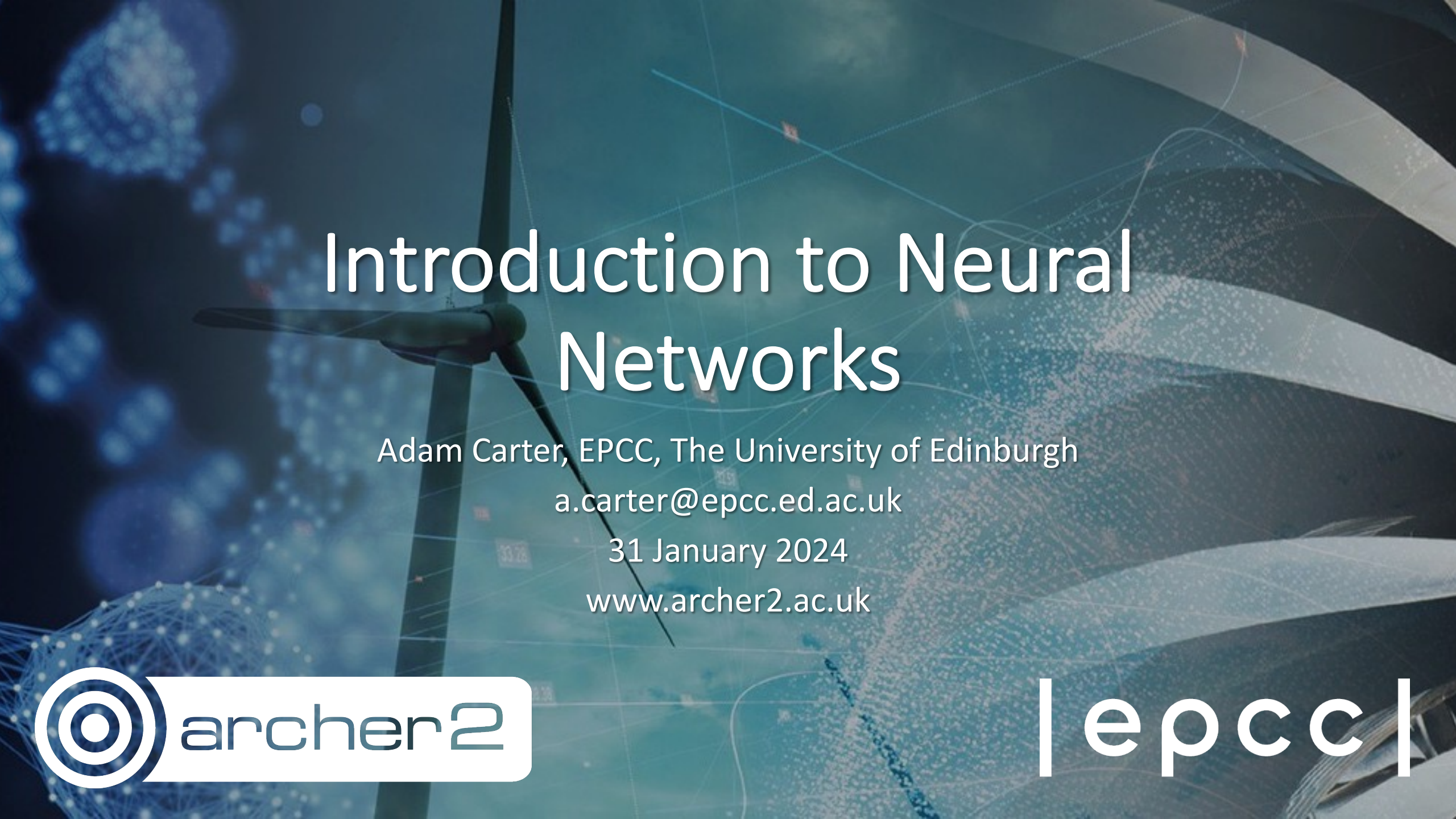


Introduction to Neural Networks

Adam Carter, EPCC, The University of Edinburgh
a.carter@epcc.ed.ac.uk

31 January 2024

www.archer2.ac.uk

archer2

epcc

Reusing this material

Partners

epcc

epcc

THE UNIVERSITY of EDINBURGH

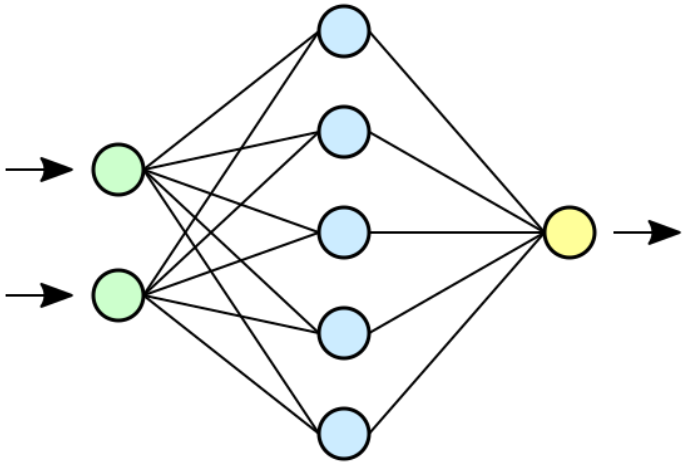
Hewlett Packard Enterprise

Overview

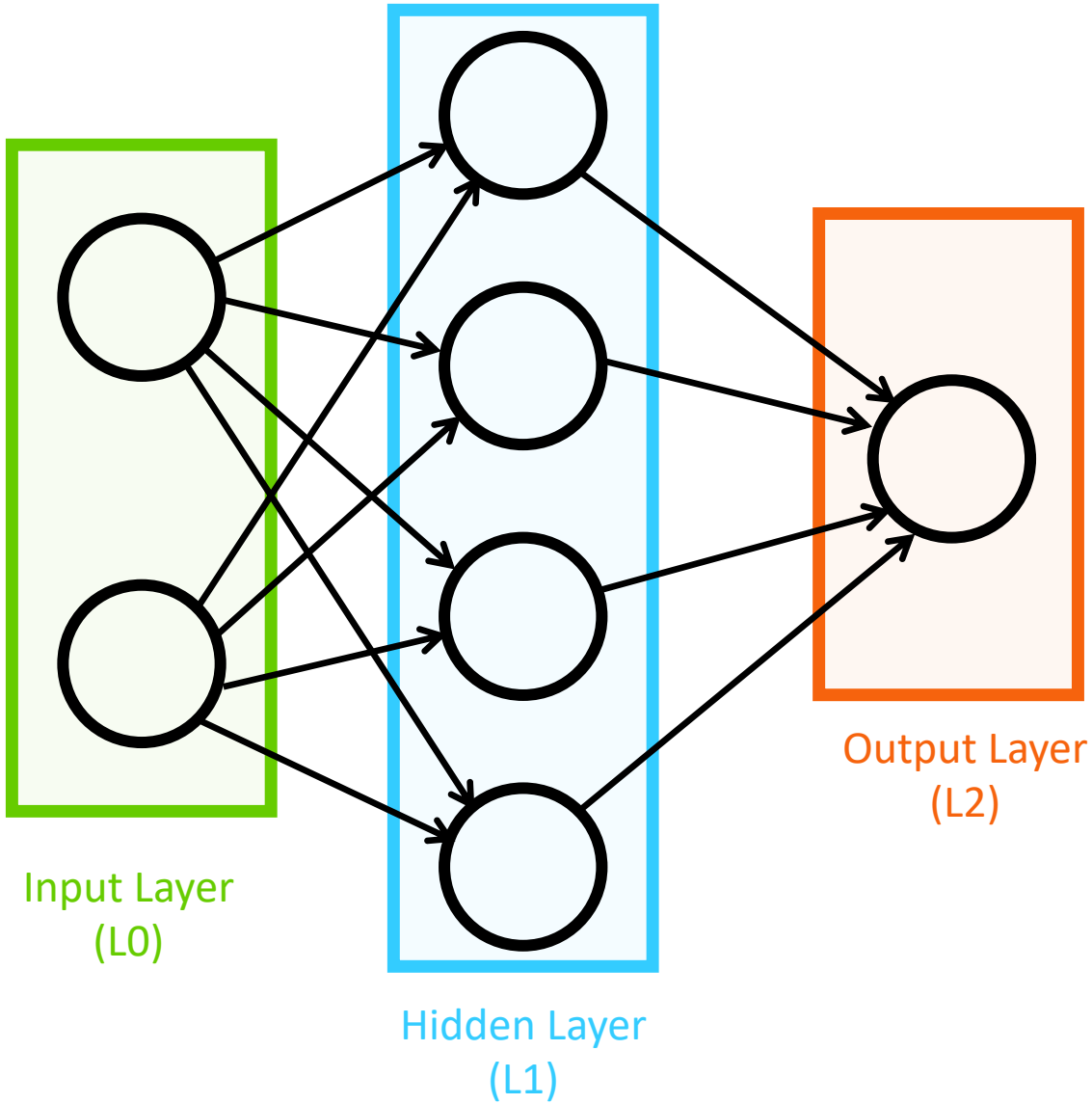
- Brief Introduction to Neural Networks
  - …introducing some terminology
- Intro to Logistic Regression
  - … to show that most of the ideas from Neural Networks come from simpler methods
- From Neurons to Networks
  - … to show how Neural Networks are built up
  - … and introducing activation functions
- Computational Graphs
- Forward Propagation
- Backward Propagation

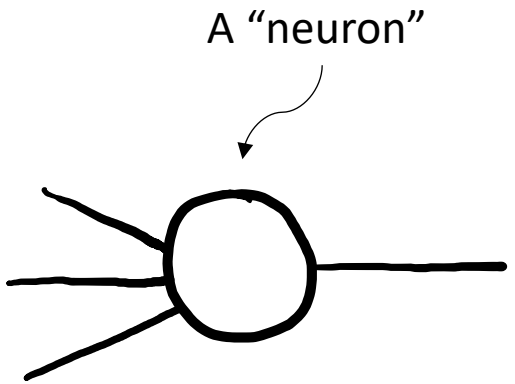
What is a Neural Network?

- “A neural network is a network or circuit of biological neurons, or, in a modern sense, an artificial neural network, composed of artificial neurons or nodes.” – Wikipedia
- “Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.” – IBM Cloud Education
- “A family of parametric, non-linear and hierarchical representation learning functions, which are massively optimized with stochastic gradient descent to encode domain knowledge, i.e. domain invariances, stationarity.” - Efstratios Gavves



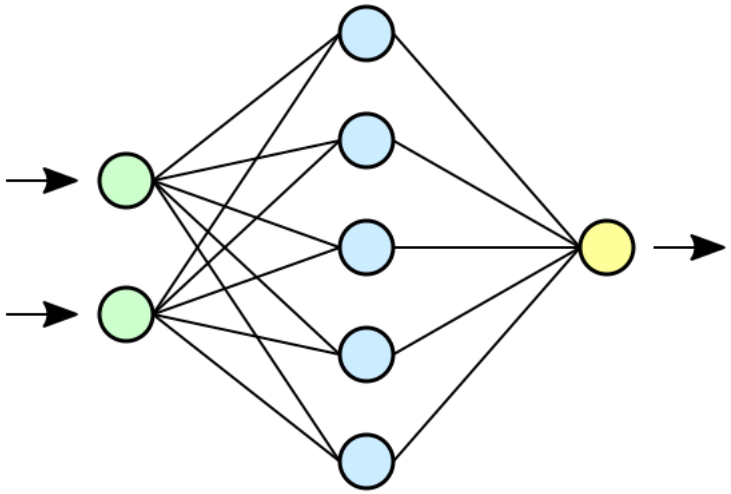
Components of a Neural Network

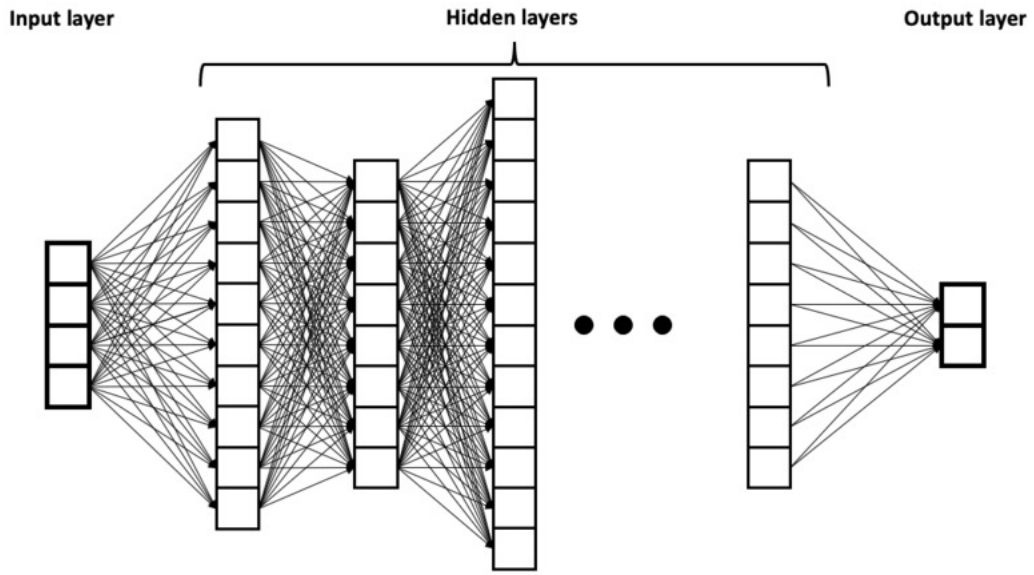




What is a Deep Neural Network?

