UNIVERSITY OF PORTSMOUTH

AI 101

Becky Canning

Institute of Cosmology and Gravitation

With huge thanks to:

- Ben, Coleman, Lorenza, Obinna, Xan

UNIVERSITY OF PORTSMOUTH

AI 101

- Key AI things to know:
  - Concepts of: feature engineering, anatomy of a model, and regularisation

- For EO:
  - Querying web APIs with Python
  - Identifying things in images
  - What might we use for time series?
  - The pitfalls often found with EO data

- For space:
  - TinyML models

AI 101

- Key AI things to know:
  - Concepts of: feature engineering, anatomy of a model, and regularisation

- For EO:
  - Querying web APIs with Python
  - Identifying things in images
  - What might we use for time series?
  - The pitfalls often found with EO data

- Clustering
- Classification

- For space:
  - TinyML models

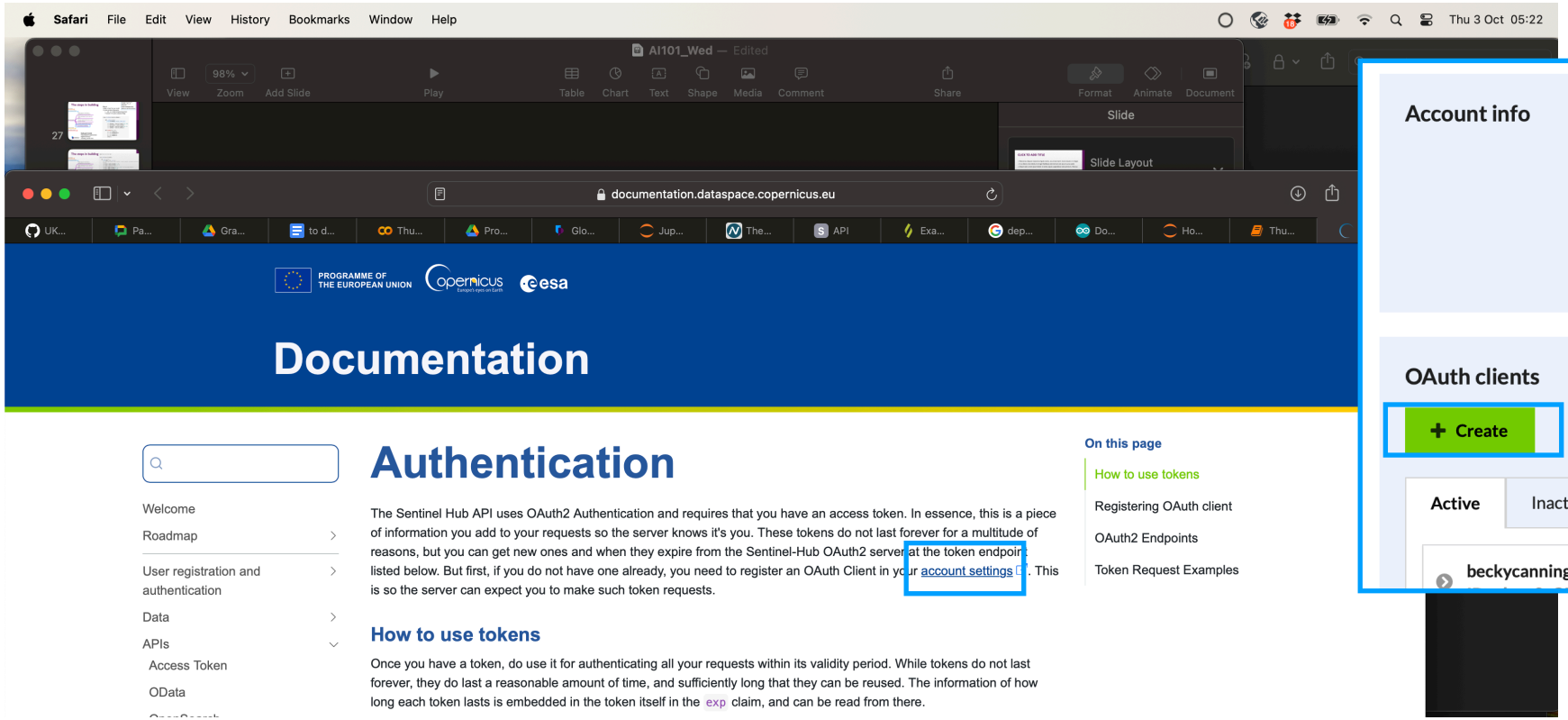
UNIVERSITY OF PORTSMOUTH

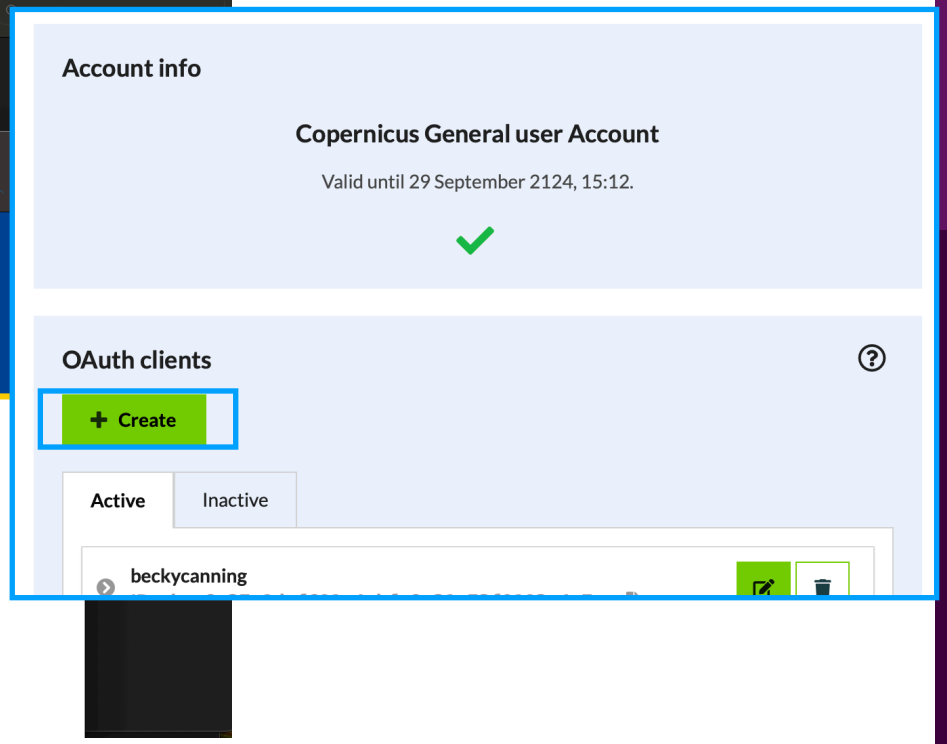
Accessing ESA data through API

Signing up and authenticating on ESA for access to ESA APIs

- https://documentation.dataspace.copernicus.eu/APIs/SentinelHub/Overview/Authentication.html#python

Register - the follow directions for 'Registering OAuth client' on the same page add this to you dashboard



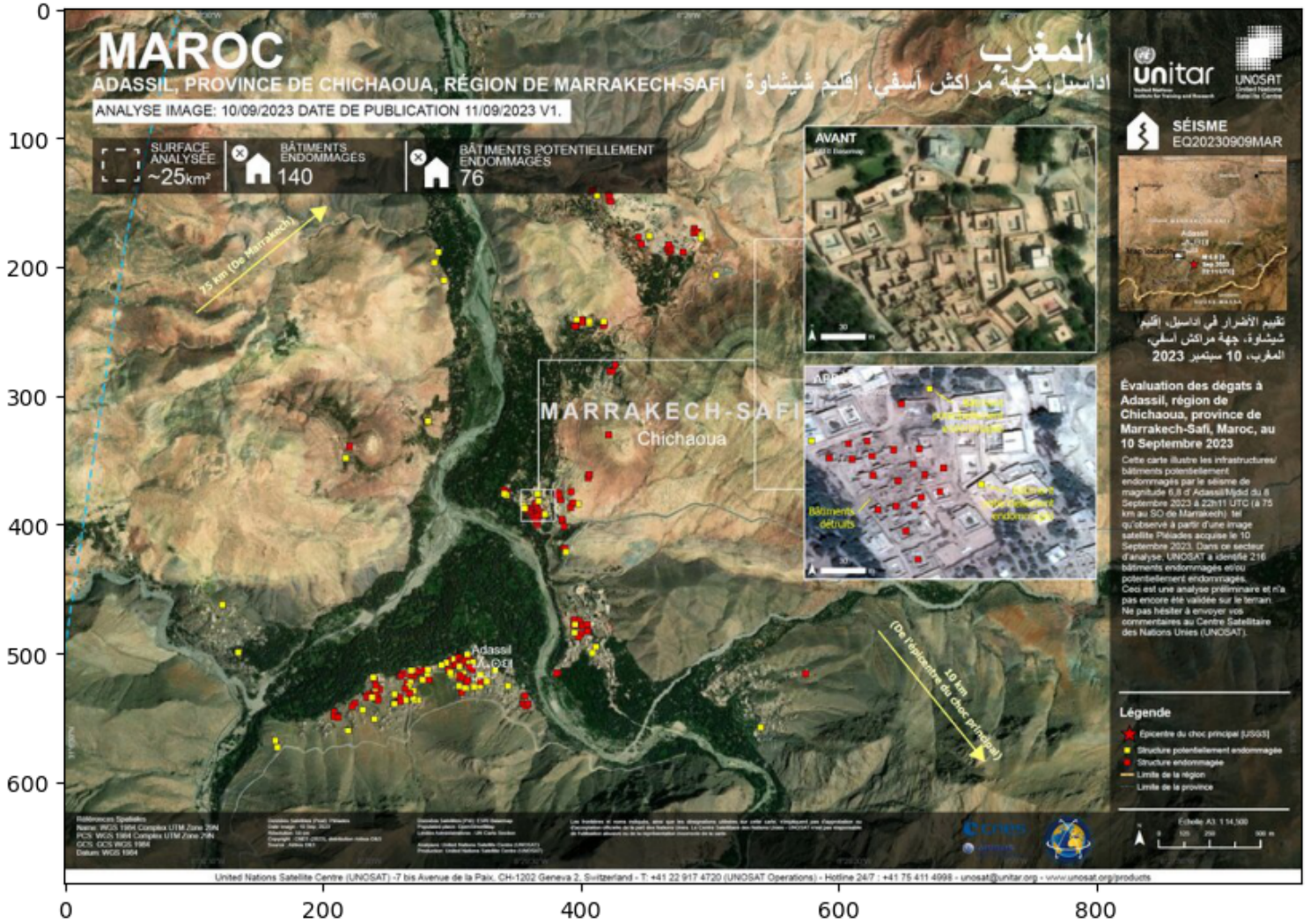


UNIVERSITY OF PORTSMOUTH

Starting from the end

- DL case study with thanks to Obinna

- A DL model for Earth Observation data



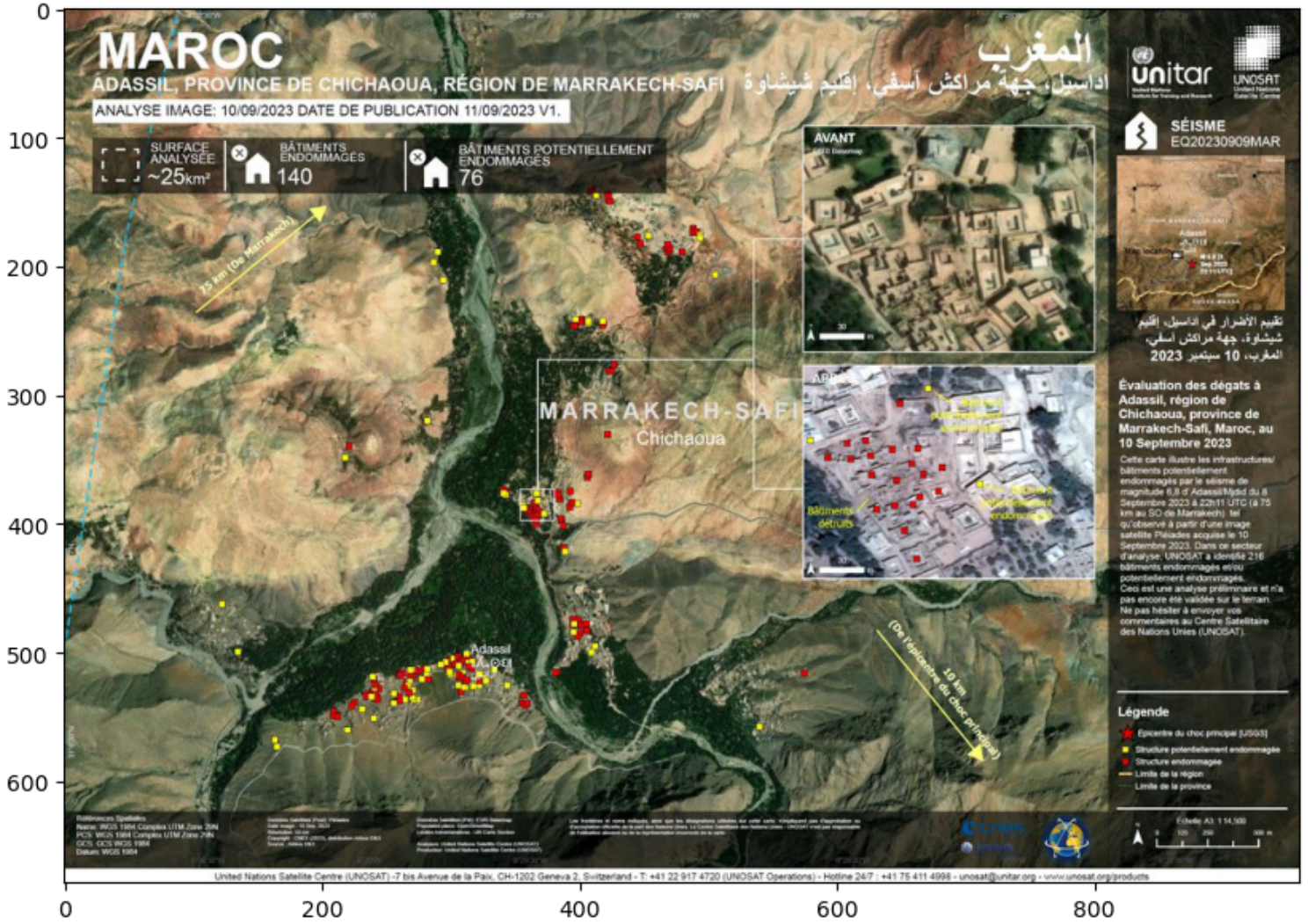
There was a 6.8 magnitude earthquake that struck the Atlas Mountains, about 75 km from Marrakech, Morocco on 8 September, 2023

