

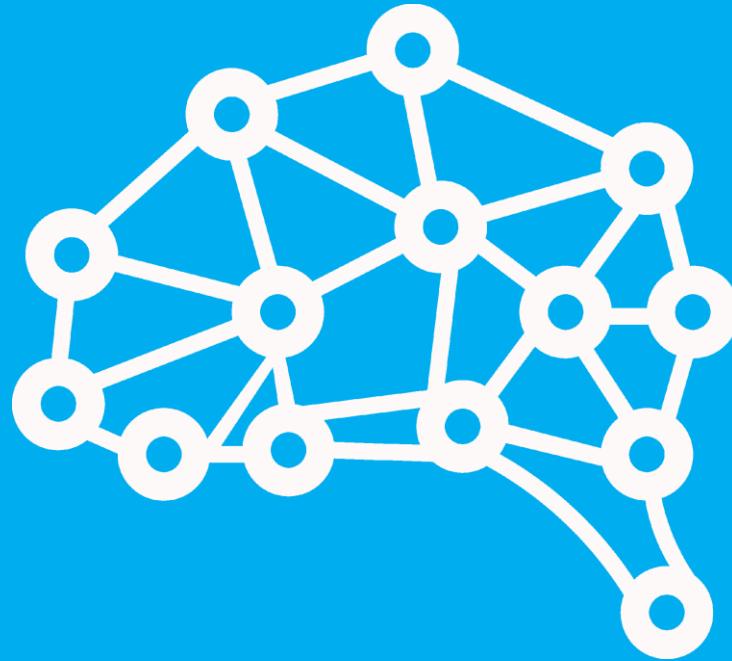
wbs

Warwick
Business
School

Data Science & Generative AI

Dr Michael Mortenson

Associate Professor (Reader)
Michael.Mortenson@wbs.ac.uk

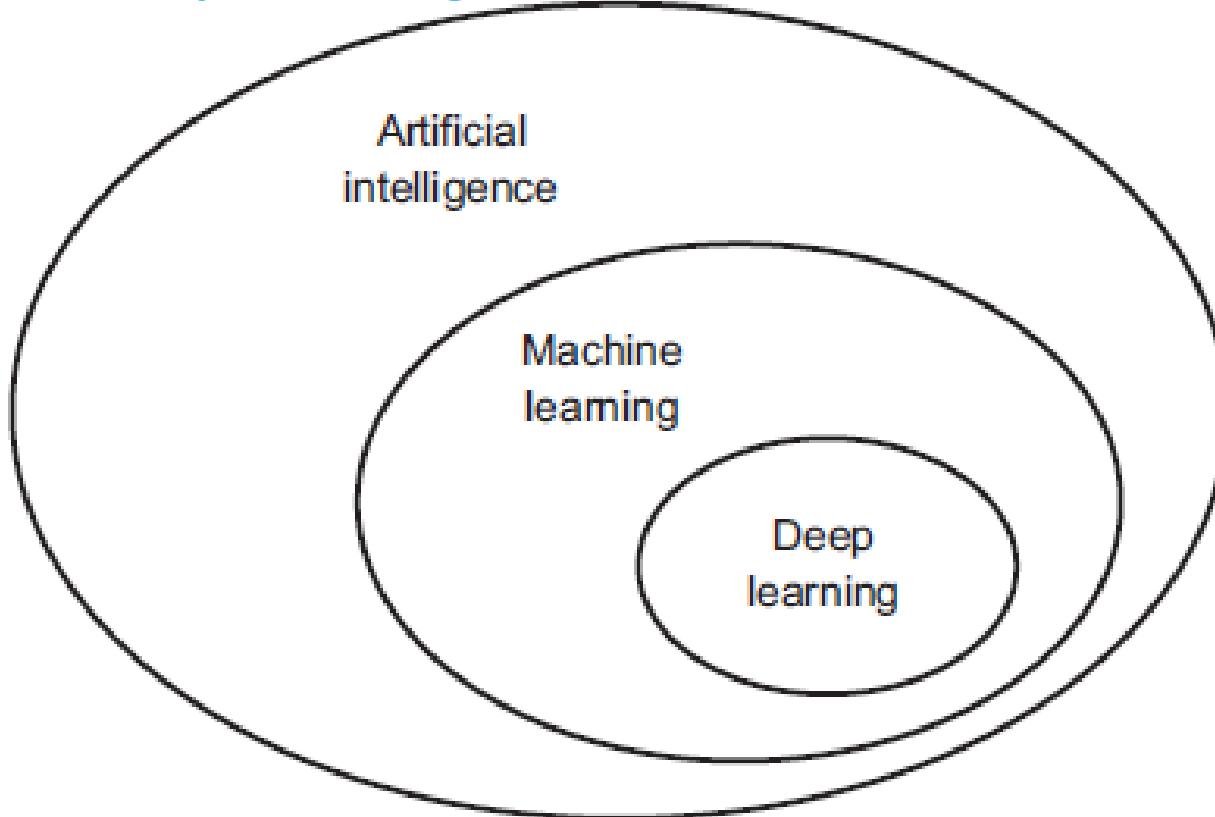


Session 6: Deep Learning & Neural Network Fundamentals

1.1 MDTGA



1.2 AI, ML & Deep Learning



1.3 Deep Learning & Data

Structured data

Databases

Semi-structured data

XML / JSON data
Email
Web pages

Unstructured data

Audio

Video

Image data

Natural language
Documents

1.4 Deep Learning & Parameters

$$Y = \underline{\alpha} + \underline{\beta}x + \varepsilon$$

= 2 learnable parameters



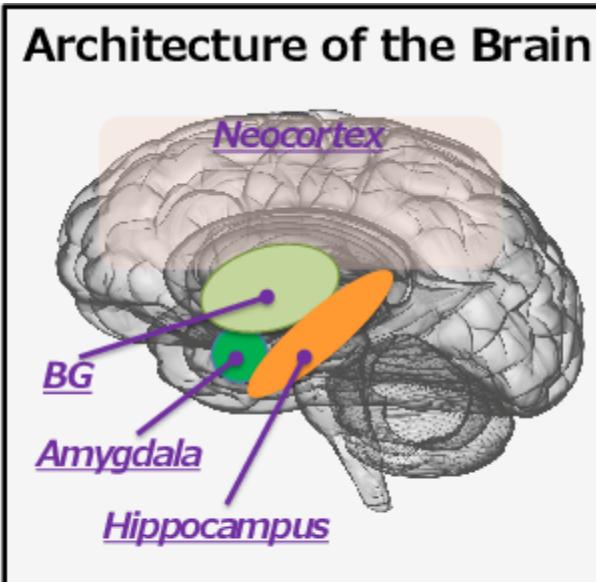
DeepSeek R1

= 685,000,000,000
learnable parameters

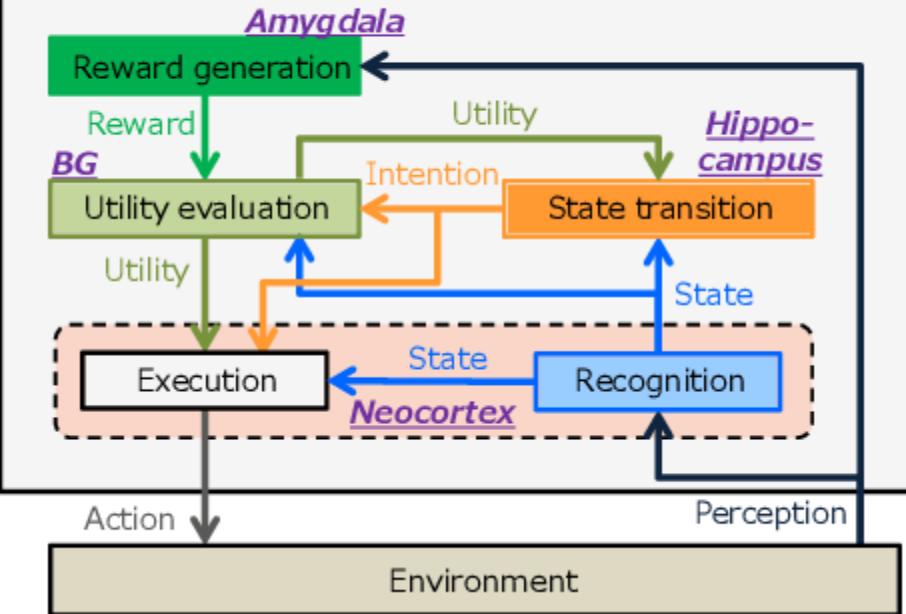
1.4 Deep Learning & Parameters

- **What does more parameters mean?**
- I would describe it as more *nuance* (and more nuance can be a good proxy for more *intelligence*):
 - We can have a simple decision tree learn which assignments pass or fail. This could be a one parameter model that takes a single X to decide.
 - We may decide this too simple (underfitting) and add in more nuance – e.g. factoring in level of study. This would be at least one more parameter.
 - And obviously we can keep adding in more and more nuance (parameters) until our model is considering a sophisticated number of nuanced factors (parameters) – better approximating human intelligence.
 - How many parameters do you need? ...

1.5 Artificial Brains?



Brain-inspired Artificial Intelligence



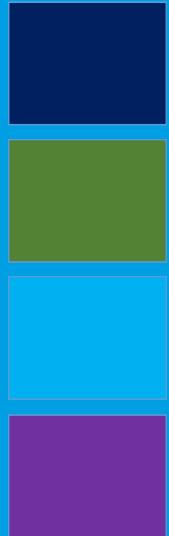
Session Aims

Introduction

Conceptualising Deep Learning: OR How I learned to stop worrying about the math & think only in layers

Neural Network Architecture

Neural Network Training Regimes



2.1 Logistic Regression by Math

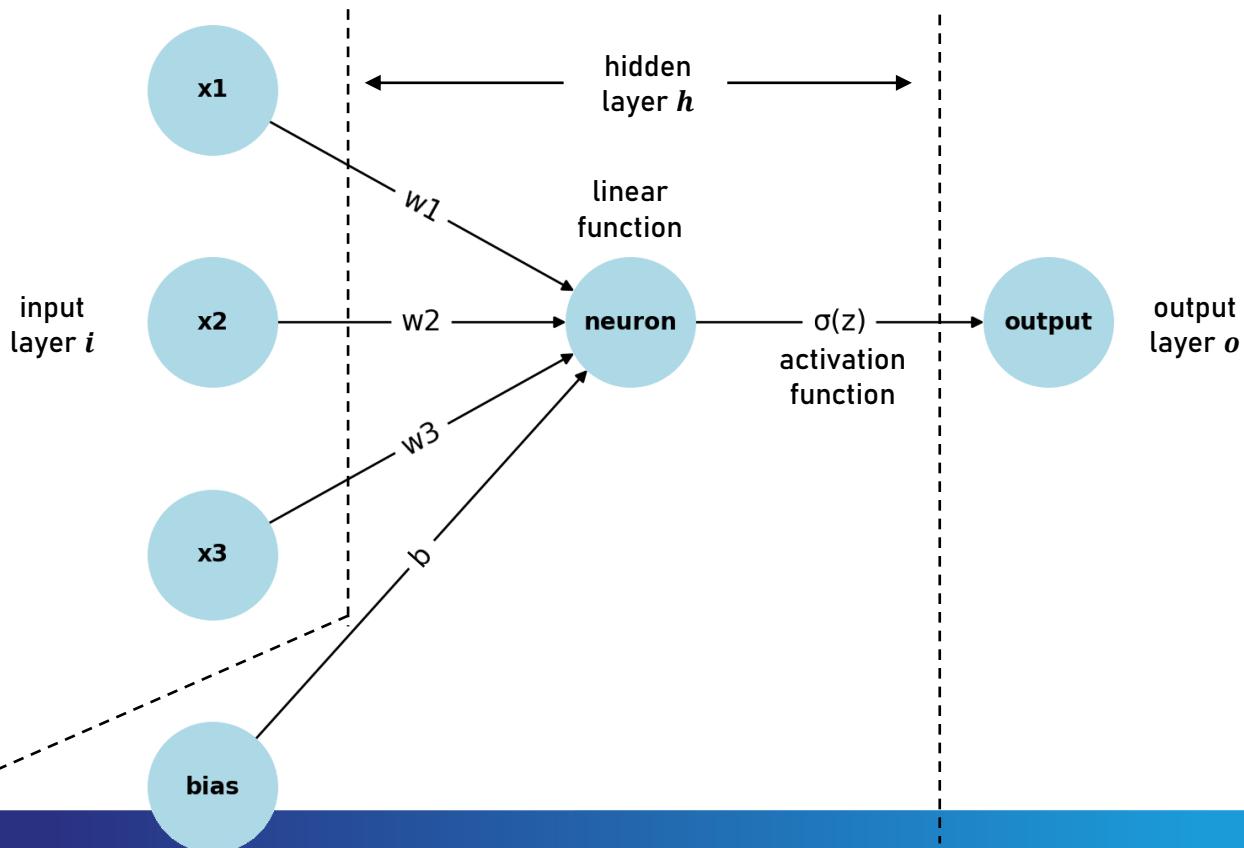