- A Deep Neural Network is simply a neural network with many hidden layers

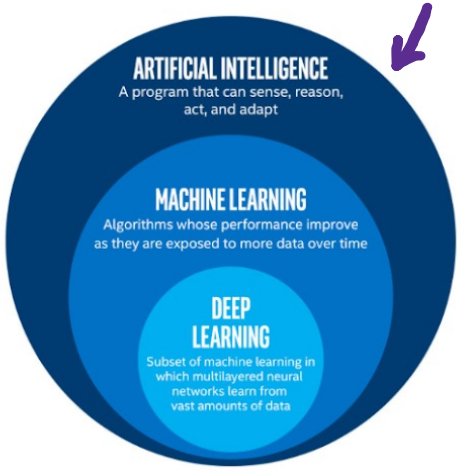


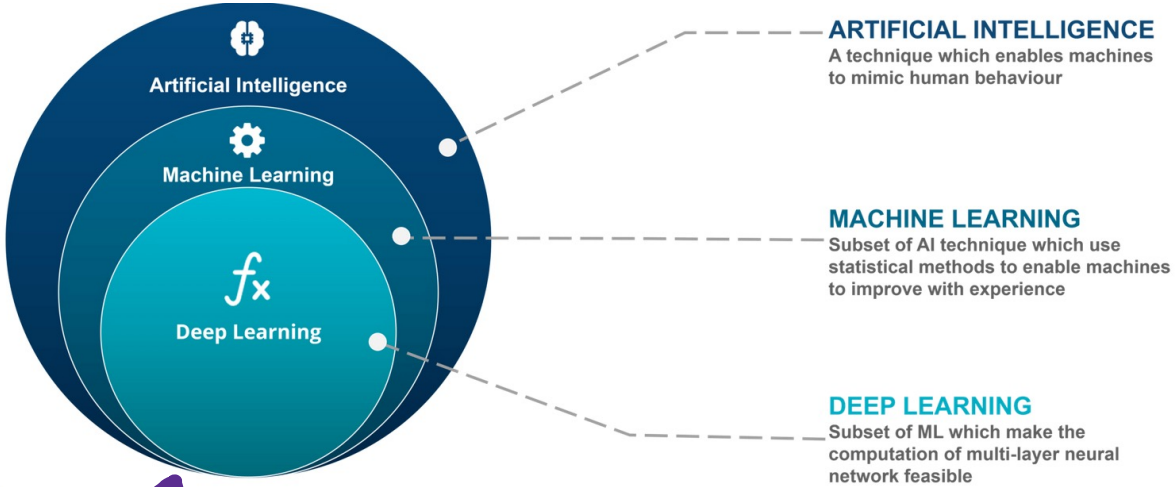


- And Deep Learning is simply Machine Learning using Deep Neural Networks
  - Some people use the terms “Deep Learning” and “Neural Networks” almost synonymously, because most of the Neural Networks used today are deep

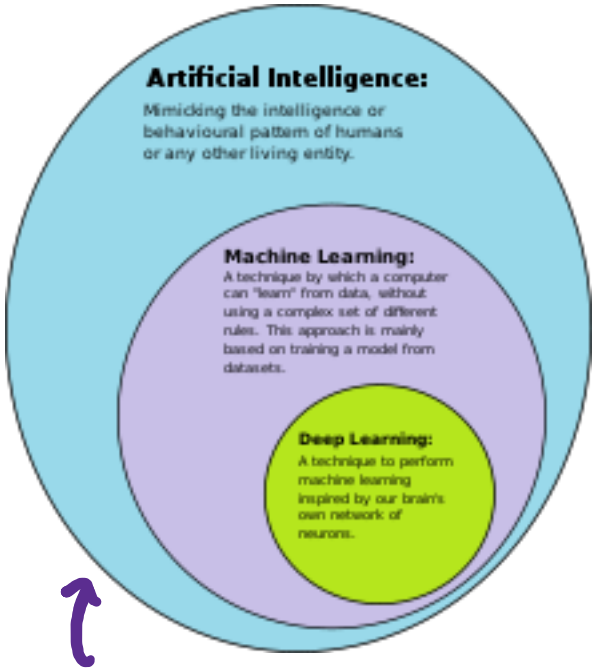
The Bigger Picture

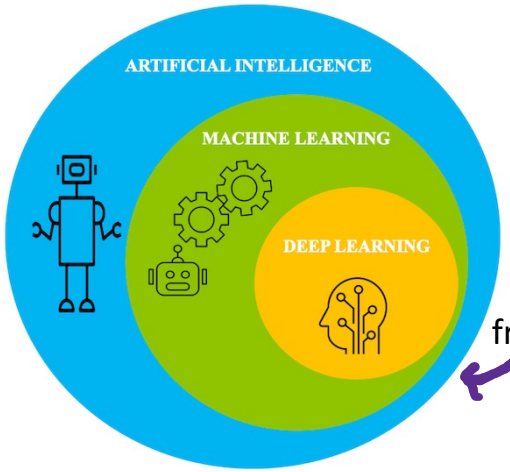
from https://builtin.com/machine-learning/what-is-deep-learning





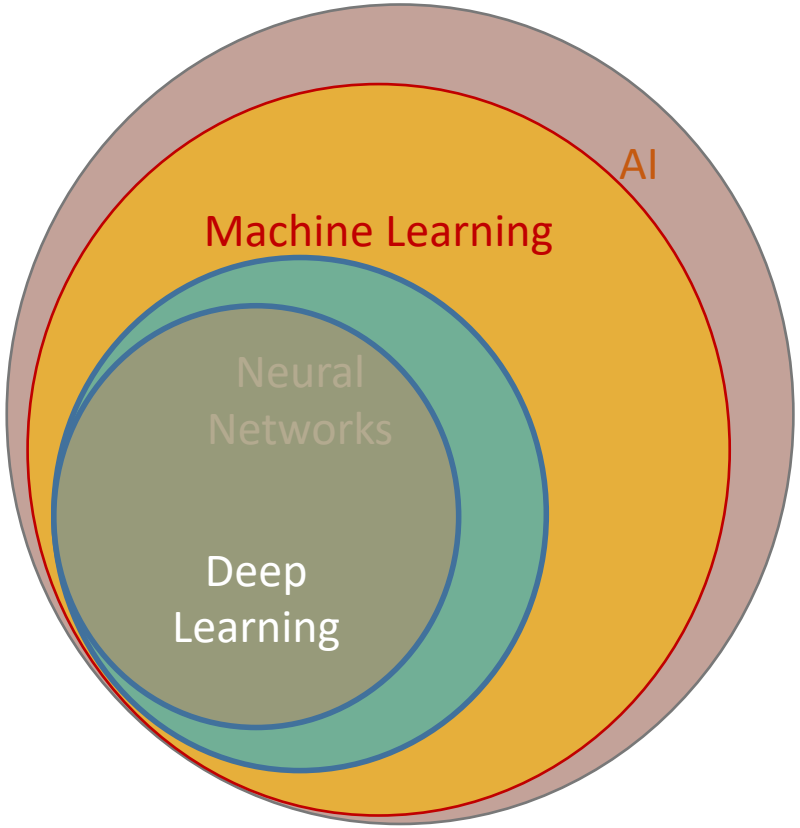
from https://www.edureka.co/blog/ai-vs-machine-learning-vs-deep-learning/

