Week 5a
Neural Networks I

Fundamentals of Machine Learning

2023-24

UNIVERSITY OF SUSSEX

Dr Benjamin Evans

Here's what we'll do over the next few weeks…

- **This week:**
  - **Neural networks I** - single-layer (Perceptron) [binary classification, supervised learning]
  - **Clustering** - *k*-means [classification, unsupervised]

- **Next week:** Catch up week. **Labs for this week (the perceptron).**
  - No lecture on Tuesday 5th March
  - *Revision lecture with me covering difficult MCQ questions: Thursday 7th March at 9am*

- **Week 7:**
  - **Pre-processing**

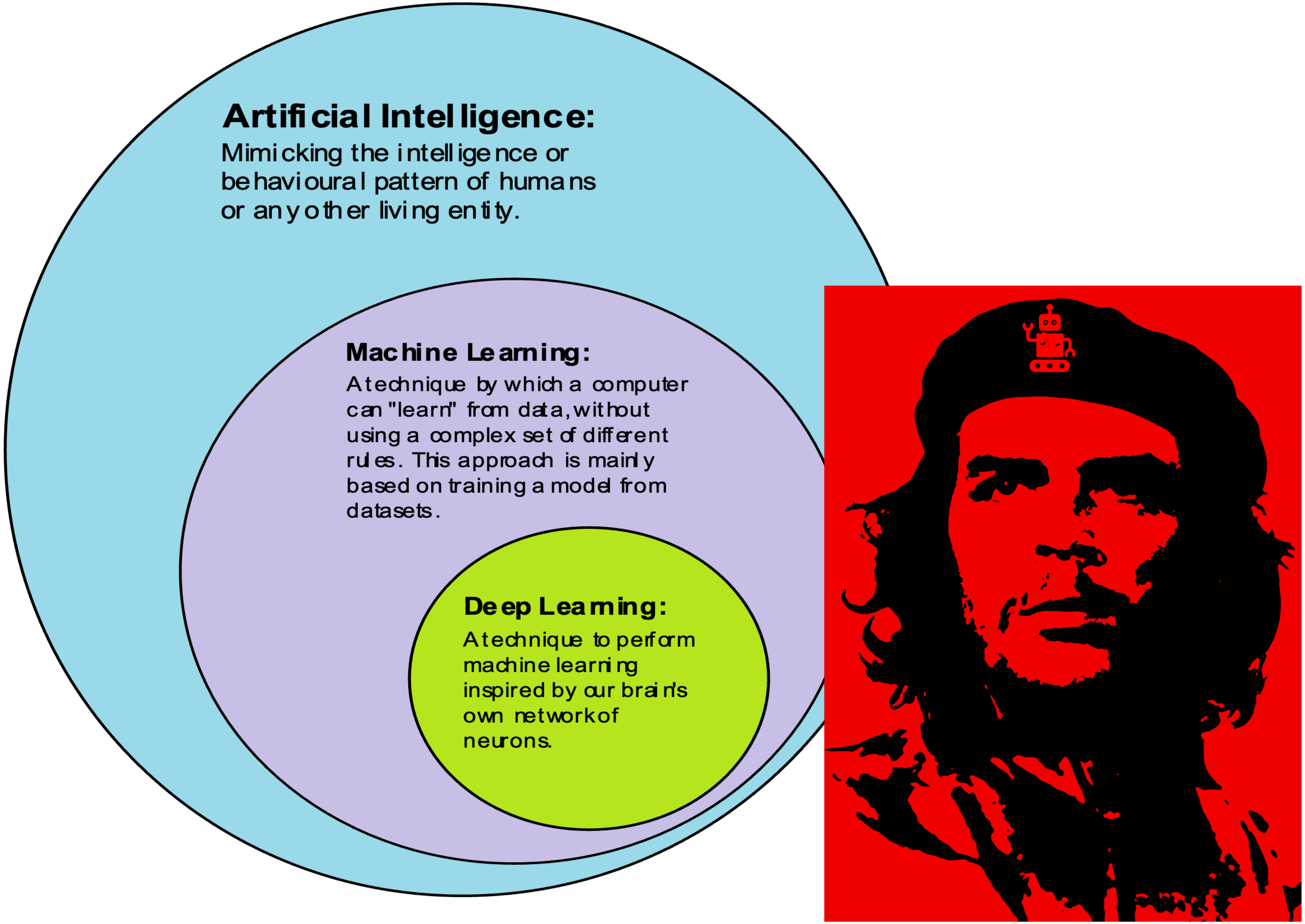
- **Week 8:**
  - **Neural networks II** – multi-layer perceptron and backpropagation

- **Office Hours:** please email to book.

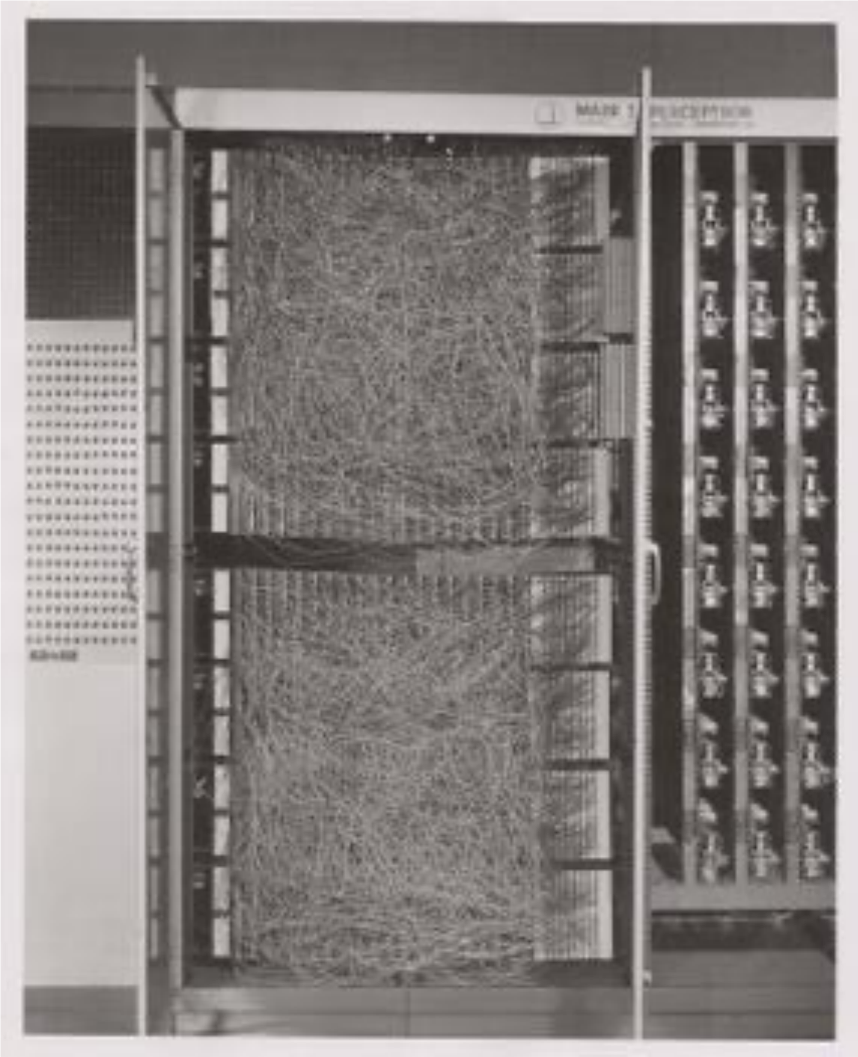
Dr Johanna Senk	Dr Benjamin Evans
Wednesdays: 11-12pm	Tuesdays: 12-1pm
Thursdays: 11-12pm	Thursdays: 3-4pm

What will you learn today?

- The Perceptron (artificial neuron)
  - Background
    - How it fits into AI
    - History & Biological Inspiration
  - Basic idea
    - Abstraction as a model
    - How it works
  - What you can do with it
    - Binary classification
    - Supervised learning
  - How is it trained / how does it learn?
    - Adjust weights according to error
    - Gradient descent

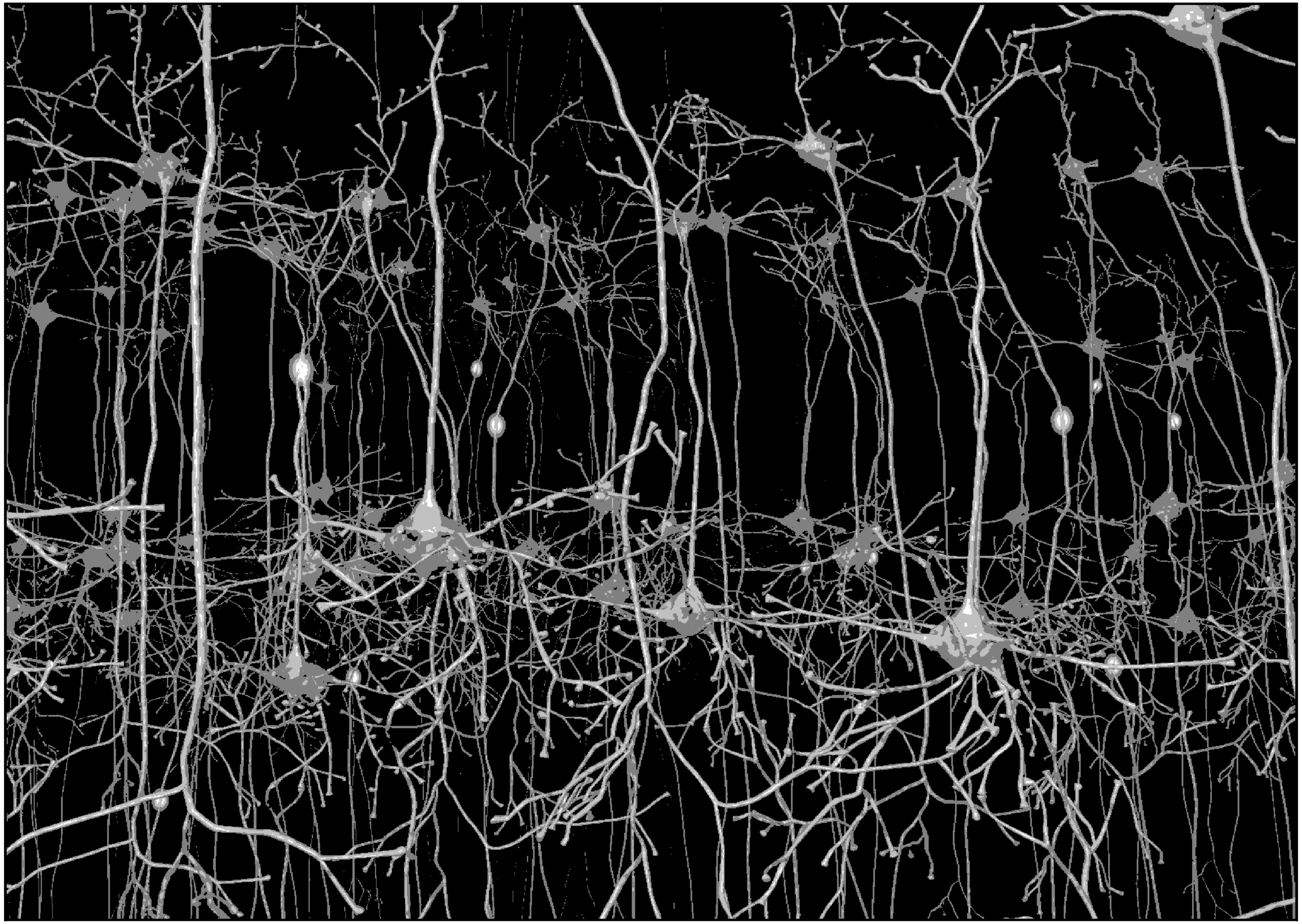

Artificial Intelligence:
Mimicking the intelligence or behavioural pattern of humans or any other living entity.
Machine Learning:
A technique by which a computer can "learn" from data, without using a complex set of different rules. This approach is mainly based on training a model from datasets.
Deep Learning:
A technique to perform machine learning inspired by our brain's own network of neurons.