Can we use satellite DL to better understand the destruction to the buildings. Identification of secure and insecure structures could help people affected?

Starting from the end

- DL case study with thanks to Obinna

- A DL model for Earth Observation data



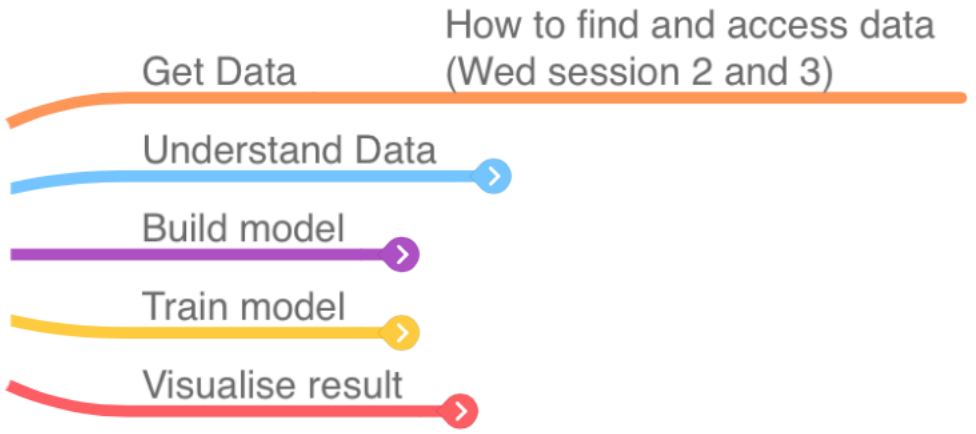
What are the steps we would take?

Starting from the end

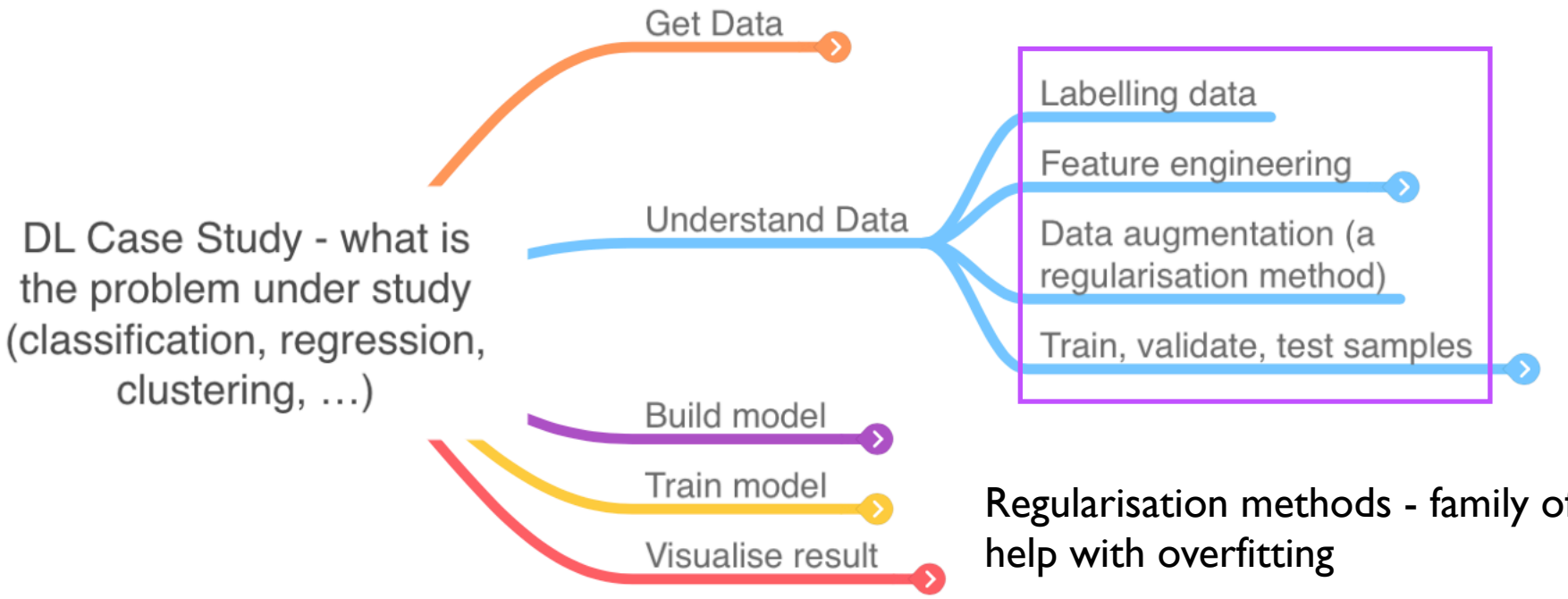
DL Case Study - what is the problem under study (classification, regression, clustering, …)

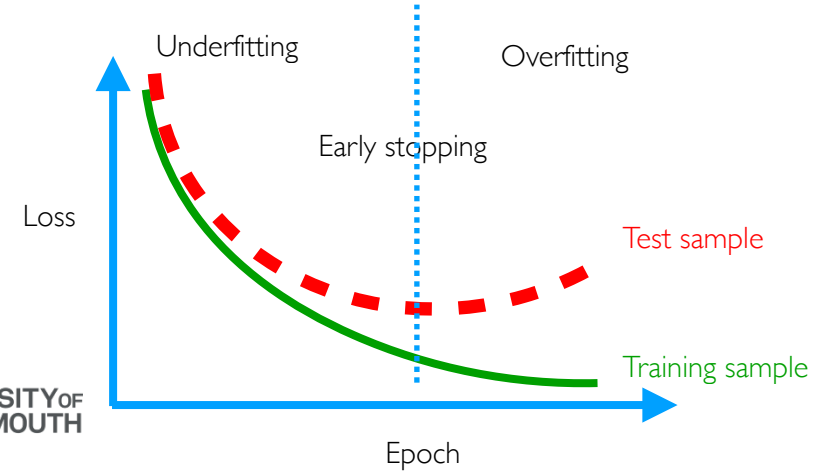
Starting from the end

DL Case Study - what is the problem under study (classification, regression, clustering, …)



Starting from the end





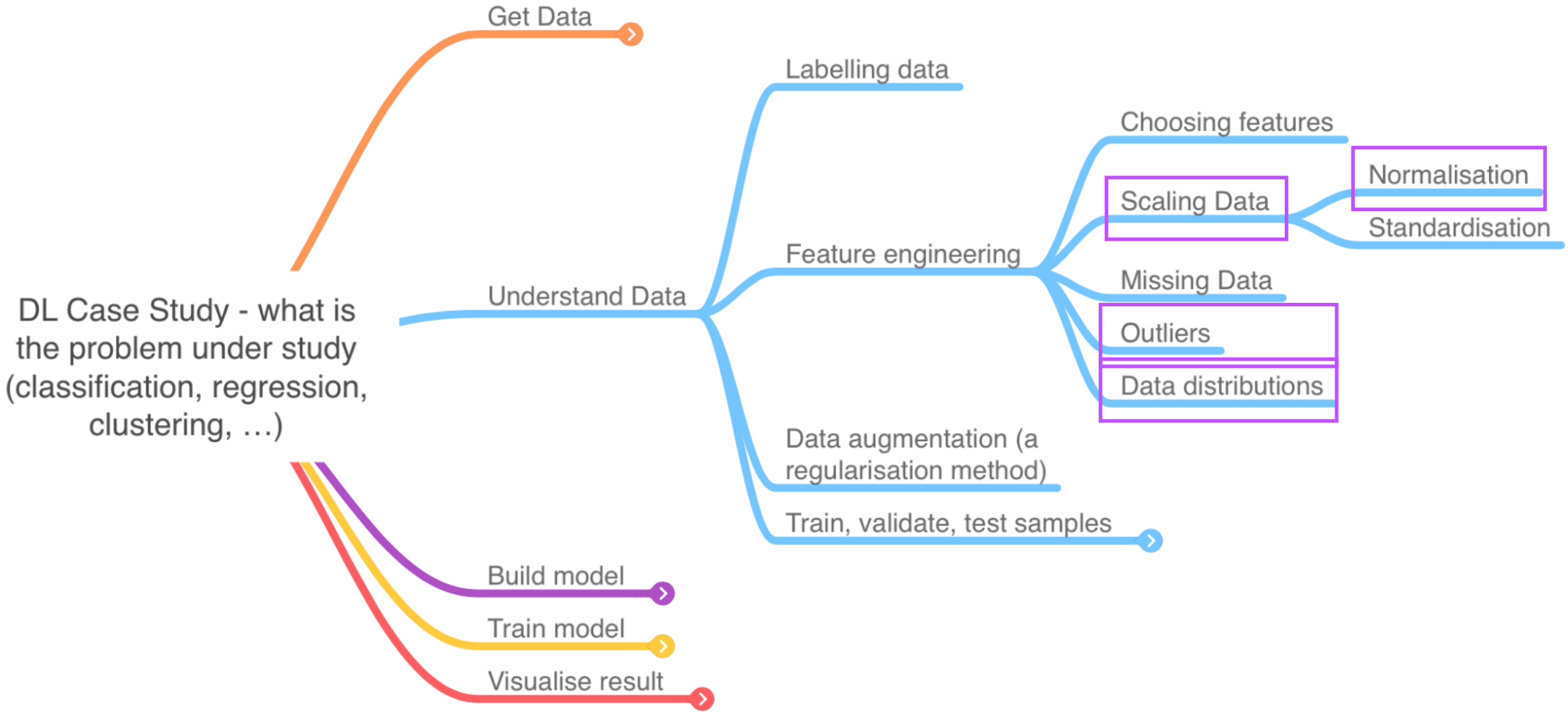
Regularisation methods - family of different methods which help with overfitting

