

UNIVERSITY OF
PORTSMOUTH

AI 101

Becky Canning

Institute of Cosmology and Gravitation



© Becky Canning

With huge thanks to:

- Ben, Coleman, Lorenza, Obinna, Xan

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
- For space:
 - TinyML models

- Clustering
- Classification

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>

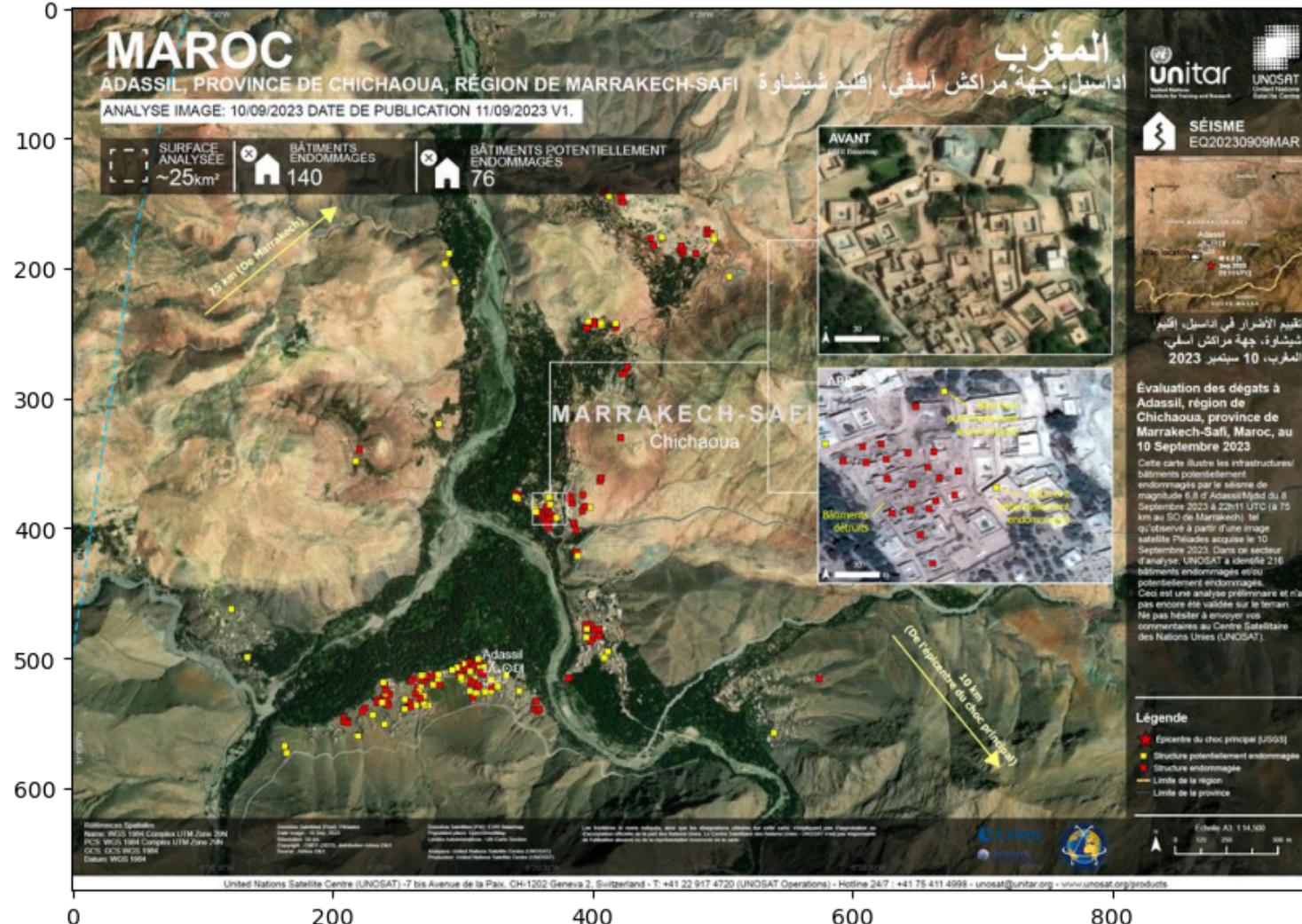
Register - the follow directions for 'Registering OAuth client' on the same page add this to you dashboard

The screenshot shows a Mac desktop with a Safari browser window open to the ESA documentation for authentication. The main page has a blue header with the word 'Documentation'. Below it, there's a search bar and a sidebar with navigation links like 'Welcome', 'Roadmap', 'User registration and authentication', 'Data', 'APIs', 'Access Token', 'OData', and 'Contributors'. The main content area is titled 'Authentication' and explains the use of OAuth2 for authentication, mentioning the need to register an OAuth Client in account settings. A 'How to use tokens' section follows, explaining the token validity period and how to reuse them. On the right side of the main page, there's a sidebar with links for 'On this page' such as 'How to use tokens', 'Registering OAuth client', 'OAuth2 Endpoints', and 'Token Request Examples'. A modal window titled 'Account info' is overlaid on the browser. It displays a 'Copernicus General user Account' that is valid until 29 September 2024, 15:12. It features a green checkmark icon. Below this, the 'OAuth clients' section is shown, with a button labeled '+ Create' highlighted with a blue box. A client named 'beckycanning' is listed, along with edit and delete icons.

- DL case study with thanks to Obinna

Starting from the end

- A DL model for Earth Observation data



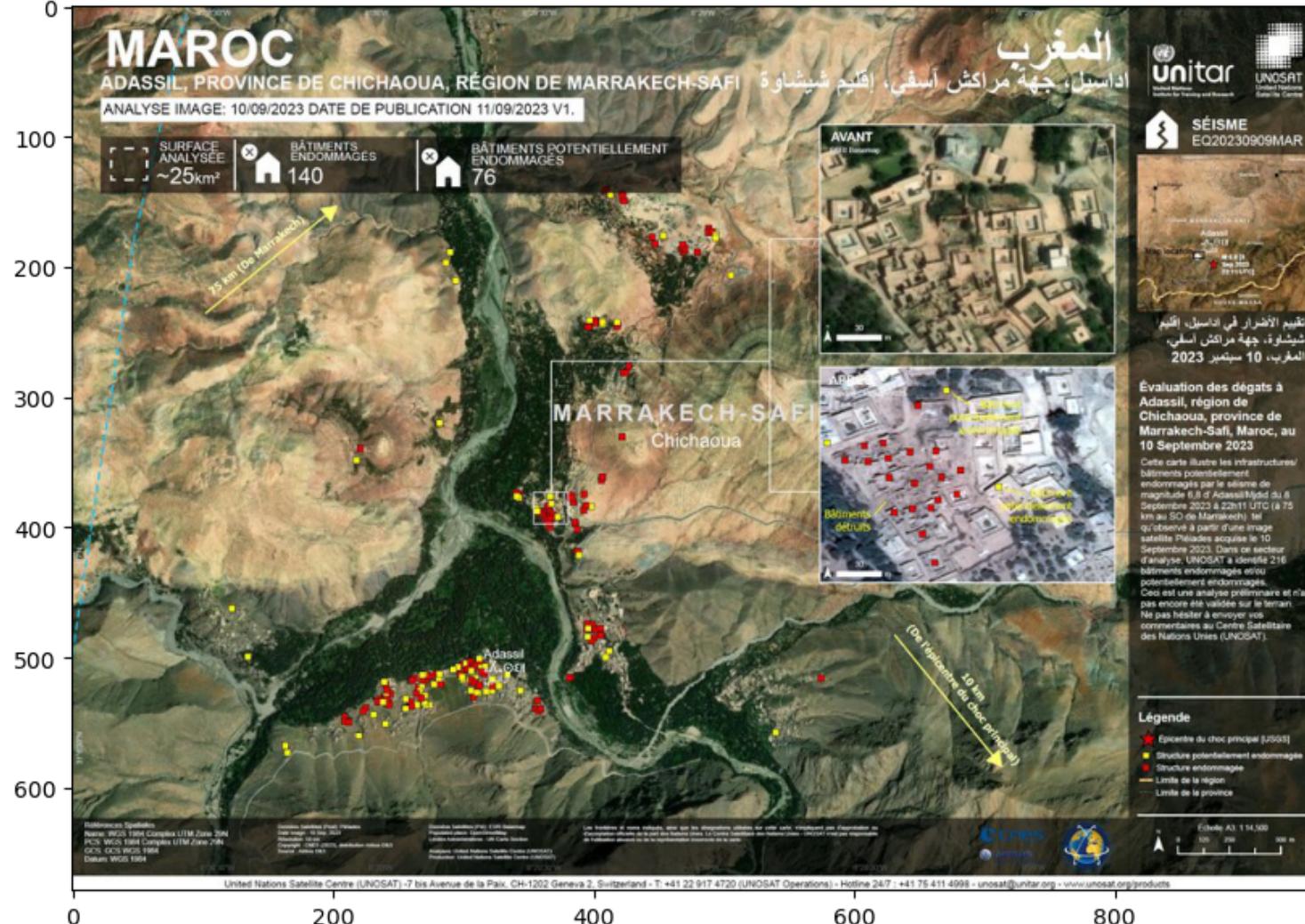
There was a 6.8 magnitude earthquake that struck the Atlas Mountains, about 75 km from Marrakech, Morocco on 8 September, 2023

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

- A DL model for Earth Observation data

- DL case study with thanks to Obinna



What are the steps we would take?

