**Creating tensors in PyTorch**

# Import torch

\_\_\_\_

# Create random tensor of size 3 by 3

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

# Calculate the shape of the tensor

tensor\_size = your\_first\_tensor.\_\_\_\_

# Print the values of the tensor and its shape

print(\_\_\_\_)

print(\_\_\_\_)

# Matrix multiplication

# Create a matrix of ones with shape 3 by 3

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

# Create an identity matrix with shape 3 by 3

identity\_tensor = torch.\_\_\_\_(3)

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

matrices\_multiplied = torch.\_\_\_\_(tensor\_of\_ones, \_\_\_\_)

print(matrices\_multiplied)

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

element\_multiplication = \_\_\_\_

print(element\_multiplication)

# Forward pass

# Initialize tensors x, y and z

x = torch.rand(\_\_\_\_, \_\_\_\_)

y = \_\_\_\_

z = \_\_\_\_

# Multiply x with y

q = \_\_\_\_

# Multiply elementwise z with q

f = \_\_\_\_

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., \_\_\_\_)

y = torch.tensor(\_\_\_\_., \_\_\_\_)

z = \_\_\_\_

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

q = \_\_\_\_

f = \_\_\_\_

# Compute the derivatives

f.\_\_\_\_

# Print the gradients

print("Gradient of x is: " + str(\_\_\_\_))

print("Gradient of y is: " + str(\_\_\_\_))

print("Gradient of z is: " + str(\_\_\_\_))

# Calculating gradients in PyTorch

# Multiply tensors x and y

q = \_\_\_\_

# Elementwise multiply tensors z with q

f = \_\_\_\_

mean\_f = torch.mean(f)

# Calculate the gradients

\_\_\_\_

# Your first neural network

# Initialize the weights of the neural network

weight\_1 = torch.rand(\_\_\_\_, \_\_\_\_)

weight\_2 = \_\_\_\_

# Multiply input\_layer with weight\_1

hidden\_1 = torch.matmul(\_\_\_\_, \_\_\_\_)

# Multiply hidden\_1 with weight\_2

output\_layer = \_\_\_\_

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(\_\_\_\_, \_\_\_\_)

self.fc2 = \_\_\_\_

def forward(self, x):

# Use the instantiated layers and return x

x = self.fc1(x)

x = \_\_\_\_

return \_\_\_\_