Trade off between bias and variance

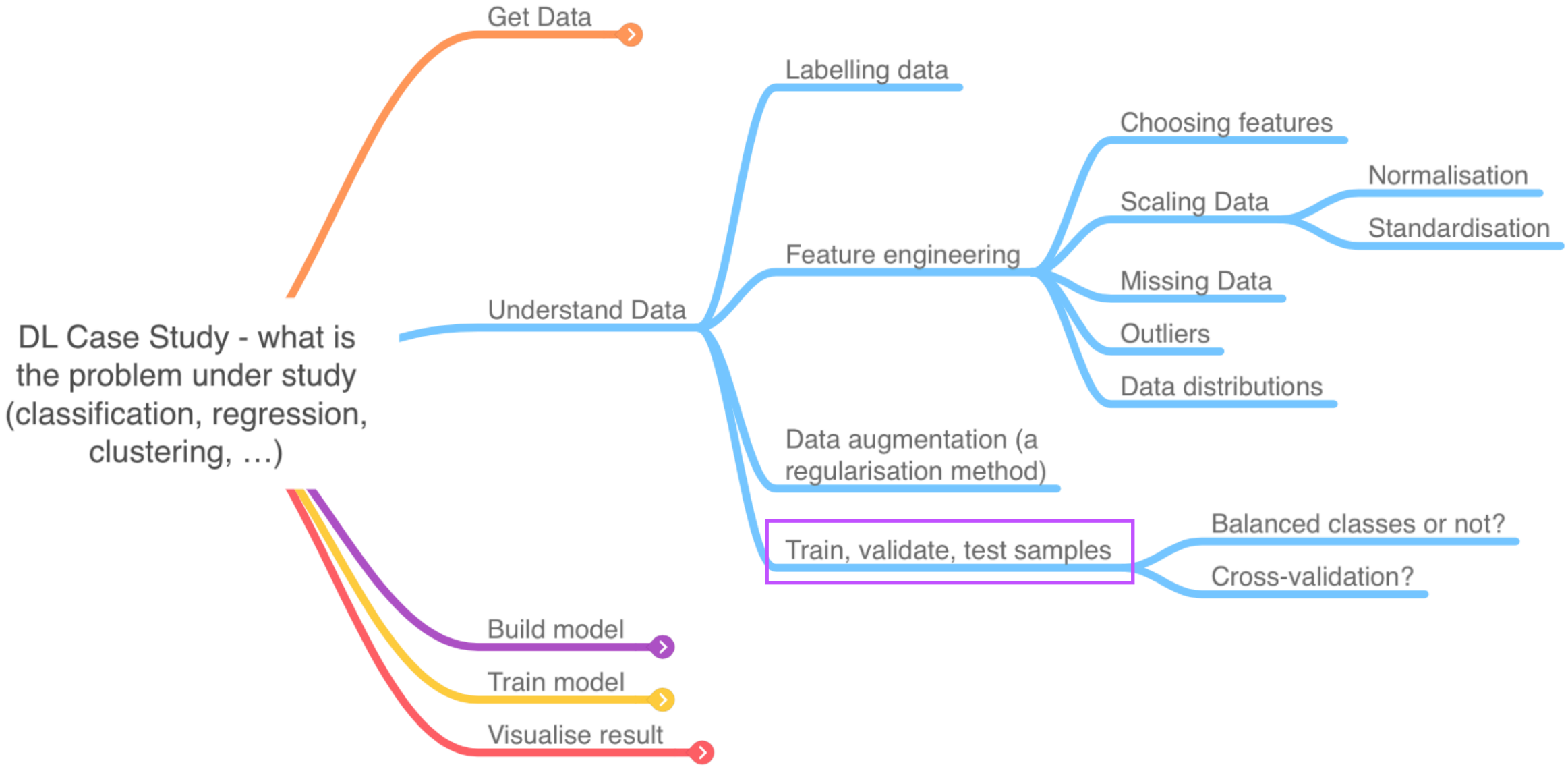
Bias: high bias is a high training error - accuracy on training set is low

Variance: high variance means a lack of generalisation - the accuracy on a specific training or validation set may be high but the variance in accuracy across many sets is low

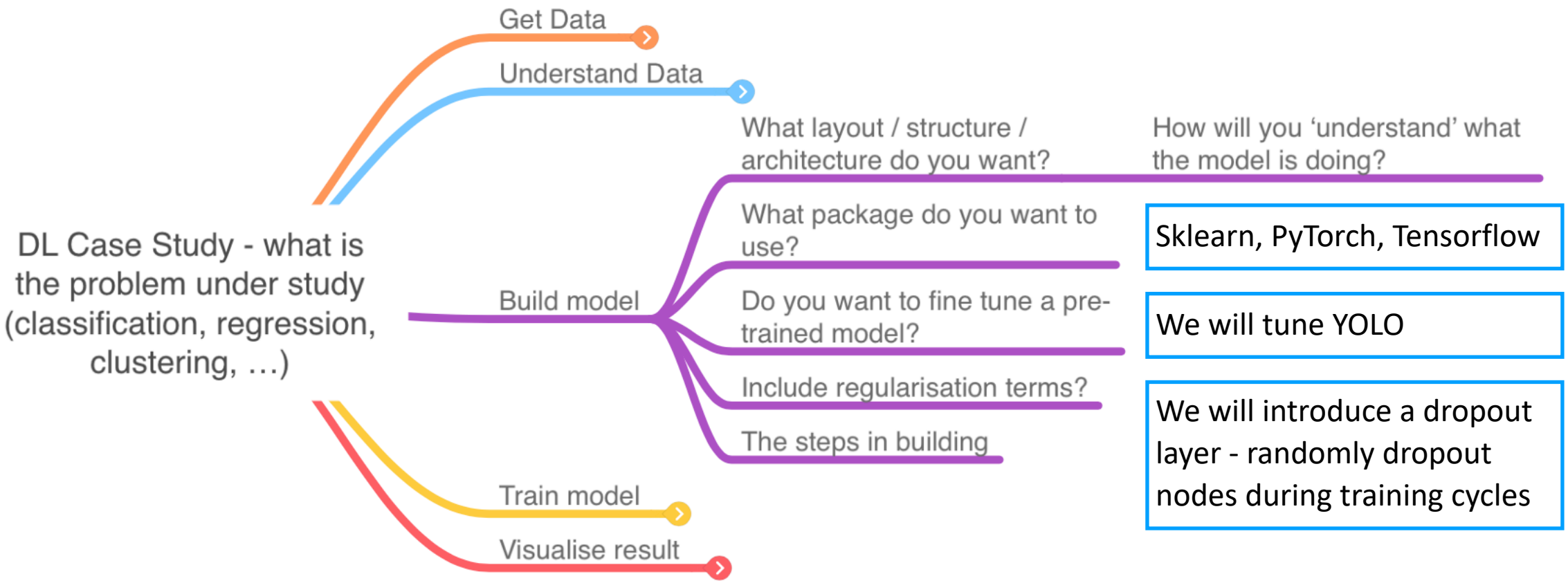
Starting from the end



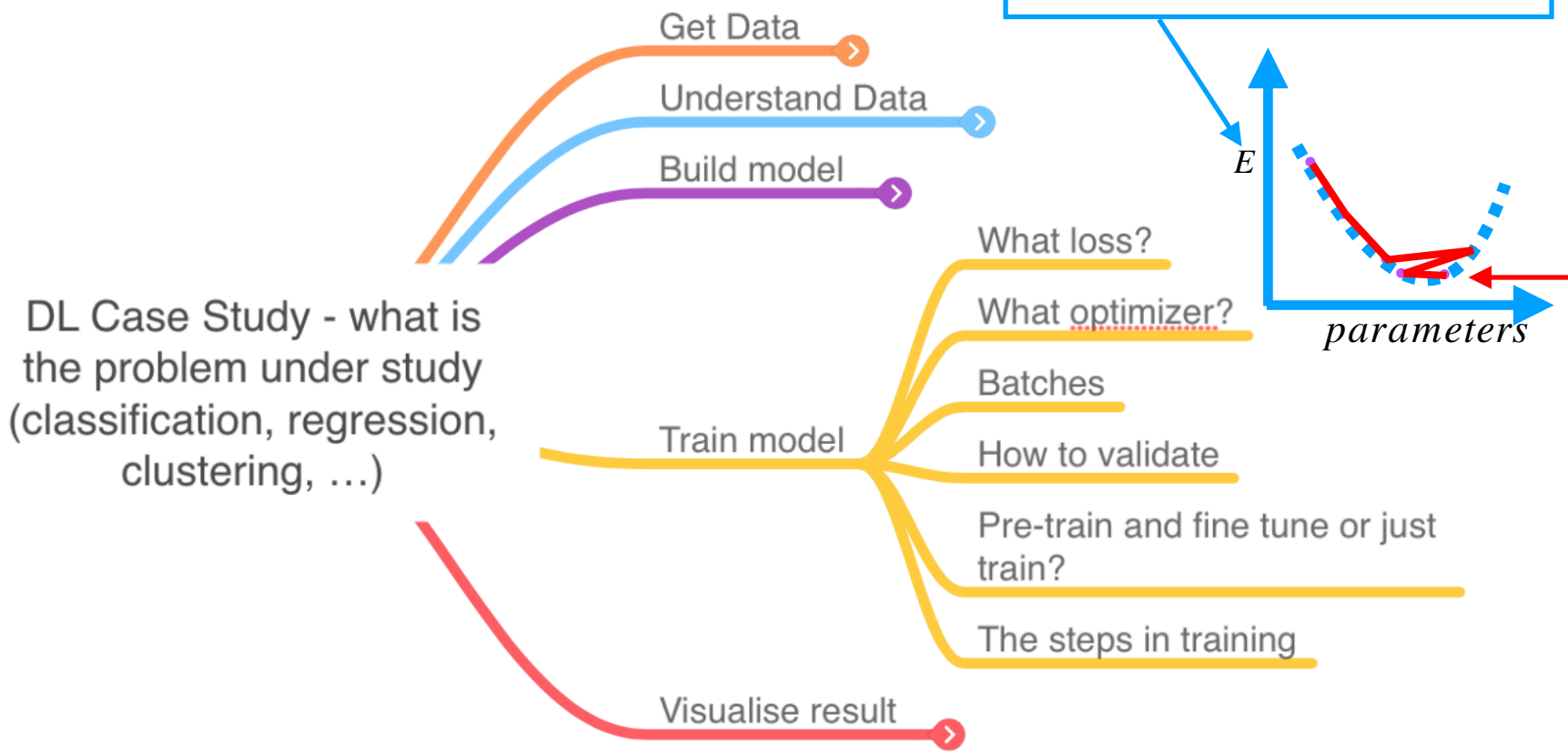
Starting from the end



Starting from the end



Starting from the end



Loss/cost - difference between your model outputs and the 'truth'

Our loss functions

We will use a cross entropy for classification (binary or categorical).

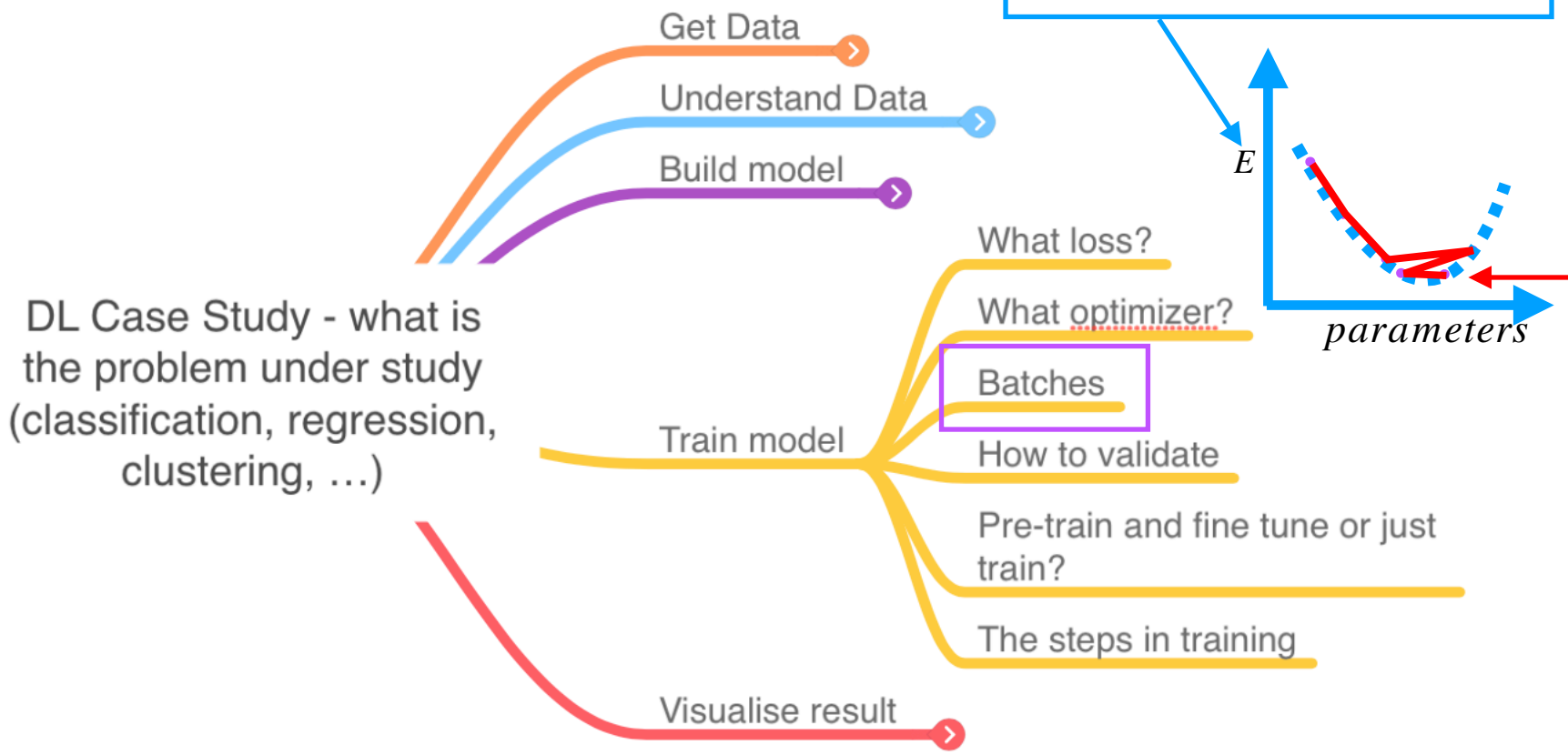
We will use mean squared error for regression.