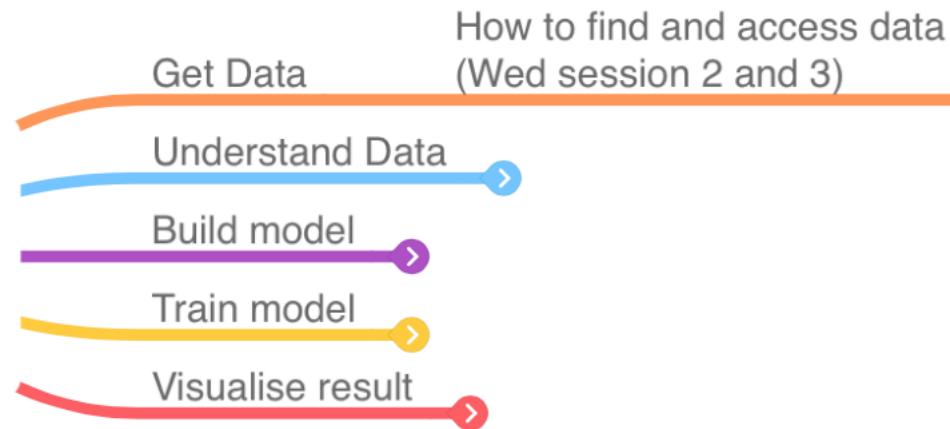
Starting from the end

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



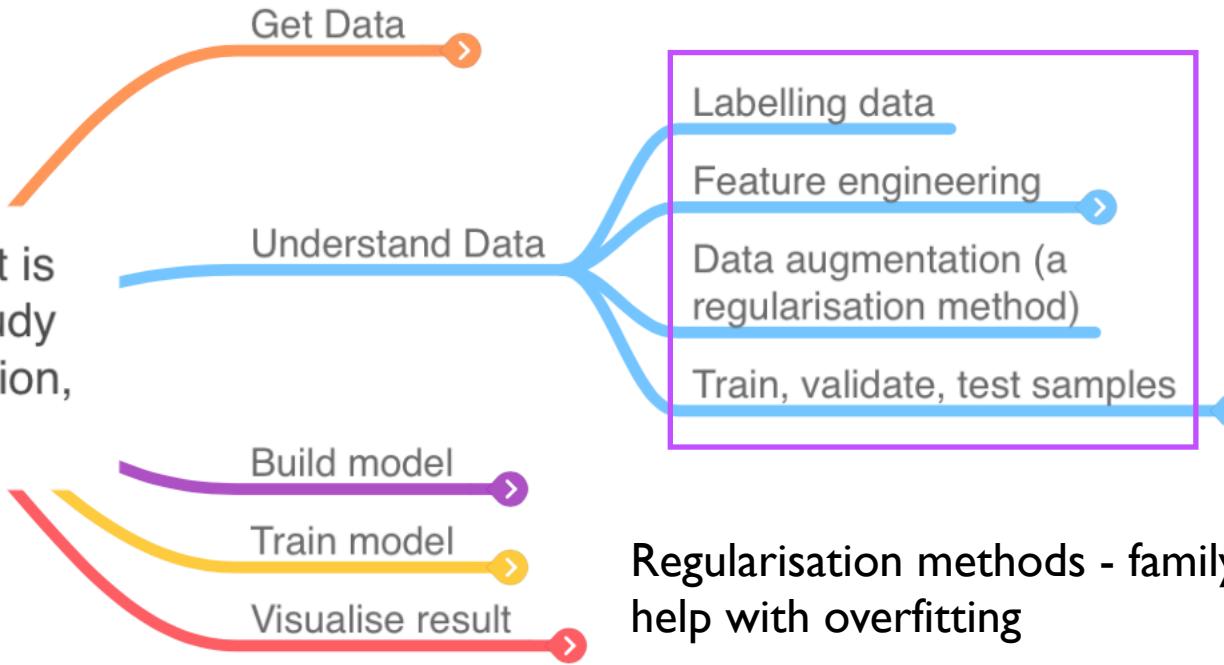
Starting from the end

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



Starting from the end

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

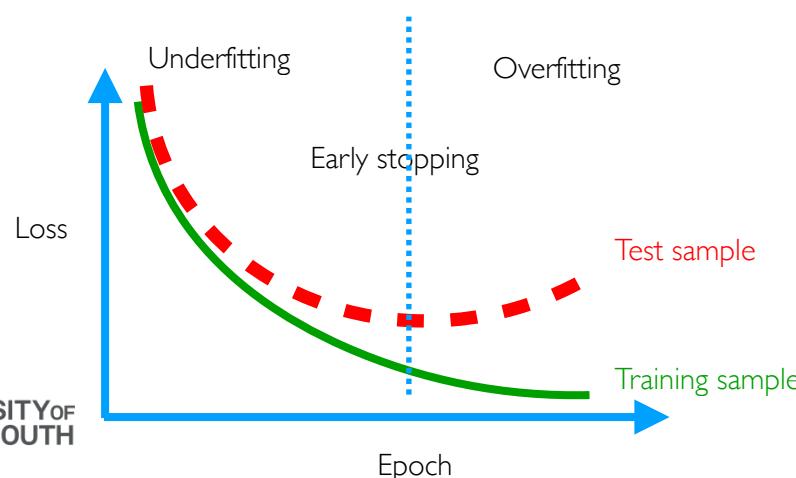


Regularisation methods - family of different methods which help with overfitting

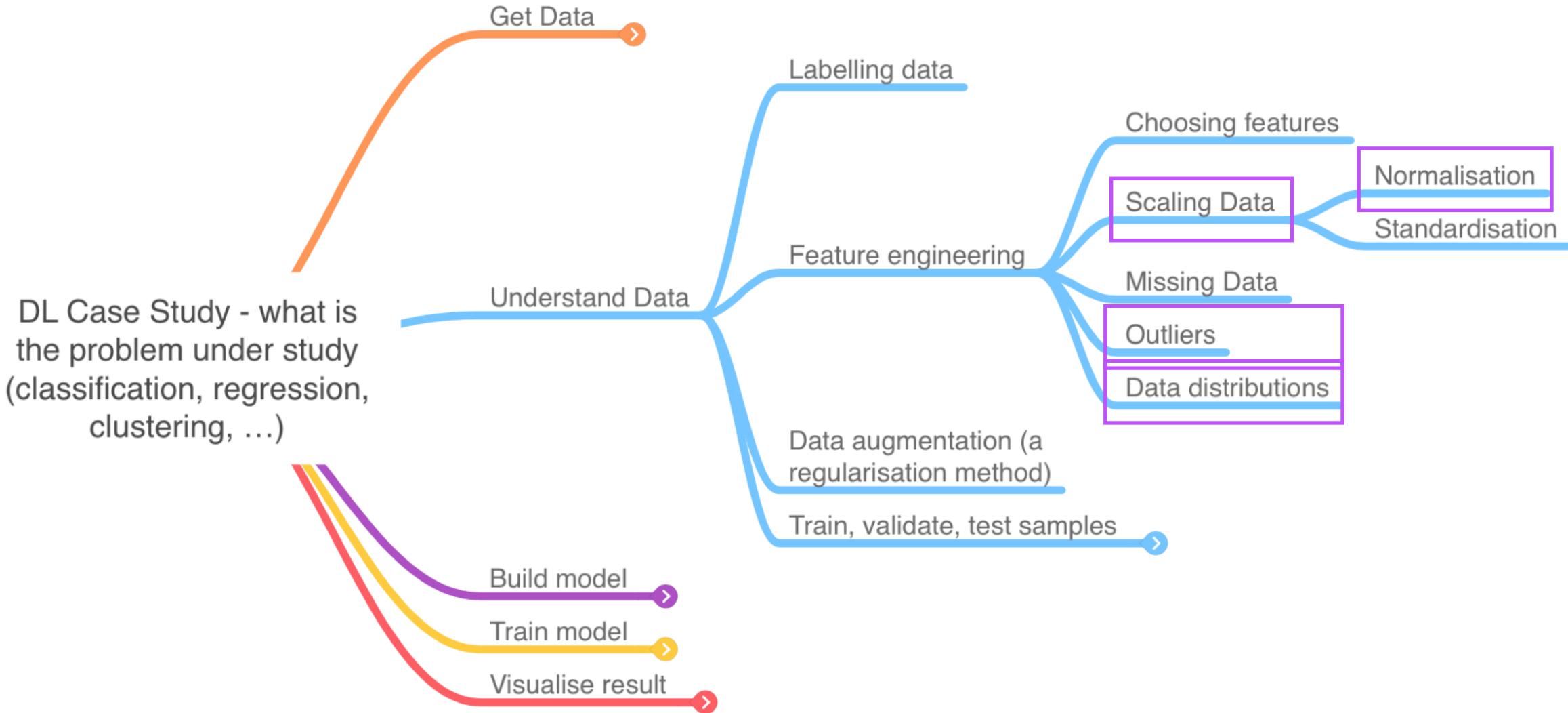
Trade off between bias and variance

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

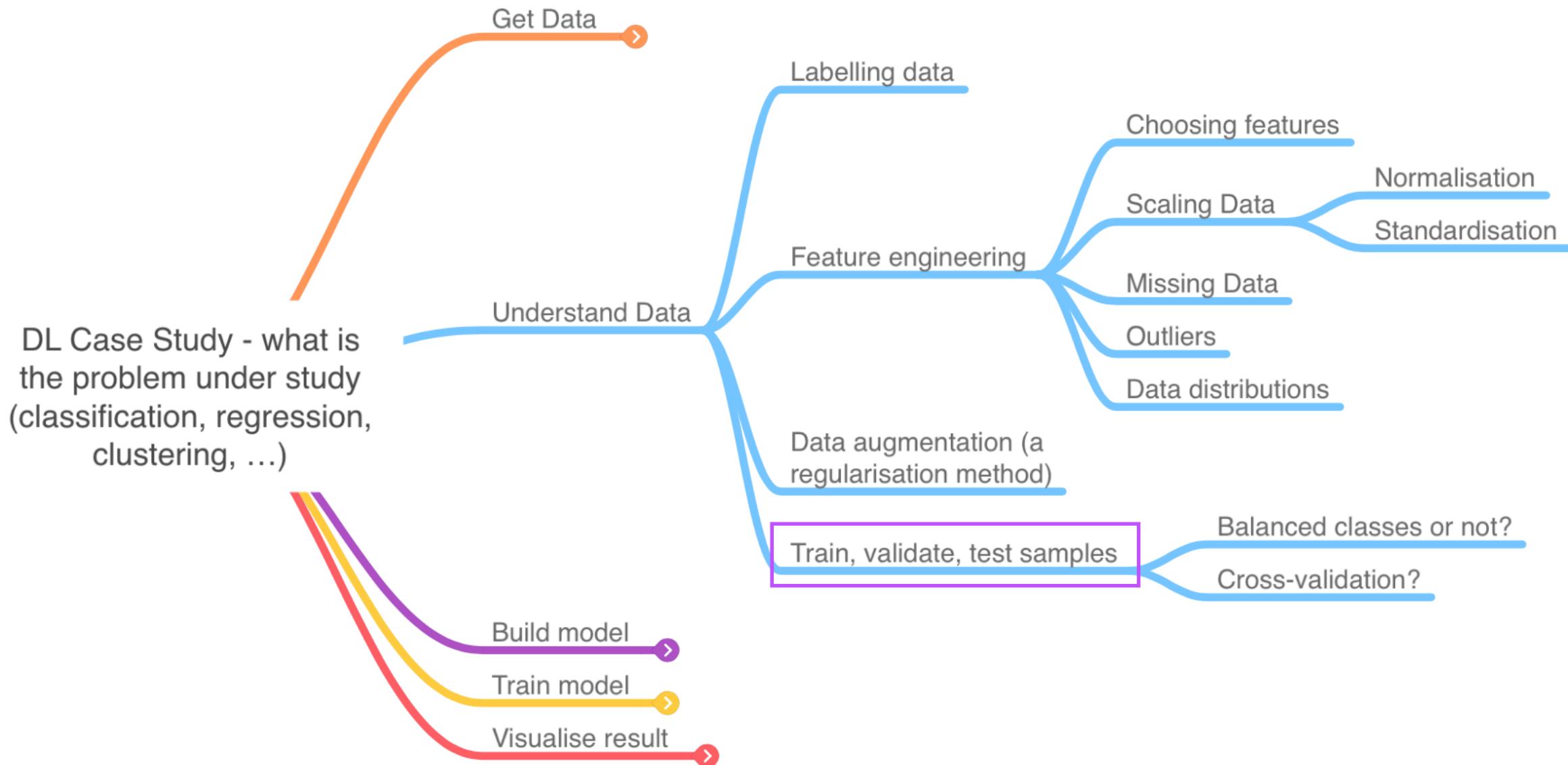
Variance: high variance means a lack of generalisation - the accuracy on a specific training or validation set may be high but the variance in accuracy across many sets is low