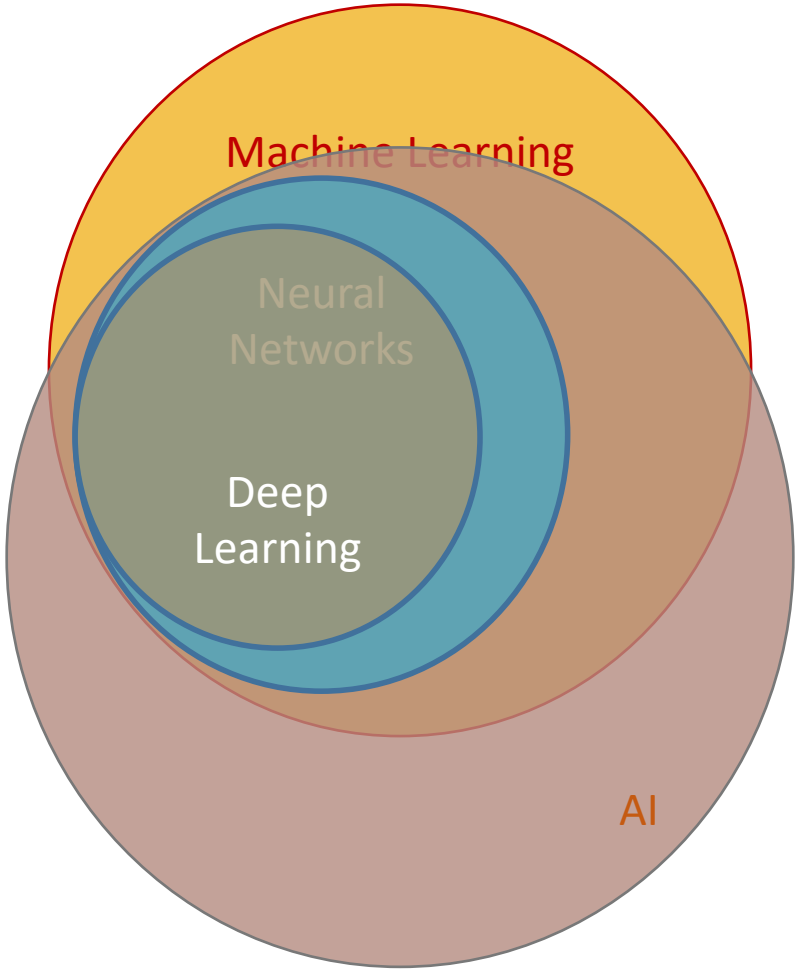


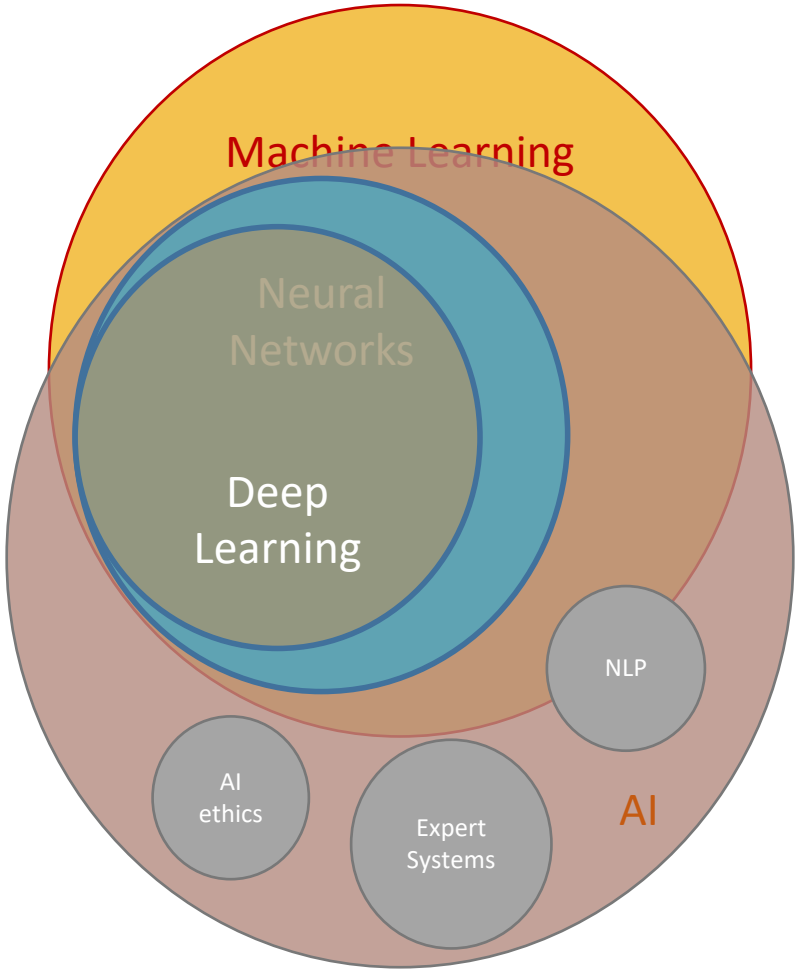


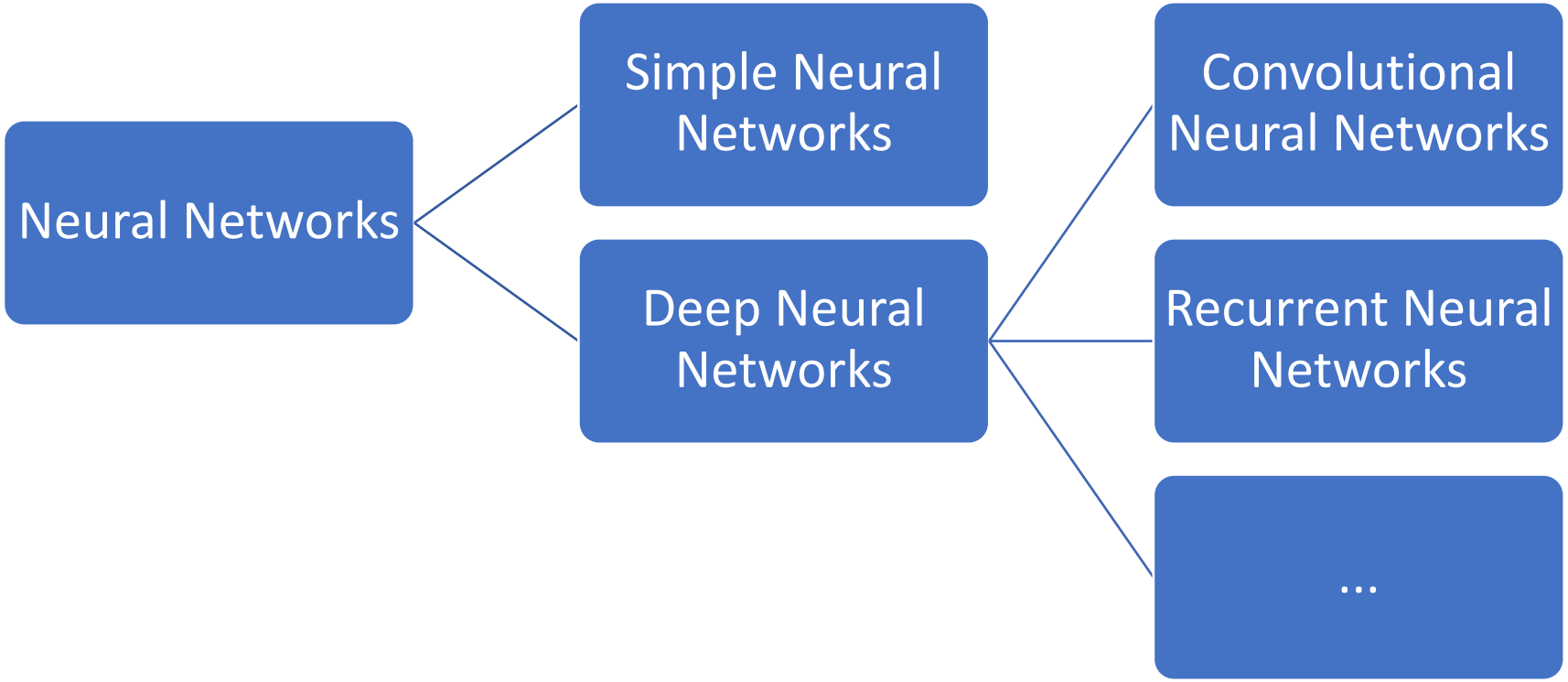
from https://developer.ibm.com/articles/an-introduction-to-deep-learning/

from https://commons.wikimedia.org/wiki/File:AI-ML-DL.svg

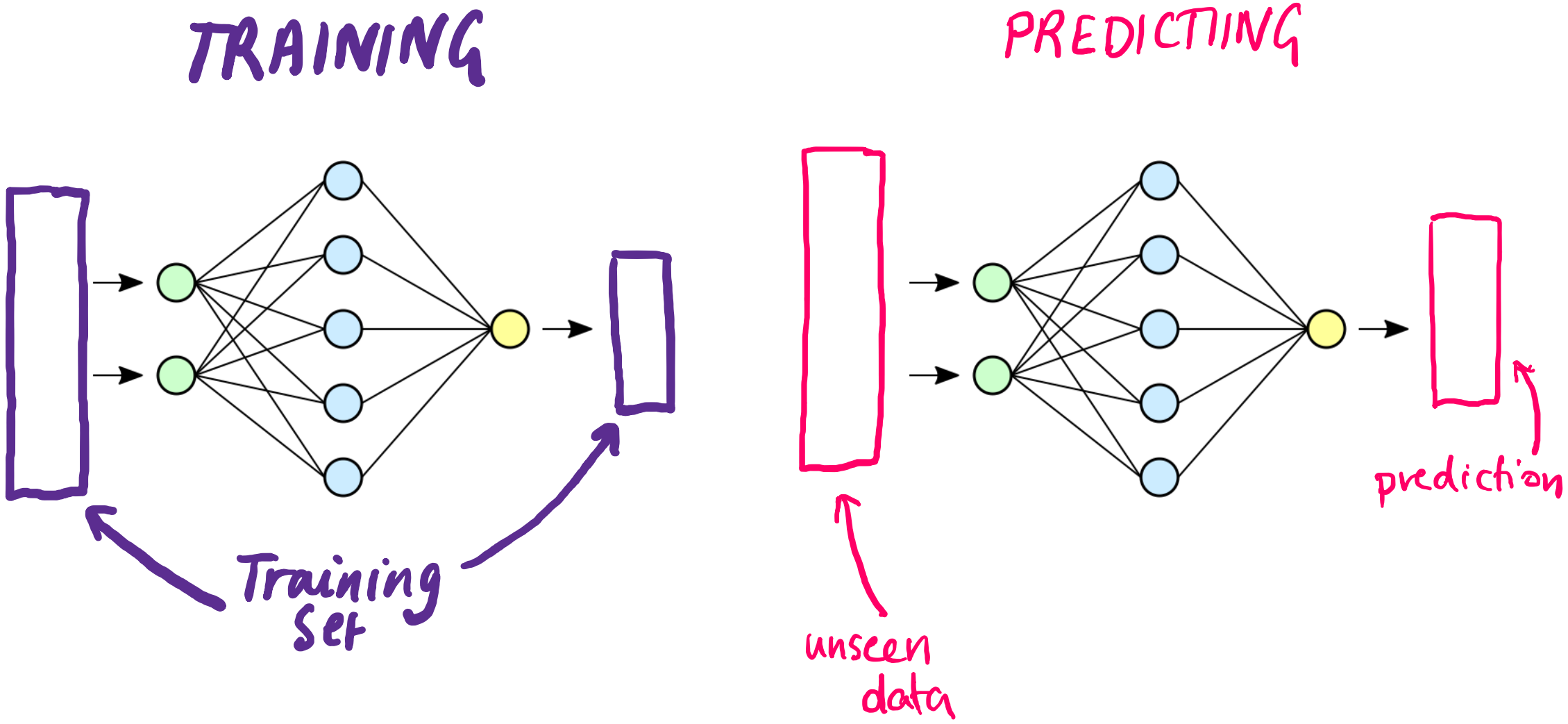

AI
Machine Learning
Neural Networks
Deep Learning


Machine Learning
Neural Networks
Deep Learning
AI

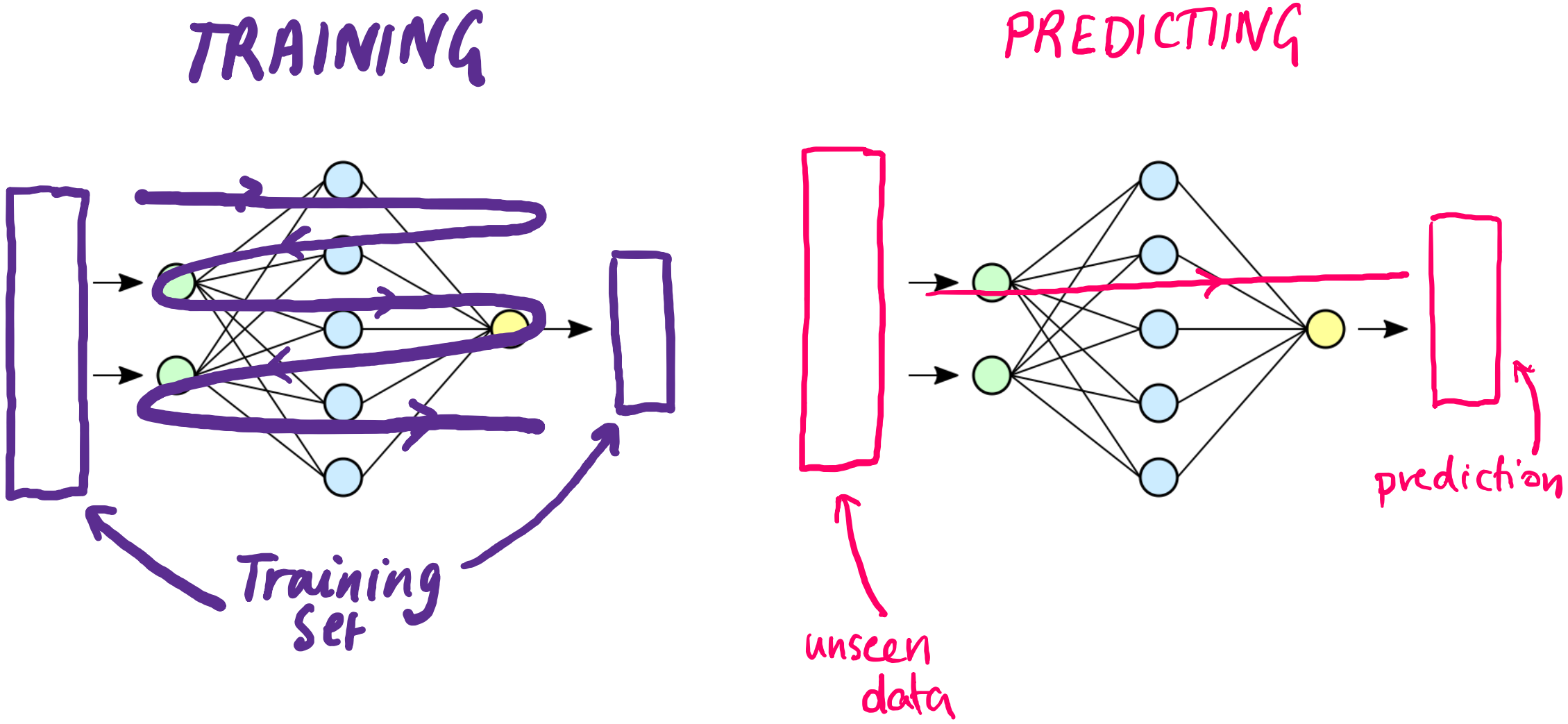

Machine Learning
Neural Networks
Deep Learning
NLP
AI ethics
Expert Systems
AI


Neural Networks
Simple Neural Networks
Deep Neural Networks
Convolutional Neural Networks
Recurrent Neural Networks
...

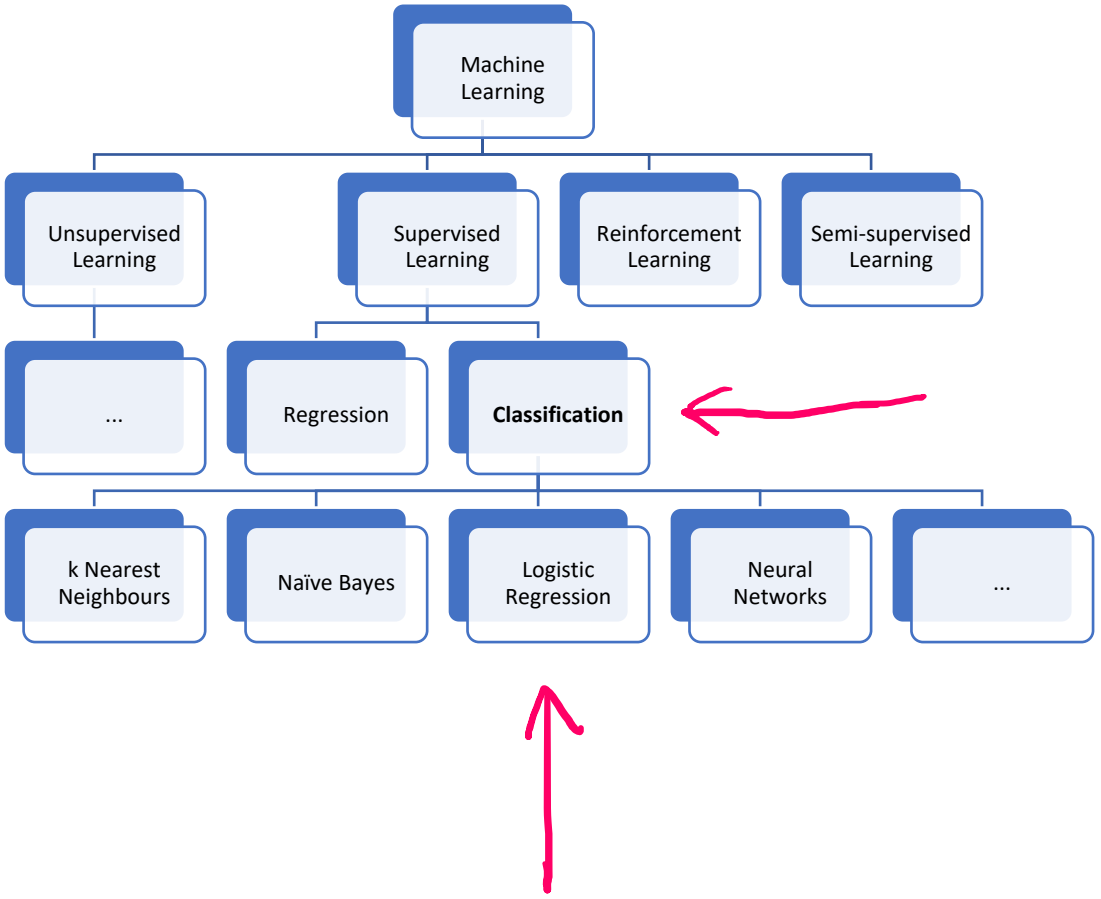
What is a Deep Neural Network?

