

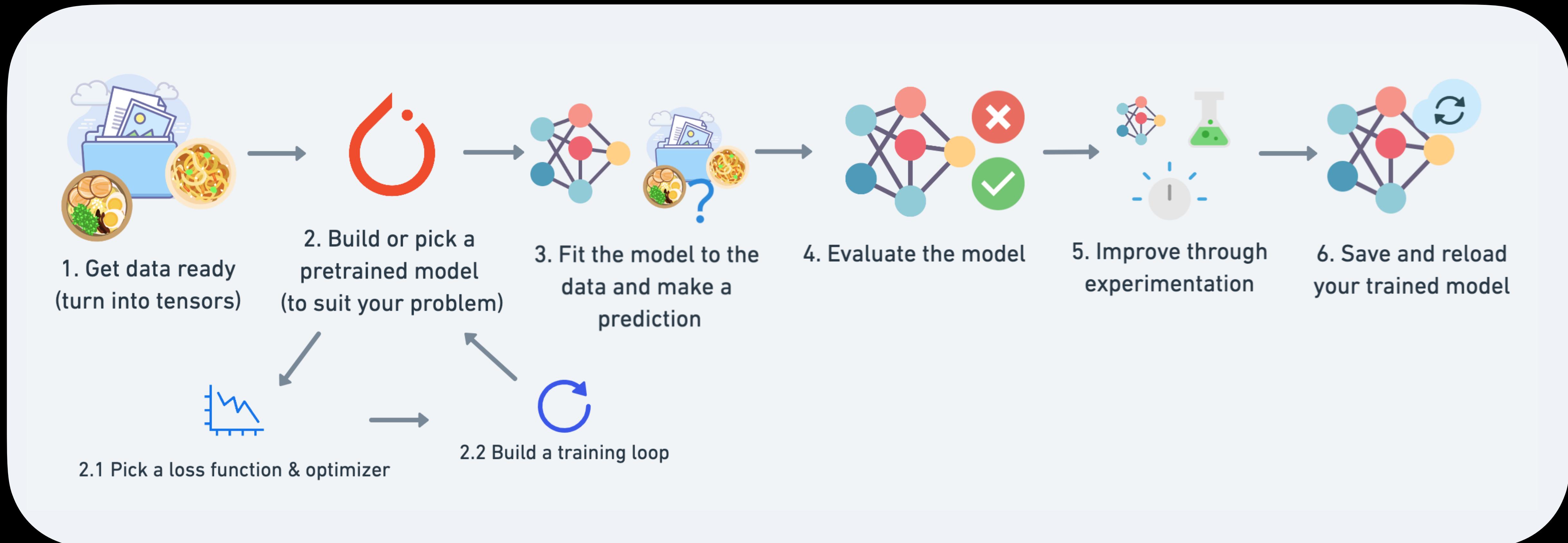


Workflow

What we're going to cover

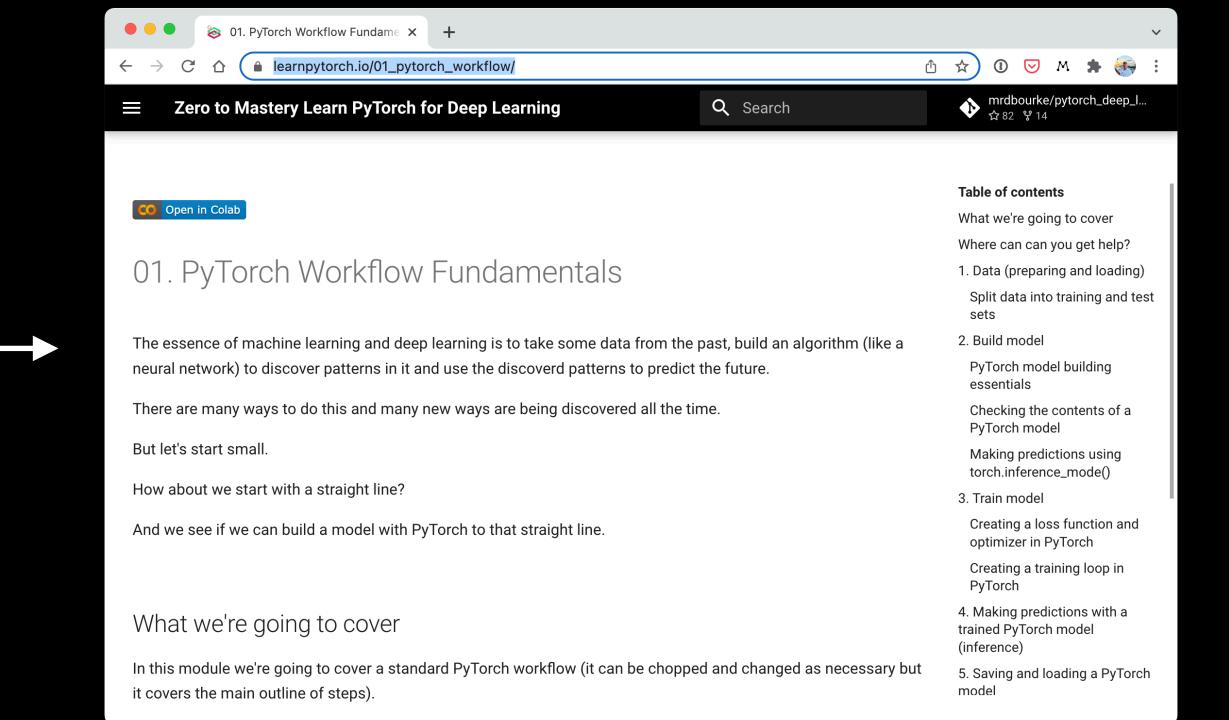
A PyTorch workflow

(one of many)



Where can you get help?

