https://github.com/mrdbourke/pytorch-deep-learning/tree/main?tab=readme-ov-file

“What is PyTorch?”

https://github.com/mrdbourke/pytorch-deep-learning/tree/main?tab=readme-ov-file

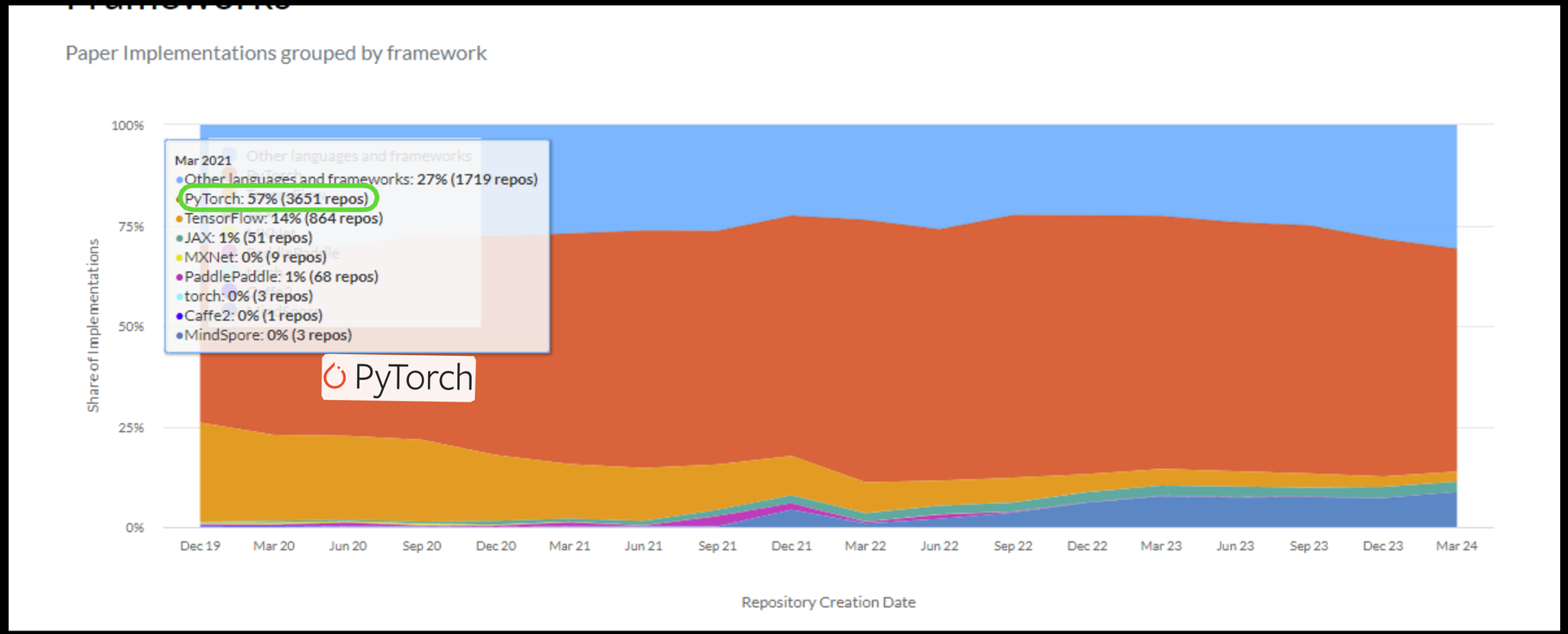
What is PyTorch?

- Most popular **research** deep learning framework*

***Source:** paperswithcode.com/trends February 2024

https://github.com/mrdbourke/pytorch-deep-learning/tree/main?tab=readme-ov-file

Why PyTorch?



Research favorite

Source: paperswithcode.com/trends February 2024

https://github.com/mrdbourke/pytorch-deep-learning/tree/main?tab=readme-ov-file

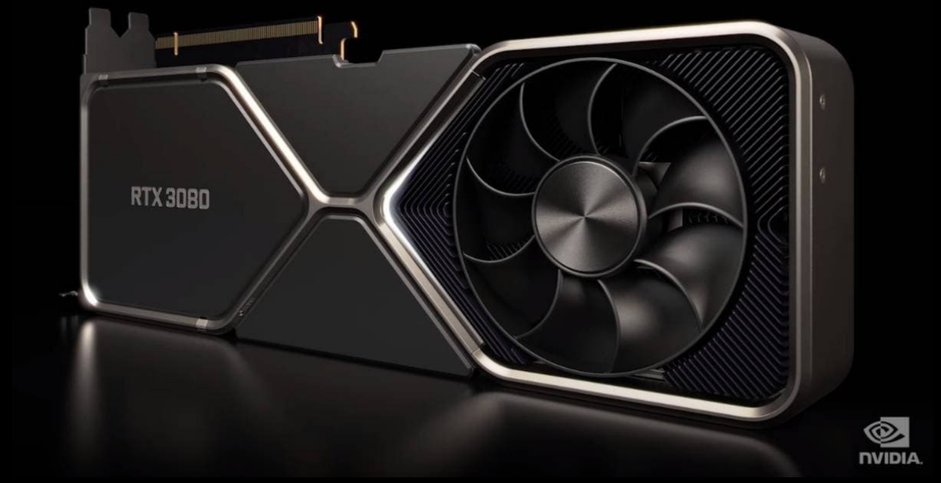
What is PyTorch?