Optimisers tell us how to step about in our parameters space.

Typically gradient decent (you pick learning rate or 'step size') or adaptive (it adapts learning rate).

We will use Adam

Starting from the end



Loss/cost - difference between your model outputs and the 'truth'

Our loss functions

We will use a cross entropy for classification (binary or categorical).

We will use mean squared error for regression.

Optimisers tell us how to step about in our parameters space.

Typically gradient decent (you pick learning rate or 'step size') or adaptive (it adapts learning rate).
We will use Adam .

Generally different algorithms change how they process batches.

UNIVERSITY OF PORTSMOUTH

Samples, batches, epochs

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

My input dataset

Samples, batches, epochs

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

My input dataset

$[x_1]$

A data sample

Samples, batches, epochs

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

My input dataset

$$[x_1]$$

A data sample

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

A batch of data - it is made of some number of samples

The algorithm works through a batch of samples then compares output to truth and updates algorithm.

Stochastic gradient descent works on a batch of only one sample.

Samples, batches, epochs

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

My input dataset

$$[x_1]$$

A data sample

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

A batch of data - it is made of some number of samples

The algorithm works through a batch of samples then compares output to truth and updates algorithm.

Stochastic gradient descent works on a batch of only one sample.

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \rightarrow \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \ldots \begin{bmatrix} x_{n-2} \\ x_{n-1} \\ x_n \end{bmatrix}$$

Samples, batches, epochs

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

My input dataset

$$[x_1]$$

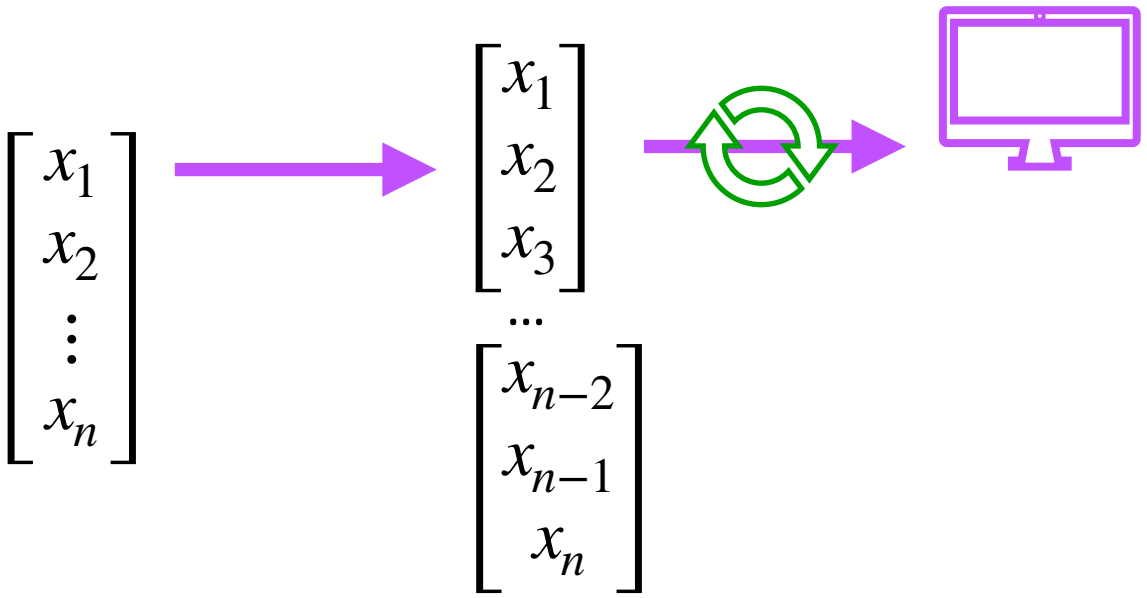
A data sample

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

A batch of data - it is made of some number of samples

The algorithm works through a batch of samples then compares output to truth and updates algorithm.

Stochastic gradient descent works on a batch of only one sample.



Samples, batches, epochs

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

My input dataset

$$[x_1]$$

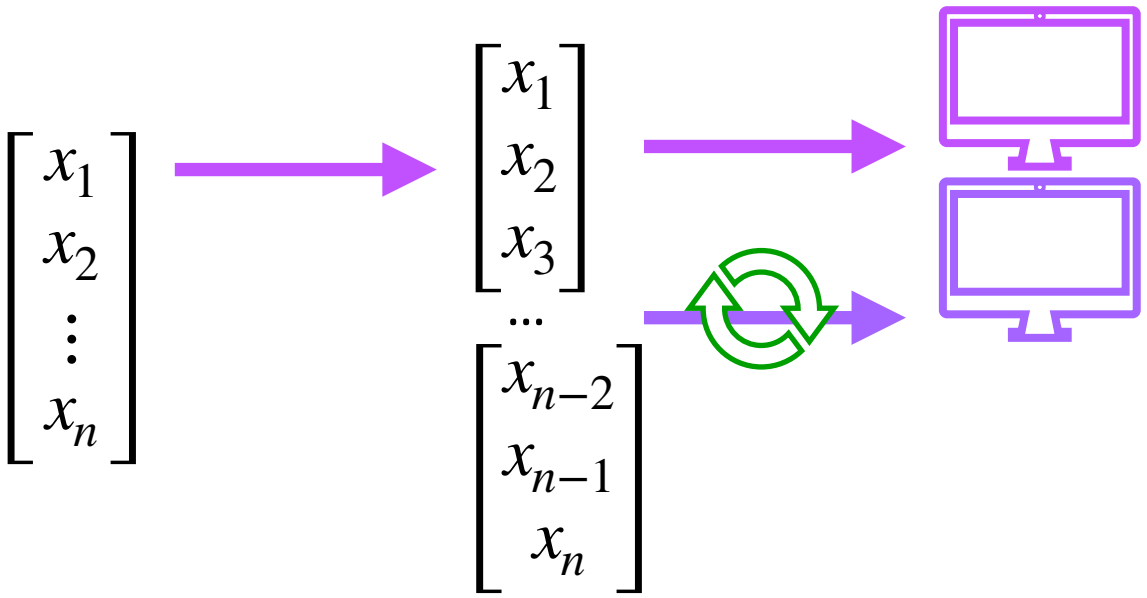
A data sample

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

A batch of data - it is made of some number of samples

The algorithm works through a batch of samples then compares output to truth and updates algorithm.

Stochastic gradient descent works on a batch of only one sample.



Samples, batches, epochs

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

My input dataset

$$[x_1]$$

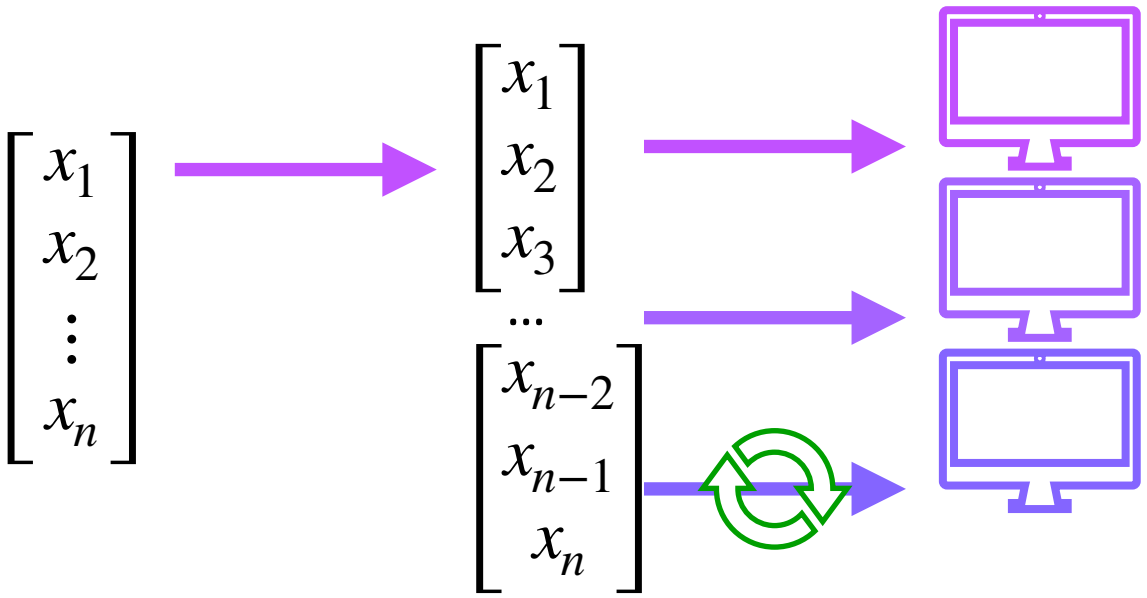
A data sample

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

A batch of data - it is made of some number of samples

The algorithm works through a batch of samples then compares output to truth and updates algorithm.

Stochastic gradient descent works on a batch of only one sample.



Samples, batches, epochs

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

My input dataset

$$[x_1]$$

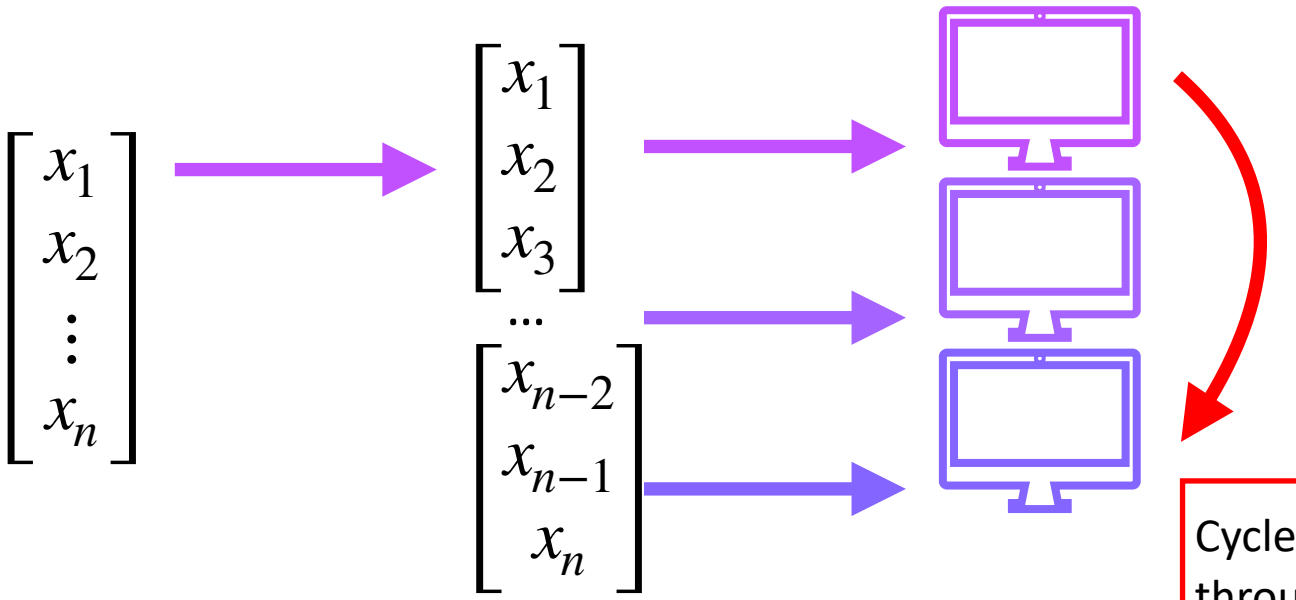
A data sample

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

A batch of data - it is made of some number of samples

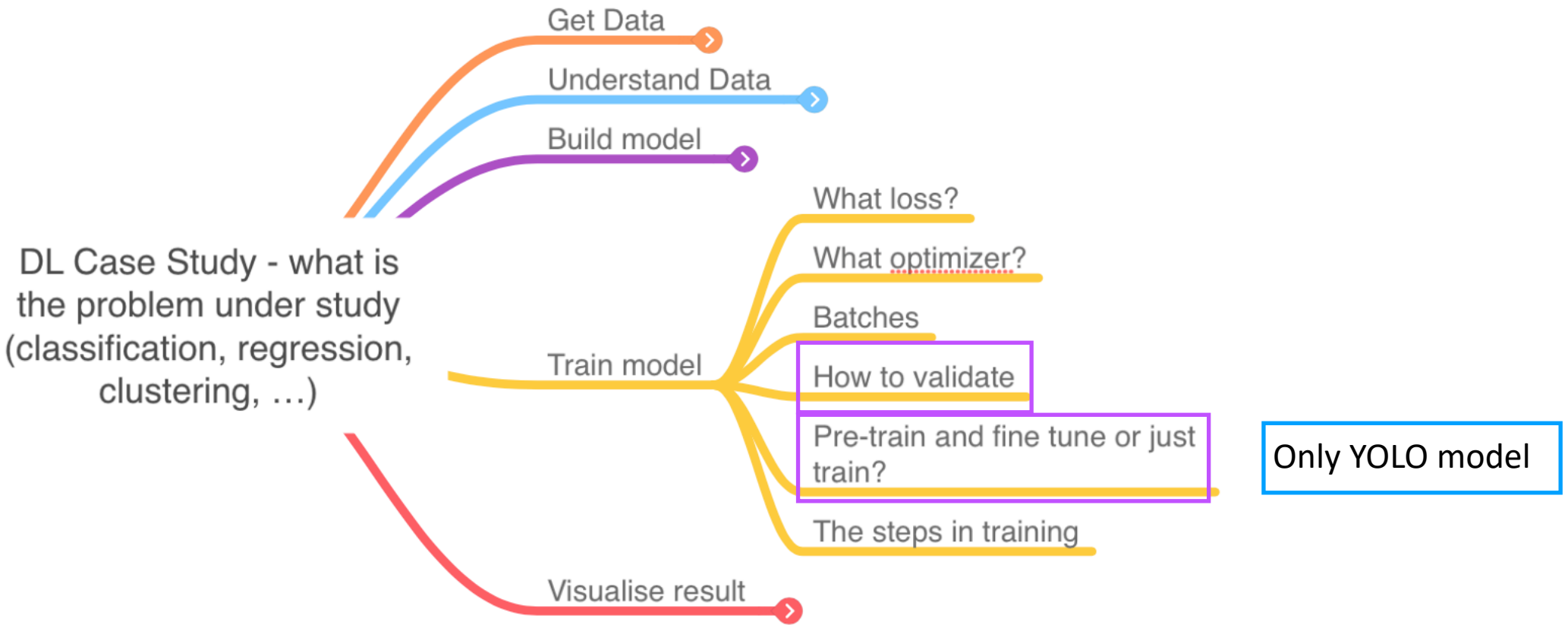
The algorithm works through a batch of samples then compares output to truth and updates algorithm.

