PRACTICAL MACHINE LEARNING: AN INTRODUCTION

Dr. Alexander Booth (he / him)
Universidad Católica del Norte, Chile
October 7th, 2024

Queen Mary
University of London

This Course: 4 Workshops (Lectures & Exercises)

First and foremost an interactive course! Please bring a laptop or share one with a friend!

- **Lecture 1 (Monday):** Introduction to machine learning with neural networks and linear regression.
- **Lecture 2 (Tuesday):** Optimisation and non-linear regression with neutral networks.
- **Lecture 3 (Thursday):** Classification and using convolutional neural networks for image classification.
- **Lecture 4 (Friday):** Robustness and adversarial examples in image classification problems.

This Course: 4 Workshops (Lectures & Exercises)

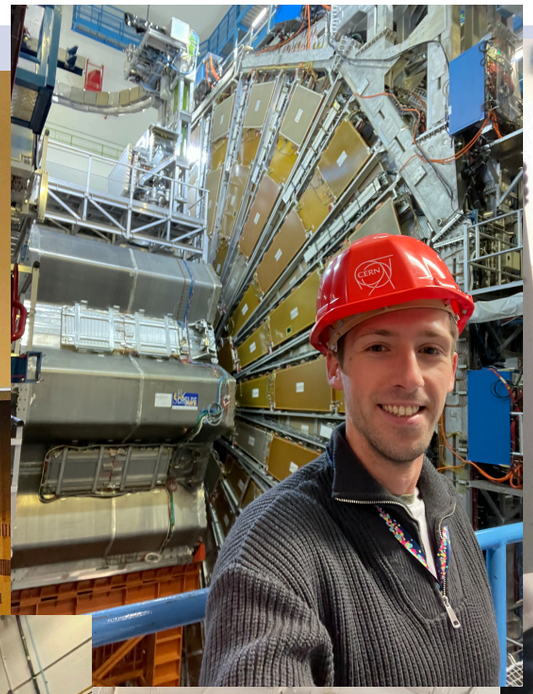
Software:

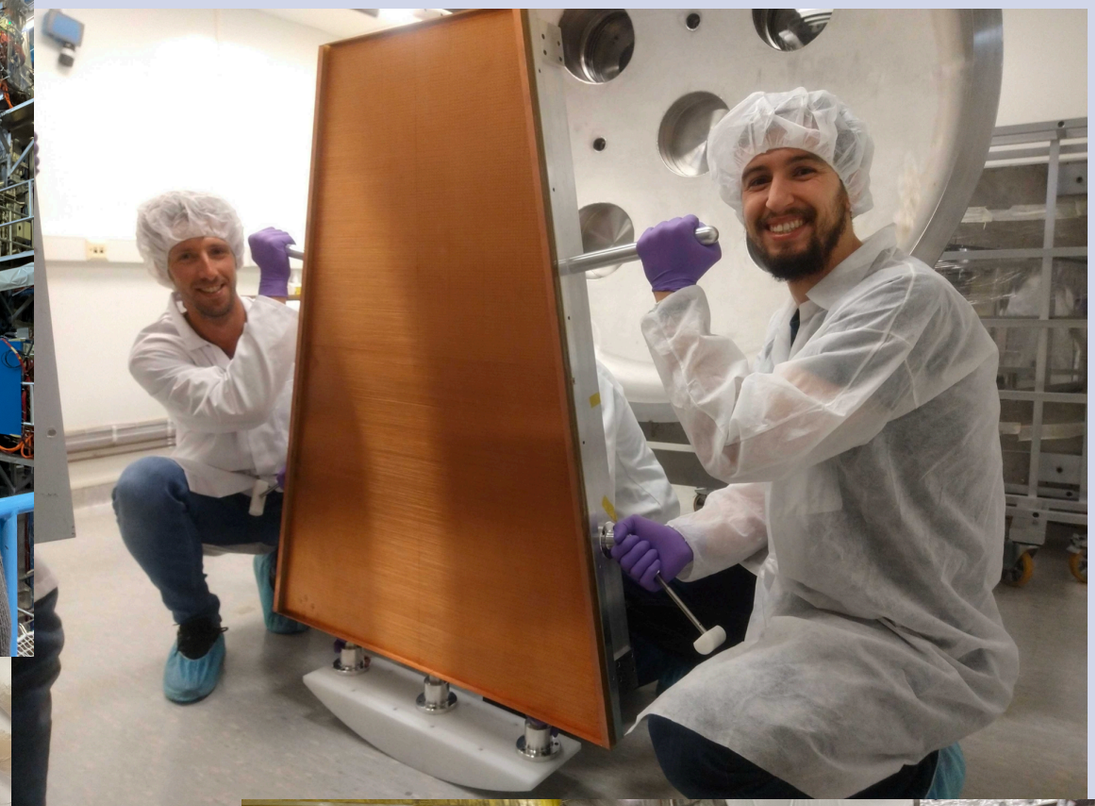
- **Python** and **TensorFlow 2.11** will be used throughout the course.
- Can access this software via **Binder** from the course Github.

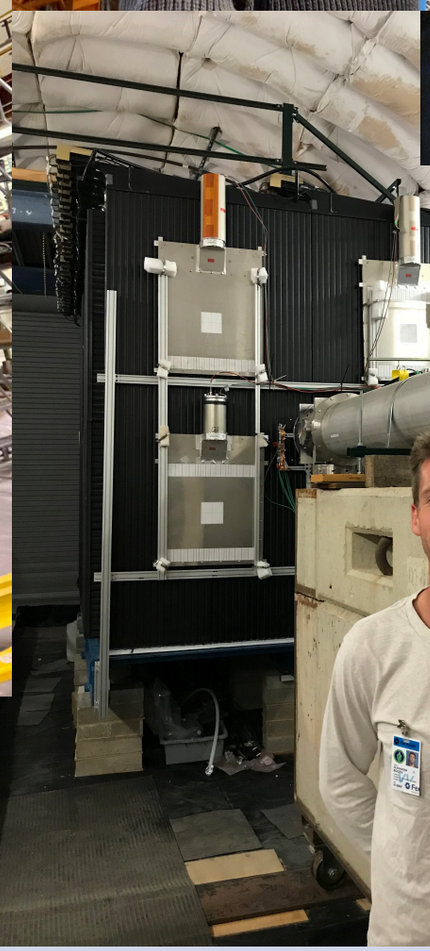
https://github.com/alexbooth92/workshop-UCDN_PracticalML.git

- Will interact with the software using **Jupyter Notebooks,** also in the Github.

I am a neutrino physicist!

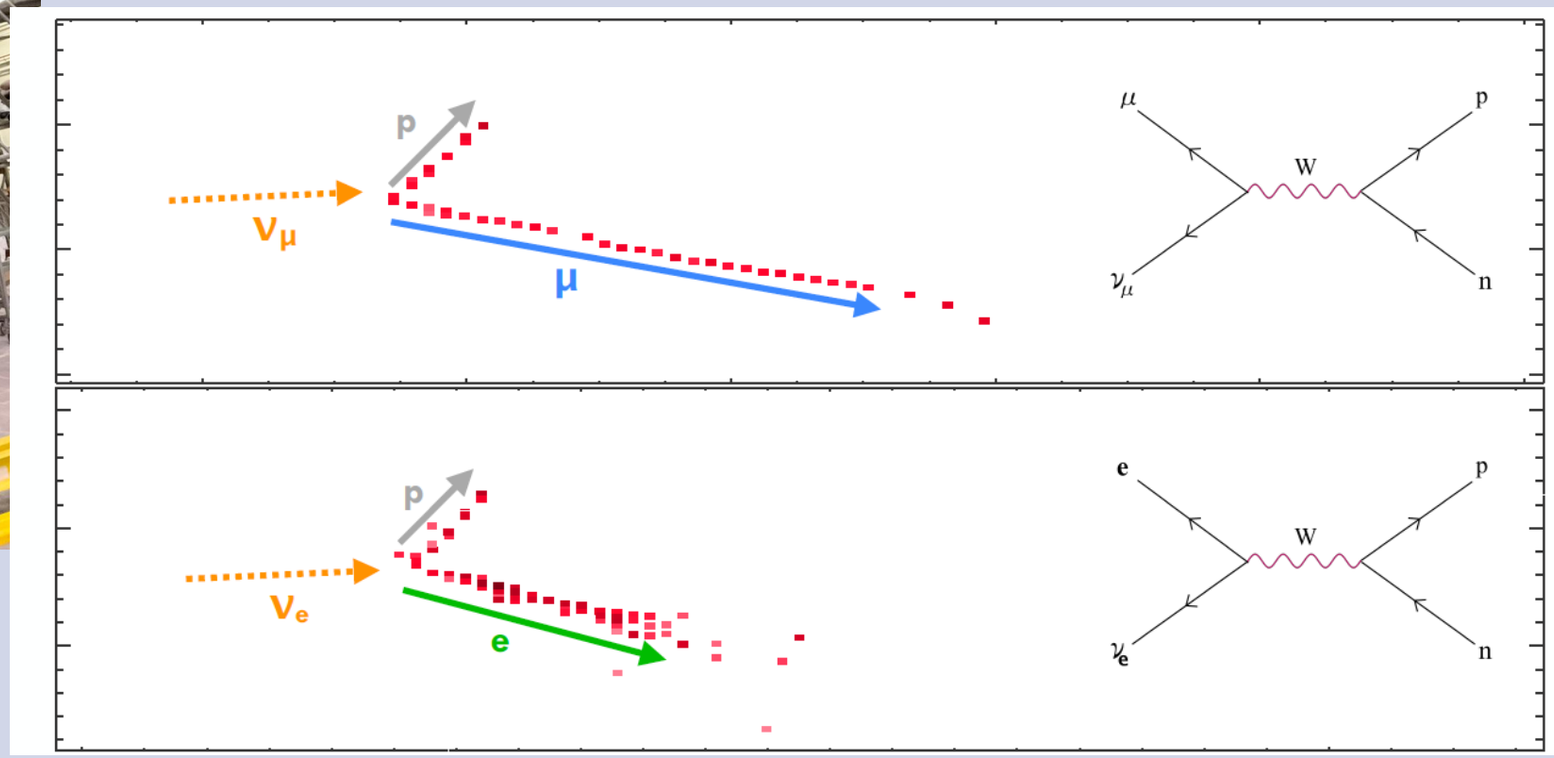






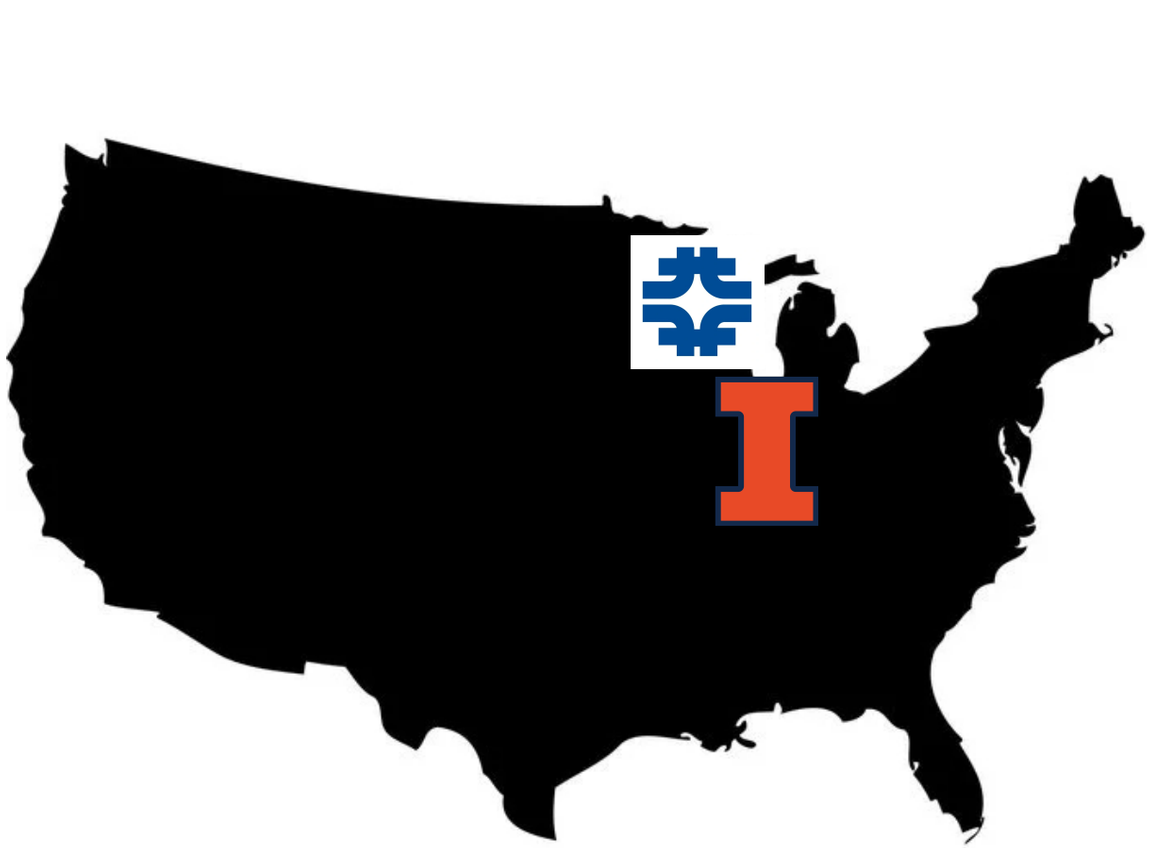
I am a neutrino physicist!

- Co-lead the Reconstruction & Machine Learning group in the NOvA experiment.
- Apply machine learning to the classification and energy estimation of neutrino interactions.

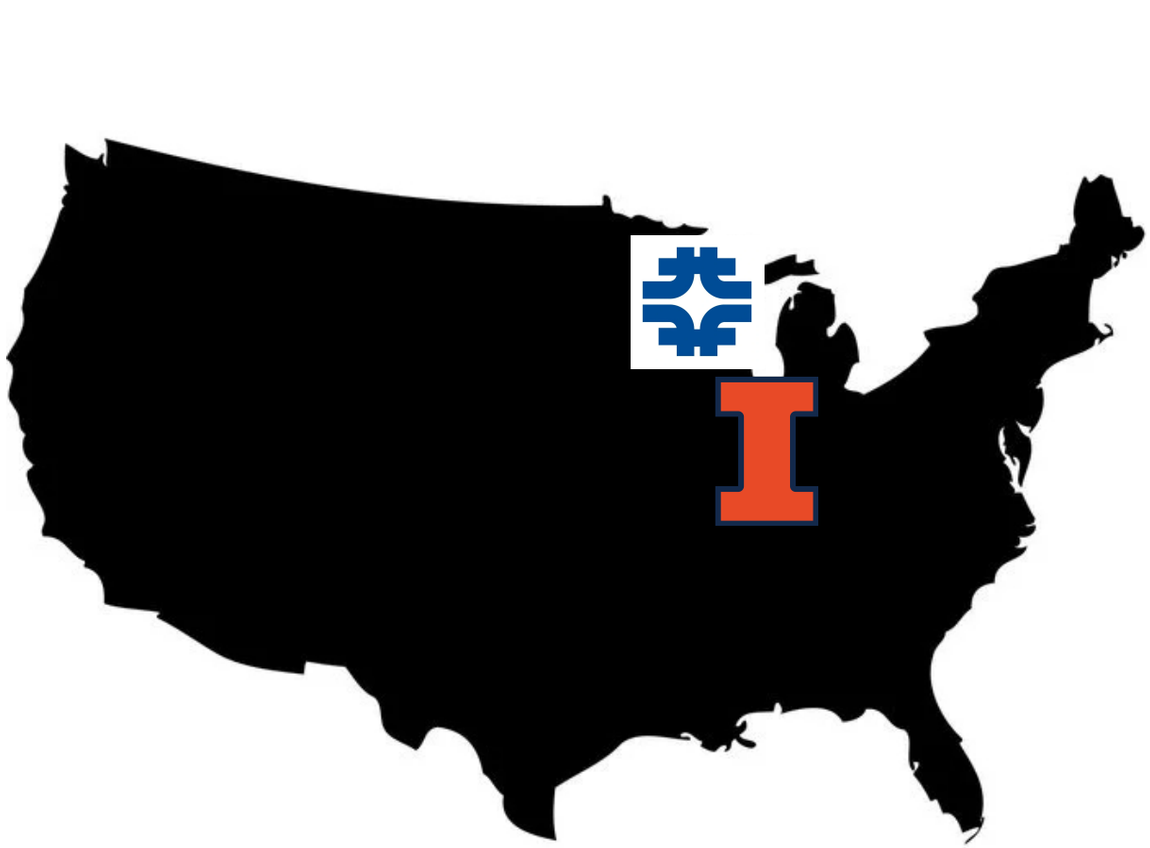


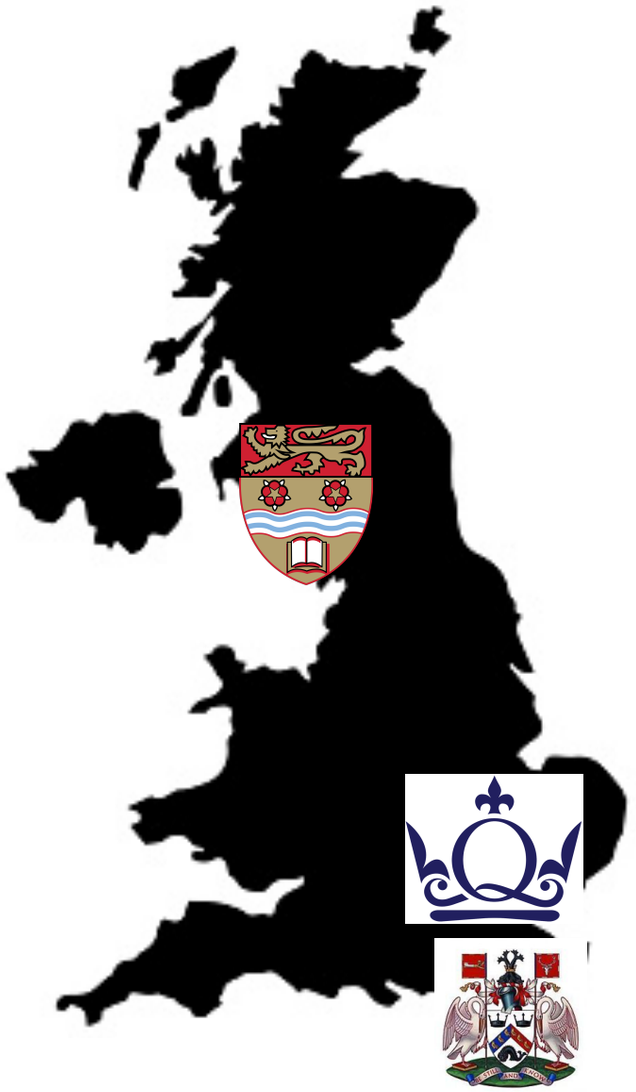
A Bit More About Me…

A Bit More About Me…



A Bit More About Me…





Let's find out about you!

PollEv.com/alexanderbooth769

What is Machine Learning?

What is Machine Learning?

"The science (and art) of programming computers so that they can learn from data".

A. Géron

Queen Mary
University of London

Can you think of examples?

What is Machine Learning?

"The science (and art) of programming computers so that they can learn from data".

A. Géron

Can you think of examples?

Image recognition

What is Machine Learning?

"The science (and art) of programming computers so that they can learn from data".

A. Géron

Queen Mary
University of London

Can you think of examples?

Image recognition

Speech recognition

What is Machine Learning?

"The science (and art) of programming computers so that they can learn from data".

A. Géron

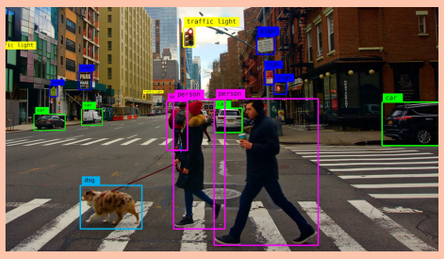
Queen Mary
University of London

Can you think of examples?

Image recognition

Speech recognition

Object detection



What is Machine Learning?

"The science (and art) of programming computers so that they can learn from data".

A. Géron

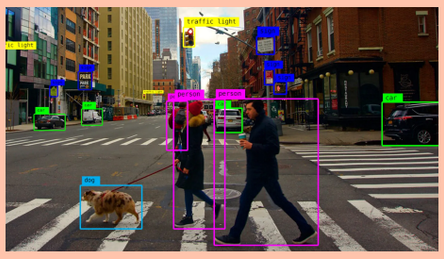
Queen Mary
University of London

Can you think of examples?

Image recognition

Speech recognition

Object detection



Art generation

What is Machine Learning?

"The science (and art) of programming computers so that they can learn from data".

A. Géron

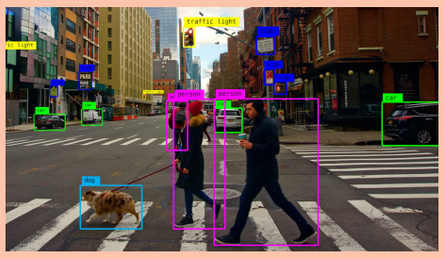
Queen Mary
University of London

Can you think of examples?

Image recognition

Speech recognition

Object detection



Art generation

Particle ID in fundamental particle physics

What is Machine Learning?

"The science (and art) of programming computers so that they can learn from data".