$$\ln\left(\frac{P}{1-P}\right) = \underline{\alpha + \beta_1 x_1} \quad \text{Regression}$$

Probability of
class 1 over
probability of
not class 1

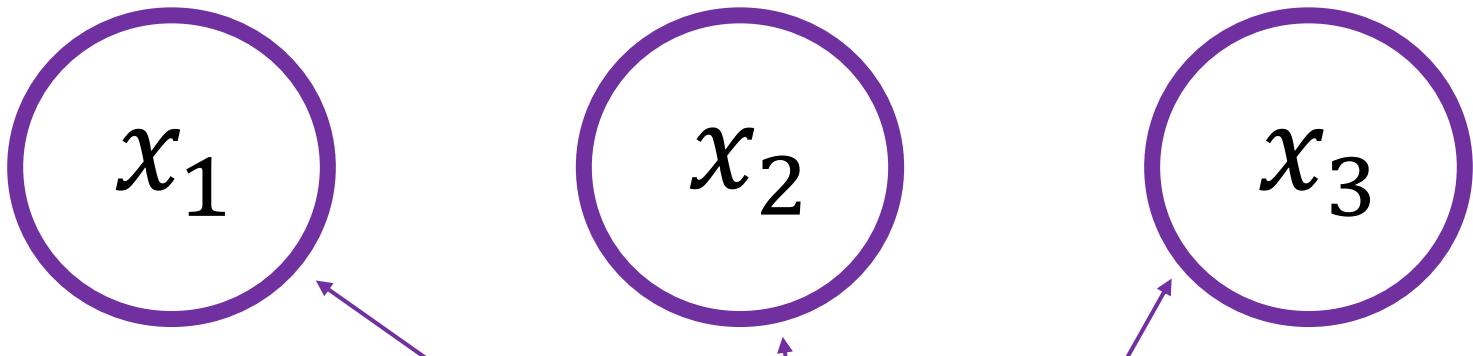
$$\frac{P}{1-P} = e^{\alpha + \beta_1 x_1}$$

$$P = \frac{e^{\alpha + \beta_1 x_1}}{1 + e^{\alpha + \beta_1 x_1}}$$

2.2 Logistic Regression by Layers



2.3 Logistic Regression by Layers (Input Layer)



Date	US_Sales	UK_Sales	Canada_Sales
01/01/2024	\$9,300.52	£4,270.45	\$8,330.70
02/01/2024	\$6,250.65	£5,680.58	\$3,650.58
03/01/2024	\$6,870.29	£8,380.25	\$1,900.04
04/01/2024	\$5,620.17	£9,420.67	\$7,260.93
05/01/2024	\$9,430.28	£8,640.58	\$8,180.83
06/01/2024	\$9,920.03	£2,150.91	\$5,900.60

input_layer
= [3,]

2.4 Logistic Regression by Layers (Linear Layer)

- After the input layer (feature space), each subsequent layer will be a transformation layer. I.e. there will be some form of transformation performed on the information from the previous layer.
- Many of these transformations will be learnable (i.e. we learn the parameters through training) but some will be unlearnable.
- In a linear layer the transformation we learn is the basic linear regression (the learnable bit of logistic regression is linear regression).

$$\hat{Y} = \alpha + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$$

$$z = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

2.4 Logistic Regression by Layers (Linear Layer)

- After the linear transformation, we apply another transformation. In logistic regression this is the *sigmoid* function (σ) - the rearranging we saw early to convert from log odds output to probability of 1 output.
- Although a transformation, the input is the linear function so we consider it part of the same layer (and it is not learnable).
- Instead we would call this an *activation function*. I.e. this (hidden) layer comprises a linear function which leads into to a sigmoid activation function.