Stochastic gradient descent works on a batch of only one sample.



Cycle of learning once through dataset is called an Epoch

Starting from the end



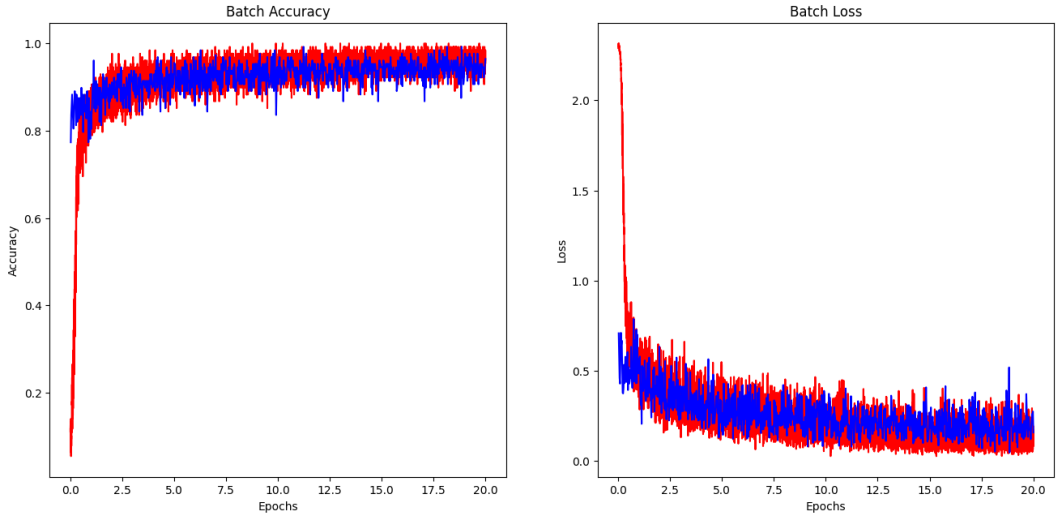
Only YOLO model

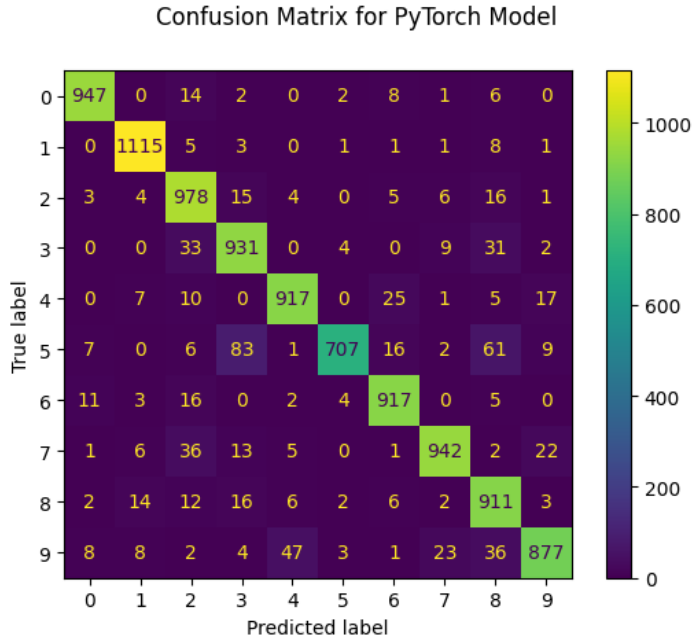
Starting from the end

DL Case Study - what is the problem under study (classification, regression, clustering, …)

- Get Data
- Understand Data
- Build model
- Train model
- Visualise result
  - Bias and variance
  - Early stopping
  - Common visuals

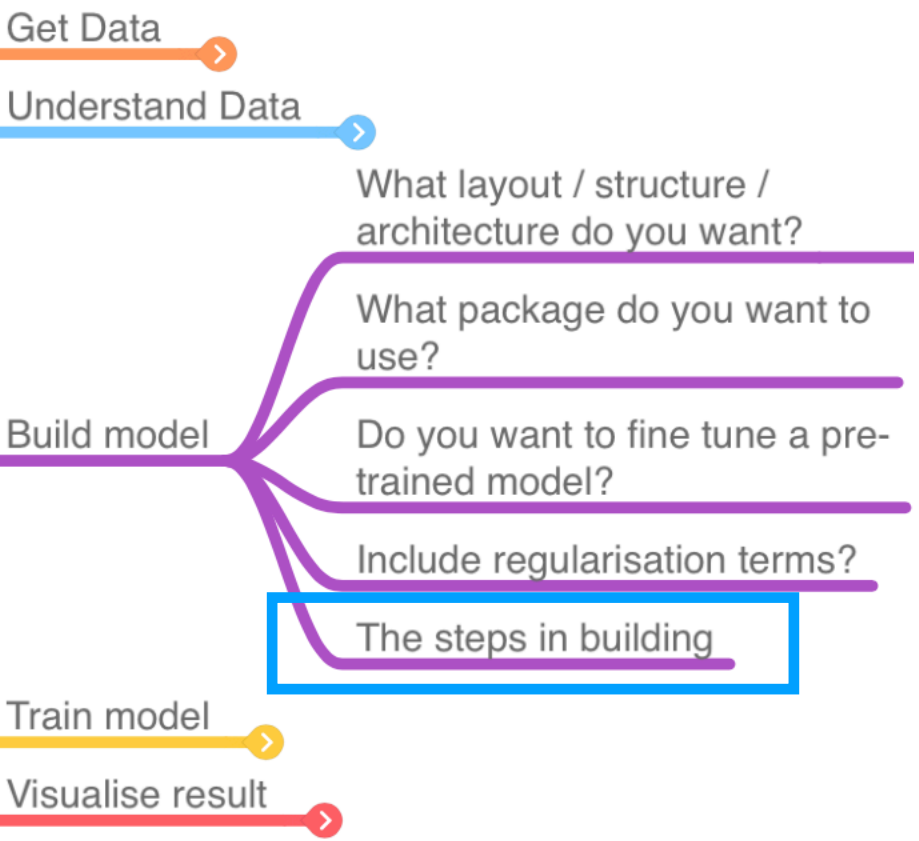
There are packages for visualisation v popular one is tensor board for tensorflow - we won't have time to use these





UNIVERSITY OF PORTSMOUTH

The steps in building



Objects - things with properties and methods

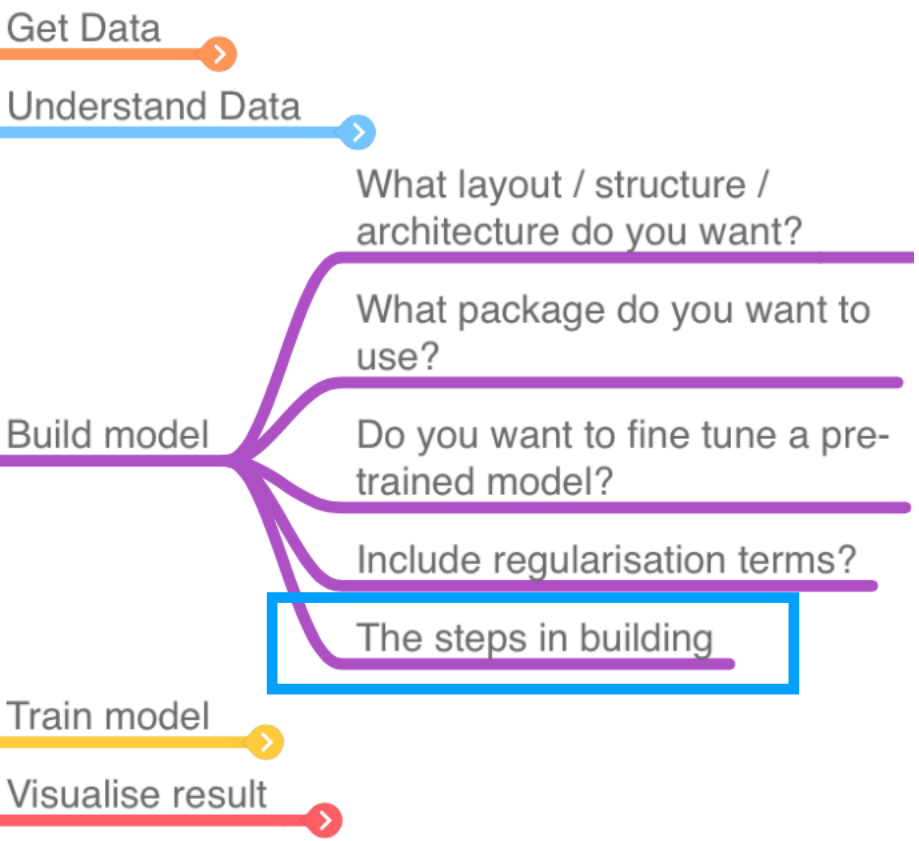
PyTorch

- Make a 'class' for your model object
- A class consists of two parts:
  - __init__() - model components go here
  - forward - this parts computes things

```
class TinyModel(torch.nn.Module):

    def __init__(self):
        super(TinyModel, self).__init__()

        self.linear1 = torch.nn.Linear(100, 200)
        self.activation = torch.nn.ReLU()
        self.linear2 = torch.nn.Linear(200, 10)
        self.softmax = torch.nn.Softmax()

    def forward(self, x):
        x = self.linear1(x)
        x = self.activation(x)
        x = self.linear2(x)
        x = self.softmax(x)
        return x
```

UNIVERSITY OF PORTSMOUTH

The steps in building



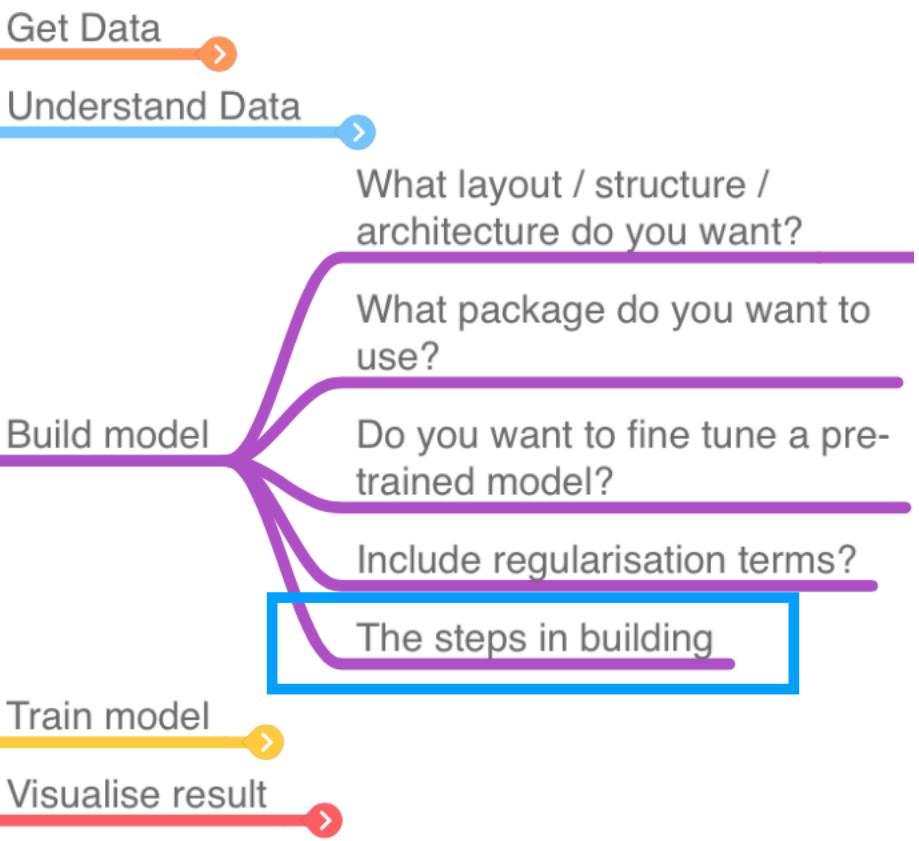
PyTorch

- Make a ‘class’ for your model
- A class consists of two parts:
  - __init__() - model components go here
  - forward - this parts computes things

```
class TinyModel(torch.nn.Module):

    def __init__(self):
        super(TinyModel, self).__init__()

        self.linear1 = torch.nn.Linear(100, 200)
        self.activation = torch.nn.ReLU()
        self.linear2 = torch.nn.Linear(200, 10)
        self.softmax = torch.nn.Softmax()

    def forward(self, x):
        x = self.linear1(x)
        x = self.activation(x)
        x = self.linear2(x)
        x = self.softmax(x)
        return x
```