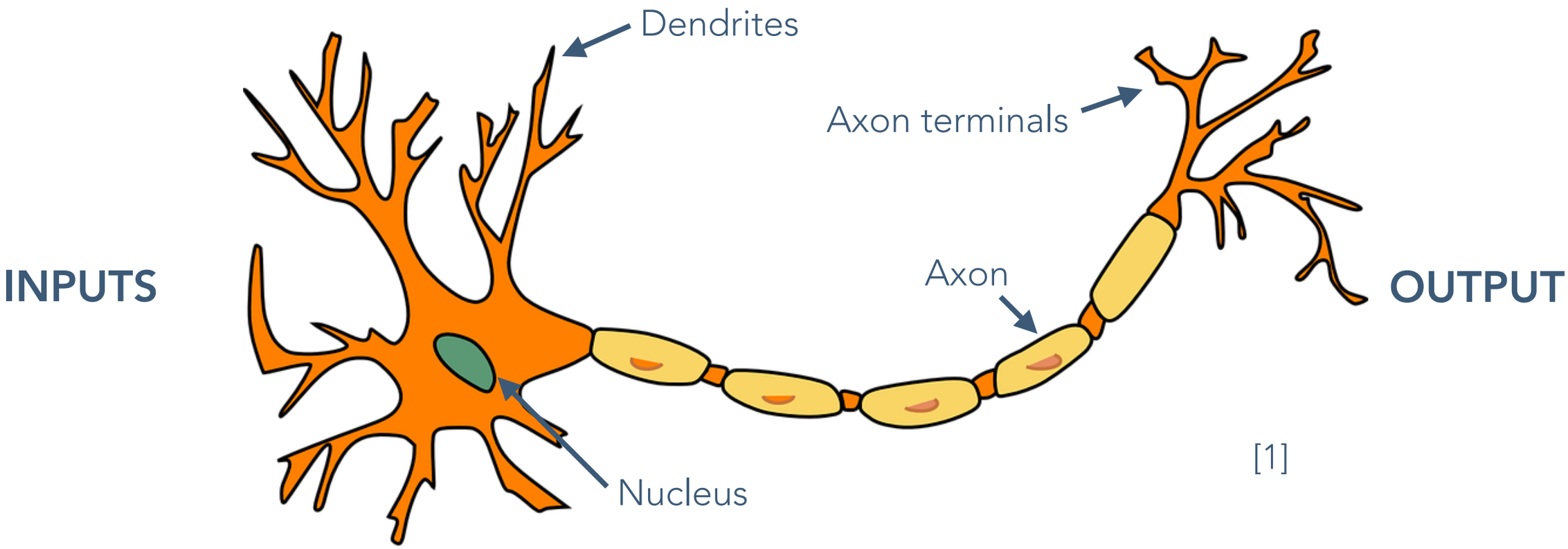
A. Géron

Queen Mary
University of London

Neurons

Neurons

Biological Neuron



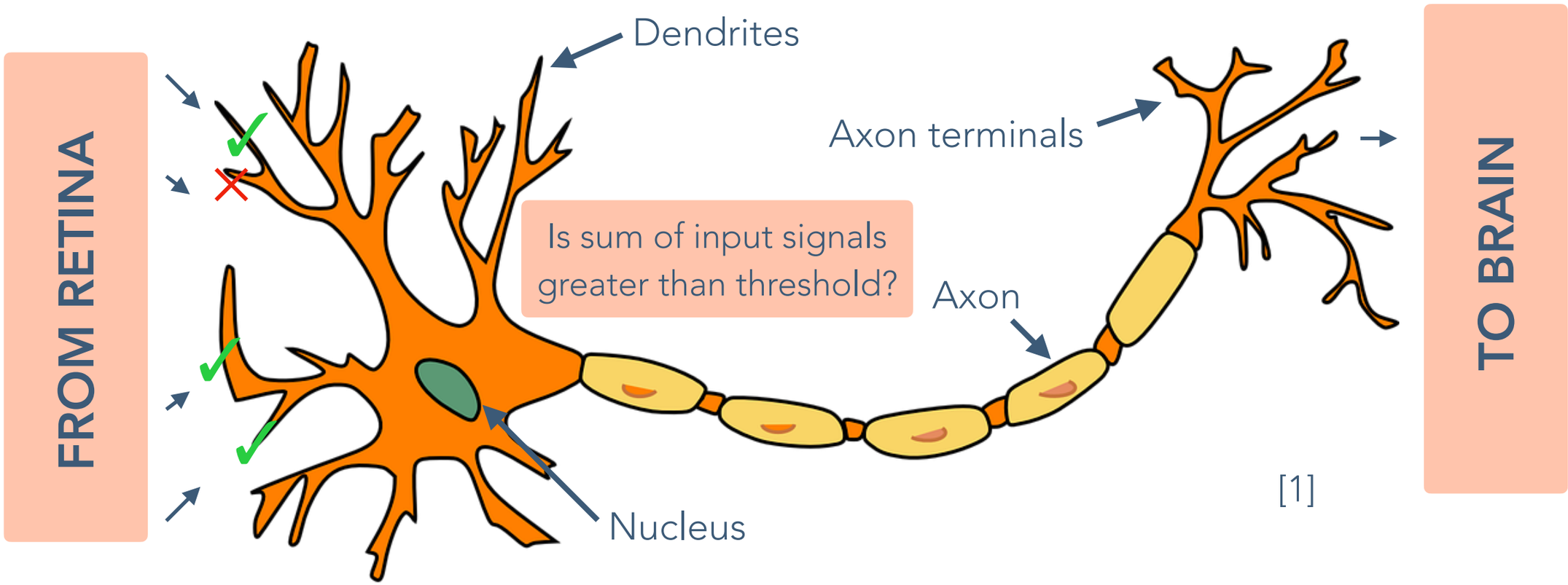
[1]

- Artificial neurons used in machine learning originate in trying to understand the operation of biological neurons.

- Responsible for processing and transmission of cellular signals.
  - Input signals arrive through dendrites - can scale signal importance.
  - Nucleus makes “decision” on if / how signals will be passed on.
  - Output signals transmitted via axon to axon terminals to be passed on.

Neurons

Biological Neuron



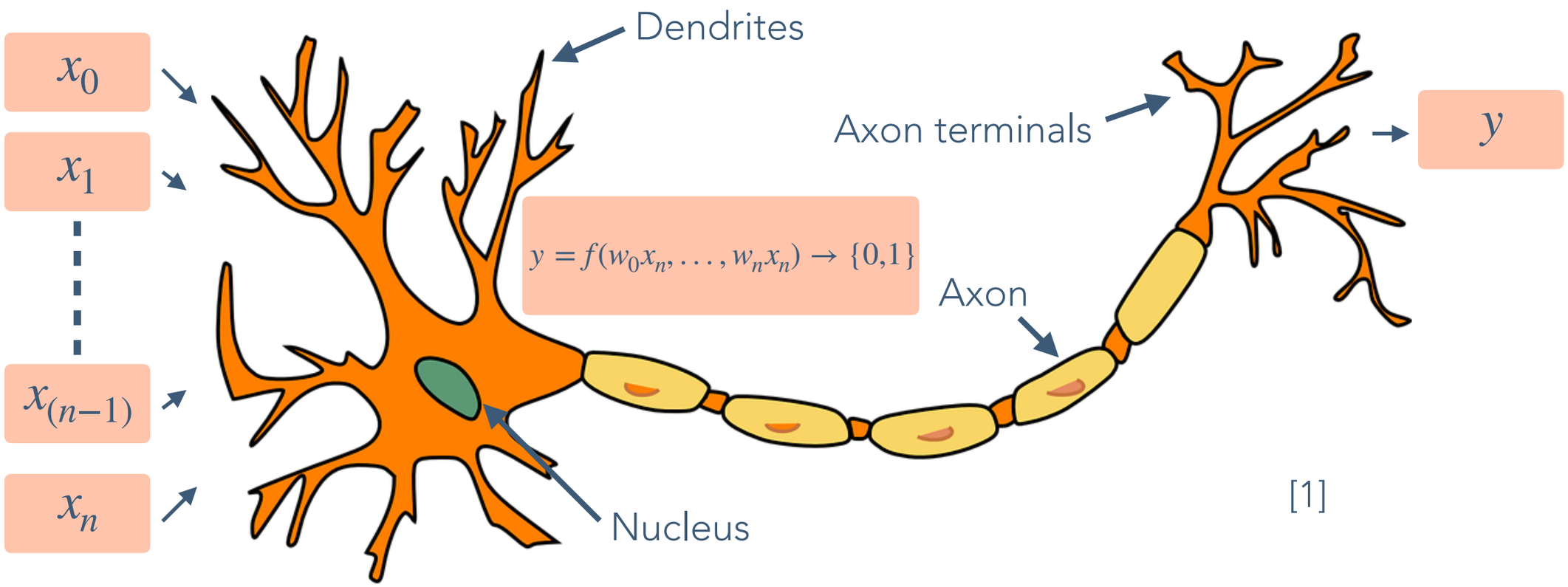
[1]

- Artificial neurons used in machine learning originate in trying to understand the operation of biological neurons.

- First artificial neuron, **"perceptron"** focussed on trying to understand how information from the retina is processed.
  - Gather input signals from several cells.
  - If total signal intensity is high enough "fire" response (all or nothing).

Neurons

Biological Neuron



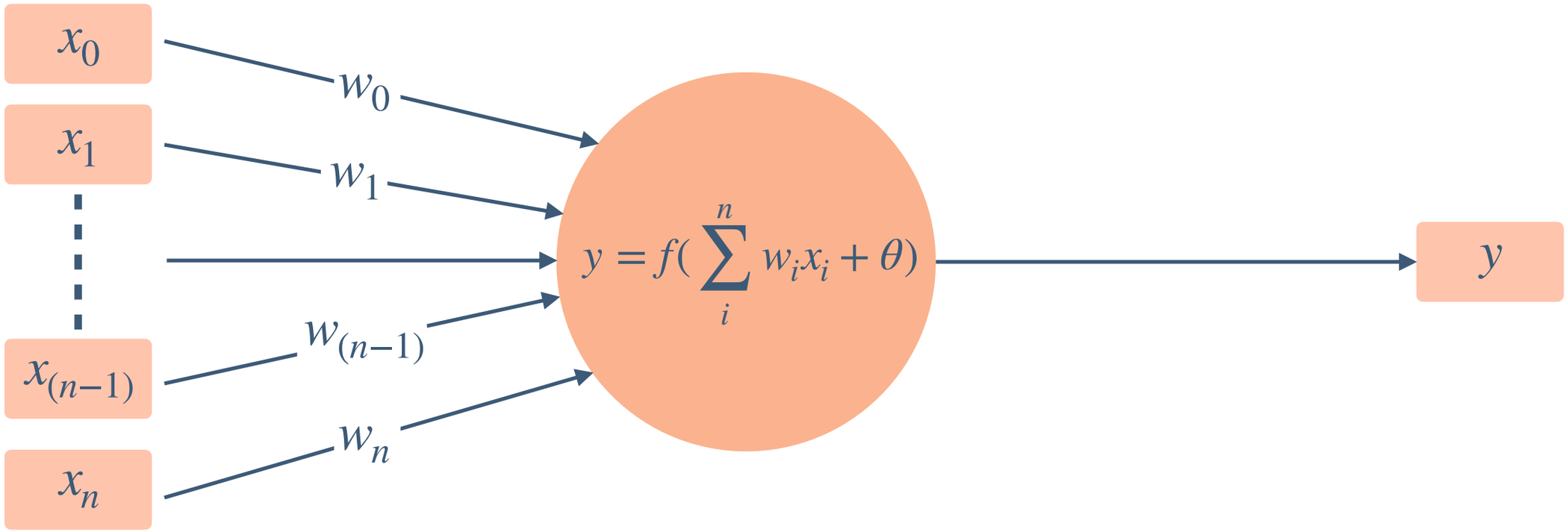
[1]

- Artificial neurons used in machine learning originate in trying to understand the operation of biological neurons.

- First artificial neuron, **"perceptron"** focussed on trying to understand how information from the retina is processed.
  - Gather input signals from several cells.
  - If total signal intensity is high enough "fire" response (all or nothing).

Neurons

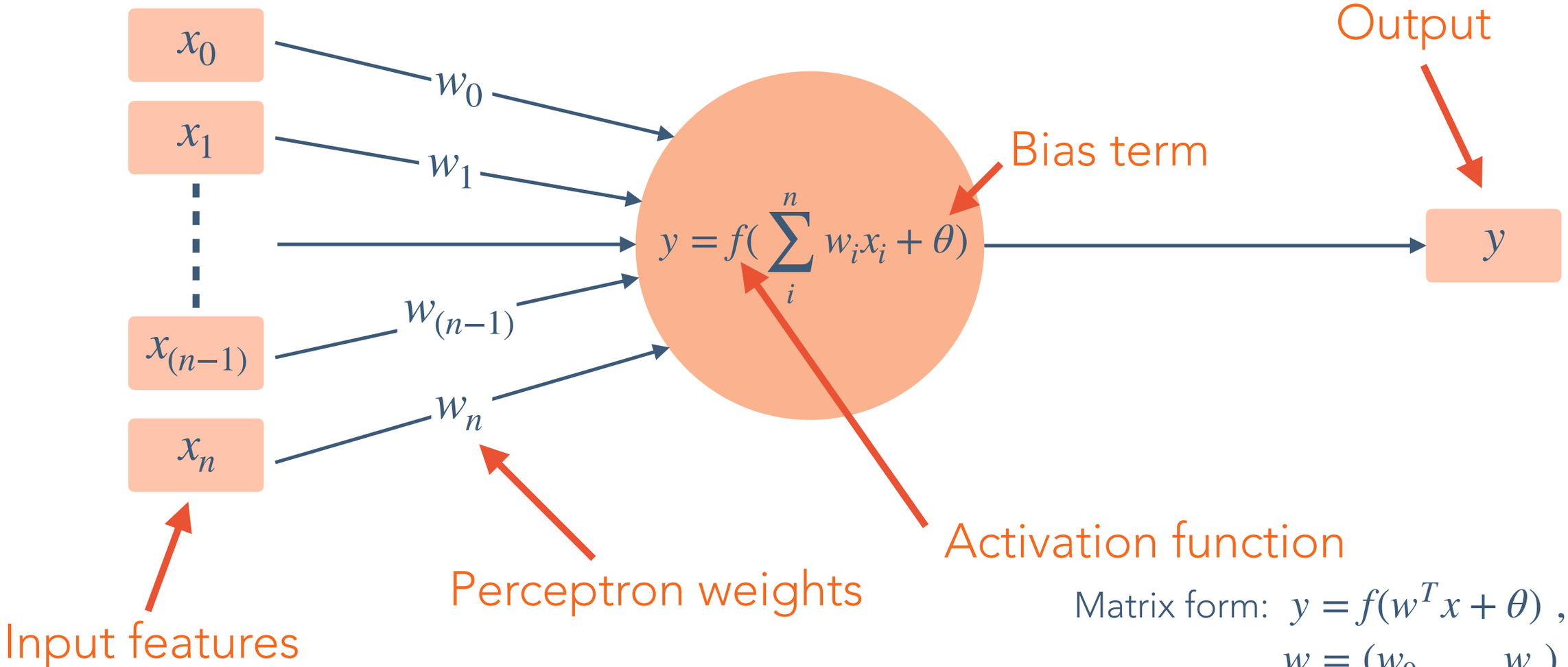
Artificial Neuron -> Perceptron (Node)



- Idea of perceptron can be extended to describe a more generalised picture.
  - Take some number, n, of **input features**.
  - Compute the sum of each of the features multiplied by some factor assigned to it (indicating the importance of that information).
  - Compare the sum against some reference threshold.
  - Perceptron is “active” (y=1) if above threshold, not active if below (y=0).

Neurons

Artificial Neuron -> Perceptron (node)

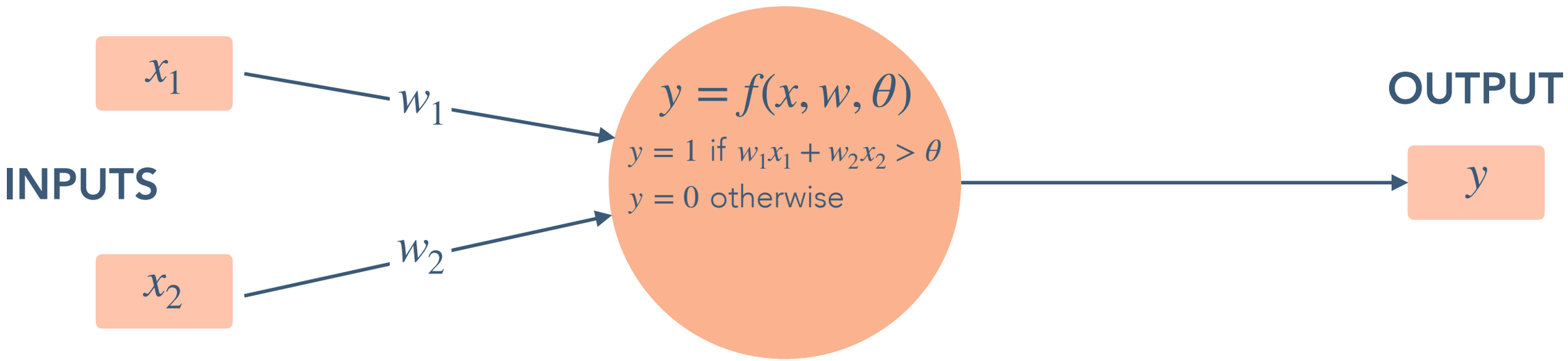


Matrix form: $y = f(w^T x + \theta)$,

$w = (w_0, \ldots, w_n)$,

$x = (x_0, \ldots, x_n)$

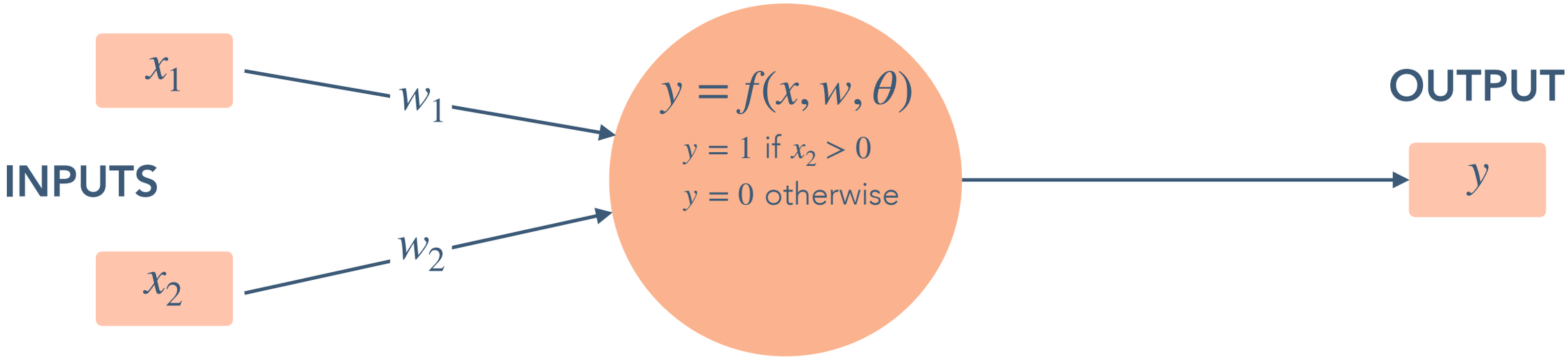
Perceptrons: Illustrative Example



- Consider a measurement of two quantities x_1 and x_2 and a perceptron with **binary activation function**.

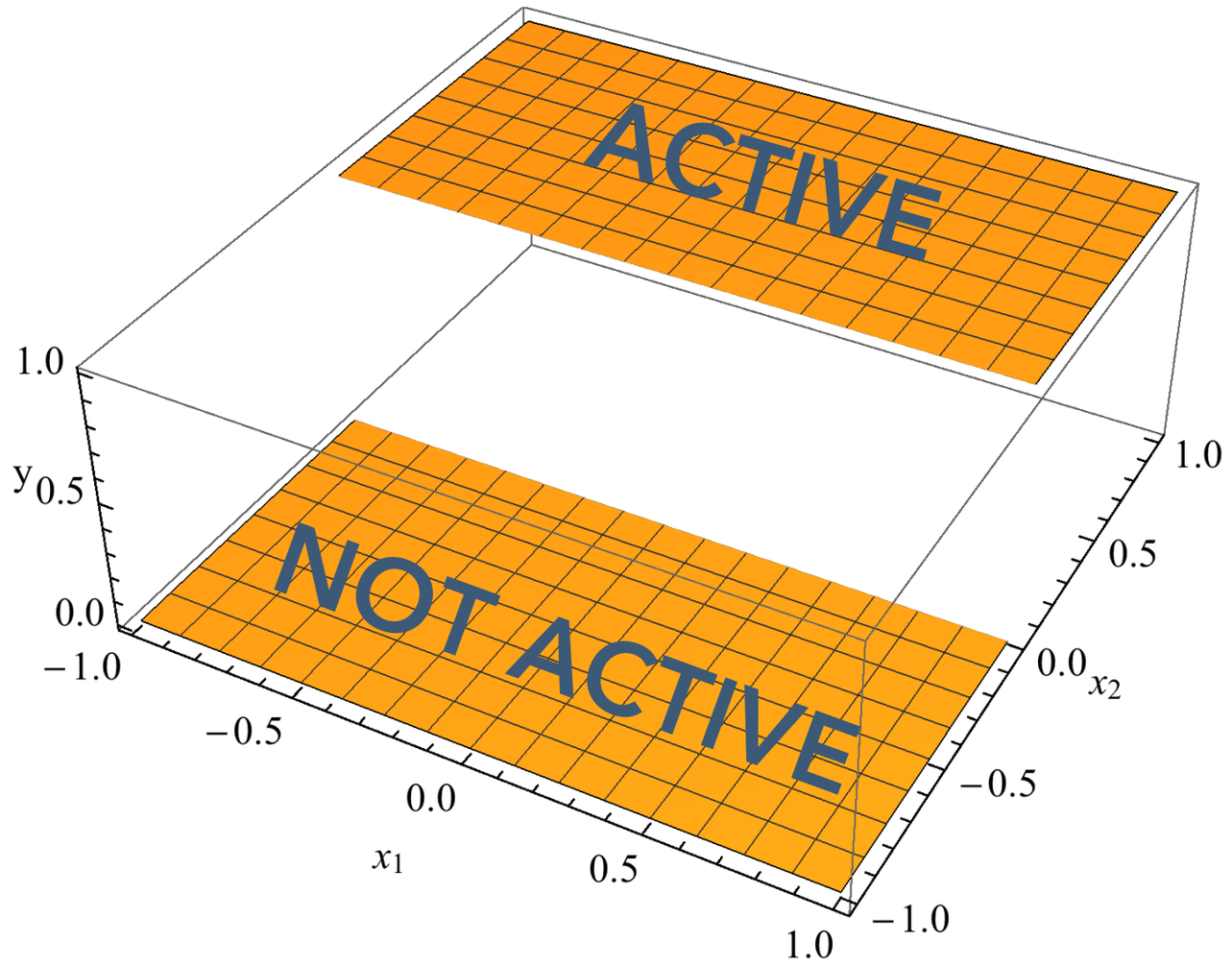
- Based on these measurements lets determine if the perceptron is active / to give an output (y = 1) or not (y = 0) **for a given choice of weights**.

