Introduction to deep learning with PyTorch

Language translation, self-driving cars, medical diagnosis, and chat-bots

PyTorch is intuitive and similar to NumPy

```
import torch
my_list = [[1,2,3], [4,5,6]]
tensor = torch.tensor(my_list)
```

A tensor is similar to an array or matrix, and is the building block of neural networks

This code converts the data to a compatible format

Tensors can also be added/subtracted, only if they have the same exact shape

We can also perform element-wise multiplication (a * b), and matrix multiplication (a @ b)

Neural networks and layers

input neurons = features
output neurons = classes (to predict)

```
import torch.nn as nn
input_tensor = torch.tensor([[0.34, 0.45, -0.23]])
linear_layer = nn.Linear(in_features=3, out_features=2)
output = linear_layer(input_tensor)

# we can then print layer.weight and/or layer.bias parameters
```

Hidden layers and parameters

```
# three sequential linear layers
model = nn.Sequential(
    nn.Linear(n_features, 8),
    nn.Linear(8,4),
    nn.Linear(4, n_classes)
) # layers within nn.Sequential() are hidden layers
```

more hidden layers = more parameters = higher model capacity = longer training and more expensive computation

```
# .numel() returns the number of elements in the tensor
for parameter in model.parameters():
    total += parameter.numel()
print(total) # ex: 46
```

To manually calculate the number of parameters:

weights = out_features x in_features

biases = out_features

total = weights + biases