Problem Set 10: Deep Classification

Johannes C. B. Dietschreit, Sascha Mausenberger

Due: 16.12.2024

The problems below are meant to be solved using Python functions and libraries. You can do so by either writing short scripts or using Jupyter notebooks. **Report your results in the Quiz on Moodle** to obtain a grade.

Introduction

This PSet is an introduction to convolutional neural networks. Make sure to have the package `torchvision` installed. We will apply convolutions to colored pictures of objects. The second half of this PSet is meant the paint a clearer picture regarding the similarities and differences between deep and linear models.

The Problems

1 Recognizing Objects

Use the following code to download a data set from torchvision. They will be objects with the data already stored as PyTorch tensors or numpy arrays.

```
import torch
from torchvision import datasets, transforms

transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),  # Randomly flip some images horizontally
    transforms.RandomCrop(32, padding=4),  # Randomly crop with padding
    transforms.ToTensor(),  # Convert image to tensor
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))  # Normalize
    with mean and std
])
train_dataset = datasets.CIFAR10(root='./data', train=True, download=True, transform=
    transform)
test_dataset = datasets.CIFAR10(root='./data', train=False, download=True, transform=
    transform)

train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=50, shuffle=
    True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=50, shuffle=False
    )
```

1.1 How much larger is the training set in comparison to the test set?

(0.5 P)

1.2 What do the first 5 pictures in the training set represent?

(0.5 P)

1.3 We will start with a fully connected neural network and attempt to classify the pictures in their vectorized (flattened) form. There are several ways to set up a model in pytorch. Simple models can be put together with `nn.sequential`, however, in general, every model is its own class and a child of `nn.Module`.

```
from torch import nn, optim

input_size = 1024
hidden_sizes = [256, 64]
output_size = 10

# using nn.Sequential
model = nn.Sequential(nn.Linear(3*input_size, hidden_sizes[0]),
                      nn.ReLU(),
                      nn.Linear(hidden_sizes[0], hidden_sizes[1]),
                      nn.ReLU(),
                      nn.Linear(hidden_sizes[1], output_size),
                      nn.LogSoftmax(dim=1))

# general way of setting up a model
class Dense(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(3*input_size, hidden_sizes[0])
        self.fc2 = nn.Linear(hidden_sizes[0], hidden_sizes[1])
        self.fc3 = nn.Linear(hidden_sizes[1], 10)

    def forward(self,x):
        batch_size = x.shape[0]
        x = x.reshape(batch_size, -1)
        x = nn.functional.relu(self.fc1(x))
        x = nn.functional.relu(self.fc2(x))
        # Convert output into a categorical probability distribution
        x = nn.functional.log_softmax(self.fc3(x), dim=1)
        return x
```

Feel free to change the hidden layer sizes and to add or remove layers, but report your results with the sizes chosen in the code above.

How many parameters does the linear model from above have?

(1.0 P)

1.4 Train the deep model for only 10 epochs (can already take several minutes) using

```
criterion = nn.NLLLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

Feel free to reuse code from last week for writing the training loop.

Report the accuracy on the test set rounded to 2 digits after the decimal point.

(1.0 P)

1.5 Write a convolutional neural network (CNN) that uses `nn.Conv2d(3, 32, kernel_size=5, padding=1)`, `nn.MaxPool2d(2, 2)`, `nn.Conv2d(32, 64, kernel_size=3, padding=1)`, `nn.MaxPool2d(2, 2)`, a hidden layer of size 64 and a final linear layer that reduces to the necessary number of samples. In that order. Use SiLU activation after both convolutions and ReLU after the last hidden layer. This time, do not apply the softmax function.

How many parameters does this CNN have?

(1.0 P)

1.6 Train the CNN with the similar settings as the feed forward model: 10 epochs, Adam optimizer with lr=0.001, CrossEntropyLoss (since we do not apply the logarithmic softmax). Report the accuracy on the test rounded to 2 digits after the decimal point.

(1.0 P)

1.7 Earlier in the course we have seen that support vector machines are very robust classifiers. Let's compare how they perform on this task.

Turn the pytorch sets into numpy arrays if necessary and train an SVC classifier `SVC(kernel='rbf')` with the default settings. Note, this can take several minutes as SVCs scale poorly with training set size. Thus use only the first 10k points from the training set

Report the accuracy on the test set rounded to 2 digits after the decimal point.

(0.5 P)

1.8 As SVMs are sparse kernel models, they make use with less parameters than there is training data (they only use the support vectors). How many support vectors or "parameters" has the SVC model have (for reference, you used 10k training points)?

(0.5 P)

2 Learning Features

When performing classification with a dense neural network the last layer usually has as many outputs as there are classes (K). The predicted class label is then selected by the softmax function

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j+1}^{K} e^{z_j}}$$

In words, the softmax applies the standard exponential function to each element of the input vector of length K and normalizes these values by dividing by the sum of all these exponentials. Comparing a simple dense neural network with Logistic Regression, one can see that the LR model is simply a "neural network" without hidden layers.

2.1 Load the numpy arrays `X_classification.npy` and `y_classification.npy`, split them 50:50 using the `train_test_split` with `random_state=42` as usual. Train a logistic regression model and report the accuracy on the test set rounded to 3 digits after the comma.

(0.5 P)

2.2 Visualize the learned decision boundary against a scatter plot of all data.

(1.0 P)

2.3 One can clearly see that the decision boundary of the logistic regression method is not sufficient to correctly classify this fictitious data. The problem is that based on our two features we are too constricted to predict the decision boundary with a "single layer network". Thus, we replace the LR model with a deep one instead to learn better features. Use the code below.

```
class DeepClassifier(nn.Module):
    def __init__(self, n_hidden:int=1, hidden_dim:int = 32):
        super().__init__()
        torch.manual_seed(42)
        self.n_hidden = n_hidden
        self.first = nn.Linear(2, hidden_dim)
        self.between = [nn.Linear(hidden_dim, hidden_dim) for i in range(self.n_hidden
    -1)]
        self.last = nn.Linear(hidden_dim, 2)

    def forward(self,x):
        batch_size = x.shape[0]
        x = nn.functional.leaky_relu(self.first(x))
        for i in range(self.n_hidden-1):
            x = nn.functional.leaky_relu(self.between[i](x))
        x = nn.functional.log_softmax(self.last(x), dim=1)
        return x
```

```
class CustomDataset(torch.utils.data.Dataset):
    def __init__(self, features, labels):
        self.X = torch.from_numpy(features).float()
        self.y = torch.from_numpy(labels)

    def __len__(self):
        return len(self.y)

    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]

traindataset = CustomDataset(X_train, y_train)
trainloader = torch.utils.data.DataLoader(traindataset,
                                          batch_size=64, shuffle=True)
```

Train this model with the same settings as in problem 1.4 regarding loss function and optimizer. Train a model with 1 hidden layer and a layer dimension of 2 for 100 epochs. Report the accuracy on the test set rounded to 3 digits after the comma.

(1.0 P)

2.4 Train independent models with 1 hidden layer and a hidden layer dimension/length of [2, 4, 8, 16, 32, 64]. Again, use the same loss function and optimizer. Train for 100 epochs.

How many dimensions are at least necessary to achieve an accuracy on the test set of 0.99 or more?

(1.5 P)