- Follow along with the code



Motto #1: "If in doubt, run the code"

- Try it for yourself

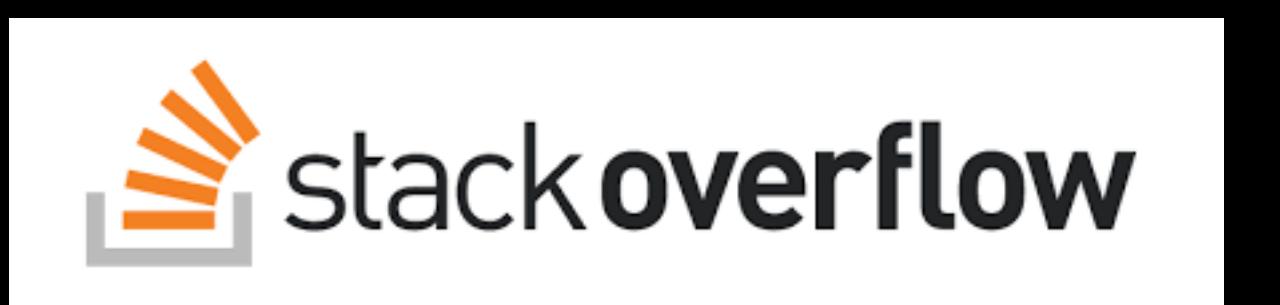
```
1 # Create a Linear Regression model
2 class LinearRegressionModel(nn.Module):
3     def __init__(self):
4         super().__init__()
5         self.weights = nn.Parameter(torch.randn(1, 1))
6         self.bias = nn.Parameter(torch.randn(1, 1))
7
8     def forward(self, x: torch.Tensor) -> torch.Tensor:
9         return self.weights * x + self.bias
```

A kind of Tensor that is to be considered a module parameter.
Parameters are `~torch.Tensor` subclasses, that have a
very special property when used with `Module`s - when they
are assigned as Module attributes they are automatically added to
the `parameters` and `named_parameters` - `Module` parameters.
This is important because it allows PyTorch to automatically
track these tensors when doing backpropagation.

Alright there's a fair bit going on above but let's break it down bit by bit.

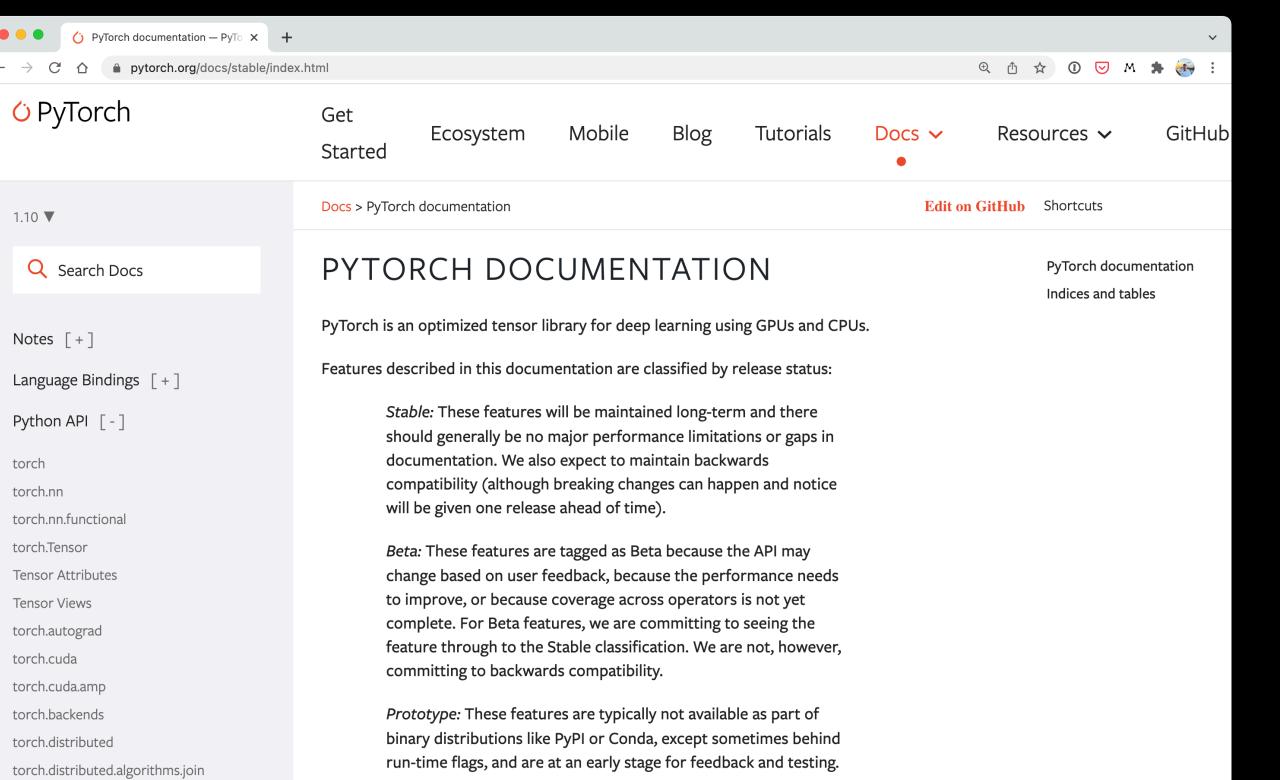
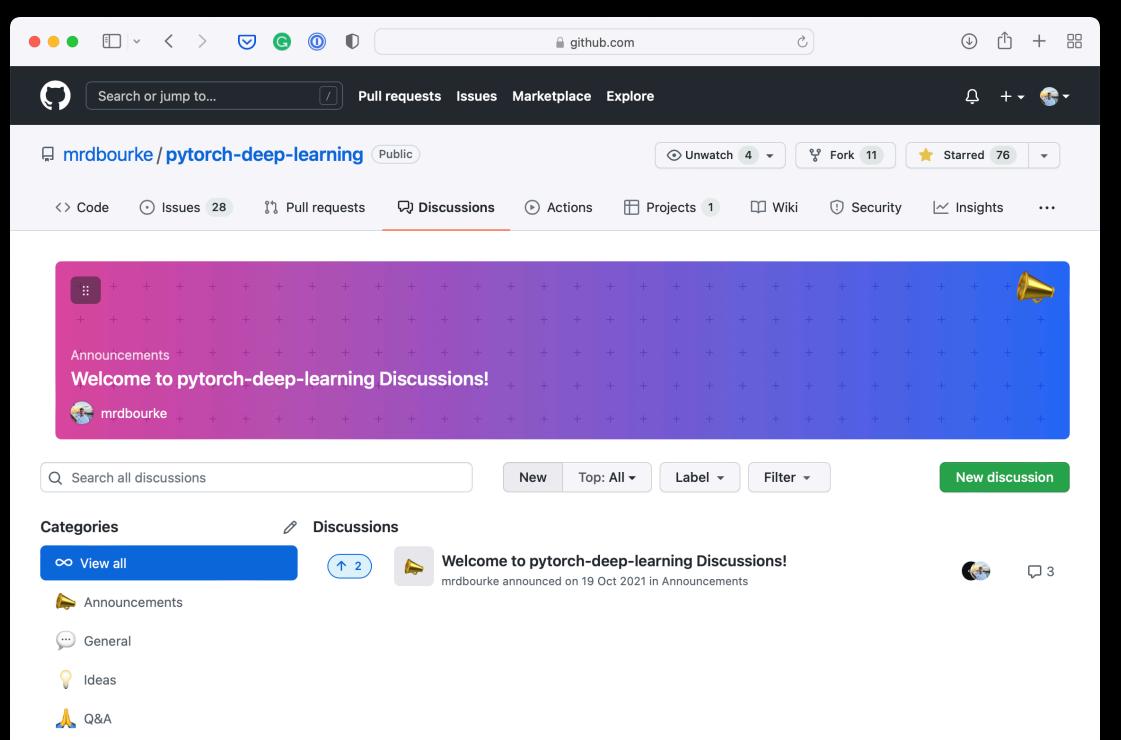
Resource: We'll be using Python classes to create bits and pieces for building neural networks. If you're unfamiliar with Python

