Part 2, Lab #2: Basic Knowledge

Ke Xu 3190110360

Due April 6th, 2023 11:59 PM CST

Logistics and Lab Submission

See the **BlackBoard**.

What You Will Need To Know For This Lab

This lab covers:

- Design the dataset.
- Design the convolutional neural network for the image classification.
- Training and testing the model.

The submission procedure is provided below:

- You will be provided with a Jupyter Notebook for this lab where you need to implement the provided functions as needed for each question. Follow the instructions provided in this Jupyter Notebook (.ipynb) to implement the required functions.
- Upload the **PDF** (screen shot) file of your Jupyter Notebook (.ipynb file).
- Your grades and feedbacks will appear on BlackBoard. You will have a chance to re-submit your code, only if you have *reasonable* submissions before the deadline (i.e. not an empty script).

Problem 1 : CNN

1. Design the first convolutional layers. The first convolutional layer has 16 filters of size 3x3, padding=1, stride=1, please print the size of output. Please refer from nn.Conv2d().

In [1]:
```
import torch
import torch.nn as nn
# Input 2 images
images = torch.randn(2, 3, 600, 800) # batch size, channel, height, weight

conv1 = nn.Conv2d(3,16,(3,3),padding=1,stride=1) # input channel, fileters
output1 = conv1(images)
print(output1.shape)
```

```
torch.Size([2, 16, 600, 800])
```

1. Design the second convolutional layers. Please use the **group convolution**. The group number is 4. The second convolutional layer has 32 filters of size 3x3, padding=1, stride=2, please print the size of output.

In [2]:
```
conv2 = nn.Conv2d(16,32,(3,3),padding=1,stride=2,groups=4) # input channel
output2 = conv2(output1)
print(output2.shape)
```

```
torch.Size([2, 32, 300, 400])
```

1. Design the third convolutional layers. Please use the **dilation**. The third convolutional layer has 16 filters of size 3x3, padding=1, stride=1, dilation=(4,2), please print the size of output.

In [3]:
```
conv3 = nn.Conv2d(32,16,(3,3),padding=1,stride=1,dilation=(4,2)) # input ch
output3 = conv3(output2)
print(output3.shape)
```

```
torch.Size([2, 16, 294, 398])
```

Problem 2 : RNN

1. Create RNN with input size 10, hidden size 20, please print the size of output. Please refer from nn.RNN().

In [4]:
```
# Create input sequence of length 3, with batch size 1
x = torch.randn(3, 1, 10)

# Initialize hidden state with batch size 1
hidden = torch.zeros(1, 1, 20)

# Initialized RNN
rnn = nn.RNN(10,20) # input size, hidden size
output, hidden = rnn(x, hidden)
print(output.shape)
print(hidden.shape)
```

```
torch.Size([3, 1, 20])
torch.Size([1, 1, 20])
```

1. Initialize hidden state with batch size 1, hidden size is 20, please use torch.zeros().

In [5]:
```
hidden = torch.zeros(1,1,20) # input size, batch size, hidden size
print(hidden.shape)
```

```
torch.Size([1, 1, 20])
```

1. Design a fully connected layer to output the final result, the output size is 5. Please refer from nn.Linear().

In [6]:
```
fc = nn.Linear(20,5) # input size, output size
output = fc(output)
print(output.shape)
```

```
torch.Size([3, 1, 5])
```

Problem 3 : LSTM

1. Create the LSTM model with an input size of 10, a hidden size of 20, and one LSTM layer. Please refer from nn.LSTM()

In [7]:
```
input_size = 10
hidden_size = 20
num_layers = 1
# Create the LSTM model
lstm = nn.LSTM(10,20) # input size, hidden size
print(lstm)

# Define a sample input sequence of length 5
input_seq = torch.randn(5, 1, 10)
```

```
LSTM(10, 20)
```

1. Initialize the hidden state and cell state by using torch.randn(). Print the output shape and final hidden state and cell state.

In [8]:
```
# Initialize the hidden state and cell state, size = (num_layers, batch_si
h0 = torch.randn(num_layers, 1, hidden_size)
c0 = torch.randn(num_layers, 1, hidden_size)

# Pass the input sequence and initial states through the LSTM
output, (hn, cn) = lstm(input_seq, (h0, c0)) # input, (h0,c0)

# Print the output shape and final hidden state and cell state
print('Output shape:', output.shape)
print('Final hidden state:', hn.shape)
print('Final cell state:', cn.shape)
```

```
Output shape: torch.Size([5, 1, 20])
Final hidden state: torch.Size([1, 1, 20])
Final cell state: torch.Size([1, 1, 20])
```

Problem 4 : Transformer

1. Define the input and output embeddings, please refer from nn.Embedding().

In [9]:
```
# Define the input sequence and target sequence
input_seq = torch.tensor([[0, 1, 2, 3]])
tgt_seq = torch.tensor([[2, 3, 1, 0], [1,2,3,0]])

# Define input and output embeddings, embedding size is 128
embedding = nn.Embedding(4,128)  # embedding dim, embedding size

src = embedding(input_seq)
tgt = embedding(tgt_seq)

print(src.shape)
print(tgt.shape)
```

```
torch.Size([1, 4, 128])
torch.Size([2, 4, 128])
```

1. Define encoder layer, num_layer=1, the number of heads in the multiheadattention models is 4, the dimension of the hidden layer is 512. Please refer from nn.TransformerEncoder() and nn.TransformerEncoderLayer().

In [10]:
```
# Define encoder layer
encoder_layer = nn.TransformerEncoderLayer(d_model=128,nhead=4) # dim of i
transformer_encoder = nn.TransformerEncoder(encoder_layer,num_layers=1) # 
encoded = transformer_encoder(src)
print(encoded.shape)
```

```
torch.Size([1, 4, 128])
```

1. Define decoder layer, num_layer=1, the number of heads in the multiheadattention models is 4, the dimension of the hidden layer is 512. Please refer from nn.TransformerDecoder() and nn.TransformerDecoderLayer().

In [11]:
```
# Define encoder layer
decoder_layer = nn.TransformerDecoderLayer(d_model=128,nhead=4) # dim of i
transformer_decoder = nn.TransformerDecoder(decoder_layer,num_layers=1) # 
decoded = transformer_decoder(tgt, encoded)
print(decoded.shape)
```

```
torch.Size([2, 4, 128])
```