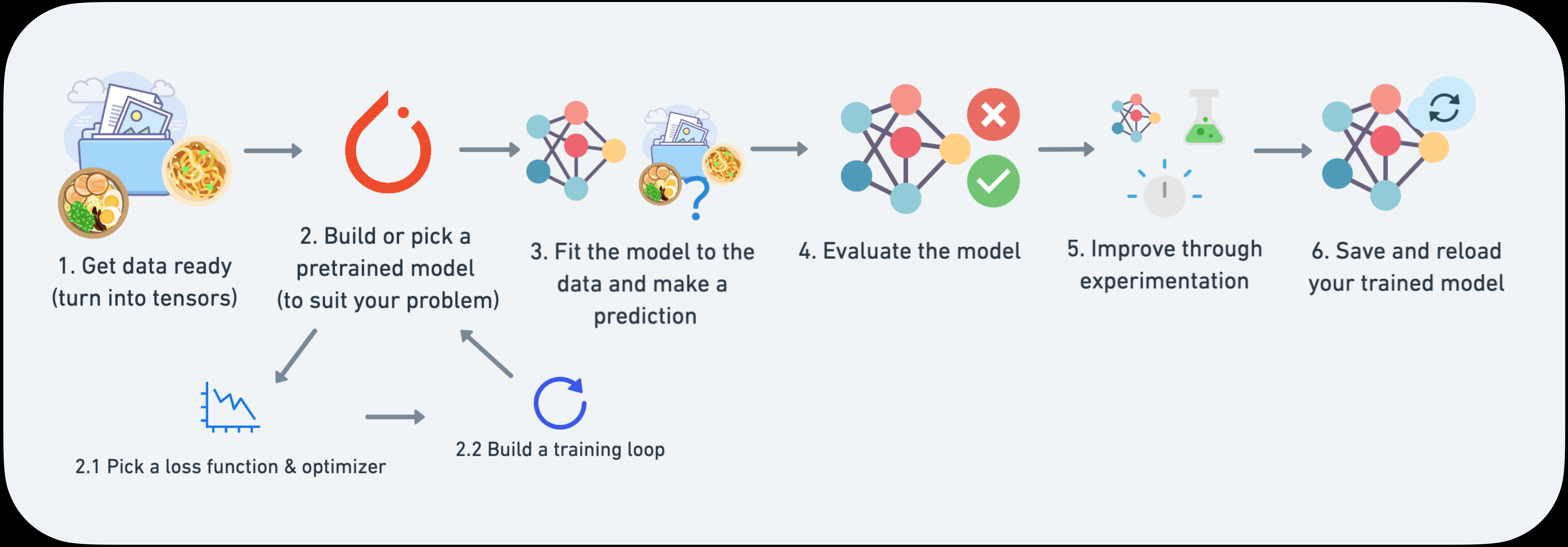
PyTorch

Workflow

What we're going to cover

A PyTorch workflow

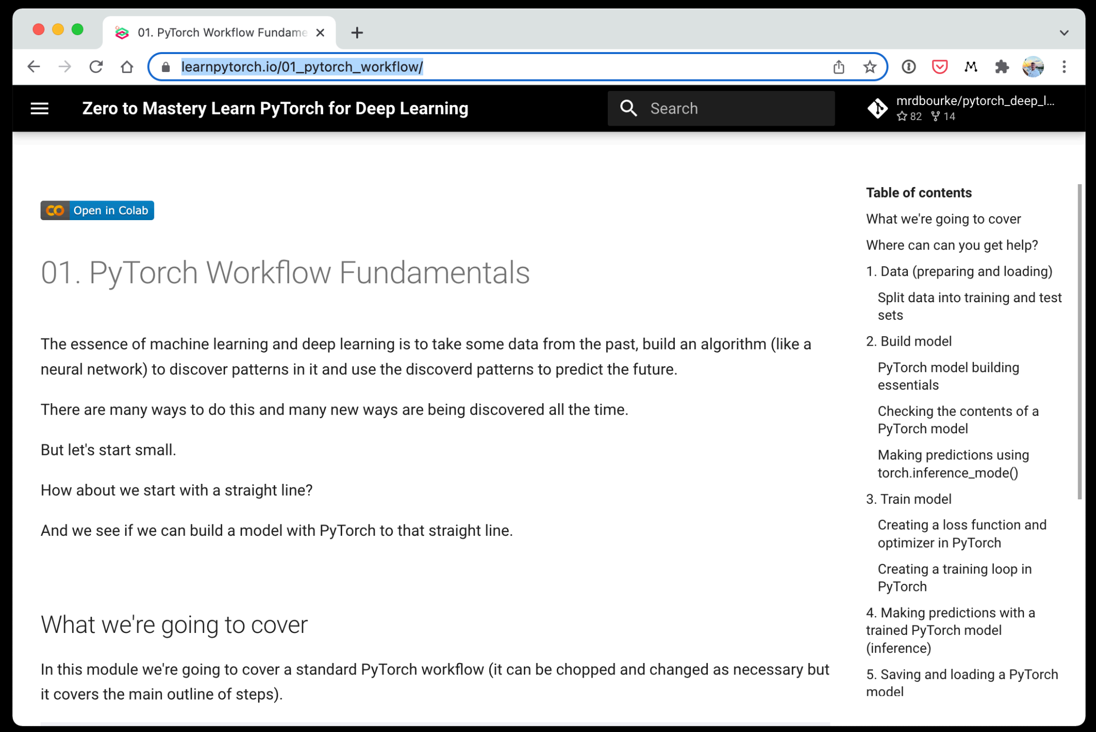
(one of many)



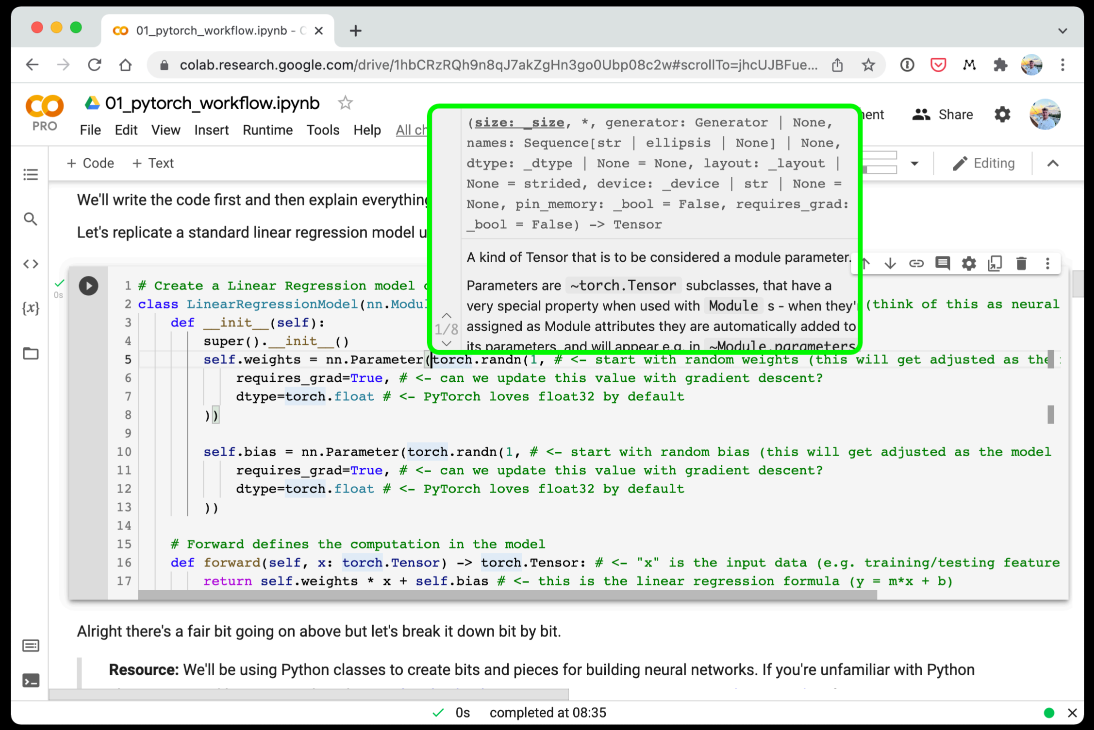
Where can you get help?

