



Warwick  
Business  
School

# Data Science & Generative AI

Dr Michael Mortenson  
Associate Professor (Reader)  
*michael.mortenson@wbs.ac.uk*




## Week 6: Machine Learning in Practice

## 1.1 Deep Learning Recap


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

= 2 learnable parameters



DeepSeek R1

+

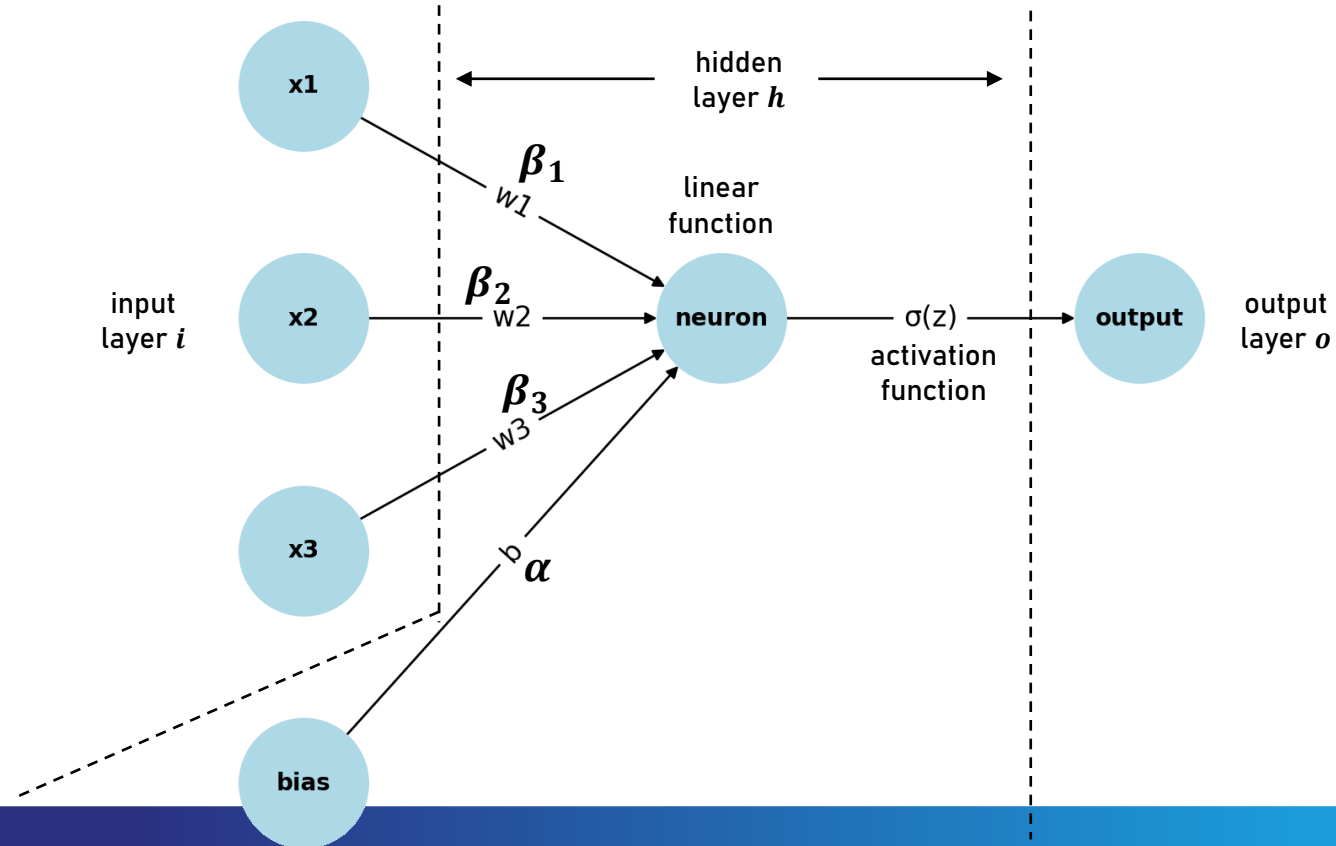


= 685,000,000,000  
learnable parameters

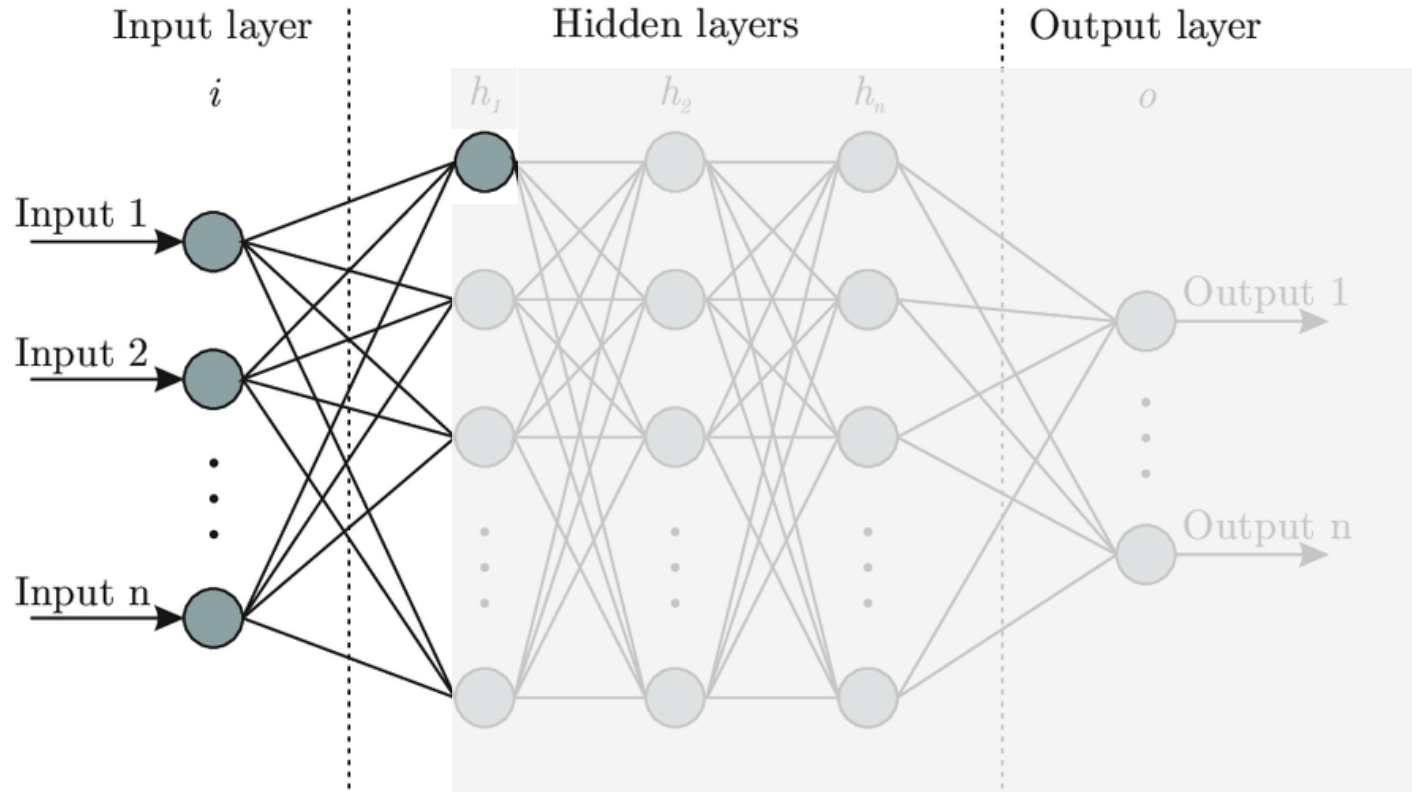