Starting from the end

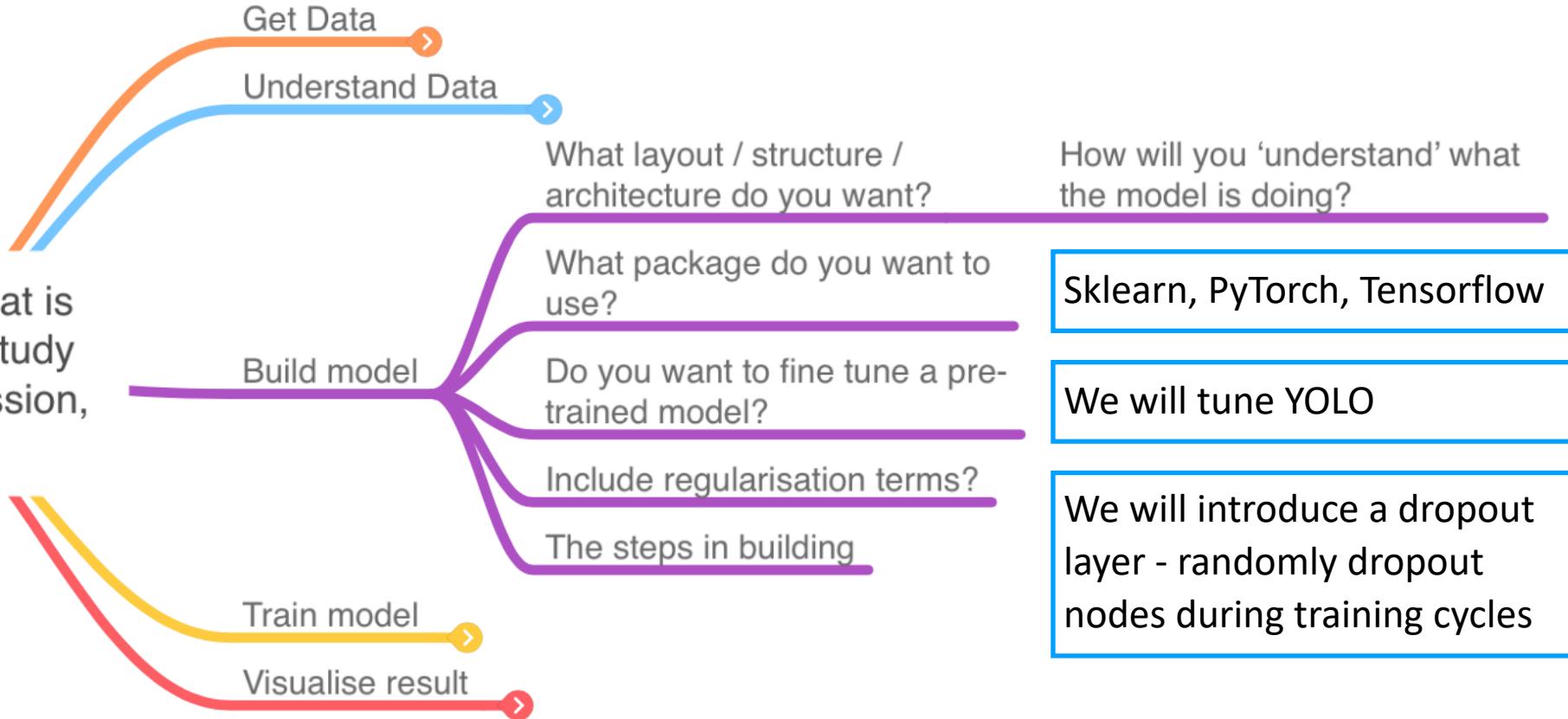


Starting from the end



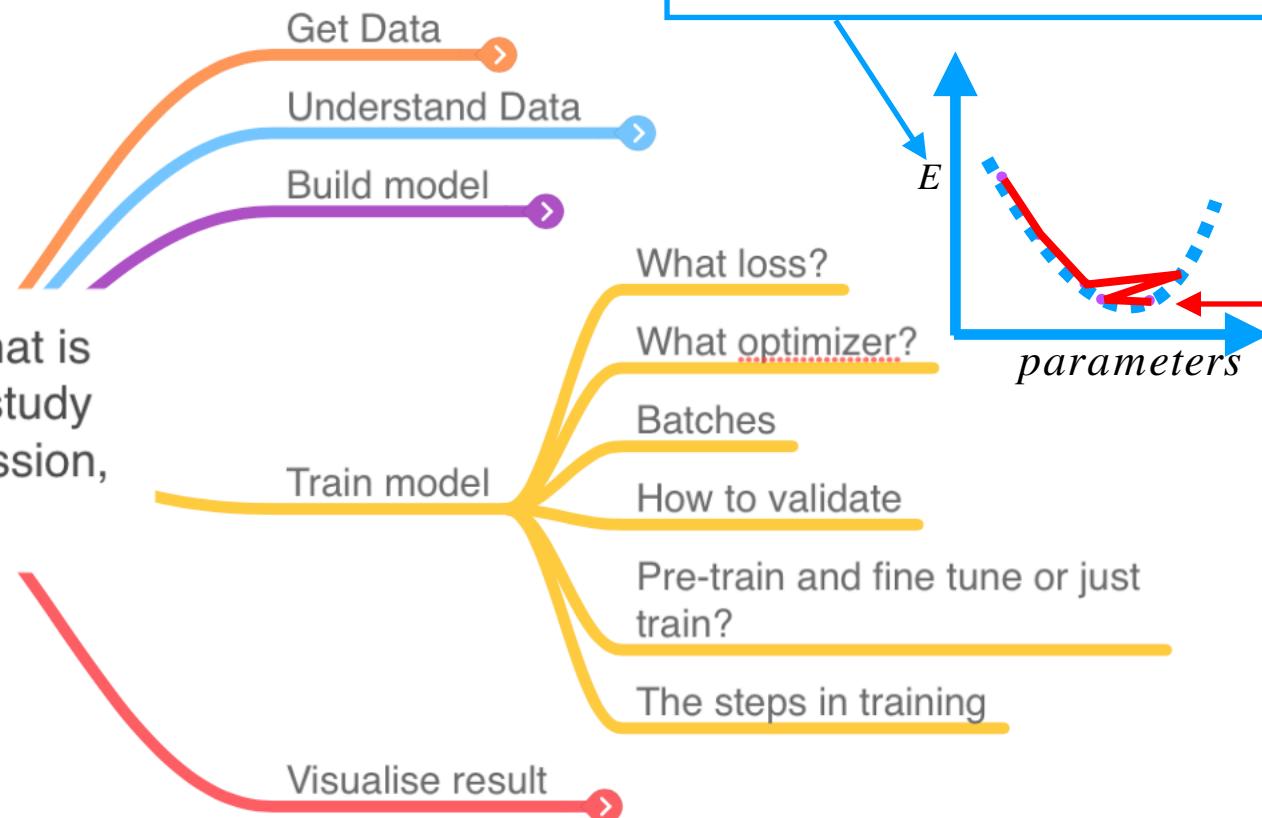
Starting from the end

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, ...)



Our loss functions

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

We will use mean squared error for regression.

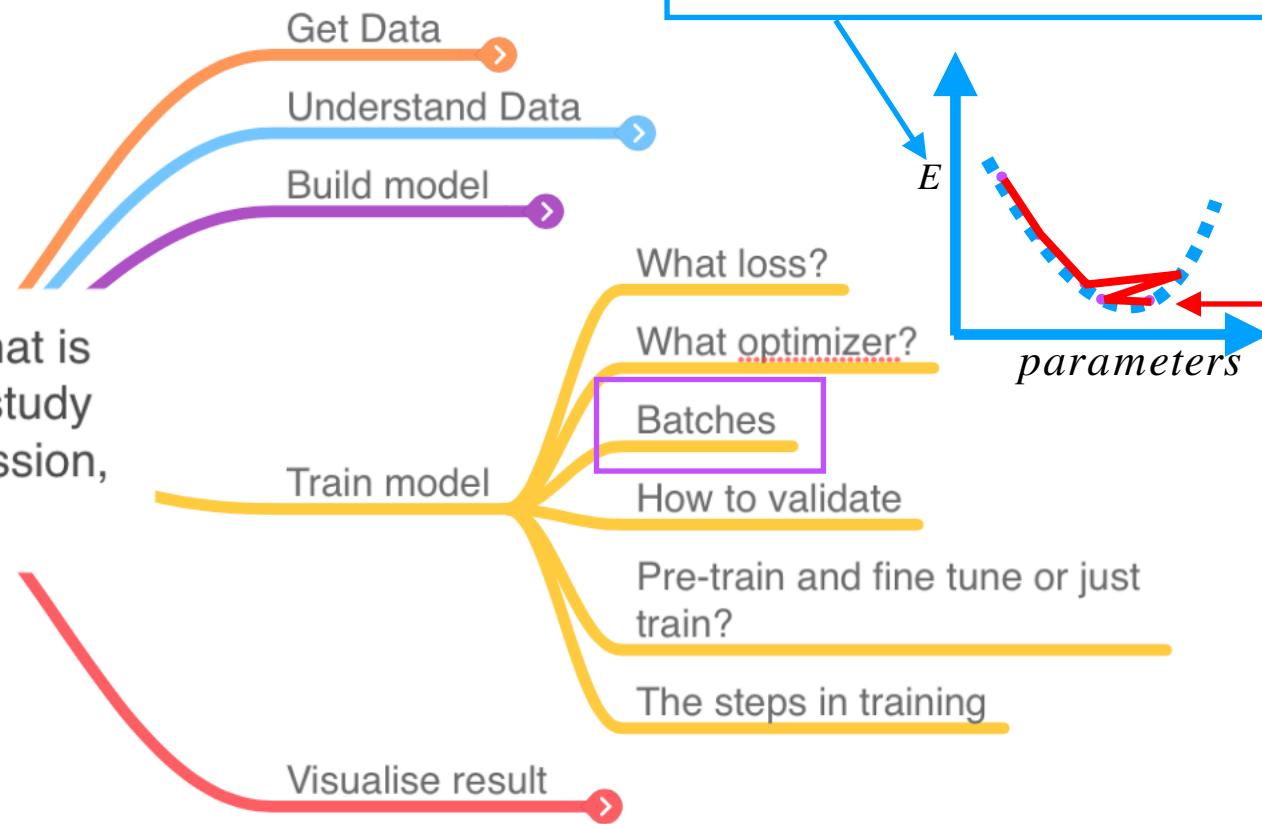
Optimisers tell us how to step about in our parameters space.

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

We will use Adam

Starting from the end

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



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.

Samples, batches, epochs

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

My input dataset

Samples, batches, epochs

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \boxed{\text{My input dataset}} \quad [x_1] \quad \boxed{\text{A data sample}}$$

Samples, batches, epochs

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

My input dataset

$$[x_1]$$

A data sample

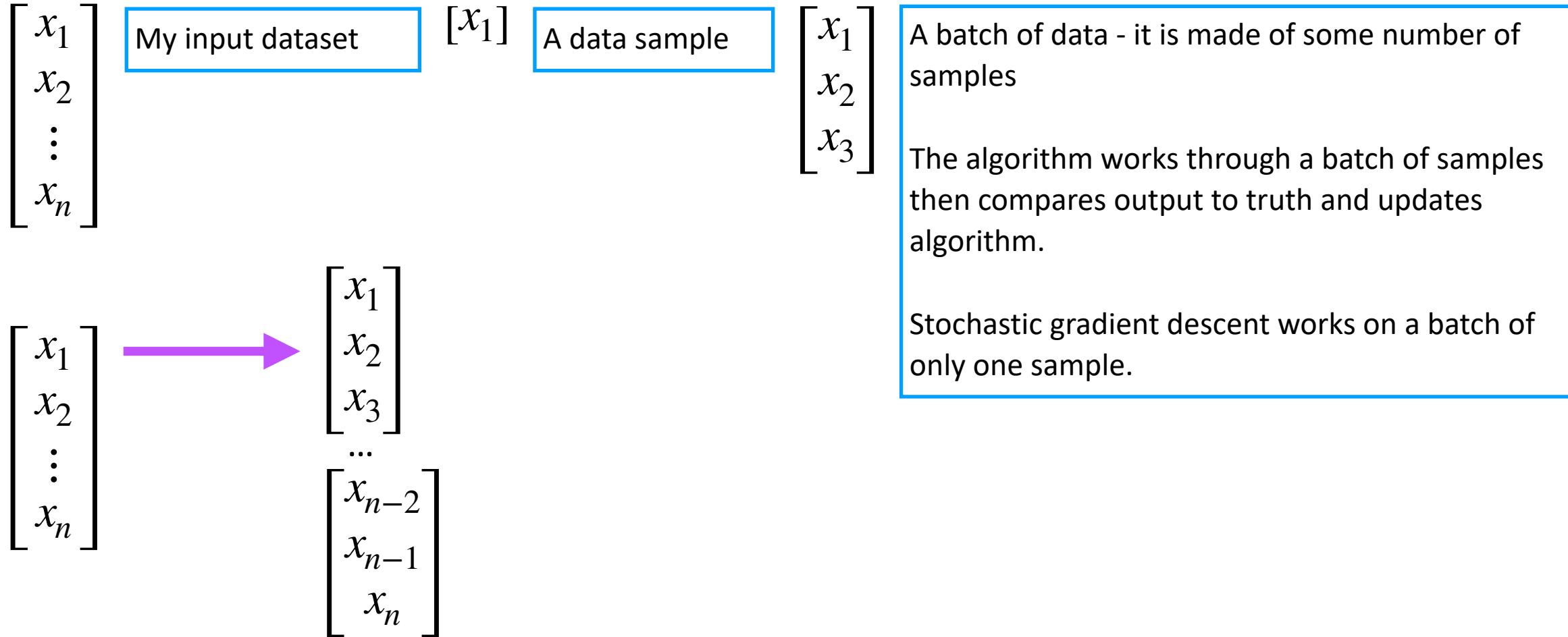
$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