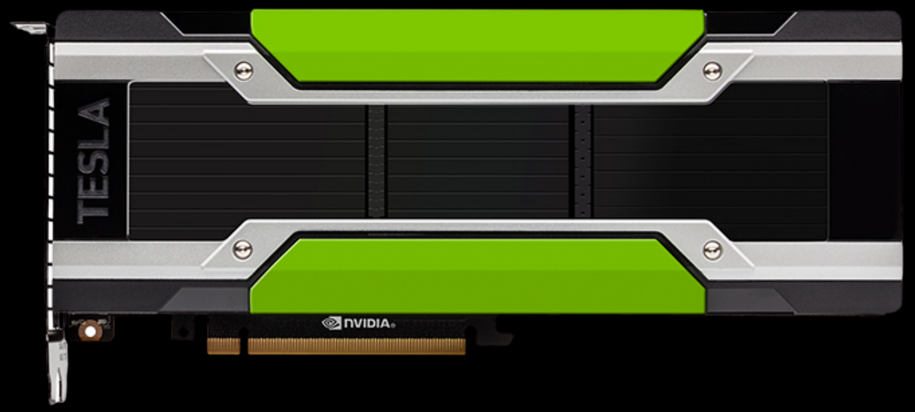
- Most popular **research** deep learning framework*
- Write fast deep learning code in Python (able to run on a GPU/many GPUs)

*Source: paperswithcode.com/trends February 2024

https://github.com/mrdbourke/pytorch-deep-learning/tree/main?tab=readme-ov-file

What is a GPU/TPU?





GPU (Graphics Processing Unit)

TPU (Tensor Processing Unit)

https://github.com/mrdbourke/pytorch-deep-learning/tree/main?tab=readme-ov-file

What is PyTorch?

- Most popular **research** deep learning framework*
- Write fast deep learning code in Python (able to run on a GPU/many GPUs)
- Able to access many pre-built deep learning models (Torch Hub/ `torchvision.models, Hugging Face`)
- Whole stack: preprocess data, model data, deploy model in your application/cloud
- Originally designed and used in-house by Facebook/Meta (now open-source and used by companies such as Tesla, Microsoft, OpenAI)

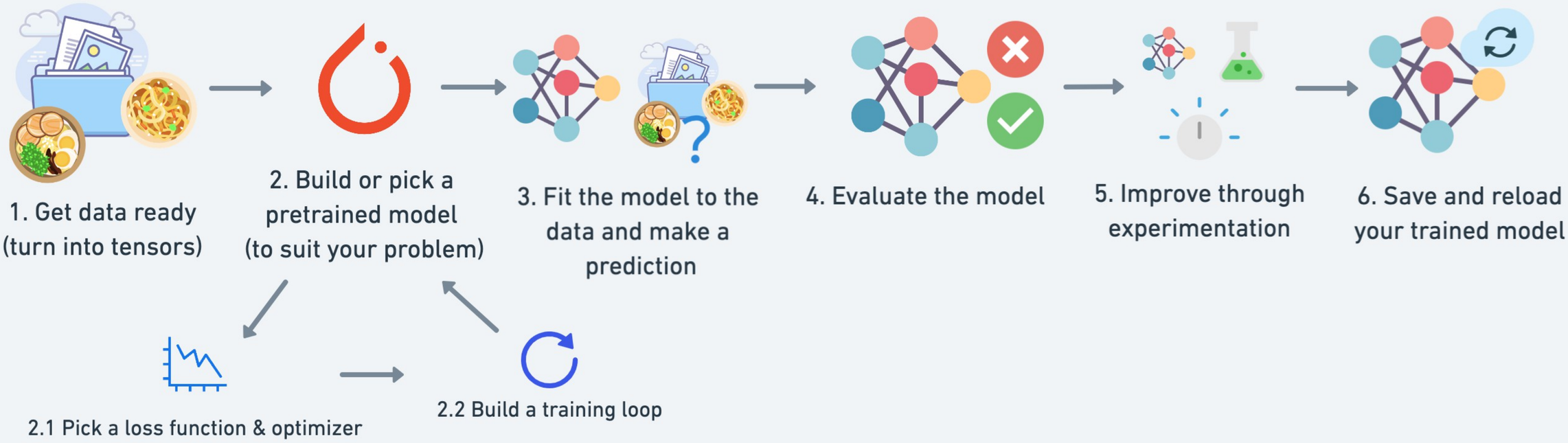
***Source:** paperswithcode.com/trends February 2024

https://github.com/mrdbourke/pytorch-deep-learning/tree/main?tab=readme-ov-file

What we're going to cover

A PyTorch workflow

(one of many)



https://github.com/mrdbourke/pytorch-deep-learning/tree/main?tab=readme-ov-file

“What is a tensor?”