UNIVERSITY OF PORTSMOUTH

The steps in building



Really good tutorial:
https://pytorch.org/tutorials/beginner/introyt/modelsyt_tutorial.html

UNIVERSITY OF PORTSMOUTH

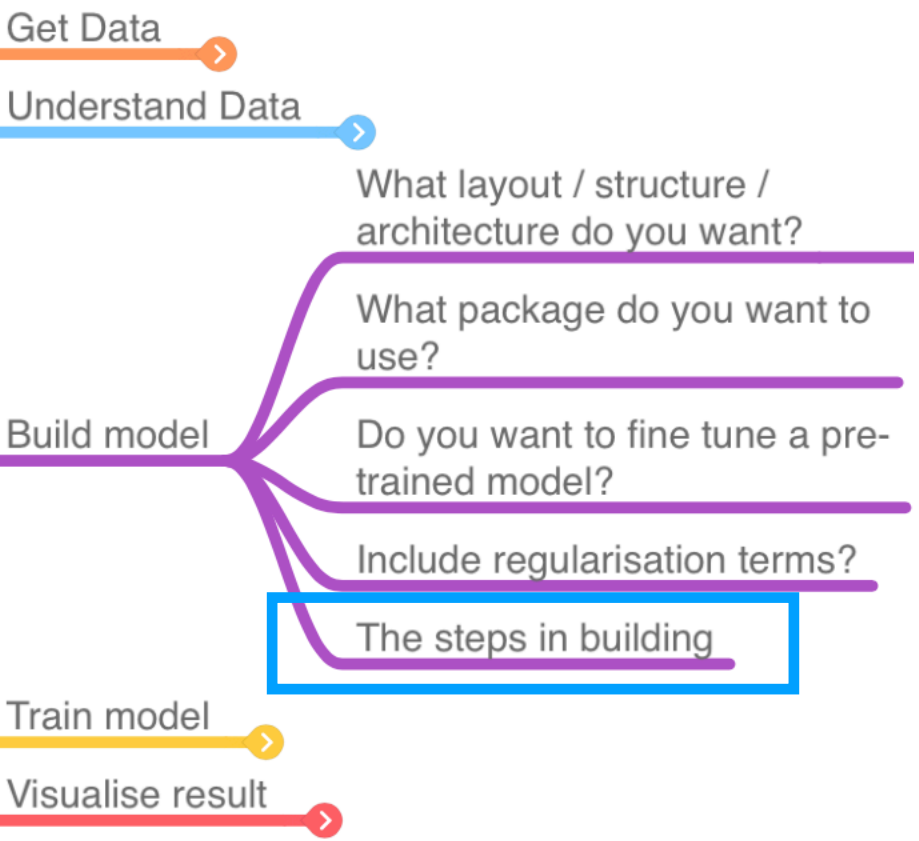
Sometimes seen as:
super()__init__()
Gives the torch.nn.Module class ability to call this function

PyTorch

- Make a 'class' for your model
- A class consists of two parts:
  - __init__() - model components go here
  - forward - this parts computes things

```
class TinyModel(torch.nn.Module):

    def __init__(self):
        super(TinyModel, self).__init__()

        self.linear1 = torch.nn.Linear(100, 200)
        self.activation = torch.nn.ReLU()
        self.linear2 = torch.nn.Linear(200, 10)
        self.softmax = torch.nn.Softmax()

    def forward(self, x):
        x = self.linear1(x)
        x = self.activation(x)
        x = self.linear2(x)
        x = self.softmax(x)
        return x
```

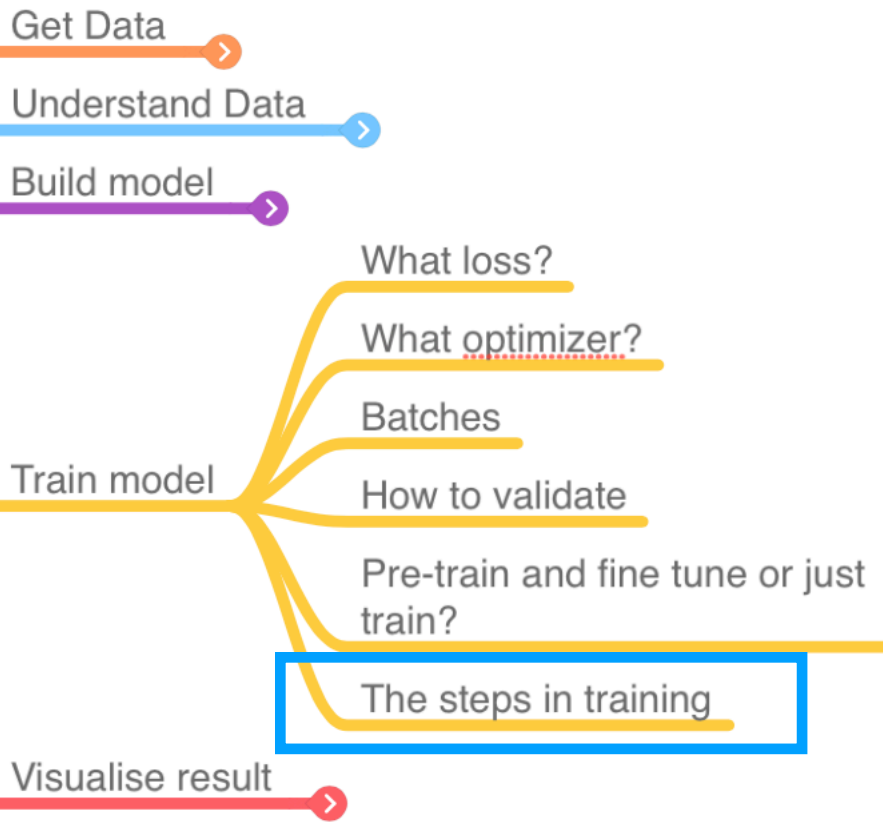
The steps in building




```python
import torch.functional as F


class LeNet(torch.nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        # 1 input image channel (black & white), 6 output channels, 5x5 square convolution
        # kernel
        self.conv1 = torch.nn.Conv2d(1, 6, 5)
        self.conv2 = torch.nn.Conv2d(6, 16, 3)
        # an affine operation: y = Wx + b
        self.fc1 = torch.nn.Linear(16 * 6 * 6, 120)  # 6*6 from image dimension
        self.fc2 = torch.nn.Linear(120, 84)
        self.fc3 = torch.nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square you can only specify a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

    def num_flat_features(self, x):
        size = x.size()[1:]  # all dimensions except the batch dimension
        num_features = 1
        for s in size:
            num_features *= s
        return num_features
```

UNIVERSITY OF PORTSMOUTH

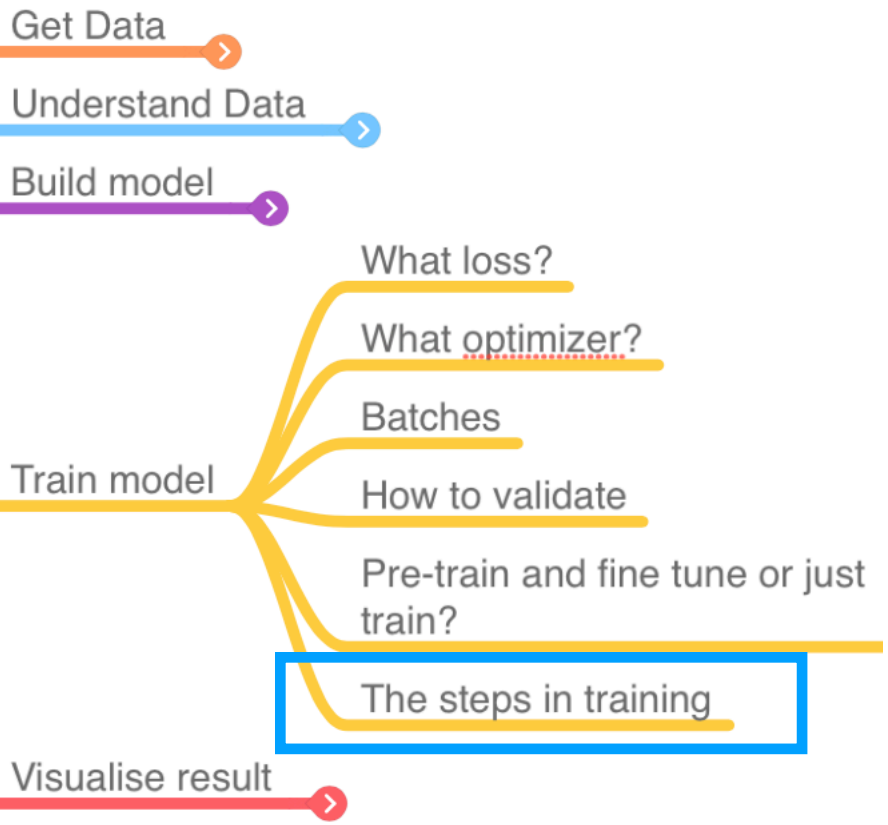
The steps in training



- Put model into 'train' mode: model.train()
- For each 'batch':
  - Reset gradient to zero - incase not
  - Predict outputs from inputs
  - Calculate loss
  - Compute gradients using backpropagation
  - Update parameters
  - Compute accuracy and loss

Really good doc to read:
https://pytorch.org/tutorials/beginner/introyt/trainingyt.html

The steps in training

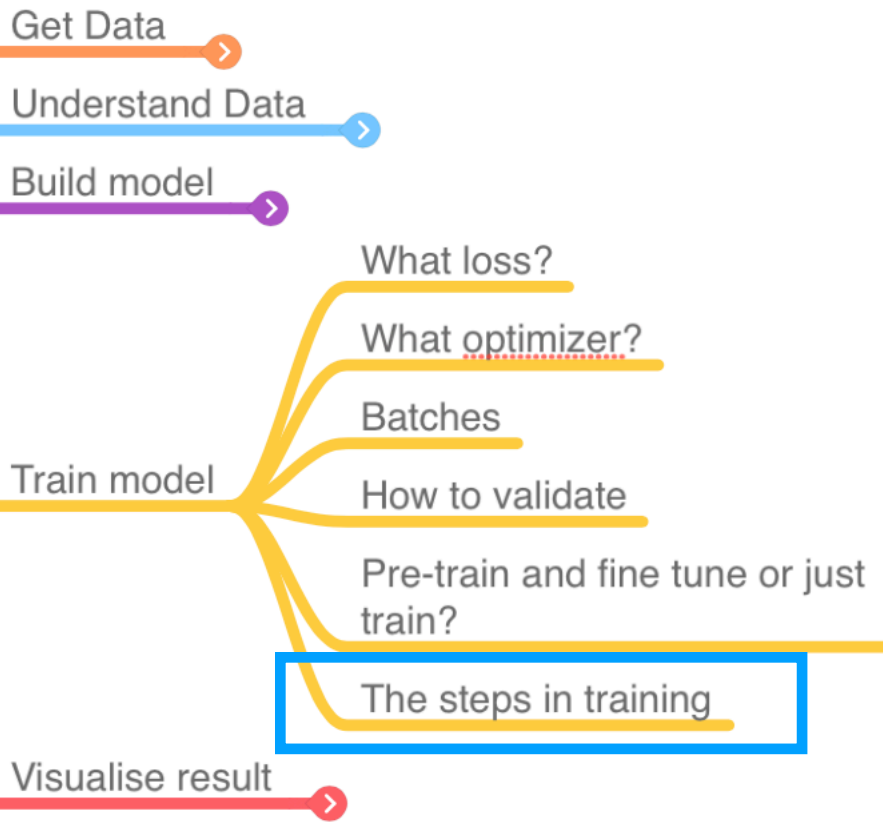


- Put model into 'train' mode: model.train()
- For each 'batch':
  - Reset gradient to zero - incase not
  - Predict outputs from inputs
  - Calculate loss
  - Compute gradients using backpropagation
  - Update parameters
  - Compute accuracy and loss

```
model.train() # model is now in 'train' mode
for epoch in range(5):
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad() #reset grdient to zero
        outputs = model(images) # predict outputs from inputs
        loss = nn.CrossEntropyLoss()(outputs, labels) # calculate loss
        loss.backward() # compute gradients using backpropagation
        optimizer.step() # update parameters
```