The Mark I Perceptron at the
Cornell Aeronautical Laboratory
Invented by Frank Rosenblatt, 1957



"The Navy revealed the embryo of an electronic computer that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."

New York Times 8/vii/1958



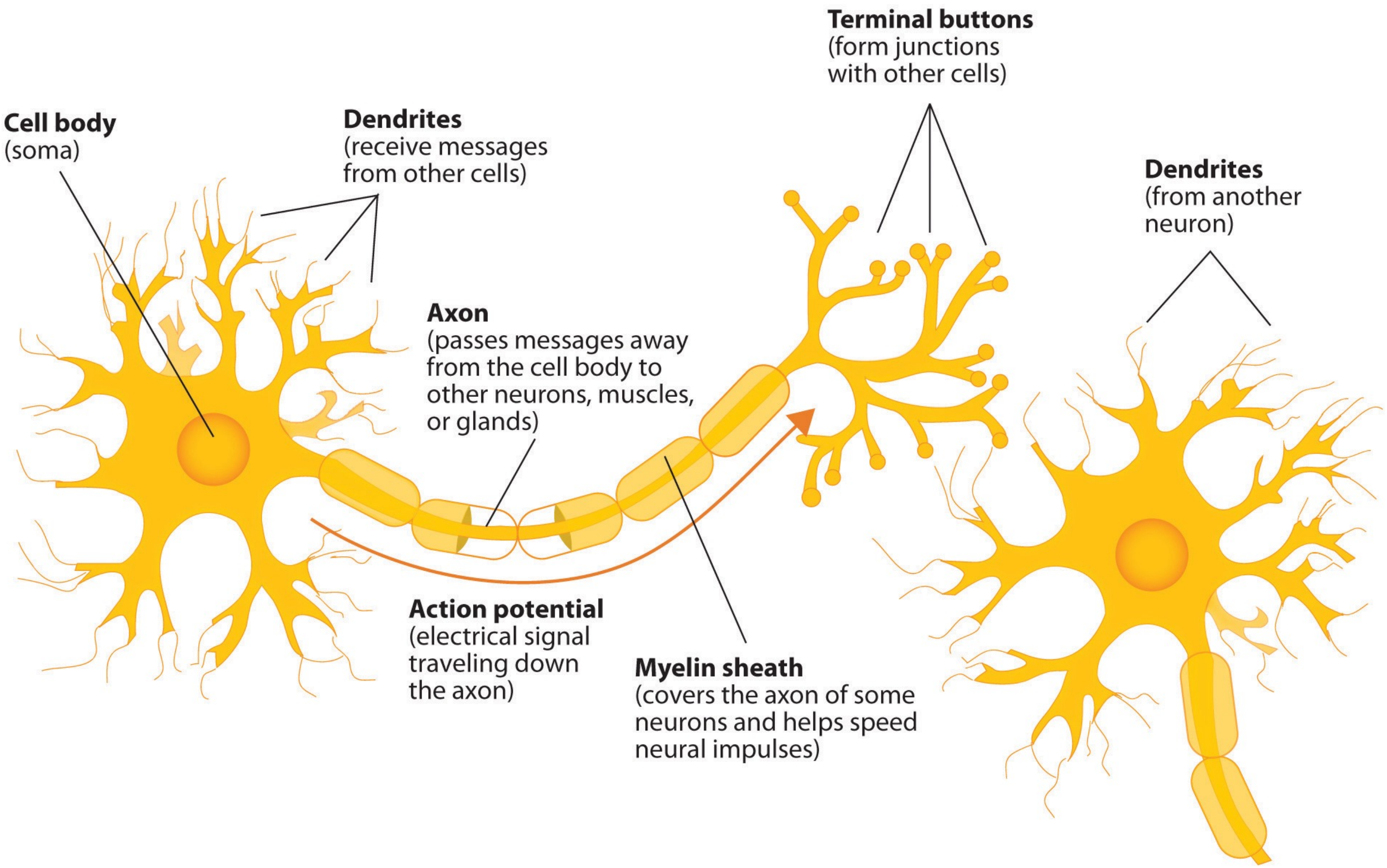
DIGITAL STUDIO SA

CG image of the vertical organization of neurons in the primary visual cortex (V1).

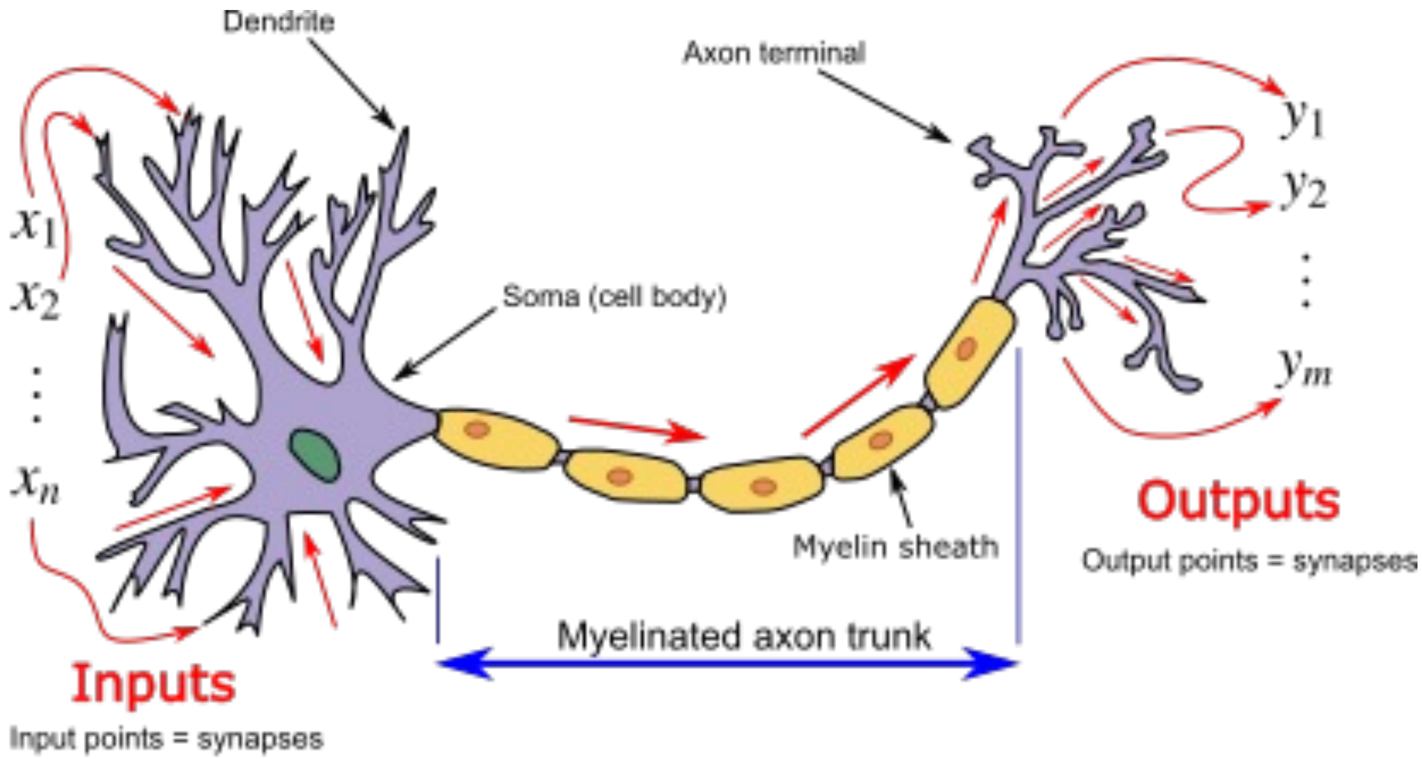
Smooth stellate and spiny stellate cells relay visual information coming out from the retina to pyramidal cells, themselves doing a first basic computation of visual motion perception.