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

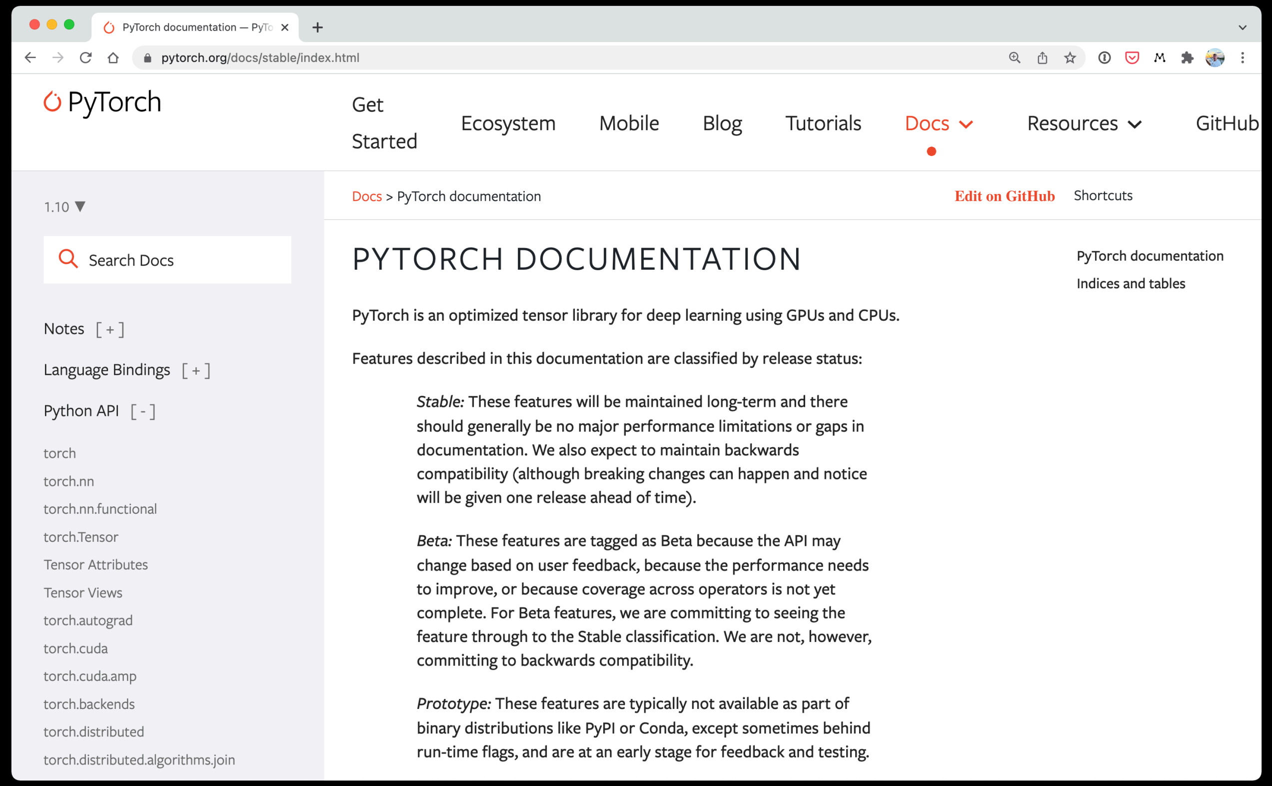
- Follow along with the code



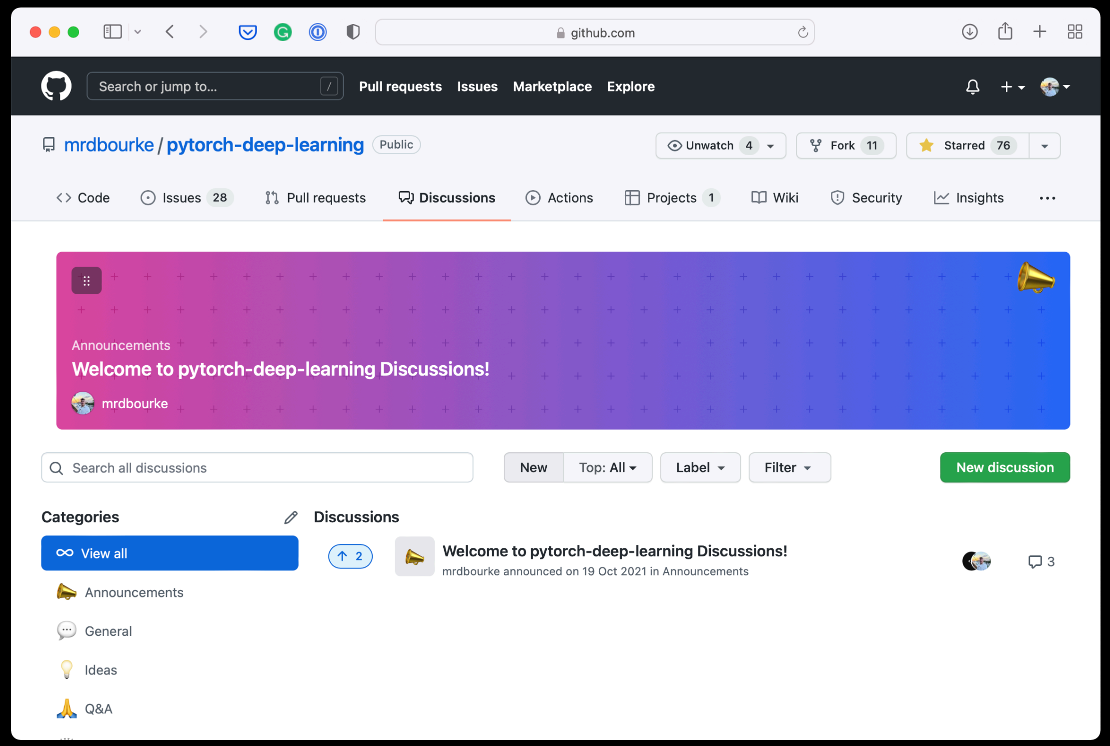
- Try it for yourself
- Press SHIFT + CMD + SPACE to read the docstring