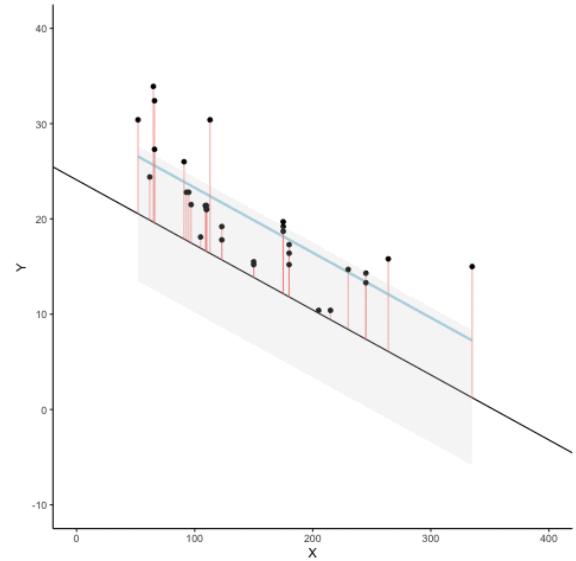
2.5 Logistic Regression by Layers (Output Layer)

- After the linear layer (linear transformation + sigmoid activation function), we are able to output predictions. This layer has two functions essentially:
 1. At **training time** this layer is used to calculate loss (typically for logistic regression this is binary cross-entropy – sometimes called log-likelihood loss). I.e. calculate the loss, update the parameters, start again.
 2. At **prediction time** we evaluate the outputs of the linear layer with a simple rule of:

$$\hat{Y} = \begin{cases} 1, & \text{if } z \geq 0.5 \\ 0, & \text{otherwise} \end{cases}$$

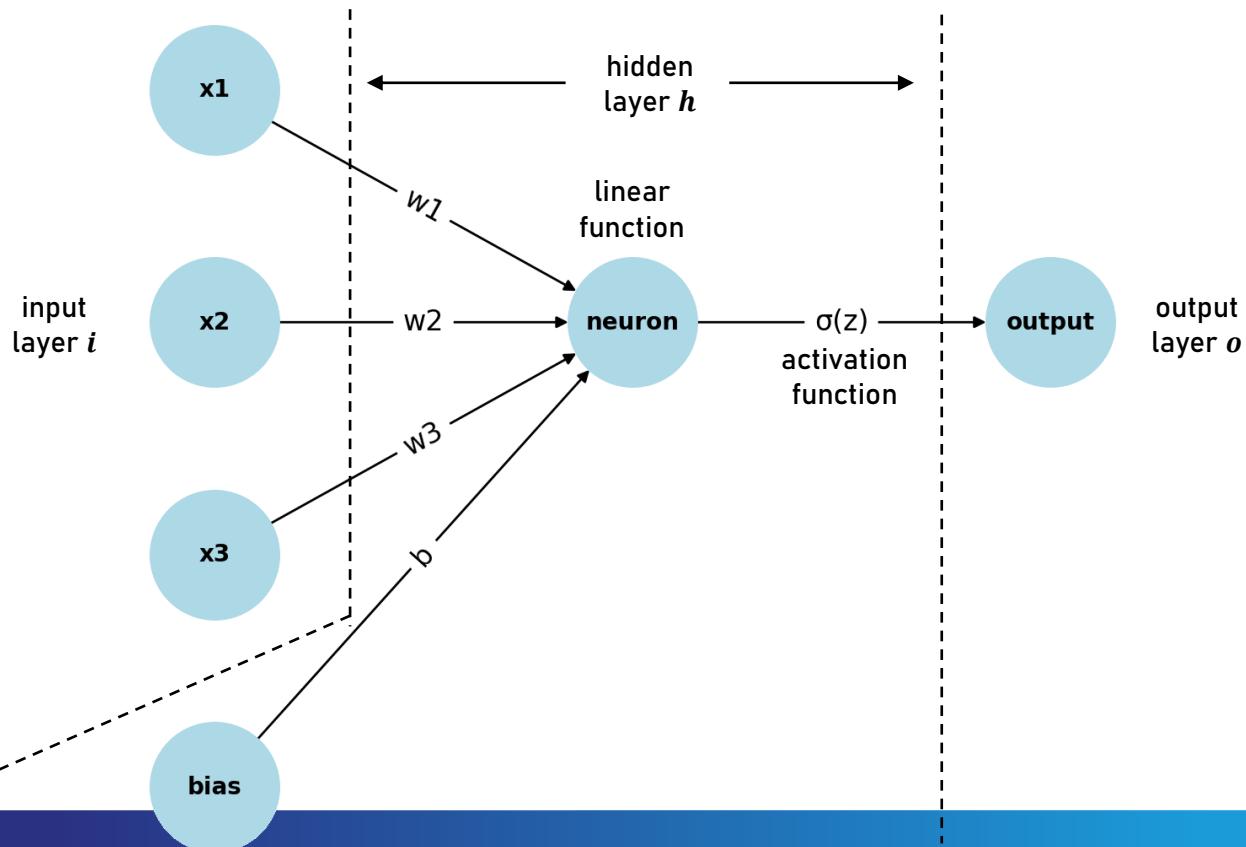
2.5 Logistic Regression by Layers (Output Layer)

- A loss function is the objective we want to minimise when finding the best line(s). In linear regression this was ordinary least squares (OLS).
- In the case of OLS this is basically minimises the error from the line, but more commonly we include some regularisation (e.g. L1/L2) as well ... i.e. its normally more than just error.
- In a single training iteration its called a loss function, but across full training its called a cost function (this is the only difference).