Perceptrons: Illustrative Example

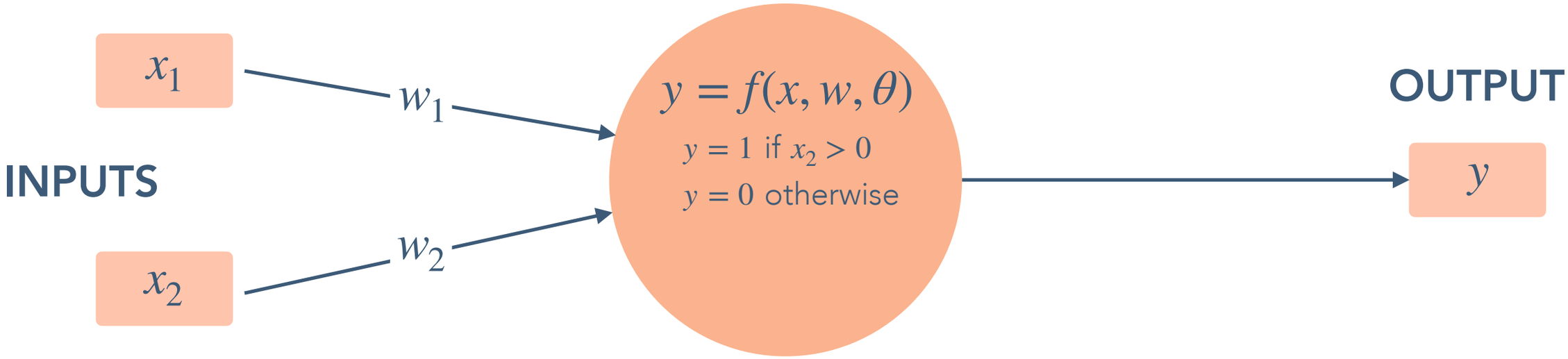


Weight choice 1: $w_1 = 0, w_2 = 1, \theta = 0$

Decision based on x_2 only.



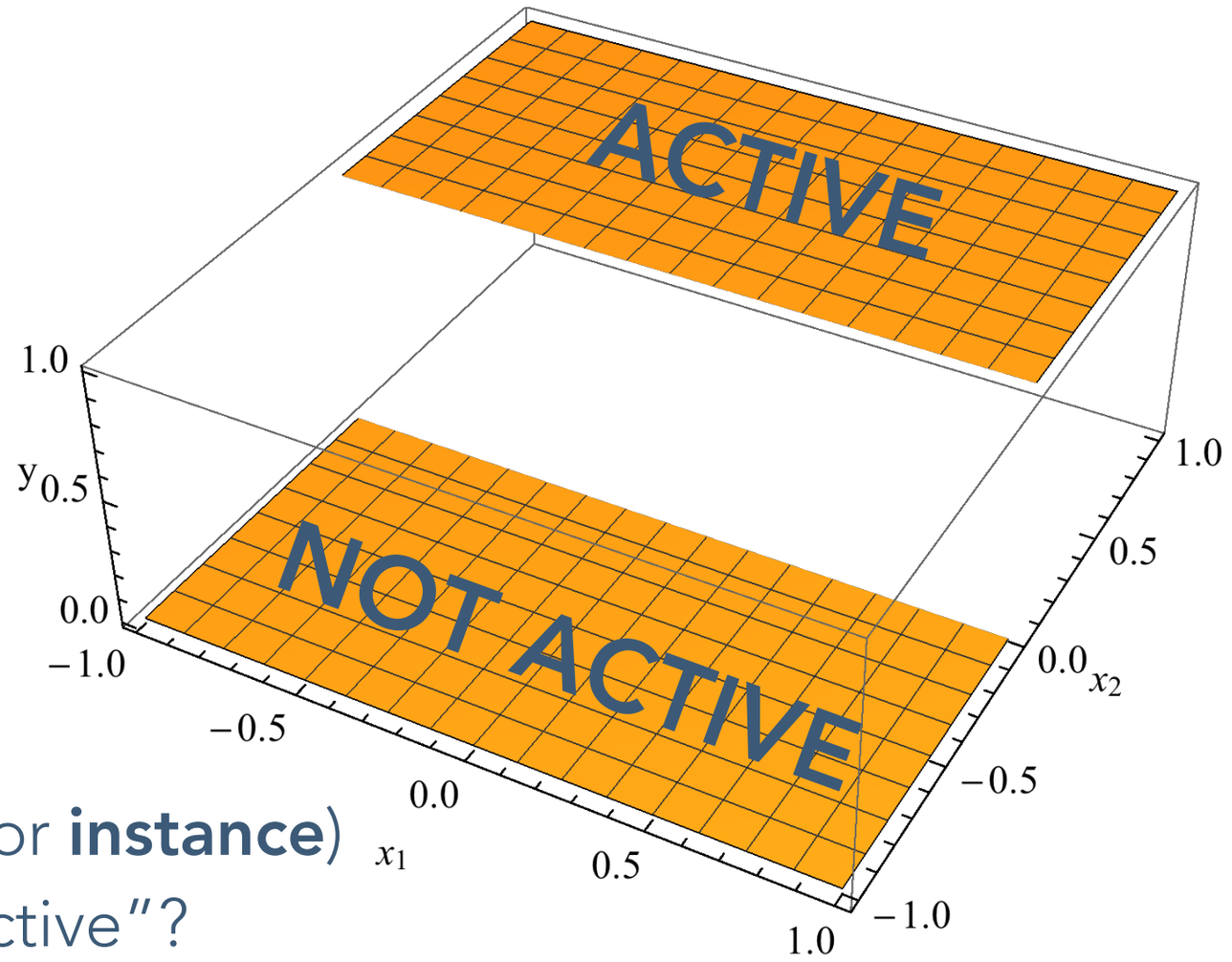
Perceptrons: Illustrative Example



Weight choice 1: $w_1 = 0,\ w_2 = 1,\ \theta = 0$

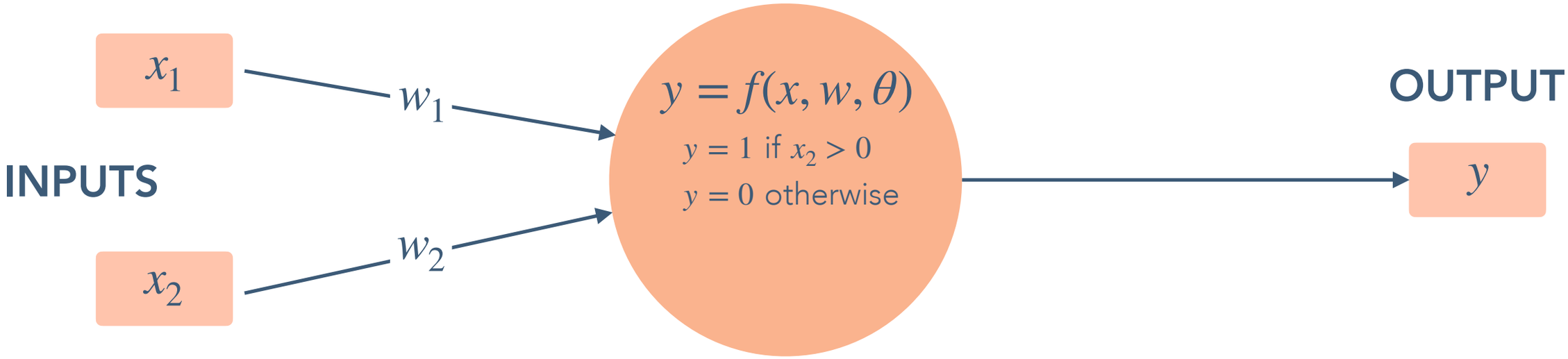
Input set 1: $x_1 = 0.2,\ x_2 = -0.5$

Input set 2: $x_1 = -0.6,\ x_2 = 0.8$



For which input set (or **instance**) is the perceptron "active"?

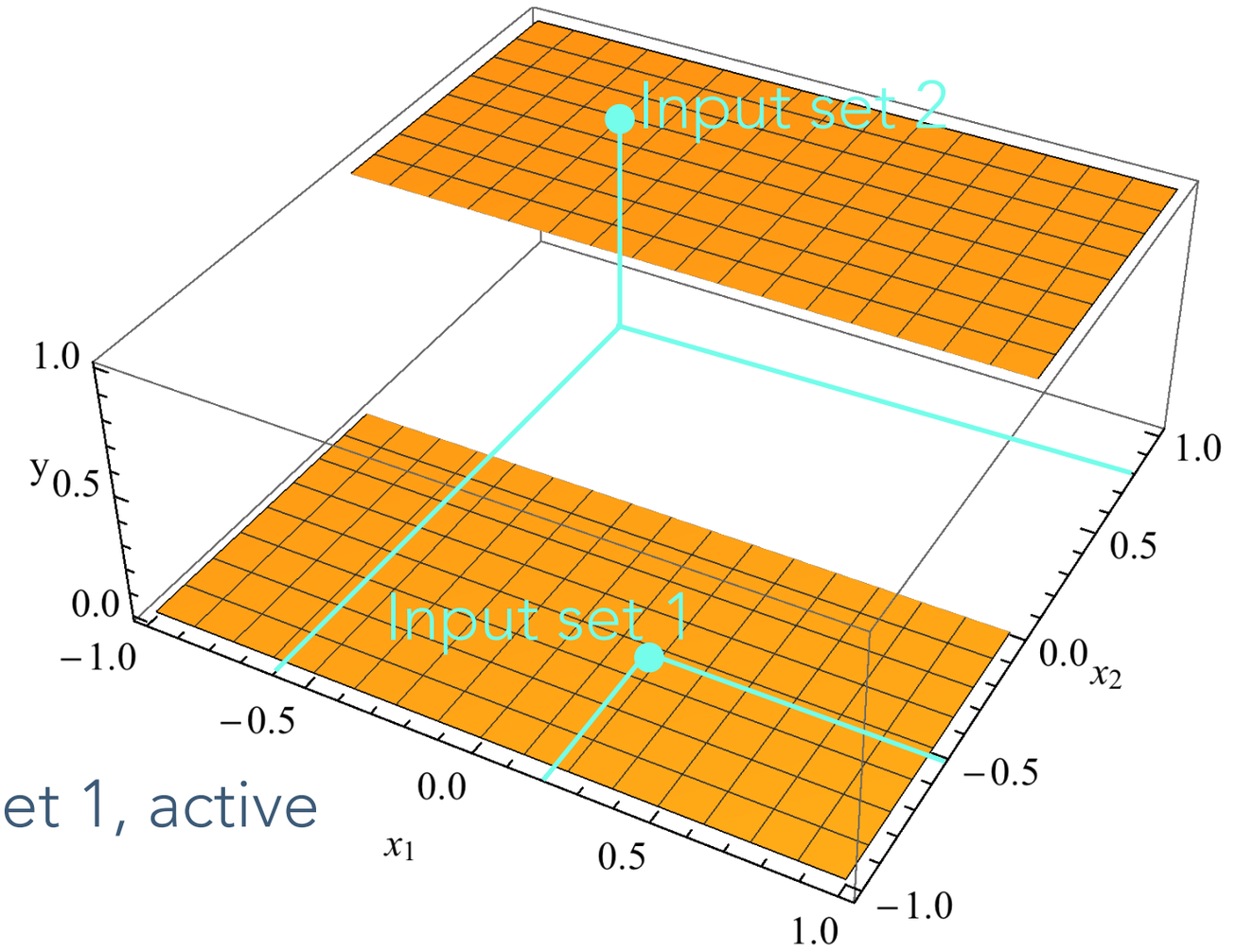
Perceptrons: Illustrative Example



Weight choice 1: $w_1 = 0,\ w_2 = 1,\ \theta = 0$

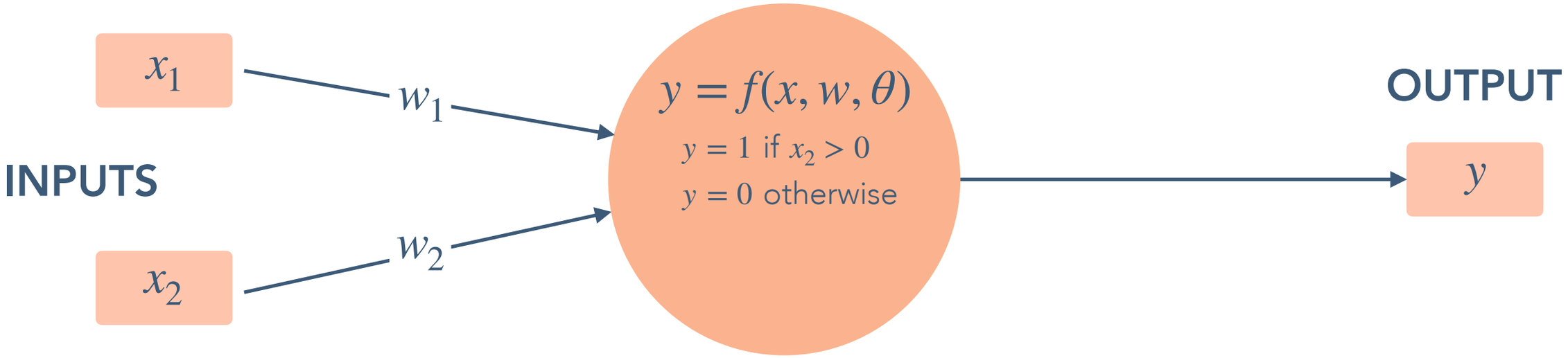
Input set 1: $x_1 = 0.2,\ x_2 = -0.5$

Input set 2: $x_1 = -0.6,\ x_2 = 0.8$



Not active for set 1, active for set 2

Perceptrons: Illustrative Example

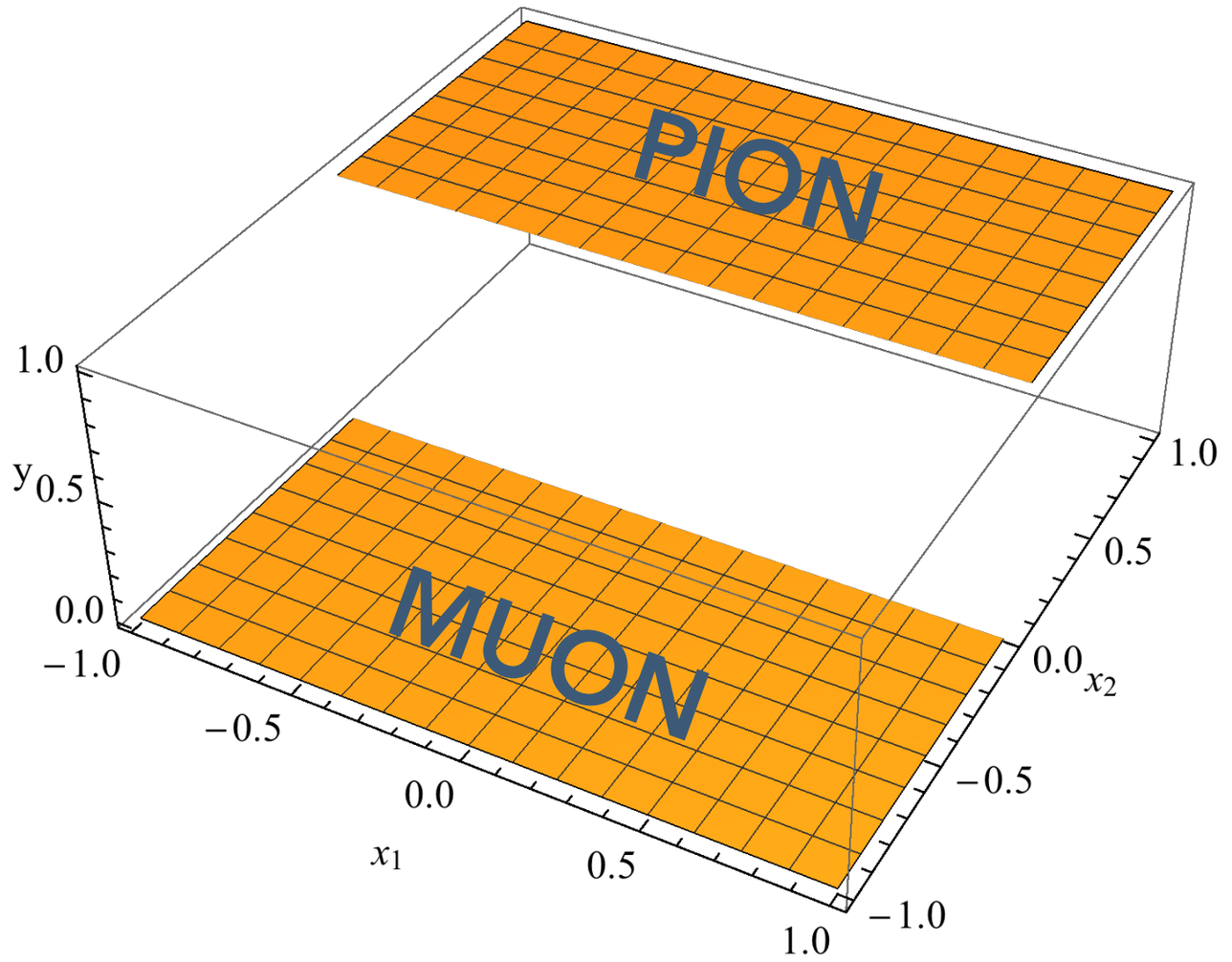


Weight choice 1: $w_1 = 0, w_2 = 1, \theta = 0$

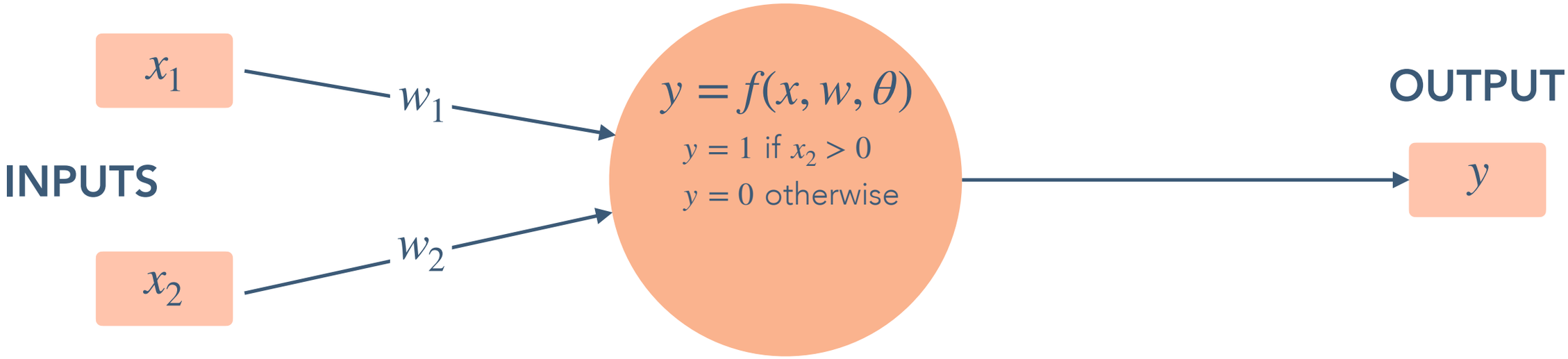
In particle physics we often use machine learning to identify different types of particle, e.g **pion** vs. **muon** -> **classification**.

Particle candidate 1: $x_1 = 0.2, x_2 = -0.5$

Particle candidate 2: $x_1 = -0.6, x_2 = 0.8$



Perceptrons: Illustrative Example



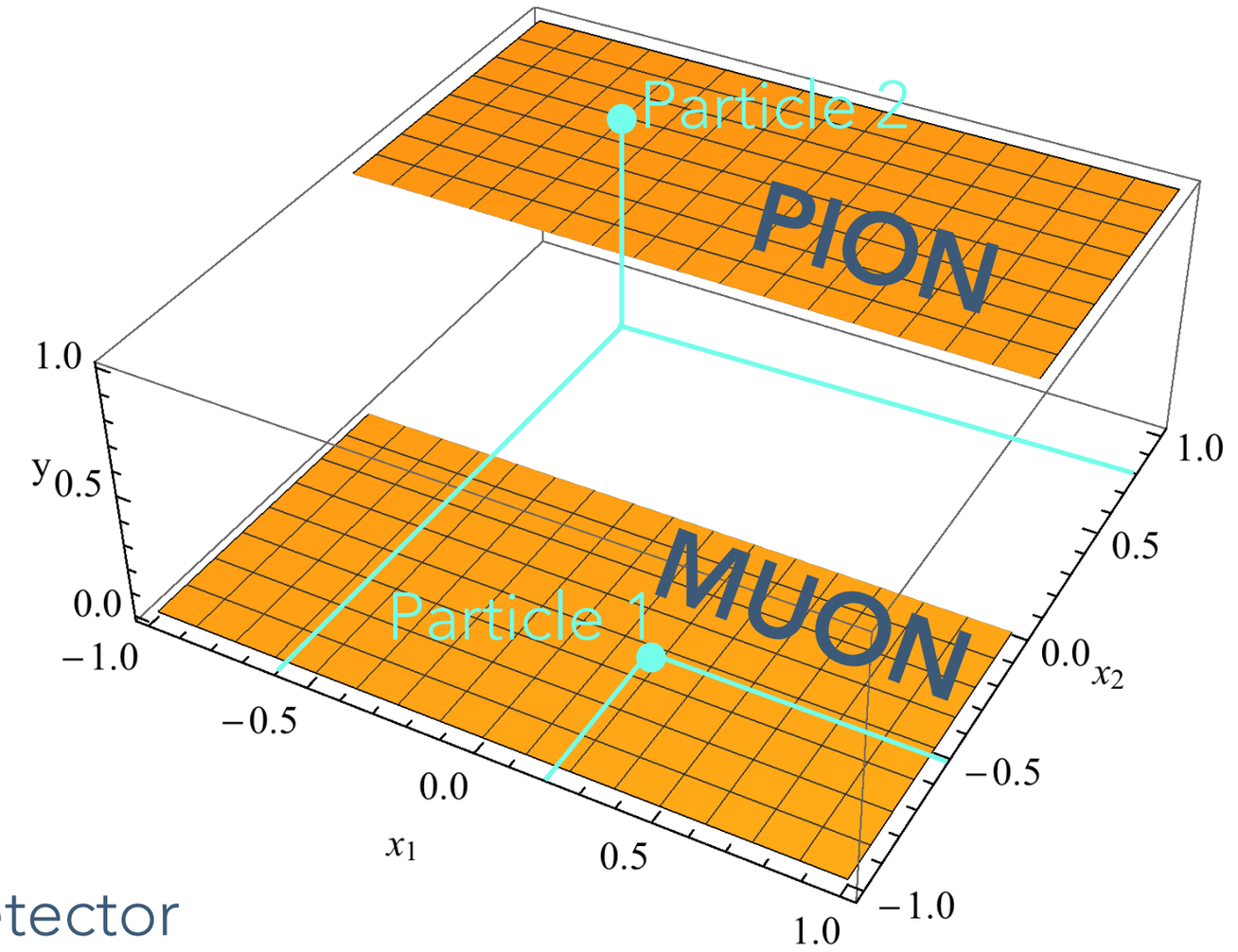
Weight choice 1: $w_1 = 0, w_2 = 1, \theta = 0$

In particle physics we often use machine learning to identify different types of particle, e.g **pion** vs. **muon**.