## 1.2 Deep Learning & Data



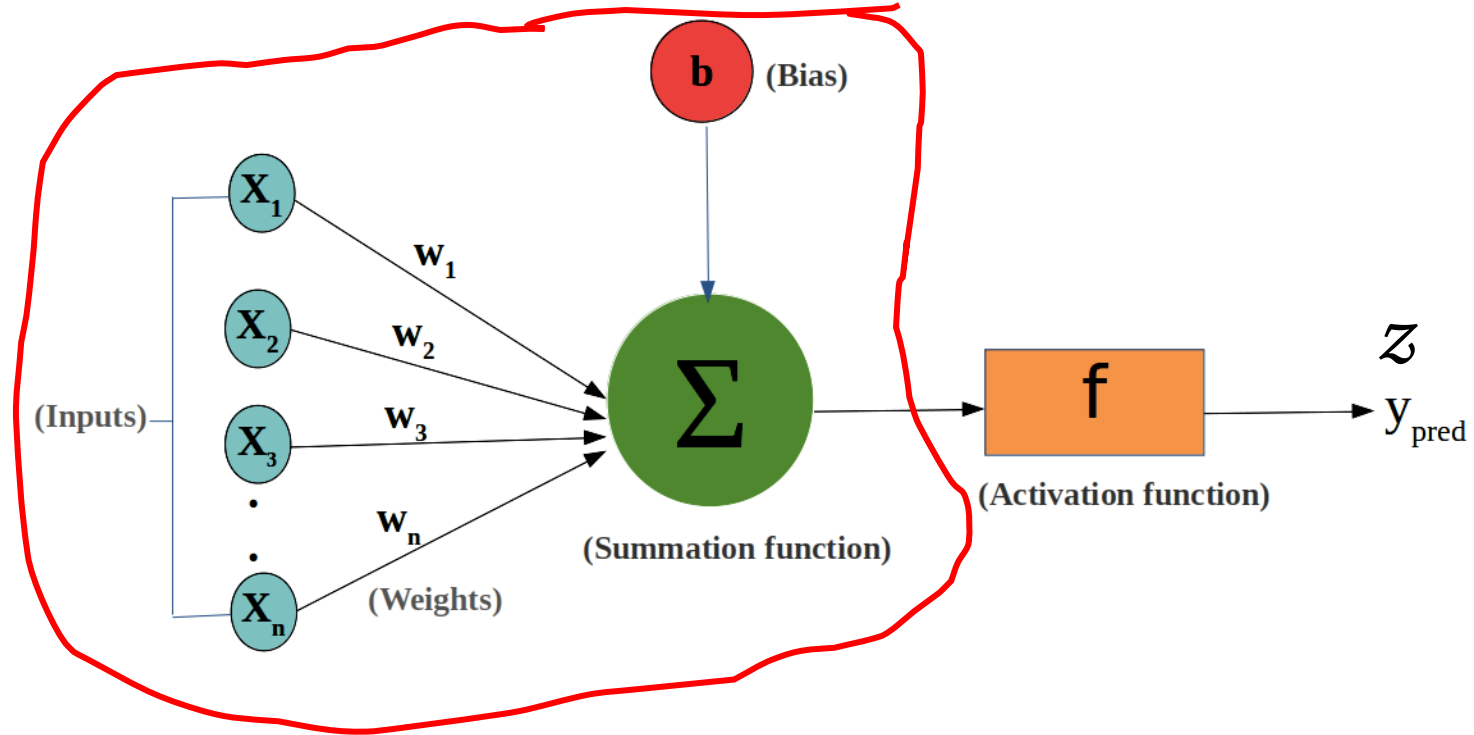
## 1.3 Logistic Regression by Layers



## 1.4 Neural Nets: A neuron ( $n_i^{[l]}$ )



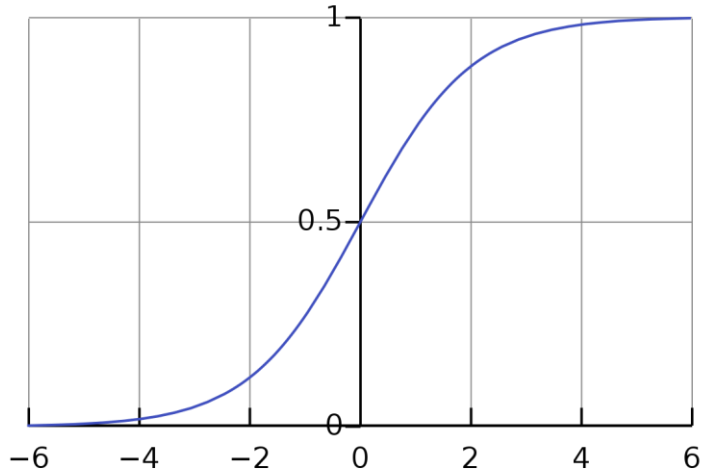
## 1.4 Neural Nets: A neuron ( $n_i^{[l]}$ )