- Search for it



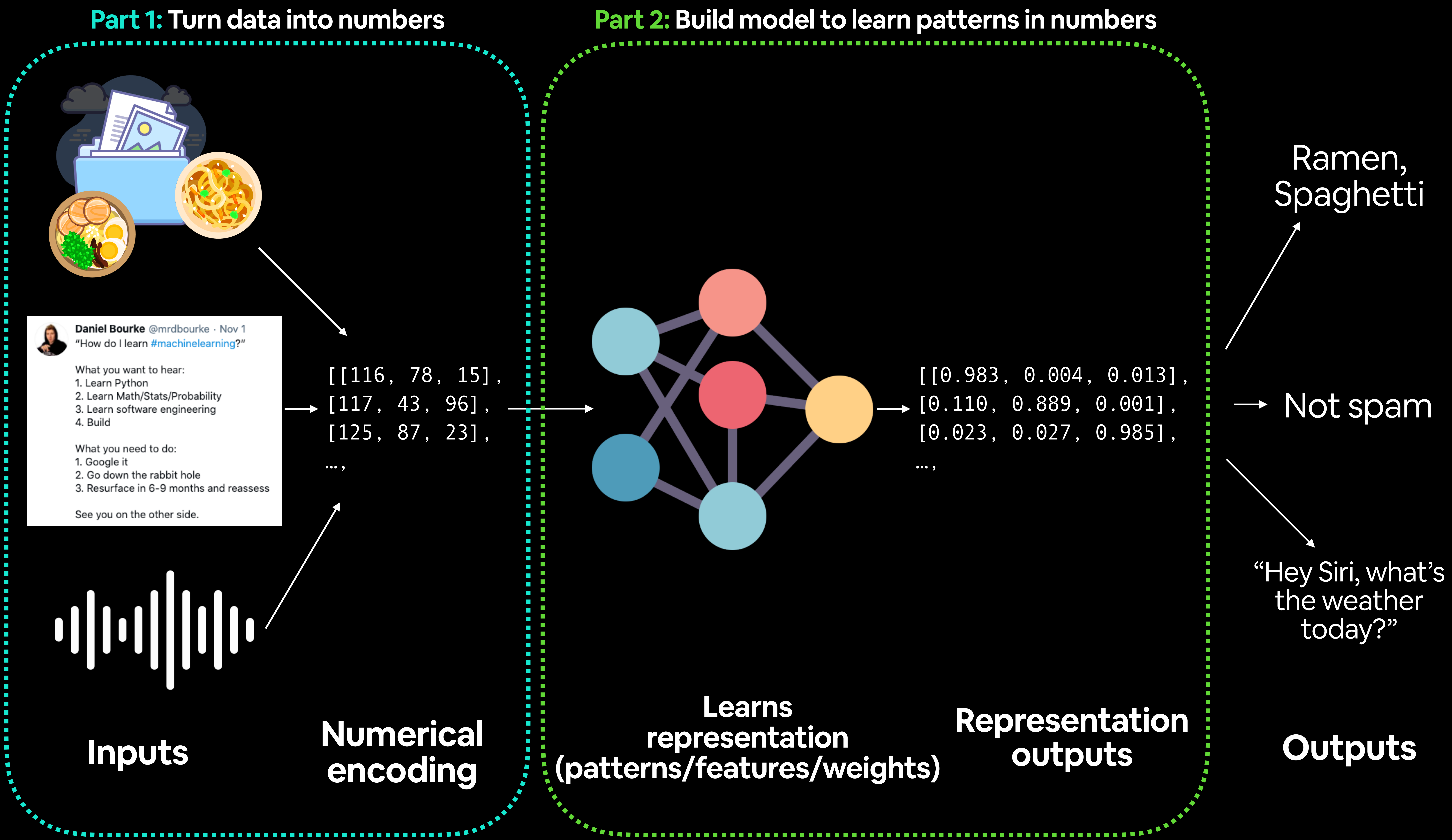
- Try again
- Ask



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
Part 2: Build model to learn patterns in 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.
[[116, 78, 15],
[117, 43, 96],
[125, 87, 23],
...,
[[0.983, 0.004, 0.013],
[0.110, 0.889, 0.001],
[0.023, 0.027, 0.985],
...,
Ramen, Spaghetti
Not spam
"Hey Siri, what's the weather today?"
Inputs
Numerical encoding
Learns representation (patterns/features/weights)
Representation outputs
Outputs

Three datasets

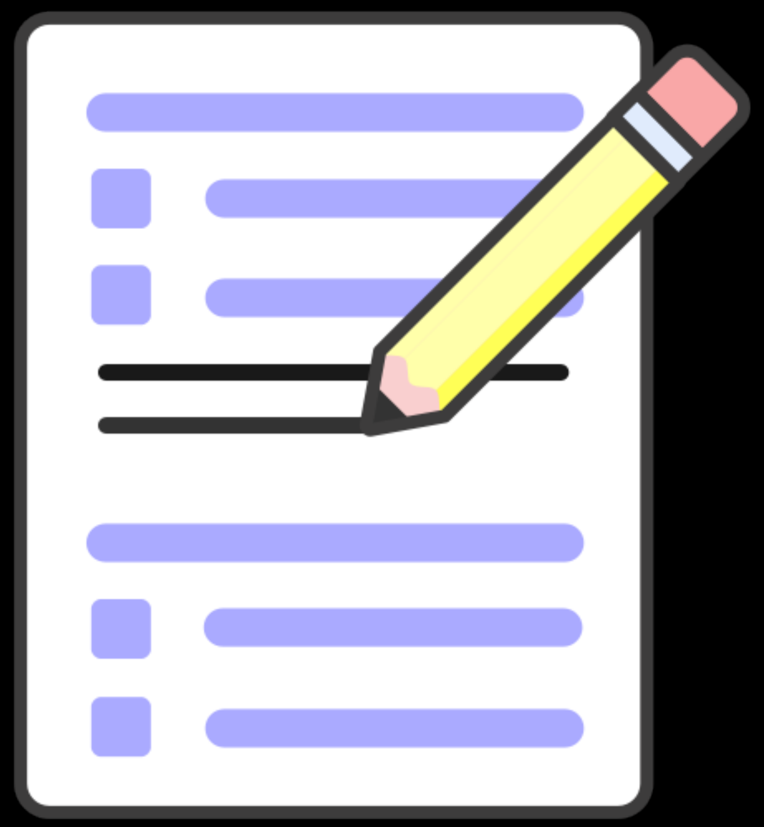
(possibly the most important concept in machine learning...)

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.

```python
# Create a linear regression model in PyTorch
class LinearRegressionModel(nn.Module):
    def __init__(self):
        super().__init__()

        # Initialize model parameters
        self.weights = nn.Parameter(torch.randn(1,
            requires_grad=True,
            dtype=torch.float
        ))

        self.bias = nn.Parameter(torch.randn(1,
            requires_grad=True,
            dtype=torch.float
        ))

    # forward() defines the computation in the model
    def forward(self, x: torch.Tensor) -> torch.Tensor:
        return self.weights * x + self.bias
```

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)