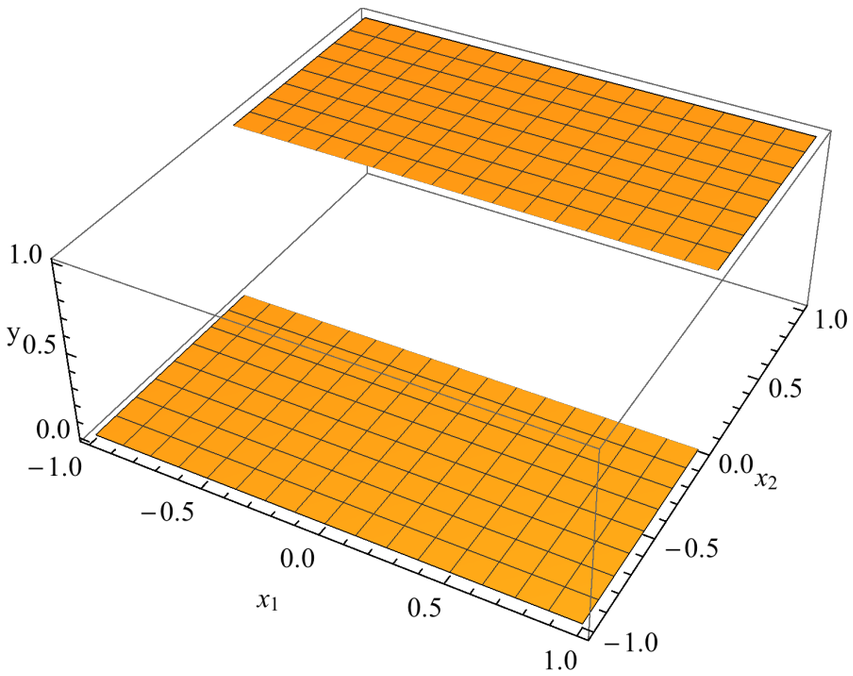
Particle candidate 1: $x_1 = 0.2, x_2 = -0.5$

Particle candidate 2: $x_1 = -0.6, x_2 = 0.8$

x_1 ~ momentum x_2 ~ angle in detector



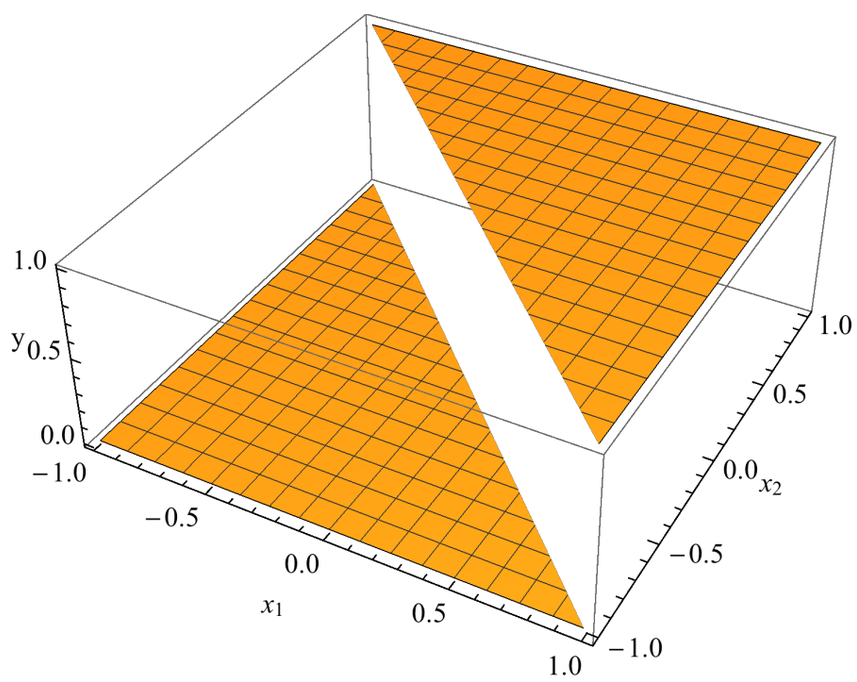
Perceptrons: Illustrative Example



Weight choice 1:

$w_1 = 0,\ w_2 = 1,\ \theta = 0$

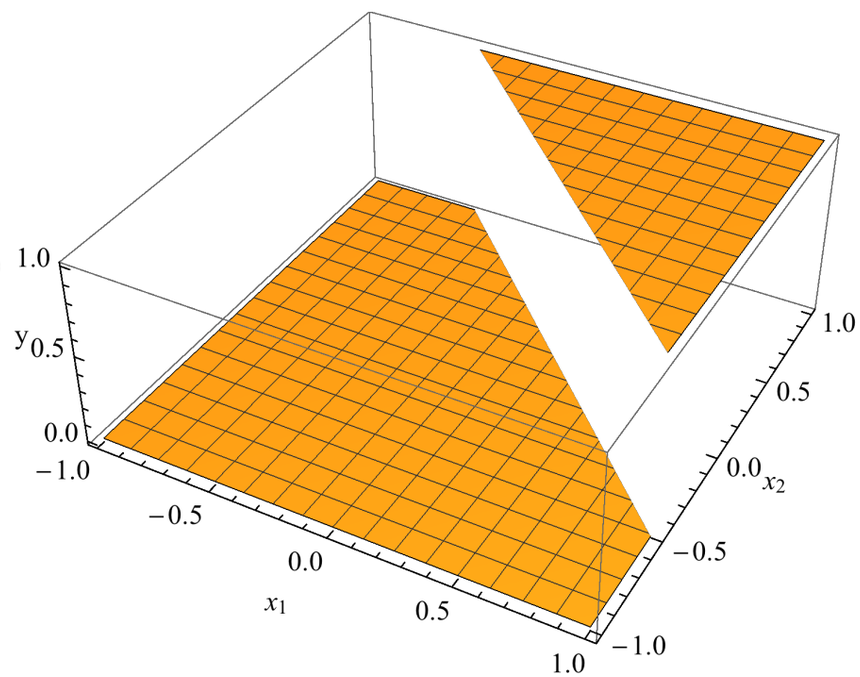
Decision based on x_2 only.



Weight choice 2:

$w_1 = 1,\ w_2 = 1,\ \theta = 0$

Rotate decision plane .



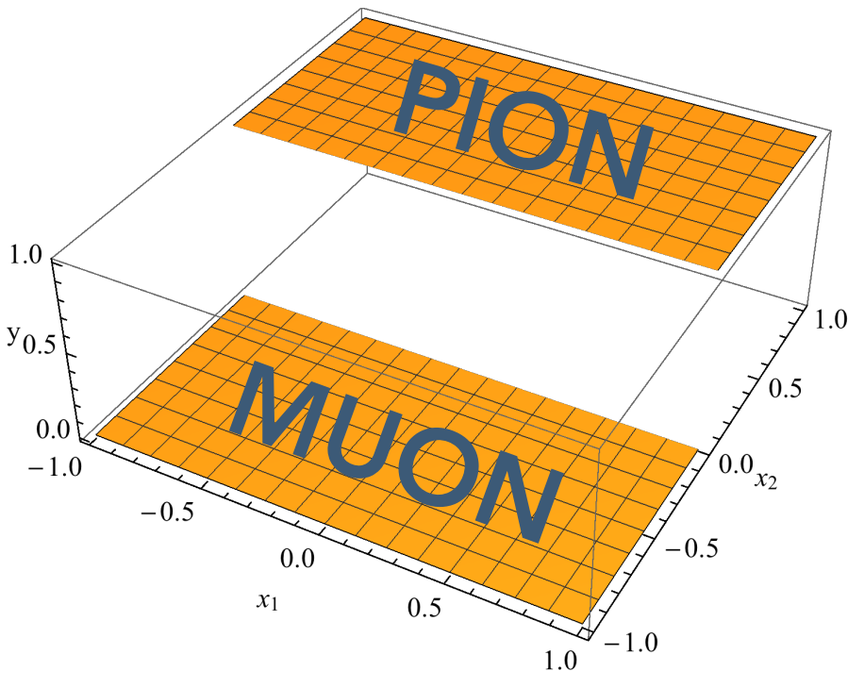
Weight choice 2:

$w_1 = 1,\ w_2 = 1,\ \theta = 0.5$

Shift decision plane away from origin.

Choice of weights & bias changes the response of the perceptron.

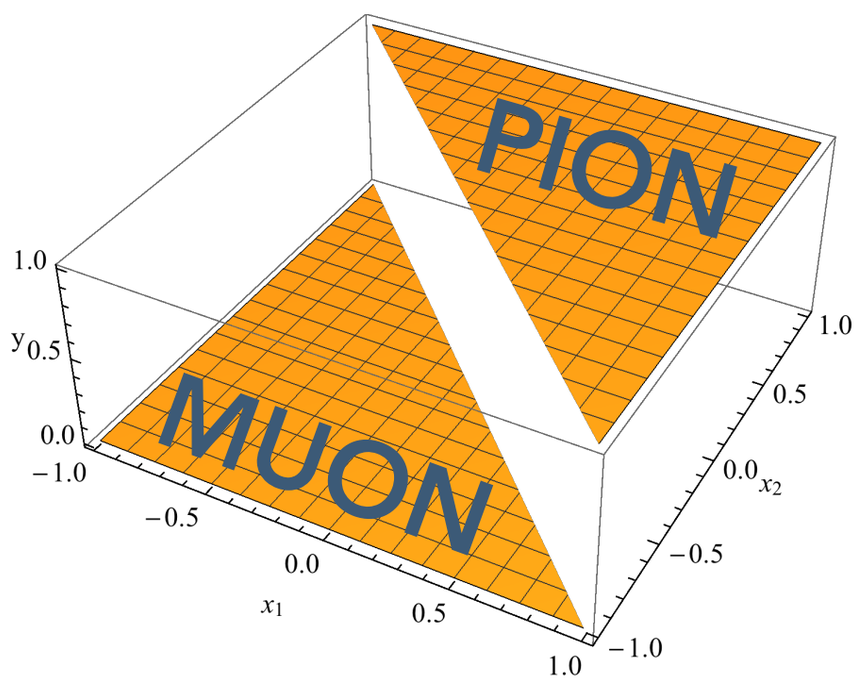
Perceptrons: Illustrative Example



Weight choice 1:

$w_1 = 0, w_2 = 1, \theta = 0$

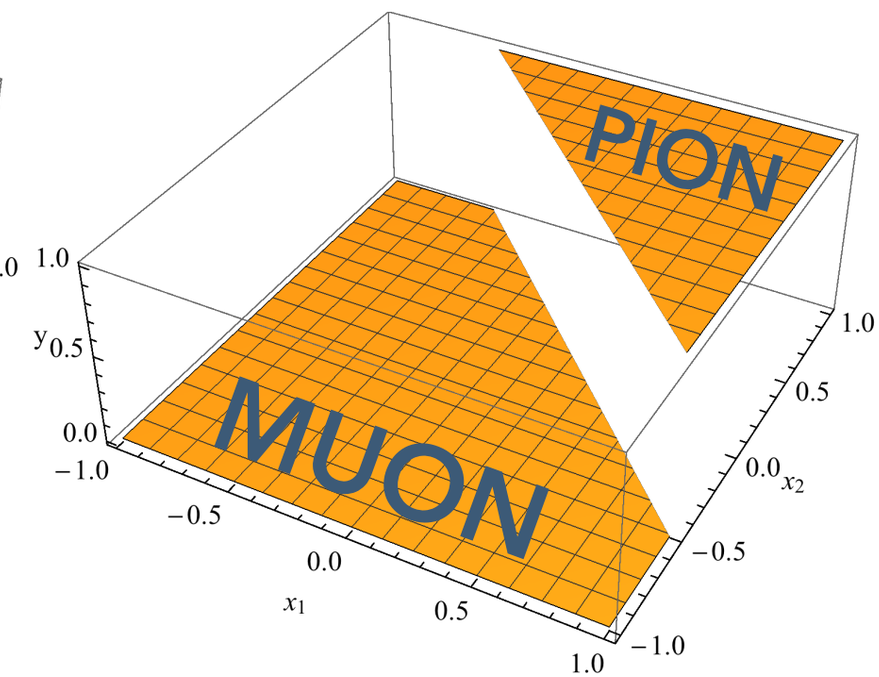
Decision based on x_2 only.



Weight choice 2:

$w_1 = 1, w_2 = 1, \theta = 0$

Rotate decision plane .



Weight choice 2:

$w_1 = 1, w_2 = 1, \theta = 0.5$

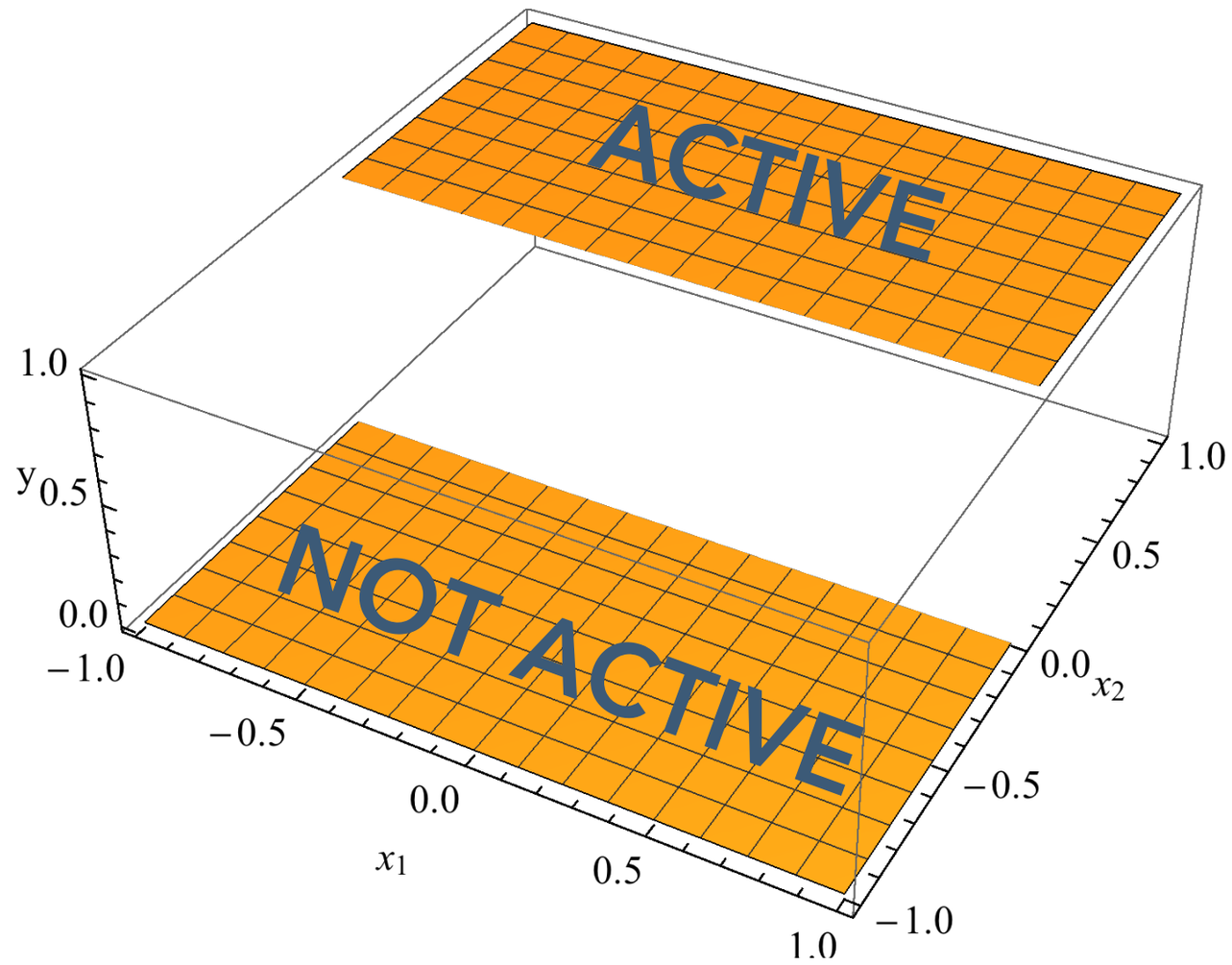
Shift decision plane away from origin.

Optimisation of weights and bias (e.g to give you the best particle classification) is *"training"*.

More Activation Functions

- So far have used **binary activation function**.
  - Gives "all or nothing" response.

- Can be useful to provide an output that is continuous between two extremes.
  - That requires different forms of activation function.



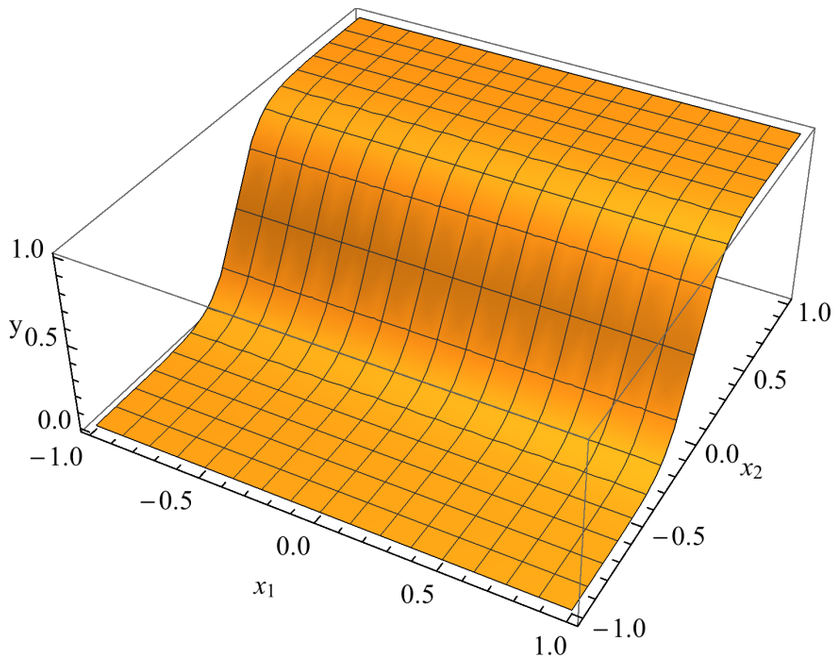
- TensorFlow has the following activation functions (`tf.nn.ACTIVATIONFUNCTION`)
  - relu (covered here)
  - leaky_relu (covered here)
  - relu6
  - crelu
  - elu
  - selu
  - softplus
  - softsign
  - dropout
  - bias_add
  - sigmoid (covered here)
  - tanh

Activation Functions: Logistic (Sigmoid)

A common activation function found in neural networks.

$$y = \frac{1}{1 + e^{w^T x + \theta}}$$

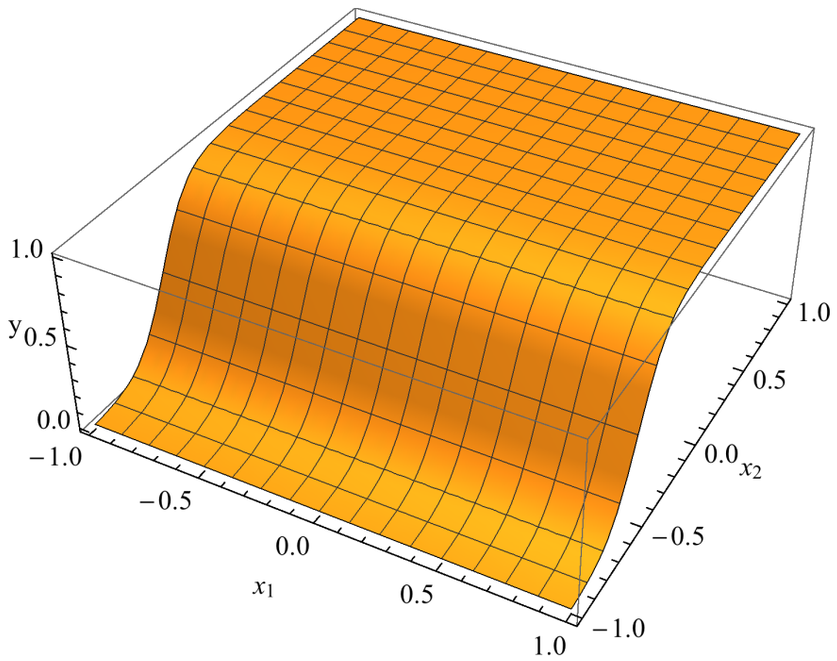
$$= \frac{1}{1 + e^{(w_1 x_1 + w_2 x_2 + \theta)}}$$