## 1.4 Neural Nets: A neuron ( $n_i^{[l]}$ )

### Sigmoid Function

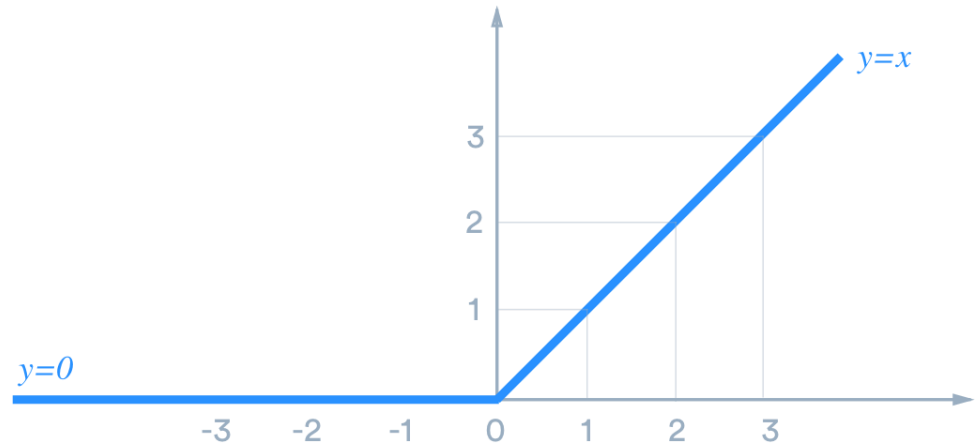
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



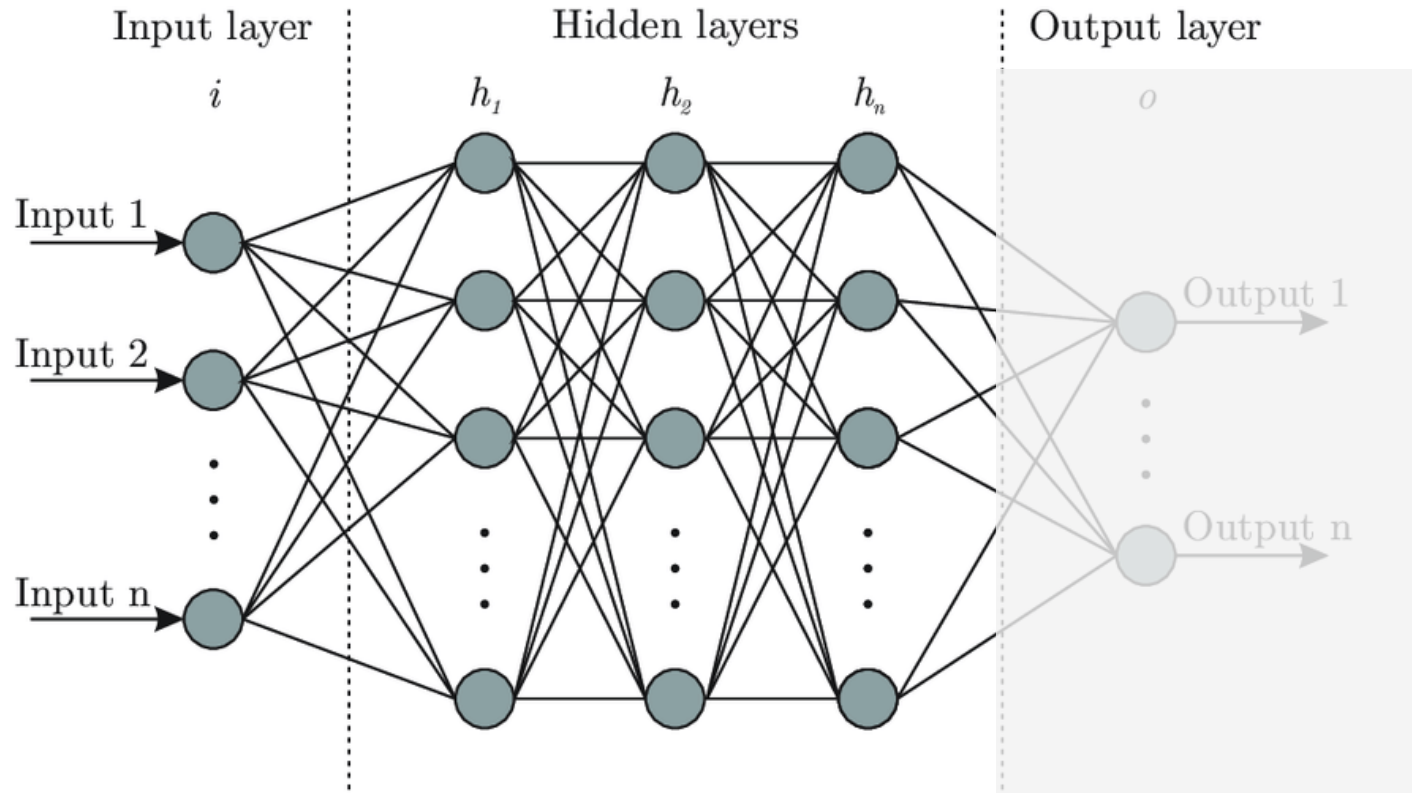
### Rectified Linear Unit (ReLU)