version of July 2000

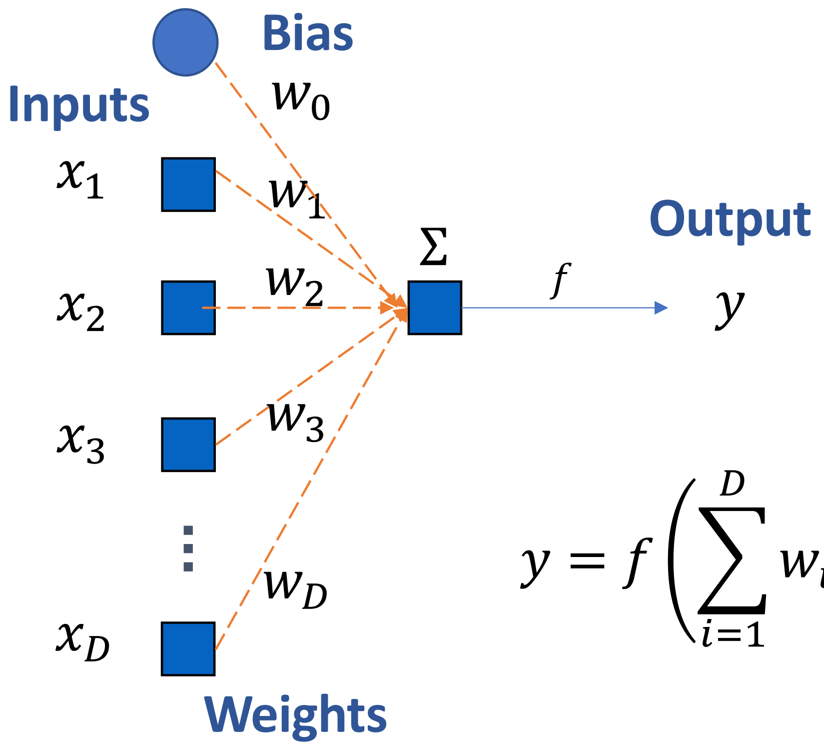
Inspiration from biological neurons



Inspiration from biological neurons

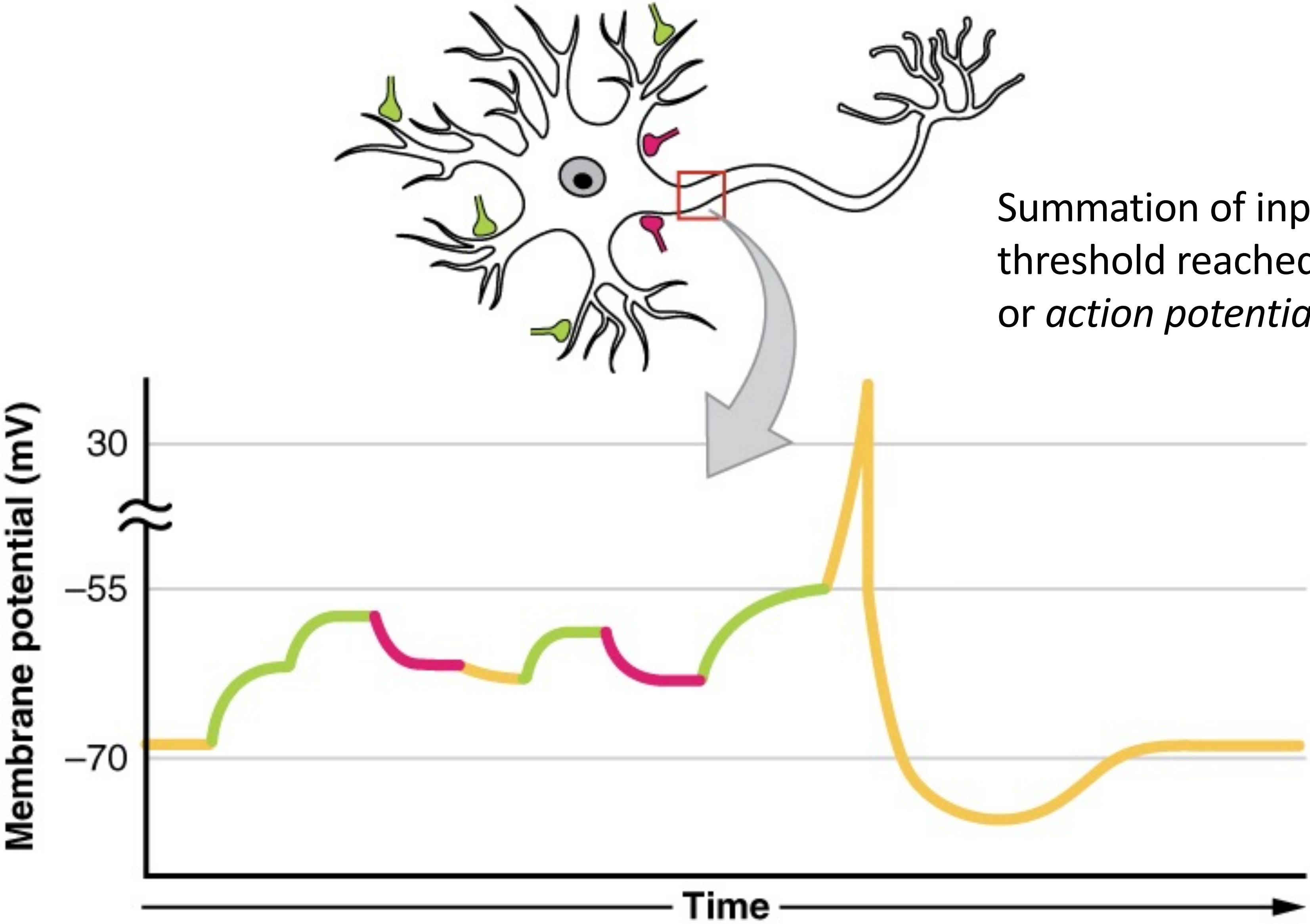


Inspiration from biological neurons



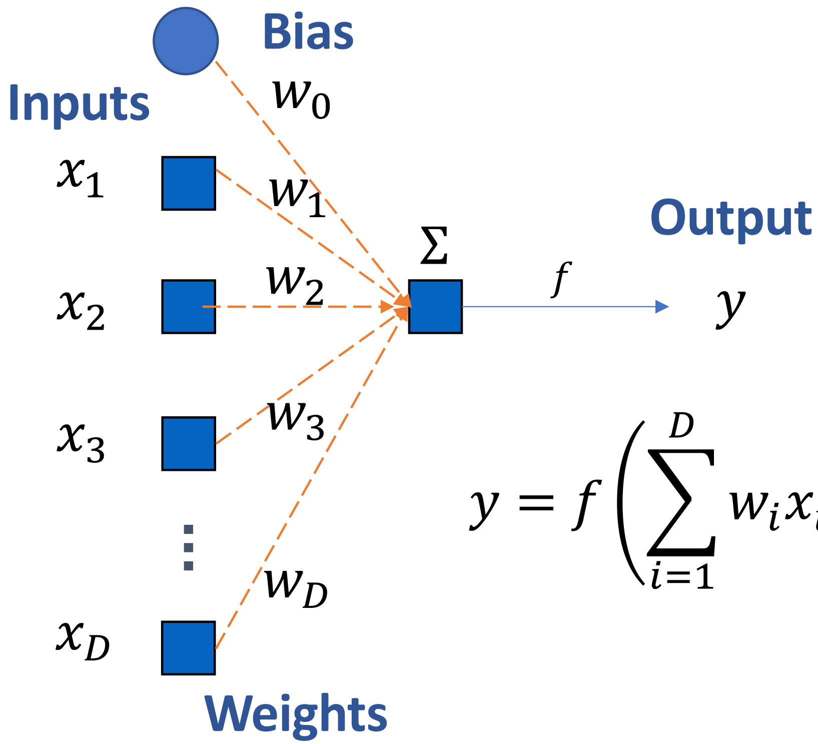
$$y = f\left(\sum_{i=1}^{D} w_i x_i + w_0\right)$$

Recording from a real neuron: membrane potential



Summation of inputs until threshold reached and a *spike* or *action potential* is emitted.

An artificial neuron – *The Perceptron*

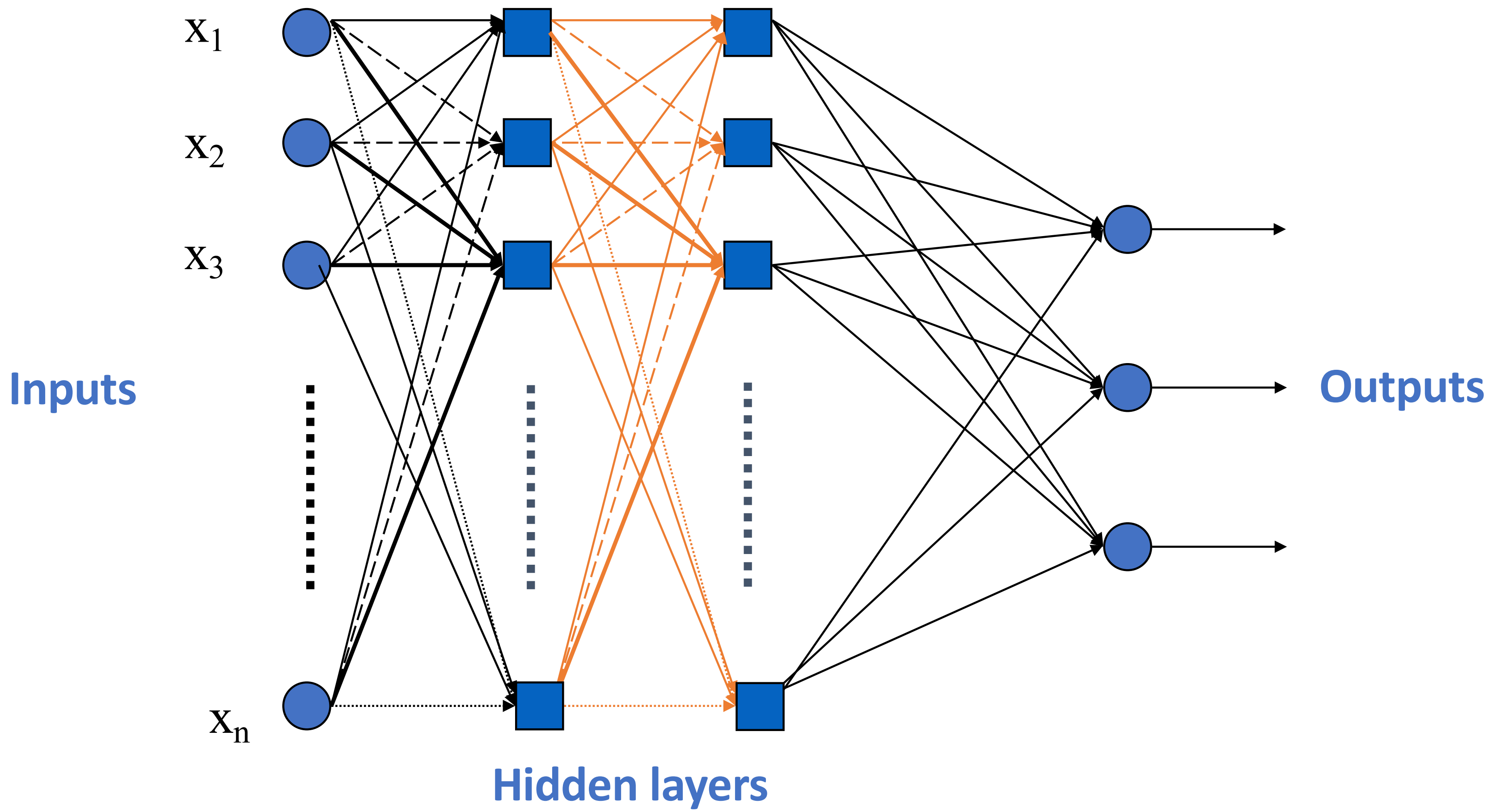


$$y = f\left(\sum_{i=1}^{D} w_i x_i + w_0\right)$$

Components of a general artificial neuron:

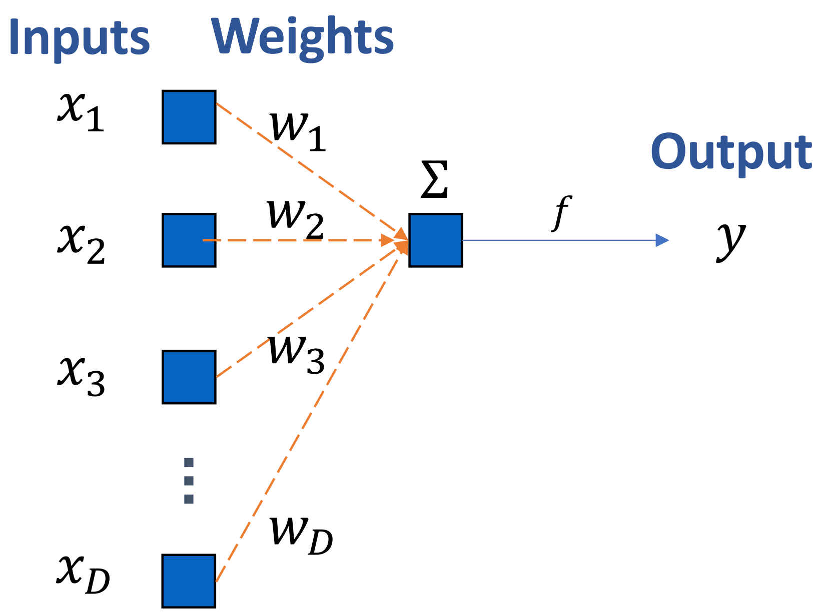
1. A set of **inputs**, $\mathbf{x_i}$
2. A set of **weights**, $w_1, w_2, \ldots, w_D$
3. A **bias**, w_0
4. An **activation function**, f
5. An **output**, y

Multi-layer perceptron – deep neural network



The single perceptron

Can be viewed as a model…



… or a function.

$$y = f\left(\sum_{i=1}^{D} w_i \cdot x_i + w_0\right)$$

$$y = f\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}\right)$$

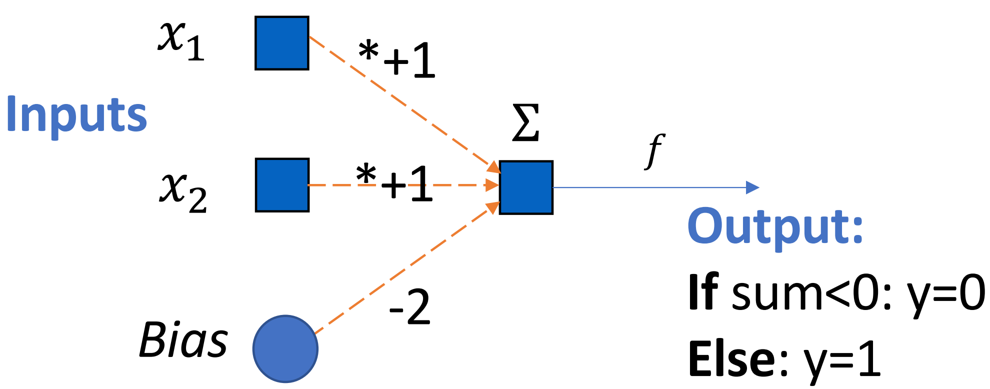
- Takes the values for each of the features, x_i as **input**
- Scales the features by their **weights** w_i and sums the products
- Passes result through a *non-linear* activation function, f
- Produces a binary classification as **output**, y, that is a function of the features: {0, 1} or {-1, 1}

What are the parameters?