$w_1 = 0$

$w_2 = 10$

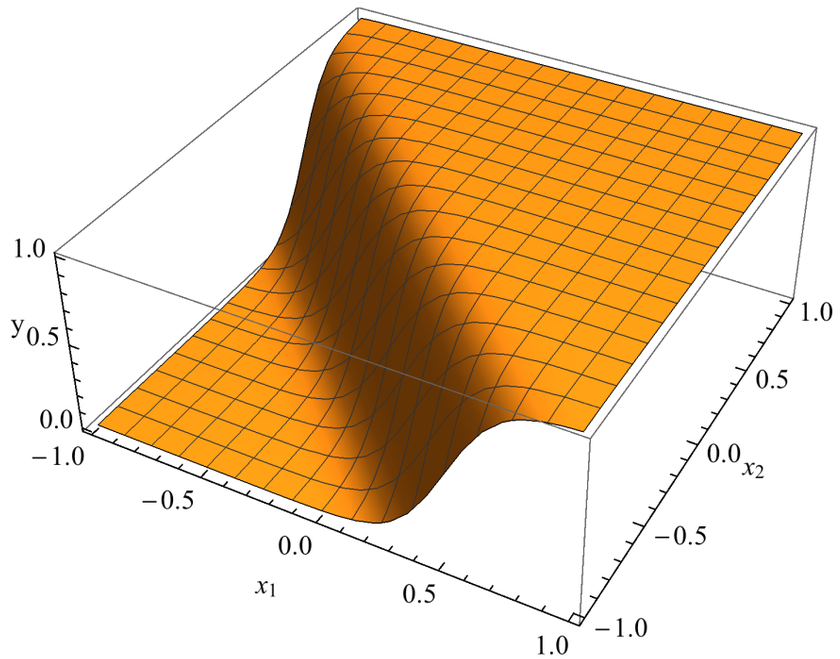
$\theta = 0$



$w_1 = 0$

$w_2 = 10$

$\theta = -5$



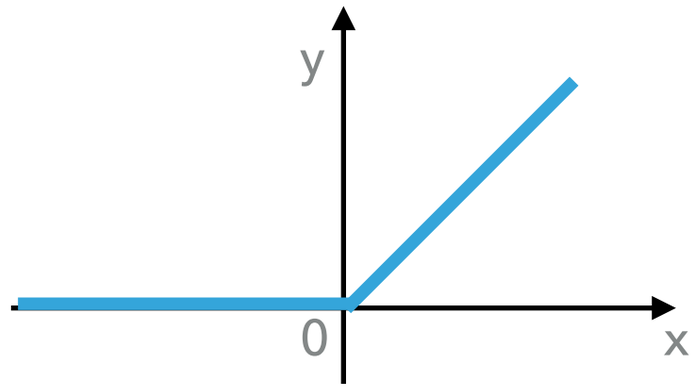
$w_1 = 10$

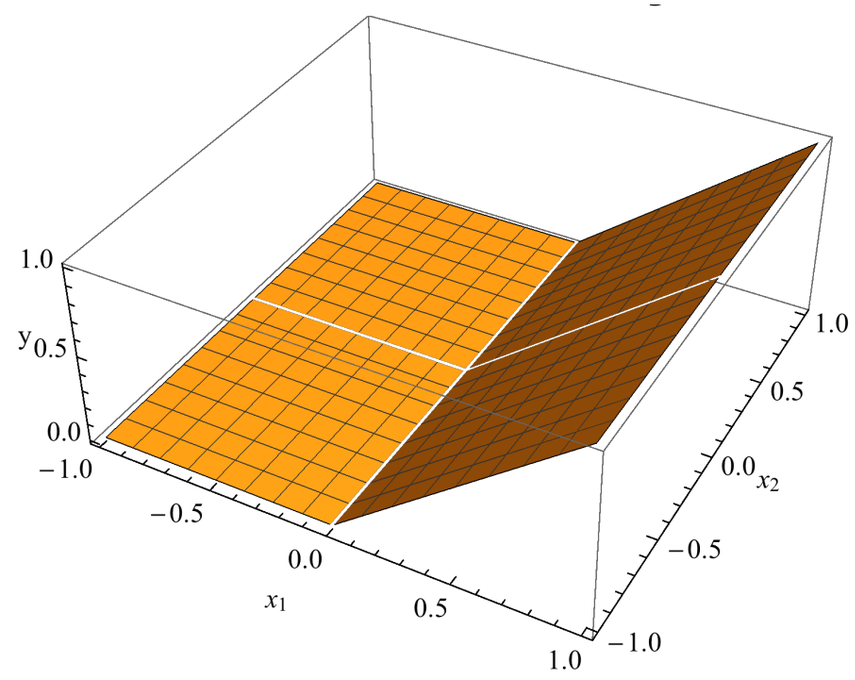
$w_2 = 10$

$\theta = -5$

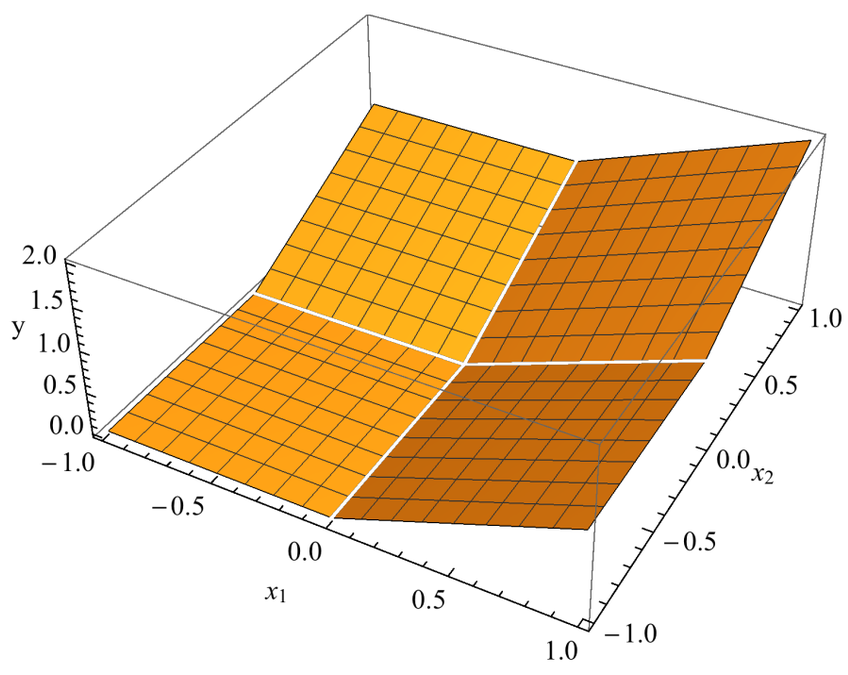
Activation Functions: ReLU

- The **Re**ctified **L**inear **U**nit activation function is commonly used in **convolutional neural networks (CNNs)**.
  - If $x < 0$, $y = 0$
  - Otherwise $y = x$

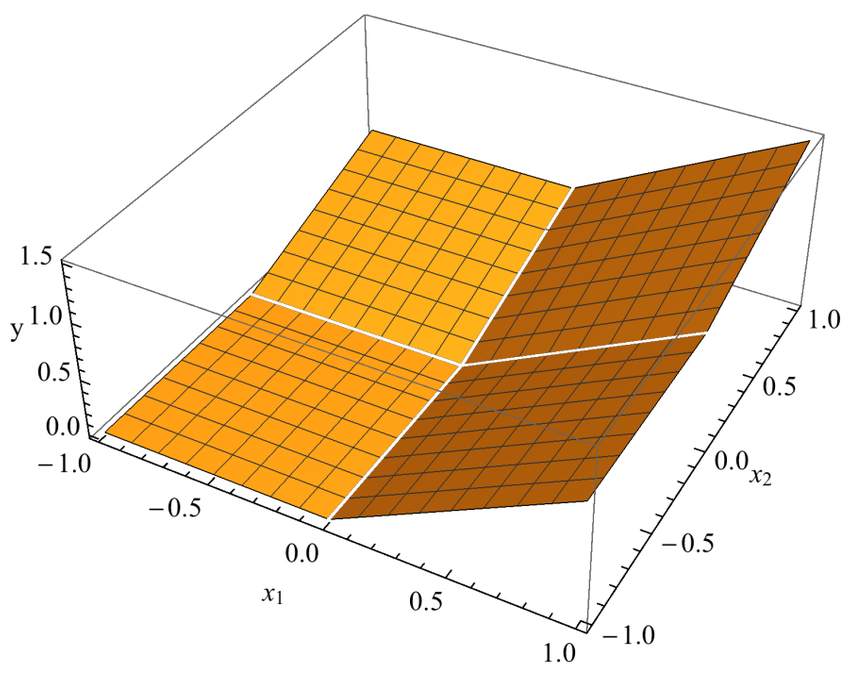




$w_1 = 1, w_2 = 0$



$w_1 = 1, w_2 = 1$

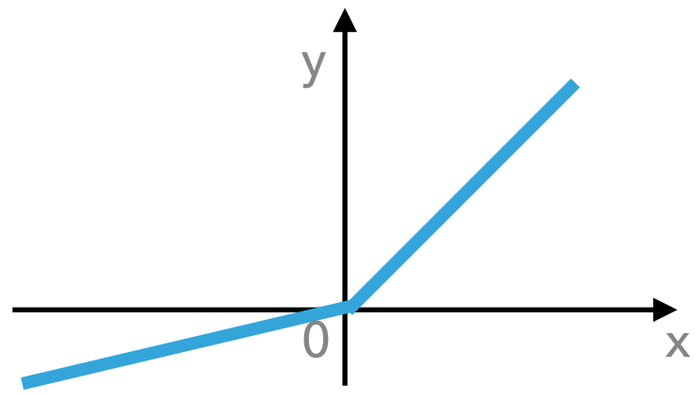


$w_1 = 1, w_2 = 0.5$

Importance of features in the perceptron still depends on weights.

Activation Functions: ReLU Family

- **Leaky ReLU** is a slightly modified version of **ReLU.** For some small constant a:
  - If $x < 0$, $y = ax$
  - Otherwise $y = x$



- **P**arametrised **ReLU**, a is a learnable parameter.

$$f(x) = \max(0, x) + a \min(0, x)$$

Which Activation Function to Use?

It depends!

- **Sigmoid** outputs values between 0 and 1, perfect for probability-based models. Face challenges during training - vanishing and exploding gradient problem.
  - Vanishing gradient: as gradient is **backpropagated**, gets multiplied by the derivative of the activation function at each layer. If derivative is small, repeatedly multiplying causes gradient to diminish -> updates to weights (particularly in early layers) become small -> slow to learn.
  - Exploding gradient: occurs when gradients during **backpropagation** become excessively large, destabilising the model (caused by improper initialisation of parameters or inappropriate learning rate).

- **ReLU** is efficient to calculate with, avoids vanishing / exploding gradient (gradient is 0 or 1) but can lead to inactive neurons.

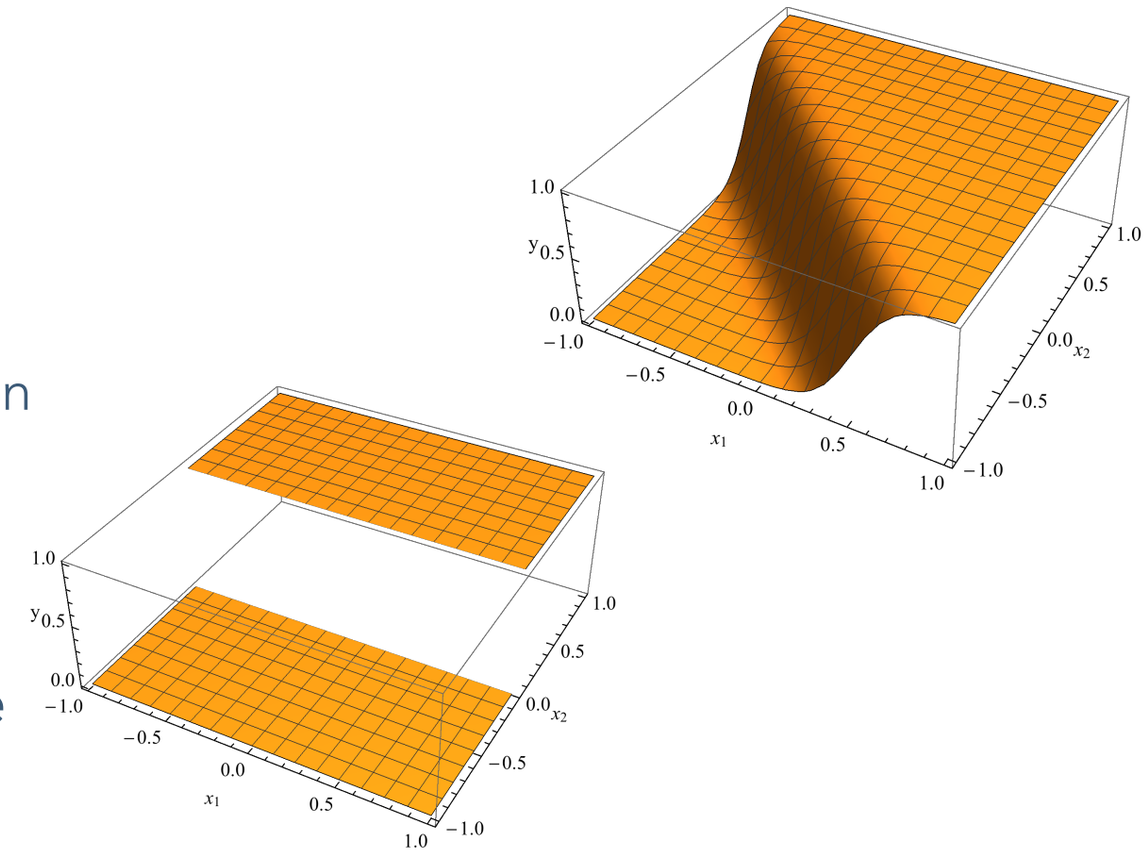
- **Leaky ReLU** less inactive neurons but network can be sensitive to chosen value of a -> inconsistent for different choices.

- **PReLU** enhances model accuracy and convergence but fine-tuning the (learnable) a parameter can be time consuming.

Artificial Neural Networks

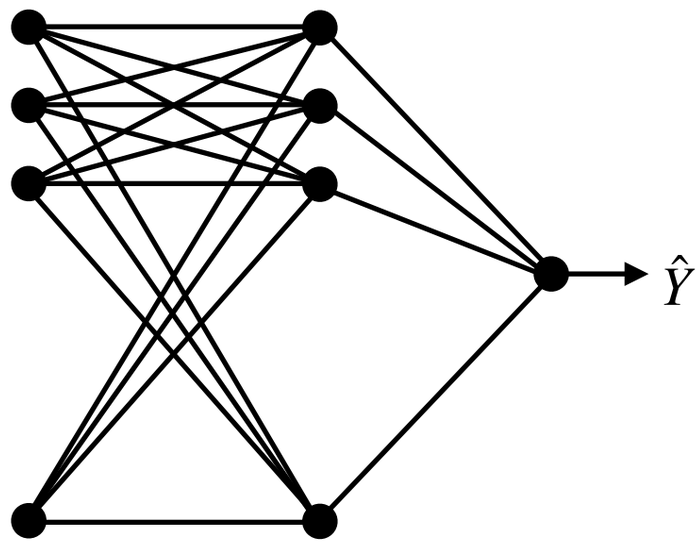
Artificial Neural Networks (ANNs)

- A **single neuron** can be thought of as defining a **hyperplane** that separates the input feature space into two regions.
  - Binary threshold perceptron this is “literally true”.
  - Other perceptrons discussed have a gradual transition from one region to another.



- Can **combine perceptrons** to impose multiple hyperplanes on the input feature space to divide the data into different regions.

- Such a system is an ANN. In HEP, usually synonymous with a **multi-layer perceptron (MLP)**:
  - An MLP has multiple layers of perceptrons.
  - Output of first layer fed into subsequent layer and so on.
  - Ultimately responses of the final layer are brought together to compute an **overall value** $\hat{Y}$ for the network response.



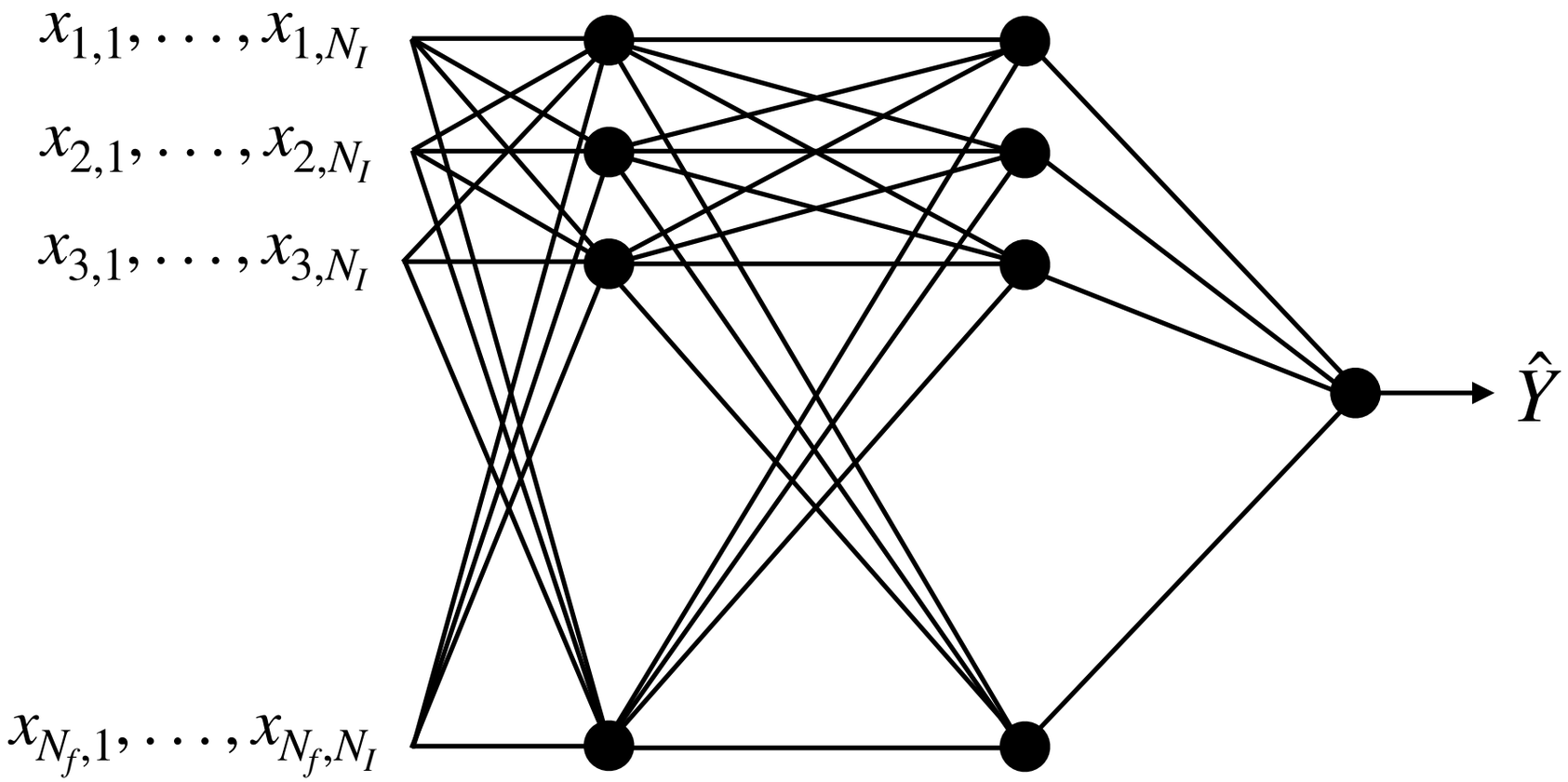
Multi-layer Perceptrons: Illustrative Example

Initial features →

Target feature (to predict)

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & \cdots & x_{1,N_I} \\ x_{2,1} & x_{2,2} & x_{2,3} & \cdots & x_{2,N_I} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{N_f,1} & x_{N_f,2} & x_{N_f,3} & \cdots & x_{N_f,N_I} \end{bmatrix} \quad \text{Instances} \downarrow$$

$$Y = [y_{I_1}, y_{I_2}, \ldots, y_{I_{N_I}}]$$



Multi-layer Perceptrons: Illustrative Example

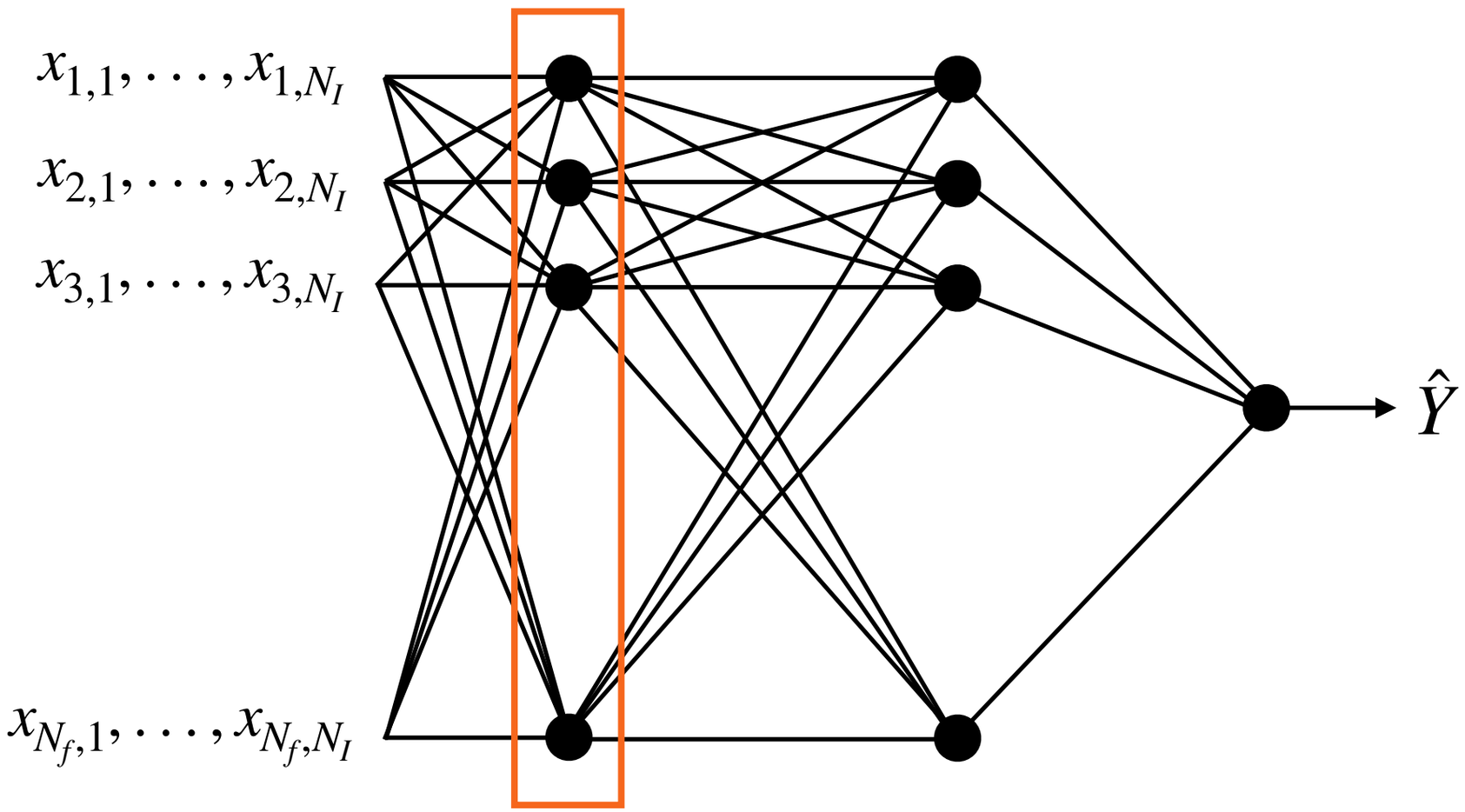
Initial features →

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & \cdots & x_{1,N_I} \\ x_{2,1} & x_{2,2} & x_{2,3} & \cdots & x_{2,N_I} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{N_f,1} & x_{N_f,2} & x_{N_f,3} & \cdots & x_{N_f,N_I} \end{bmatrix}$$