https://github.com/mrdbourke/pytorch-deep-learning/tree/main?tab=readme-ov-file

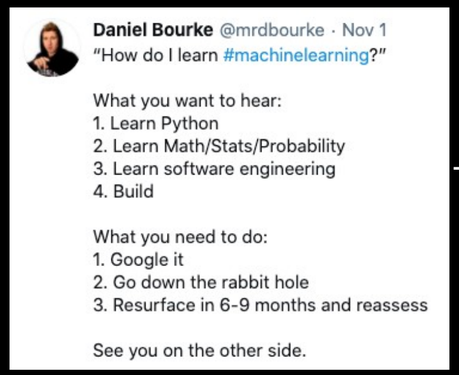
Neural Networks

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

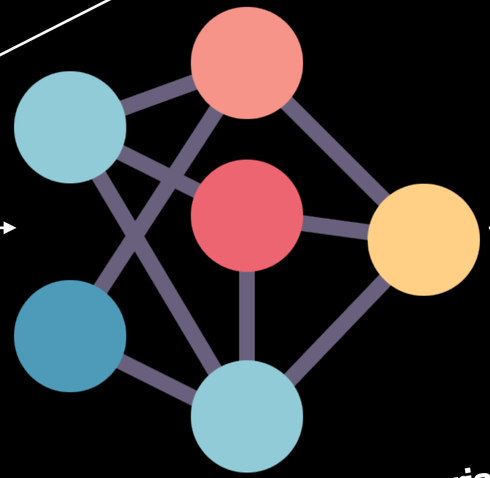
These are tensors!

(a human can understand these)

Ramen, Spaghetti




```
[[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],
...,
```

Not spam

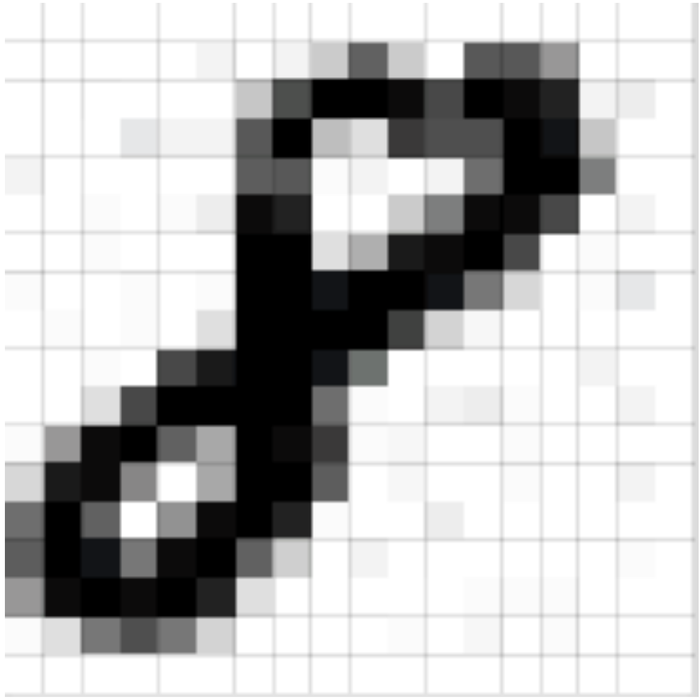
(choose the appropriate neural network for your problem)

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

Inputs | **Numerical encoding** | **Learns representation (patterns/features/weights)** | **Representation outputs** | **Outputs**

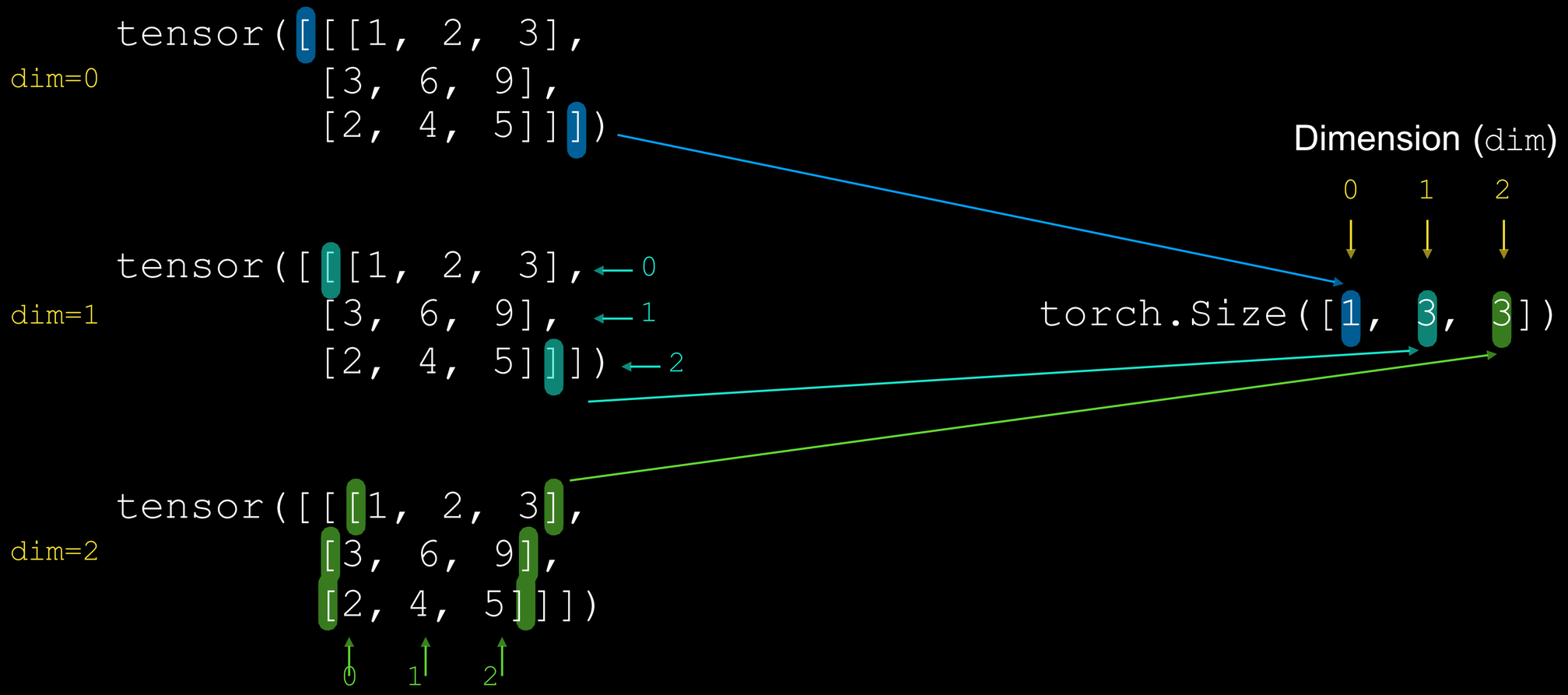
https://github.com/mrdbourke/pytorch-deep-learning/tree/main?tab=readme-ov-file