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

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



## 1.5 Neural Nets: The full set of layers ( $L$ )



## 1.6 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

## 1.6 Batches, Iterations & Epochs

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

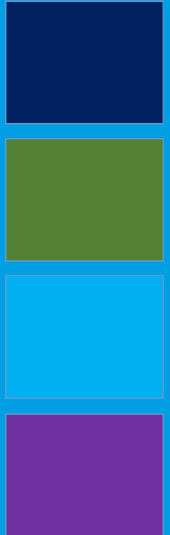
# Session Aims

Introduction

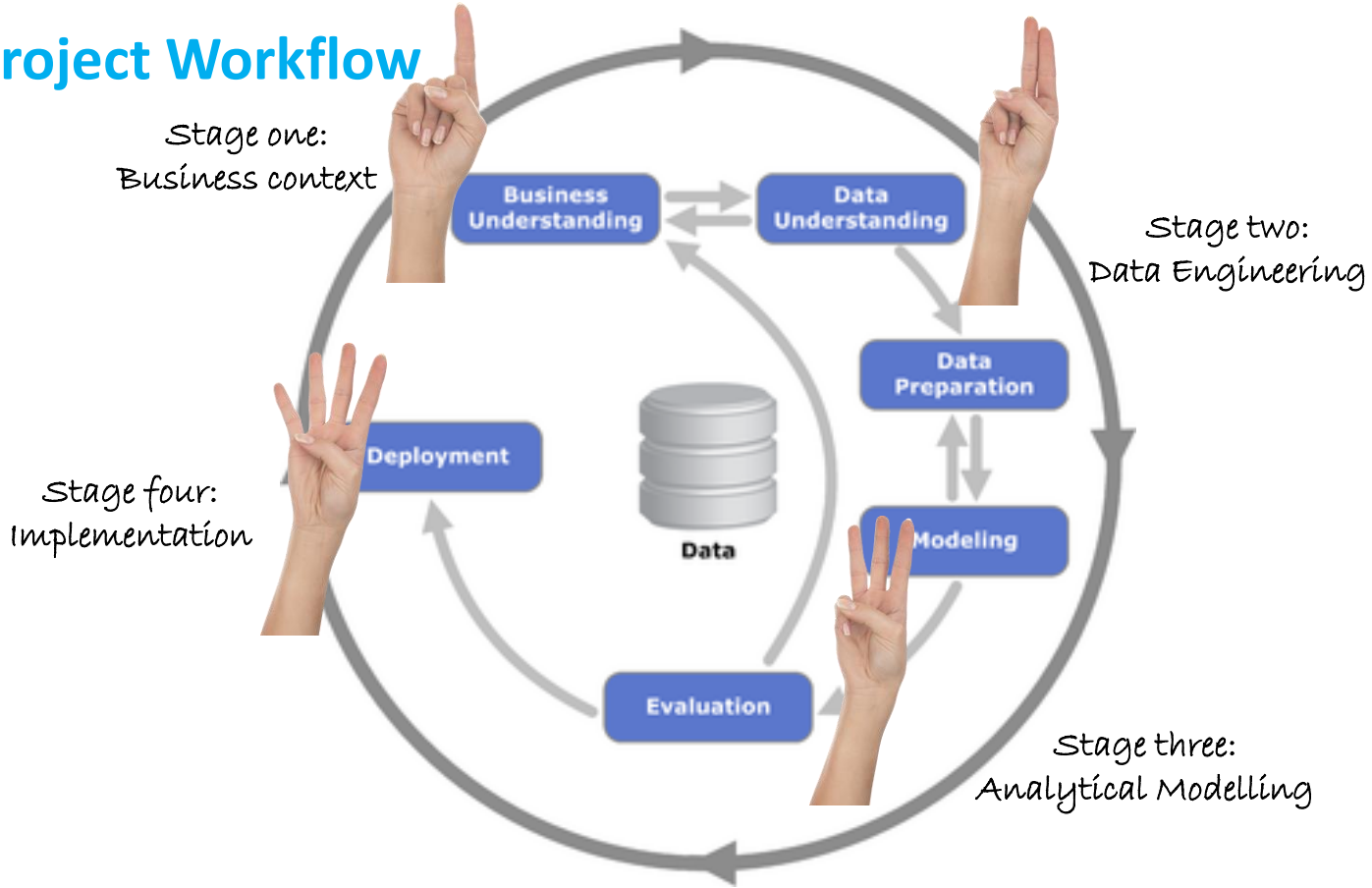
**Machine Learning in Practice**

Group Project

Home Time



## 2.1 Project Workflow



## 2.2 Business Understanding

- All the problem structuring approaches / situation analysis techniques required to establish the context.
- Establish what we are trying to predict. Does  $Y$  exist? Is it continuous or discrete? Is it better to model the problem as continuous or discrete? How should we prepare  $Y$  correctly (e.g. what threshold is best to use to convert to classes?)
- What is the best measure of success – the best score to use?
  - For regression, normally this means is it better to treat all error equally (absolute error) or to punish bigger errors more (MSE)?
  - For classification, which is worse ... mispredictions or missed predictions (precision or recall)? Or both equally bad?

## 2.2 Business Understanding

- What does the business think is least worse?
  - Misprediction (precision) – when I make predictions how often am I right?
  - Missed predictions (recall) – when a real example exists, how often am I able to predict it?
  - Both are bad (accuracy or F1 score).
- How imbalanced is my data? If imbalanced we can't use accuracy (and may need to focus more on the minority class).
- Does predicting one class matter more than the other?
  - If “YES” then use the metrics just for that class.
  - If “NO” then use macro averages.

## 2.3 Data Understanding & Preparation

- Collecting, aggregating and cleaning the data:
  - How can I verify the quality of the data?
  - What is the correct unit of aggregation?
- How can I create new predictive features from the data?
  - SME workshops
  - Feature stores
  - Data-driven feature engineering
- How can I prepare the data for modelling?
  - Scaling the data
  - Separate the  $Y$  value.
  - Training/test split.

## 2.3 Data Understanding & Preparation

*“Feature engineering is the process of transforming raw data into relevant information for use by machine learning models. [...] **Because model performance largely rests on the quality of data used during training**, feature engineering is a crucial preprocessing technique that requires selecting the most relevant aspects of raw training data for both the predictive task and model type under consideration”*