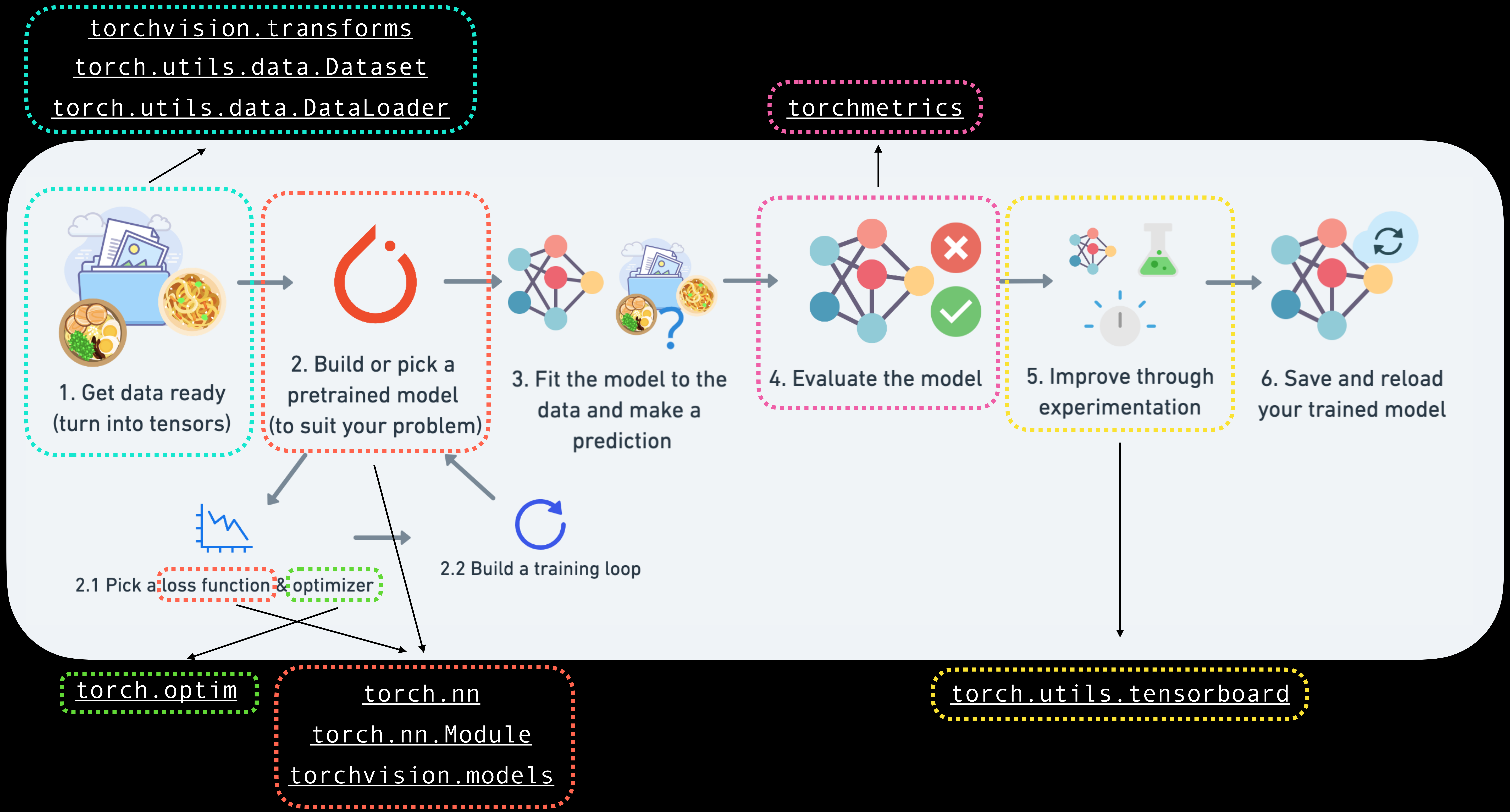
`requires_grad=True` means PyTorch will track the gradients of this specific parameter for use with `torch.autograd` and gradient descent (for many `torch.nn` modules, `requires_grad=True` is set by default)

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

PyTorch essential neural network building modules

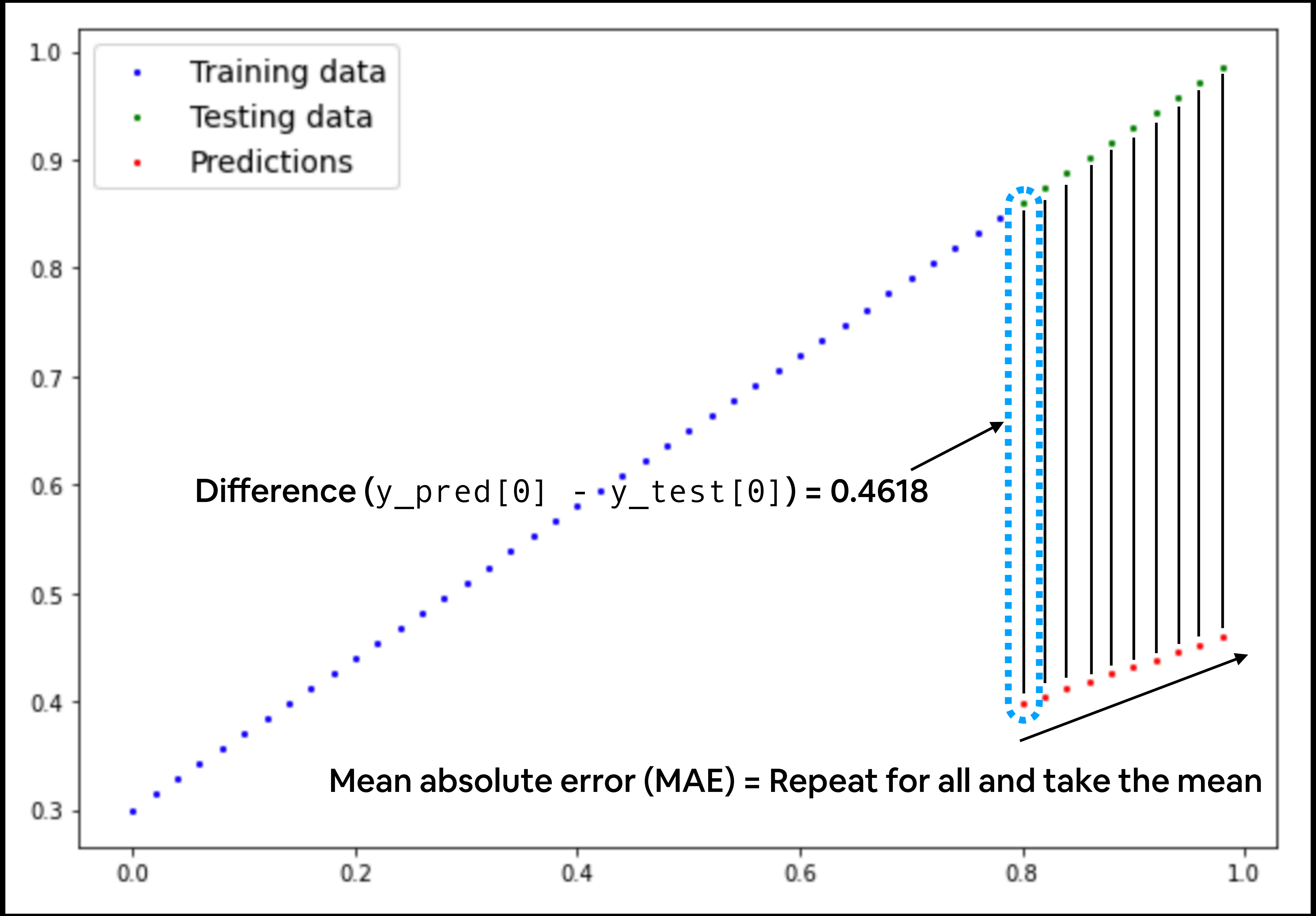
PyTorch module	What does it do?
`torch.nn`	Contains all of the building blocks for computational graphs (essentially a series of computations executed in a particular way).
`torch.nn.Module`	The base class for all neural network modules, all the building blocks for neural networks are subclasses. If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

See more: https://pytorch.org/tutorials/beginner/ptcheat.html



See more: https://pytorch.org/tutorials/beginner/ptcheat.html

Mean absolute error (MAE)




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

See more: https://pytorch.org/tutorials/beginner/ptcheat.html#loss-functions

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

```python
# Pass the data through the model for a number of epochs (e.g. 100)
for epoch in range(epochs):

    # Put model in training mode (this is the default state of a model)
    model.train()

    # 1. Forward pass on train data using the forward() method inside
    y_pred = model(X_train)

    # 2. Calculate the loss (how different are the model's predictions to the true values)
    loss = loss_fn(y_pred, y_true)

    # 3. Zero the gradients of the optimizer (they accumulate by default)
    optimizer.zero_grad()

    # 4. Perform backpropagation on the loss
    loss.backward()

    # 5. Progress/step the optimizer (gradient descent)
    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

```python
# Setup empty lists to keep track of model progress
epoch_count = []
train_loss_values = []
test_loss_values = []

# Pass the data through the model for a number of epochs (e.g. 100) pochs):
for epoch in range(epochs):

    ### Training loop code here ###

    ### Testing starts ###

    # Put the model in evaluation mode
    model.eval()

    # Turn on inference mode context manager
    with torch.inference_mode():
      # 1. Forward pass on test data
      test_pred = model(X_test)

      # 2. Caculate loss on test data
      test_loss = loss_fn(test_pred, y_test)

    # Print out what's happening every 10 epochs
    if epoch % 10 == 0:
        epoch_count.append(epoch)
        train_loss_values.append(loss)
        test_loss_values.append(test_loss)
        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)