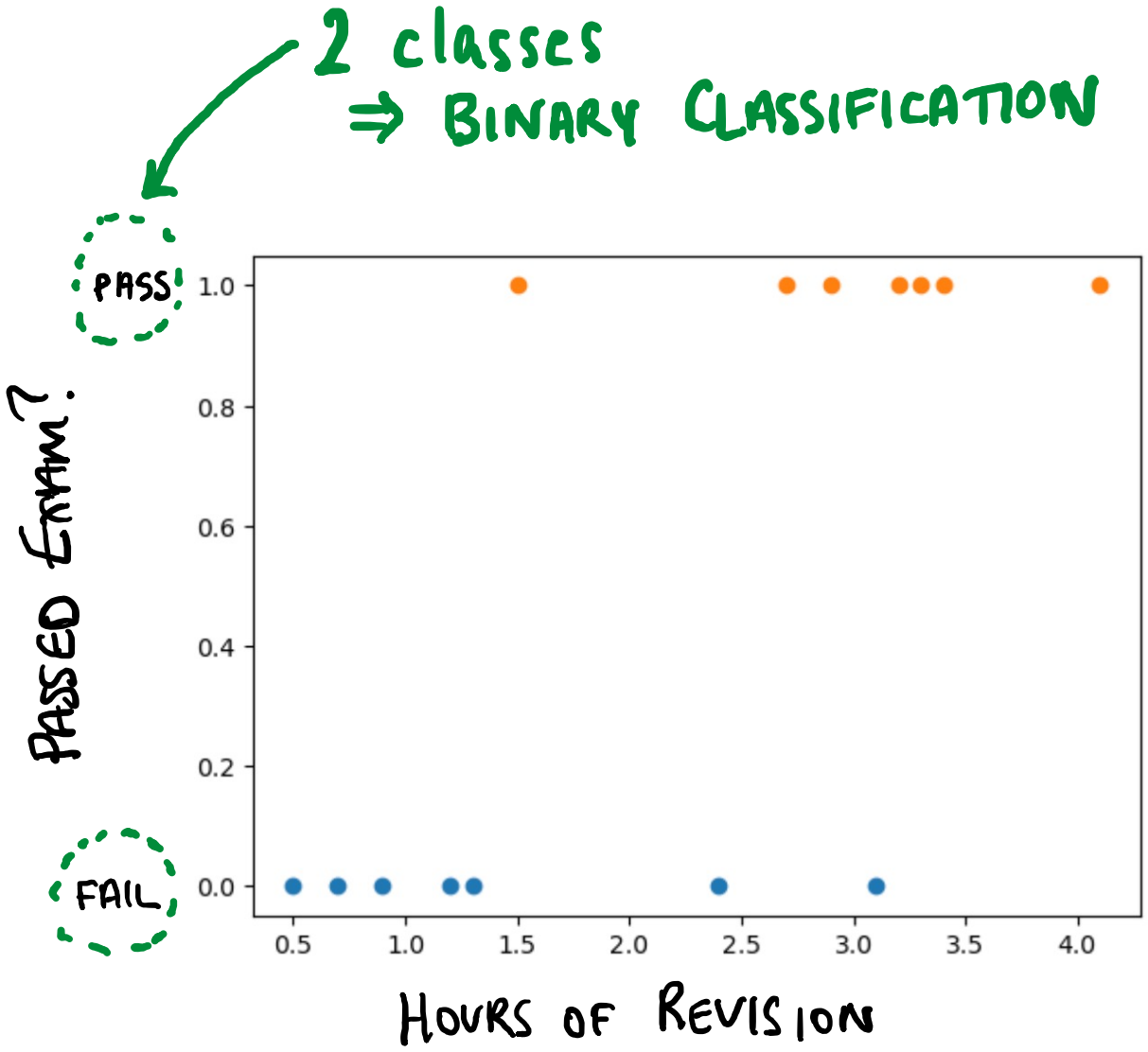


What is a Deep Neural Network?



A simpler problem: Logistic Regression...