- Press SHIFT + CMD + SPACE to read the docstring



- Search for it

- Try again

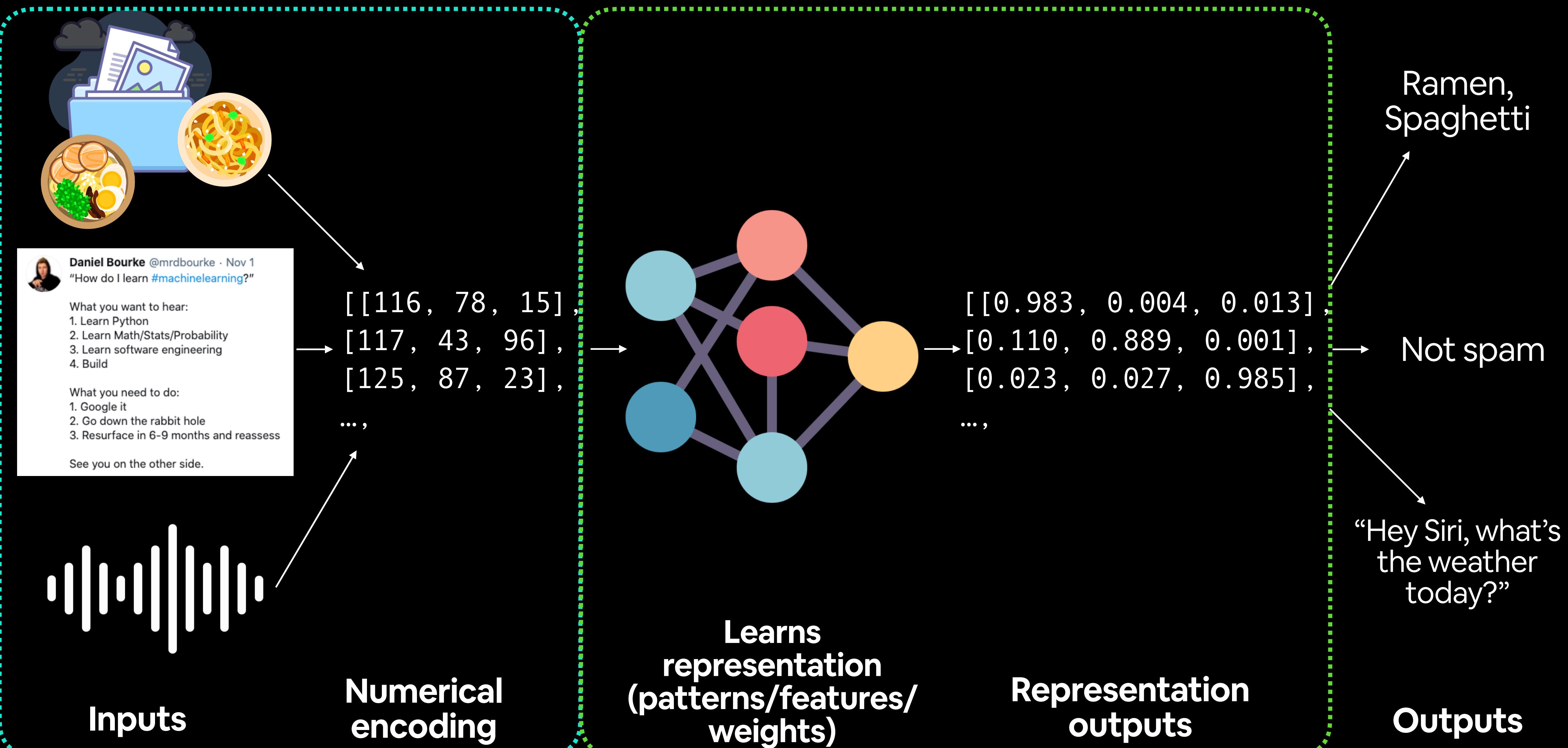


- Ask

<https://www.github.com/mrdbourke/pytorch-deep-learning/discussions>

Let's code!

Machine learning: a game of two parts



Part 1: Turn data into numbers



Daniel Bourke @mrdbourke · Nov 1
"How do I learn #machinelearning?"