An Image is a matrix of pixel values



https://www.kdnuggets.com/2016/11/intuitive-explanation-convolutional-neural-networks.html/2

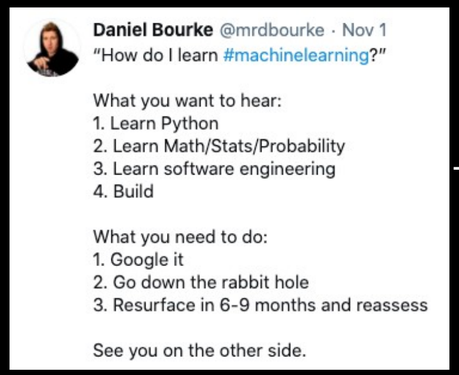
Tensor dimensions



https://github.com/mrdbourke/pytorch-deep-learning/tree/main?tab=readme-ov-file

01_tensors.ipynb

Machine learning: a game of two parts



Inputs

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

Numerical encoding

Learns representation (patterns/features/ weights)

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

Representation outputs

Ramen, Spaghetti

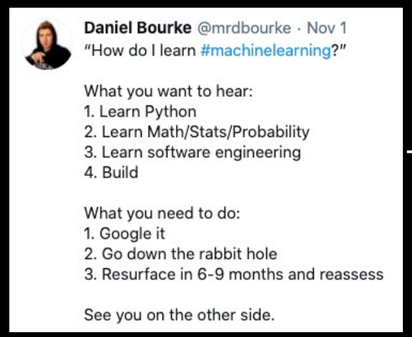
Not spam

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

Outputs

Part 1: Turn data into numbers

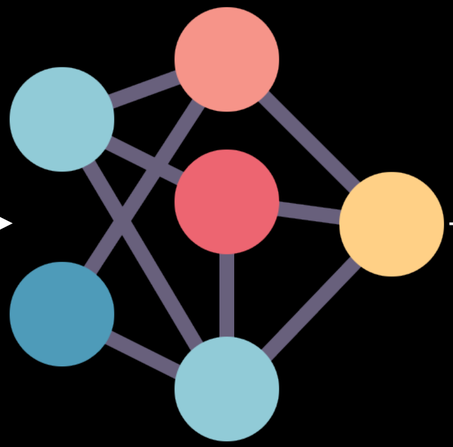
Part 2: Build model to learn patterns in numbers



Inputs

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

Numerical encoding



Learns representation (patterns/features/weights)

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

Representation outputs

Ramen, Spaghetti

Not spam

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

Outputs

Three datasets

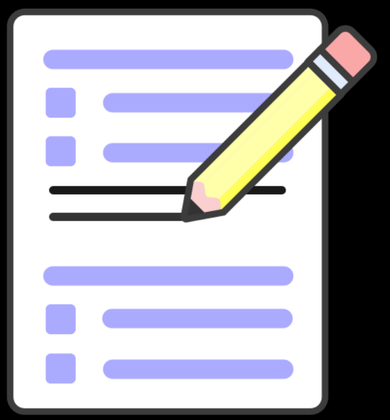
(possibly the most important concept in machine learning…)

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.

https://github.com/mrdbourke/pytorch-deep-learning/tree/main?tab=readme-ov-file

02_dataset_and_dataloader.ipynb

03_transforms.ipynb

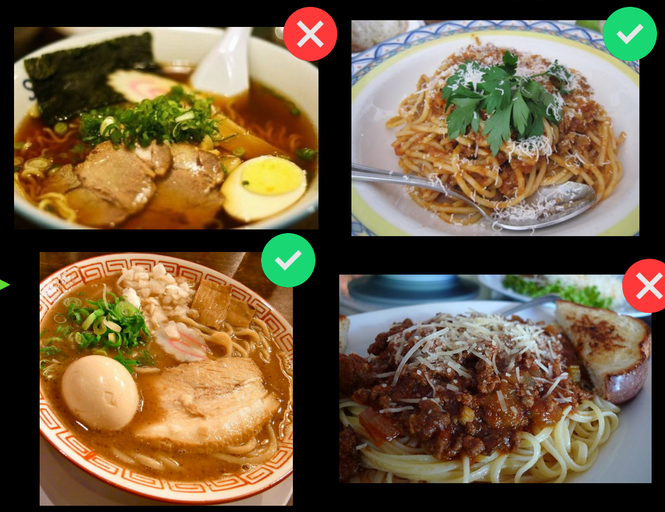
Supervised learning

(overview)