Really good doc to read:
https://pytorch.org/tutorials/beginner/introyt/trainingyt.html

The steps in training

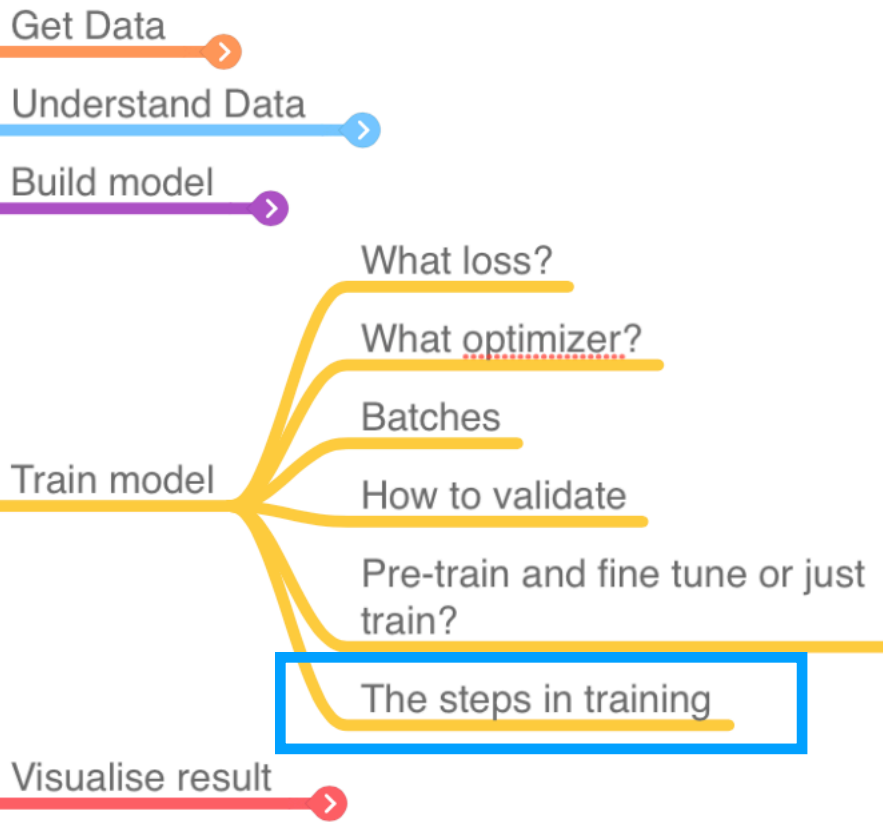


- Put model into 'train' mode: model.train()
- For each 'batch':
  - Reset gradient to zero - incase not
  - Predict outputs from inputs
  - Calculate loss
  - Compute gradients using backpropagation
  - Update parameters
  - Compute accuracy and loss

```
model.train() # model is now in 'train' mode
for epoch in range(5):
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad() #reset grdient to zero
        outputs = model(images) # predict outputs from inputs
        loss = nn.CrossEntropyLoss()(outputs, labels) # calculate loss
        loss.backward() # compute gradients using backpropagation
        optimizer.step() # update parameters
```

Really good doc to read:
https://pytorch.org/tutorials/beginner/introyt/trainingyt.html

The steps in training



- Put model into 'train' mode: model.train()
- For each 'batch':
  - Reset gradient to zero - incase not
  - Predict outputs from inputs
  - Calculate loss
  - Compute gradients using backpropagation
  - Update parameters
  - Compute accuracy and loss

```
model.train() # model is now in 'train' mode
for epoch in range(5):
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad() #reset grdient to zero
        outputs = model(images) # predict outputs from inputs
        loss = nn.CrossEntropyLoss()(outputs, labels) # calculate loss
        loss.backward() # compute gradients using backpropagation
        optimizer.step() # update parameters
```

Really good doc to read:
https://pytorch.org/tutorials/beginner/introyt/trainingyt.html

AI 101 - Wed + Thurs

- **TASK:** Build a model in PyTorch from scratch to classify the fashionMNIST dataset
  - **Aim to understand and apply:**
    - Exploring the data - how many categories and labels - why do we do this - how might data distributions affect your model output?
    - Anatomy of a model - steps for a model, steps for training
    - Key steps to evaluate model progress

- **For reference or for the aficionados!**
  - 2 Clustering notebooks - sklearn and NN (autoencoders)

- **DL - Thurs 11am - an example applied to remote sensing data**
  - **Aim:** to understand the jargon so you could pull this tutorial apart and understand its key features

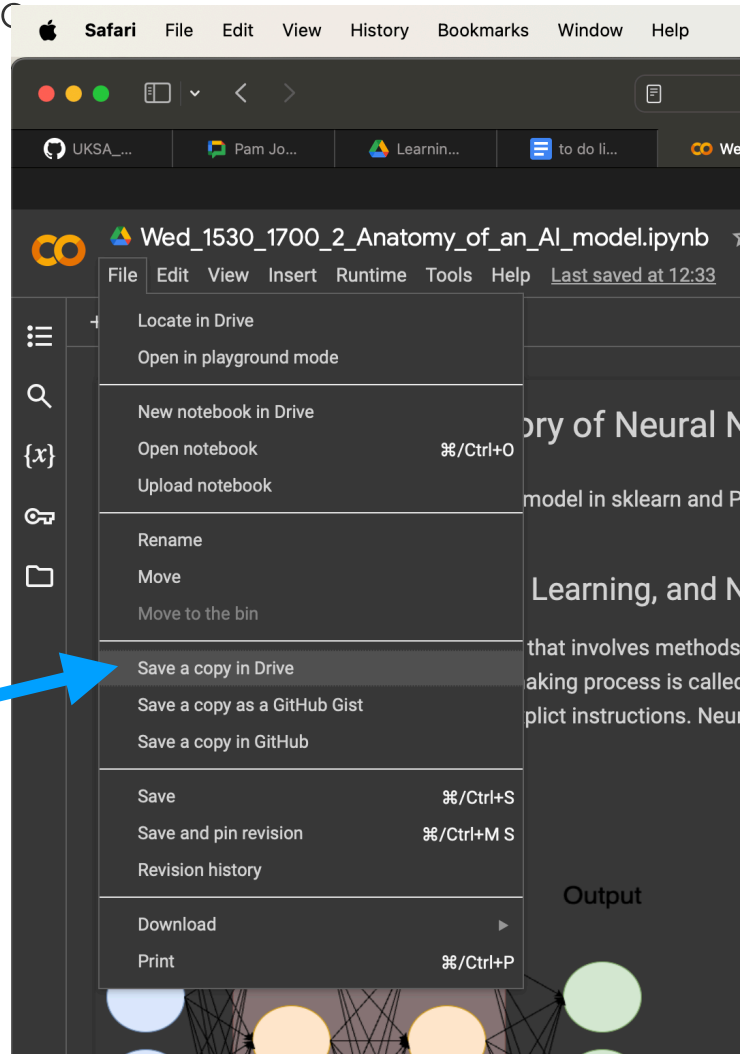
UNIVERSITY OF PORTSMOUTH

Plan A (ish)

- If on windows or linux: https://148.197.234.100:32444/ sign-up on Jupyter http://148.197.234.100:32088
- If on Mac… Username and password, ssh port forwarding e.g. I would do
  - ssh -L 8443:localhost:32444 becky@148.197.234.100

Plan B

- Go to Moodle -> Go to Phase 2 -> Click link for Plan B (or click this link https://drive.google.com/drive/folders/1nS4kCUwKO7gAI_TUjEcADtDSjfb_aq8b?usp=share_link)
- Double click on a notebook: it will open in colab
- Go File -> Save a copy in Drive: this will open a new copy on your browser - edit this copy as you please



UNIVERSITY OF PORTSMOUTH

The steps in building

PyTorch

- Make a ‘class’ for your model
- A class consists of two parts:
  - __init__() - model components go here
  - forward - this parts computes things

```
class TinyModel(torch.nn.Module):

    def __init__(self):
        super(TinyModel, self).__init__()

        self.linear1 = torch.nn.Linear(100, 200)
        self.activation = torch.nn.ReLU()
        self.linear2 = torch.nn.Linear(200, 10)
        self.softmax = torch.nn.Softmax()

    def forward(self, x):
        x = self.linear1(x)
        x = self.activation(x)
        x = self.linear2(x)
        x = self.softmax(x)
        return x
```

Tensorflow

- Model flow defined in one function
- A class consists of two parts:
  - __init__() - model components go here
  - forward - this parts computes things

```
from tensorflow.keras import layers, models

# Define the model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

The steps in training

PyTorch
- Put model into 'train' mode: model.train()
- For each 'batch':
  - Reset gradient to zero - incase not
  - Predict outputs from inputs
  - Calculate loss
  - Compute gradients using backpropagation
  - Update parameters
  - Compute accuracy and loss

Tensorflow
- Compile model
- Then fit model
  - 'Fit' does all the other things

