dl-assignment-2

September 10, 2020

1 Deep Learning — Assignment 2

Second assignment for the 2020 Deep Learning course (NWI-IMC058) of the Radboud University.

Twan van Laarhoven (tvanlaarhoven@cs.ru.nl) and Gijs van Tulder (g.vantulder@cs.ru.nl)

September 2020

Names:

Group:

Instructions: * Fill in your names and the name of your group. * Answer the questions and complete the code where necessary. * Re-run the whole notebook before you submit your work. * Save the notebook as a PDF and submit that in Brightspace together with the `.ipynb` notebook file. * The easiest way to make a PDF of your notebook is via File > Print Preview and then use your browser's print option to print to PDF.

1.1 Objectives

In this assignment you will 1. Learn how to define and train a neural network with pytorch 2. Experiment with convolutional neural networks 3. Investigate the effect of dropout and batch normalization

1.2 Required software

If you haven't done so already, you will need to install the following additional libraries: * `torch` and `torchvision` for PyTorch, * `d2l`, the library that comes with Dive into deep learning book, * `sounddevice` to play audio, * `python_speech_features` to compute MFCC features.

All libraries can be installed with `pip install`.

```
[ ]: %matplotlib inline
import os
import numpy as np
from d2l import torch as d2l
import torch
from torch import nn
from scipy.io import wavfile
```

1.3 2.1 Digits dataset

The d2l book uses a dataset of images as a running example (FashionMNIST). In this assignment we will investigate CNNs in a completely different domain: speech recognition.

The dataset we use is the free spoken digits dataset, which can be found on https://github.com/Jakobovski/free-spoken-digit-dataset. This dataset consists of the digits 0 to 9, spoken by different speakers. The data comes as .wav files.

Use `git clone` to download the dataset.

Below is a function to load the data. We pad/truncate each sample to the same length. The raw audio is usually stored in 16 bit integers, with a range -32768 to 32767, where 0 represents no signal. Before using the data, it should be normalized. A common approach is to make sure that the data is between 0 and 1 or between -1 and 1.

Update the below code to normalize the data to a reasonable range

[ ]:
```
samplerate = 8000
def load_waveform(file, size = 6000):
    samplerate, waveform = wavfile.read(file)
    # Take first 6000 samples from waveform. With a samplerate of 8000 that␣
 ↪corresponds to 3/4 second
    # Pad with 0s if the file is shorter
    waveform = np.pad(waveform,(0,size))[0:size]
    # Normalize waveform
    # TODO: Your code here.
    return waveform
```

The following code loads all .wav files in a directory, and makes it available in a pytorch dataset.

Load the data into a variable `data`

[ ]:
```
class SpokenDigits(torch.utils.data.Dataset):
    def __init__(self, data_dir):
        digits_x = []
        digits_y = []
        for file in os.listdir(data_dir):
            if file.endswith(".wav"):
                waveform = load_waveform(os.path.join(data_dir, file))
                label = int(file[0])
                digits_x.append(waveform)
                digits_y.append(label)
        # convert to torch tensors
        self.x = torch.from_numpy(np.array(digits_x, dtype=np.float32))
        self.x = self.x.unsqueeze(1) # One channel
        self.y = torch.from_numpy(np.array(digits_y))
    def __len__(self):
        return len(self.x)
    def __getitem__(self, idx):
        return self.x[idx], self.y[idx]
```

```
# TODO: Your code here.
```

Describe the dataset: how many samples are there, what is their dimensionality? How many classes are there?

TODO: your answer here.

Here is code to play samples from the dataset to give you an idea what it "looks" like.

```
[ ]: import sounddevice as sd
     def play(sample):
         sd.play(sample[0][0], samplerate)
         print(sample[1])
     play(data[0])
```

```
[ ]: train_prop = 2/3
     train_count = int(len(data) * train_prop)
     train, test = torch.utils.data.random_split(data, [train_count,␣
      ↪len(data)-train_count])
```

The code above is code to split the data into a training and test set. It uses 2/3 of the data for training.

Discuss an advantage and disadvantage of using more of the data for training

TODO: your answer here.

Finally, we split the data into batches:

```
[ ]: data_params = {'batch_size': 32}
     train_iter = torch.utils.data.DataLoader(train, **data_params)
     test_iter  = torch.utils.data.DataLoader(test,  **data_params)
```

1.4 2.2 One dimensional convolutional neural network

We will now define a network architecture. We will use a combination of convolutional layers and pooling. Note that we use 1d convolution and pooling here, instead of the 2d operations used for images.