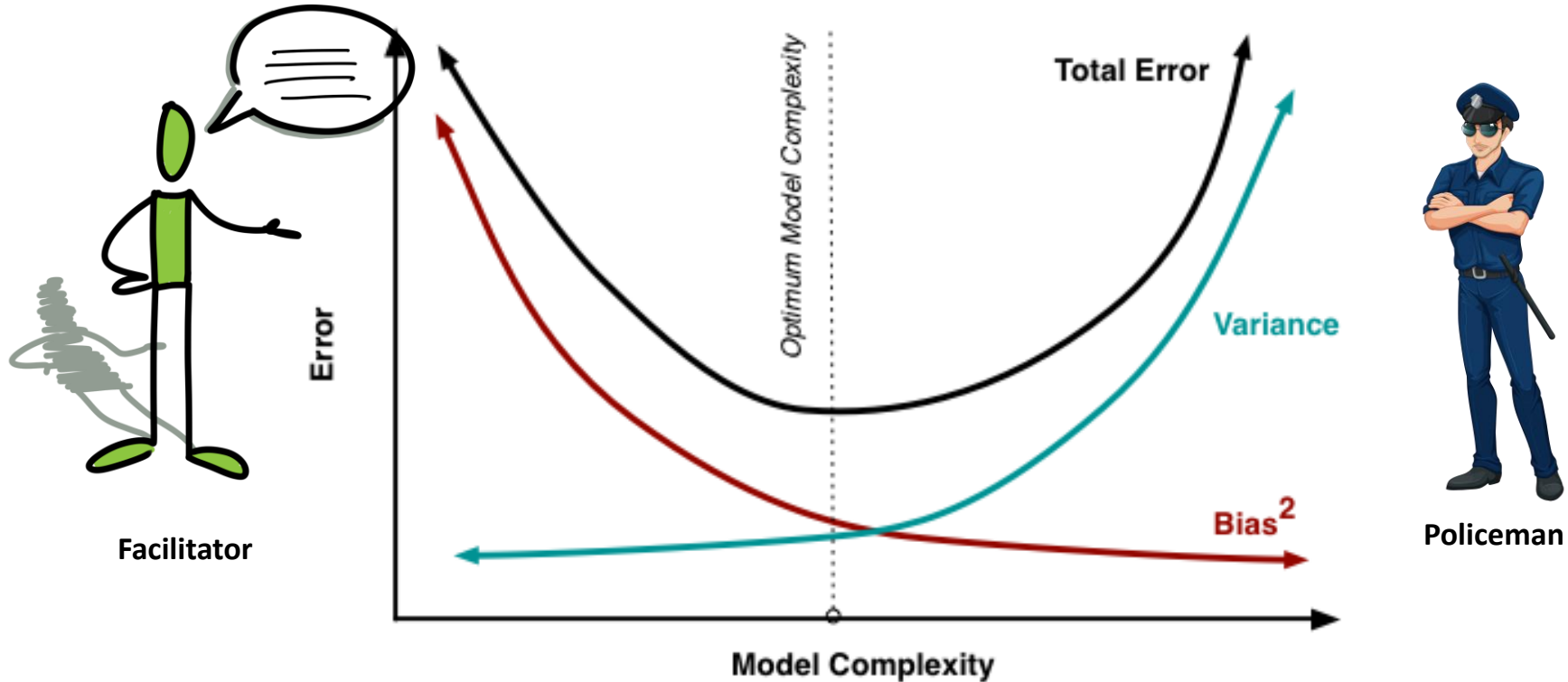
## 2.4 Modelling and Evaluation

- What algorithms should I use?
  - If you are not doing deep learning its probably an ensemble approach that will work best.
  - I.e. Random forest and/or gradient boosting decision trees.
- Train the model(s).
- (Optional) evaluate the model(s) on the training data.
- Hyperparameter optimise the model(s).
- (Optional) evaluate the model(s) on the training data.
- Evaluate the model(s) on the test data.
- Troubleshoot.

## 2.5 The “I Spent Ages on That and the Scores are Trash” Feeling



## 2.6 Under- and Over-Fitting



## 2.6 Under- and Over-Fitting

- Has my model overfit?
  - Is the test performance significantly worse than training performance?
  - Is the complexity of my modelling approach excessive given the amount of data I have?
- What should I do?
  - Increase regularisation in the model (varies by algorithm).
  - Use a simpler model to see if this performs better.
  - Increase the size of your dataset or improve its cleaning / feature engineering.

## 2.6 Under- and Over-Fitting

- Has my model underfit?
  - Is the test performance about the same or even better than training performance?
  - Is the complexity of my modelling approach too simplistic given the complexity of the problem space?
- What should I do?
  - Decrease regularisation in the model (varies by algorithm) ... within reason!
  - Use a more complex model to see if this performs better.
  - Increase the size of your dataset or improve its cleaning / feature engineering.

## 2.7 Imbalanced Samples

- The University of Warwick has developed a model to predict when students may (sadly) decide to drop out of their course.
- Based on 2024/25 data (not really I made all of this up), 9 out of 10 students complete their studies.
- We have developed a model and in training it shows an accuracy of 88%.
- Is the model any good?