See more: https://discuss.pytorch.org/t/model-eval-vs-with-torch-no-grad/19615 & PyTorch Twitter announcement of torch.inference_mode()

```python
# Setup optimization loop(s)
epochs = 10000

### Train time!
# Loop through the epochs
for epoch in range(epochs):
  # Set the model to train mode (this is the default)
  model.train()

  # 1. Do the forward pass
  y_pred = model(X_train)

  # 2. Calculate the loss (how wrong the model is)
  loss = loss_fn(y_pred, y_train)

  # 3. Zero the optimizer gradients (they accumulate by default)
  optimizer.zero_grad()

  # 4. Perform backpropagation (with respect to the model's parameters)
  loss.backward()

  # 5. Step the optimizer (gradient descent)
  optimizer.step()

  ### Test time!
  # Set the model to eval mode (this turns off settings not needed for testing)
  model.eval()
  # Turn on inference mode context manager (removes even more things not needed for inference)
  with torch.inference_mode():
    # 1. Do the forward pass
    test_pred = model(X_test)

    # 2. Calculate the loss
    test_loss = loss_fn(test_pred, y_test)

  # Print out what's happenin'!
  print(f"Epoch: {epoch} | Train loss: {loss:.4f} | Test loss: {test_loss:.4f}")
```

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

```python
# Test function
def test_step(model, loss_fn, data, labels):
  # Turn on evaluation mode
  model.eval()
  # Setup inference mode context manager
  with torch.inference_mode():
    # 1. Forward pass
    test_pred = model(data)
    # 2. Calculate the loss
    test_loss = loss_fn(test_pred, labels)
  return test_loss
```

```python
# Create a linear regression model in PyTorch
class LinearRegressionModel(nn.Module):
    def __init__(self):
        super().__init__()

        # Initialize model parameters
        self.weights = nn.Parameter(torch.randn(1,
            requires_grad=True,
            dtype=torch.float
        ))

        self.bias = nn.Parameter(torch.randn(1,
            requires_grad=True,
            dtype=torch.float
        ))

    # forward() defines the computation in the model
    def forward(self, x: torch.Tensor) -> torch.Tensor:
        return self.weights * x + self.bias
```

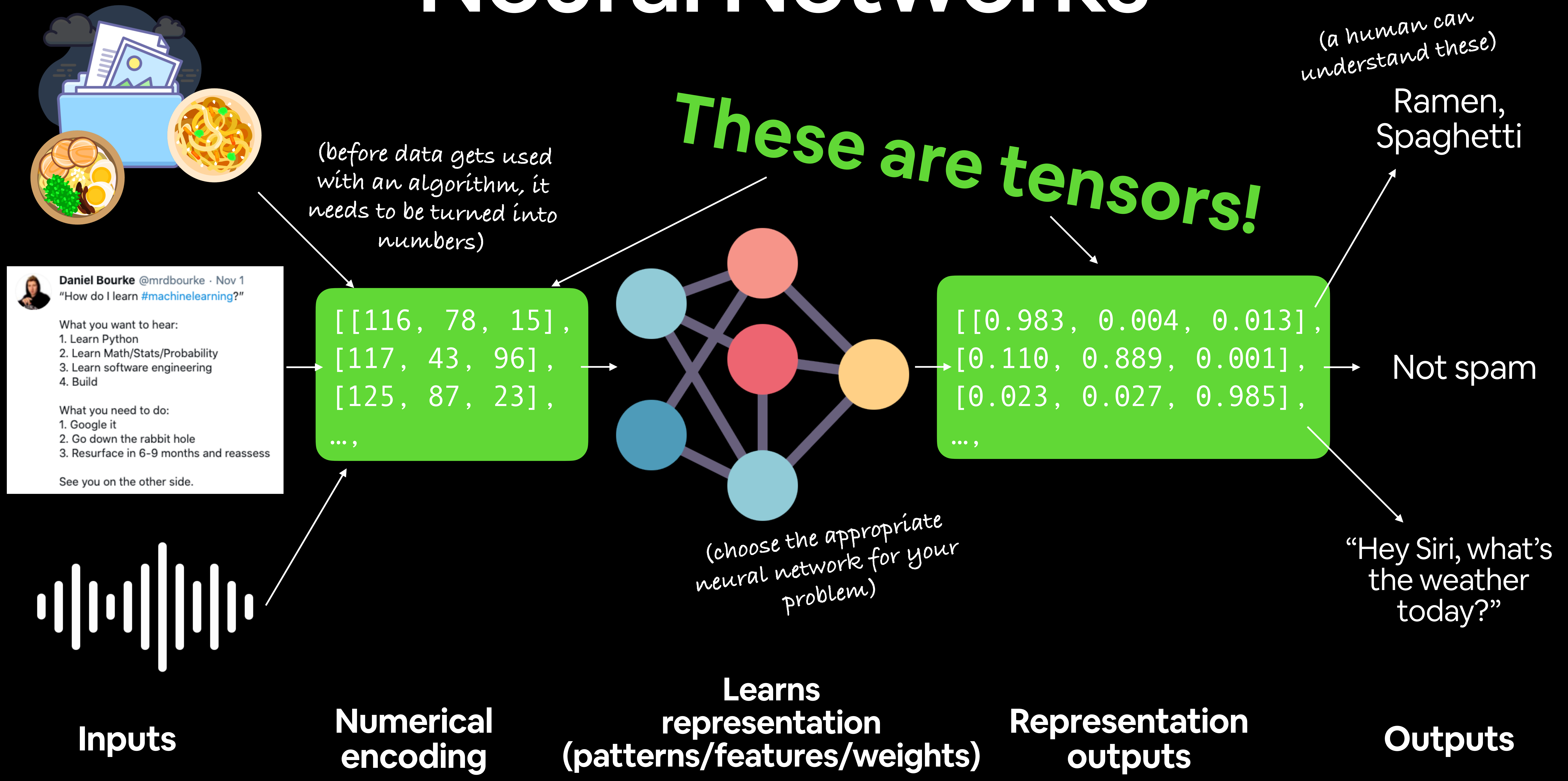
Linear regression model with `nn.Parameter`