What you want to hear:
1. Learn Python
2. Learn Math/Stats/Probability
3. Learn software engineering
4. Build

What you need to do:
1. Google it
2. Go down the rabbit hole
3. Resurface in 6-9 months and reassess

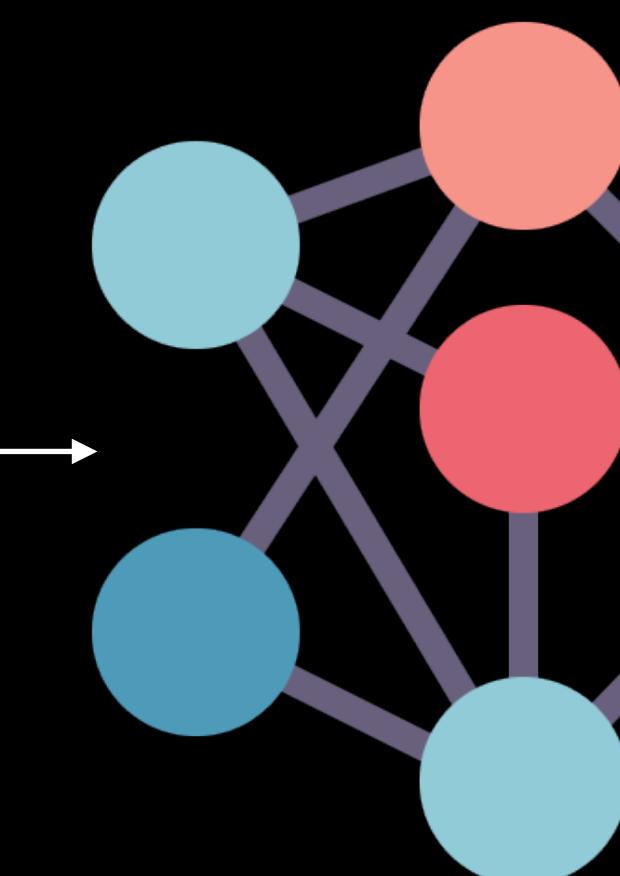
See you on the other side.



Inputs

Numerical encoding

Part 2: Build model to learn patterns in numbers



Learns representation (patterns/features/weights)

Representation outputs

Ramen,
Spaghetti

→ Not spam

"Hey Siri, what's the weather today?"

Outputs

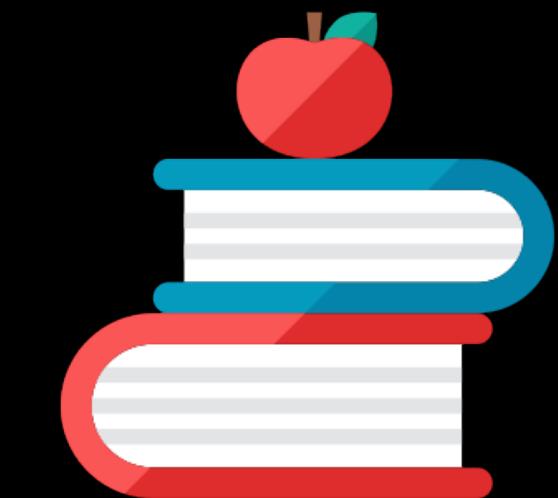
Three datasets

(possibly the most important concept in machine learning...)

Model learns patterns from here



**Course materials
(training set)**



**Practice exam
(validation set)**

Tune model patterns



**Final exam
(test set)**

See if the model is ready for the wild

Generalization

The ability for a machine learning model to perform well on data it hasn't seen before.

```
1 # Create a linear regression model in PyTorch
2 class LinearRegressionModel(nn.Module):
3     def __init__(self):
4         super().__init__()
5
6         # Initialize model parameters
7         self.weights = nn.Parameter(torch.randn(1,
8             requires_grad=True,
9             dtype=torch.float
10        ))
11
12        self.bias = nn.Parameter(torch.randn(1,
13            requires_grad=True,
14            dtype=torch.float
15        ))
16
17        # forward() defines the computation in the model
18    def forward(self, x: torch.Tensor) -> torch.Tensor:
19        return self.weights * x + self.bias
20
```

Subclass `nn.Module`

(this contains all the building blocks for neural networks)

Initialise **model parameters** to be used in various computations (these could be different layers from `torch.nn`, single parameters, hard-coded values or functions)

`requires_grad=True` means PyTorch will track the gradients of this specific parameter for use with `torch.autograd` and gradient descent (for many `torch.nn` modules, `requires_grad=True` is set by default)

Any subclass of `nn.Module` needs to override `forward()` (this defines the forward computation of the model)

PyTorch essential neural network building modules

PyTorch module	What does it do?
<u>torch.nn</u>	Contains all of the building blocks for computational graphs (essentially a series of computations executed in a particular way).
<u>torch.nn.Module</u>	The base class for all neural network modules, all the building blocks for neural networks are subclasses. If you're building a neural network in PyTorch, your models should subclass <u>nn.Module</u> . Requires a <u>forward()</u> method be implemented.
<u>torch.optim</u>	Contains various optimization algorithms (these tell the model parameters stored in <u>nn.Parameter</u> how to best change to improve gradient descent and in turn reduce the loss).
<u>torch.utils.data.Dataset</u>	Represents a map between key (label) and sample (features) pairs of your data. Such as images and their associated labels.
<u>torch.utils.data.DataLoader</u>	Creates a Python iterable over a torch Dataset (allows you to iterate over your data).

`torchvision.transforms`

`torch.utils.data.Dataset`

`torch.utils.data.DataLoader`

`torchmetrics`



1. Get data ready
(turn into tensors)



2. Build or pick a
pretrained model
(to suit your problem)



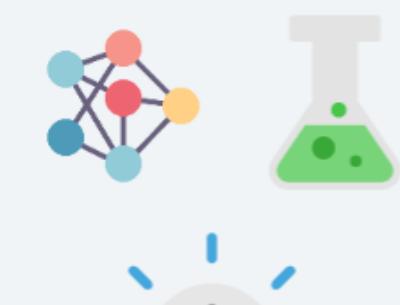
3. Fit the model to the
data and make a
prediction