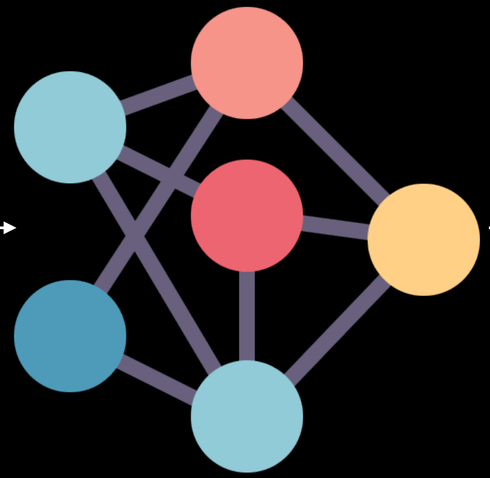
1. Initialise with random weights (only at beginning)

```
[[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],
…,
```



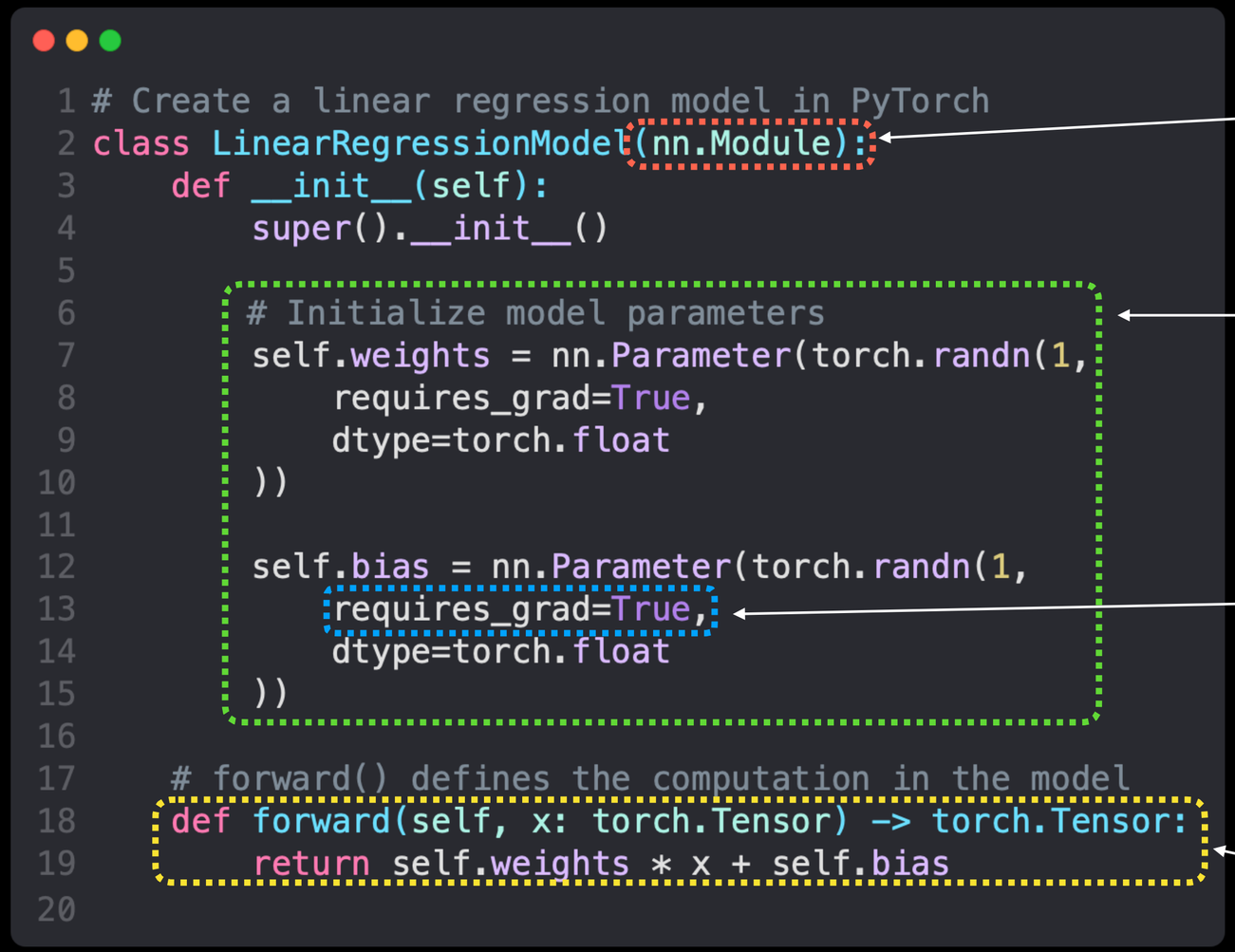
```
[[0.983, 0.004, 0.013],
[0.110, 0.889, 0.001],
[0.023, 0.027, 0.985],
…,
```