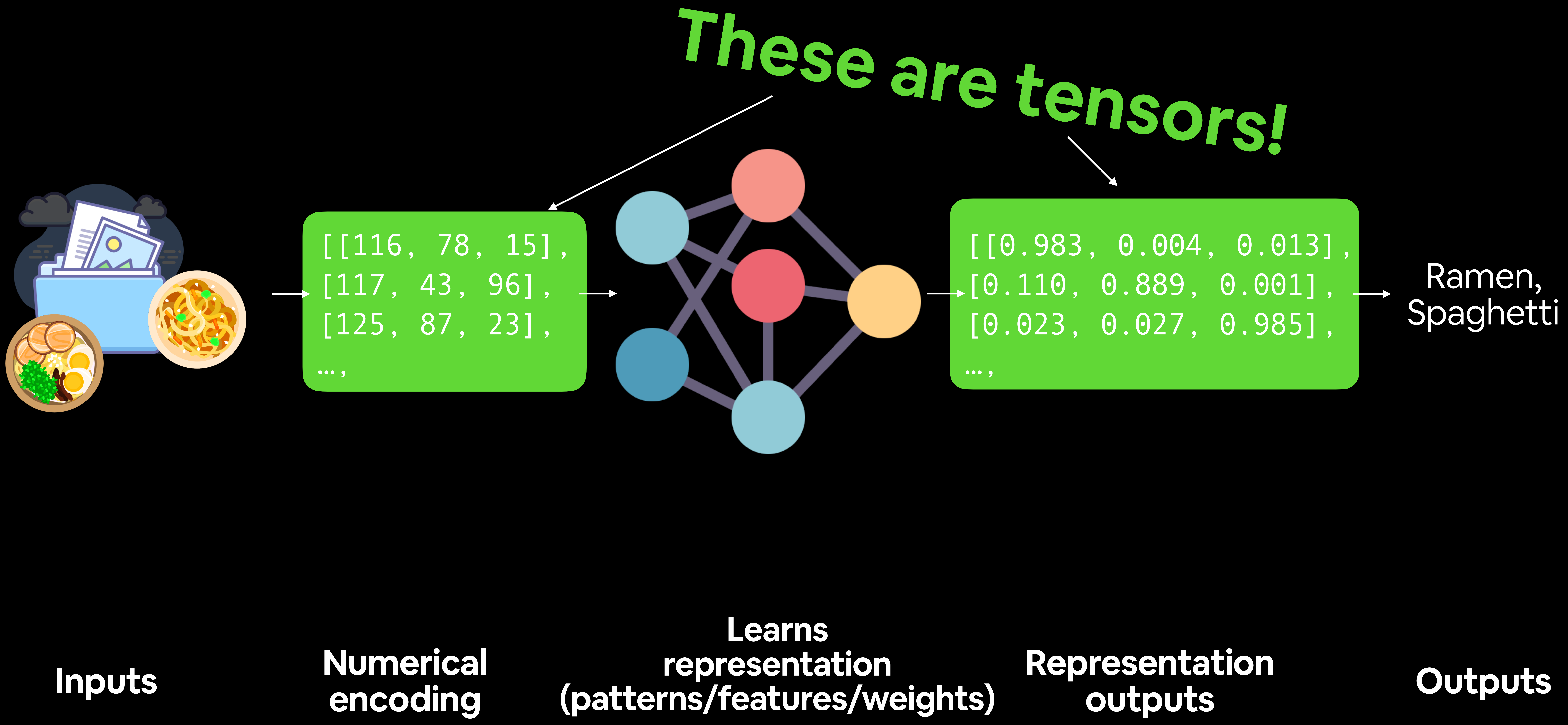
```python
# Create a linear regression model in PyTorch with nn.Linear
class LinearRegressionModel(nn.Module):
    def __init__(self):
        super().__init__()
        # Use nn.Linear() for creating the model parameters
        self.linear_layer = nn.Linear(in_features=1,
                                      out_features=1)

    # forward() defines the computation in the model
    def forward(self, x: torch.Tensor) -> torch.Tensor:
        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],
[0.019, 0.182, 0.555],
...,
2. Show examples
[[116, 78, 15],
[117, 43, 96],
[125, 87, 23],
...,
[[0.983, 0.004, 0.013],
[0.110, 0.889, 0.001],
[0.023, 0.027, 0.985],
...,
Ramen,
Spaghetti
3. Update representation outputs
4. Repeat with more examples
Inputs
Numerical encoding
Learns representation (patterns/features/weights)
Representation outputs
Outputs

Neural Networks




These are tensors!
[[116, 78, 15],
[117, 43, 96],
[125, 87, 23],
...,
[[0.983, 0.004, 0.013],
[0.110, 0.889, 0.001],
[0.023, 0.027, 0.985],
...,
Ramen,
Spaghetti
Inputs
Numerical encoding
Learns representation (patterns/features/weights)
Representation outputs
Outputs

How to approach this course

```python
# 1. Construct a model class that subclasses nn.Module
class CircleModelV0(nn.Module):
    def __init__(self):
        super().__init__()
        # 2. Create 2 nn.Linear layers
        self.layer_1 = nn.Linear(in_features=2, out_features=5)
        self.layer_2 = nn.Linear(in_features=5, out_features=1)

    # 3. Define a forward method containing the forward pass computation
    def forward(self, x):
        # Pass the data through both layers
        return self.layer_2(self.layer_1(x))

