

Welcome to Pytorch Chat-bot App

Ask Pytorch related questions here!

How to create a simple neural network using Pytorch?

Based on the provided context, I will answer your question about creating a simple neural network using PyTorch.

To define a simple fully connected neural network in PyTorch, we can use the following code:

```
import torch
import torch.nn as nn

# Define the model architecture
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(784, 128) # input layer (28x28 images) -> hidden layer
        self.relu1 = nn.ReLU()
        self.dropout1 = nn.Dropout(p=0.2)
        self.fc2 = nn.Linear(128, 10) # hidden layer (128 units) -> output layer (10 classes)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu1(out)
        out = self.dropout1(out)
        out = self.fc2(out)
        return out

# Initialize the model
model = Net()

# Print the model's architecture
print(model)
```

In this code snippet:

- We first import the necessary modules: `torch` and `torch.nn`.

- We define a class called `Net` that inherits from PyTorch's `nn.Module`. This is our neural network model.
- Inside the `Net` class, we specify three layers:
 - A fully connected (dense) layer (`fc1`) with 784 inputs and 128 outputs. We use this to map input data into a hidden representation.
 - An activation function (`relu1`) using the ReLU (Rectified Linear Unit) activation function for non-linearity.
 - A dropout layer (`dropout1`) with a dropout rate of 0.2 to prevent overfitting by randomly setting some units to zero during training.
 - Another fully connected layer (`fc2`) with 128 inputs and 10 outputs, mapping the hidden representation back into the desired output space.
- In the `forward` method of our model, we define how data flows through our network:
 - Input data passes through the first dense layer (`fc1`).
 - The output is then passed through an activation function (`relu1`) to introduce non-linearity.
 - We apply dropout (`dropout1`) with a 20% chance of dropping out units.
 - Finally, we pass the output through another dense layer (`fc2`) for final output computation.
- We initialize our model by creating an instance of `Net`.
- To print our model's architecture, we simply call the `print` function on it.

This simple example demonstrates how to build a fully connected neural network in PyTorch. You can experiment with this basic structure and modify it as needed for your specific use case.