$$\min \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

2.6 Logistic Regression by Layers (in Full)



2.7 Layers Are All You Need



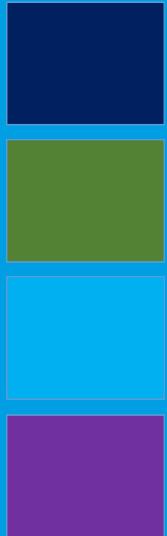
Session Aims

Introduction

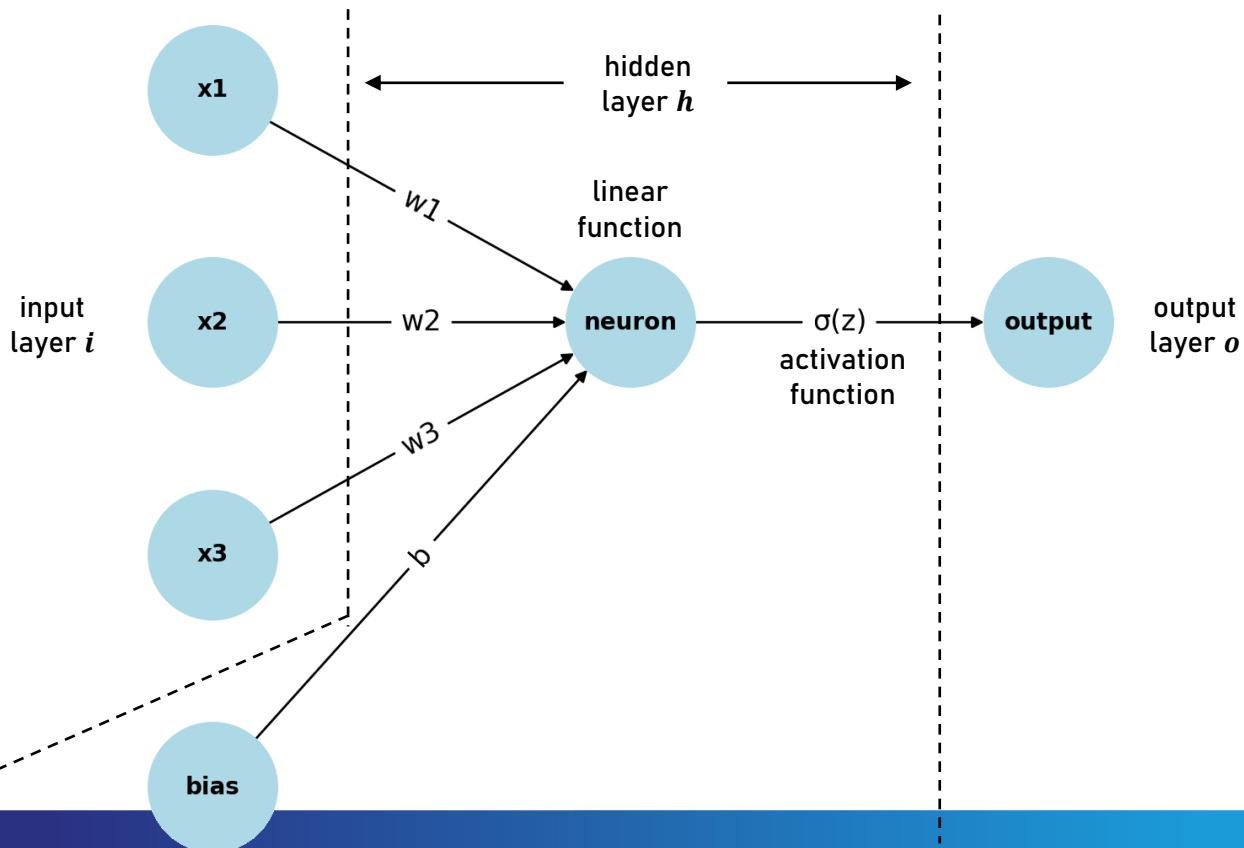
Conceptualising Deep Learning: OR How I learned to stop worrying about the math & think only in layers