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

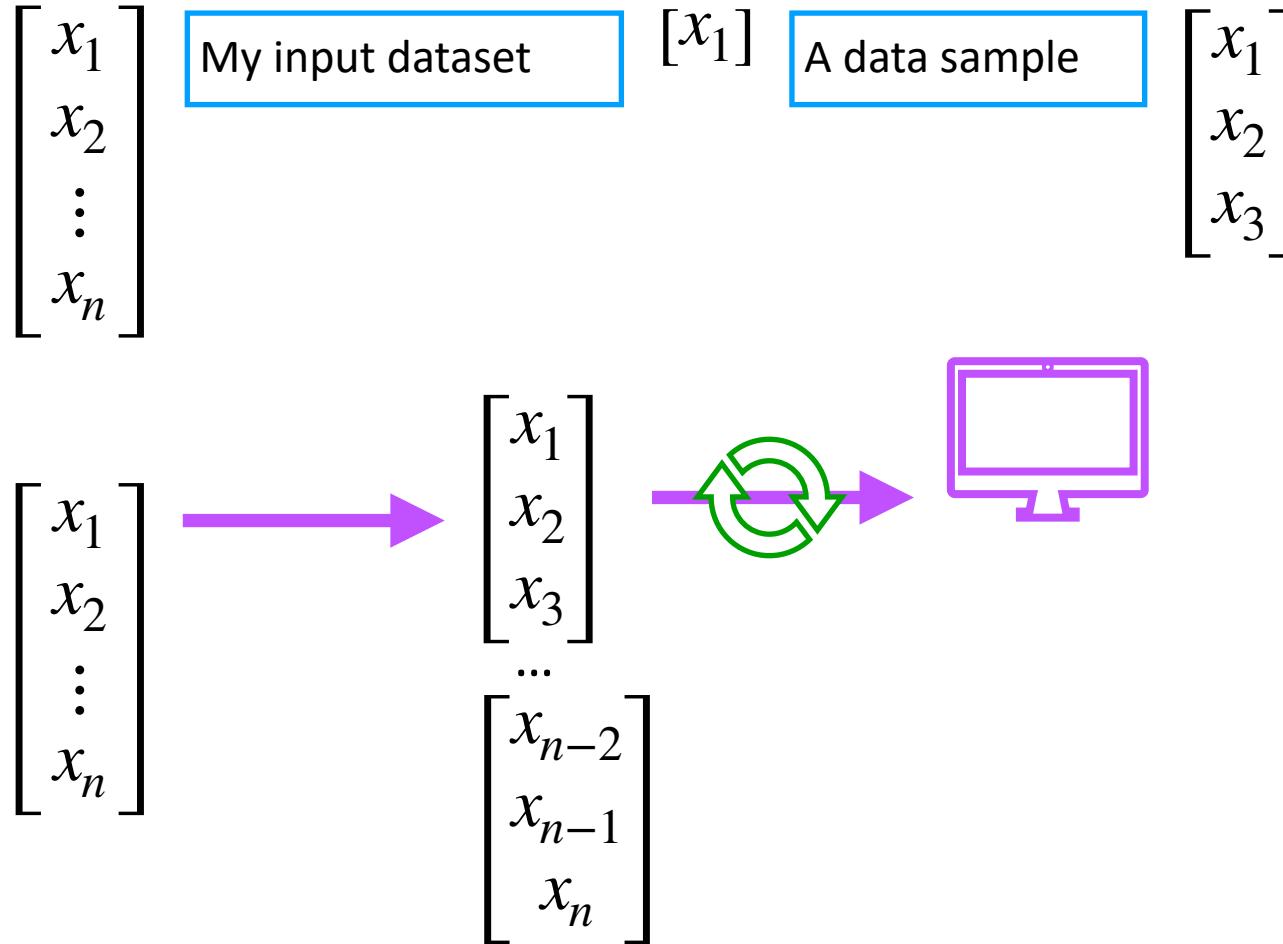
The algorithm works through a batch of samples then compares output to truth and updates algorithm.

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

Samples, batches, epochs



Samples, batches, epochs

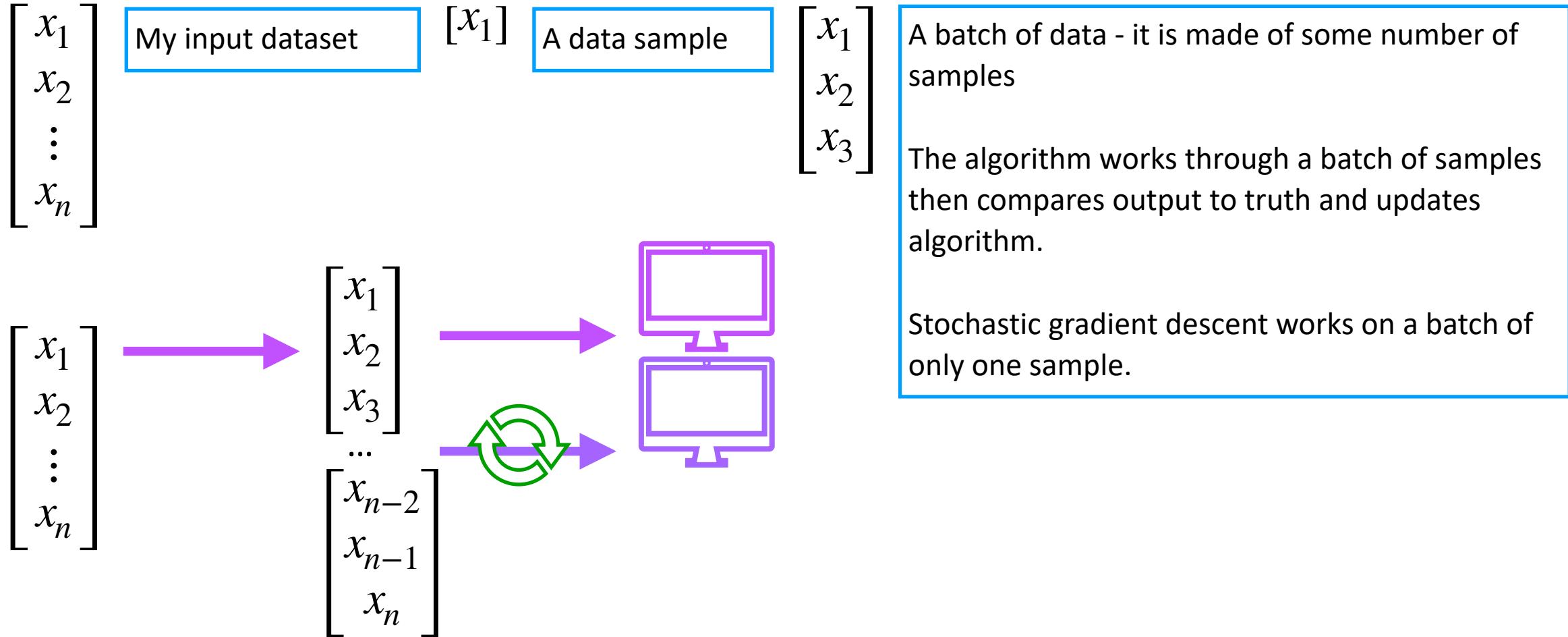


A batch of data - it is made of some number of samples

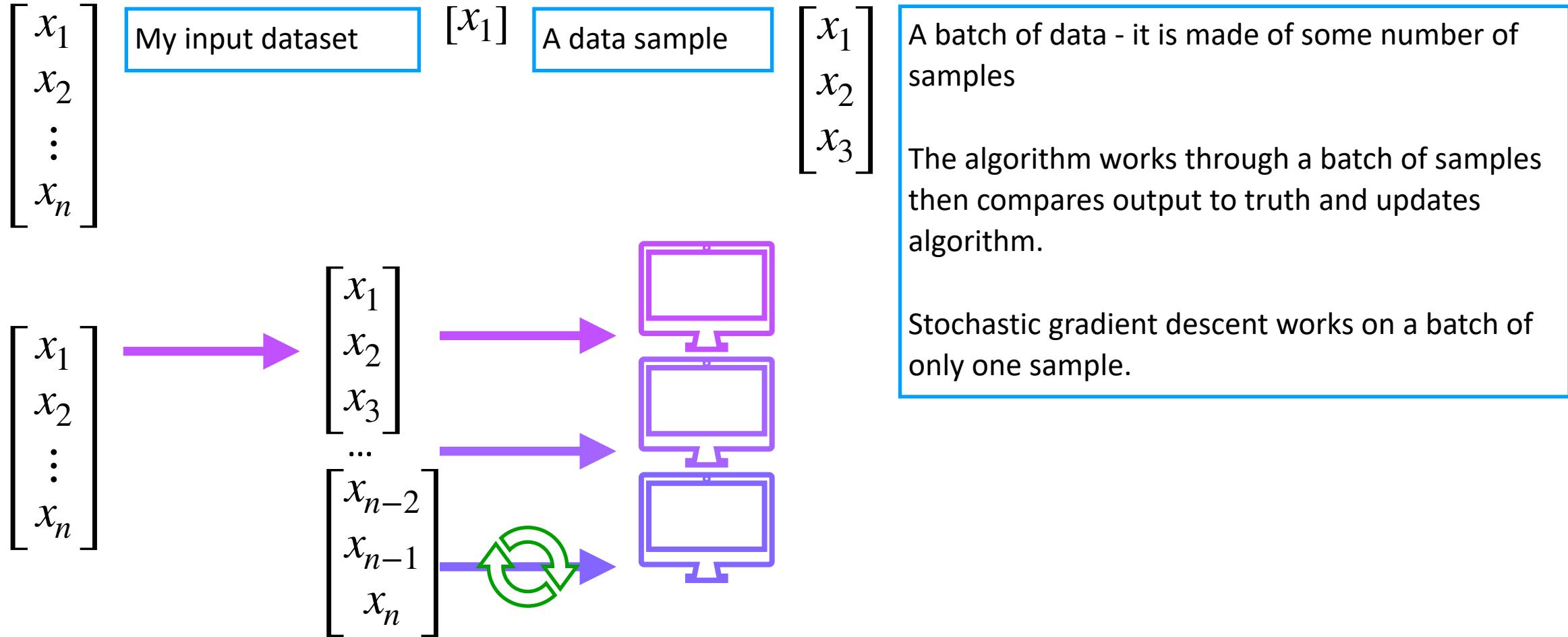
The algorithm works through a batch of samples then compares output to truth and updates algorithm.