↕ Instances

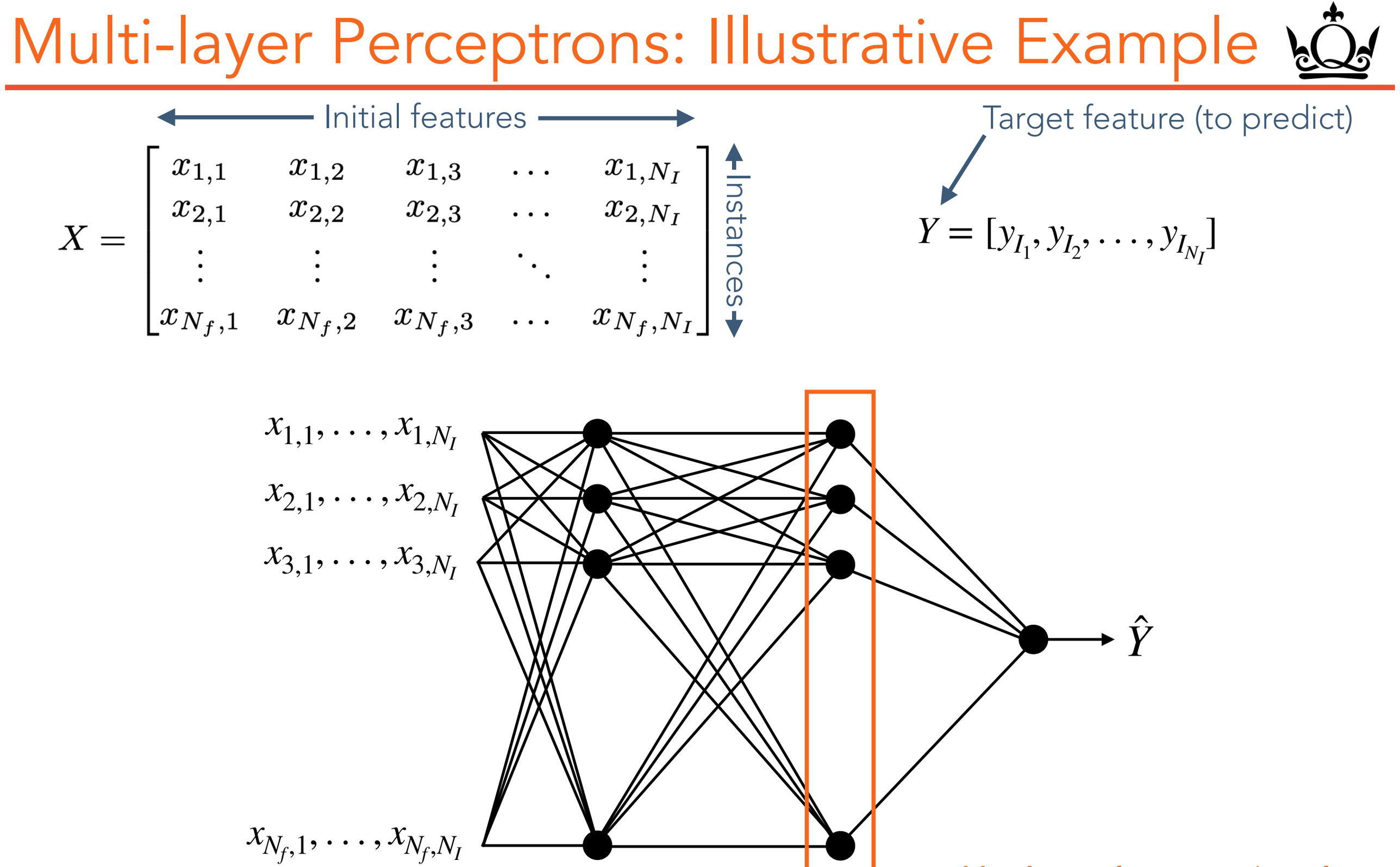
Target feature (to predict) ↓

$$Y = [y_{I_1}, y_{I_2}, \ldots, y_{I_{N_I}}]$$



Input layer of N_f perceptrons (one for each dimension of input *feature* space).

Multi-layer Perceptrons: Illustrative Example



This particular layer is **fully connected (dense)** since each node in the layer is connected to every other node in previous layer.

Hidden layer of some number of perceptrons, N_{H_1}; at least one for each dimension of the input feature space.

Multi-layer Perceptrons: Illustrative Example

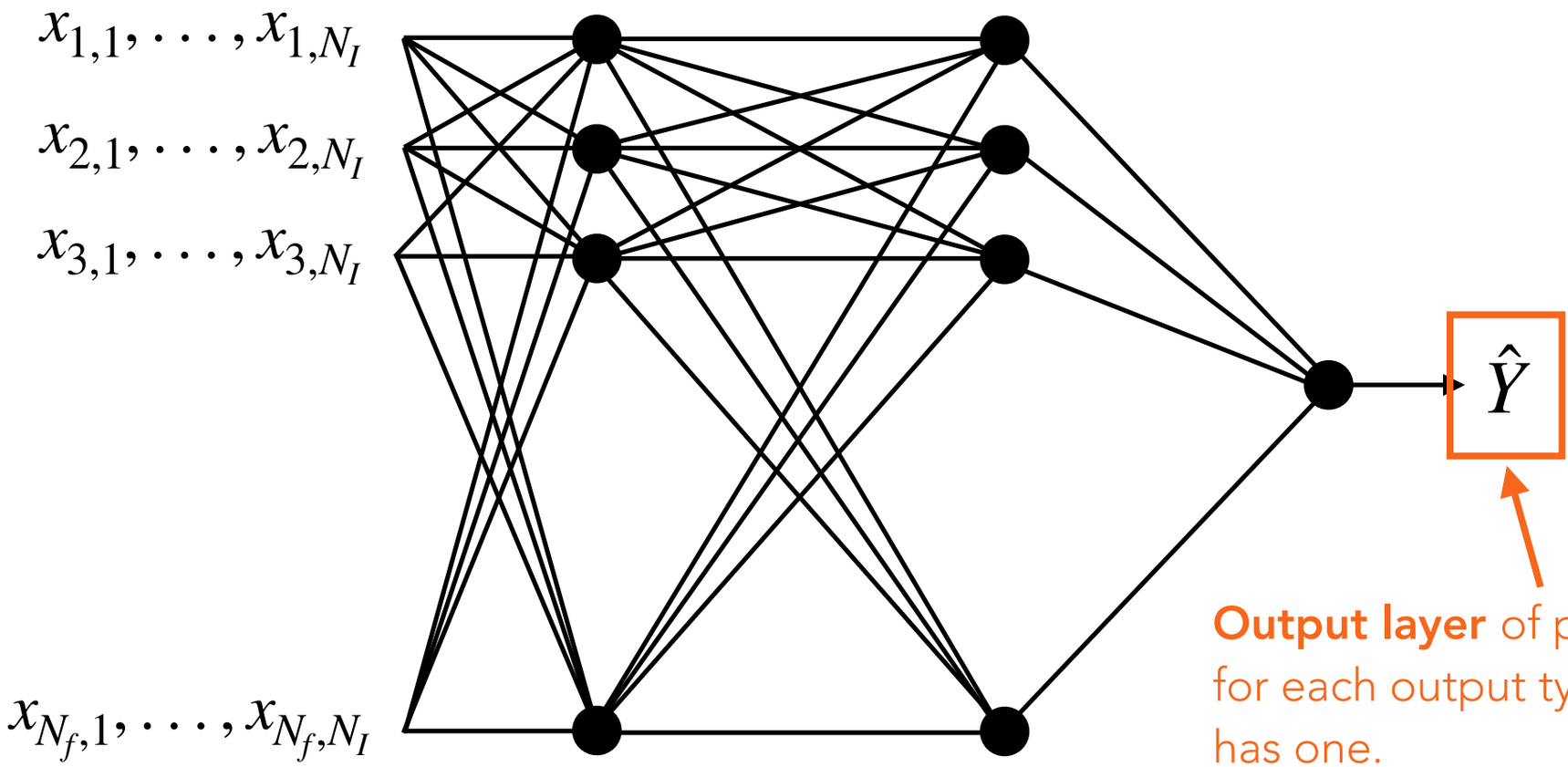
Initial features →

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & \cdots & x_{1,N_I} \\ x_{2,1} & x_{2,2} & x_{2,3} & \cdots & x_{2,N_I} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{N_f,1} & x_{N_f,2} & x_{N_f,3} & \cdots & x_{N_f,N_I} \end{bmatrix}$$

↕ Instances

Target feature (to predict) ↓

$$Y = [y_{I_1}, y_{I_2}, \ldots, y_{I_{N_I}}]$$



Output layer of perceptrons, one for each output type. This network has one.

Multi-layer Perceptrons: Illustrative Example

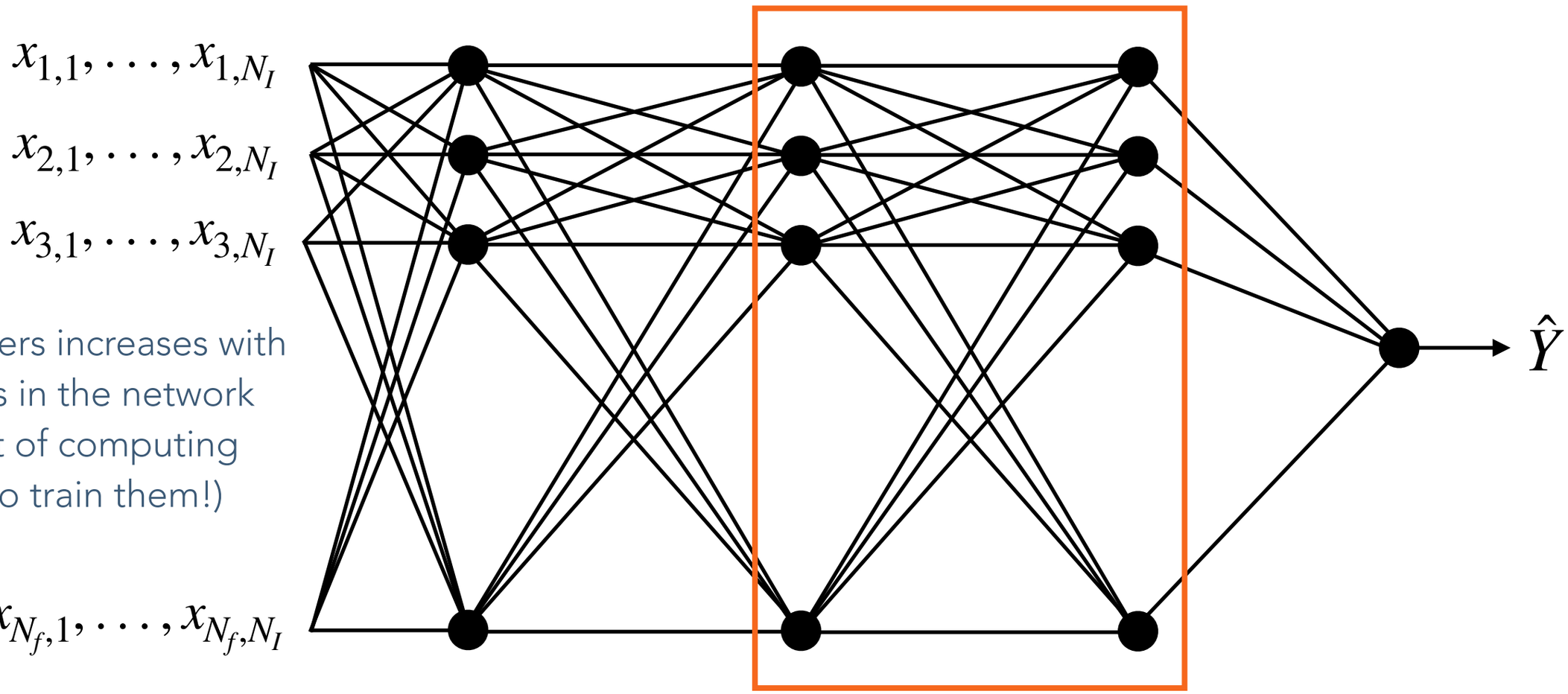
Initial features

Instances

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & \cdots & x_{1,N_I} \\ x_{2,1} & x_{2,2} & x_{2,3} & \cdots & x_{2,N_I} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{N_f,1} & x_{N_f,2} & x_{N_f,3} & \cdots & x_{N_f,N_I} \end{bmatrix}$$

Target feature (to predict)

$$Y = [y_{I_1}, y_{I_2}, \ldots, y_{I_{N_I}}]$$



Number of parameters increases with the number of layers in the network (as does the amount of computing resources required to train them!)

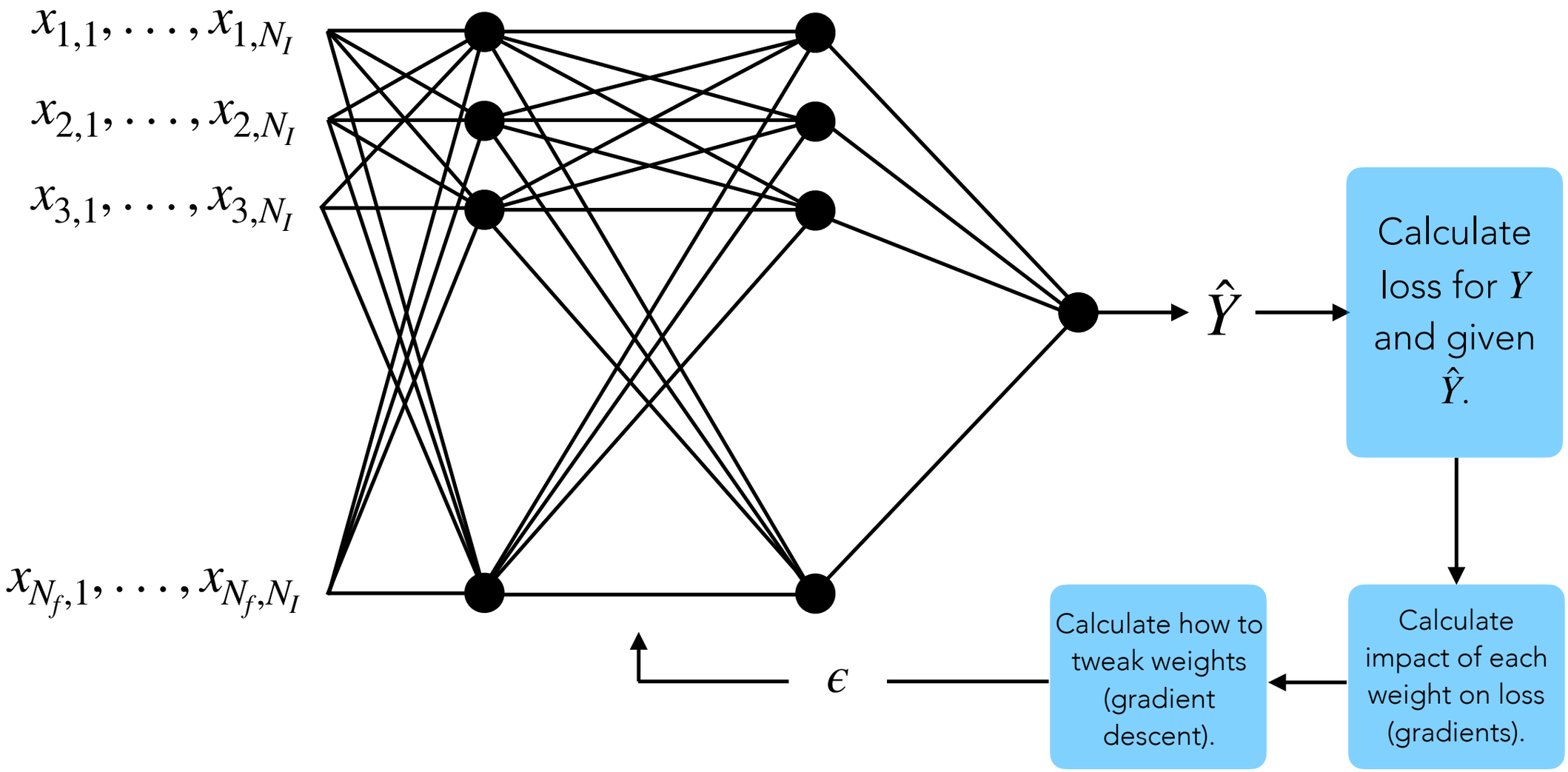
The **depth** of a network relates to the number of hidden layers -> greater than ~ a couple-> **deep NN**.

Training

- Each neuron has its own weights & bias (parameters). The **optimisation / tuning** of these parameters is referred to as **training**.

- Tune based on some metric (figure of merit) called the **loss function**.

- Usually want to **minimise the loss** - usually done by **gradient descent**. Tells us how to **tweak weights to reduce the loss**.

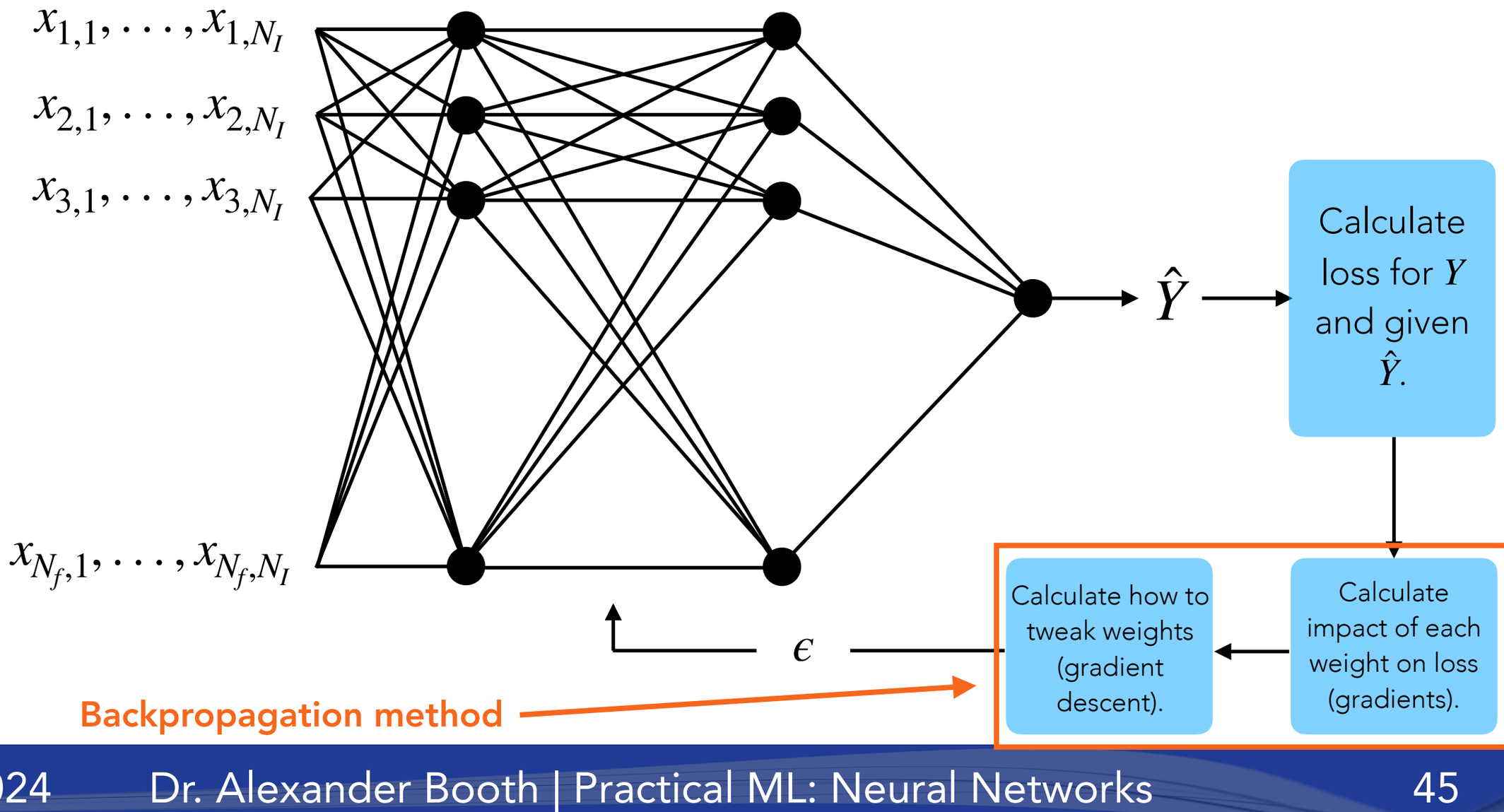


Training

- Each neuron has its own weights & bias (parameters). The **optimisation / tuning** of these parameters is referred to as **training**.

- Tune based on some metric (figure of merit) called the **loss function**.