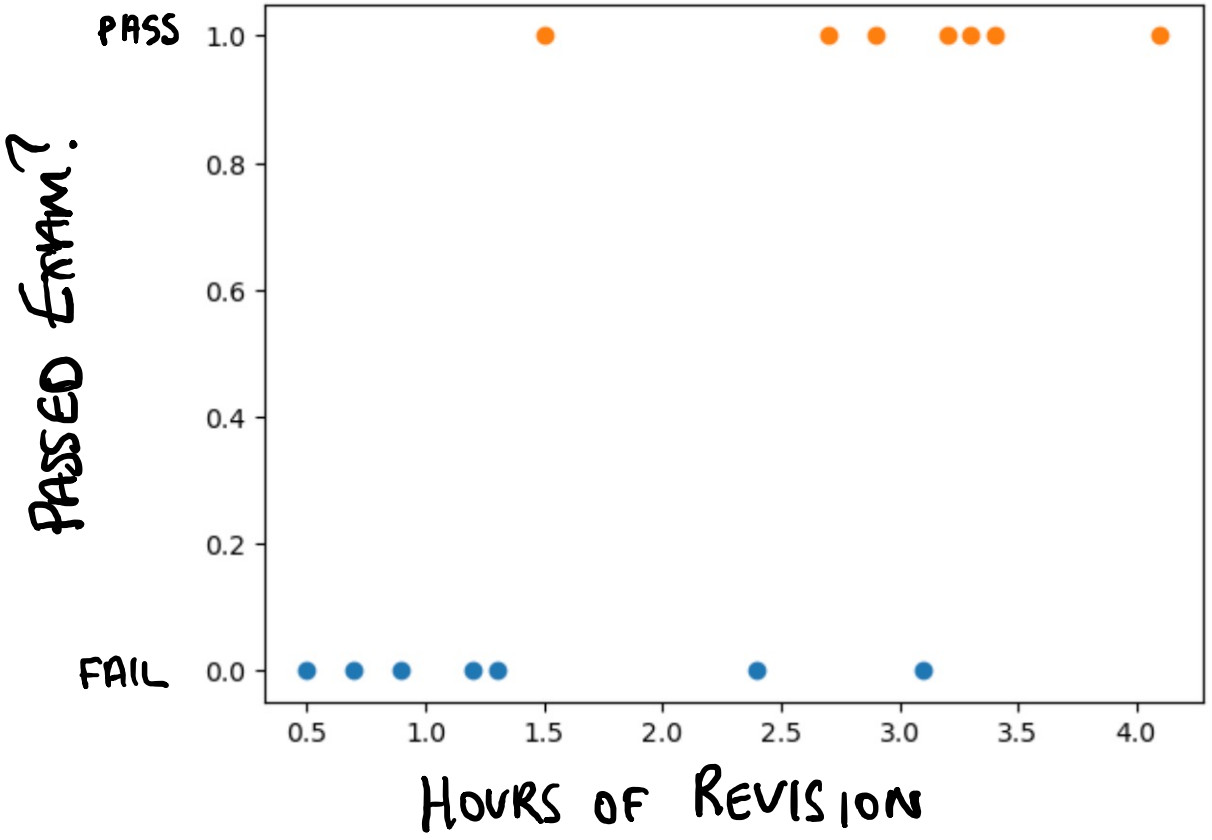


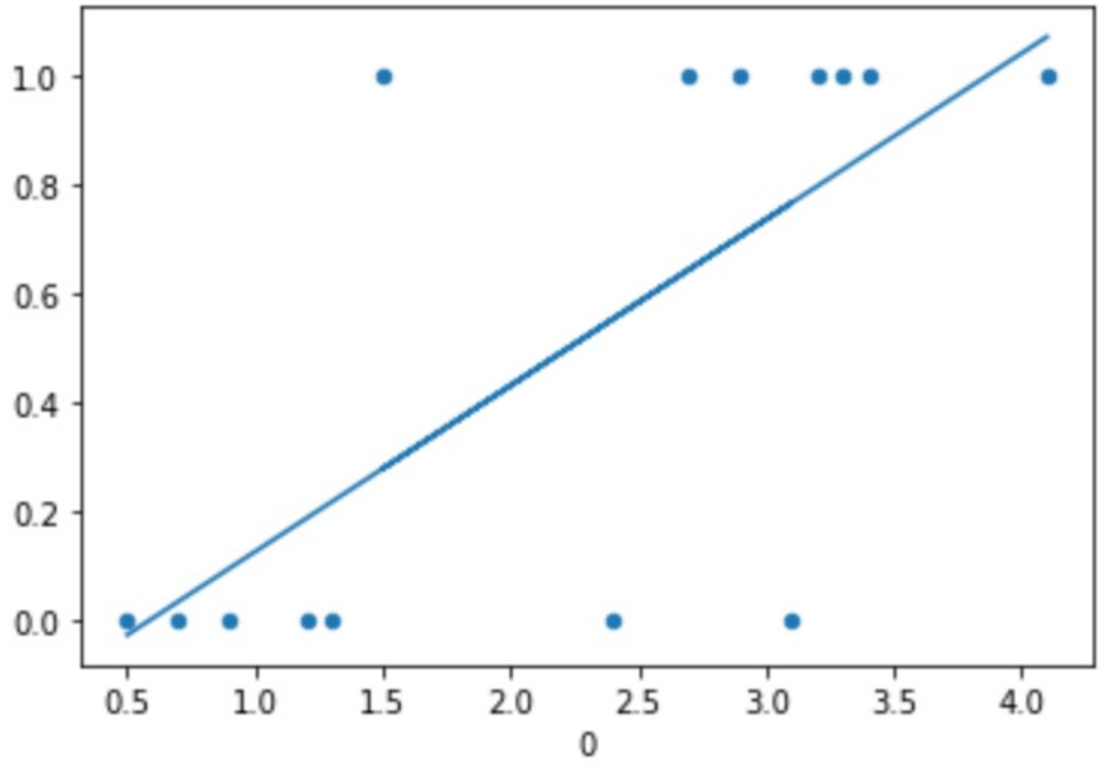


Logistic Regression

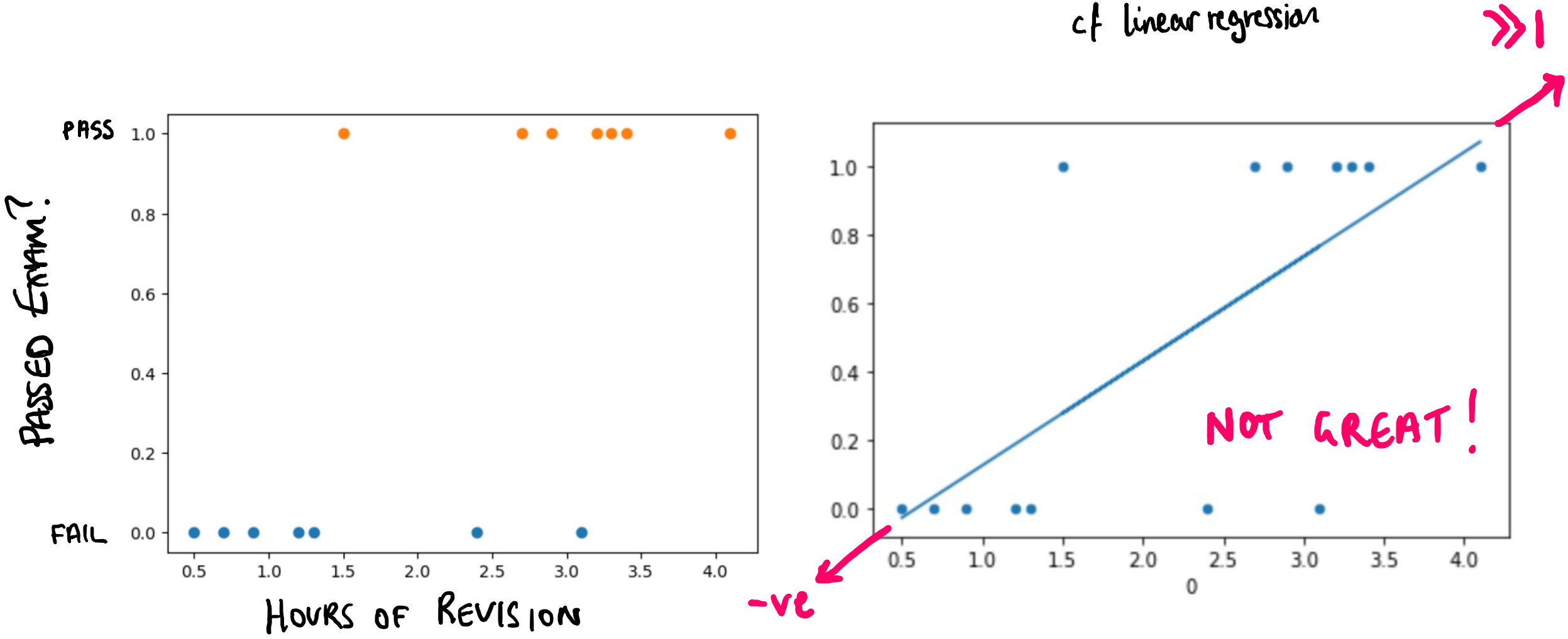
epcc

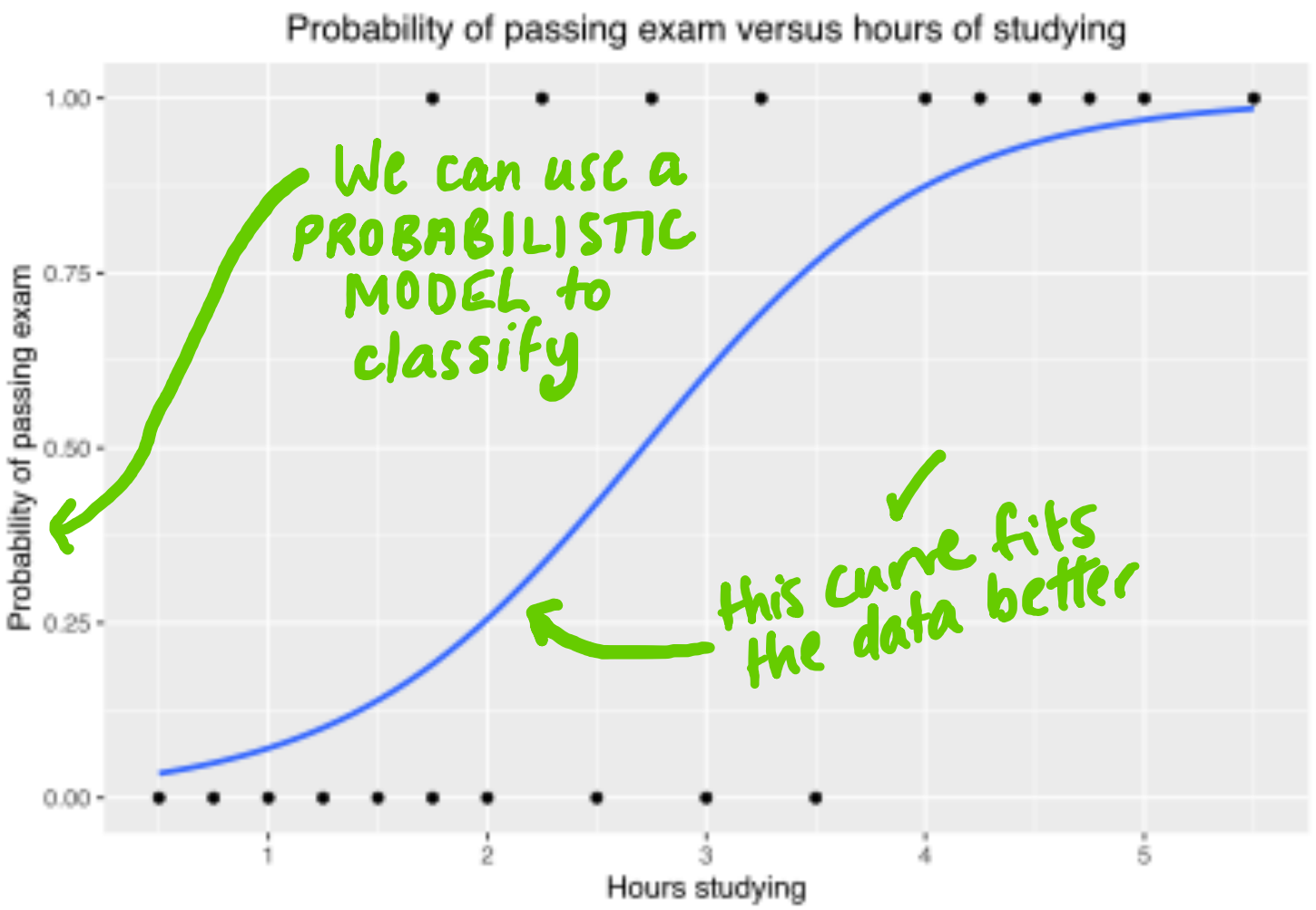
cf linear regression

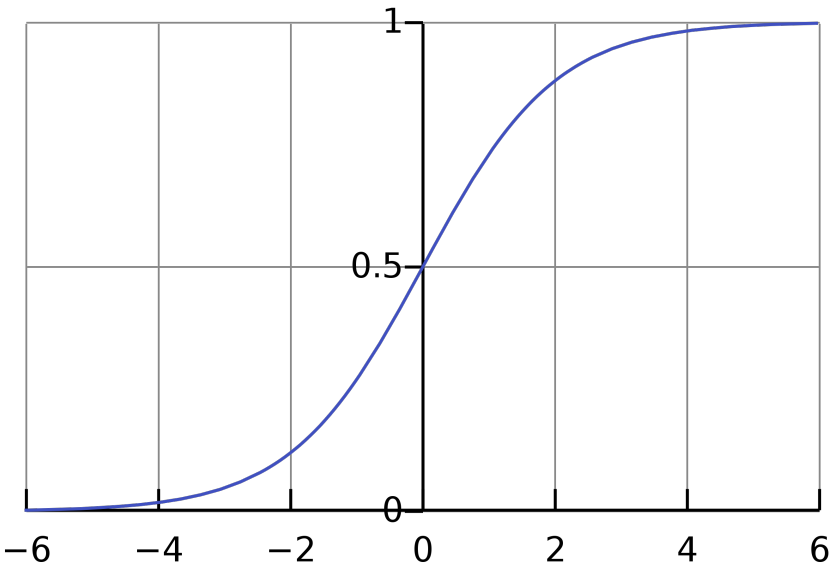




Let's go back to Logistic Regression...

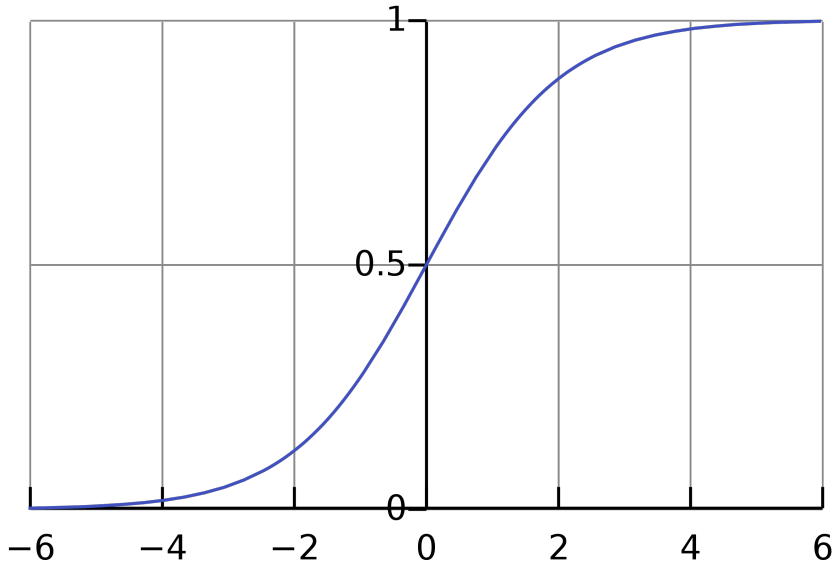






$$f(z) = \frac{1}{1 + e^{-z}}$$

The logistic function



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

- $z = \theta^{\mathrm{T}} x$
- $= \begin{pmatrix} \theta_0 & \theta_1 & \theta_2 & \cdots \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \end{pmatrix}$
- $= \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots$
- So the parameters that you fit define this *linear* equation (which is why this is a generalised *linear* model)

- Classify as ‘1’ if f(z) > 0.5

$$f(z) = \frac{1}{1 + e^{-z}} > 0.5$$

$$\Rightarrow \quad 1 > (1 + e^{-z}) \times 0.5$$

$$1 > 0.5 + 0.5e^{-z}$$

$$\Rightarrow \quad 0.5 > 0.5e^{-z}$$

$$\Rightarrow \quad 1 > e^{-z}$$

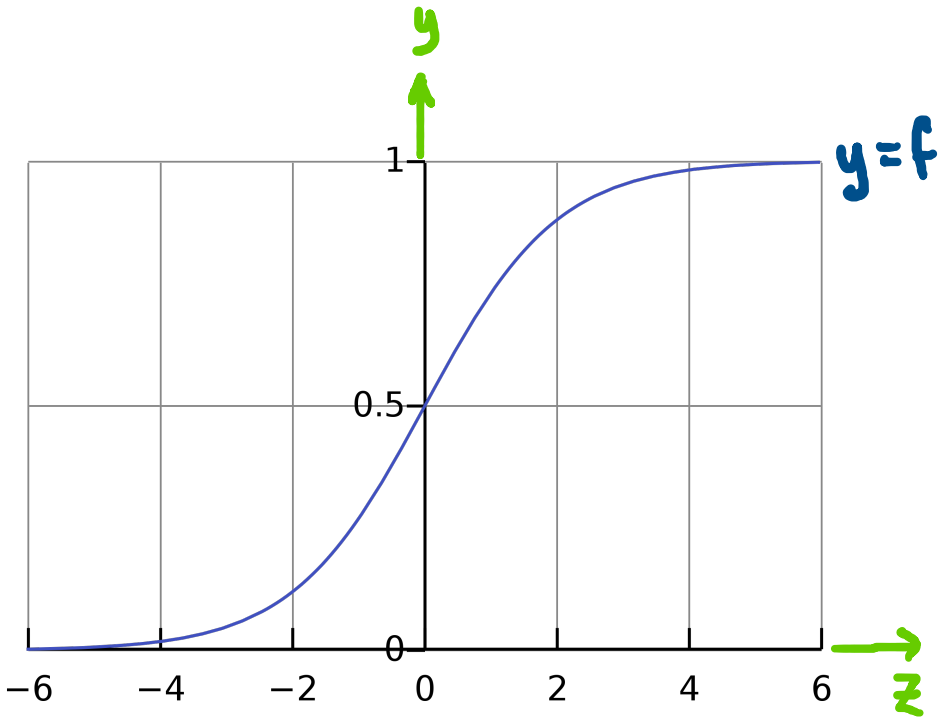
$$\Rightarrow \quad \ln(1) > \ln(e^{-z})$$

$$\Rightarrow \quad 0 > -z$$

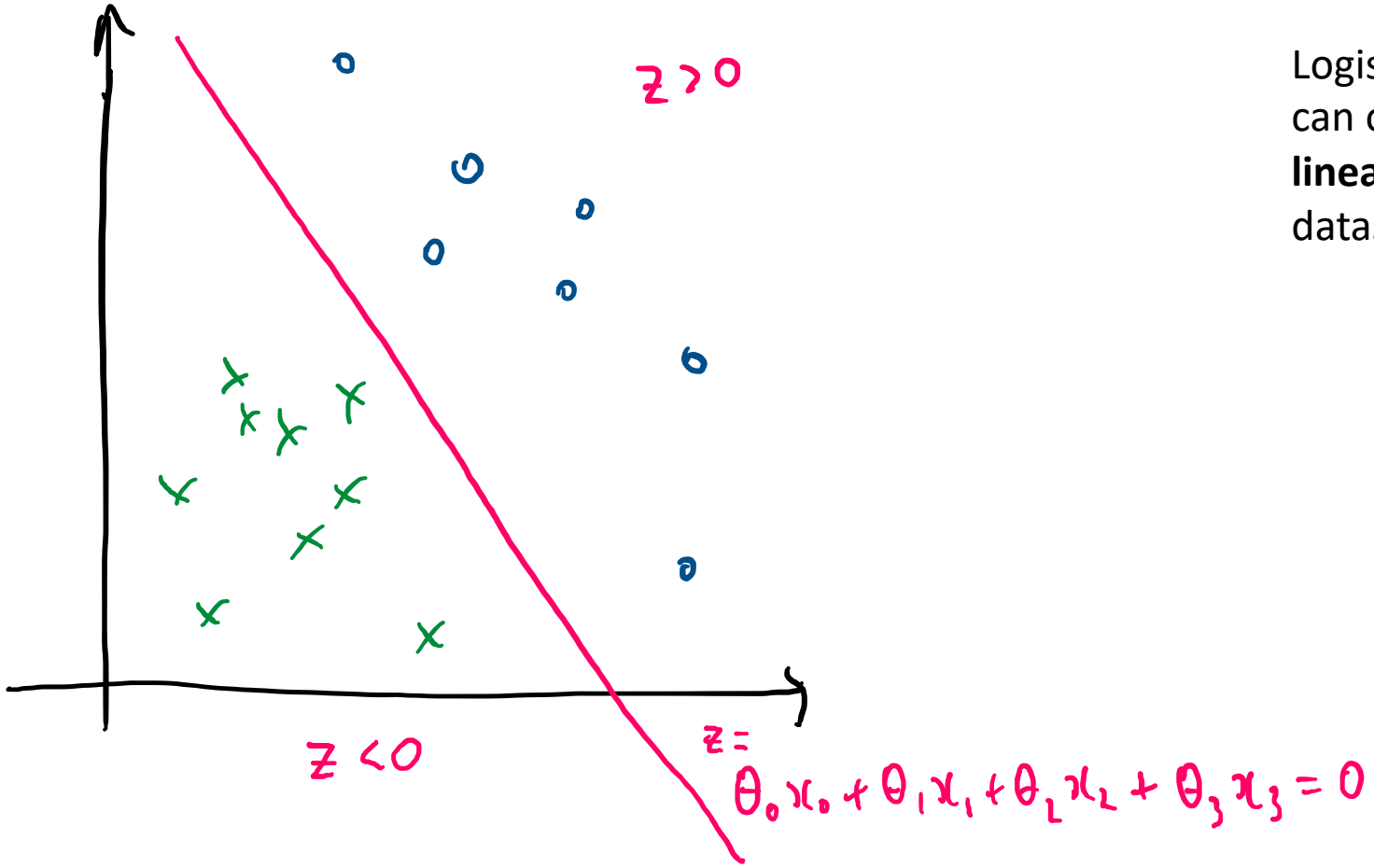
$$\Rightarrow \quad z > 0$$

$$\Rightarrow \quad \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots > 0$$

equation of a hyperplane



$$y = f(z) = \frac{1}{1 + e^{-z}}$$



Logistic Regression can only deal with **linearly separable** datasets

Choosing a Cost Function

epcc

- Recall, for linear regression:

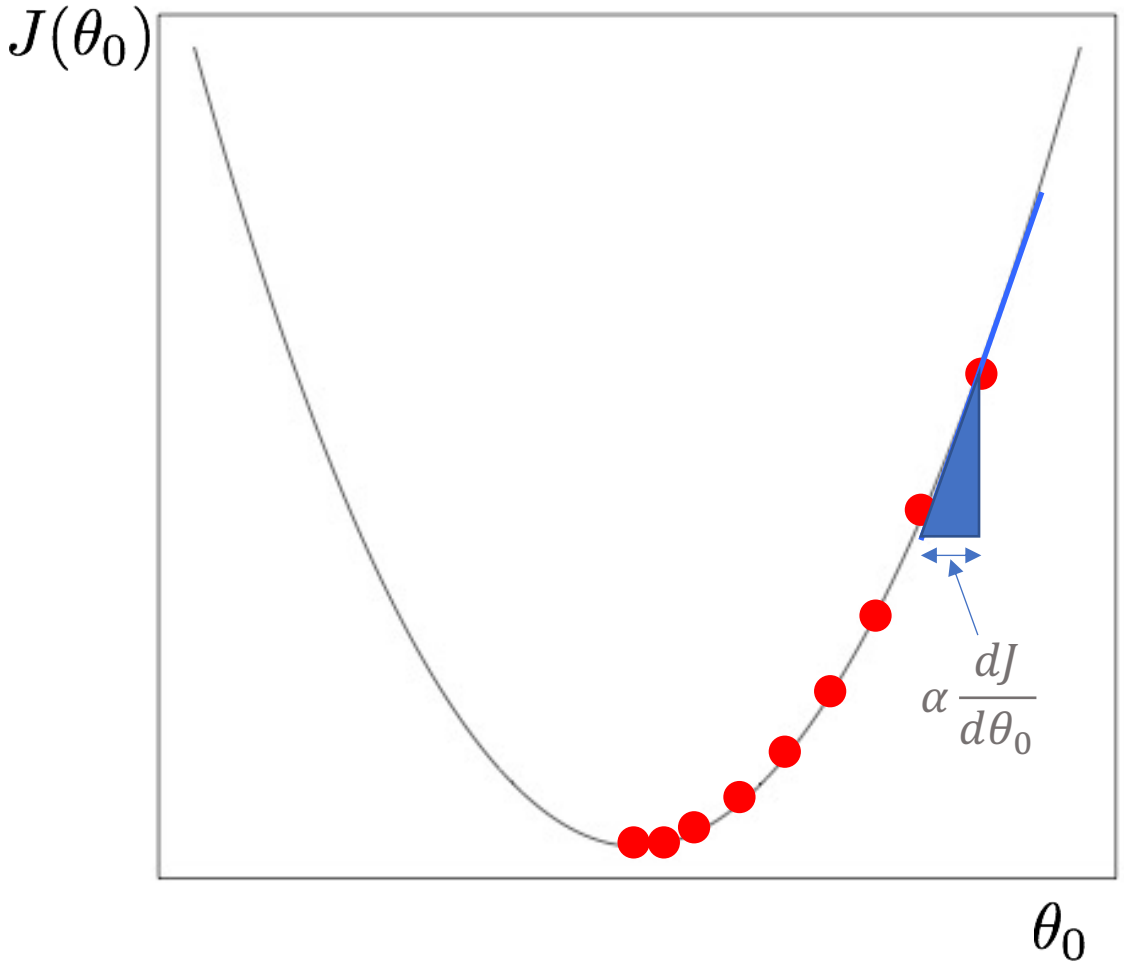
$$L = \sum_{i=1}^{n} \left(y_i - f(x_i)\right)^2$$

- For logistic regression:

$$L = \sum_{i=1}^{n} y_i \log f(x_i) + (1 - y_i)\log(1 - f(x_i))$$

  - This expression can be derived under certain conditions by making the assumption that the data are independent & identically distributed. In practice, it can be used more generally

Gradient descent with one parameter



$$\theta_0 := \theta_0 - \alpha \frac{\mathrm{d}}{\mathrm{d}\theta_0} J(\theta_0)$$

How a neural network is built, trained and used

- Construct: Decide on the architecture of the network
  - How many layers, how the layers are connected, choice of **activation function**
  - Weights are initialised [equivalent to picking your initial straight line in linear regression]
- Train:
  - Repeat until you're happy that the network is trained:
    - Feed data into the neural network
    - Forward Pass: Make a prediction based on the current weights and calculate the **cost function** [analogously to how we calculated a cost function in linear regression]
    - Backward Pass, *aka "Back Propagation"* to find the *derivative* of the **cost function**
    - Update the weights and biases using the derivatives that you've calculated

this iterative approach is the **gradient descent** optimisation algorithm

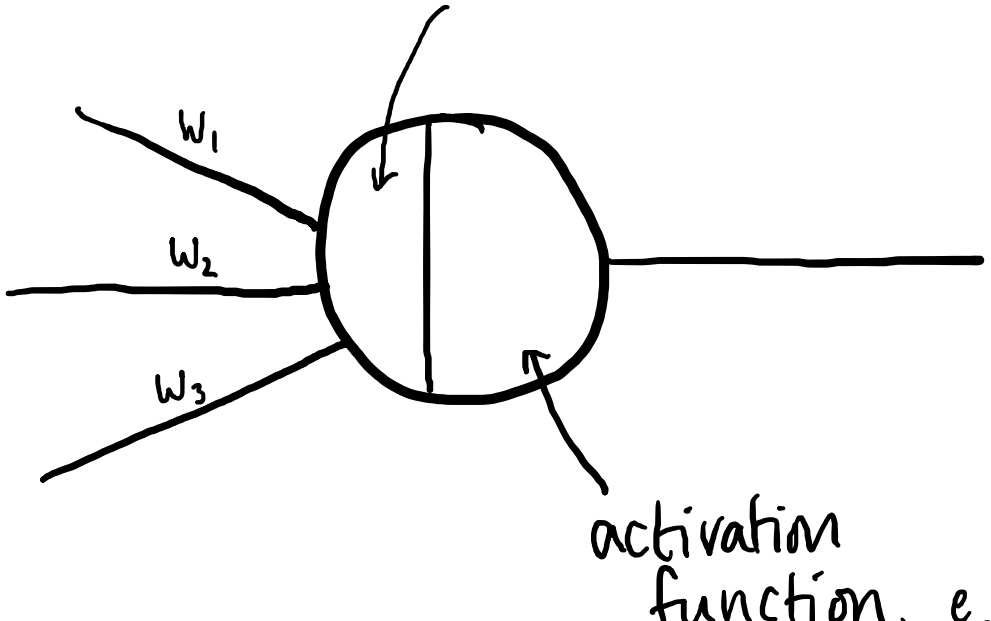
- Predict:
  - A single forward pass through the network

- The following explanation is inspired to a great extent by Andrew Ng's very good approach to teaching Neural Networks, which he uses, for example on his courses on Coursera, such as the course *Neural Networks and Deep learning,* available at https://www.coursera.org/learn/neural-networks-deep-learning.

- Material is also inspired by lectures given in the past by EPCC and the School of Informatics at the University of Edinburgh.

A "neuron"

$$z = w^T x + b \quad (= \theta^T x)$$

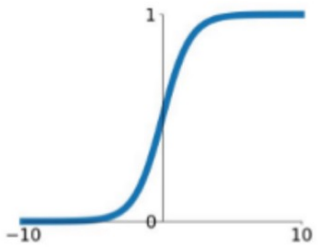


$$a = \sigma(z) = \frac{1}{1+e^{-z}}$$

Common Activation Functions

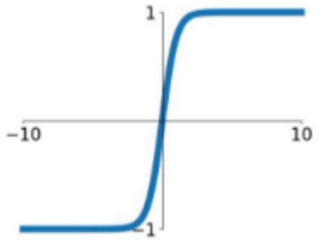
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



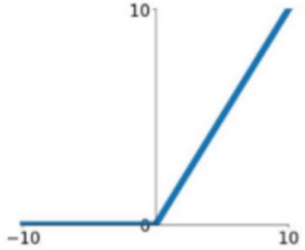
tanh

$$\tanh(x)$$



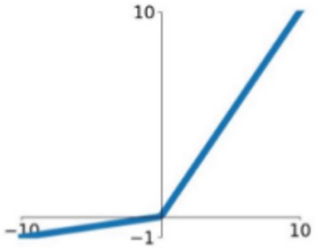
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

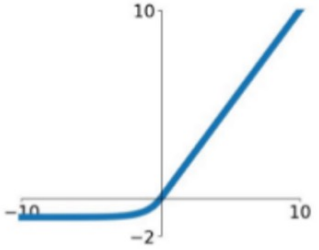


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

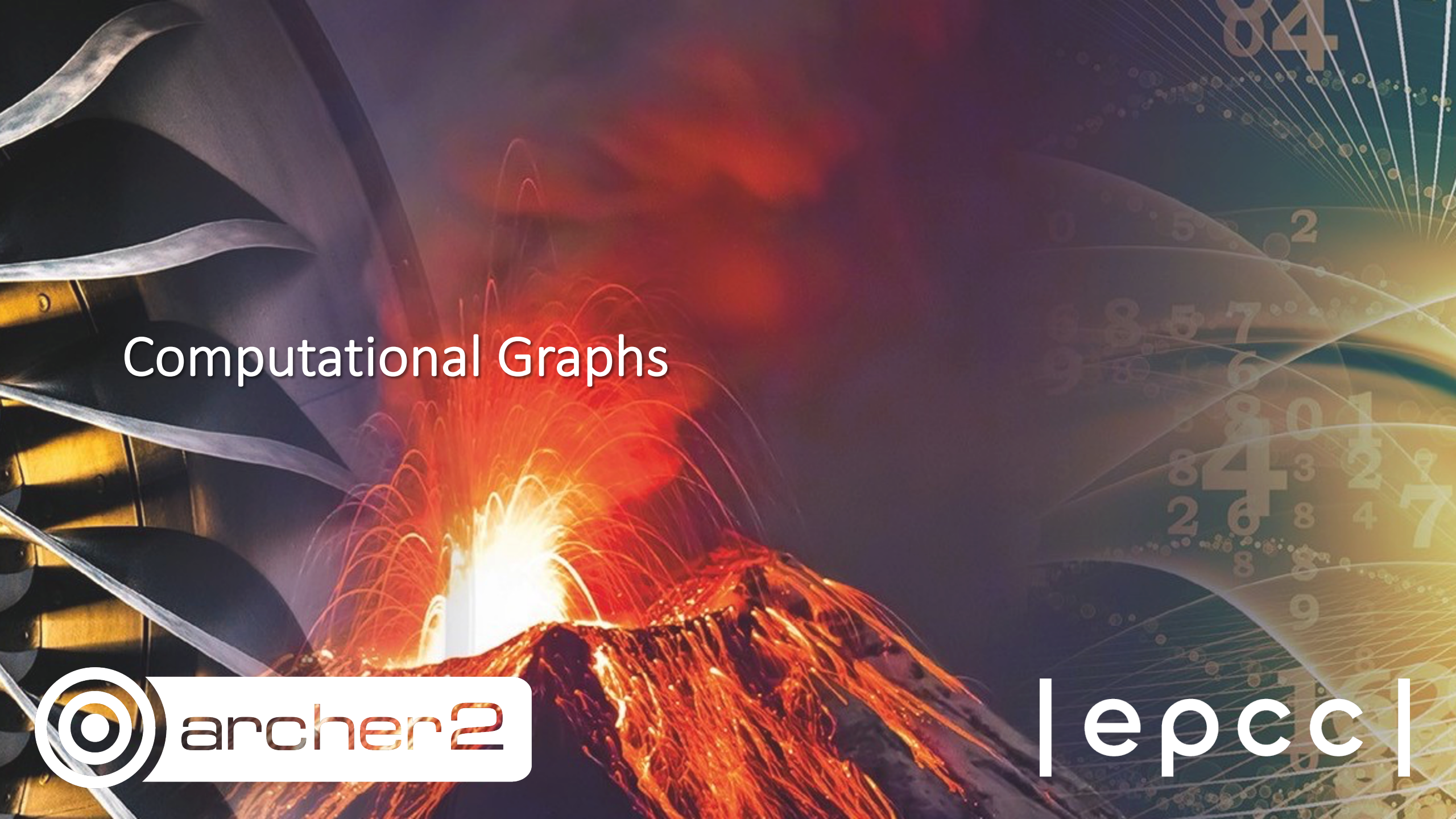
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



From https://towardsdatascience.com/complete-guide-of-activation-functions-34076e95d044