```
# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
model.fit(train_images, train_labels, epochs=5)
```

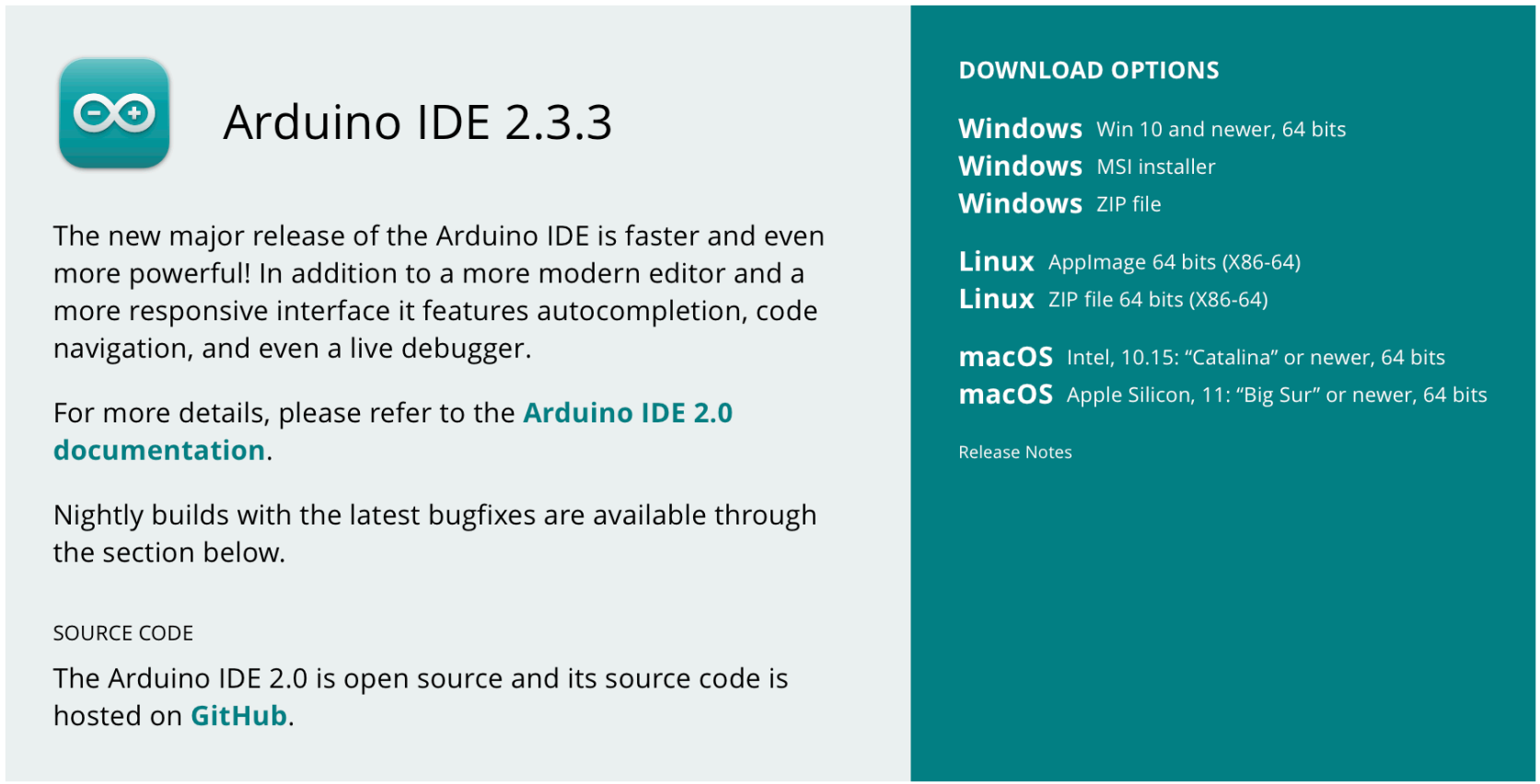
```
model.train() # model is now in 'train' mode
for epoch in range(5):
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad() #reset grdient to zero
        outputs = model(images) # predict outputs from inputs
        loss = nn.CrossEntropyLoss()(outputs, labels) # calculate loss
        loss.backward() # compute gradients using backpropagation
        optimizer.step() # update parameters
```

Deploying a model

Get the Arduino IDE

- https://www.arduino.cc/en/software

Follow install and open IDE



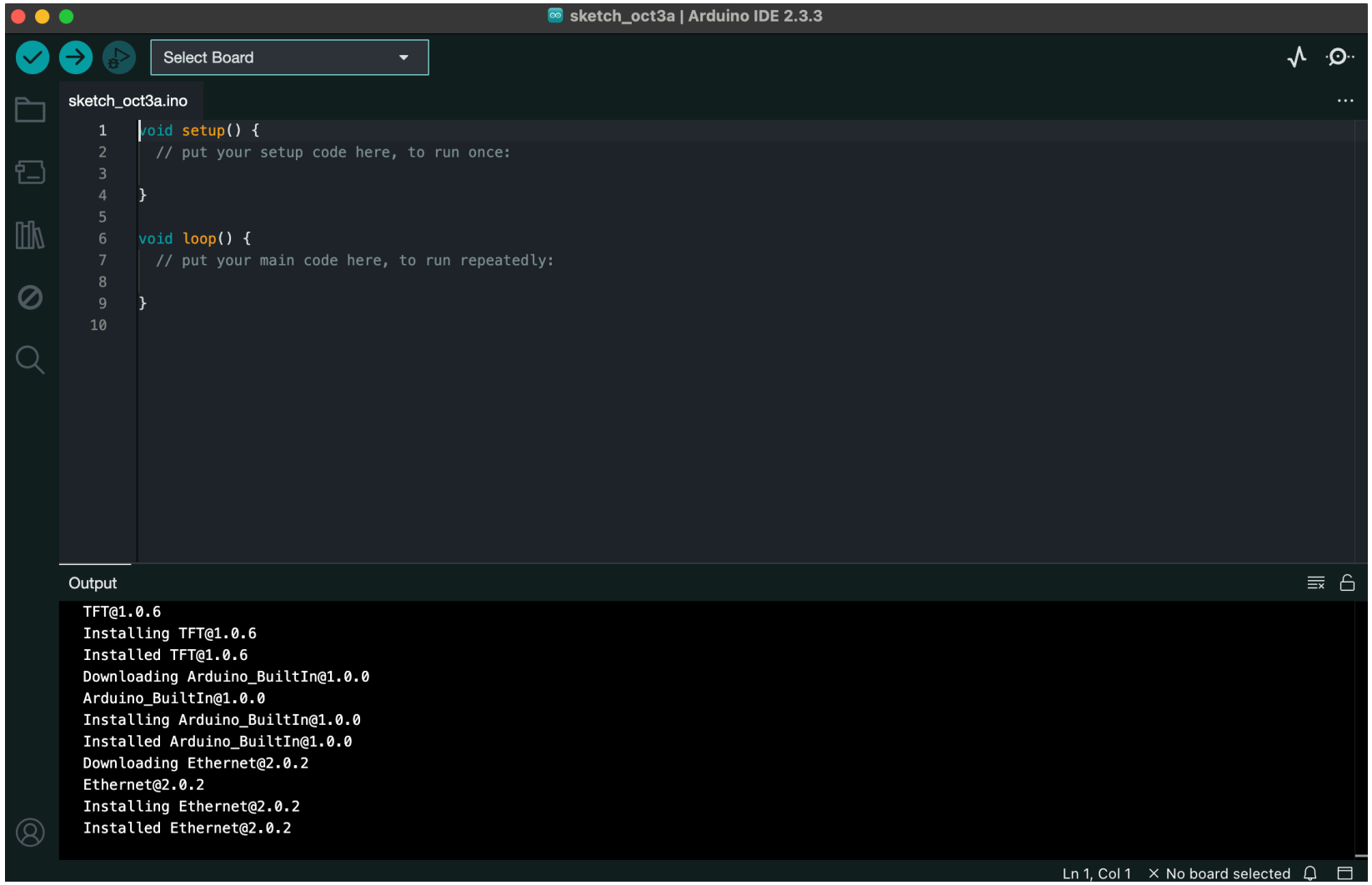
UNIVERSITY OF PORTSMOUTH

Deploying a model

Get the Arduino IDE

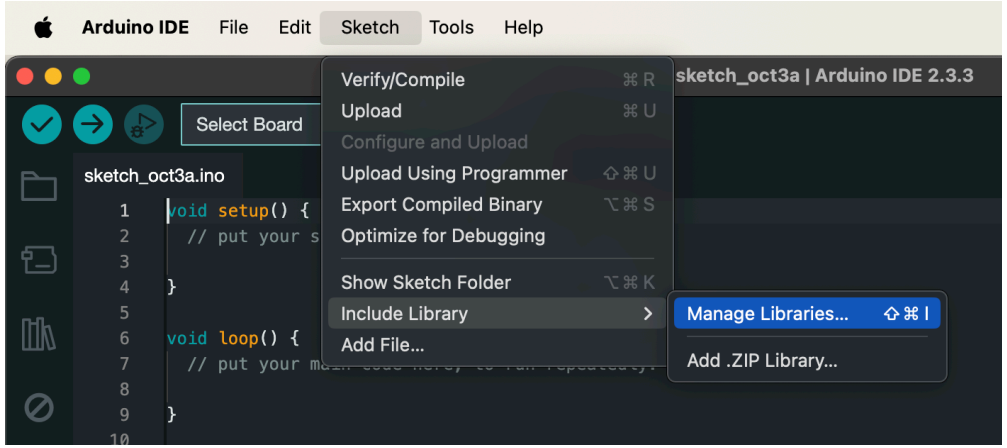
- https://www.arduino.cc/en/software

Follow install and open IDE

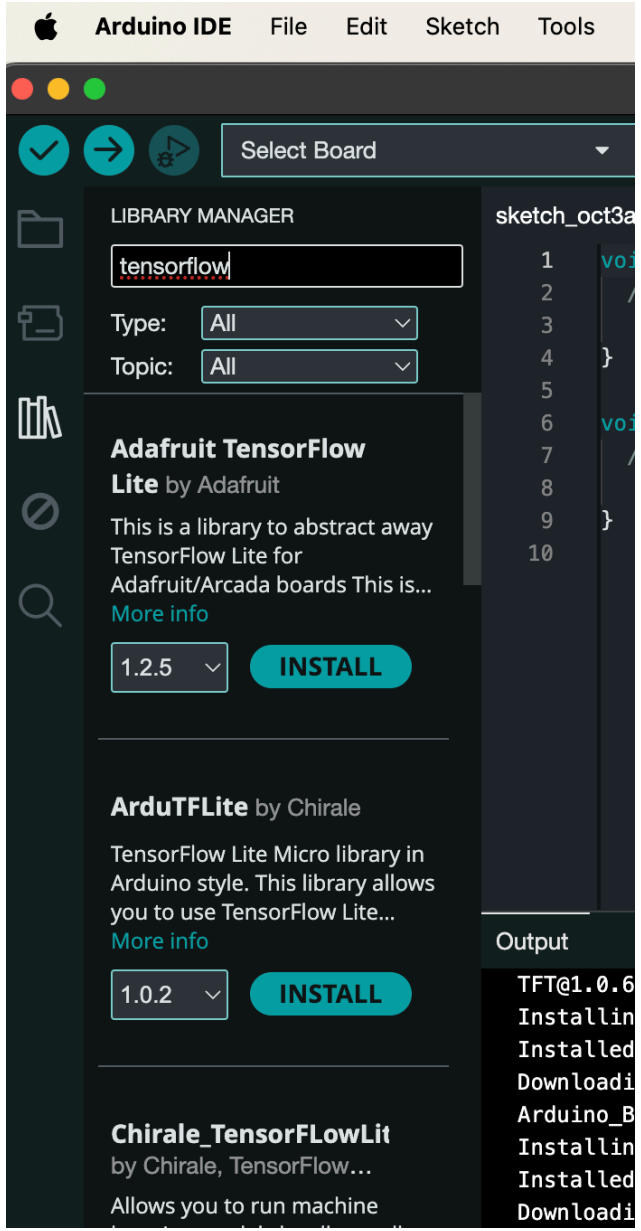


Deploying a model

Go to:
Sketch
Include Library
Manage Libraries



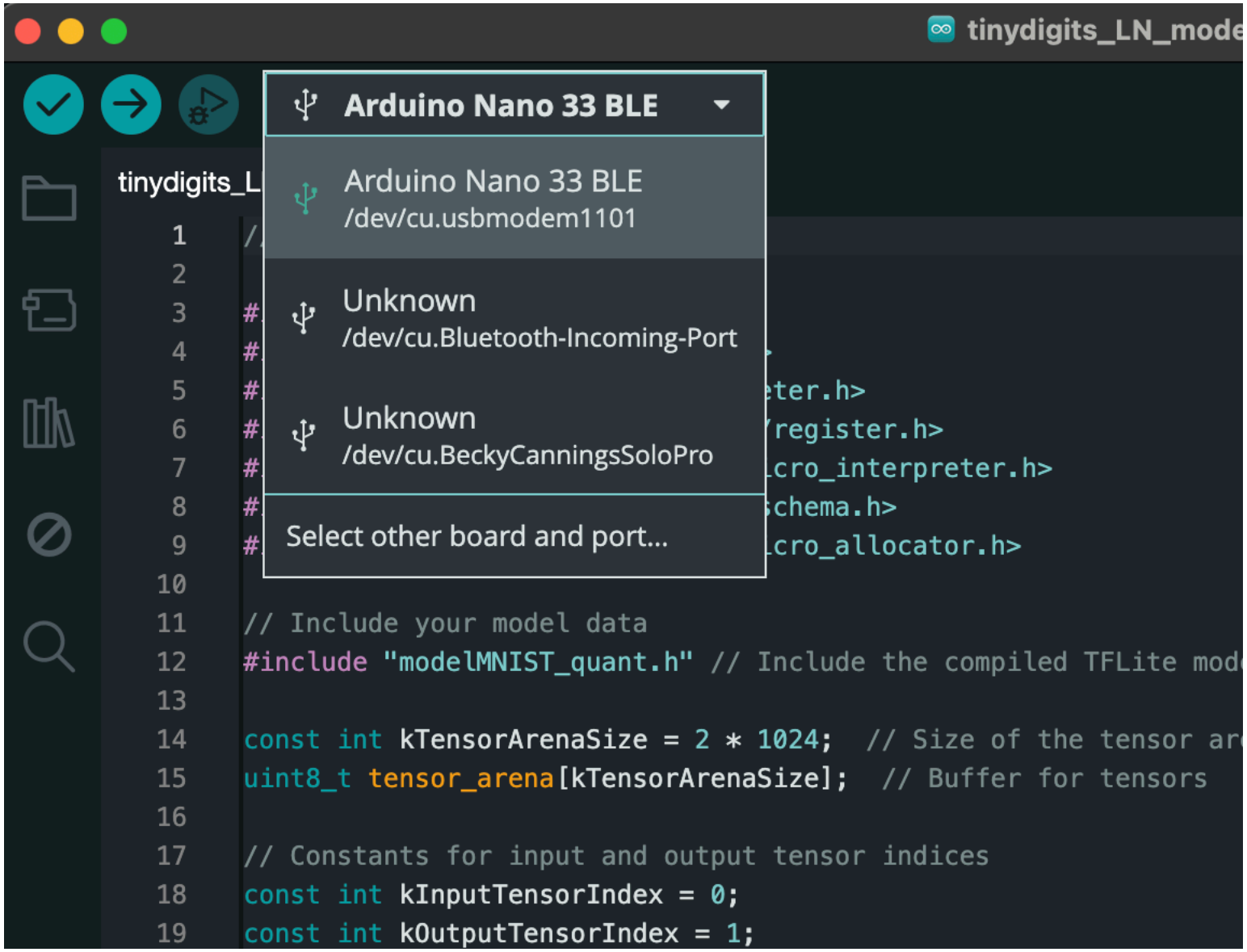
Search for tensorflow
Install



UNIVERSITY OF PORTSMOUTH

Deploying a model

Connect your Arduino
It will detect the board
Click on the board and the install automatically - **not manual install**



UNIVERSITY OF PORTSMOUTH

Deploying a model

As a test deploy we will try to deploy someone else's model - lets load the Harvard TinyML package…
File -> Examples -> hello world

Woop - now let's get back to tensorflow and try to understand and change parameters in the model - we will cover model deployment together in a supervision