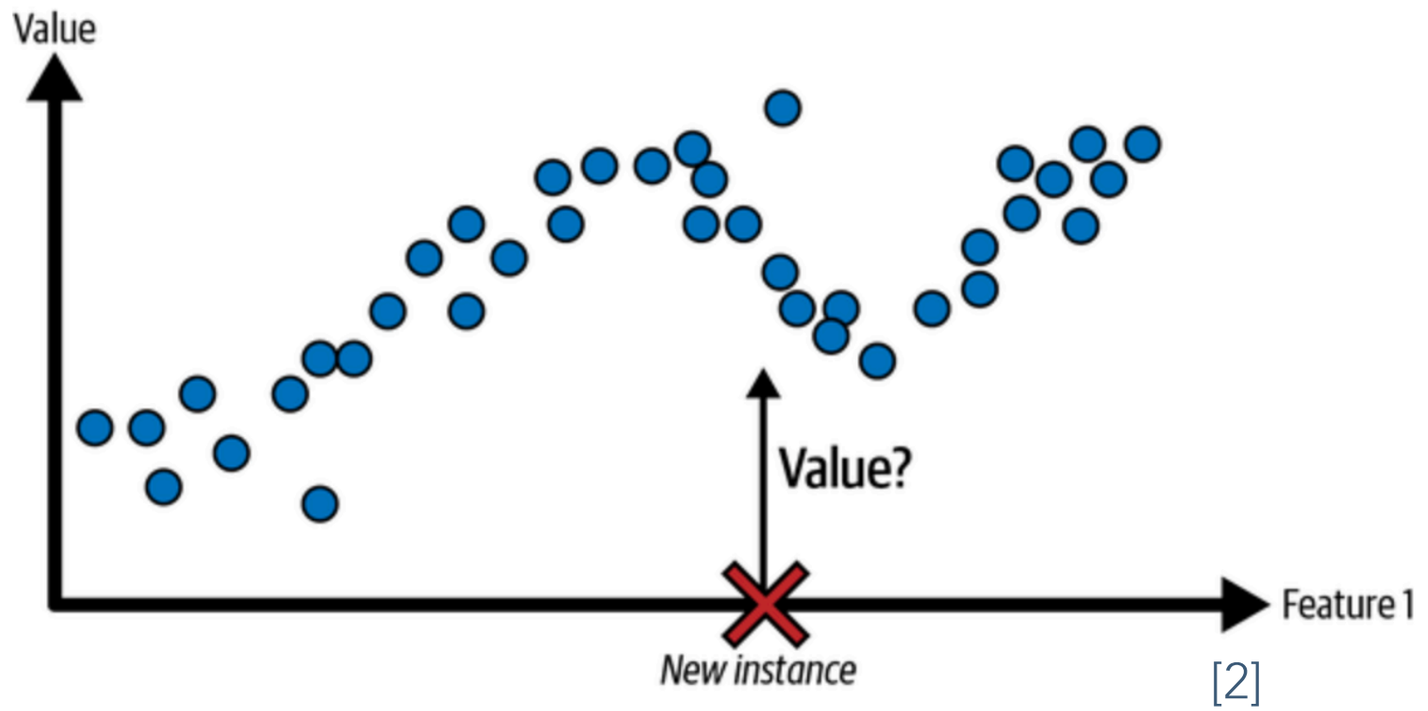
- Usually want to **minimise the loss** - usually done by **gradient descent**. Tells us how to **tweak weights to reduce the loss**.



Linear Regression

What is Regression?

- Predicting a target **numeric value** (**response**) given a set of features.
  - House prices given number of bedrooms.
  - Time to run a marathon given resting heart rate and blood pressure.

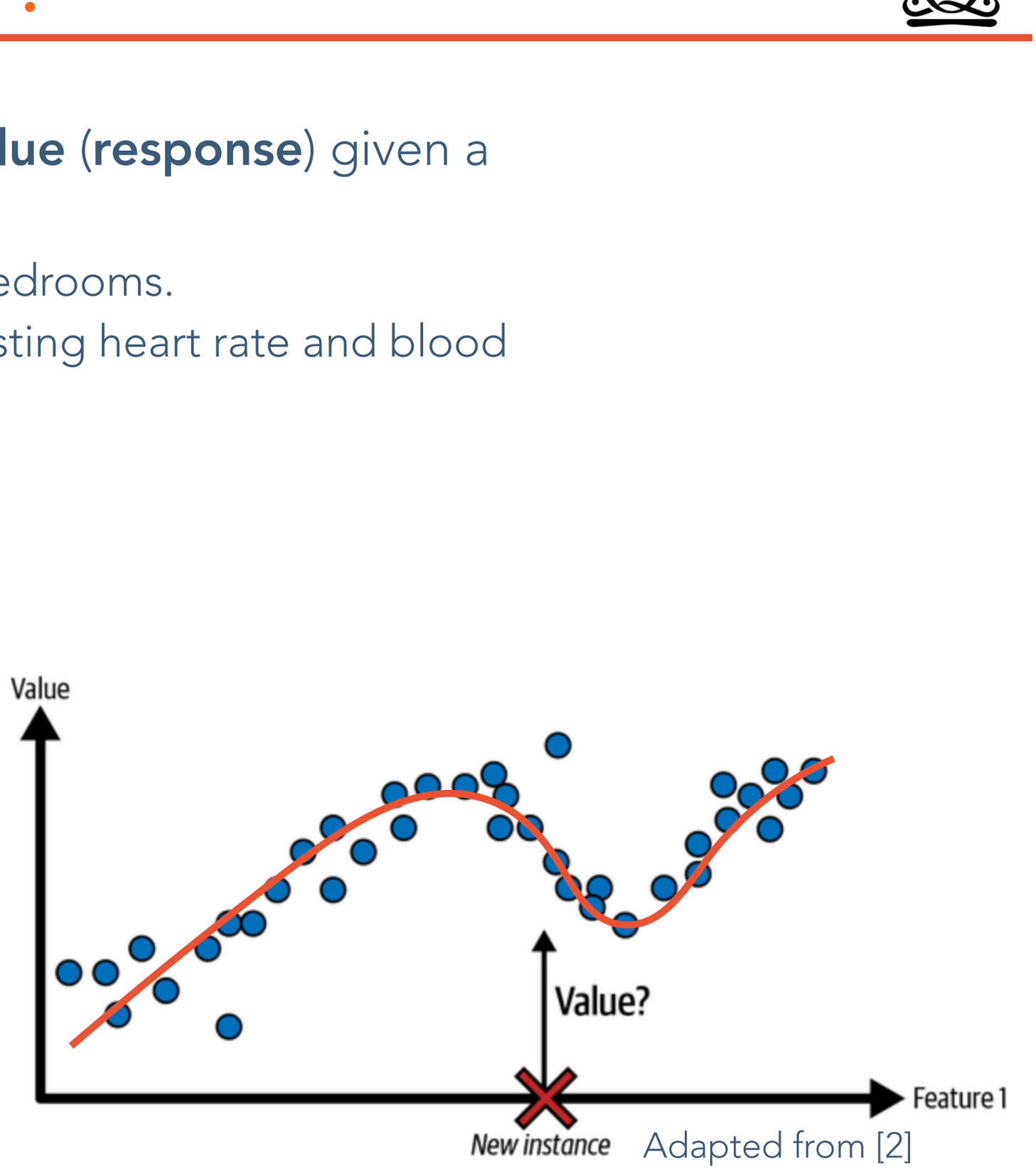


[2]

What is Regression?

- Predicting a target **numeric value** (**response**) given a set of features.
  - House prices given number of bedrooms.
  - Time to run a marathon given resting heart rate and blood pressure.

- "Curve fitting" - parametrising relationship between a set of independent variables and a dependent variable.



Adapted from [2]

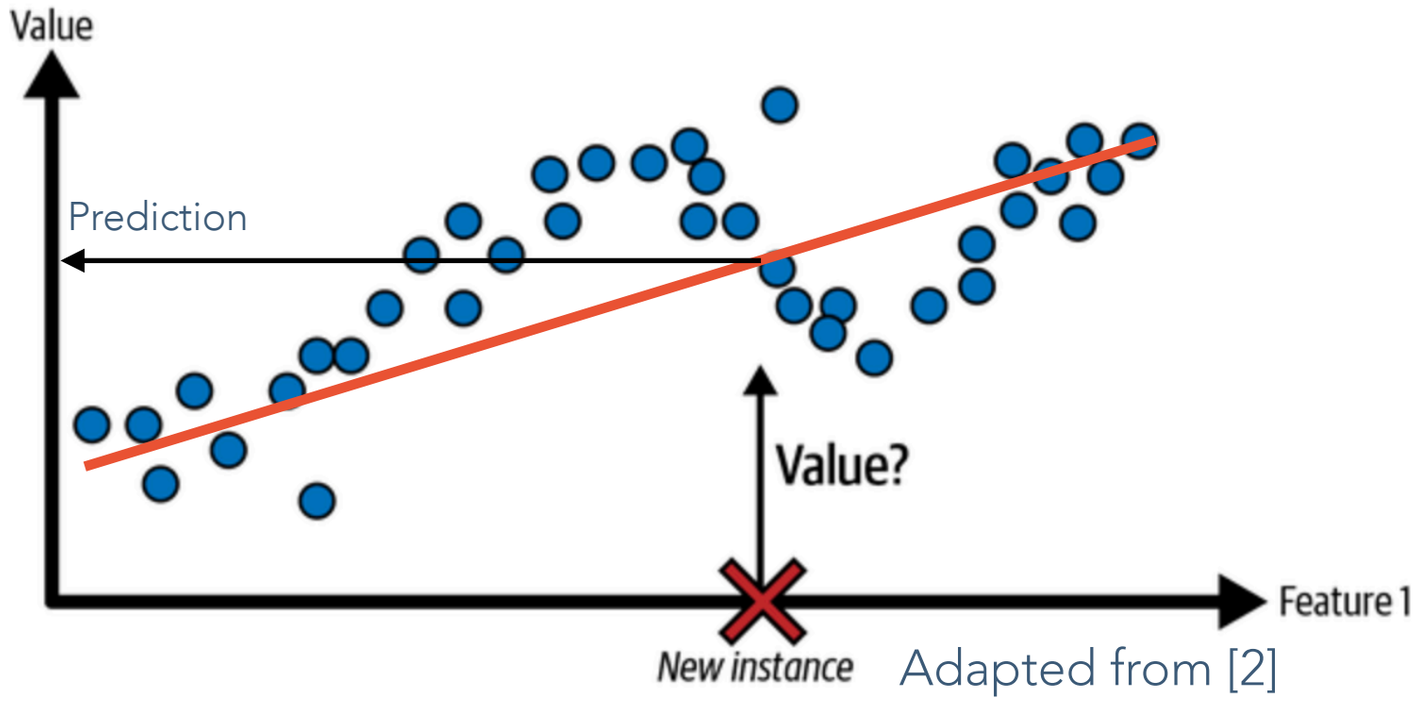
Linear Regression

- Assume the data have a linear relationship -> apply a linear model.

$$y = mx + c$$

- Describes a straight line where:
  - m is the slope of the line
  - c is the constant offset (y value at $x = 0$).

- The problem: selecting the values of m and c in order to obtain the best possible model of the data.



Adapted from [2]

Linear Regression

- Assume the data have a linear relationship -> apply a linear model.

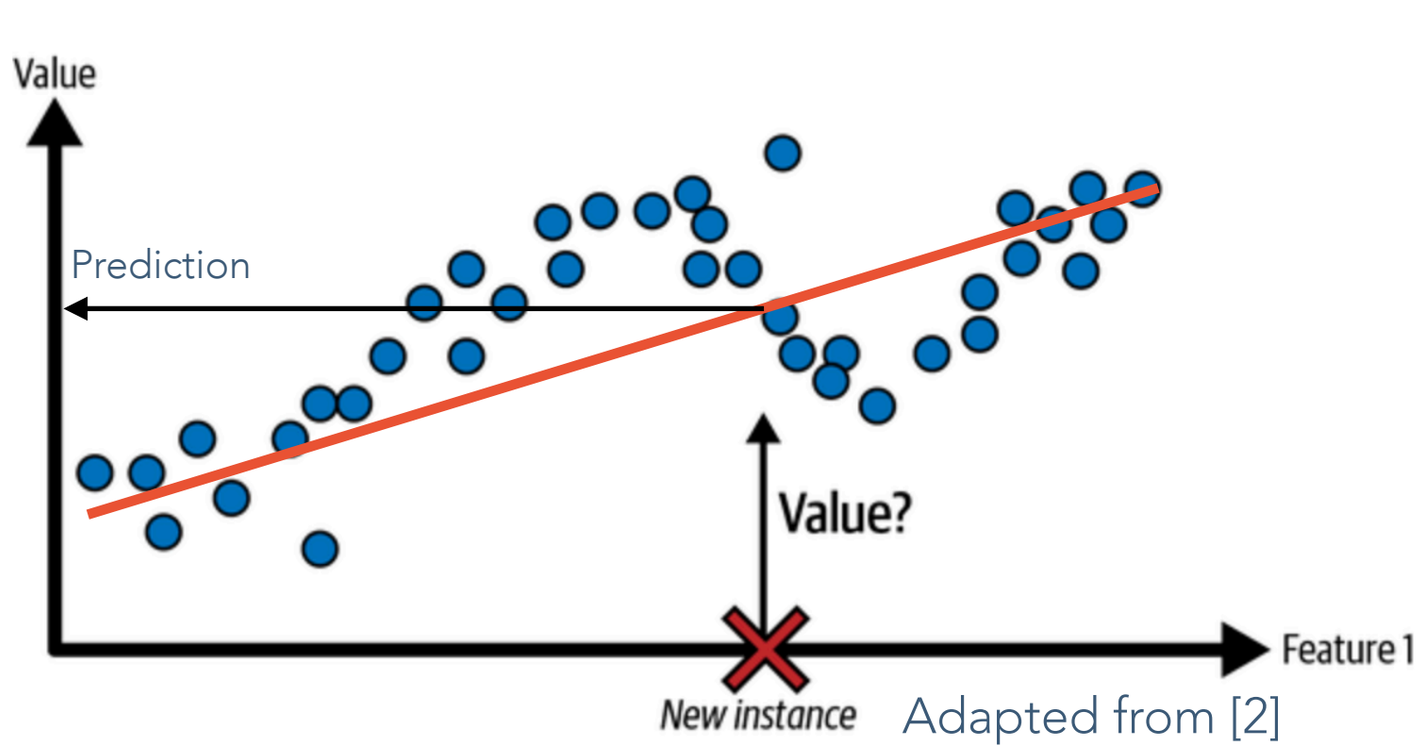
$$y = mx + c$$

- Describes a straight line where:
  - m is the slope of the line
  - c is the constant offset (y value at $x = 0$).

Define a method that can be used to extract optimal values.

- The problem: **selecting the values** of m and c in order to obtain the **best possible model** of the data.

Define a figure of merit.



Adapted from [2]

Linear Regression: Figure of Merit

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{y_i - (mx_i + c)}{\sigma_i} \right)^2$$

- χ^2: sum of the normalised square residual of each instance.
  - y_i: y value for the i^{th} instance.
  - x_i: x value for the i^{th} instance.
  - σ_i: error on the value of y for the i^{th} instance.
  - m and c: model parameters to be determined.

Measure of the total "distance" of each point from the line.

- Optimal value of m and c is the pair that produces the smallest χ^2.
  - No guarantee that this choice will result in a good model (overfitting / overtraining discussed tomorrow).

- *Note:* if it's assumed that all data points have a similar error, simplify the problem by neglecting σ_i (setting values to unity).

Linear Regression: Optimisation

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{y_i - (mx_i + c)}{\sigma_i} \right)^2$$

- Systematically vary m and c to minimise the χ^2.
  - Conceptually - visualise choosing pairs of m and c, and for each point in this 2D space, compute χ^2.
  - From the ensemble of points in this hyperspace, can select the minimum.

- Algorithmically this is expensive so we use algorithms that approximate the search for the minimum that is more computationally efficient (and adaptable to high dimension parameter spaces - more features).
  - We will use a **gradient descent parameter optimisation** algorithm. For now treat this optimisation process as a black box (revisit in detail in lecture 2).
  - *Note:* this particular problem is solvable analytically.

Linear Regression: Illustrative Example in 1D

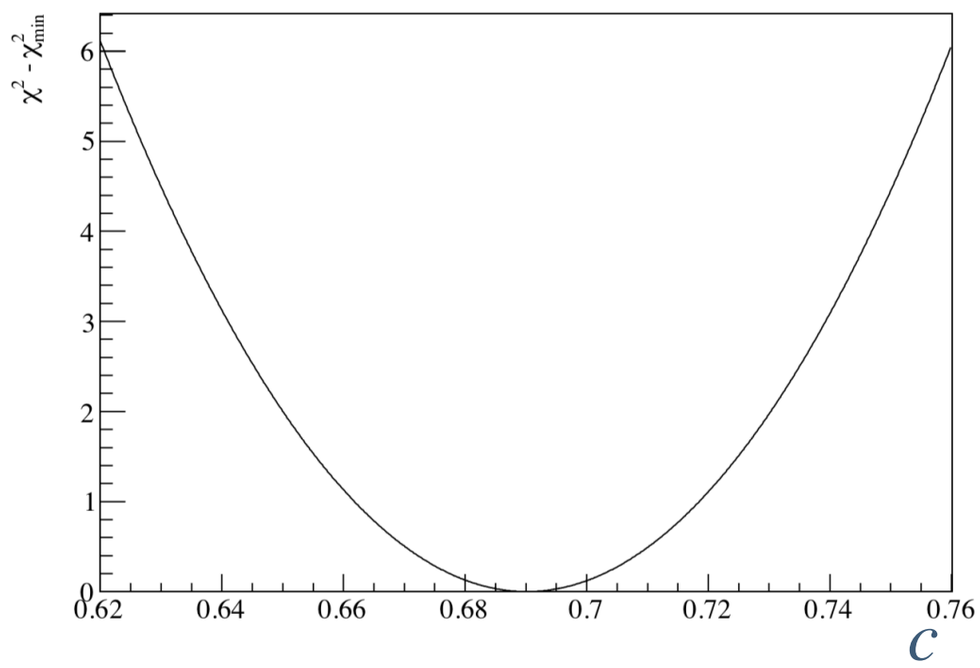
- Take an ensemble of measurements of some quantity S.

i =	1	2	3	4	5	6	7
S	0.662	0.625	0.897	0.614	0.925	0.694	0.601
σ_S	0.039	0.091	0.100	0.160	0.160	0.061	0.239

- "Extract the average value of S from these data" -> fit a line with a gradient of zero -> fitting $y = c$.

$$\chi^2 = \sum_{i=1}^{7} \left(\frac{S_i - c}{\sigma_{S_i}} \right)^2$$

- Optimisation algorithm: choose several sensible values for c, calculate the χ^2 for each one.



Average of S is at the minimum.

Linear Regression: Illustrative Example in 1D

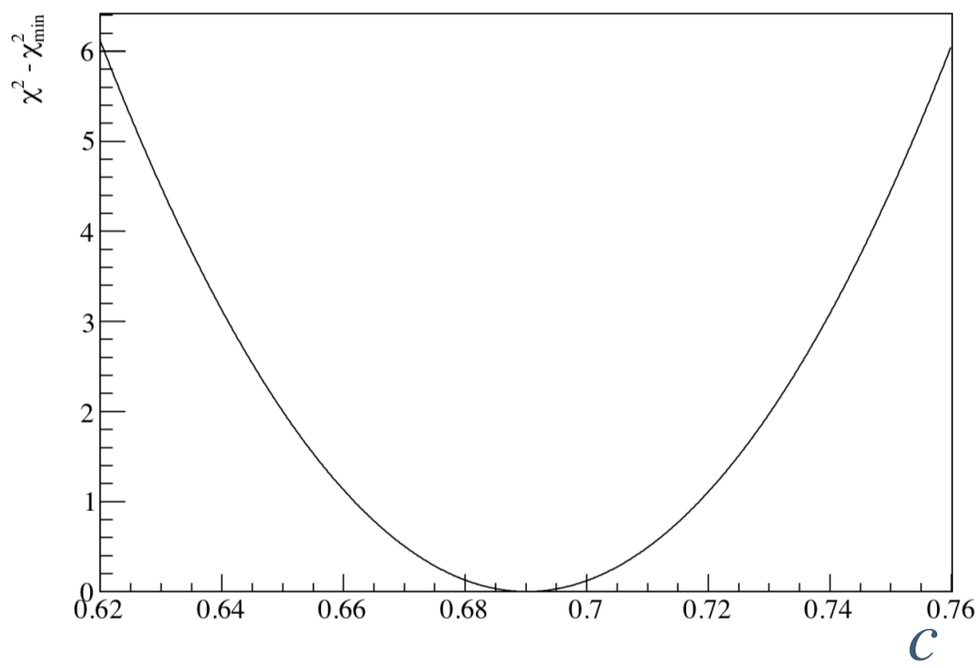
- Take an ensemble of measurements of some quantity S.

i =	1	2	3	4	5	6	7
S	0.662	0.625	0.897	0.614	0.925	0.694	0.601
σ_S	0.039	0.091	0.100	0.160	0.160	0.061	0.239

- "Extract the average value of S from these data" -> fit a line with a gradient of zero -> fitting $y = c$.

$$\chi^2 = \sum_{i=1}^{7} \left(\frac{S_i - c}{\sigma_{S_i}} \right)^2$$

- Optimisation algorithm: **choose several sensible values for** c, calculate the χ^2 for each one.



"Grid search"

Average of S is at the minimum.

Linear Regression: Today's Notebook

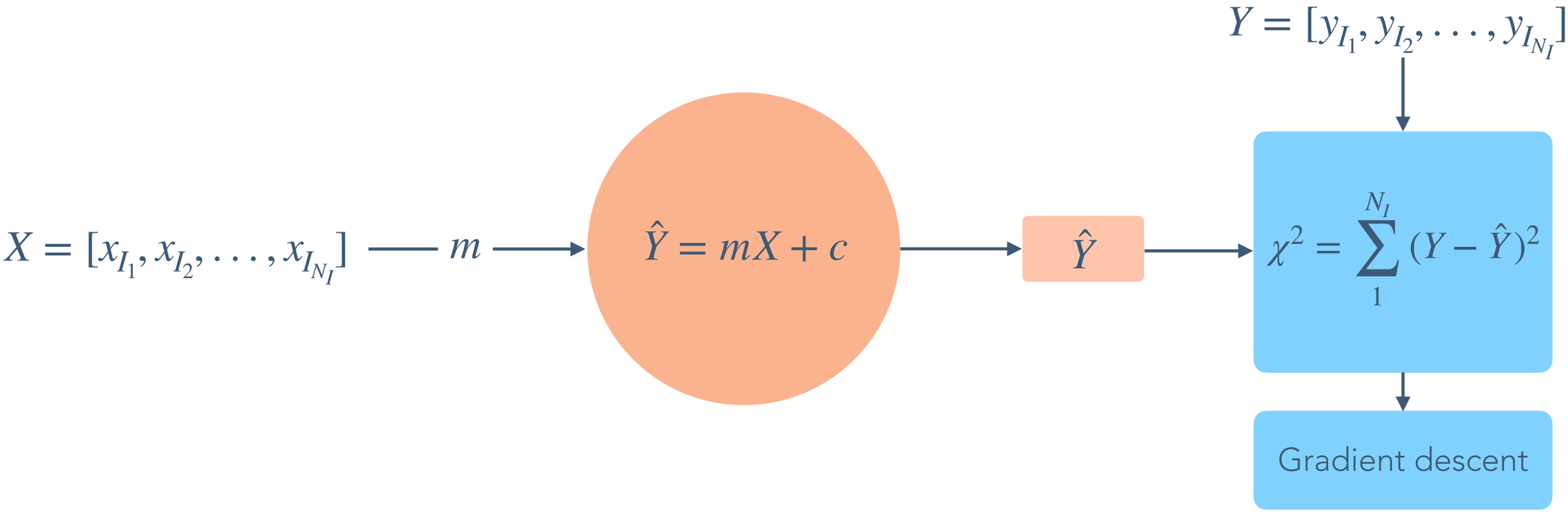
workshop-UCDN_PracticalML/lecture1_1/LinearRegression.ipynb

Perform linear regression using a Keras model.