Computational Graphs

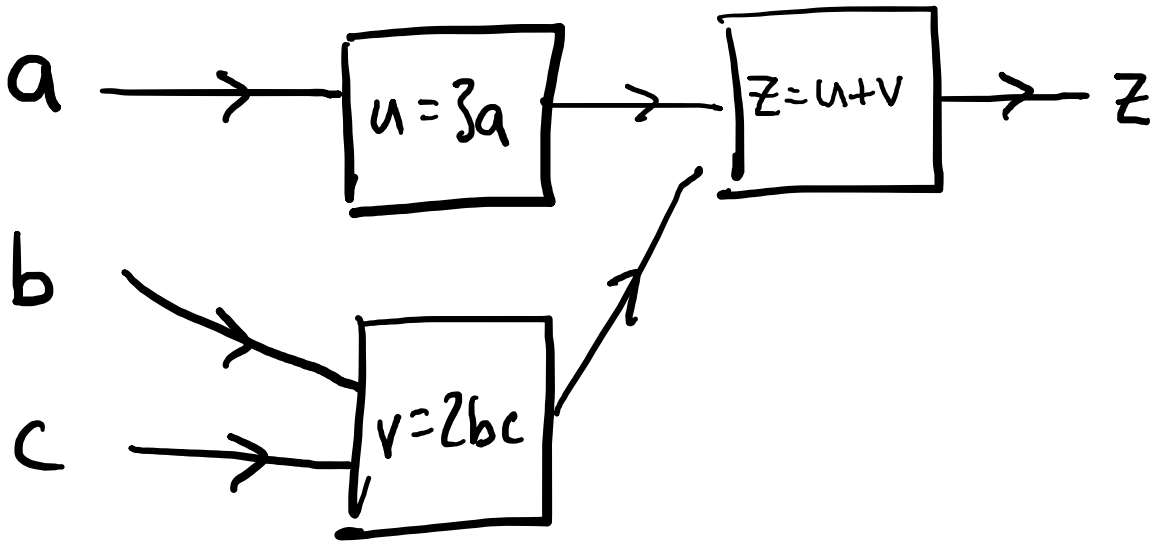


Computation Graphs to illustrate Propagation in Neural Networks

epcc

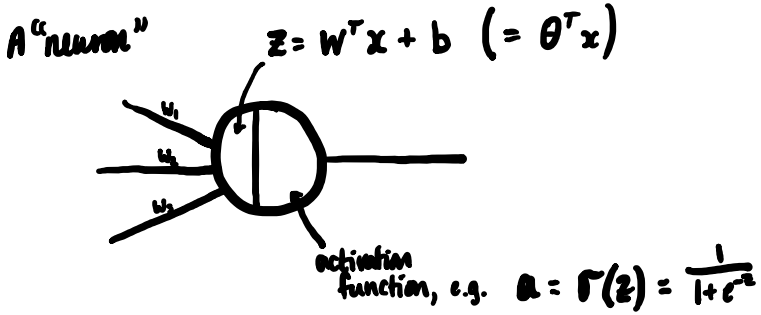
$$z = 3a + 2bc \rightarrow \begin{cases} z = u + v \\ u = 3a \\ v = 2bc \end{cases}$$

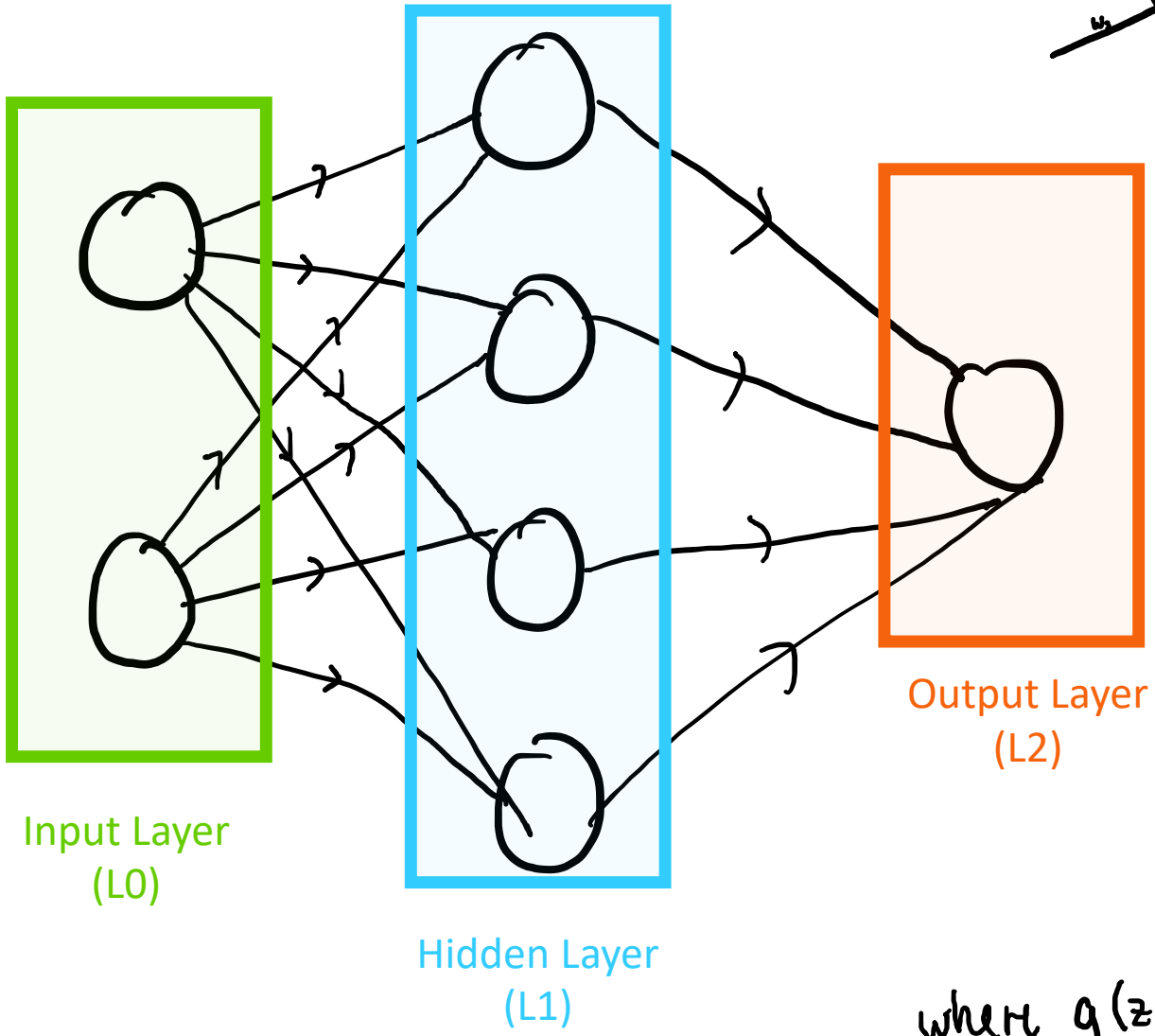
$$z = 3a + 2bc \rightarrow \begin{cases} z = u + v \\ u = 3a \\ v = 2bc \end{cases}$$



Forward Propagation

Forward Propagation





$$z^{[1]} = W^{[1]} x + b^{[1]}$$

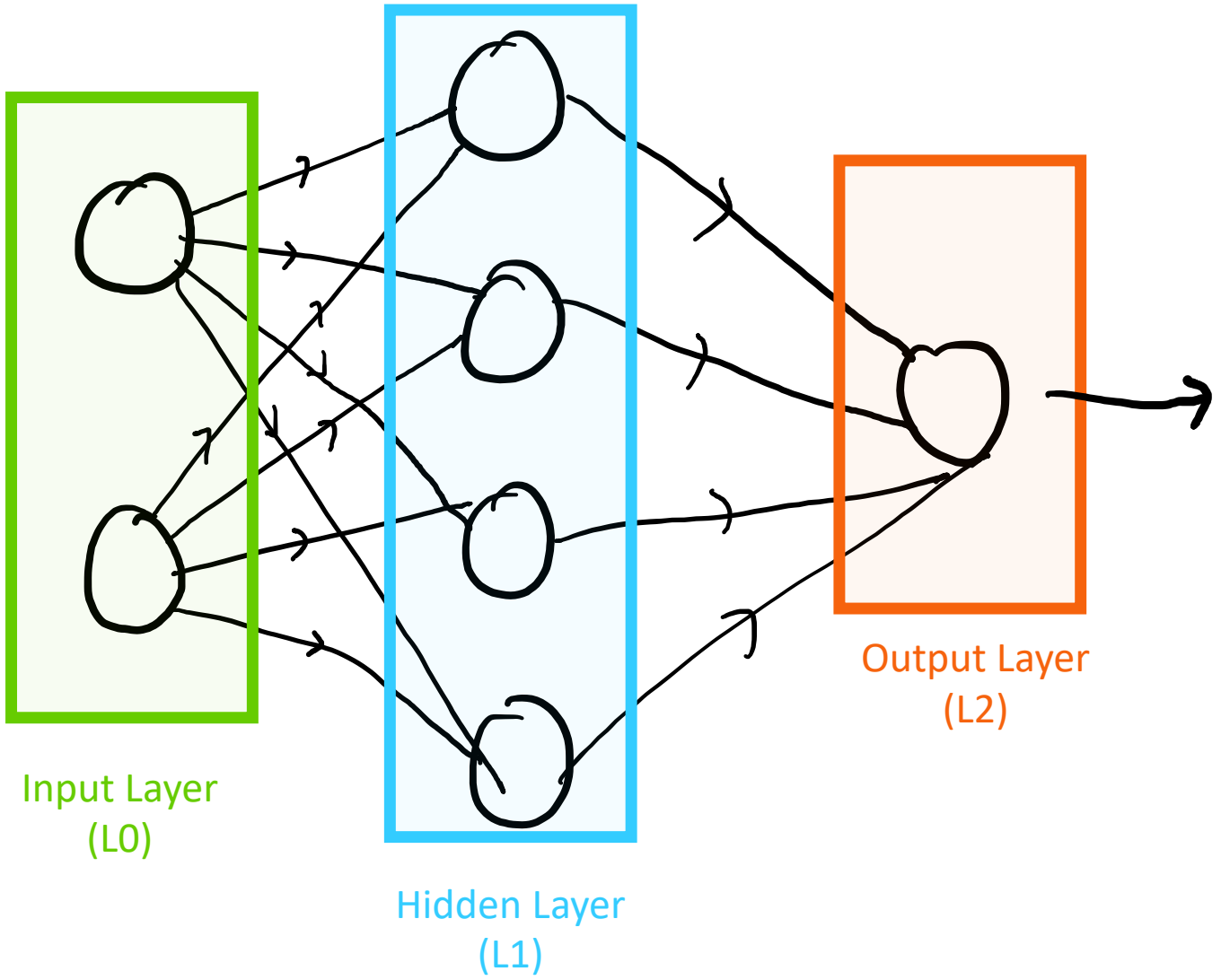
$$a^{[1]} = g(z^{[1]})$$

$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$

$$a^{[2]} = g(z^{[2]})$$

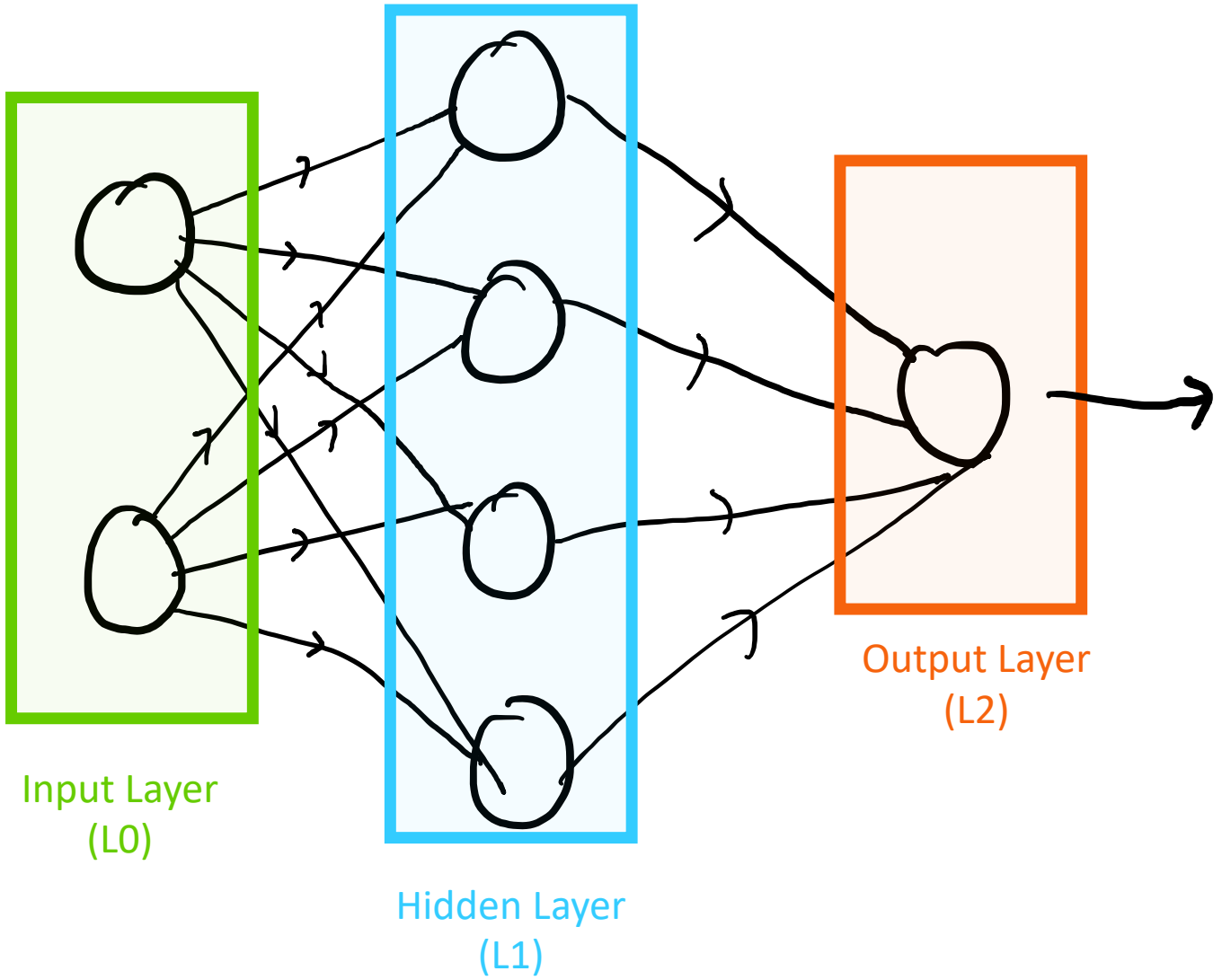
where $g(z) = \begin{cases} \sigma(z) \\ \dots \end{cases}$