- The **weights**, $w_1, w_2, \ldots, w_D$
- The **bias**, w_0
- (The **activation function**, f)

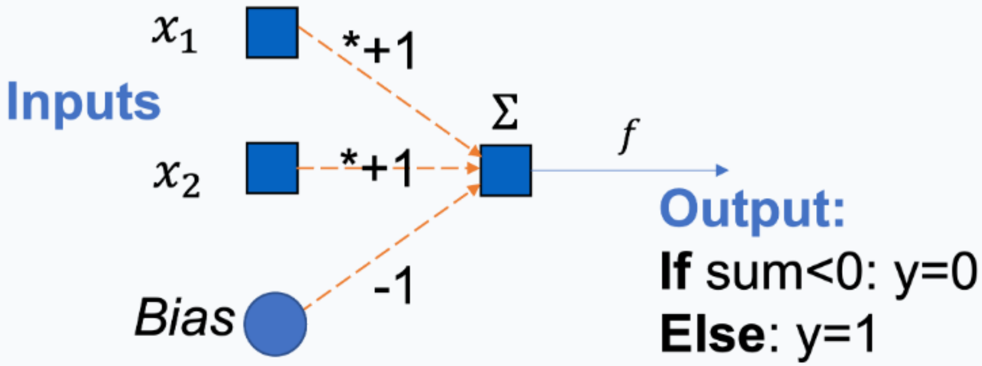
The most basic single perceptron

- Very simple artificial neuron can perform basic logical operations such as:
  - **AND**
  - **OR**
  - **NOT**



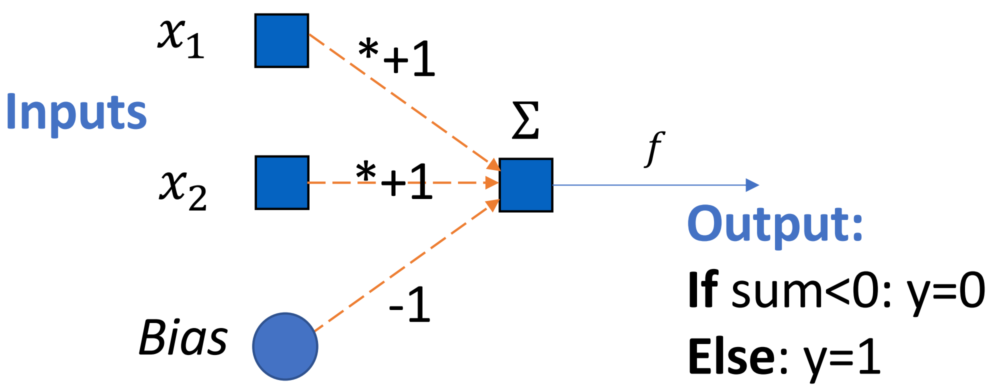
x_1	x_2	x_1 AND x_2
0	0	0
0	1	0
1	0	0
1	1	1

What logic gate is this perceptron computing?



- NOT x1
- x1 AND x2
- x1 OR x2
- NOT (x1 AND x2)
- None of the above

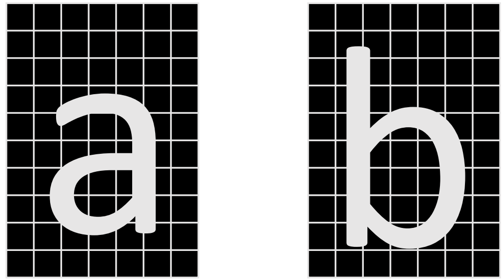
What logic gate is this perceptron computing?



x_1	x_2	x_1 OR x_2
0	0	0
0	1	1
1	0	1
1	1	1

- Idea: Groups of these "neuronal" logic gates could carry out *any* computation, even though each neuron was very limited.
  - Could computers be built from these simple units and reproduce the computational power of biological brains?
  - Are *biological* neurons performing logical operations?

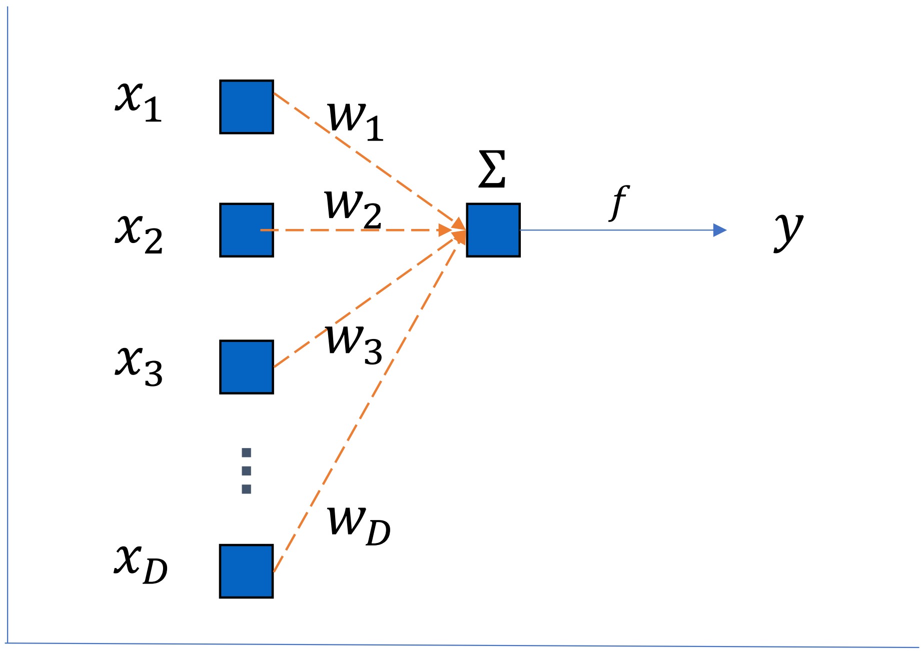
E.g. Handwritten digit classification:



- First need a data set to learn from: sets of characters
- How are they represented? E.g. as an input vector $\underline{x} = (x_1, \ldots, x_D)$ to the network (e.g. vector of ones and zeroes for each pixel according to whether it is black/white).
- Set of input vectors is our **training set** which have already been labelled as *a*'s and *b*'s.
- Given a training set, our goal is to tell if a new image is an *a* or *b* i.e. classify it into one of 2 classes C_1 (all *a*'s) or C_2 (all *b*'s) (in general one of k classes $C_1, \ldots, C_k$)

Intuition: real neural networks do this well, so maybe artificial ones can do the same.

For 2 class classification we want the network output y (a function of the inputs and network parameters) to be:

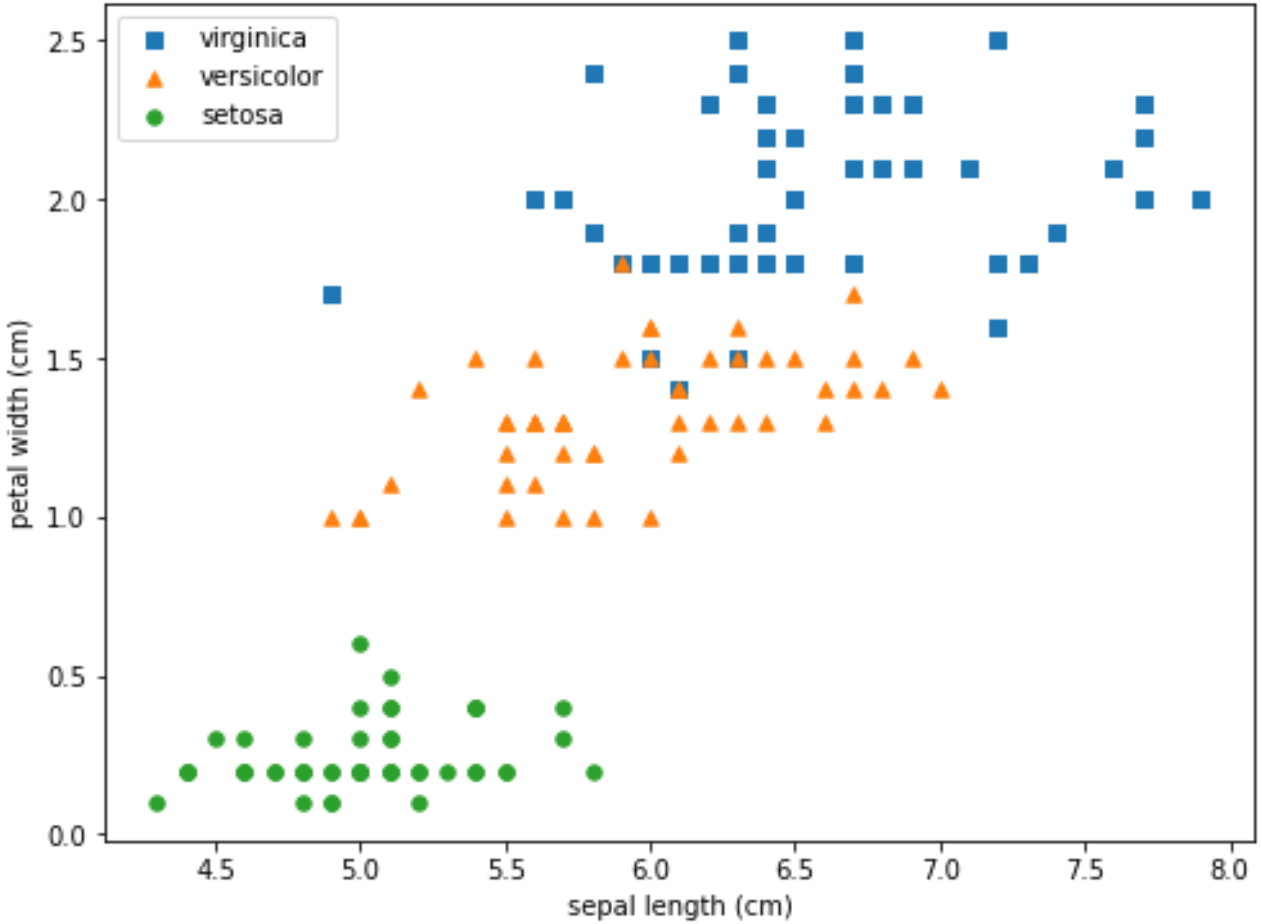


y(x) = 1 if x is an image of letter *a*

y(x) = -1 if x is an image of letter *b*

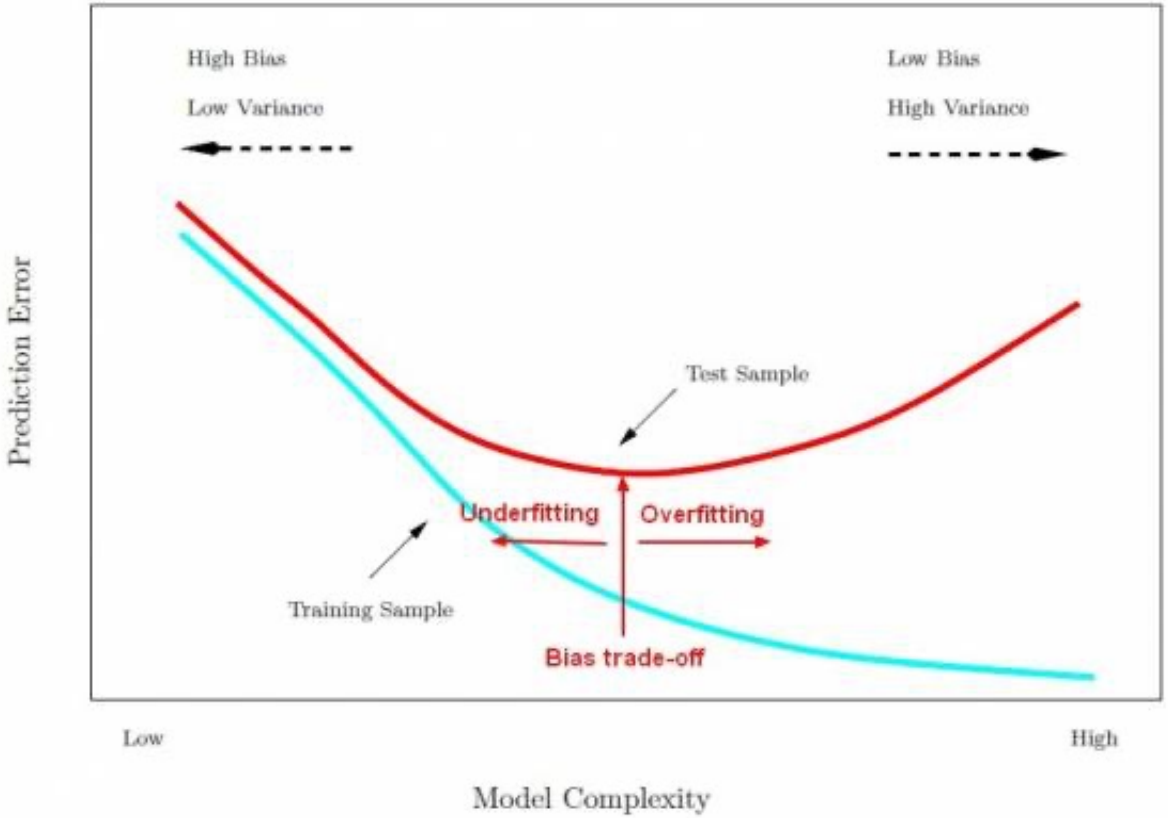
$$y = f\left(\sum_{i=1}^{D} w_i x_i + w_0\right)$$

E.g. 2. Iris species A, or not species A?