- Generate "fake" data according to some linear distribution with added noise, ($X = [x_{I_1}, x_{I_2}, \ldots, x_{I_{N_I}}]$, $Y = [y_{I_1}, y_{I_2}, \ldots, y_{I_{N_I}}]$) split into two pieces:
  - **Training set (~90% of total dataset):** dataset used in the optimisation process to create a representative model of the data.
  - **Test set (~10% of total dataset):** dataset unseen in the training process, used to assess generalised performance of the trained model.

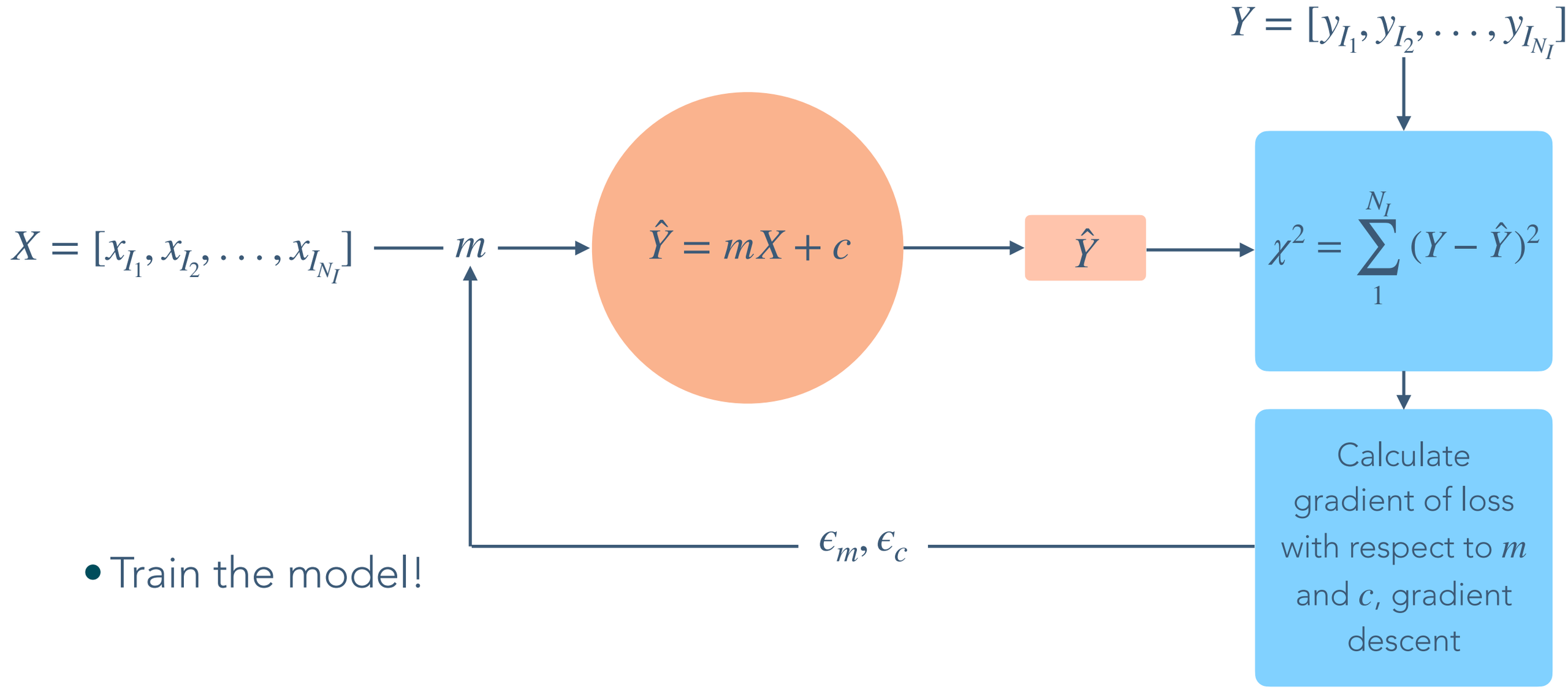
- Build model (parameters are a single weight (m) and single bias (c)):
  - Single dense layer with **linear activation function** (nothing done to weighted sum of inputs).
  - Define **loss function** "mean square error" -> χ^2.
  - Define optimisation algorithm (ADAM -> gradient descent) and choose the **learning rate** ("step size").

Linear Regression: Today's Notebook



- Build model (parameters are a single weight (m) and single bias (c)):
  - Single dense layer with **linear activation function** (nothing done to weighted sum of inputs).
  - Define **loss function** "mean square error" -> χ^2.
  - Define optimisation algorithm (ADAM -> gradient descent) and choose the **learning rate** ("step size").

Linear Regression: Today's Notebook

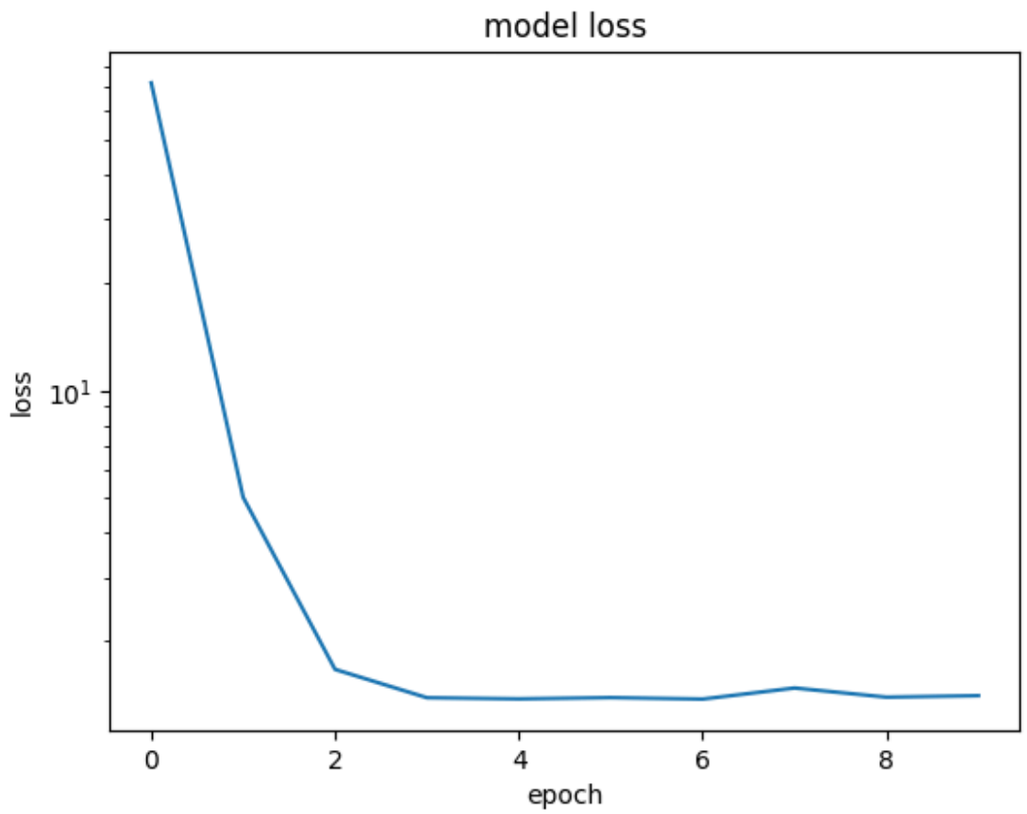


- Train the model!

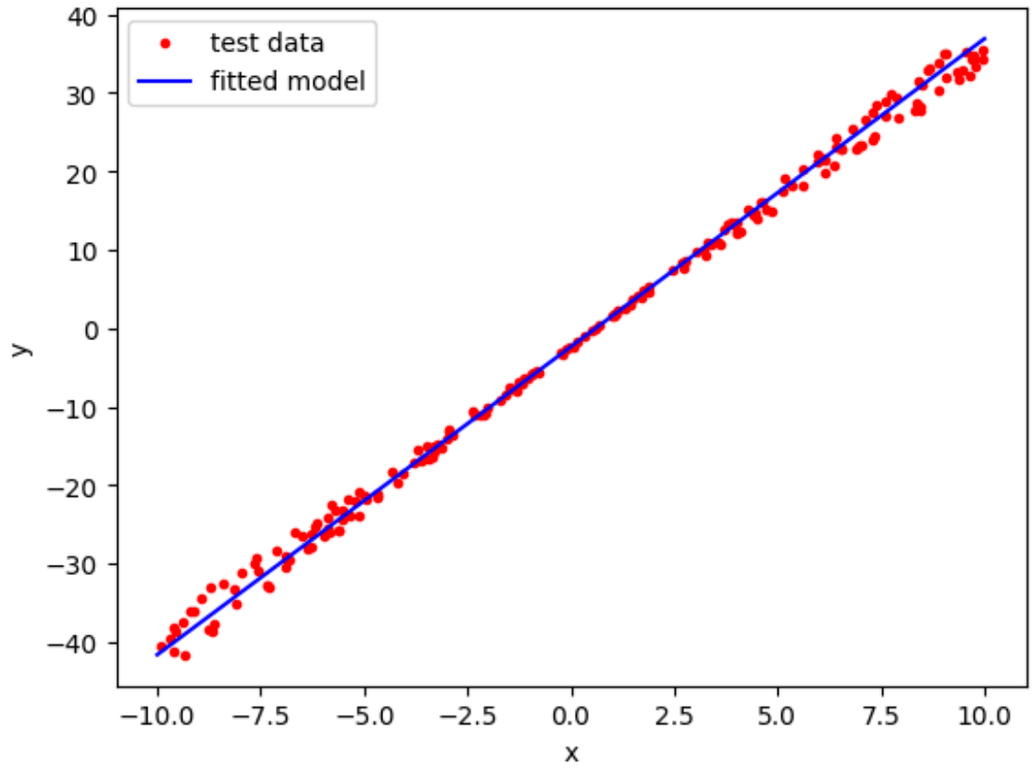
Linear Regression: Today's Notebook

- Study how the solution depends on:
  - The learning rate.
  - The number of **training epochs** (number of times the optimisation is run).

Plotting the loss function evolution as a function of training epoch



Plotting the optimised model and test data

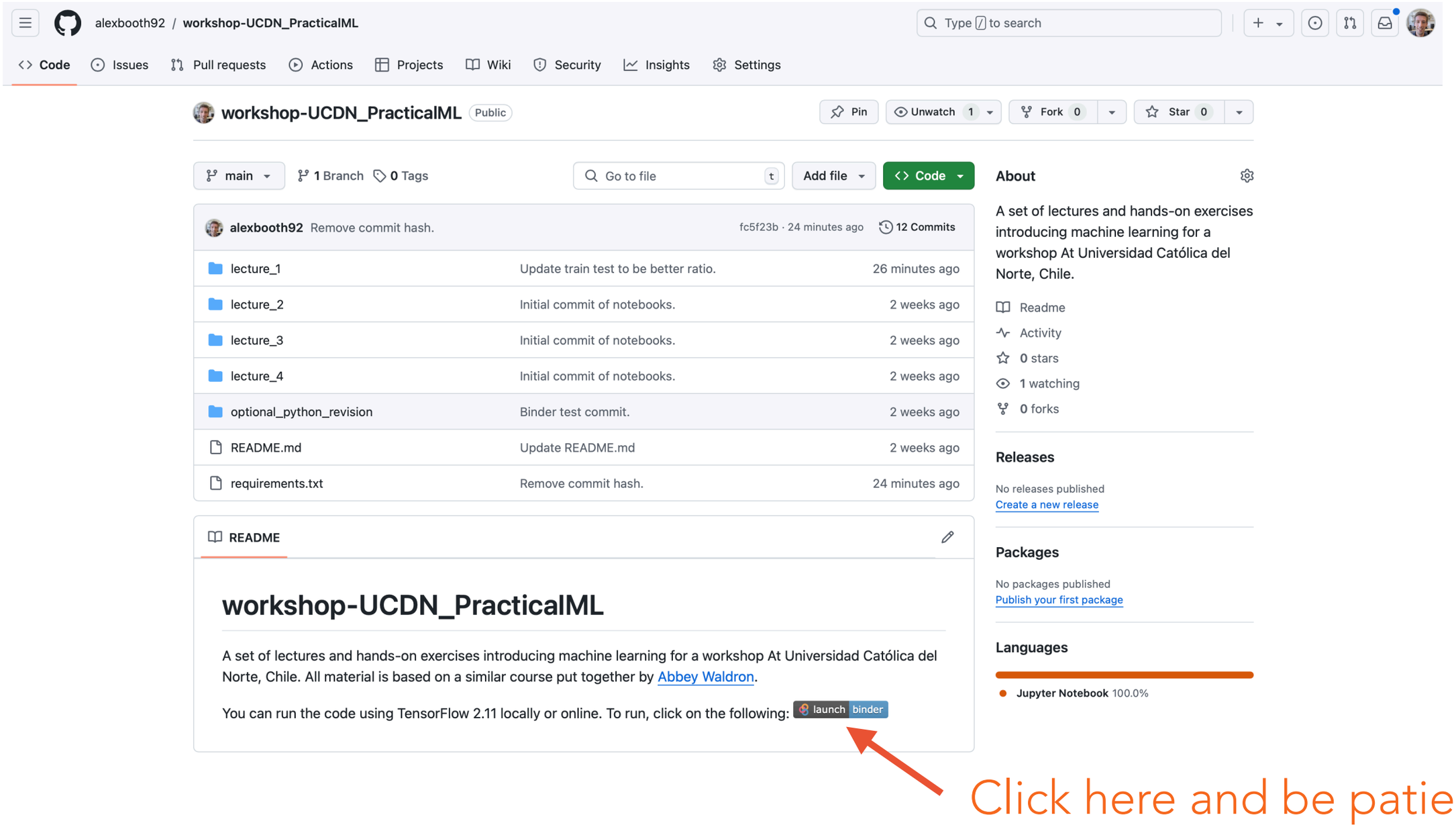



```
m =  3.9265254
c =  -2.354052
MSE loss =
 Fitted line xrange = [-10, 10], y range = [-41.61930561065674, 36.91120147705078]
```

Accessing the Notebooks

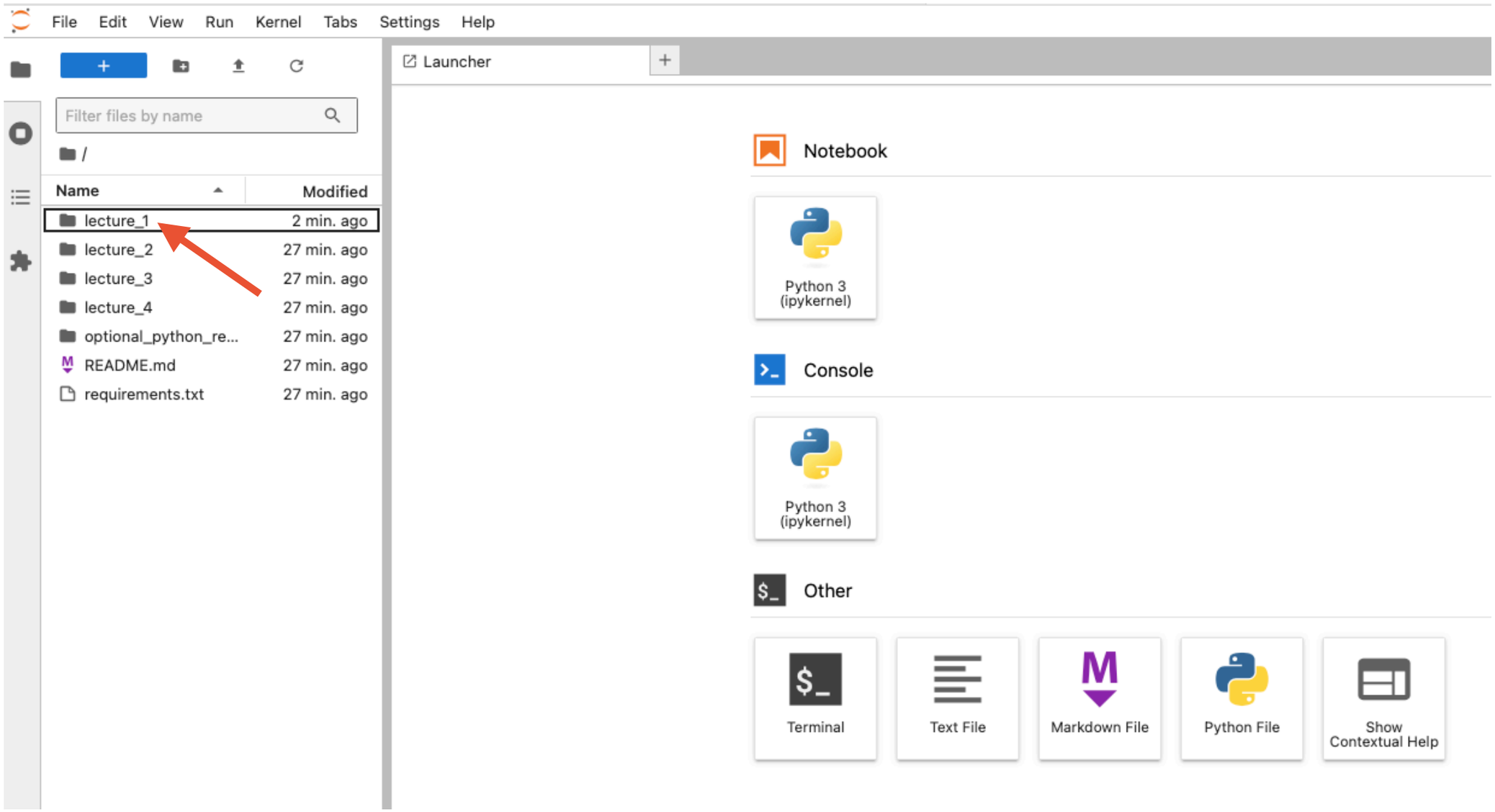
Follow this link:

https://github.com/alexbooth92/workshop-UCDN_PracticalML.git

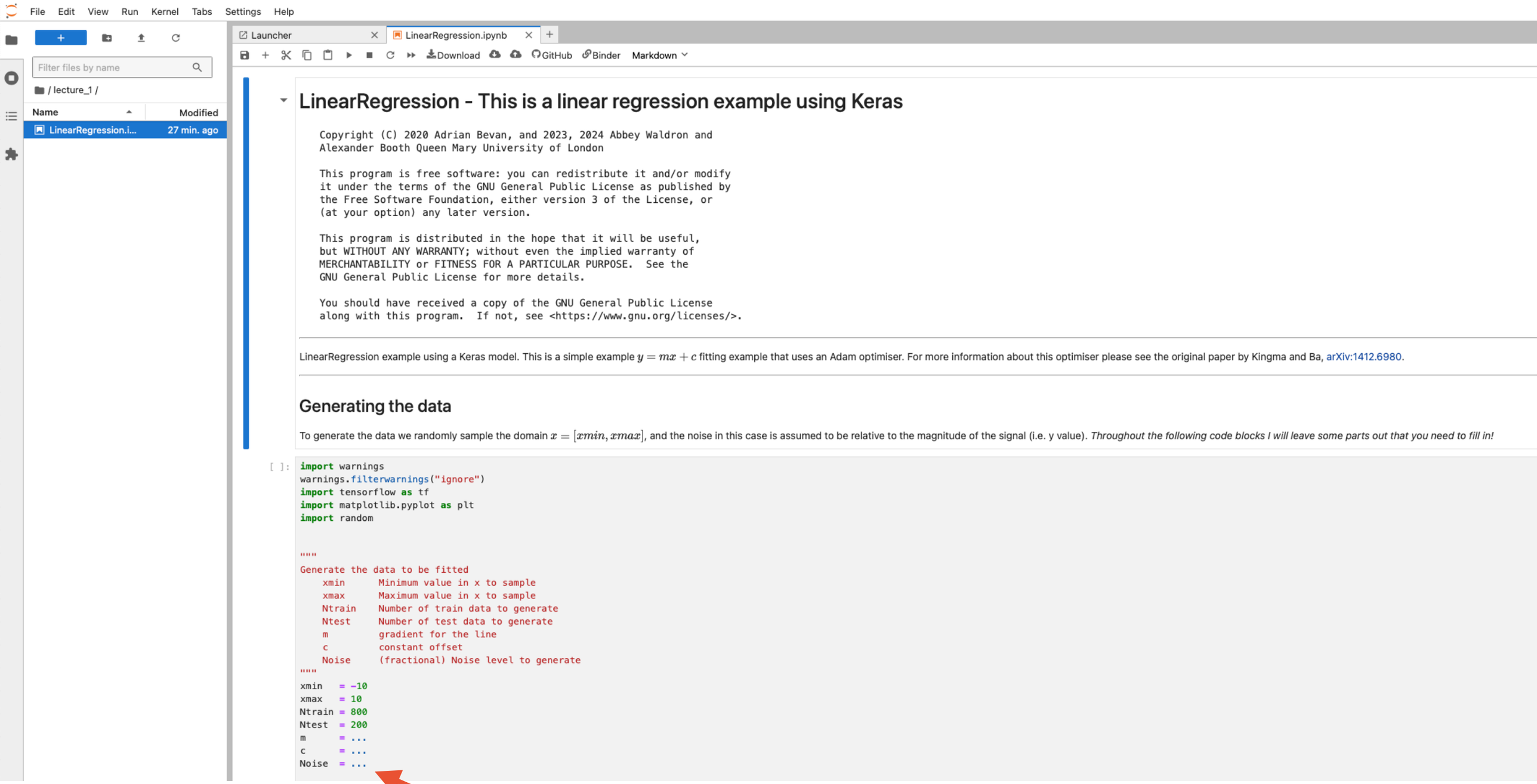


Click here and be patient!

Accessing the Notebooks



Accessing the Notebooks



Three dots mean you have to fill this in yourself!

Have Fun!

Queen Mary
University of London

References

[1] https://pixabay.com/vectors/neuron-nerve-cell-axon-dendrite-296581/

[2] A. Géron. Hands on Machine Learning with Scikit-Learn & TensorFlow, 2017.