Ramen, Spaghetti

3. Update representation outputs

4. Repeat with more examples

Inputs	Numerical encoding	Learns representation (patterns/features/weights)	Representation outputs	Outputs



```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 diMerent layers from `torch.nn`, single parameters, hard-coded values or functions)

`requires_grad=True` means PyTorch will track the gradients of this speciLc 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)

https://github.com/mrdbourke/pytorch-deep-learning/tree/main?tab=readme-ov-file

```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`

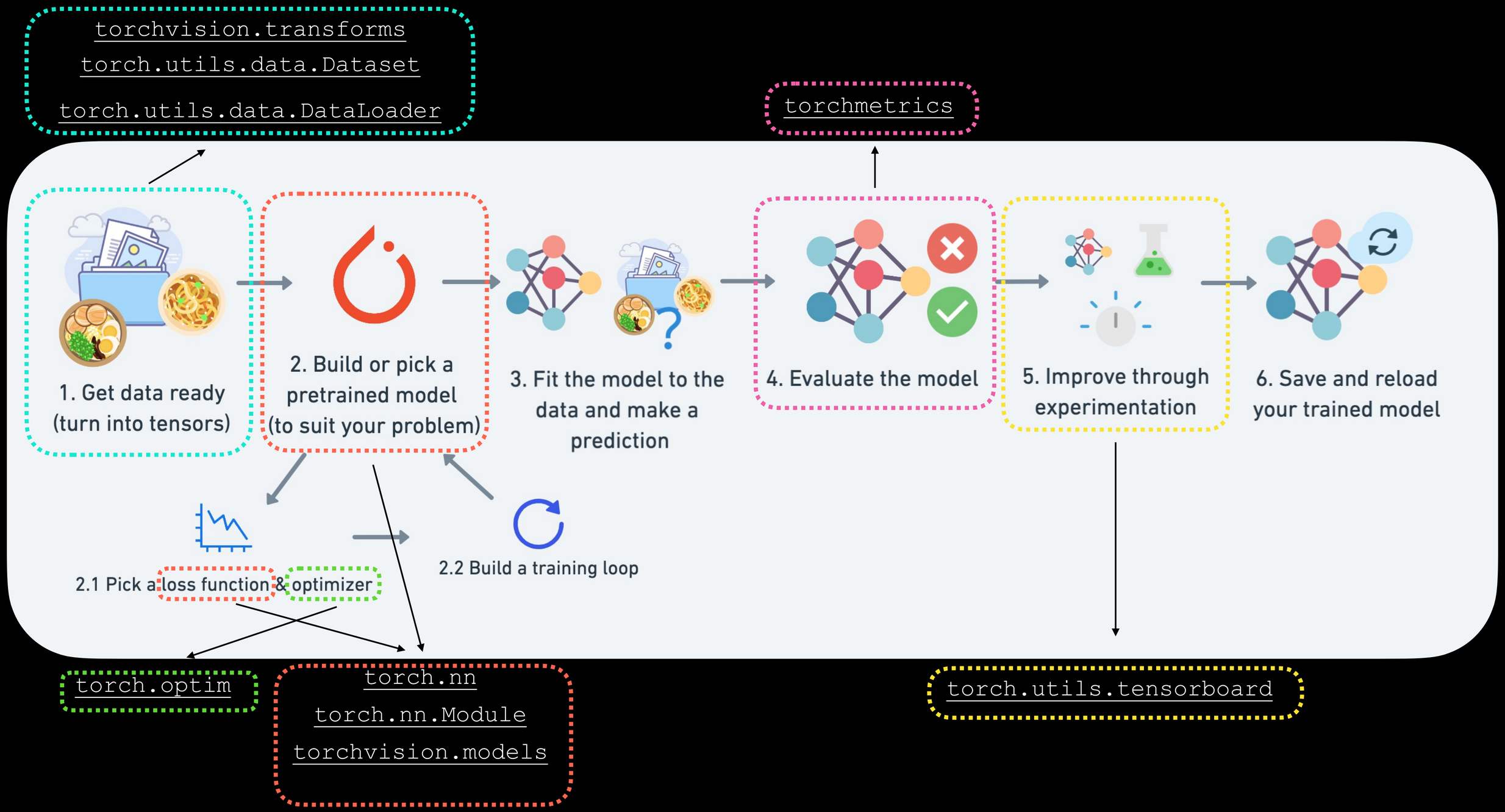
https://github.com/mrdbourke/pytorch-deep-learning/tree/main?tab=readme-ov-file

