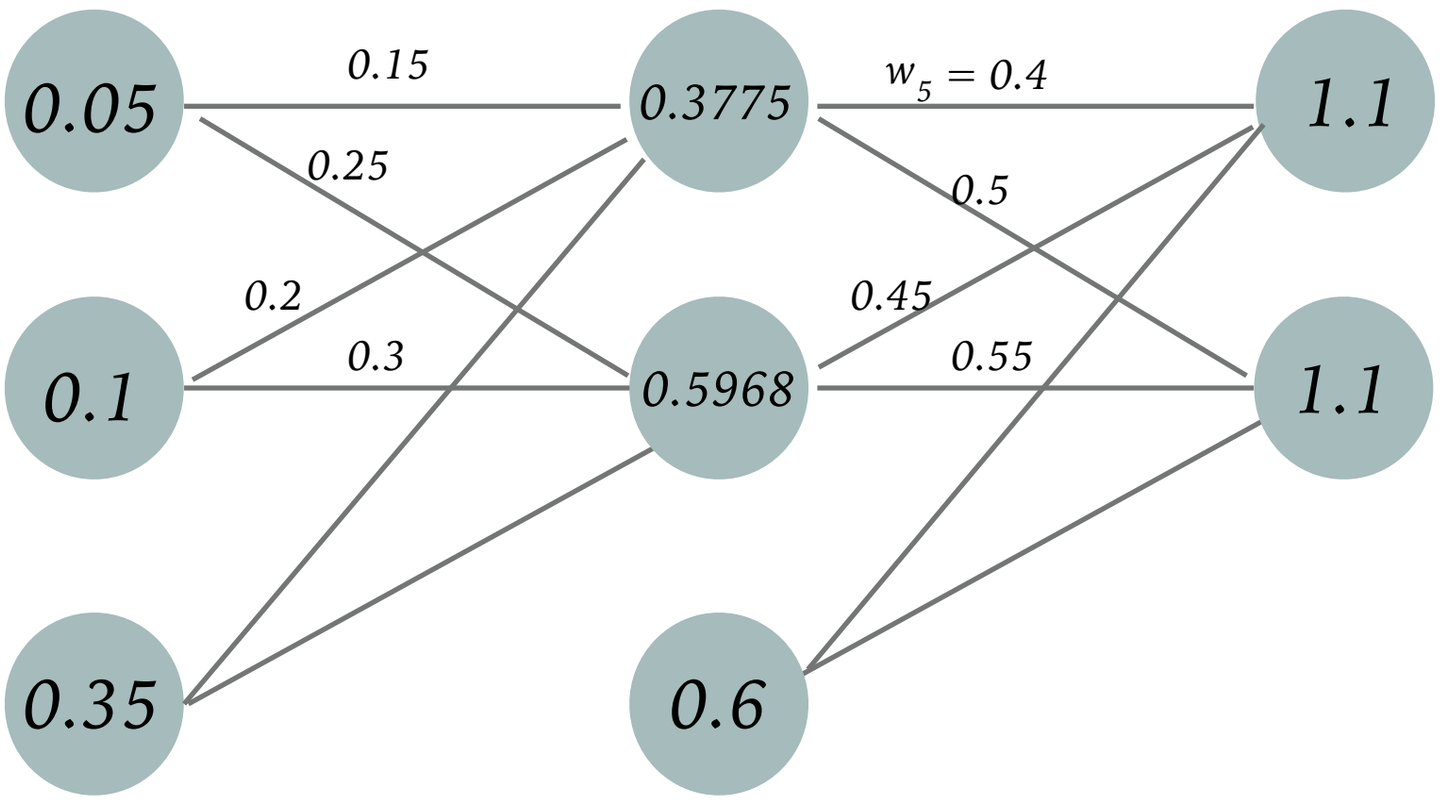


- Imagine we wanted to tweak *w5* . How would loss *L* change if we changed *w5* ?
- That's a question the partial derivative dE_{total} / dw_5 can answer.
- *The maths part are out of the scope of this module. We only briefly discuss it here so you understand better how neural networks are trained.*

DERIVATIVES

- Essentially, derivatives help us understand how a change in one variable, affects the other.
- e.g. the speed that the car travels dictates the distance covered.
- Derivatives: https://www.youtube.com/watch?v=N2PpRnFqnqY
- Partial Derivatives: https://www.youtube.com/watch?v=AXqhWeUEtQU&t=106s