Complete the network architecture, look at the d2l book chapters 6 and 7 for examples

```
[ ]: net = torch.nn.Sequential(
         nn.Conv1d(1, 4, kernel_size=5), nn.ReLU(),
         nn.AvgPool1d(kernel_size=2, stride=2),
         # TODO: Add two more convolutional layers, ReLU layers and pooling layers;
         #       doubling the number of channels each time
         # TODO: Your code here.
         nn.Flatten(),
         nn.Linear(11872, 128), nn.ReLU(),
         nn.Linear(128, 64), nn.ReLU(),
    nn.Linear(64, 10))
```

The first fully connected layer has input dimension 11872, where does that number come from?

TODO: your answer here

Hint: think about how (valid) convolutional layers and pooling layers with stride affect the size of the data.

How many parameters are there in the model? I.e. the total number of weights and biases

[ ]:
```
# TODO: Compute the number of parameters
# Hint: use net.parameters() and param.nelement()
```

Suppose that instead of using convolutions, we had used only fully connected layers. How many parameters would be needed in that case approximately?

TODO: your answer here

The FashionMNIST dataset used in the book has 60000 training examples. How large is our training set? How would the difference affect the number of epochs that we need? Compare to chapter 6.6 and 7.1 of the book.

How many epochs do you think are needed?

[ ]:
```
lr, num_epochs = 0.01, 10 # TODO: change
```

We will use the code from the d2l book to train the network. In particular, the `train_ch6` function, defined in chapter 6.6. This function is available in the `d2l` library. However, this function has a bug: it only initializes the weights for 2d convolutional layers, not for 1d convolutional layers.

Make a copy of the train_ch6 function, and correct the error

[ ]:
```
def train(net, train_iter, test_iter, num_epochs, lr, device=d2l.try_gpu()):
    # TODO: your code here (copied and corrected from train_ch6)
```

Now train the network.

[ ]:
```
train(net, train_iter, test_iter, num_epochs, lr)
```

Is the training converged?

If the training has not converged, maybe you need to change the number of epochs and/or the learning rate.

TODO: Document the changes that you made and their effect.

1.5 2.3 Questions and evaluation

Does the network look like it is overfitting or underfitting?

TODO: your answer here

Is what we have here a good classifier? Could it be used in a realistic application?

TODO: discuss your answer

Do you think there is enough training data compared to the dimensions of the data and the number of parameters?

TODO: your answer here

How could the classifier be improved?

TODO: your answer here

The free spoken digits datasets has recordings from several different speakers. Is the test set accuracy a good measure of how well the trained network would perform for recognizing the voice of a new speaker? And if not, how could that be tested instead?

TODO: your answer here.

1.6 2.4 Variations

One way in which the training might be improved is with dropout or with batch normalization.

Make a copy of the network architecture below, and add dropout

Hint: see chapter 7.1 for an example that uses dropout.

```
[ ]: net_dropout = "TODO: your network here"
     train(net_dropout, train_iter, test_iter, num_epochs, lr)
```

How does dropout change the results?

TODO: your answer here

Make a copy of the original network architecture, and add batch normalization to all convolutional and linear layers.

Hint: see chapter 7.5 for an example.

```
[ ]: net_batchnorm = "TODO: your network here"
     train(net_batchnorm, train_iter, test_iter, num_epochs, lr)
```

How does batch normalization change the results?

TODO: your answer here

1.7 2.5 Bonus: feature extraction

Given enough training data a deep neural network can learn to extract features from raw data like audio and images. However, in some cases it is still necesary to do manual feature extraction. For speech recognition, a popular class of features are MFCCs.

Here is code to extract these features. You will need to install the `python_speech_features` first.

```
[ ]: from python_speech_features import mfcc
```

```
def load_waveform_mfcc(file, size = 6000):
    samplerate, waveform = wavfile.read(file)
    waveform = np.pad(waveform,(0,size))[0:size] / 32768
    return np.transpose(mfcc(waveform, samplerate))
```

Implement a variation of the dataset that uses these features

```
[ ]: class SpokenDigitsMFCC(torch.utils.data.Dataset):
         # TODO: Your code here.
         pass

     data_mfcc = SpokenDigitsMFCC(data_dir) # TODO: your data directory here
     train_count_mfcc = int(len(data_mfcc) * train_prop)
     train_mfcc, test_mfcc = torch.utils.data.random_split(data, [train_count_mfcc,␣
      ↪len(data_mfcc)-train_count_mfcc])
     train_iter_mfcc = torch.utils.data.DataLoader(train_mfcc, **data_params)
     test_iter_mfcc  = torch.utils.data.DataLoader(test_mfcc,  **data_params)
```

The MFCC features will have 13 channels instead of 1 (the `unsqueeze` operation is not needed).

Inspect the shape of the data, and define a new network architecture that accepts data with this shape

```
[ ]: # Your code here.
```

Train the network with the mfcc features.

```
[ ]: # Your code here.
```

Is there a neural-network based alternative to mfcc features?

TODO: your answer here

1.8 The end

Well done! Please double check the instructions at the top before you submit your results.