Supervised learning

Recall the bias / variance trade-off from last week…



We use the labelled data to perform **supervised learning** (training, adaptation) i.e.:

Change the weights between neurons according to the training examples (and possibly prior knowledge of the problem)

The purpose of learning is to minimize:

- **training errors** on learning data: **learning error**
- **prediction errors** on new, unseen data: **generalization error**

The Perceptron as a **classifier**

For D-dimensional data, a perceptron consists of:
D **weights**, a **bias** and a thresholding **activation function**.

For 2D data we have:

1. Weighted Sum of the inputs

2. Pass through Heaviside function:
T(z) = -1 if z < 0
T(z) = 1 if z >= 0

Output = class decision

Hint: View the bias as another weight from an input which is constantly on

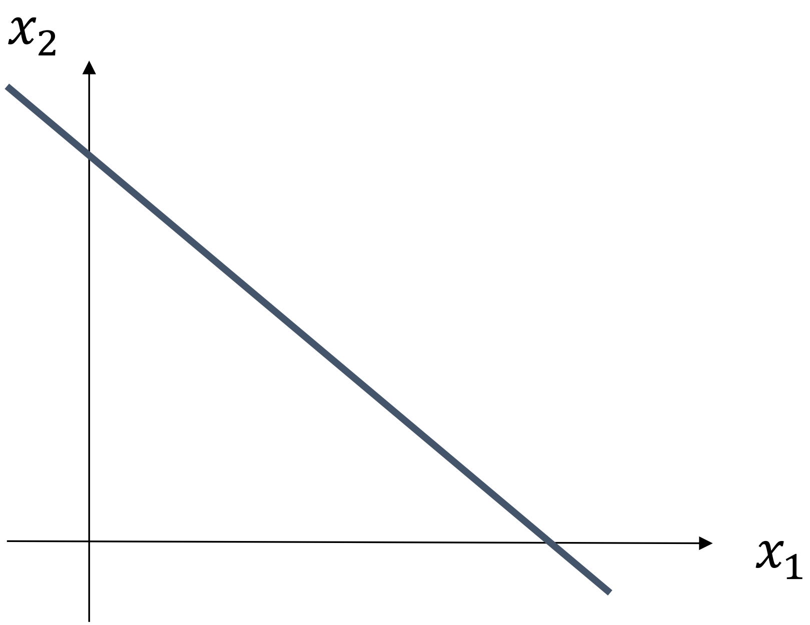
$z = \underline{w}.\underline{x}$ where $\underline{x}$ is $[1, x_1, x_2, \ldots, x_D]$
and $\underline{w}$ is $[w_0, w_1, w_2, \ldots, w_D]$

Interpretation of the weights

Since the Heaviside function is thresholded at 0, the decision boundary is where:

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

This is the equation of a **straight line**:



Recall: $y = mx + c$

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2} \qquad \text{if } w_2 \neq 0$$

$$x_1 = -\frac{w_0}{w_1} \qquad \text{if } w_2 = 0$$

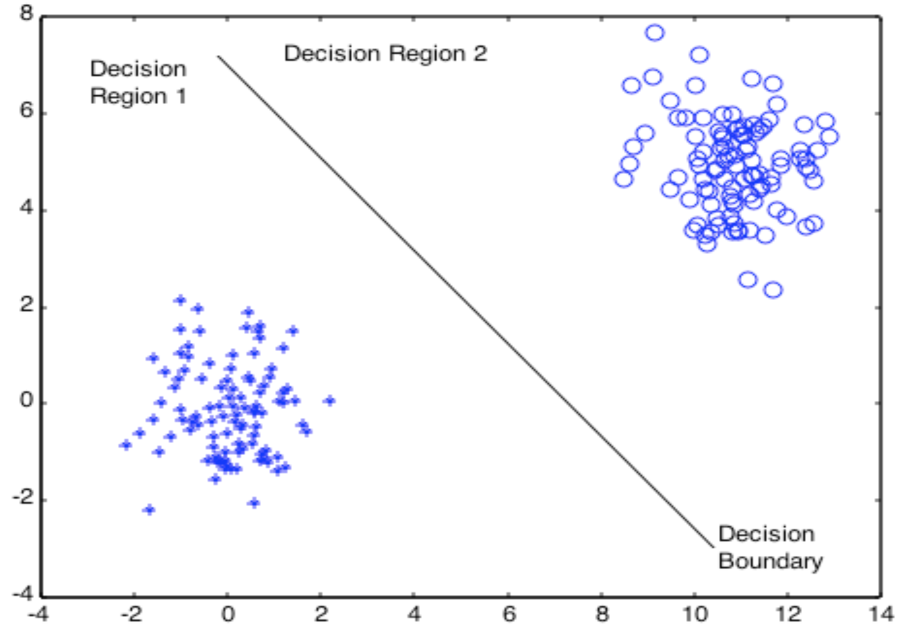
The prediction is: $sgn(\mathbf{w}^{\mathrm{T}}\mathbf{x})$
→ we only care about which side of the decision boundary we are on, *not* how far we are from it.

Perceptron = Linear discriminant function

The **discriminant is always *linear* for single perceptron!**

i.e., output y depends only on a **linear** function of the inputs:

$$z = \sum_{i=1}^{D} w_i x_i + w_0 = \mathbf{w}.\mathbf{x}$$

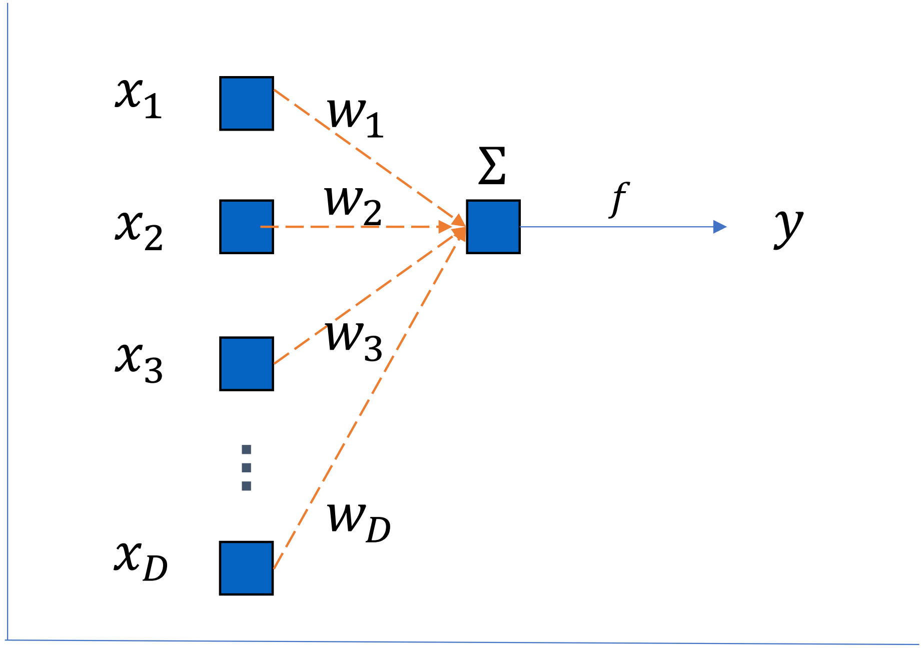


Separate the two classes using a straight line in feature space.

In 3D, it's a plane. In higher dimensions, it's a *hyper-plane*.

Activation function

$$y = f(z) \qquad z = \sum_{i=1}^{D} w_i x_i + w_0 = \mathbf{w} \cdot \mathbf{x}$$



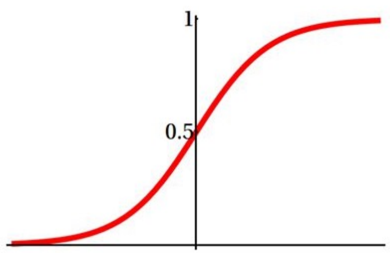
We have been considering the activation function to be the **Heaviside** step function:

$$\boldsymbol{T(z)} = \begin{cases} \mathbf{-1 \text{ if } z < 0} \\ \mathbf{1 \text{ if } z \geq 0} \end{cases}$$

Other activation functions can also be used.

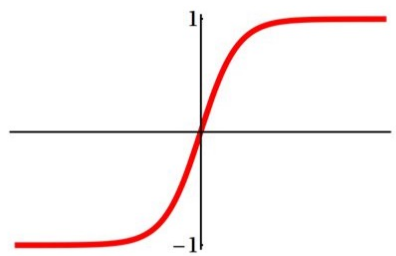
Common choices are the **sigmoid** or **tanh** functions, also called *logistic functions*.

$f(z) = 1/(1 + e^{-z})$



sigmoid

$f(z) = \tanh(z)$



tanh

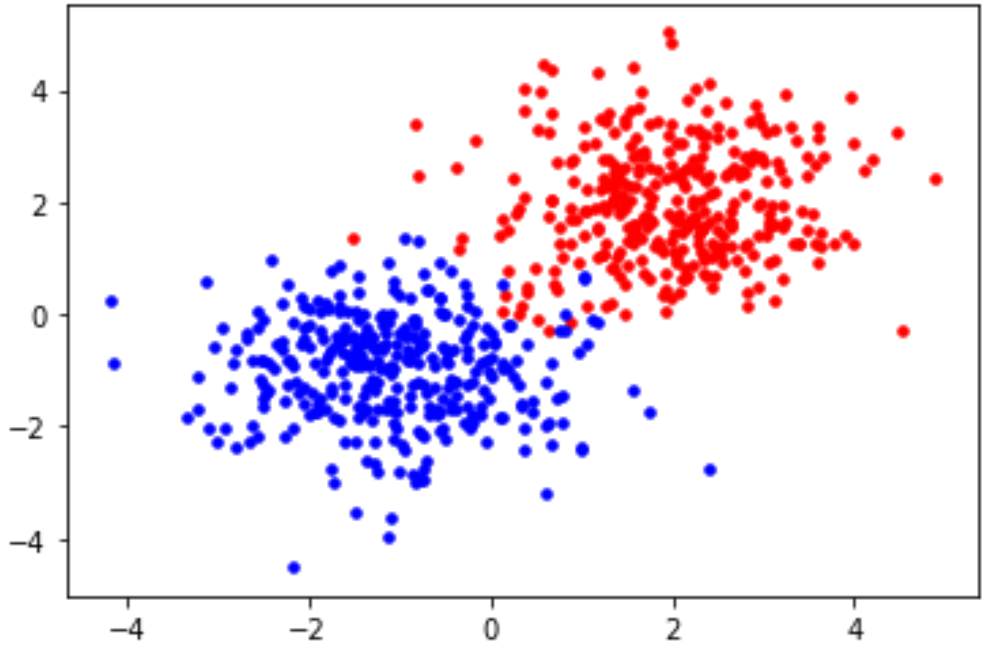
Relationship to logistic regression

We have seen that the perceptron is a linear discriminant function.

