INTRODUCTION TO PYTORCH, A DEEP LEARNING LIBRARY

datalab

Exercise 1

Getting started with PyTorch tensors

Tensors are PyTorch's core data structure and the foundation of deep learning. They're similar to NumPy arrays but have unique features.

Here you have a Python list named temperatures containing daily readings from two weather stations. Try converting this into a tensor!

Instructions

- Begin by importing torch.
- Create a tensor from the Python list temperatures.

```python
# Import PyTorch
import torch

temperatures = [[72, 75, 78], [70, 73, 76]]

# Create a tensor from temperatures
temp_tensor = torch.tensor(temperatures)

print(temp_tensor)
```

```
tensor([[72, 75, 78],
        [70, 73, 76]])
```

Exercise 2

Checking and adding tensors

While collecting temperature data, you notice the readings are off by two degrees. Add two degrees to the temperatures tensor after verifying its shape and data type with torch to ensure compatibility with the adjustment tensor.

The torch library and the temperatures tensor are loaded for you.

Instructions 1

- Display the shape of the adjustment tensor.
- Display the data type of the adjustment tensor.

```
adjustment = torch.tensor([[2, 2, 2], [2, 2, 2]])

# Display the shape of the adjustment tensor
print("Adjustment shape:", adjustment.shape)

# Display the type of the adjustment tensor
print("Adjustment type:", adjustment.dtype)

print("Temperatures shape:", temp_tensor.shape)
print("Temperatures type:", temp_tensor.dtype)
```

```
Adjustment shape: torch.Size([2, 3])
Adjustment type: torch.int64
Temperatures shape: torch.Size([2, 3])
Temperatures type: torch.int64
```

Instructions 2

- Add the `temperatures` and `adjustment` tensors.

```
adjustment = torch.tensor([[2, 2, 2], [2, 2, 2]])

# Add the temperatures and adjustment tensors
corrected_temperatures = temp_tensor + adjustment
print("Corrected temperatures:", corrected_temperatures)
```

```
Corrected temperatures: tensor([[74, 77, 80],
        [72, 75, 78]])
```

Exercise 3

Linear layer network

Neural networks often contain many layers, but most of them are linear layers. Understanding a single linear layer helps you grasp how they work before adding complexity.

Apply a `linear layer` to an input tensor and observe the output.

Instructions

- Create a Linear layer that takes 3 features as input and returns 2 outputs.
- Pass input_tensor through the linear layer.

```
#Import neural network from rorch library
#import torch library when use in the exercise
import torch.nn as nn

input_tensor = torch.tensor([[0.3471, 0.4547, -0.2356]])

# Create a Linear layer
linear_layer = nn.Linear(
                         in_features=3,
                         out_features=2
                         )

# Pass input_tensor through the linear layer
output = linear_layer(input_tensor)

print(output)
```

```
tensor([[-0.4160,  0.6729]], grad_fn=<AddmmBackward0>)
```

Exercise 4

Exercise 5

Your first neural network

It's time for you to implement a small neural network containing two linear layers in sequence.

Instructions

- Add a container for stacking layers in sequence.

```
# Import libraries when use in the exercise

input_tensor = torch.Tensor([[2, 3, 6, 7, 9, 3, 2, 1]])

# Create a container for stacking linear layers
model = nn.Sequential(nn.Linear(8, 4),
                nn.Linear(4, 1)
                )

output = model(input_tensor)
print(output)
```

```
tensor([[0.9745]], grad_fn=<AddmmBackward0>)
```

Exercise 5

Exercise 6

Counting the number of parameters

Deep learning models are famous for having a lot of parameters. With more parameters comes more computational complexity and longer training times, and a deep learning practitioner must know how many parameters their model has.

In this exercise, you'll first calculate the number of parameters manually. Then, you'll verify your result using the `.numel()` method.

Instructions 1/2

Question Manually calculate the number of parameters of the model below. How many does it have? Use the console as a calculator.

```
        model = nn.Sequential(nn.Linear(9, 4),
                              nn.Linear(4, 2),
                              nn.Linear(2, 1))
```

Possible answers

1. 53 (correct)

2. 16

3. 29

Instructions 2/2

- Use `.numel()` to confirm your manual calculation by iterating through the model's parameters to updating the total variable.

```python
import torch.nn as nn

model = nn.Sequential(nn.Linear(9, 4),
                      nn.Linear(4, 2),
                      nn.Linear(2, 1))

total = 0

# Calculate the number of parameters in the model
for p in model.parameters():
  total += p.numel()

print(f"The number of parameters in the model is {total}")
```

```
The number of parameters in the model is 53
```