2.1 Pick a loss function & optimizer



2.2 Build a training loop



4. Evaluate the model



5. Improve through
experimentation



6. Save and reload
your trained model

`torch.optim`

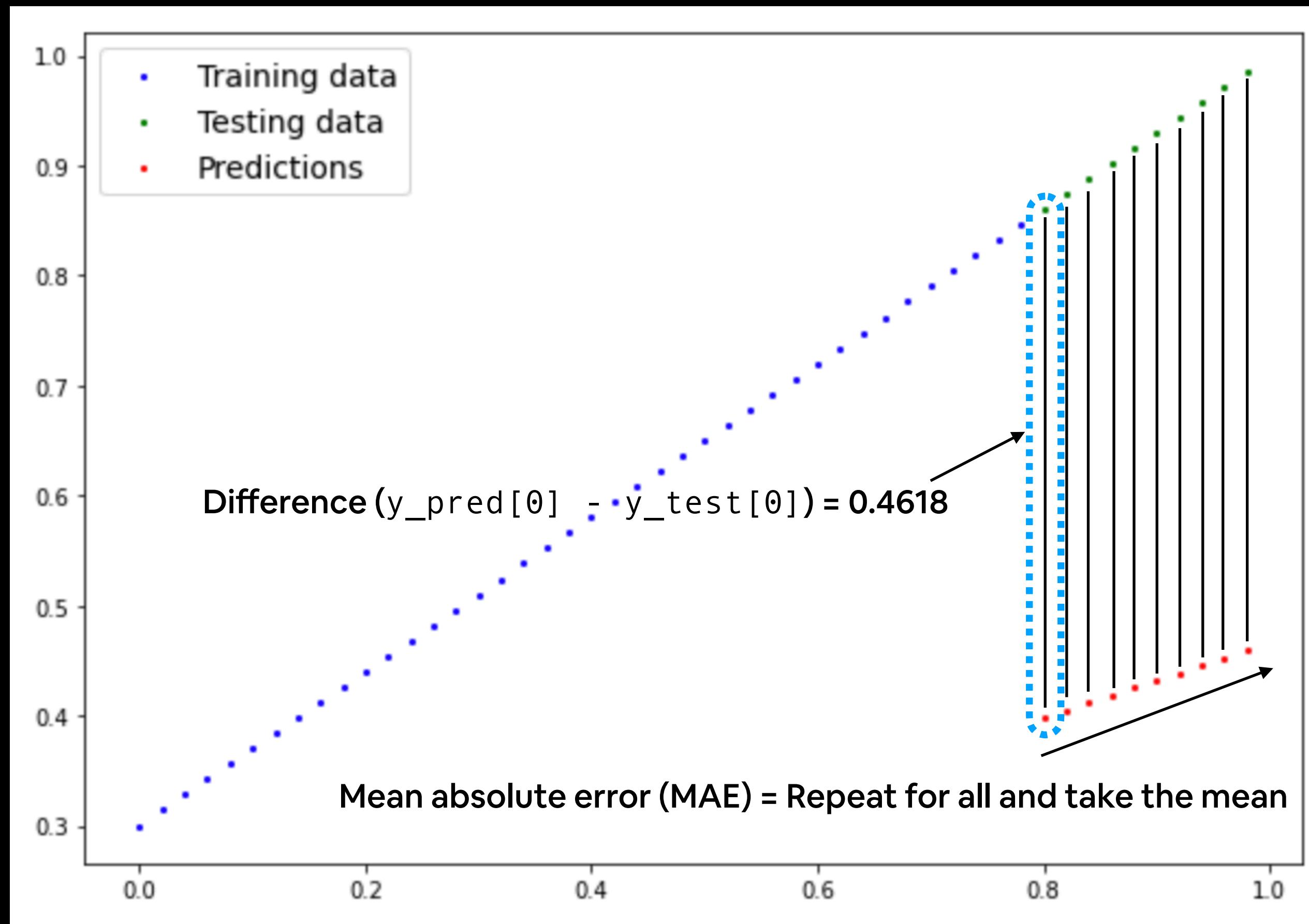
`torch.nn`

`torch.nn.Module`

`torchvision.models`

`torch.utils.tensorboard`

Mean absolute error (MAE)



```
MAE_loss = torch.mean(torch.abs(y_pred-y_test))  
or  
MAE_loss = torch.nn.L1Loss
```



Daniel Bourke
@mrdbourke

Let's sing the [@PyTorch](#) optimization loop song 🎵

It's train time!
do the forward pass,
calculate the loss,
optimizer zero grad,
losssss backwards!

Optimizer step step step

Let's test now!
with torch no grad:
do the forward pass,
calculate the loss,
watch it go down down down!

Source: [@mrdbourke Twitter](#) & [see the video version on YouTube](#).

PyTorch training loop

```
1 # Pass the data through the model for a number of epochs (e.g. 100)
2 for epoch in range(epochs):
3     # Put model in training mode (this is the default state of a model)
4     model.train()
5
6     # 1. Forward pass on train data using the forward() method inside
7     #      the model
8     y_pred = model(X_train)
9
10    # 2. Calculate the loss (how different are the model's predictions to the true values)
11    loss = loss_fn(y_pred, y_true)
12
13    # 3. Zero the gradients of the optimizer (they accumulate by default)
14    optimizer.zero_grad()
15
16    # 4. Perform backpropagation on the loss
17    loss.backward()
18
19    # 5. Progress/step the optimizer (gradient descent)
20    optimizer.step()
```

Note: all of this can be turned into a function

Pass the data through the model for a number of epochs
(e.g. 100 for 100 passes of the data)

Pass the data through the model, this will perform the
`forward()` method located within the model object

Calculate the loss value (how wrong the model's
predictions are)

Zero the optimizer gradients (they accumulate every
epoch, zero them to start fresh each forward pass)

Perform backpropagation on the loss function (compute
the gradient of every parameter with
`requires_grad=True`)

Step the optimizer to update the model's parameters with
respect to the gradients calculated by `loss.backward()`

PyTorch testing loop

```
1 # Setup empty lists to keep track of model progress
2 epoch_count = []
3 train_loss_values = []
4 test_loss_values = []
5
6 # Pass the data through the model for a number of epochs (e.g. 100) pochs):
7 for epoch in range(epochs):
8
9     ### Training loop code here #####
10
11    ### Testing starts #####
12
13    # Put the model in evaluation mode
14    model.eval()
15
16    # Turn on inference mode context manager
17    with torch.inference_mode():
18        # 1. Forward pass on test data
19        test_pred = model(X_test)
20
21        # 2. Calculate loss on test data
22        test_loss = loss_fn(test_pred, y_test)
23
24    # Print out what's happening every 10 epochs
25    if epoch % 10 == 0:
26        epoch_count.append(epoch)
27        train_loss_values.append(loss)
28        test_loss_values.append(test_loss)
29        print(f"Epoch: {epoch} | MAE Train Loss: {loss} | MAE Test Loss: {test_loss} ")
```

