**Creating tensors in PyTorch**

# Import torch

import torch

# Create random tensor of size 3 by 3

your\_first\_tensor = torch.rand(3, 3)

# Calculate the shape of the tensor

tensor\_size = your\_first\_tensor.shape

# Print the values of the tensor and its shape

print(your\_first\_tensor)

print(tensor\_size)

# Matrix multiplication

# Create a matrix of ones with shape 3 by 3

tensor\_of\_ones = torch.ones(3, 3)

# Create an identity matrix with shape 3 by 3

identity\_tensor = torch.eye(3)

# Do a matrix multiplication of tensor\_of\_ones with identity\_tensor

matrices\_multiplied = torch.matmul(tensor\_of\_ones, identity\_tensor)

print(matrices\_multiplied)

# Do an element-wise multiplication of tensor\_of\_ones with identity\_tensor

element\_multiplication = tensor\_of\_ones \* identity\_tensor

print(element\_multiplication)

# Forward pass

# Initialize tensors x, y and z

x = torch.rand(1000, 1000)

y = torch.rand(1000, 1000)

z = torch.rand(1000, 1000)

# Multiply x with y

q = torch.matmul(x, y)

# Multiply elementwise z with q

f = z \* q

mean\_f = torch.mean(f)

print(mean\_f)

# Backpropagation using PyTorch

# Initialize x, y and z to values 4, -3 and 5

x = torch.tensor(4., requires\_grad=True)

y = torch.tensor(-3., requires\_grad=True)

z = torch.tensor(5., requires\_grad=True)

# Set q to sum of x and y, set f to product of q with z

q = x + y

f = q \* z

# Compute the derivatives

f.backward()

# Print the gradients

print("Gradient of x is: " + str(x.grad))

print("Gradient of y is: " + str(y.grad))

print("Gradient of z is: " + str(z.grad))

# Calculating gradients in PyTorch

# Multiply tensors x and y

q = torch.matmul(x, y)

# Elementwise multiply tensors z with q

f = z \* q

mean\_f = torch.mean(f)

# Calculate the gradients

mean\_f.backward()

# Your first neural network

# Initialize the weights of the neural network

weight\_1 = torch.rand(784, 200)

weight\_2 = torch.rand(200, 10)

# Multiply input\_layer with weight\_1

hidden\_1 = torch.matmul(input\_layer, weight\_1)

# Multiply hidden\_1 with weight\_2

output\_layer = torch.matmul(hidden\_1, weight\_2)

print(output\_layer)

# Your first PyTorch neural network

class Net(nn.Module):

def \_\_init\_\_(self):

super(Net, self).\_\_init\_\_()

# Instantiate all 2 linear layers

self.fc1 = nn.Linear(784, 200)

self.fc2 = nn.Linear(200, 10)

def forward(self, x):

# Use the instantiated layers and return x

x = self.fc1(x)

x = self.fc2(x)

return x