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

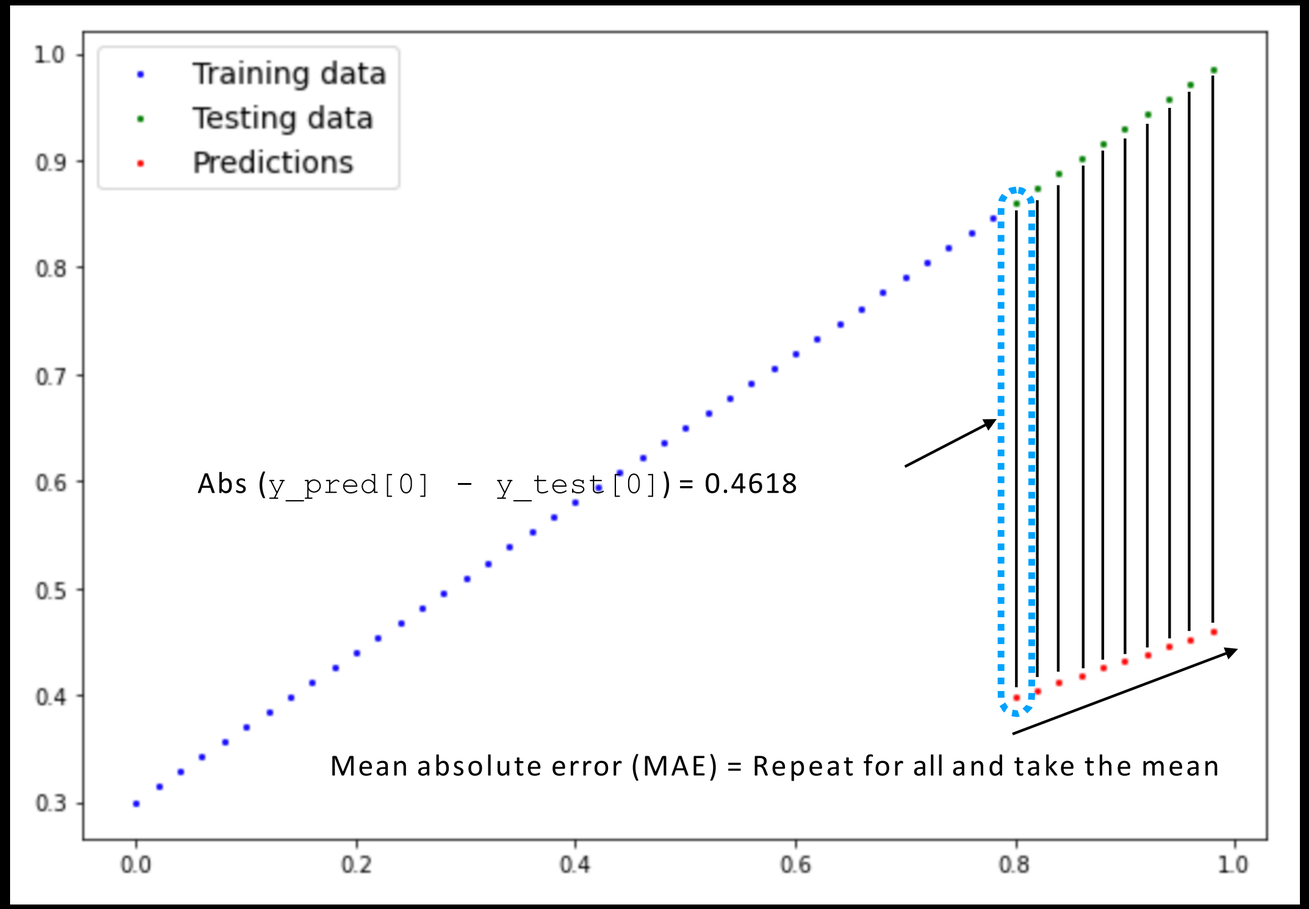
https://github.com/mrdbourke/pytorch-deep-learning/tree/main?tab=readme-ov-file



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

https://github.com/mrdbourke/pytorch-deep-learning/tree/main?tab=readme-ov-file

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

https://github.com/mrdbourke/pytorch-deep-learning/tree/main?tab=readme-ov-file

Binary Cross Entropy (torch.nn.BCELoss)

https://towardsdatascience.com/cross-entropy-negative-log-likelihood-and-all-that-jazz-47a95bd2e81

Cross Entropy Loss (torch.nn.CrossEntropyLoss)

https://towardsdatascience.com/cross-entropy-negative-log-likelihood-and-all-that-jazz-47a95bd2e81

04_building_the_neural_network.ipynb
05_autograd.ipynb

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()`

https://github.com/mrdbourke/pytorch-deep-learning/tree/main?tab=readme-ov-file

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 oM functionality used for training but not evaluation)