Note: all of this can be turned into a function

Create empty lists for storing useful values (helpful for tracking model progress)

Tell the model we want to evaluate rather than train (this turns off functionality used for training but not evaluation) *(faster performance!)*

Turn on `torch.inference_mode()` context manager to disable functionality such as gradient tracking for inference (gradient tracking not needed for inference)

Pass the test data through the model (this will call the model's implemented `forward()` method)

Calculate the test loss value (how wrong the model's predictions are on the test dataset, lower is better)

Display information outputs for how the model is doing during training/testing every ~10 epochs (note: what gets printed out here can be adjusted for specific problems)

```
1 # Setup optimization loop(s)
2 epochs = 10000
3
4 ### Train time!
5 # Loop through the epochs
6 for epoch in range(epochs):
7     # Set the model to train mode (this is the default)
8     model.train()
9
10    # 1. Do the forward pass
11    y_pred = model(x_train)
12
13    # 2. Calculate the loss (how wrong the model is)
14    loss = loss_fn(y_pred, y_train)
15
16    # 3. Zero the optimizer gradients (they accumulate by default)
17    optimizer.zero_grad()
18
19    # 4. Perform backpropagation (with respect to the model's parameters)
20    loss.backward()
21
22    # 5. Step the optimizer (gradient descent)
23    optimizer.step()
24
25 ### Test time!
26 # Set the model to eval mode (this turns off settings not needed for testing)
27 model.eval()
28 # Turn on inference mode context manager (removes even more things not needed for inference)
29 with torch.inference_mode():
30     # 1. Do the forward pass
31     test_pred = model(x_test)
32
33     # 2. Calculate the loss
34     test_loss = loss_fn(test_pred, y_test)
35
36 # Print out what's happenin'!
37 print(f"Epoch: {epoch} | Train loss: {loss:.4f} | Test loss: {test_loss:.4f}")
```

```
1 # Train function
2 def train_step(model, loss_fn, optimizer, data, labels):
3     # Turn on train mode (this is default but we turn it on anyway)
4     model.train()
5     # 1. Forward pass
6     y_pred = model(data)
7     # 2. Calculate the loss
8     loss = loss_fn(y_pred, labels)
9     # 3. Zero optimizer gradients
10    optimizer.zero_grad()
11    # 4. Perform backpropagation
12    loss.backward()
13    # 5. Perform gradient descent
14    optimizer.step()
15    return loss
```

```
1 # Test function
2 def test_step(model, loss_fn, data, labels):
3     # Turn on evaluation mode
4     model.eval()
5     # Setup inference mode context manager
6     with torch.inference_mode():
7         # 1. Forward pass
8         test_pred = model(data)
9         # 2. Calculate the loss
10        test_loss = loss_fn(test_pred, labels)
11        return test_loss
```

```
1 # Create a linear regression model in PyTorch
2 class LinearRegressionModel(nn.Module):
3     def __init__(self):
4         super().__init__()
5
6         # Initialize model parameters
7         self.weights = nn.Parameter(torch.randn(1,
8             requires_grad=True,
9             dtype=torch.float
10            ))
11
12         self.bias = nn.Parameter(torch.randn(1,
13             requires_grad=True,
14             dtype=torch.float
15            ))
16
17     # forward() defines the computation in the model
18     def forward(self, x: torch.Tensor) -> torch.Tensor:
19         return self.weights * x + self.bias
20
```

Linear regression model with nn.Parameter

```
1 # Create a linear regression model in PyTorch with nn.Linear
2 class LinearRegressionModel(nn.Module):
3     def __init__(self):
4         super().__init__()
5
6         # Use nn.Linear() for creating the model parameters
7         self.linear_layer = nn.Linear(in_features=1,
8                                         out_features=1)
9
10    # forward() defines the computation in the model
11    def forward(self, x: torch.Tensor) -> torch.Tensor:
12        return self.linear_layer(x)
```

Linear regression model with nn.Linear

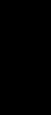
Supervised learning (overview)



1. Initialise with random weights (only at beginning)

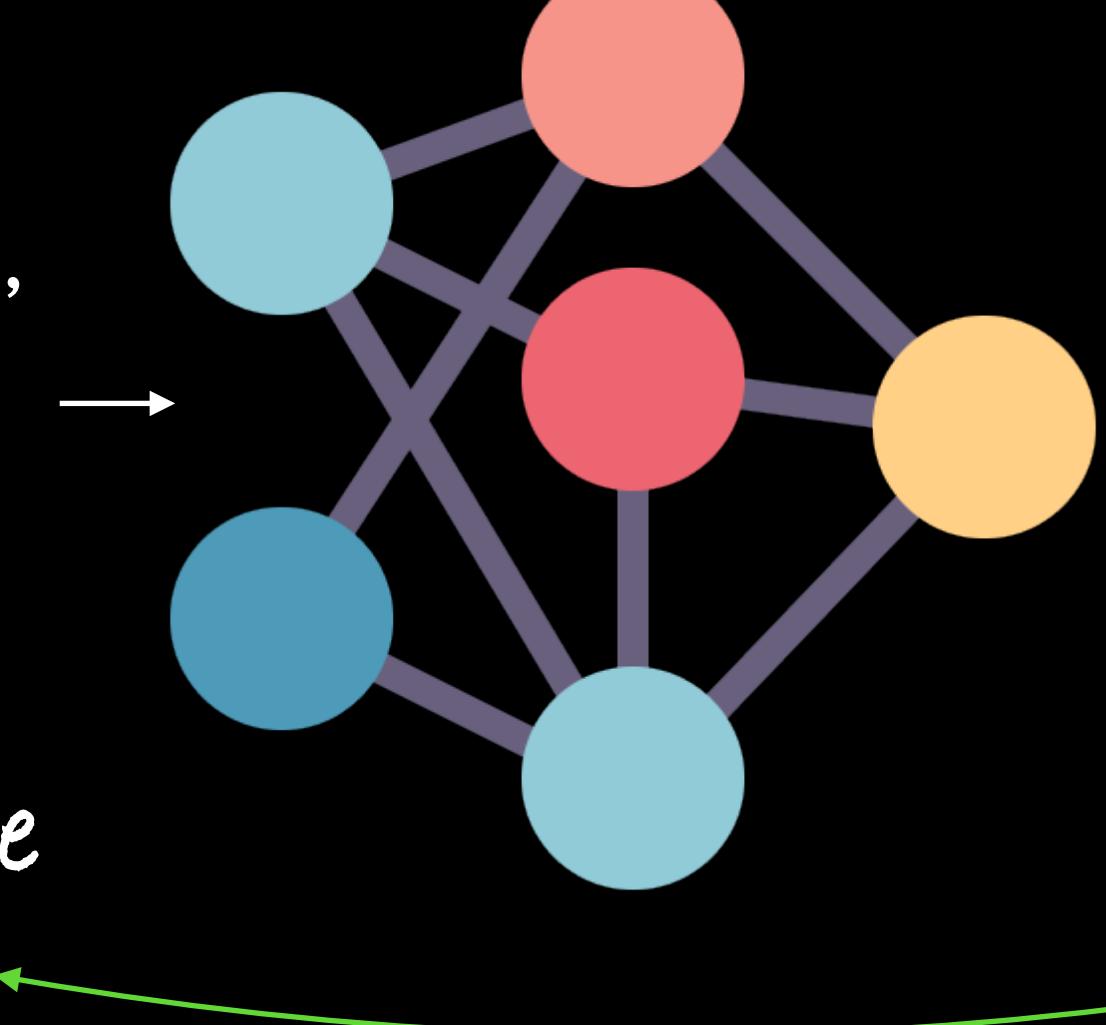
$[[0.092, 0.210, 0.415],$
 $[0.778, 0.929, 0.030], \rightarrow$
 $[0.019, 0.182, 0.555],$

