Explanation of ParentCNN Class (for MNIST data)

This document provides a detailed explanation for every line under the # 1. Parent CNN for mnist data section in your code.

python

class ParentCNN(nn.Module):

Defines a new neural network class called ParentCNN that inherits from PyTorch's nn.Module, which is the base class for all neural network modules in PyTorch.

python

    def __init__(self):

The constructor method for the class. This is called when you create a new instance of ParentCNN.

python

        super(ParentCNN, self).__init__()

Calls the constructor of the parent class (nn.Module) to properly initialize the module.

python

        self.conv1 = nn.Con2d(1, 16, kernel_size=3, paddings=1)

Defines the first convolutional layer. Note: There are typos here. It should be nn.Conv2d (not

Con2d) and padding=1 (not paddings=1). This layer takes 1 input channel (grayscale image), outputs 16 channels, uses a 3x3 kernel, and applies padding of 1.

python

```
self.conv2 = nn.Conv2d(16, 32, kernel_size=3, paddings=1)
```

Defines the second convolutional layer. Note: Same typo as above; should be padding=1. This layer takes 16 input channels, outputs 32 channels, uses a 3x3 kernel, and applies padding of 1.

python

```
self.pool = nn.MaxPool2d(2, 2)
```

Defines a max pooling layer with a 2x2 window and stride of 2, which reduces the spatial dimensions of the feature maps.

python

```
self.fc1 = nn.Linear(32 * 7 * 7, 128)
```

Defines the first fully connected (linear) layer. It takes the flattened output from the convolutional layers (assuming input images are 28x28, after two 2x2 poolings, the size is 7x7) and outputs 128 features.

python

```
self.fc2 = nn.Linear(128, 10) #10 classes for Mnist data
```

Defines the second fully connected layer, mapping 128 features to 10 output classes (digits 0-9 for MNIST).

python

```
self.relu = nn.RELU()
```

Defines the ReLU activation function. Note: Typo here; should be nn.ReLU() (not RELU()).

python

```
def foward(self, x):
```

Defines the forward pass method. Note: Typo here; should be forward, not foward. This method specifies how the input data flows through the network.

python

```
x = self.pool(self.relu(self.conv1(x)))
```

Applies the first convolution, then ReLU activation, then max pooling to the input x.

python

```
x = self.pool(self.relu(self.conv2(x)))
```

Applies the second convolution, ReLU, and max pooling.

python

```
x = x.view(-1, 32 * 7 * 7)
```

Flattens the output tensor to a shape suitable for the fully connected layer. -1 infers the batch size.

python

```
x = self.relu(self.fc1(x))
```

Applies the first fully connected layer and then the ReLU activation.

python

```
x = self.fc2(x)
```

Applies the second fully connected layer to produce the final output logits.

python

```
return x
```

Returns the output logits.

python

```
def get_features(self, x):
```

Defines a method to extract features from the network before the final classification layer.

python

```
x = self.pool(self.relu(self.conv1(x)))
x = self.pool(self.relu(self.conv2(x)))
x = x.view(-1, 32 * 7 * 7)
x = self.relu(self.fc1(x))
```

These lines repeat the forward pass up to the first fully connected layer, returning the feature

representation (128-dimensional vector) for the input.

Summary of corrections needed:

- Change nn.Con2d to nn.Conv2d

- Change paddings=1 to padding=1

- Change nn.RELU() to nn.ReLU()

- Change def foward to def forward