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

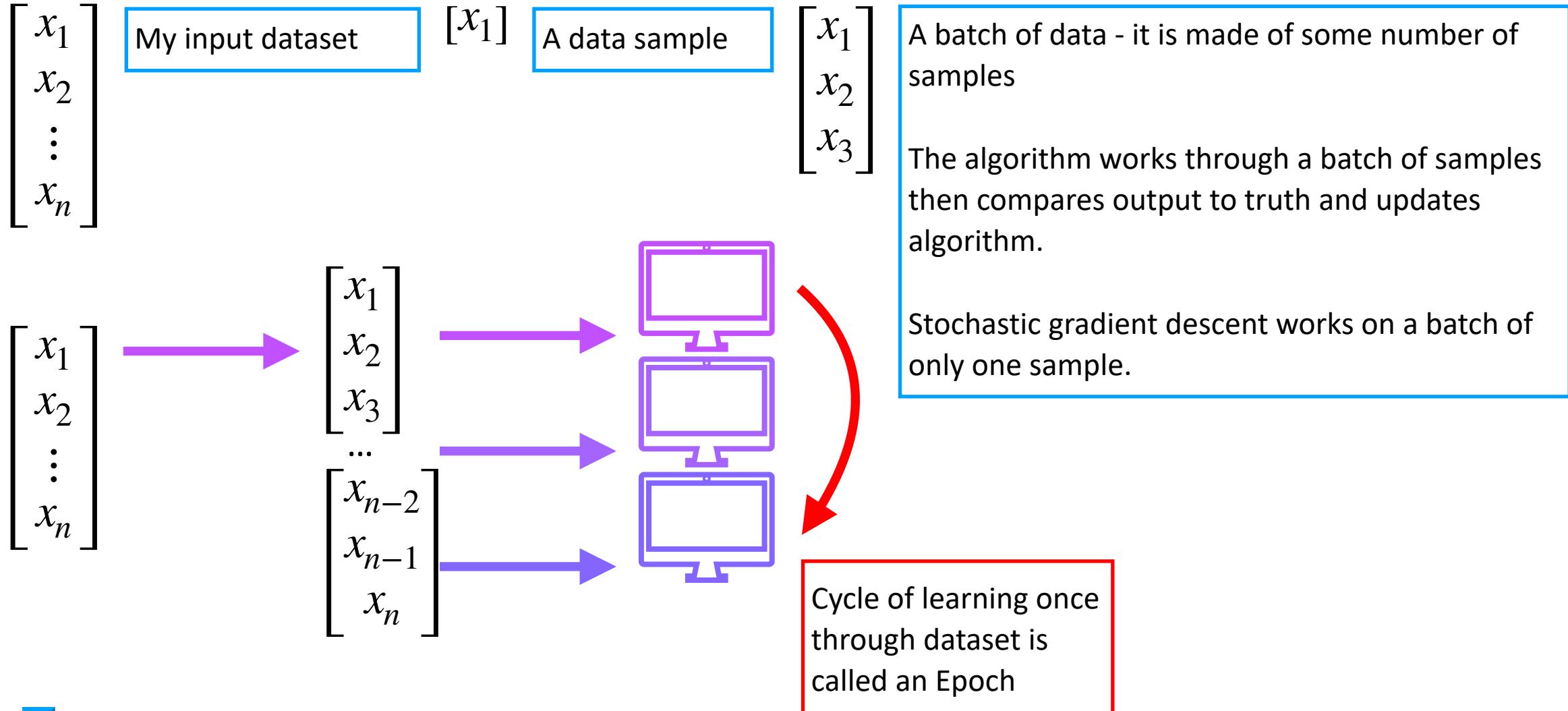
Samples, batches, epochs



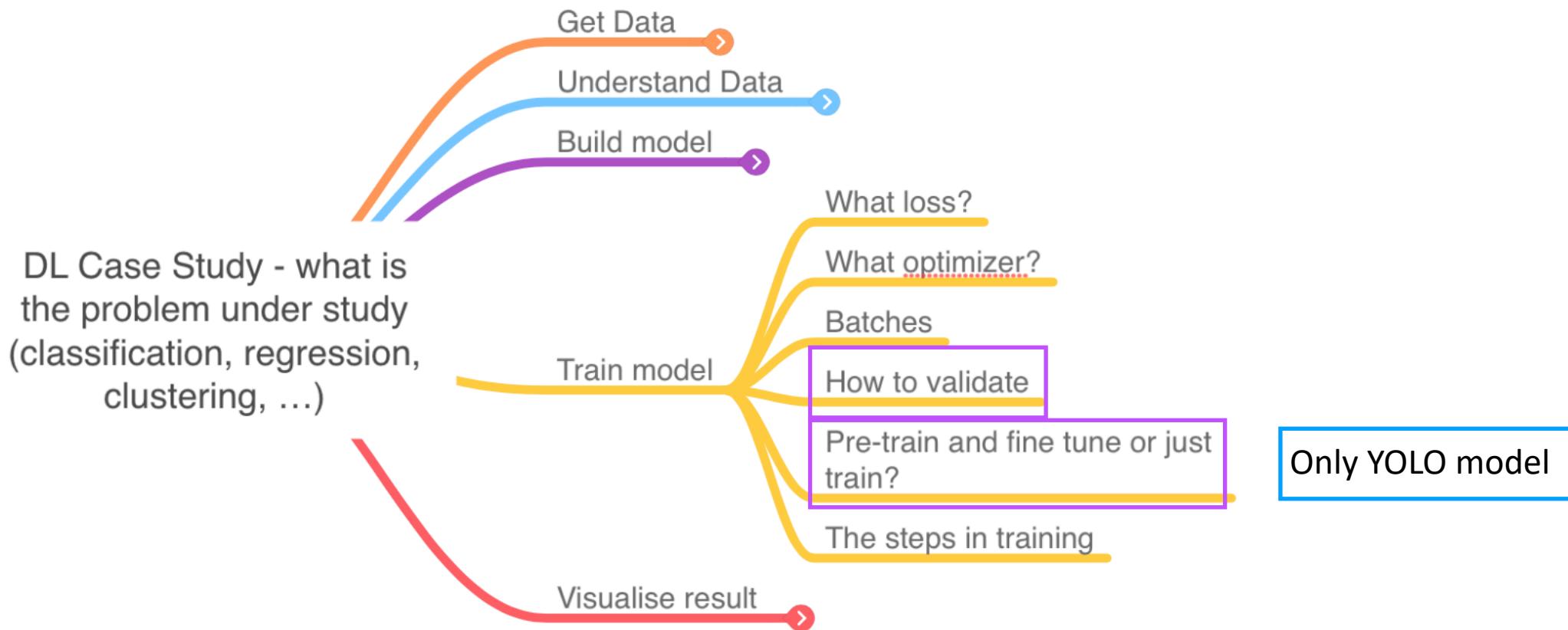
Samples, batches, epochs



Samples, batches, epochs

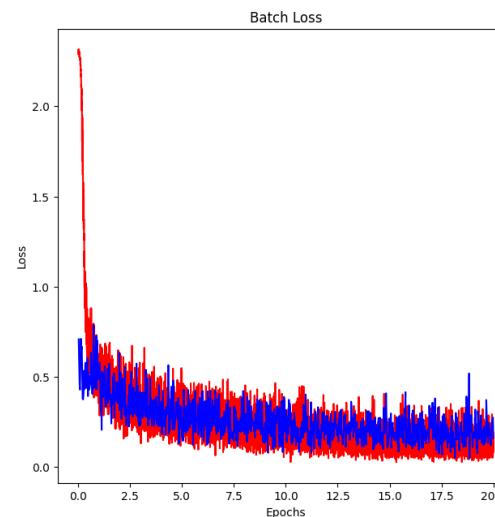
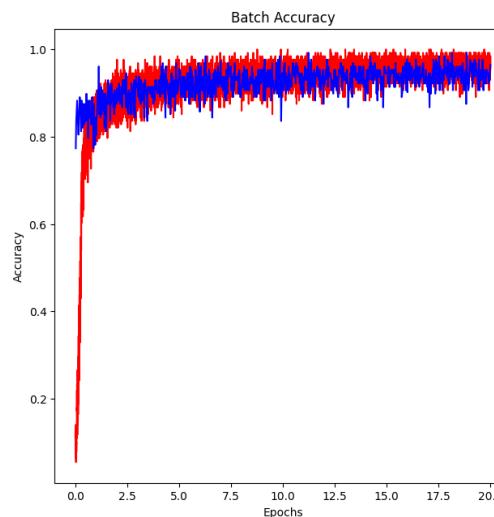
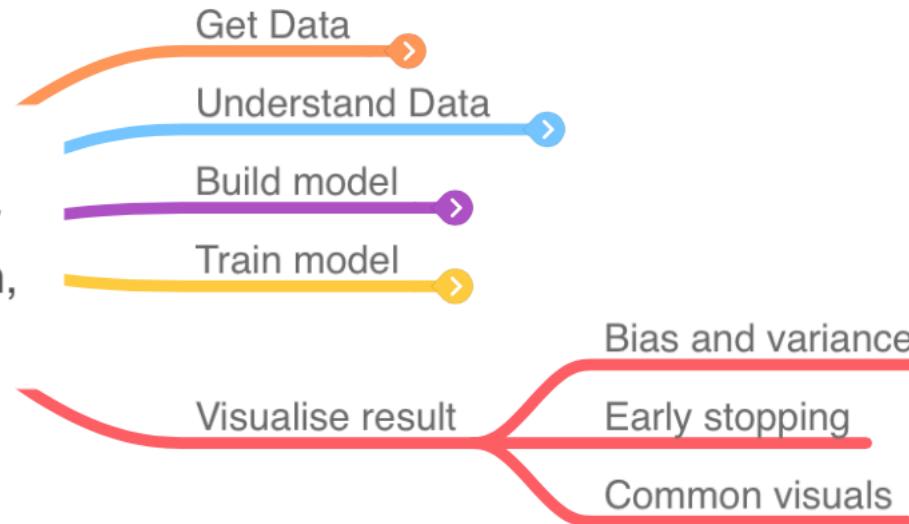


Starting from the end

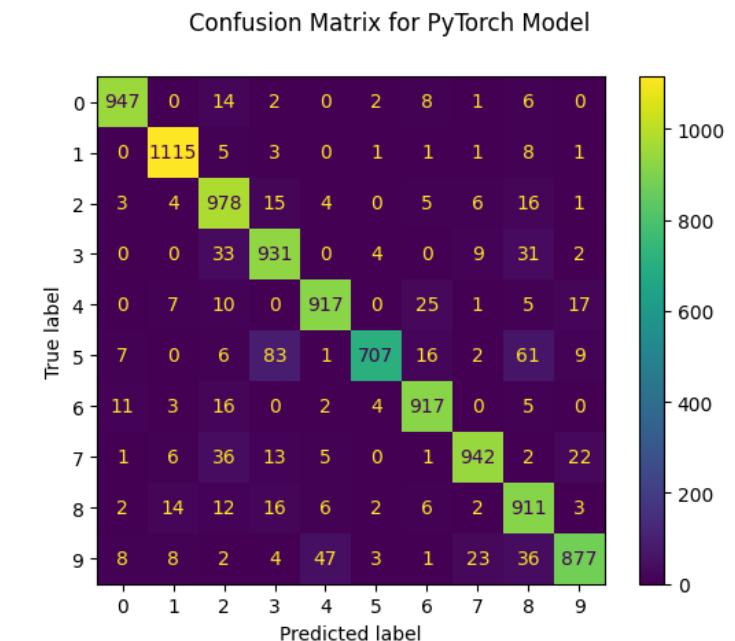


Starting from the end

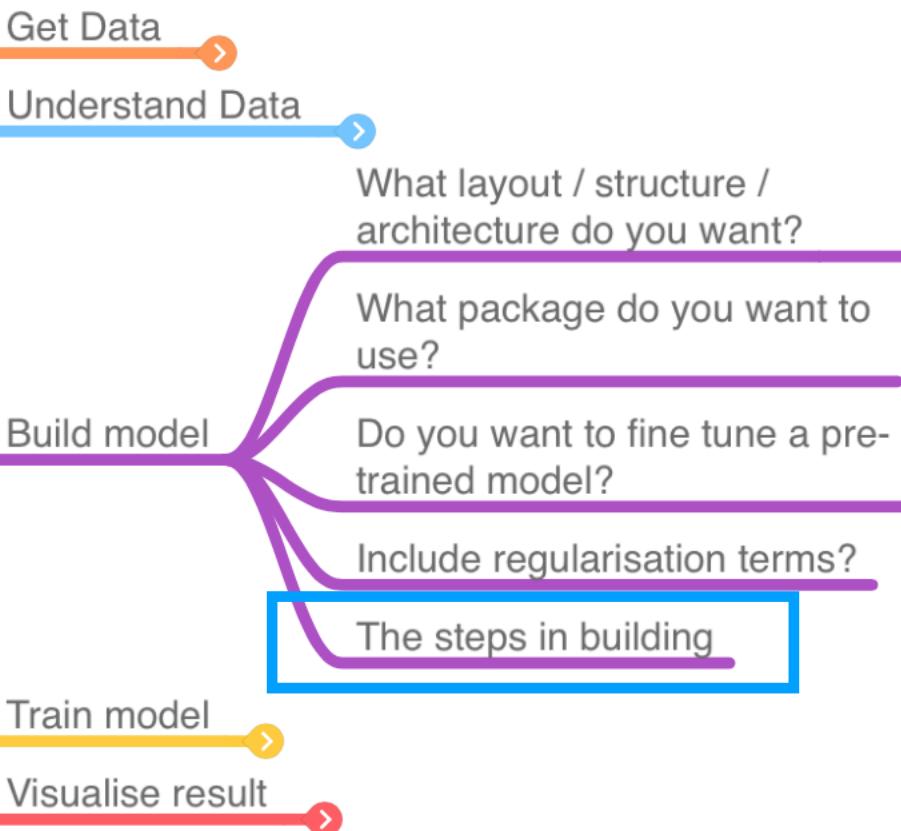
DL Case Study - what is the problem under study
(classification, regression, clustering, ...)