...,



$\rightarrow [[116, 78, 15],$
 $\rightarrow [117, 43, 96], \rightarrow$
 $[125, 87, 23],$

...,



4. Repeat with more examples

$\rightarrow [[0.983, 0.004, 0.013],$
 $\rightarrow [0.110, 0.889, 0.001], \rightarrow$
 $[0.023, 0.027, 0.985],$

...,

3. Update representation outputs

Ramen,
Spaghetti

Inputs

Numerical encoding

Learns representation patterns/features/weights

Representation outputs

Outputs

2. Show examples

Neural Networks

These are tensors!

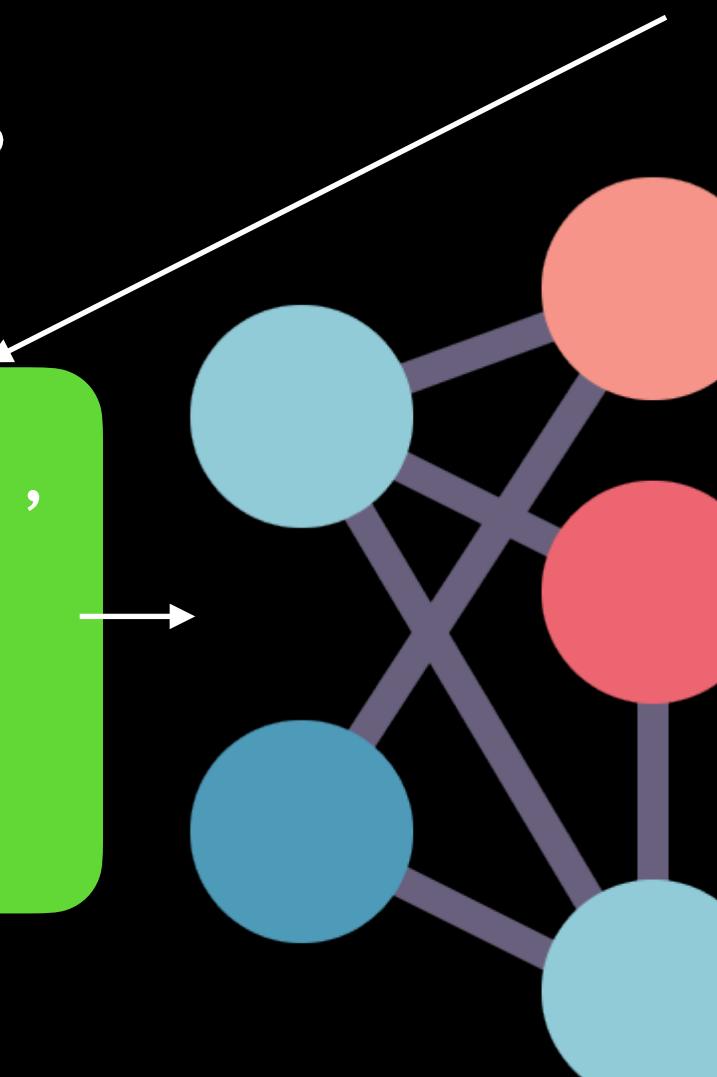


Inputs

Numerical encoding

(before data gets used with an algorithm, it needs to be turned into numbers)

```
[[116, 78, 15],  
 [117, 43, 96],  
 [125, 87, 23],  
 ... ,
```



Learns representation (patterns/features/weights)

```
[[0.983, 0.004, 0.013],  
 [0.110, 0.889, 0.001],  
 [0.023, 0.027, 0.985],  
 ... ,
```