Neural Network Architecture

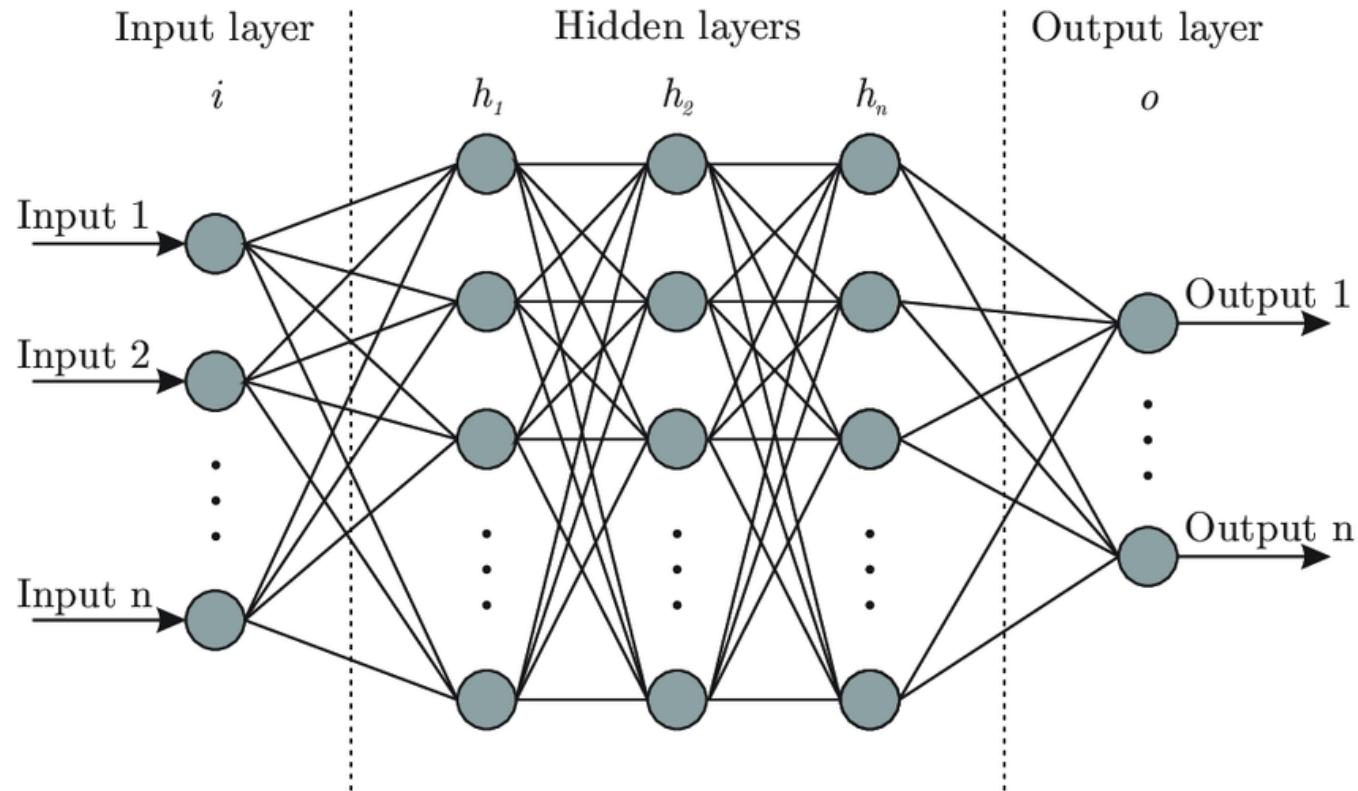
Neural Network Training Regimes



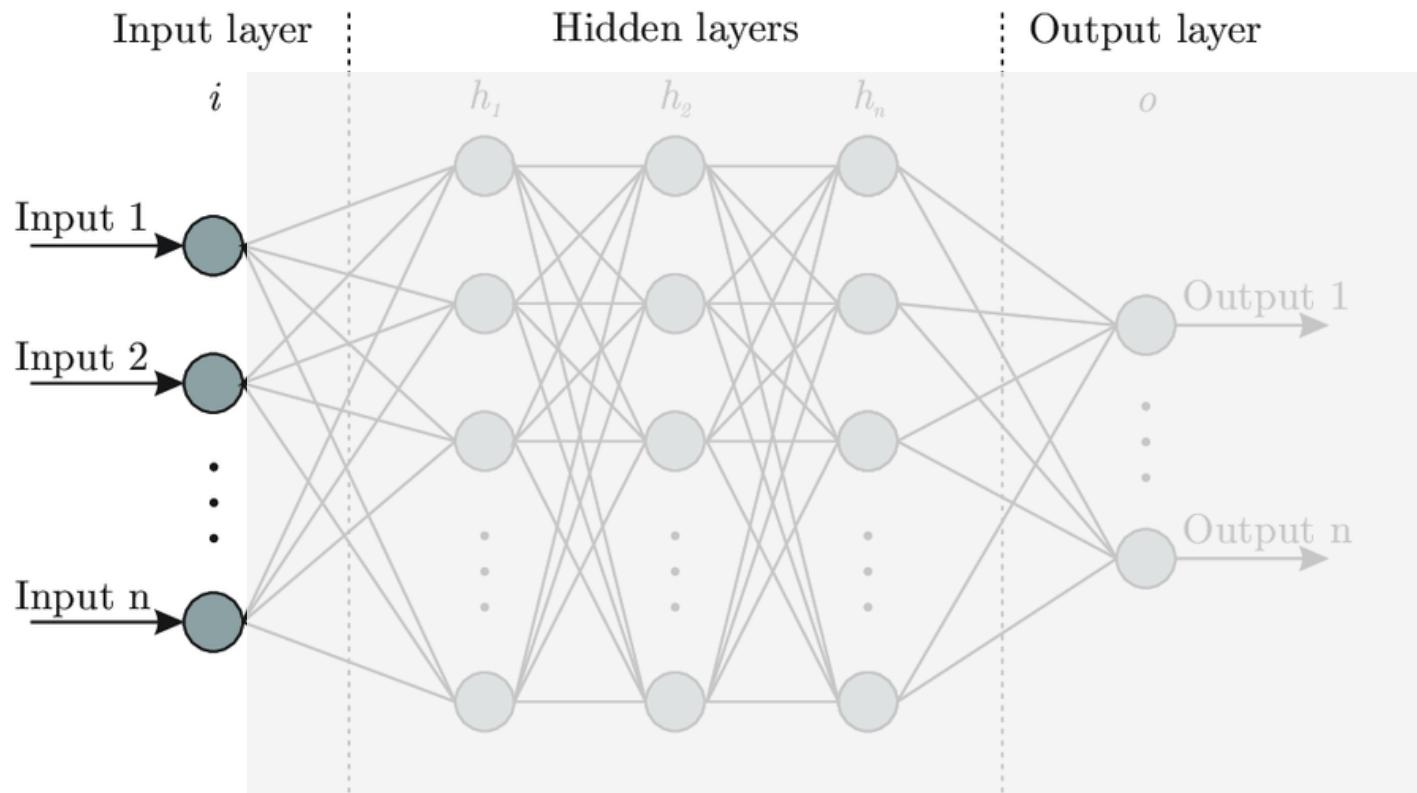
3.1 Neural Network Layers



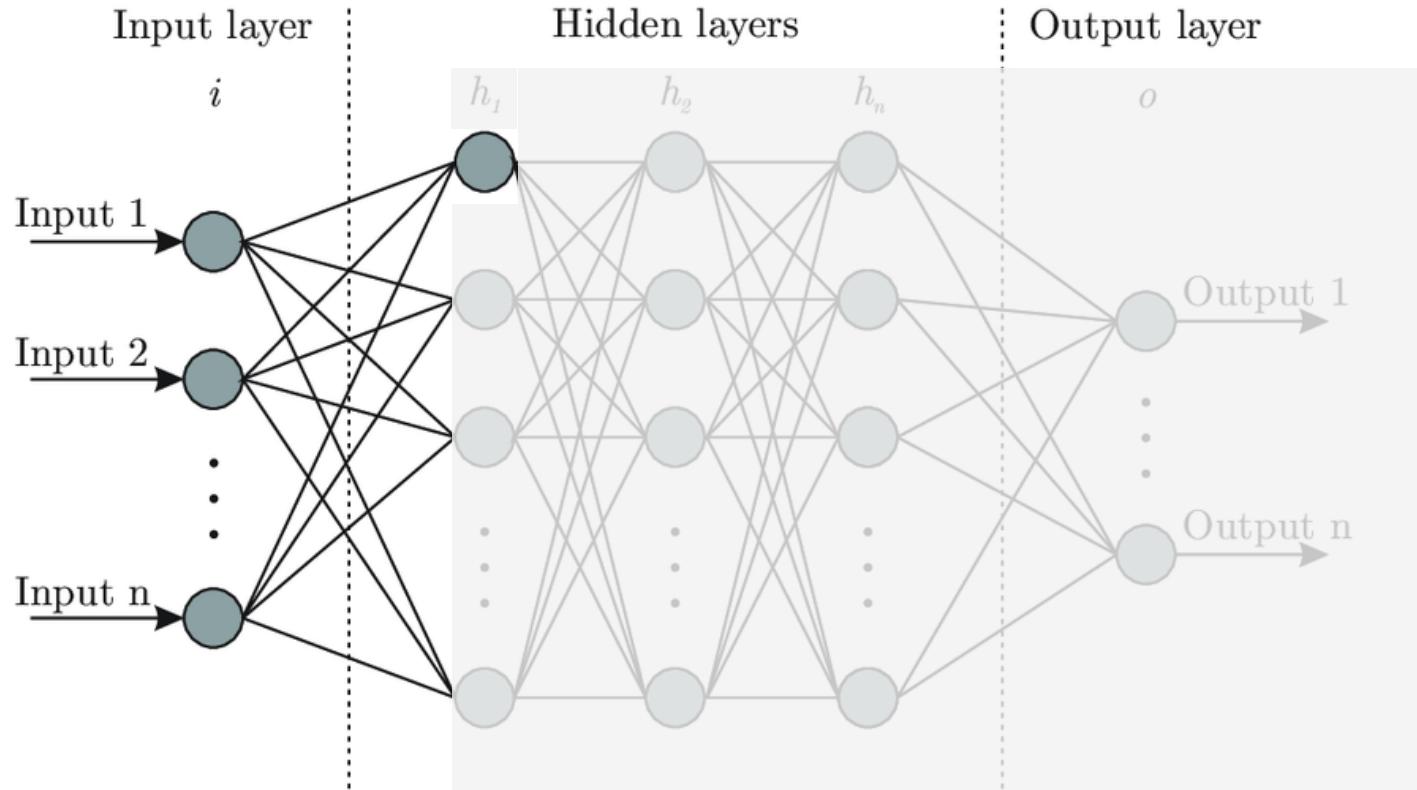
3.1 Neural Network Layers



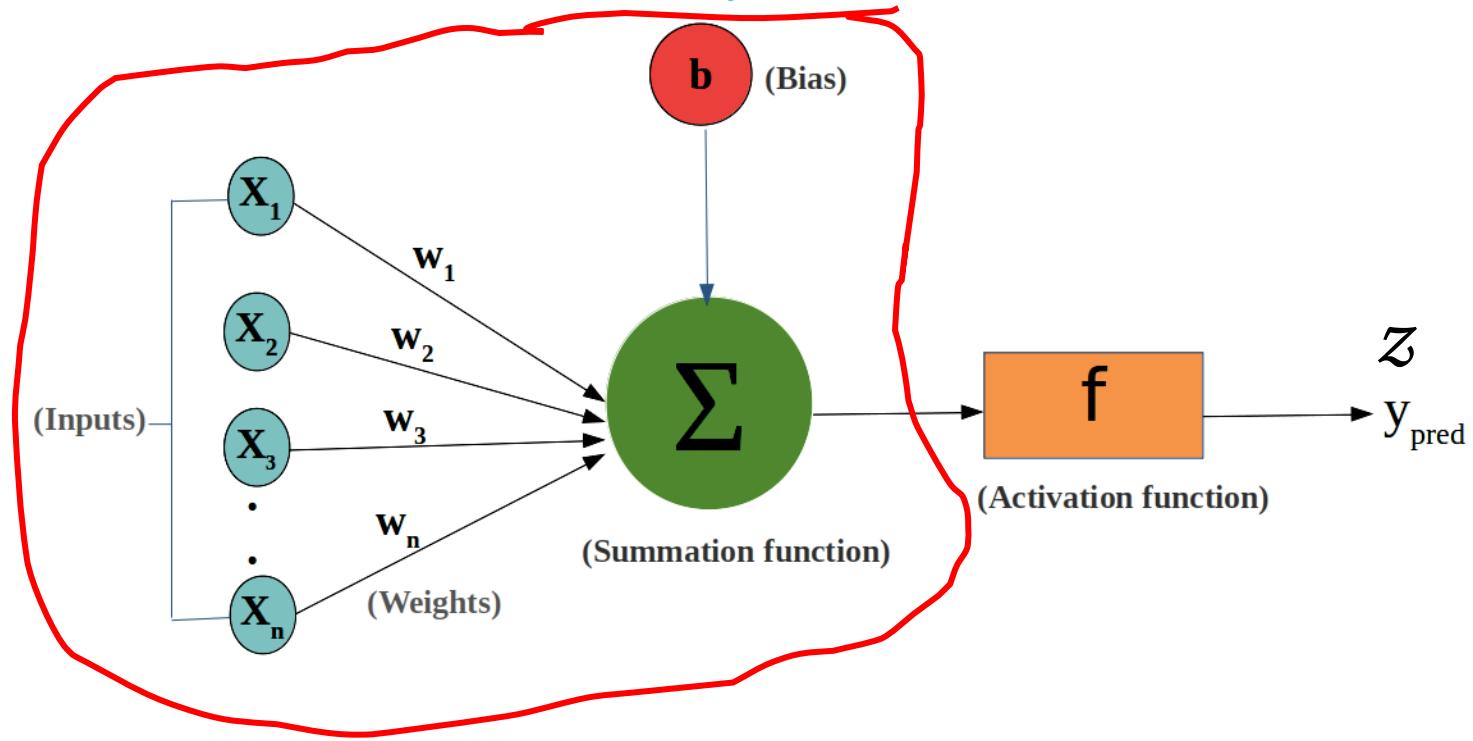
3.2 Neural Network Layers: Input