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



The steps in building



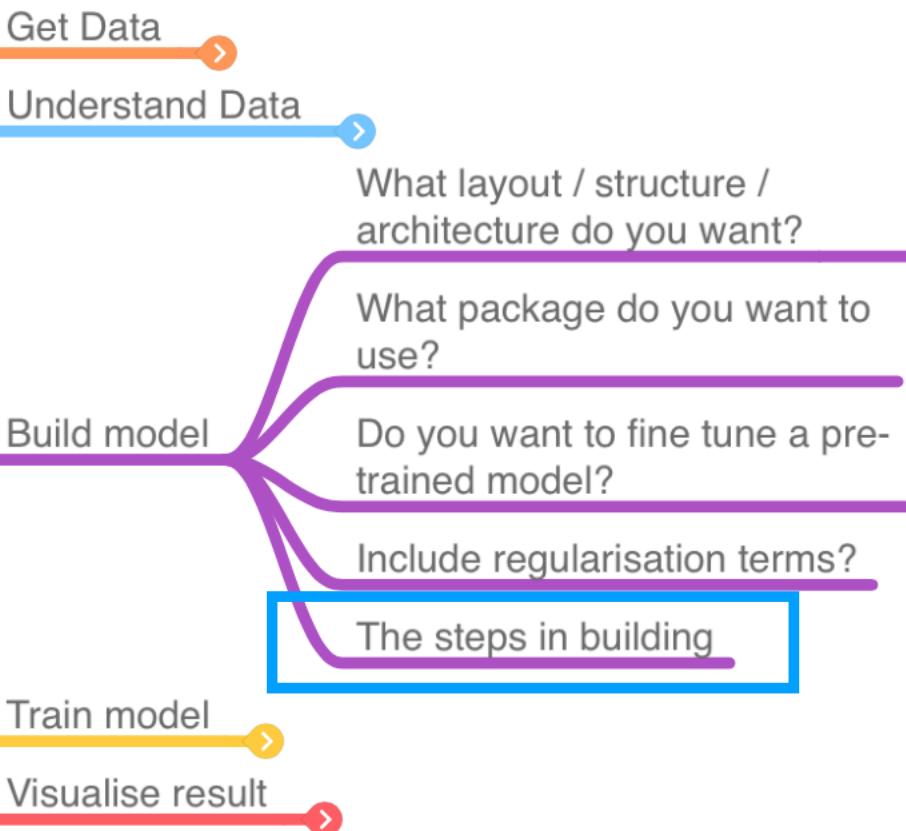
Objects - things with properties and methods

PyTorch

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

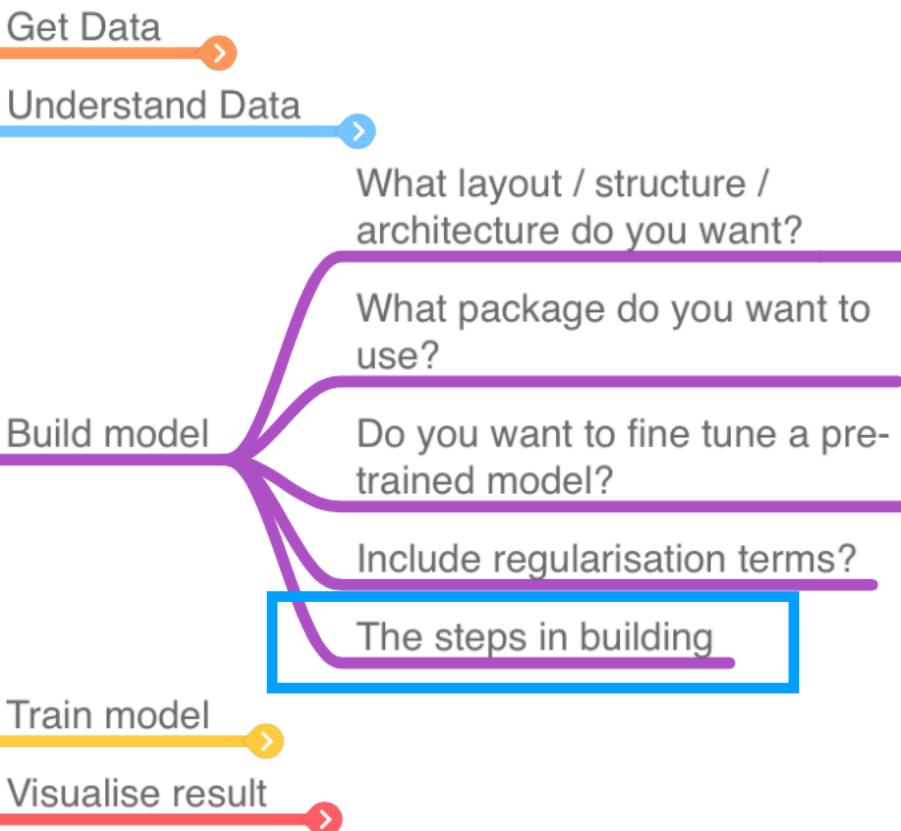


PyTorch

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



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

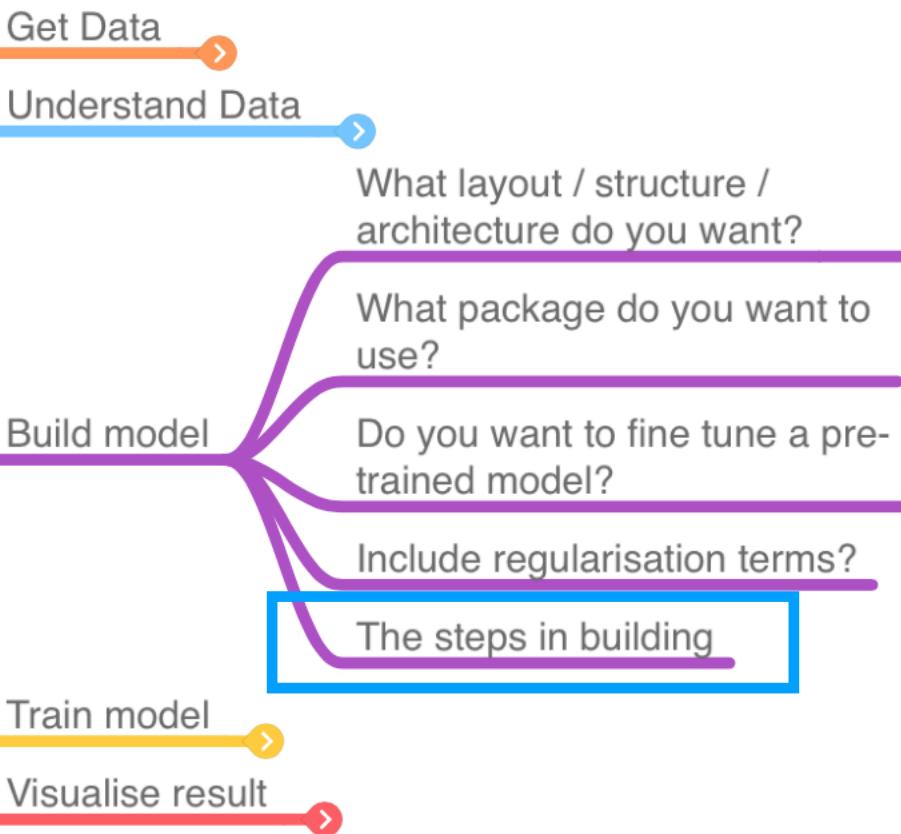
PyTorch

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

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

The steps in building



```
import torch.functional as F

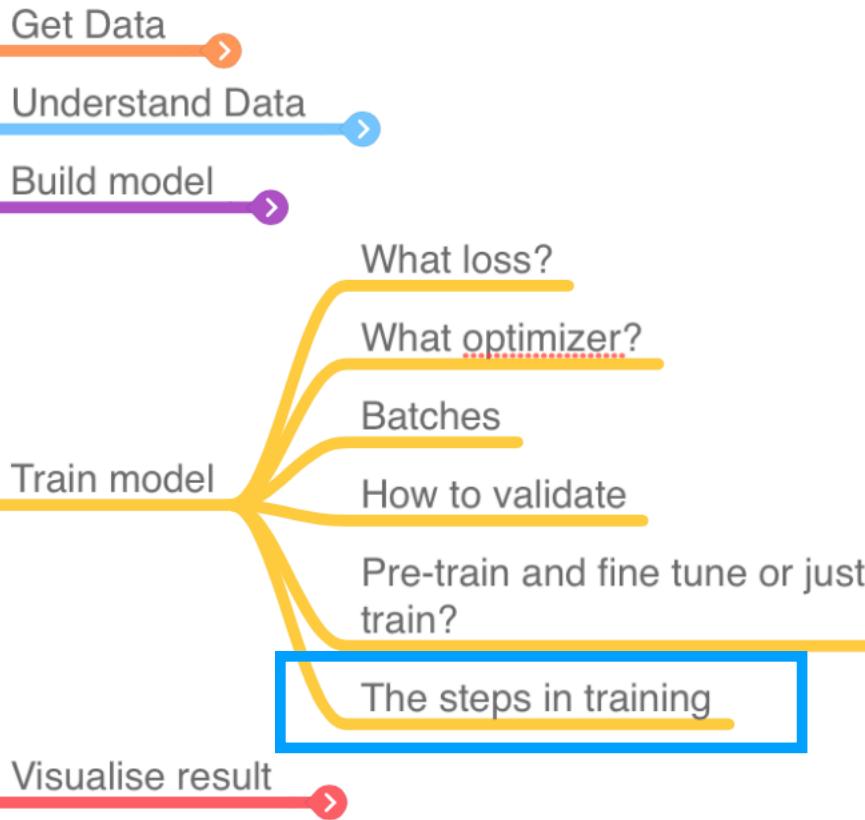
class LeNet(torch.nn.Module):

    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
```