Outputs

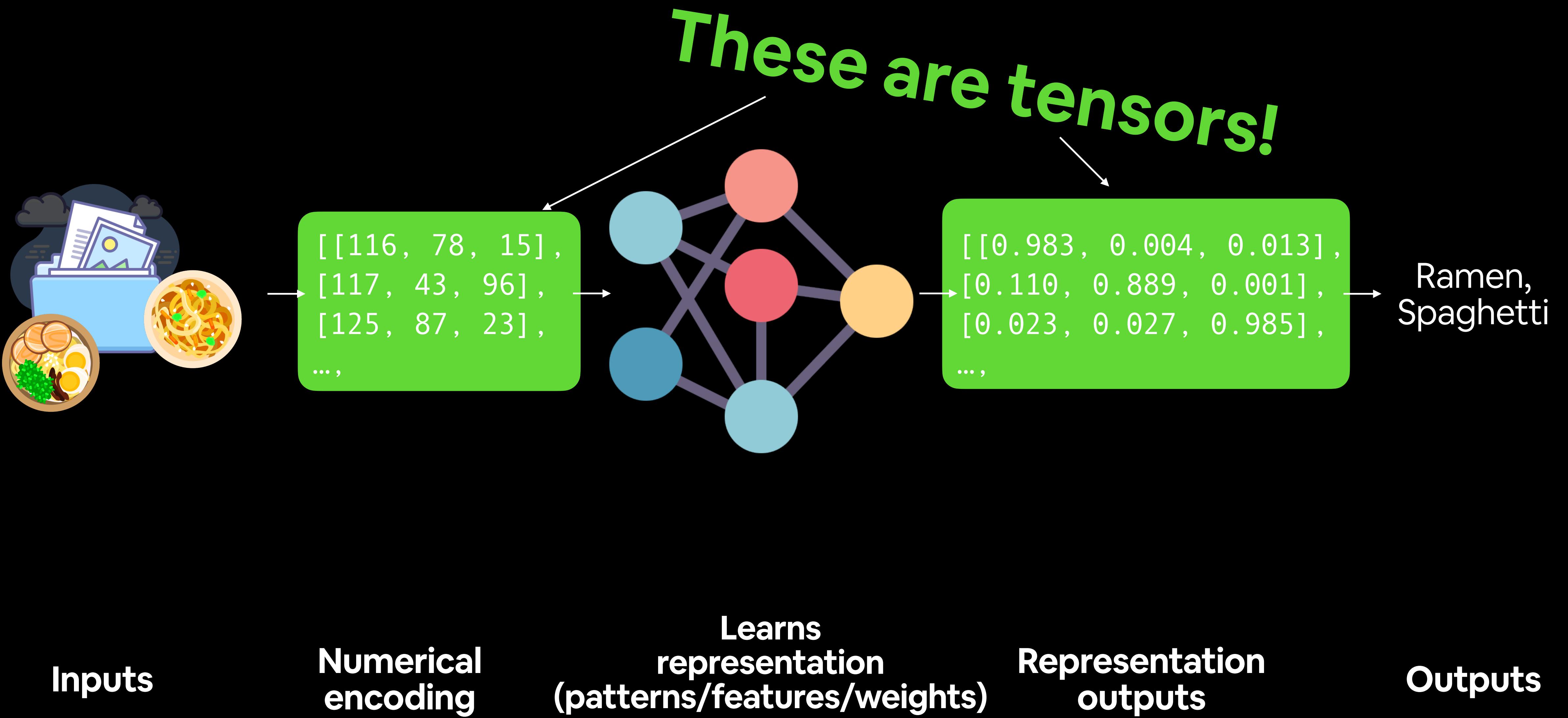
(a human can understand these)

Ramen,
Spaghetti

Not spam

"Hey Siri, what's the weather today?"

(choose the appropriate neural network for your problem)



How to approach this course

```
1 # 1. Construct a model class that subclasses nn.Module
2 class CircleModelV0(nn.Module):
3     def __init__(self):
4         super().__init__()
5         # 2. Create 2 nn.Linear layers
6         self.layer_1 = nn.Linear(in_features=2, out_features=5)
7         self.layer_2 = nn.Linear(in_features=5, out_features=1)
8
9     # 3. Define a forward method containing the forward pass computation
10    def forward(self, x):
11        # Pass the data through both layers
12        return self.layer_2(self.layer_1(x))
13
14 # 4. Create an instance of the model and send it to target device
15 model_0 = CircleModelV0().to(device)
16 model_0
```

1. Code along

Motto #1: if in doubt, run the code!



(including the
“dumb” ones)

4. Ask questions

Motto #2:
Experiment, experiment,
experiment!



2. Explore and experiment



5. Do the exercises



3. Visualize what you don't understand



6. Share your work

How **not** to approach this course

Avoid:



“I can’t learn
”

Resources

This course

Course materials

A screenshot of a GitHub repository page for 'mrdbourke/pytorch-deep-learning'. The repository has 28 issues, 11 forks, and 76 stars. The 'Code' tab is selected, showing a list of files and commits. Key commits include 'update exercises', 'Update make_docs.yml', and 'add exercises and solutions for 01'. The repository is described as 'Materials for upcoming beginner-friendly PyTorch course (work in progress)'.

<https://www.github.com/mrdbourke/pytorch-deep-learning>

Course Q&A

A screenshot of the GitHub Discussions page for the same repository. It features a purple header with the text 'Welcome to pytorch-deep-learning Discussions!'. Below the header, there's a search bar and a 'New discussion' button. Categories listed include 'Announcements', 'General', 'Ideas', and 'Q&A'. A message from 'mrdbourke' welcomes users to the discussions.

<https://www.github.com/mrdbourke/pytorch-deep-learning/discussions>

Course online book

A screenshot of the 'Zero to Mastery Learn PyTorch for Deep Learning' online book homepage. It features a dark theme with white text. The page includes a welcome message, a brief description of the course, and links to full code and resources. It also mentions an expected release date of Early 2022 and provides instructions for getting updates via the GitHub repo or email.

<https://learnpytorch.io>

PyTorch website & forums

A screenshot of the PyTorch website. The main heading is 'FROM RESEARCH TO PRODUCTION'. Below it, a subtext reads 'An open source machine learning framework that accelerates the path from research prototyping to production deployment.' There is a 'Install' button and a note about the 'PyTorch 1.10 Release, including CUDA Graphs APIs, TorchScript improvements'.

All things PyTorch

A screenshot of the PyTorch forum. The top navigation bar includes 'Sign Up', 'Log In', and a search icon. The main content area shows a list of categories: 'vision', 'nlp', 'Uncategorized', 'autograd', 'mixed-precision', 'C++', and 'distributed'. Each category has a list of posts with details like the number of topics per month and the latest post.



Daniel Bourke
@mrdbourke

Let's sing the [@PyTorch](#) optimization loop song 🎵

It's train time!
do the forward pass,
calculate the loss,
optimizer zero grad,
losssss backwards!

Optimizer step step step

Let's test now!
with torch no grad:
do the forward pass,
calculate the loss,
watch it go down down down!