CS324 Deep Learning
Assignment 2

Kuang Liang

November 16, 2024

Abstract

For this assignment, I implemented, trained and tested an MLP, a CNN and an RNN based on PyTorch. In the report, I will explain how I implemented them and the experiments I conducted with them.

1 MLP

1.1 NumPy MLP and PyTorch MLP

In Task 1 and Task 2, I tested the NumPy MLP model and the PyTorch MLP model under the same setting. As expected, they achieved the same accuracy on the datasets generated by `sklearn.datasets.make_moons` and `sklearn.datasets.make_circles`, see figs. 1 and 2.

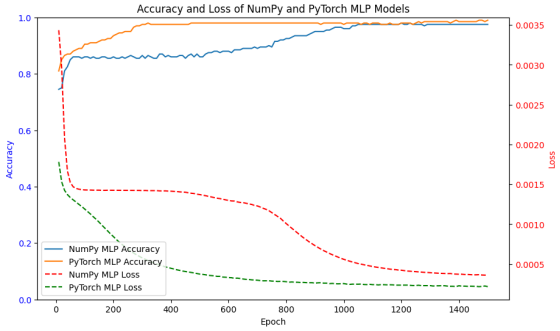

Figure 1: `make_moons` generator

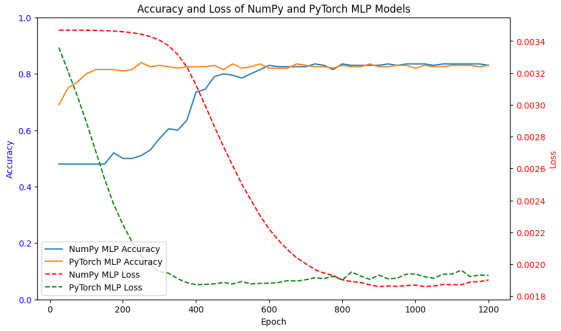

Figure 2: `make_circles` generator

1.2 CIFAR-10 experiment on MLP

In Task 3, I tried to achieve the best result on CIFAR-10 classification task with a self-defined MLPnet.

1.2.1 Data preprocessing

I have used several methods to preprocess the data[1].

1. `transforms.RandomHorizontalFlip()`: This randomly flips each image horizontally to augment the dataset by creating variations of the images.

2. `transforms.RandomCrop(32, padding=4)`: This crops the image to a 32×32 size with a padding of 4 pixels, which adds randomness to the dataset by simulating slight shifts.

3. `transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2470, 0.2435, 0.2616))`: This normalizes the image tensor using the mean and standard deviation for each RGB channel, which are the computed means and standard deviations of the CIFAR-10 dataset[2].

1.2.2 Network design

Also, I added some useful units to the MLP, see appendix A.

1. `nn.BatchNorm1d()`: This is the BN unit which reduces internal covariate shift.

2. `nn.GELU()`: This is the Gaussian Error Linear Unit which nonlinearly amplifies values greater than zero, suppresses values less than zero, and is smoother than ReLU.

3. `nn.Dropout()`: This randomly sets the output of some neurons in the neural network to zero, makes the model independent of specific neurons or feature combinations during training.

1.2.3 Experiment

I have done the ablation experiment of MLPnet on CIFAR-10, see table 1. Here, the MLP has the same linear layers as the MLPnet, and uses ReLU as the activation function. The preprocessing of data and the upgrading of the structure of the network improved the testing accuracy by 1.24%.

Data Preprocessing	Model	Testing Accuracy
w/o	MLP	50.49
w/o	MLPnet	50.44
w	MLP	48.90
w	MLPnet	51.73

Table 1: Ablation experiment of MLPnet on CIFAR-10.

[1] `transforms.ToTensor()` is emitted here as it is not for improving the performance.
[2] Parameters are from https://stackoverflow.com/questions/69747119

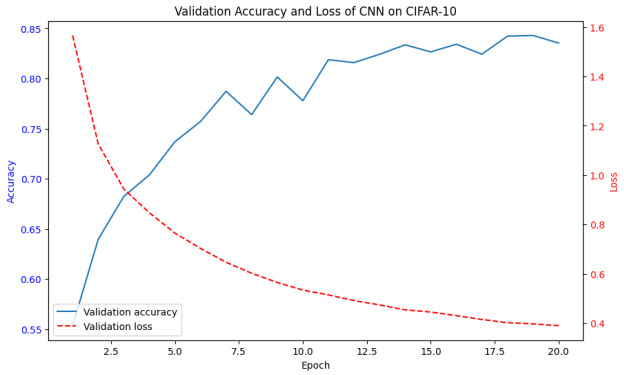

Figure 3: Validation Accuracy and Loss of CNN on CIFAR-10.

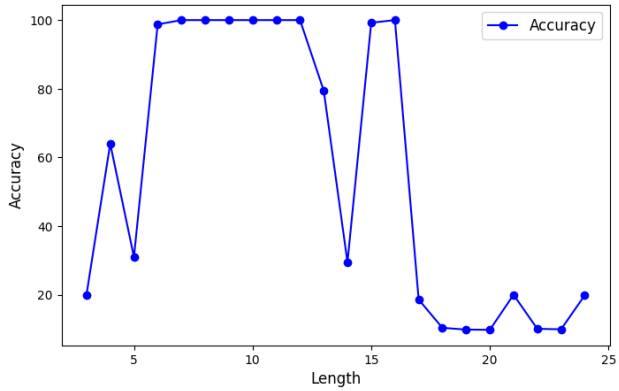

Figure 4: Validation Accuracy of RNN on palindromic number dataset of different lengths.

2 CNN

The CNN performs far better than the MLP, achieving an average testing accuracy of 83.30% on CIFAR-10, mainly because of its ability to capture spacial information. The curve of validation accuracy and loss are shown in fig. 3.

3 RNN

I tested the RNN on the palindromic number dataset, see fig. 4. The RNN is not perfectly stable on this task, and has a limited memory.

4 Conclusion

Each of the three deep learning model is suited to a specific class of problems. MLP is used for general-purpose tasks, CNN is used for spatial data like images, and RNN is used for sequential data like time-series or language.

A MLP

Here is how my MLPnet is defined in python.

```python
import torch.nn as nn
class MLPnet(nn.Module):
    def __init__(self, input_size=3 * 32 * 32, num_class=10):
        super(MLPnet, self).__init__()
        self.model = nn.Sequential(
            nn.Flatten(),
            nn.Linear(input_size, 512),
            nn.BatchNorm1d(512),
            nn.GELU(),
            nn.Linear(512, 512),
            nn.BatchNorm1d(512),
            nn.GELU(),
            nn.Linear(512, 256),
            nn.BatchNorm1d(256),
            nn.GELU(),
            nn.Linear(256, 256),
            nn.BatchNorm1d(256),
            nn.GELU(),
            nn.Linear(256, 128),
            nn.BatchNorm1d(128),
            nn.GELU(),
            nn.Dropout(p=0.5)
        )
        self.classifier = nn.Linear(128, num_class)
    def forward(self, x):
        feature = self.model(x)
        out = self.classifier(feature)
        return out
```