The steps in training

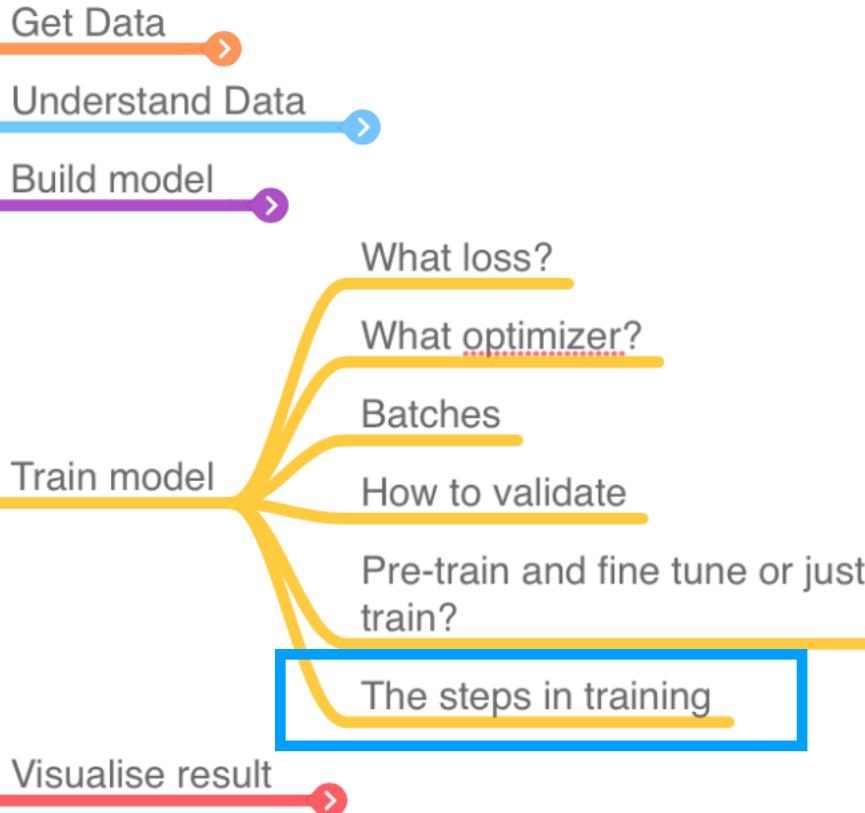


- Put model into 'train' mode: `model.train()`
- For each 'batch':
 - Reset gradient to zero - incase not
 - Predict outputs from inputs
 - Calculate loss
 - Compute gradients using backpropagation
 - Update parameters
 - Compute accuracy and loss

Really good doc to read:

<https://pytorch.org/tutorials/beginner/introyt/trainingyt.html>

The steps in training



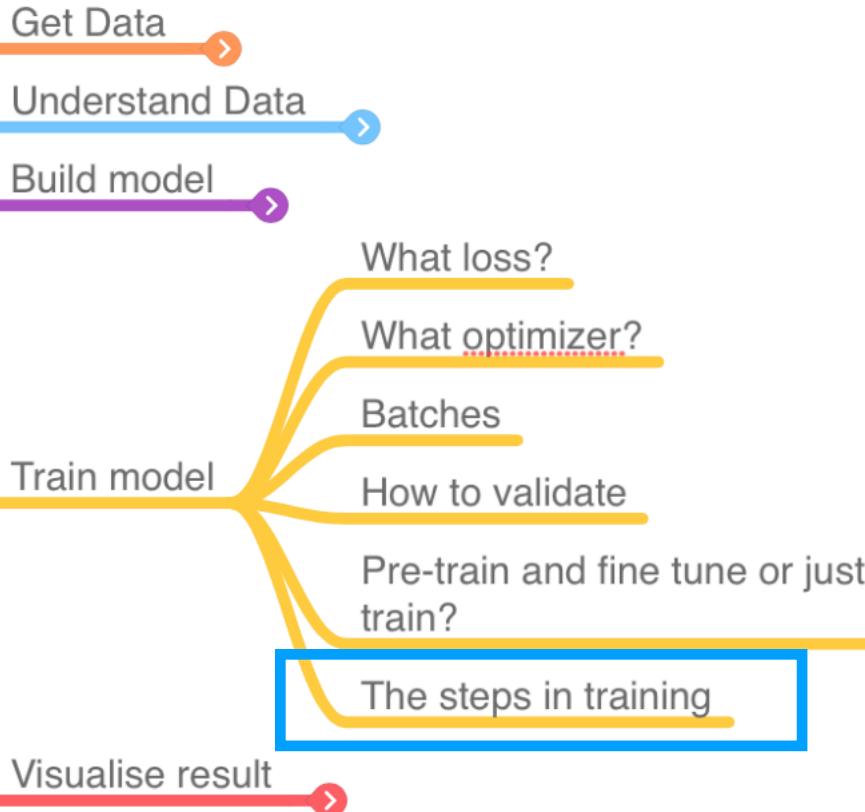
- Put model into 'train' mode: `model.train()`
- For each 'batch':
 - Reset gradient to zero - incase not
 - Predict outputs from inputs
 - Calculate loss
 - Compute gradients using backpropagation
 - Update parameters
 - Compute accuracy and loss

```
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 gradient 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



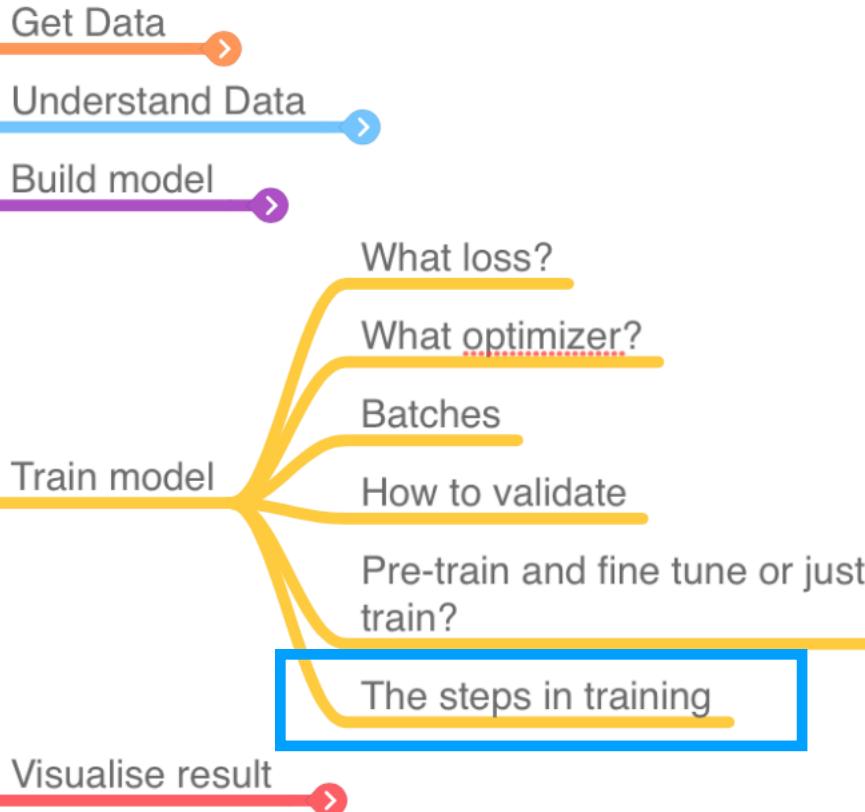
- Put model into 'train' mode: `model.train()`
- For each 'batch':
 - Reset gradient to zero - incase not
 - Predict outputs from inputs
 - Calculate loss
 - Compute gradients using backpropagation
 - Update parameters
 - Compute accuracy and loss

```
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 gradient 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 gradient 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:

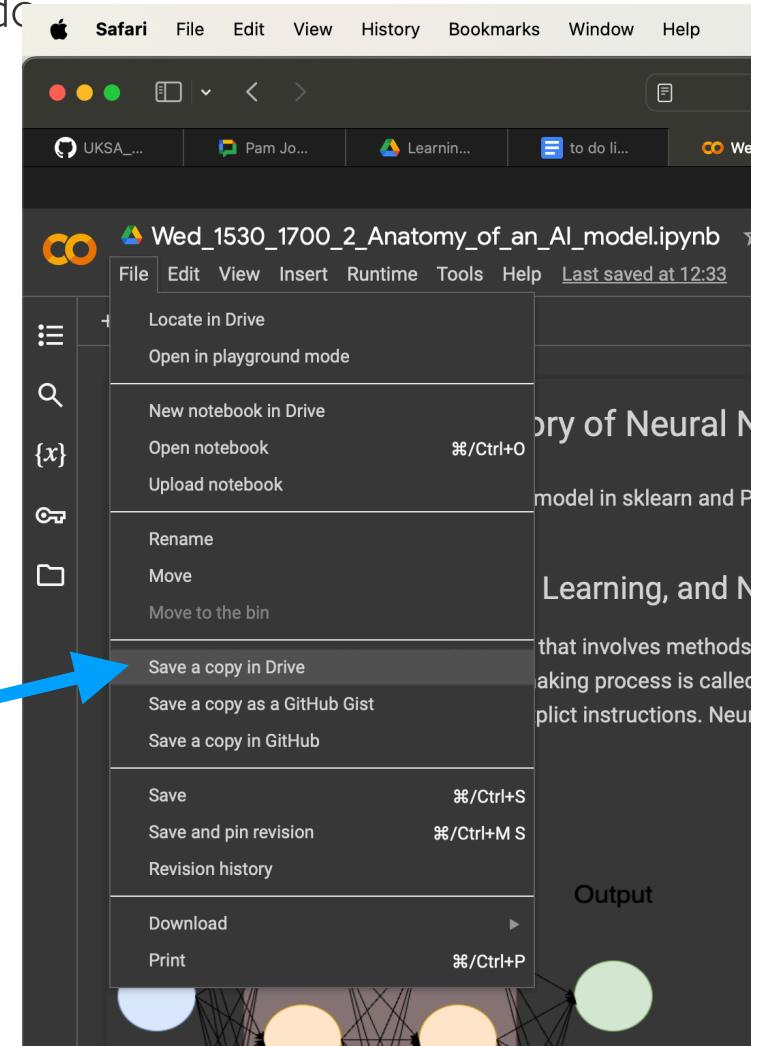
<https://pytorch.org/tutorials/beginner/introyt/trainingyt.html>

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

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_TUjEcADtDSifb_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

The steps in building

PyTorch

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

```
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 gradient 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
```

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)
```

Deploying a model

Get the Arduino IDE

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

Follow install and open IDE

Downloads



Arduino IDE 2.3.3

The new major release of the Arduino IDE is faster and even more powerful! In addition to a more modern editor and a more responsive interface it features autocompletion, code navigation, and even a live debugger.

For more details, please refer to the [Arduino IDE 2.0 documentation](#).

Nightly builds with the latest bugfixes are available through the section below.

SOURCE CODE

The Arduino IDE 2.0 is open source and its source code is hosted on [GitHub](#).

DOWNLOAD OPTIONS

Windows Win 10 and newer, 64 bits

Windows MSI installer

Windows ZIP file

Linux AppImage 64 bits (X86-64)

Linux ZIP file 64 bits (X86-64)

macOS Intel, 10.15: "Catalina" or newer, 64 bits

macOS Apple Silicon, 11: "Big Sur" or newer, 64 bits

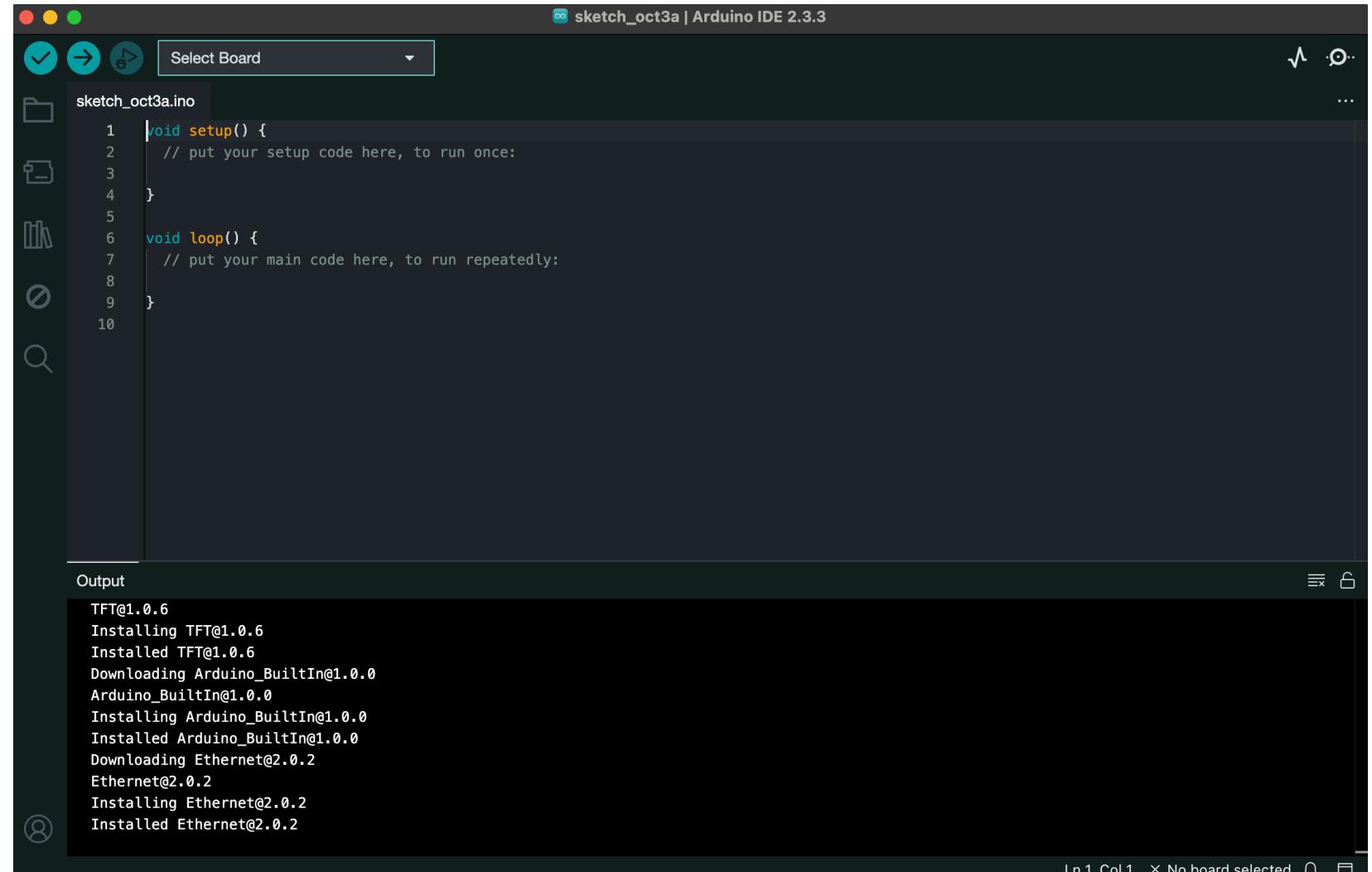
[Release Notes](#)

Deploying a model

Get the Arduino IDE

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

Follow install and open IDE



The screenshot shows the Arduino IDE interface with a dark theme. The top bar displays the title "sketch_oct3a | Arduino IDE 2.3.3". The main area shows a sketch named "sketch_oct3a.ino" containing the following code:

```
1 void setup() {
2     // put your setup code here, to run once:
3
4 }
5
6 void loop() {
7     // put your main code here, to run repeatedly:
8
9 }
10
```

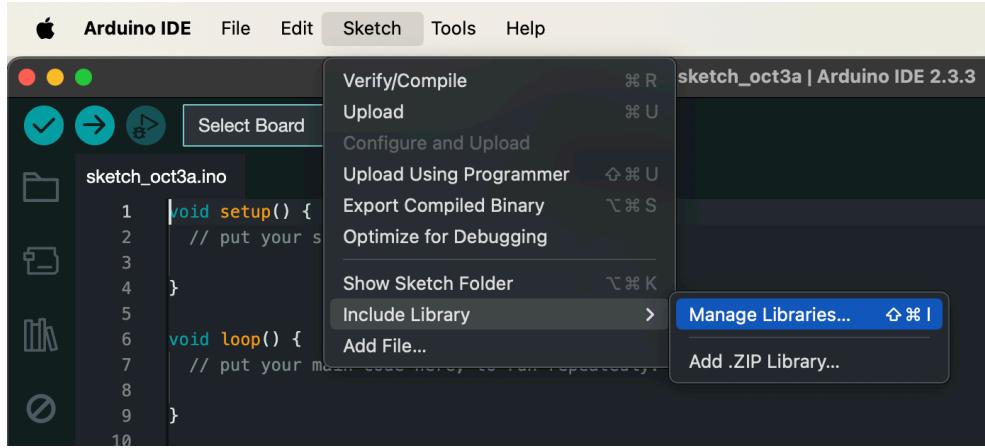
Below the code editor is an "Output" window showing the progress of library installations:

```
TFT@1.0.6
Installing TFT@1.0.6
Installed TFT@1.0.6
Downloading Arduino_BuiltIn@1.0.0
Arduino_BuiltIn@1.0.0
Installing Arduino_BuiltIn@1.0.0
Installed Arduino_BuiltIn@1.0.0
Downloading Ethernet@2.0.2
Ethernet@2.0.2
Installing Ethernet@2.0.2
Installed Ethernet@2.0.2
```

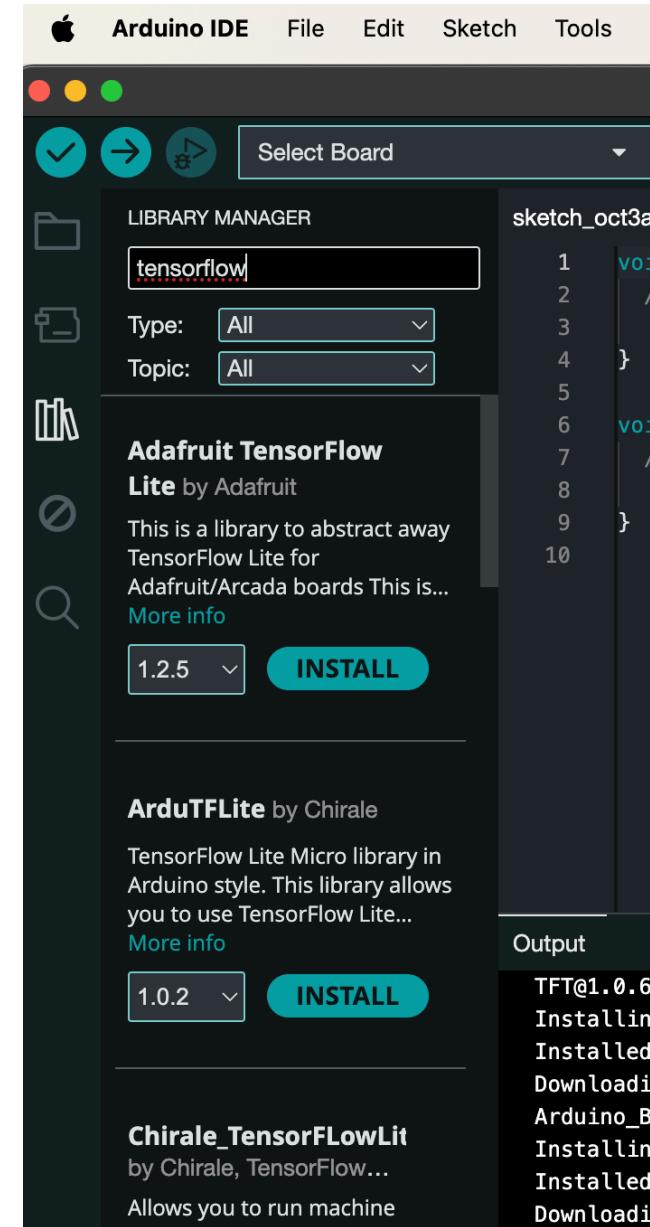
At the bottom right of the IDE, the status bar indicates "Ln 1, Col 1" and "No board selected".

Deploying a model

Go to:
Sketch
Include Library
Manage Libraries



Search for tensorflow
Install

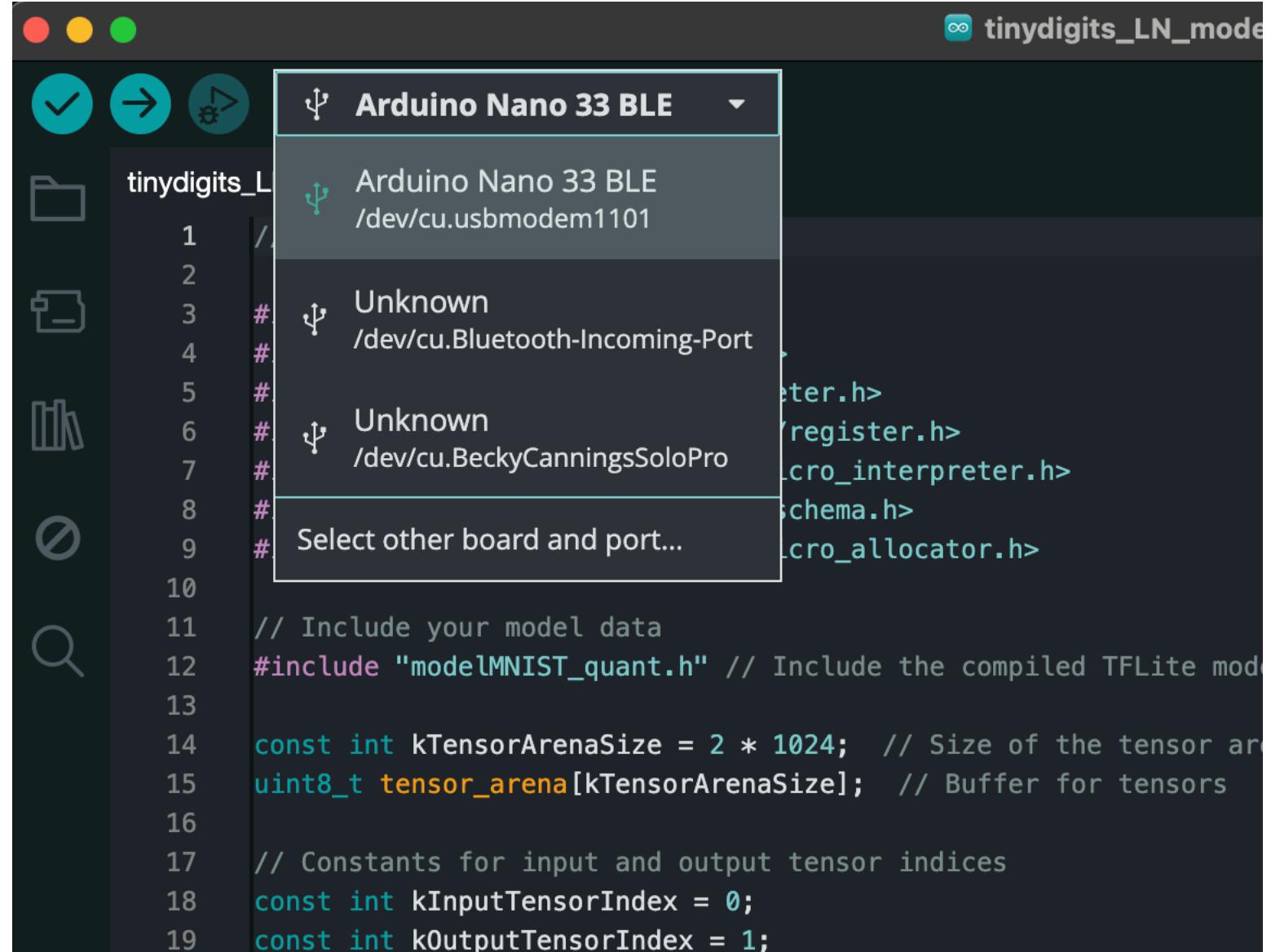


Deploying a model

Connect your Arduino

It will detect the board

Click on the board and the install
automatically - **not manual install**



The screenshot shows the Arduino IDE interface. In the top right corner, there is a status bar with the text "tinydigits_LN_mode". Below the status bar, the board selection dropdown is open, showing "Arduino Nano 33 BLE" selected. The dropdown also lists other detected boards: "Unknown /dev/cu.Bluetooth-Incoming-Port" and "Unknown /dev/cu.BeckyCanningsSoloPro". At the bottom of the dropdown menu, there is an option "Select other board and port...". The main code editor window displays the following C++ code:

```
1 // tinydigits_LN_mode
2
3 #include "modelMNIST_quant.h" // Include the compiled TFLite model
4
5 const int kTensorArenaSize = 2 * 1024; // Size of the tensor arena
6 uint8_t tensor_arena[kTensorArenaSize]; // Buffer for tensors
7
8 // Constants for input and output tensor indices
9 const int kInputTensorIndex = 0;
10 const int kOutputTensorIndex = 1;
```

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