## 2.7 Imbalanced Samples (Hyperparameters)



```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform, randint

# this is the list of parameters we will tune. Some are fixed values some are distributions
tuned_parameters = {
    'criterion': ['gini', 'entropy'],
    'max_depth': randint(3, 9), # Draw from a uniform distribution between 3 and 15
    'min_samples_split': randint(3, 9), # Draw from a uniform distribution between 2 and 10
    'max_features': ['sqrt', 'log2', None]
}

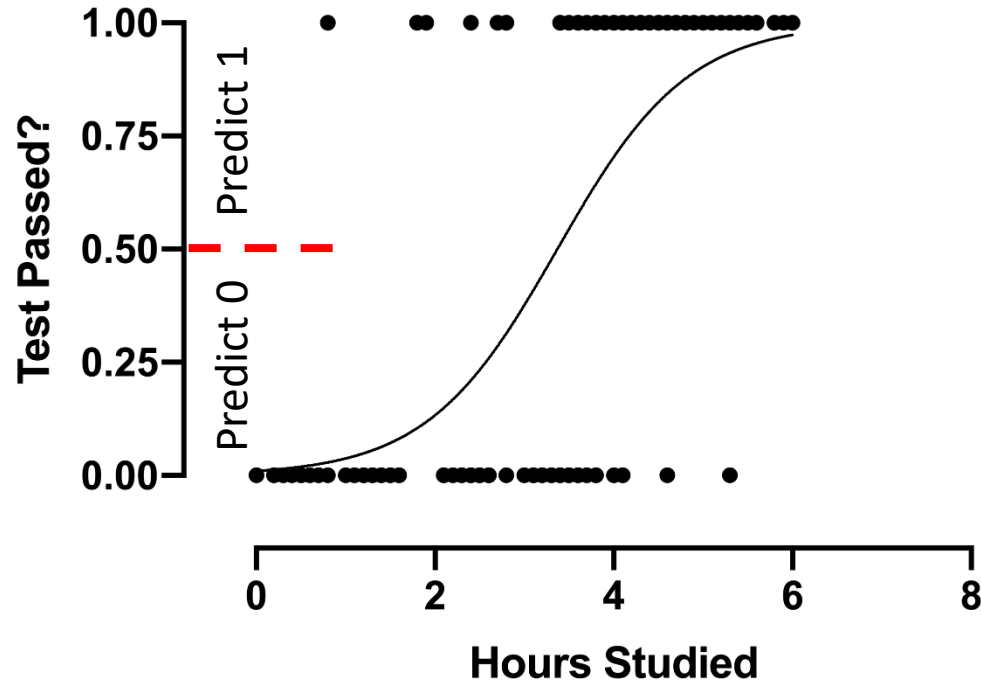
# we will specify recall based on class 1
score = 'recall' # defaults to class 1

print(f"Tuning hyperparameters for {score}")
print("\n")

# do the search using 5 folds/chunks
clf = RandomizedSearchCV(tree(), tuned_parameters, cv=5, random_state=1984,
                        scoring= score, n_iter=20, refit=True)

# pass the data to fit/train
clf.fit(X_train, Y_train)
```

## 2.7 Imbalanced Samples (Bias the Threshold)



## 2.7 Imbalanced Samples (Undersampling)

*Note: the class with the most samples is known as the **majority class** and the class(es) with fewer samples are known as the **minority class***

### Under Sampling



Rather than using all the data from our **majority class** we sample a random subset that makes the class numbers more even.

1. **Pros:** Very simple to execute; unbiased
2. **Cons:** Means throwing out good data
3. **Best use case:** when the dataset is large and relatively homogenous

## 2.7 Imbalanced Samples (Oversampling)

*Note: the class with the most samples is known as the **majority class** and the class(es) with fewer samples are known as the **minority class***

### Oversampling



Creating more examples of the minority class by creating synthetic data that has similar characteristics.

1. **Pros:** Often the most effective solution
2. **Cons:** Means generating synthetic data (i.e. fake data); can over-exaggerate characteristics; harder to execute
3. **Best Use-Case:** Where there are “bridges” which we can find through sampling

## 2.7 Imbalanced Samples (Oversampling)

### Synthetic Minority Over-sampling TEchnique

SMOTE works by creating synthetic samples of the **minority class**.

Note: synthetic samples  $\neq$  replications; replications would not help train the model.

The algorithm samples from the data from the **minority class** similarly to the bootstrapping approach. The results is synthetic data that “looks” like the real data but has random variations.

## 2.7 Imbalanced Samples (Oversampling)

As opposed to bootstrapping, which would sample anywhere in the minority class' distributions, SMOTE will more strategically select where to sample from.

The general approach is to:

1. Select a real example in the dataset at random;
2. Randomly select one of the  $k^{\text{th}}$  nearest neighbours to this example;
3. Plot a straight-line path between the example and its neighbour;
4. Randomly select another point in this path;
5. Create the synthetic example at this point.

## 2.7 Imbalanced Samples (Oversampling)



## 2.8 Which is Best?

**DON'T ASK ME –  
ASK THE DATA**