3.3 Hidden Layers: A neuron ($n_i^{[l]}$)



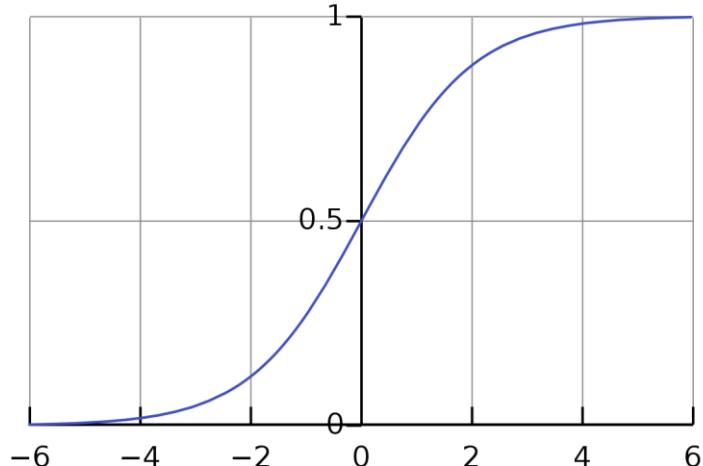
3.3 Hidden Layers: A neuron ($n_i^{[l]}$)



3.4 Hidden Layers: A neuron's activation function

Sigmoid Function

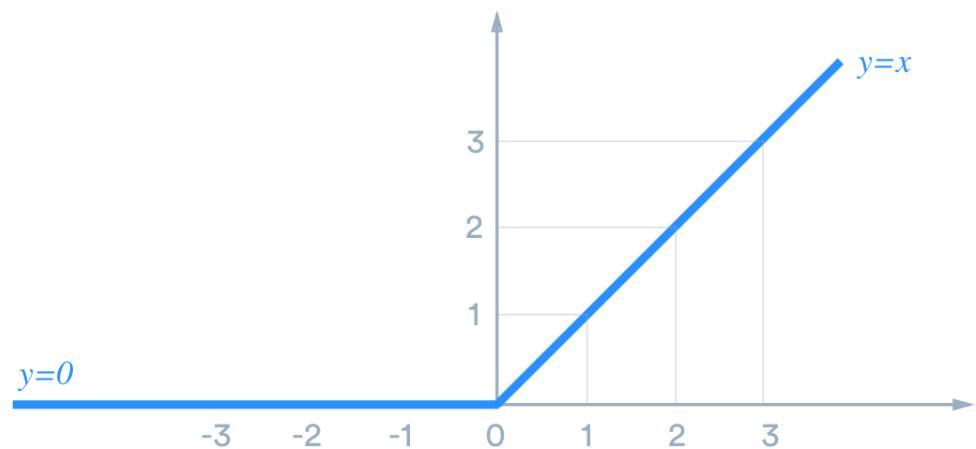
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



Rectified Linear Unit (ReLU)