EXAMPLE: CALCULATE ERROR AT WEIGHTS



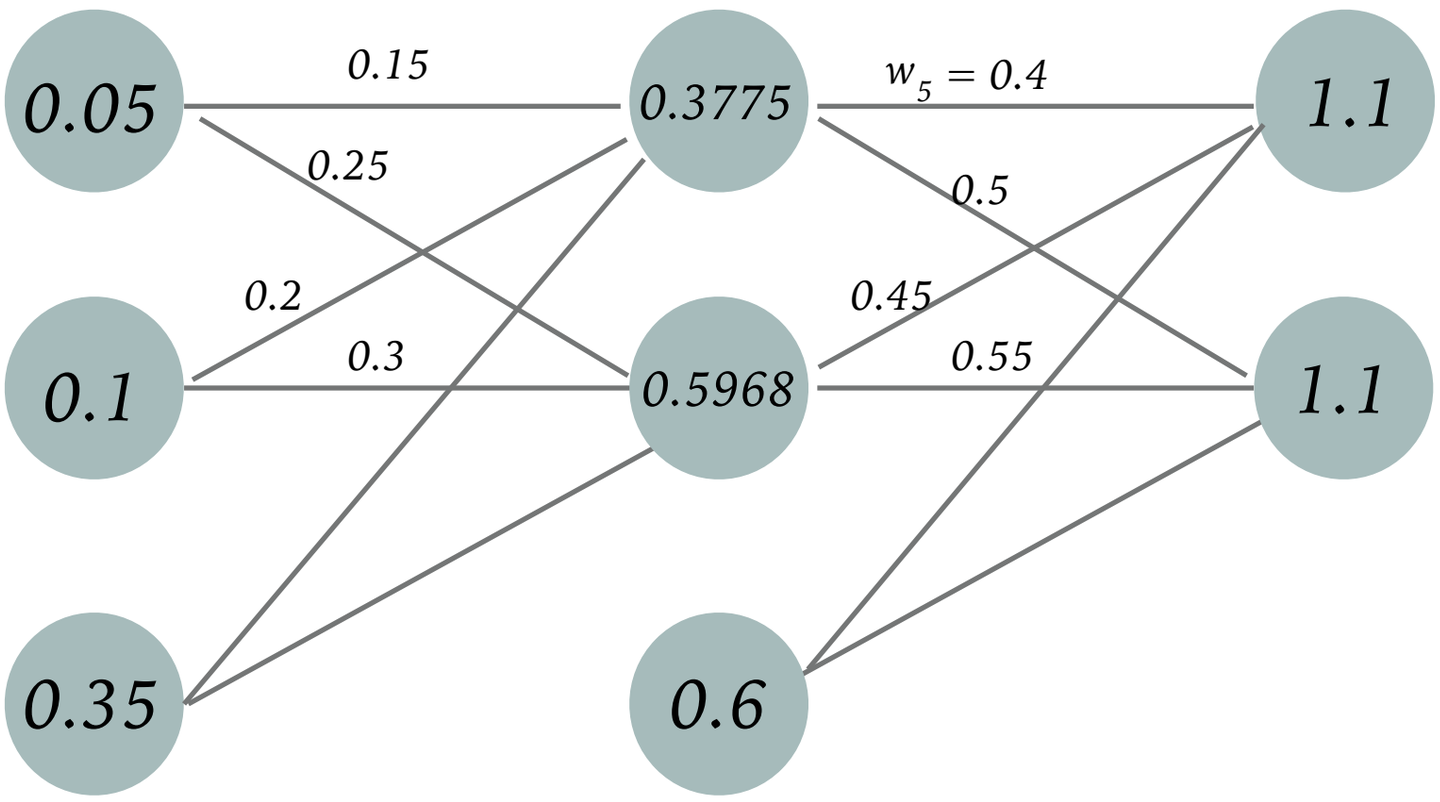
target values

$y_1 = 0.01$

$y_2 = 0.99$

$$Error\_w_5 = dE_{total} / dw_5 = (dE_{total} / dy_1) * (dy_1 / dw_5)$$

EXAMPLE: STOCHASTIC GRADIENTS DESCENT



target values

$y_1 = 0.01$

$y_2 = 0.99$

$$\text{Update\_}w_5 = w_5 - \eta(dE_{total} / dw_5)$$

Where η is a constant called a learning rate.

It dictates how fast the neural net is learns.

If we do this for every weight and bias in the network, the loss will slowly decrease and our network will improve. This way of updating the weights is commonly known as **stochastic gradient descent**.