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

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 speciLc 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
```

https://github.com/mrdbourke/pytorch-deep-learning/tree/main?tab=readme-ov-file

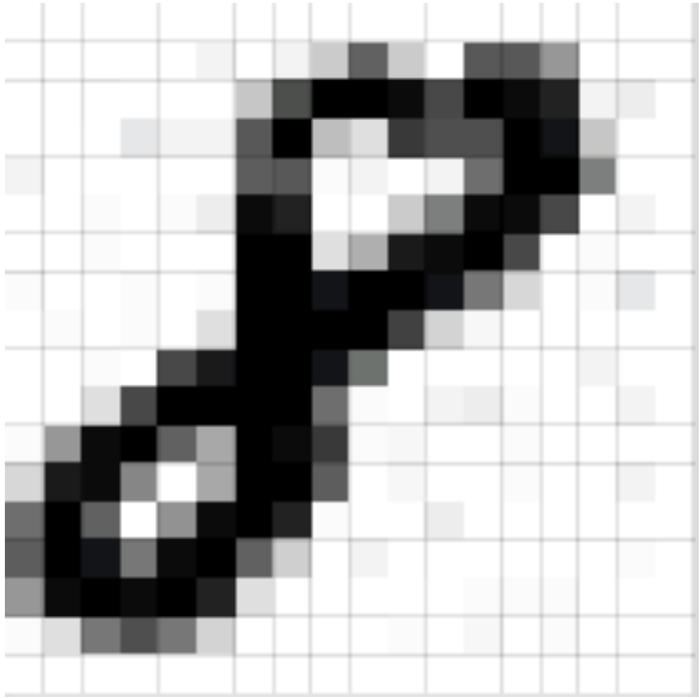
06_optimization.ipynb
07_saving_and_loading_models.ipynb

Questions

MNIST Example

https://github.com/pytorch/examples/blob/main/mnist/main.py

An Image is a matrix of pixel values

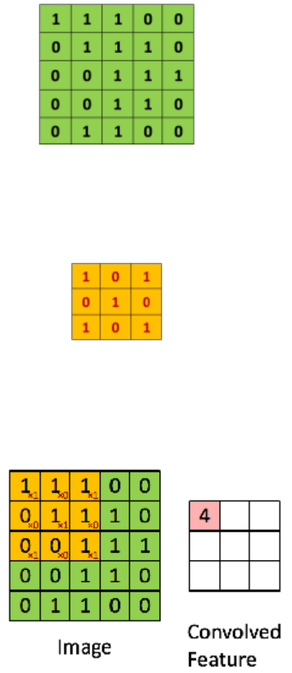


https://www.kdnuggets.com/2016/11/intuitive-explanation-convolutional-neural-networks.html/2

Convolution

We slide the orange matrix over our original image (green) by 1 pixel (also called ‘stride’) and for every position, we compute element wise multiplication (between the two matrices) and add the multiplication outputs to get the final integer which forms a single element of the output matrix (pink). Note that the 3×3 matrix “sees” only a part of the input image in each stride.

The primary purpose of Convolution in case of a ConvNet is to extract features from the input image.

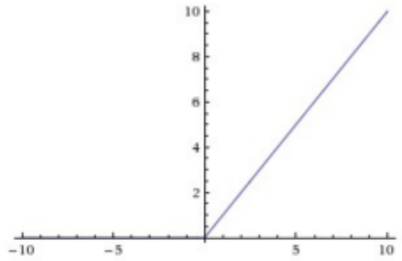


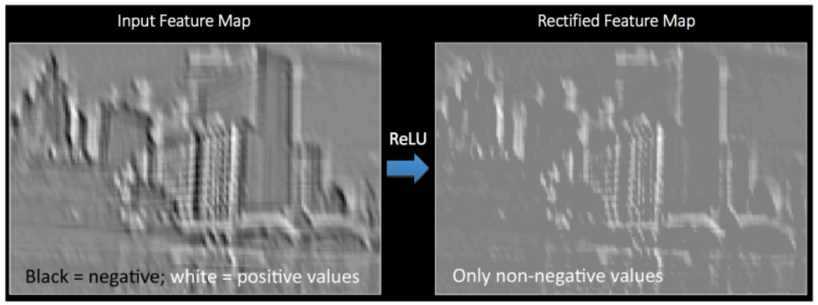
https://www.kdnuggets.com/2016/11/intuitive-explanation-convolutional-neural-networks.html/2

Introducing Non Linearity (ReLU)

ReLU stands for Rectified Linear Unit and is a non-linear operation. Its output is given by:

$$\text{Output} = \text{Max}(\text{zero}, \text{Input})$$





ReLU is an element wise operation (applied per pixel) and replaces all negative pixel values in the feature map by zero. The purpose of ReLU is to introduce non-linearity in our ConvNet, since most of the real-world data we would want our ConvNet to learn would be non-linear.

https://www.kdnuggets.com/2016/11/intuitive-explanation-convolutional-neural-networks.html/2

Convolution

Different filters can give use different feature detectors.
Not that convolution preserves spatial structure.

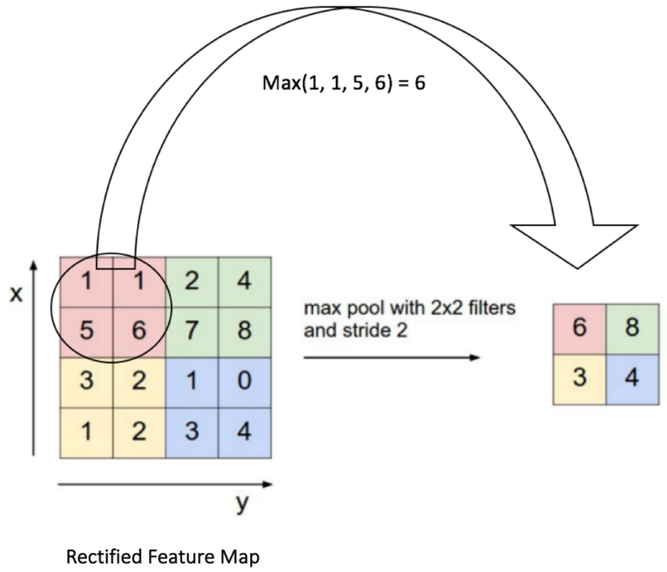
Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

https://www.kdnuggets.com/2016/11/intuitive-explanation-convolutional-neural-networks.html/2

Pooling

Spatial Pooling (also called subsampling or downsampling) reduces the dimensionality of each feature map but retains the most important information. Spatial Pooling can be of different types: Max, Average, Sum etc.

Example: 2D Max Pooling with kernel size of 2. We slide our 2 x 2 window by 2 cells (also called ‘stride’) and take the maximum value in each region.



https://www.kdnuggets.com/2016/11/intuitive-explanation-convolutional-neural-networks.html/2