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

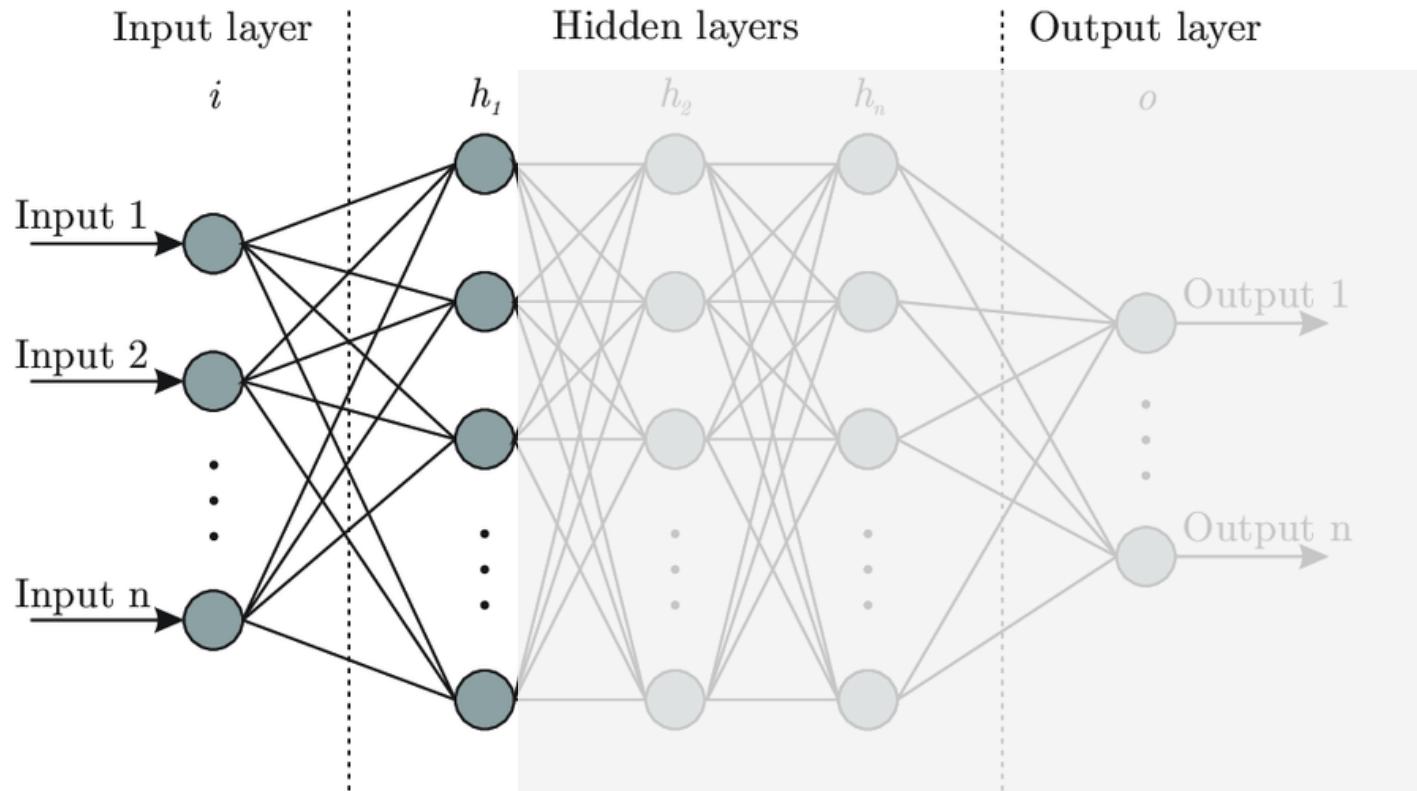
`fx = np.where(x>0, x, 0)`



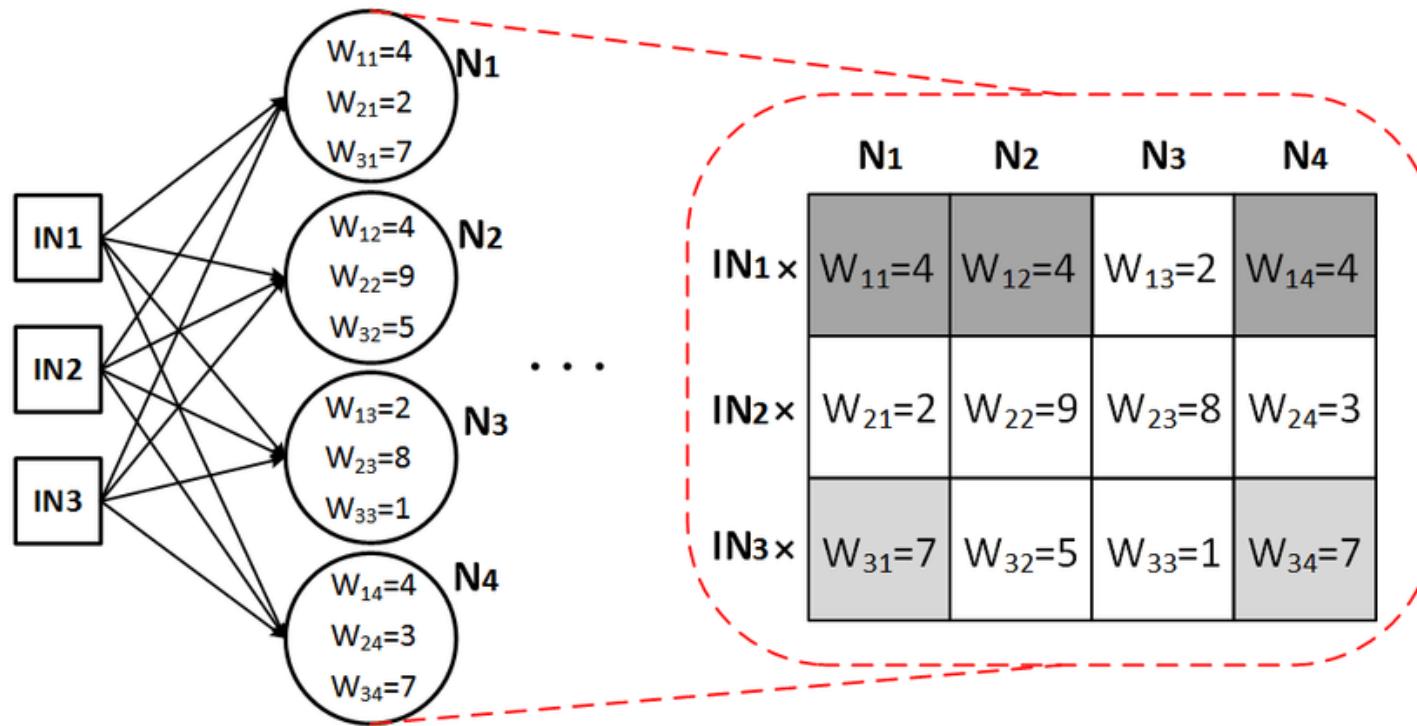
3.4 Hidden Layers: A neuron's activation function

- We use an activation function to achieve the following:
 - ✓ To add non-linearity to our model.
- We use ReLU specifically to achieve the following:
 - ✓ To turn the neuron to an on/off process. If it sees something in the input that matches its area of focus it turns on; otherwise it turns off (produces zero).
 - ✓ Compared to sigmoid (σ), it is faster and easier to compute.
 - ✓ It causes fewer issues than σ w.r.t. vanishing/exploding gradient.

3.5 Hidden Layers: A layer of neurons (l_i)



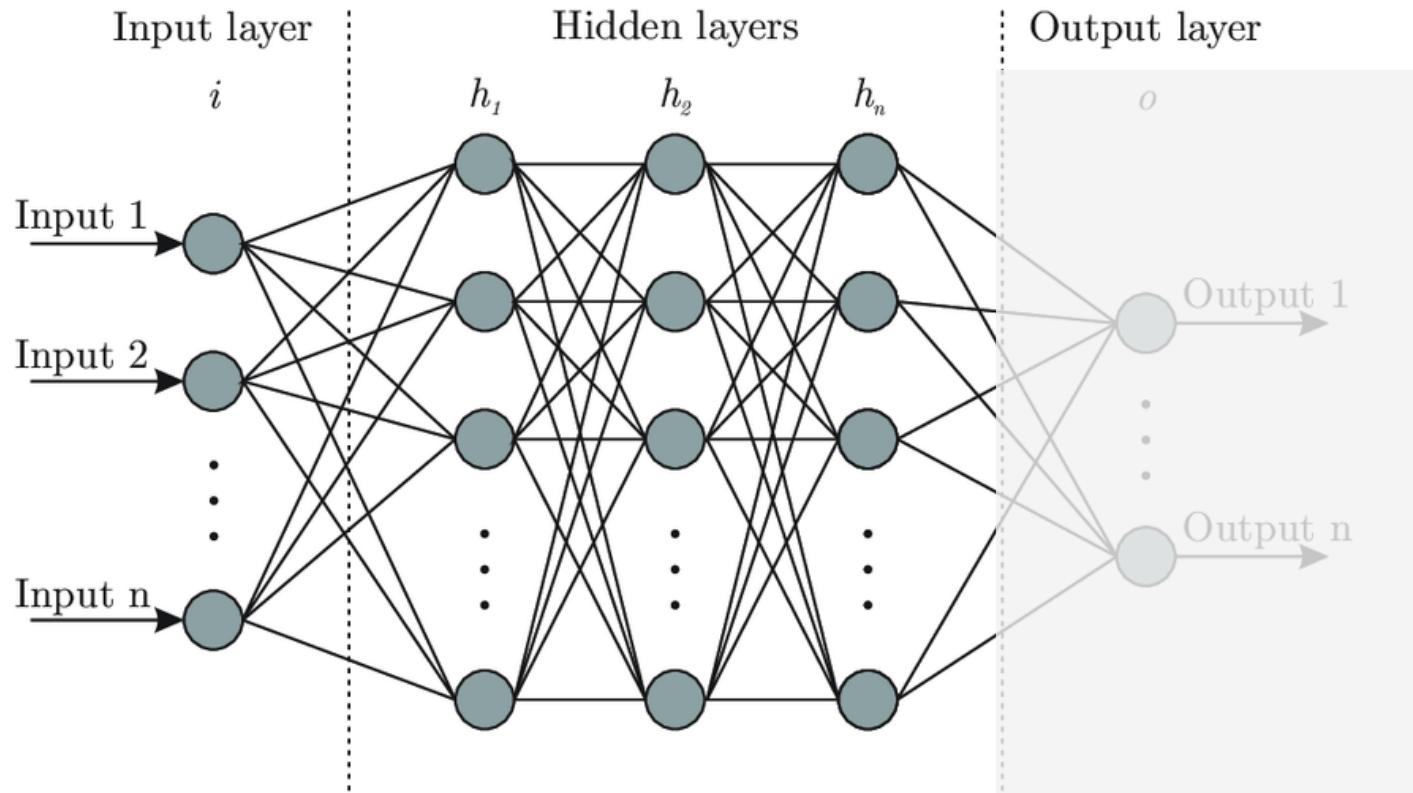
3.5 Hidden Layers: A layer of neurons (l_i)



3.5 Hidden Layers: A layer of neurons (l_i)

- Unlike our logistic regression, each linear layer has multiple neurons (the exact value is a hyperparameter effectively).
- In effect these act as *feature extractors*, working like different trees in a random forest to specialise in different things. Whilst they are not necessarily interpretable combinations, more often than not they seem to form meaningful latent variables (video coming!).
- They will all be ‘excited’ by different inputs from the previous layer (input layer or previous hidden layer).
- With the effect of ReLU, if they are ‘excited’ they return a value > 0 , and turn-off otherwise (similar to different parts of the brain firing).