Forward Propagation



So by the time we reach the end, we have calculated $\hat{y}$ and using just $\hat{y}$ and y, we can calculate $L(y, \hat{y})$

Forward Propagation



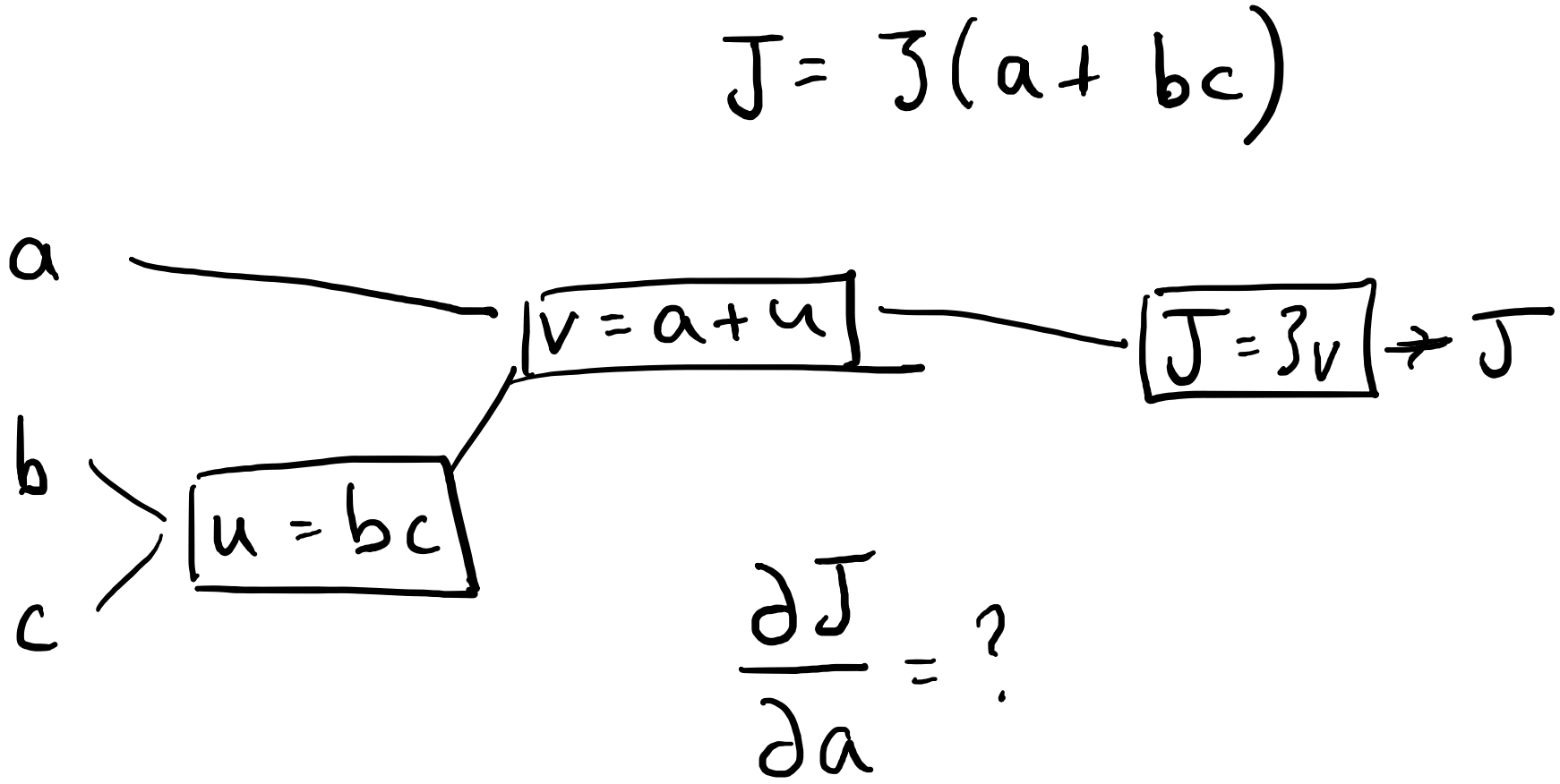
So by the time we reach the end, we have calculated $\hat{y}$ and using just $\hat{y}$ and y, we can calculate $L(y, \hat{y})$

... but for gradient descent, we need

$$\frac{\partial L}{\partial x_1}, \frac{\partial L}{\partial x_L}, \ldots$$

Back Propagation

Derivatives of a Cost Function



Calculus to the rescue!

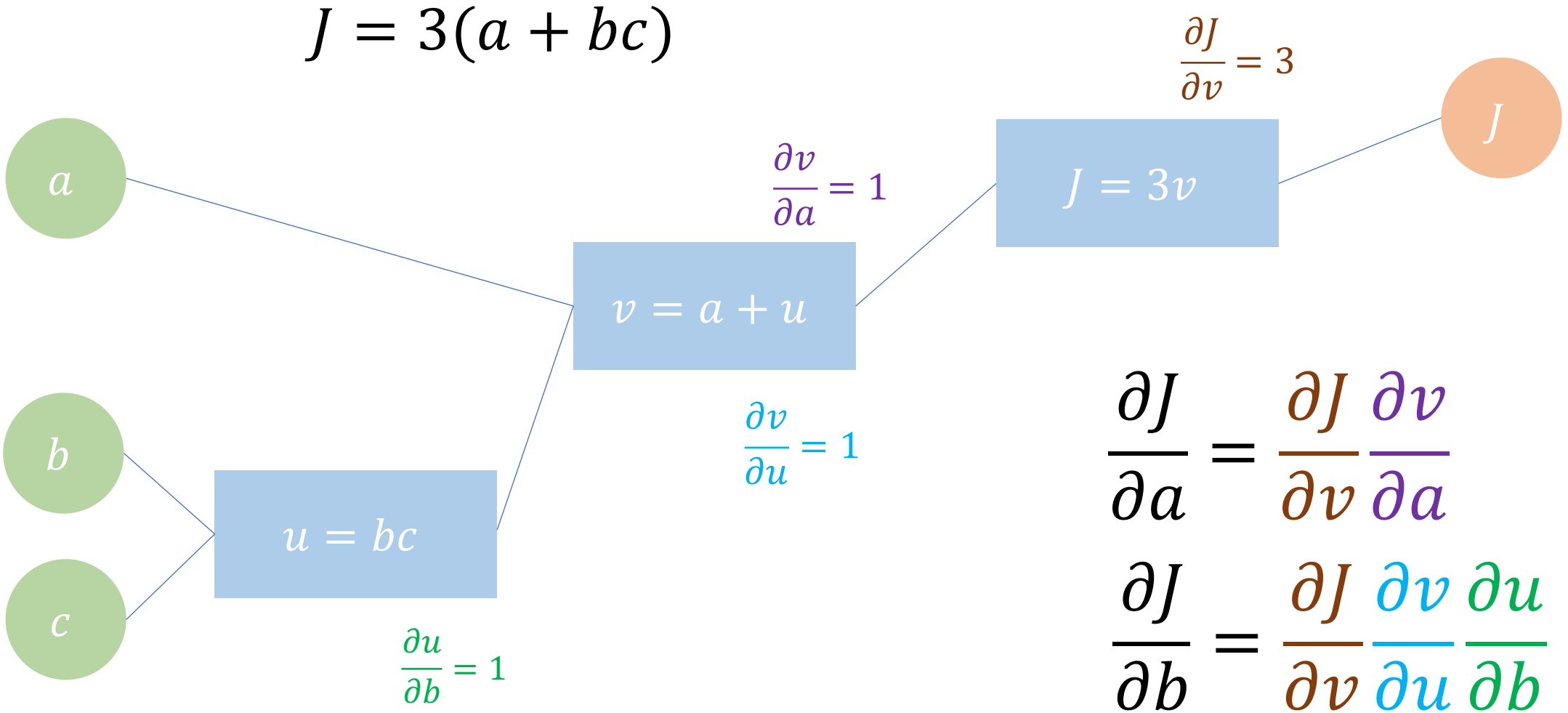
epcc

- The chain rule:
  - If u is some function of x and y is some function of u, then:

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$$

epcc

$$J = 3(a + bc)$$



$$\frac{\partial J}{\partial a} = \frac{\partial J}{\partial v}\frac{\partial v}{\partial a}$$

$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial v}\frac{\partial v}{\partial u}\frac{\partial u}{\partial b}$$


Some general points…
archer2
epcc

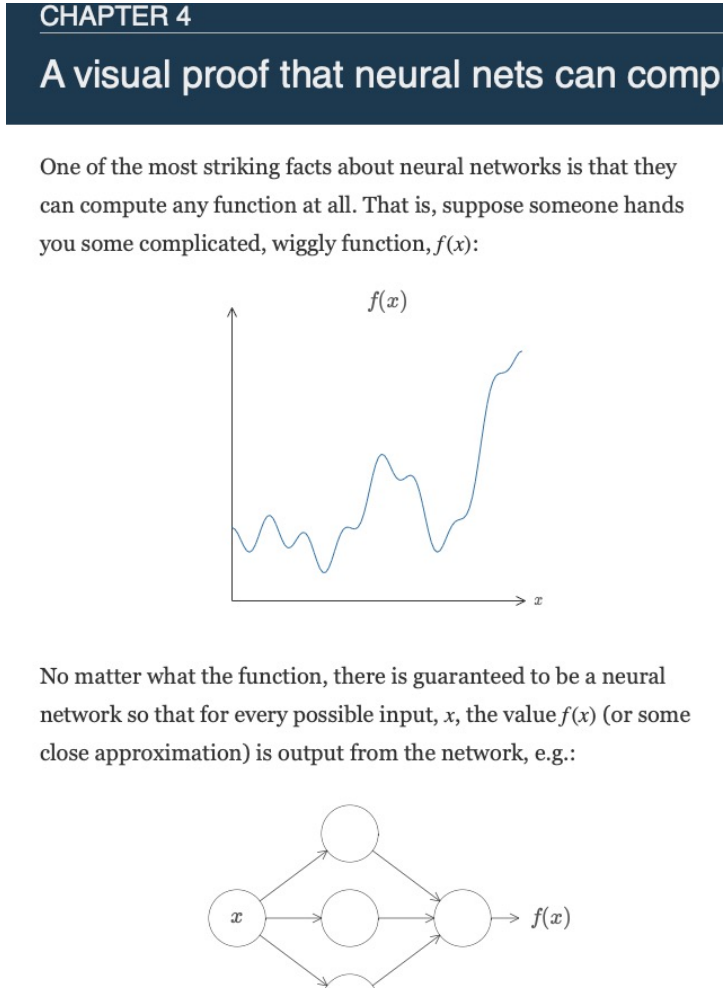
Different ways to train the model

- Once you have the idea of a NN and means to perform **forward propagation** and **back propagation** you can create variations of the approach, e.g., how you use your training set

- Do you feed in all the data at once?
  => Batch Gradient Descent
- Do you feed it in in batches?
  => Mini-batch gradient descent
- Do you select instances at random for each propagation?
  => Stochastic gradient descent

Neural networks can represent any function

- Proof of this beyond the scope of this course, but there's a nice visual "proof" that neural networks can represent any function in
  - Michael Nielson, Neural Networks and Deep Learning, chapter 4, available at http://neuralnetworksanddeeplearning.com/chap4.html



CHAPTER 4

A visual proof that neural nets can comp

One of the most striking facts about neural networks is that they can compute any function at all. That is, suppose someone hands you some complicated, wiggly function, $f(x)$:

No matter what the function, there is guaranteed to be a neural network so that for every possible input, x, the value $f(x)$ (or some close approximation) is output from the network, e.g.:

Building on the idea of a fully-connected, feed-forward neural network

epcc

- What if we have nodes that aren't fully connected to every node in the previous layer?
  - You can create layers where nodes in layer i depend only on "nearby" nodes in layer $i-1$. If the nodes represent pixels, or parts of an image, this means that you can "look for" localised features in images
  - … which turns out to be useful for image classification
  - … and which motivates the use of **convolutional neural networks**
- What if we allow connections that can skip layers
  - You can create **residual networks** like ResNet
- What if we have edges between nodes that can feed back into earlier nodes?
  - This makes computation more tricky, but it turns out that this is the idea behind **recurrent neural networks** which are used, for example, in sequence prediction for tasks in **natural language processing**