Use of a logistic activation function, together with normally-distributed data means that the activation gives you the posterior probabilities:

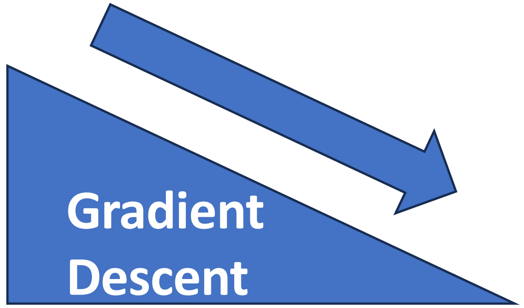
$$p(C_k|x)$$

This is **logistic regression!**



Learning / training

- Standard procedure for training the weights is by

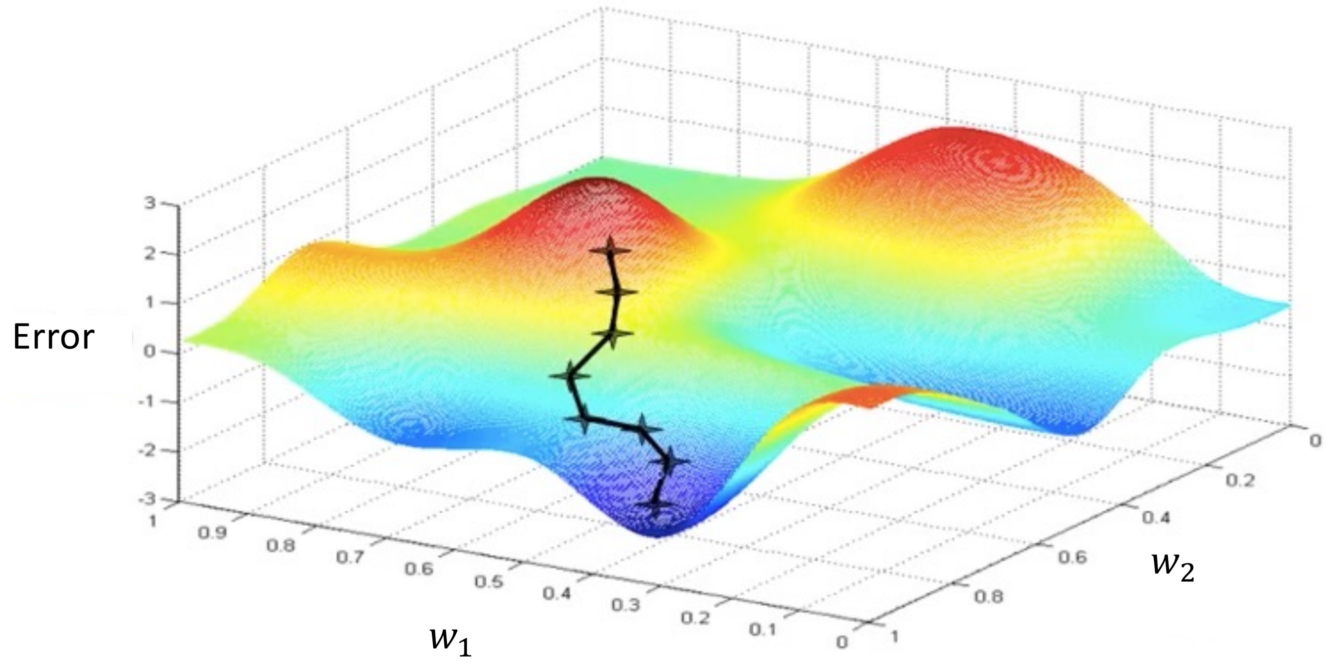


- Take a set of training data from known classes and use an error function $E(\mathbf{w})$ to specify an error for each sample.
- Update the weights with:

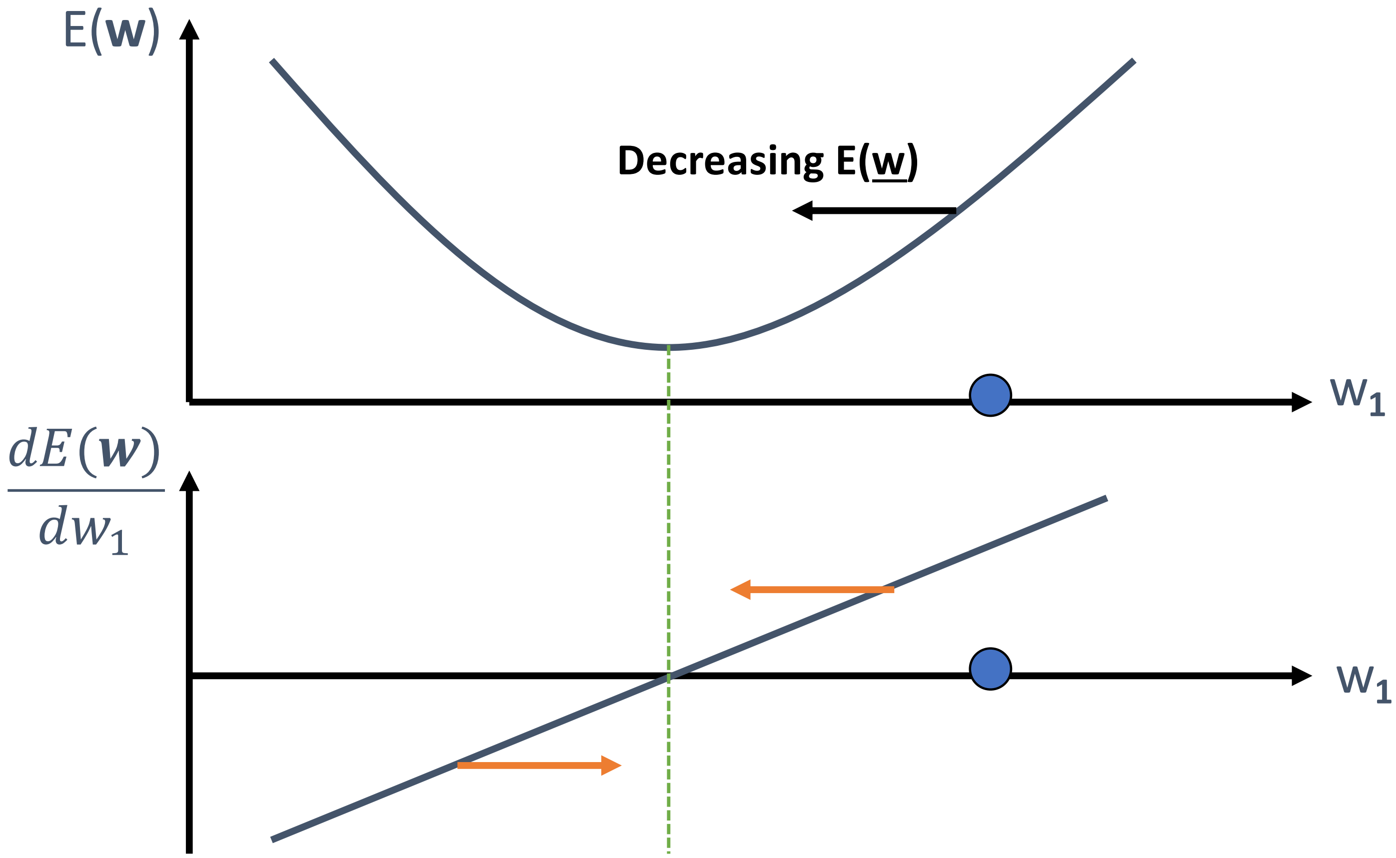
$$\mathbf{w}_{new} = \mathbf{w}_{old} - \eta \, \nabla E(\mathbf{w})$$

- Where,
  - $\nabla \mathrm{E}(\mathbf{w})$ is the rate of change of the error with respect to $\mathbf{w}$
  - η is the learning rate (positive, usually small: $0 < \eta < 1$)
- This moves us *downhill* i.e., in direction $-\nabla E(\mathbf{w})$
  - This is the direction of steepest *decrease* since $+\nabla E(\mathbf{w})$ is the gradient, i.e. the direction of steepest *increase*
- How far we go (the step size) is determined by the value of η

Visualisation of gradient descent

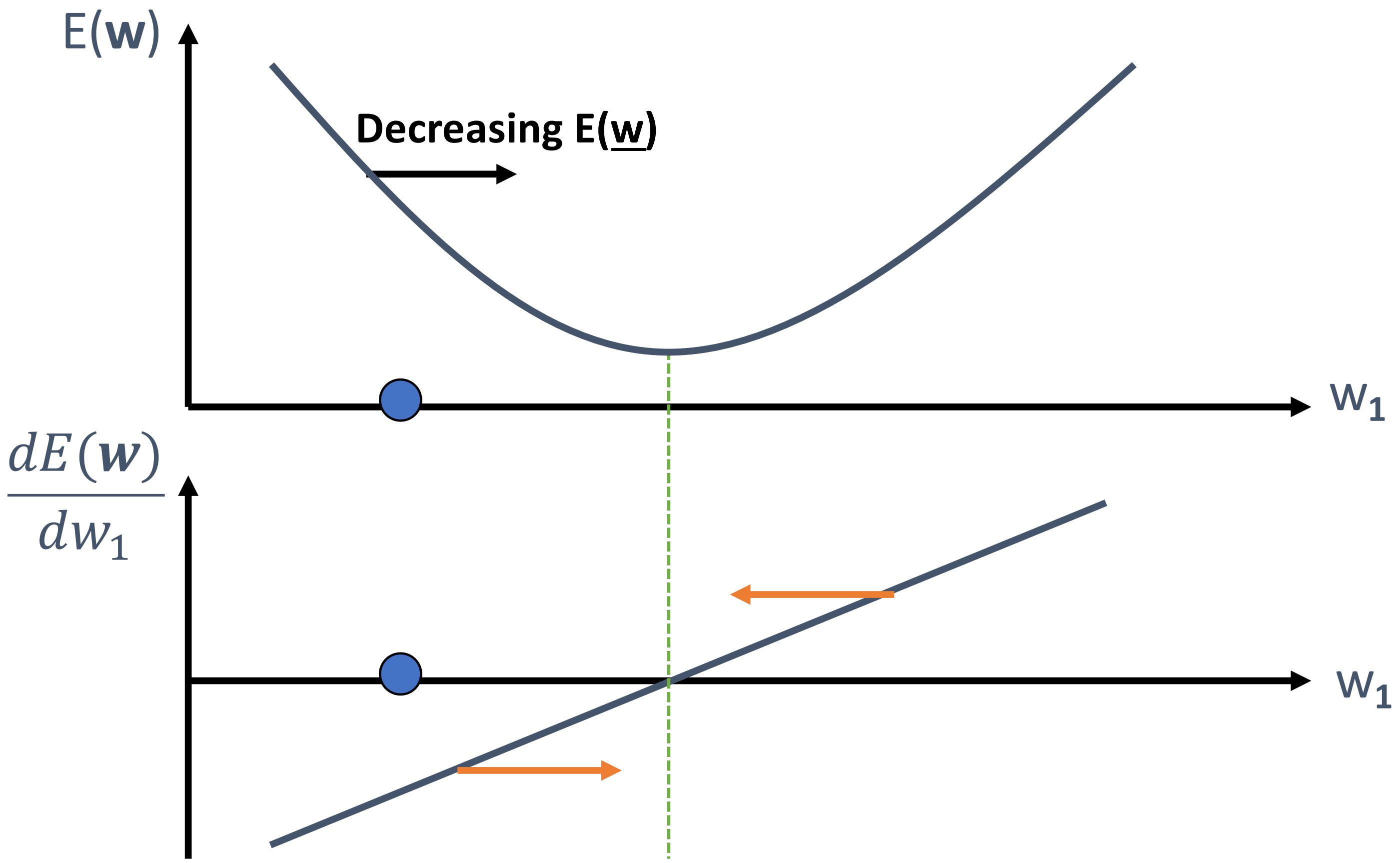


Moving Downhill: Move in direction of negative derivative



w_1 is updated to $w_1 - \eta \frac{dE(\boldsymbol{w})}{dw_1}$ and $\frac{dE(\boldsymbol{w})}{dw_1} > 0$. **i.e., the rule decreases** w_1