# 4. Create an instance of the model and send it to target device
model_0 = CircleModelV0().to(device)
model_0
```

1. Code along

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

Motto #2: Experiment, experiment, experiment!

2. Explore and experiment

Motto #3: Visualize, visualize, visualize!

3. Visualize what you don't understand

(including the "dumb" ones)

4. Ask questions

5. Do the exercises

6. Share your work

How not to approach this course

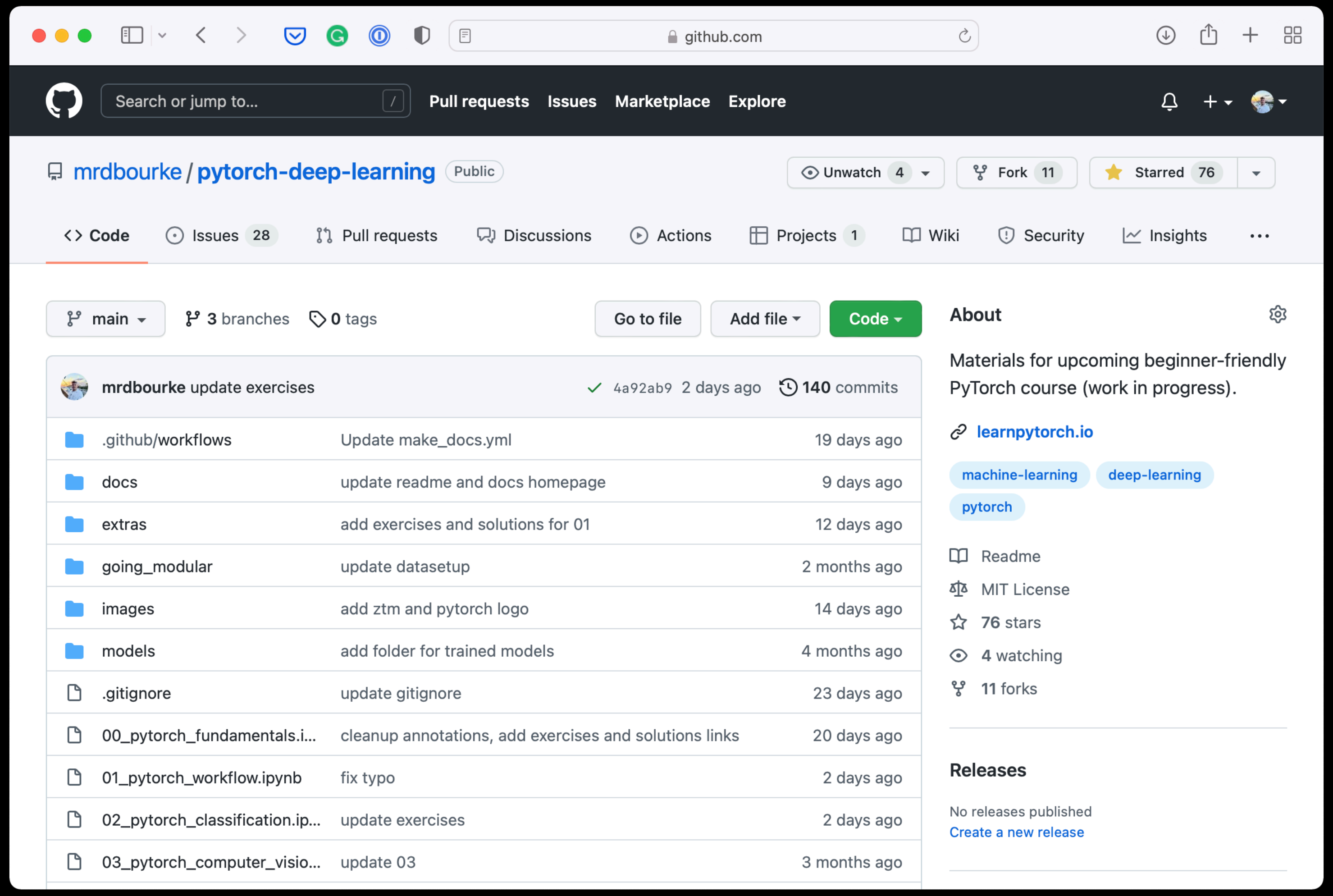
Avoid:

“I ~~can’t~~ learn _______”

Resources

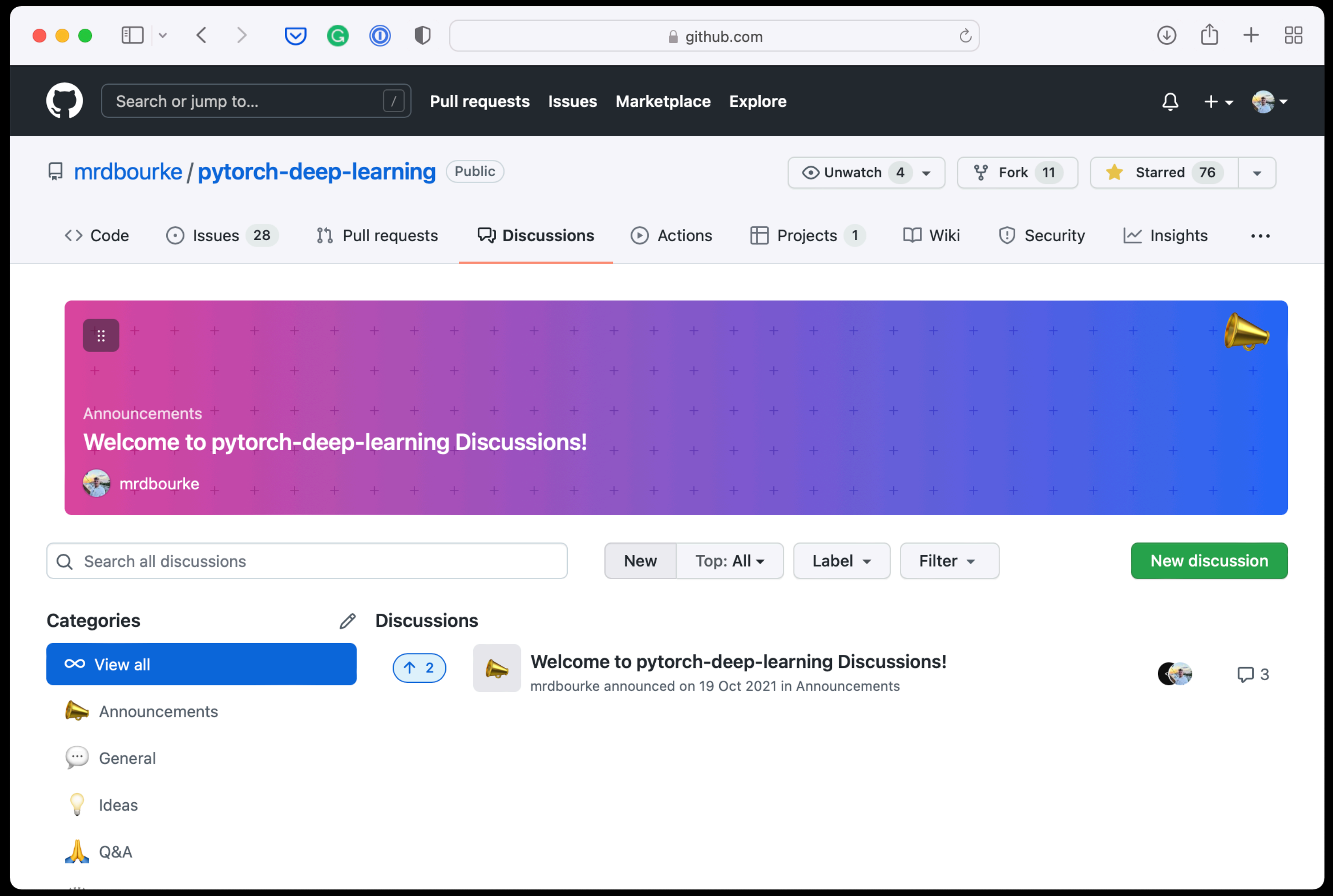