IN SUM

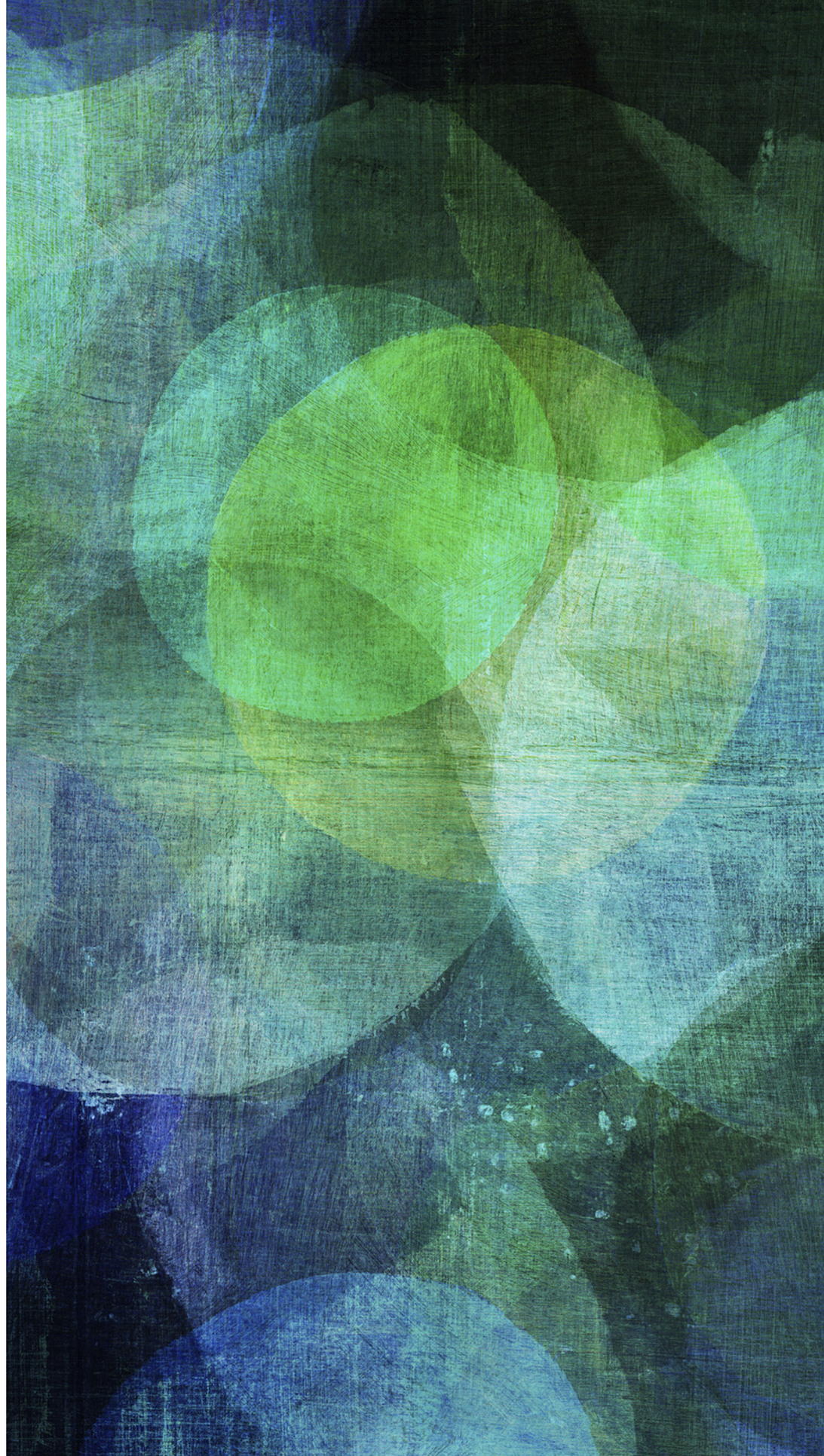
The training process of a neural ent will look like this:

- Choose **one** example from the dataset.
- Calculate all the loss.
- Use the update equation to update each weight.
- Go back to step 1.

AFTER THIS LECTURE

- You should understand at a very high level how backpropagation works.
- You don't need to know the maths, you only need to understand why they are used, i.e. understand that derivatives are just used to calculate how the weights should be changed in order to minimise the error.

Reflective Question (to be discussed at the tutorials)



REFLECTIVE QUESTION

1. Stochastic gradient descent updates the weights after each iteration. Can you think of other ways to update the weights and why would that be more/less useful?

REFERENCES/FURTHER READING

- The Perceptron by Hal Daume II: http://ciml.info/dl/v0_99/ciml-v0_99-ch04.pdf
- Neural Networks: http://ciml.info/dl/v0_99/ciml-v0_99-ch10.pdf

PRACTICAL

- The practical will focus on some core Python skills.