Moving Downhill: Move in direction of negative derivative



w_1 is updated to $w_1 - \eta \frac{dE(\mathbf{w})}{dw_1}$ and $\frac{dE(\mathbf{w})}{dw_1} < 0$. **i.e., the rule increases** w_1

Illustration of Gradient Descent (2-d)

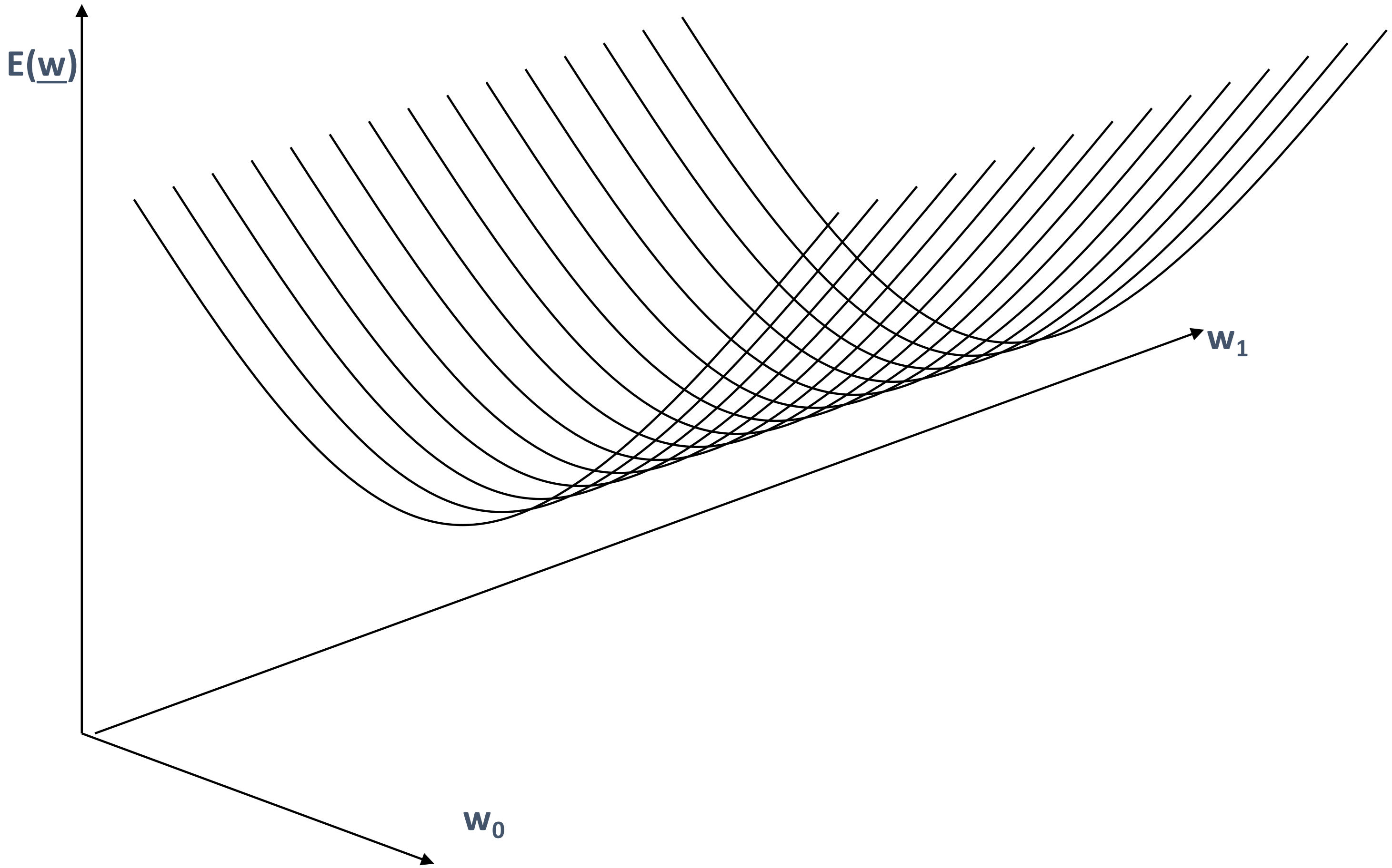


Illustration of Gradient Descent

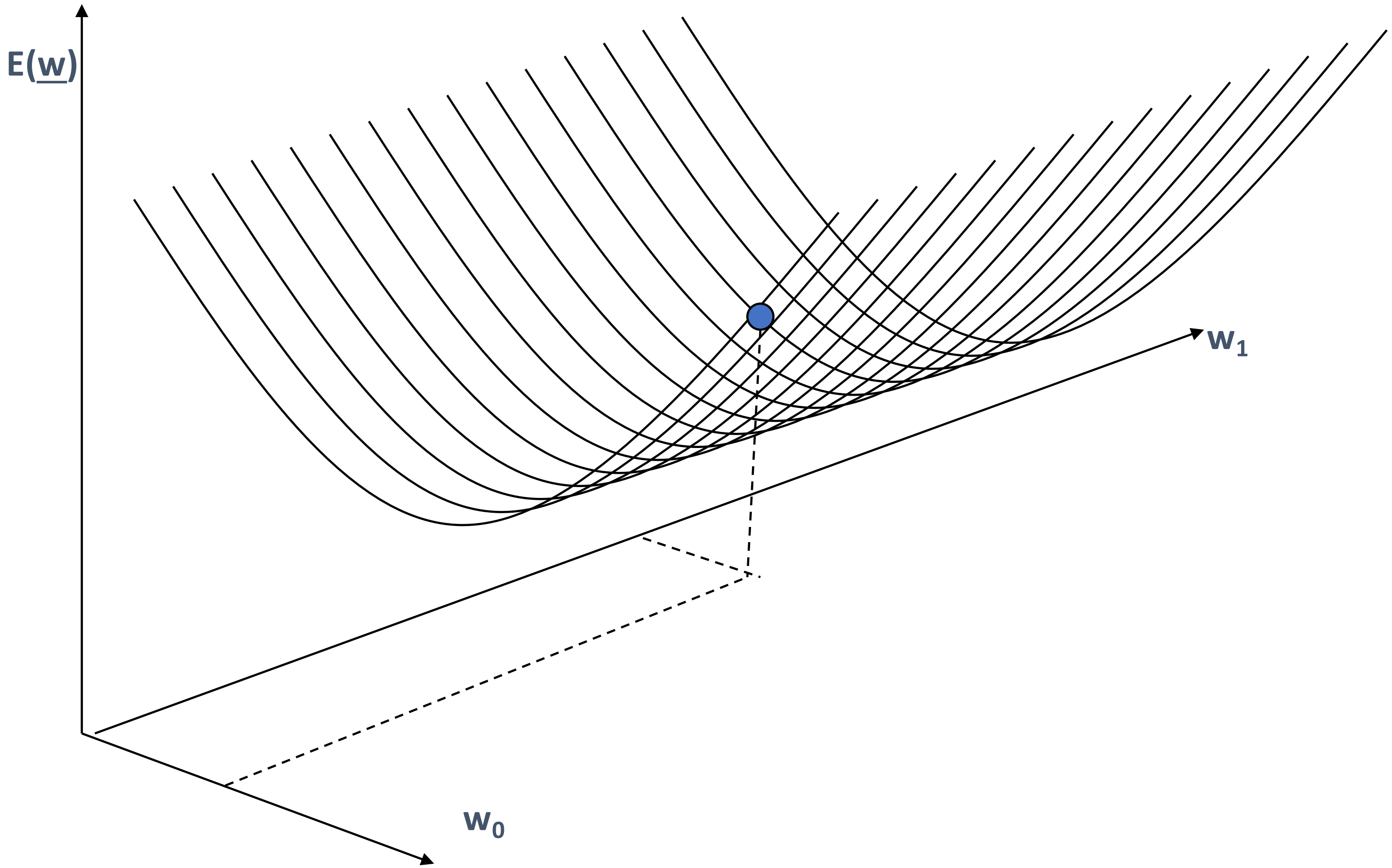


Illustration of Gradient Descent

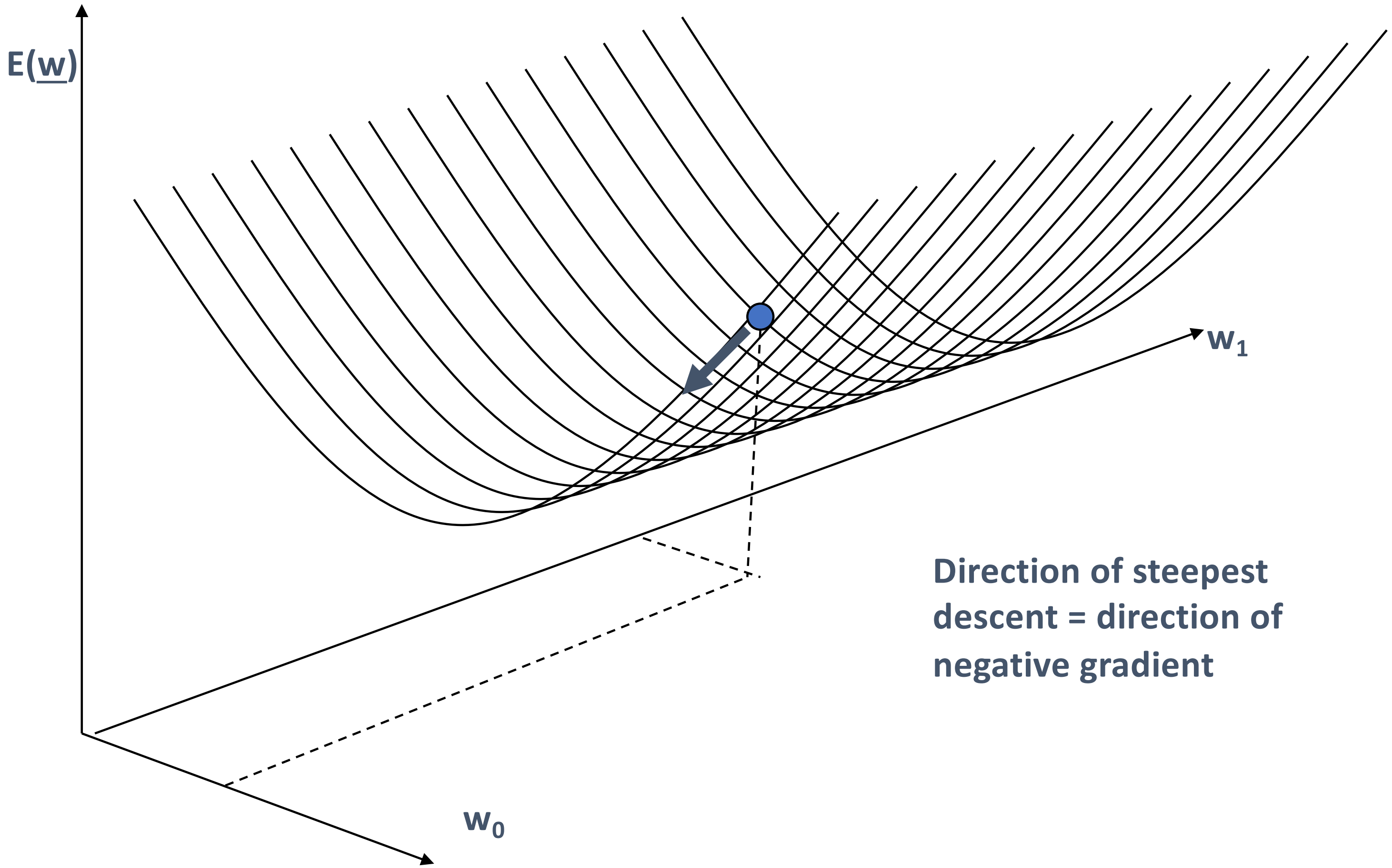
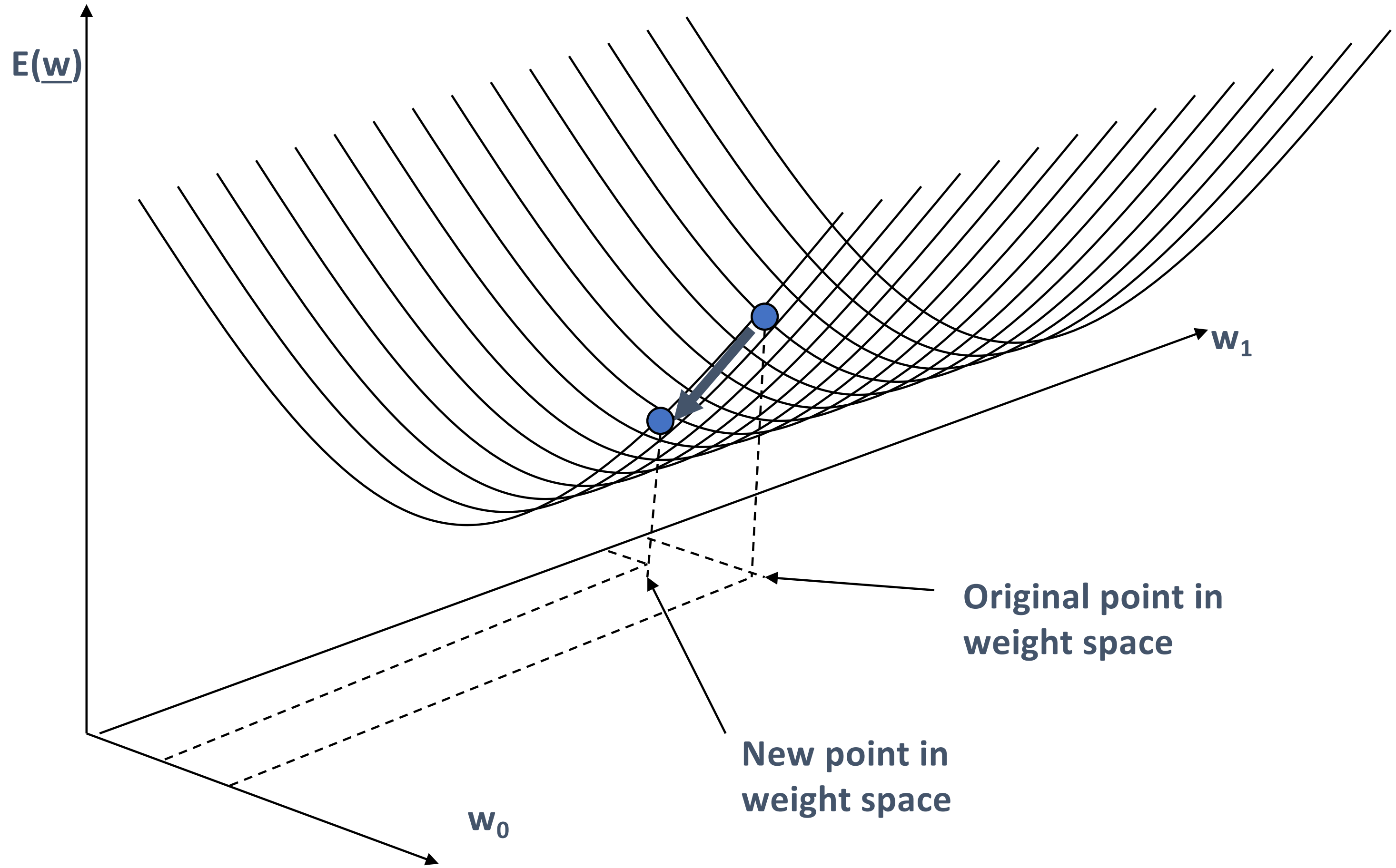


Illustration of Gradient Descent



Error for a perceptron

The prediction is $sgn(\mathbf{w}^{\mathrm{T}}\mathbf{x})$ and so the error is $E(\mathbf{w}) = 0$ when classification is correct (the predicted and actual signs match) and non-zero otherwise.

When wrong, the sign specifies the type of error (and the value indicates how far the perceptron was from getting it right).

$E(\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{x}$ if output is **+**1 but should have been **-**1

$E(\mathbf{w}) = -\mathbf{w}^{\mathrm{T}}\mathbf{x}$ if output is **-**1 but should have been **+**1

Recall:

$$\mathbf{w}^{\mathrm{T}}\mathbf{x} = \mathbf{w}\cdot\mathbf{x} = \sum_{i=1}^{D} w_i x_i + w_0$$

Or equivalently to get the correct sign, we can write:

$$E(\boldsymbol{w}) = \frac{1}{2}\mathbf{w}\cdot\mathbf{x}(y-c)$$

y is the output (prediction)

c is the true class

The sign ensures the direction of the gradient is correct for adjusting the weights.

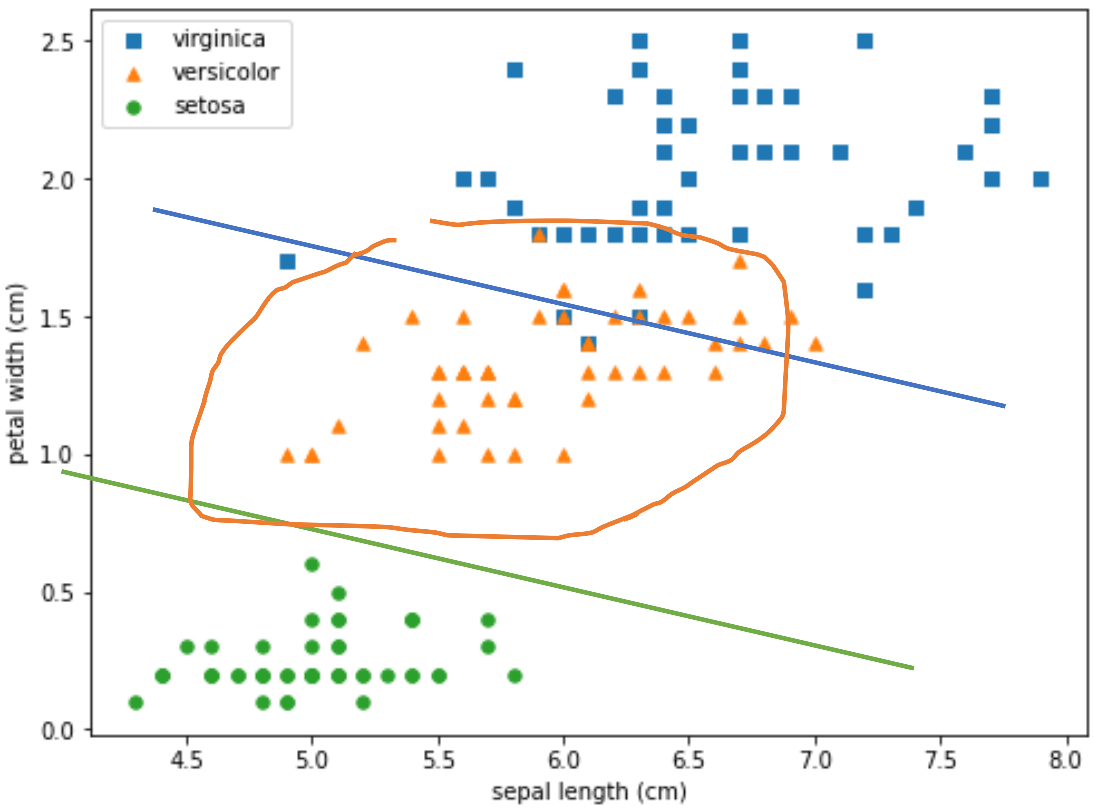
Learning for a perceptron

	Vector notation	Scalar notation
Error (loss)	$E(\mathbf{w}) = \frac{1}{2}(y - c)(\mathbf{w} \cdot \mathbf{x})$	$E(w) = \frac{1}{2}(y - c)(w_0 + w_1 x_1 + \ldots + w_d x_d)$
Gradient	$\nabla E(\mathbf{w}) = \frac{1}{2}(y - c)\mathbf{x}$	$\frac{\partial E(w)}{\partial w_i} = \frac{1}{2}(y - c)x_i$
Gradient descent Learning Rule	$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \frac{1}{2}(y - c)\mathbf{x}$	$w_i(t+1) = w_i(t) - \eta \frac{1}{2}(y - c)x_i$