This course

Course materials



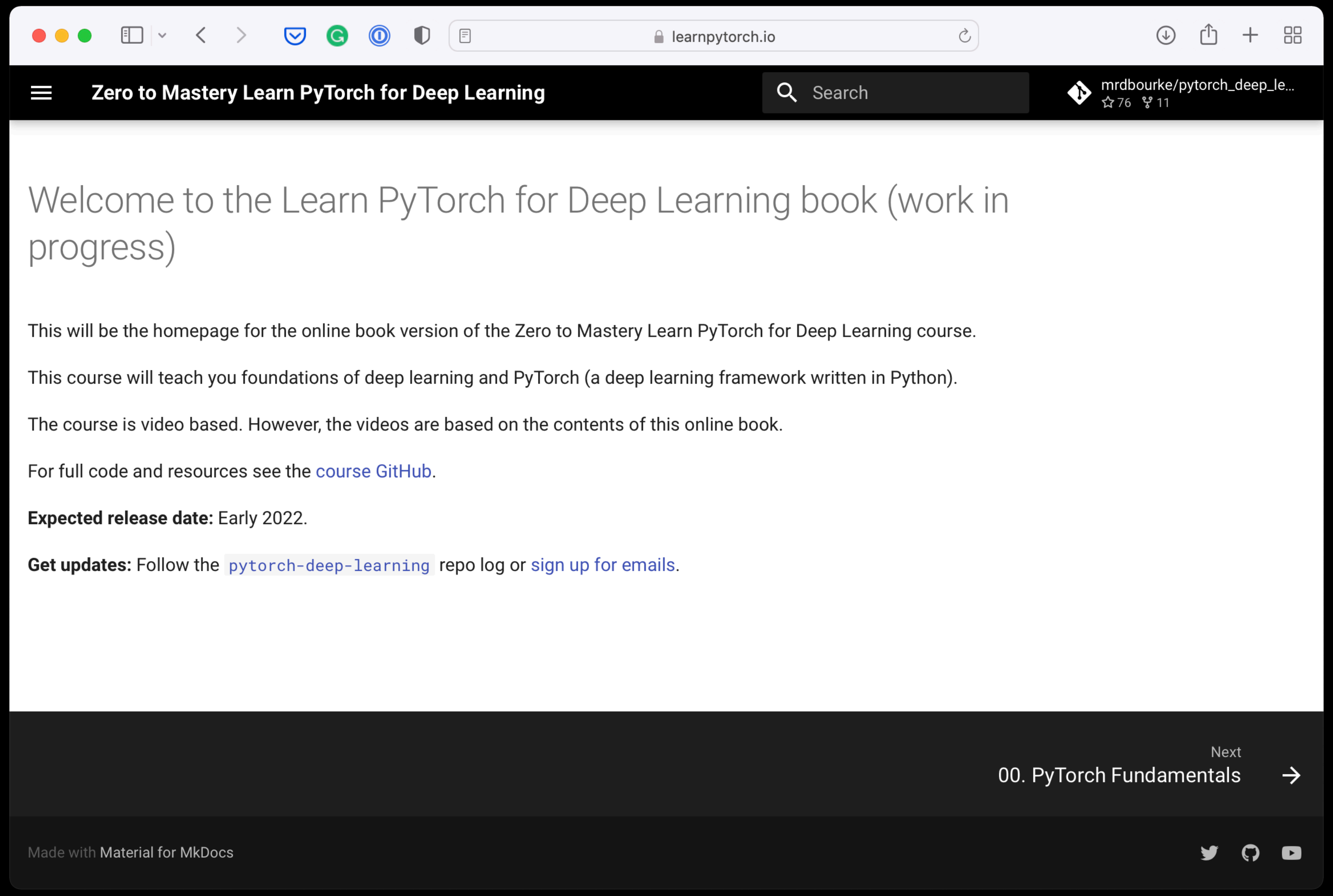
https://www.github.com/mrdbourke/pytorch-deep-learning

Course Q&A



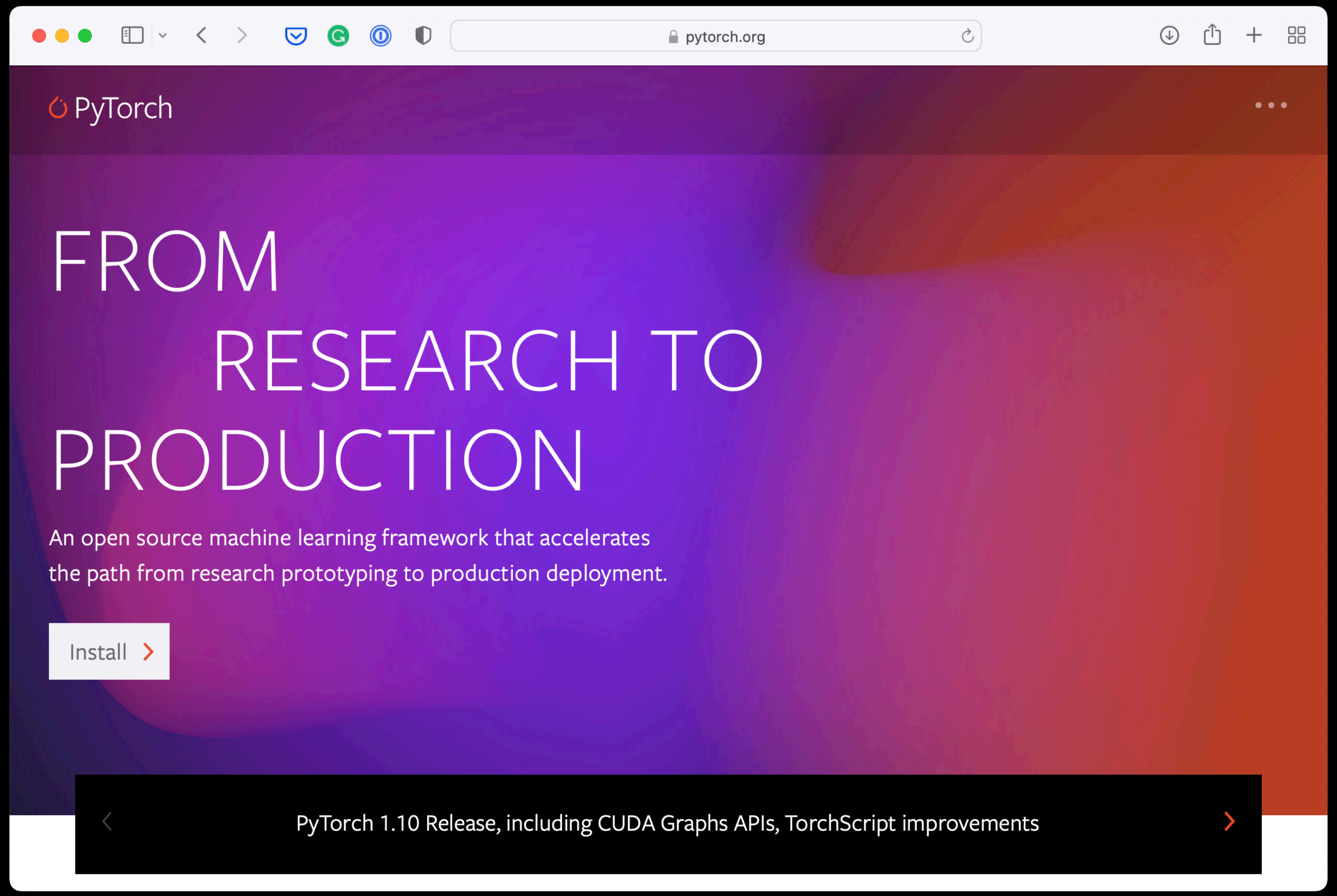
https://www.github.com/mrdbourke/pytorch-deep-learning/discussions

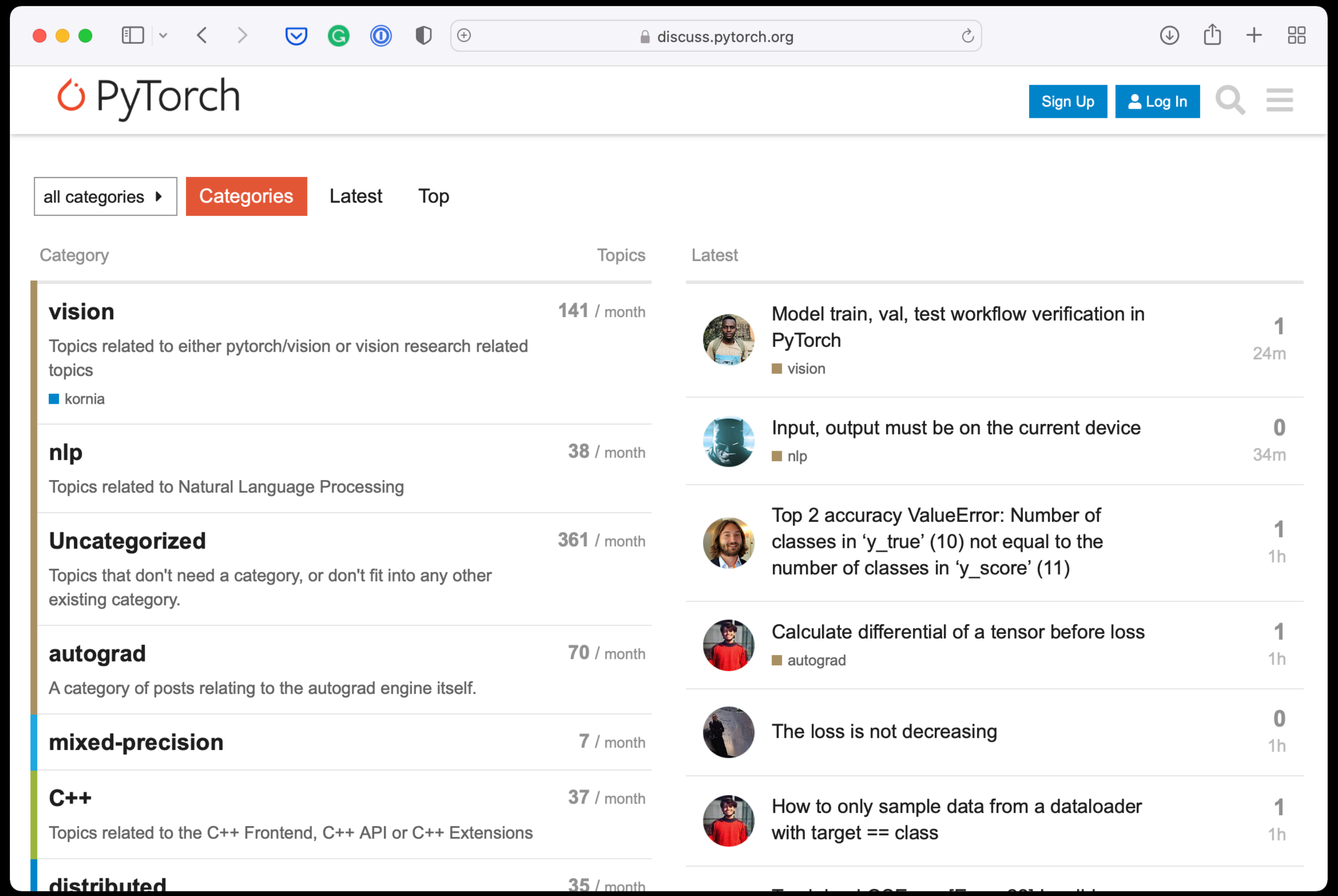
Course online book



https://learnpytorch.io

[PyTorch website](https://pytorch.org) & [forums](https://discuss.pytorch.org)





All things PyTorch

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,
lossssss 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!