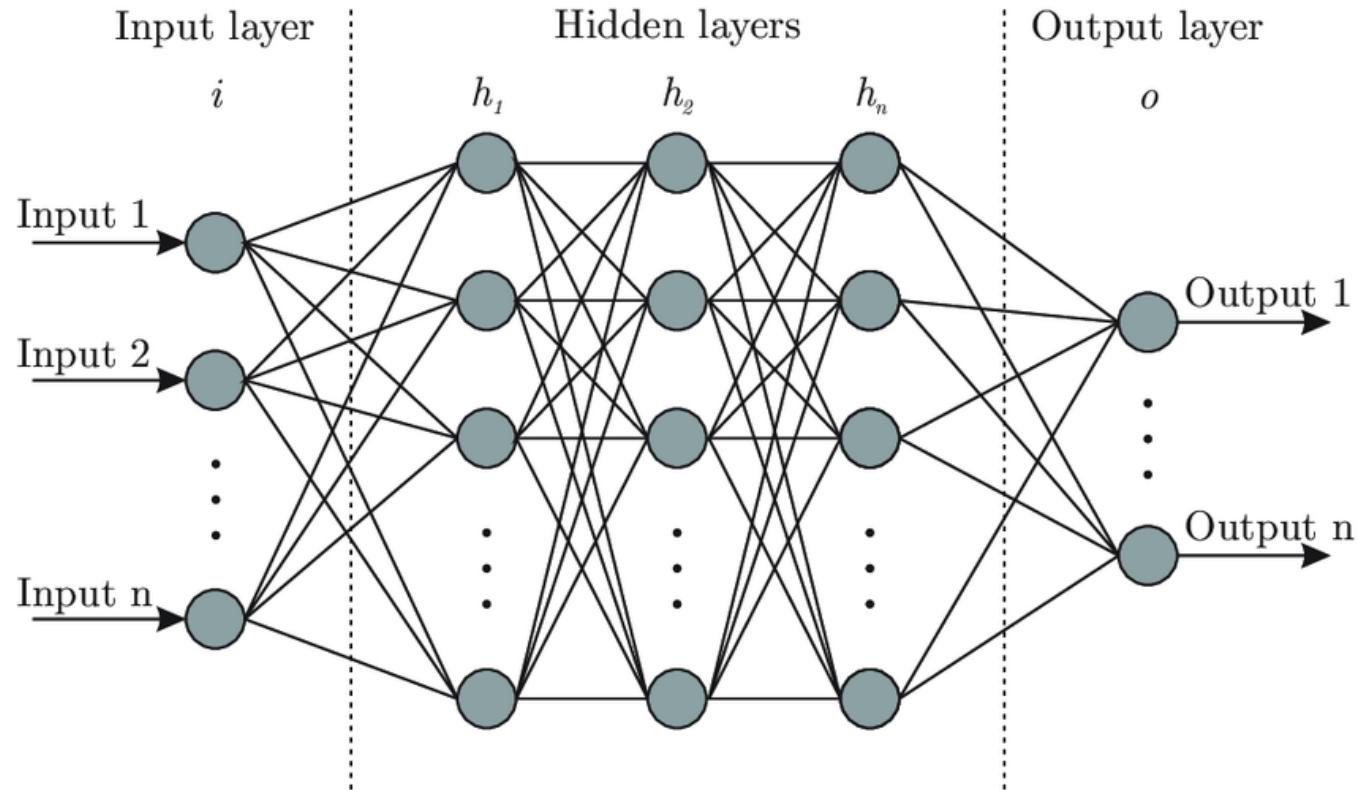
3.6 Hidden Layers: The full set of layers (L)



3.6 Hidden Layers: The full set of layers (L)

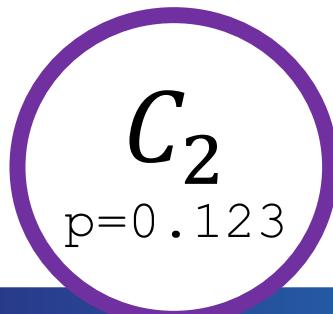
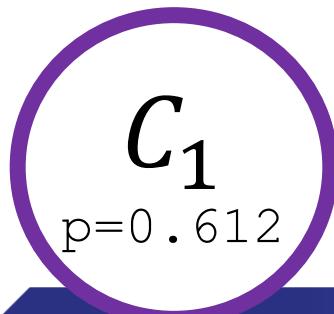


3.7 Output Layer (\hat{Y})

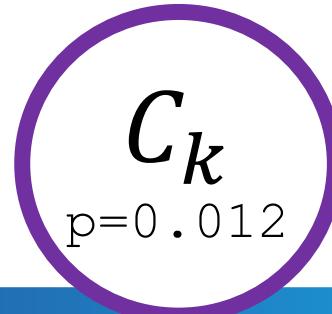


3.7 Output Layer (\hat{Y})

- There are a range of outputs that our model may produce:
 - Regression – output is (typically) a single neuron – a number. The most common loss functions would be MSE or MAE .
 - Binary classification – output is a single neuron – a value between 0 and 1 (the probability of 1). The most common loss function is binary cross entropy.
 - Multiclass classification – output is a set of neurons = K (where K is the number of classes) and each is a probability of class assignment (0-1). Most common loss function is categorical cross entropy (softmax).



...



$$\sum_{i=1}^k p(C_i) = 1$$

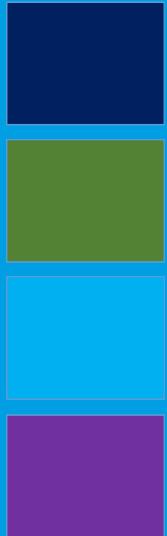
Session Aims

Introduction

Conceptualising Deep Learning: OR How I learned to stop worrying about the math & think only in layers

Neural Network Architecture

Neural Network Training Regimes



4.1 Batches, Iterations & Epochs

```
[11] # Create data
    rng = np.random.RandomState(1)
    X = np.dot(rng.rand(2, 2), rng.randn(2, 200)).T
    print(f"Full data is {len(X)} records")

    splitter = 40
    batches = []

    # Split data into batches
    for i in range(5):
        batches.append(X[i*splitter:(i+1)*splitter])

    print(f"First batch is {len(batches[1])} records")
```

Full data is 200 records
First batch is 40 records

```
iterations = 0

for batch in batches:
    iterations += 1
    print(f"Iteation #{iterations}")
```

Iteation #1
Iteation #2
Iteation #3
Iteation #4
Iteation #5

Batch: a chunk of the data we feed into the algorithm. Splitting into batches means learning loss on chunks of data rather than all, making training more computationally efficient and reducing the risk of overfitting (see SGD optimisation).

Iteration: any time we feed a batch into the algorithm

4.1 Batches, Iterations & Epochs

- An **Epoch**: the presentation of the full dataset (i.e. all the batches) to the algorithm

$$\text{Iterations} = \text{epochs} \cdot n_{\text{batches}}$$

where:

$$n_{\text{batches}} = \frac{n}{\text{batch_size}}$$

4.2 Training Neural Networks

- In modern AI it has become common to train our models for many epochs, even beyond what was considered “overfitting”.



Further Reading

- Chollet F (2021). *Deep learning with Python*. Manning: Shelter Island.
- Goodfellow I, Bengio Y and Courville A (2016). *Deep Learning*. The MIT Press: Cambridge, MA.
- Pajankar A and Joshi A (2022). *Hands-on machine learning with Python: implement neural network solutions with Scikit-learn and PyTorch*. Apress: Berkeley, CA.