On the error surface, each new weight is directly *downhill* from the old weight

η is how much to change in that direction

When does a perceptron work?



When the boundary between classes is approximately **linear** but performance degrades when the boundary is non-linear.

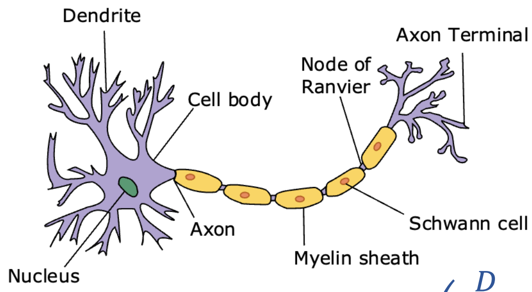
- 100% accuracy when data are **linearly separable** (e.g. OR, AND)
- Imperfect accuracy when data are **not linearly separable** (e.g. XOR)
    - enters an infinite training loop unless stopped

Hyperparameters / settings for training a perceptron

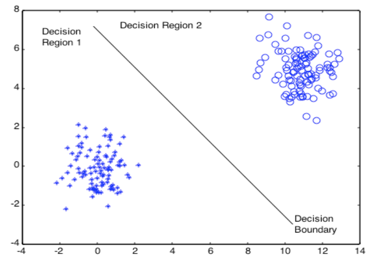
- Set **learning rate**: η
- Set initial **weight values**: $\mathbf{w}$
- When to stop?
  - Training set shown repeatedly until **stopping criteria** are met e.g., the error drops below a threshold or plateaus
  - Note, each full presentation of all patterns := '*epoch*'
- Which type of **training regime**?
  - *Sequential* (on-line, stochastic, or per-pattern): Weights updated after each pattern is presented.
  - *Batch*: Calculate the derivatives/weight changes for each pattern in the training set. Calculate total change by summing individual changes.

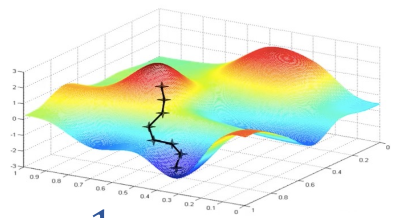
What have you learned today?

- The perceptron is a single **artificial neuron**, modelled on a biological neuron.
- It **scales the inputs by their weights**, **summing their products**, then classifies according to whether or not the **sum exceeds the threshold** of the activation function.
- It can do **binary classification**, if a linear decision boundary makes sense.
- The perceptron can learn to classify inputs by updating its weights using **gradient descent** in a **supervised learning** paradigm.



$$y = f\left(\sum_{i=1}^{D} w_i x_i + w_0\right)$$





$$w_i(t+1) = w_i(t) - \eta \frac{1}{2}(y - c)x_i$$