Your First Deep Learning Project

October 7, 2019

1 Introduction

Your initial task is to train a deep neural network on the MNIST dataset. You can start from a simple softmax classifier, but in the end you should obtain more than 99% accuracy on the test dataset. Each of you have to download this dataset and do some preprocessing, becase images are not in an usual format here. You have to convert your downloaded files to normal images and save each class in different directory. Then you should do some splitting (train, val, test). After that you should start to write your own code of training. Your code should consist of the following scripts

- main.py
- utils.py (optional)
- data_loader.py
- BaseNN.py
- DNN.py

See the zip file which I sent you.

2 main.py

I gave you this part of code. In this script are defined almost all important parameters for your training. For training your model you should use the following command line ”python3 main.py” (in ubuntu). So you have to write the remaining part of your code using this script. If it is necessary you can do some changes here. Before starting to write your code search and read about tf.app.flags.

3 utils.py

This script should be a collection of small functions and classes which make common patterns shorter and easier. This part of code is optional.

4 data_loader.py

In this script you should have class with name DataLoader, which will provide data for training. It will be better to have the following methods in your class: load_image(), batch_data_loader(), train_data_loader(), val_data_loader(), test_data_loader(). The function load_image should take path of an image as an argument and generate its matrix and label (one-hot) i.e. read the image from its path (here you can use scipy). The function batch_data_loader() should take index as an argument and return the batch of images from your specified list of paths of images (train, val, test) corresponding to that index. Other three functions should call the previous function. For example val_data_loader() should call the function data_loader() for the list of paths of images from validation dataset.

5 BaseNN.py

The main part of your code will be here. In this script you must have **abstract class** with name BaseNN. In this class you have to write everything you need for your training except of the following two functions: network(), metrics(). This two methods should be the only abstract methods for your abstract class (see descriptions of these functions in the next chapter). Note that according to main.py BaseNN must have methods create_network(), initialize_network(), train_model(), test_model(). The function create_network() should create whole graph of training:

- define placeholders for input and labels,
- "compute" prediction of your network,
- "compute" the loss function,
- setup optimizer,
- setup operation which will minimize your loss function.

The next one should initialize network randomly if you haven't checkpoint (good weights), otherwise it should initialize your network with checkpoint.

The function train_model() should train your model passing through your data num_epochs times, do validation after validation_step steps, save checkpoint after checkpoint_step steps and display loss after display_step steps (see main.py).

Every time when you will run your code, it should create folder with name base_dir in your working directory (see main.py) if it doesn't exist, then in base_dir it should create subfolder with name model_name (see main.py) and in this directory it should create subfolders with names "chekpoints" and "summaries", where will be saved your "good weights" and tensorboard summaries respectively.

6 DNN.py

This script should contain class DNN, which is derived from the class BaseNN. Here you should only implement abstract methods of the class BaseNN: network() and metrics(). The first one should take an input with shape

$$(\text{batch\_size}, \text{height\_of\_image}, \text{width\_of\_image}\ , \text{num\_channels})$$

and should give output with shape (batch_size, num_classes) using some architecture. The second one should compute the loss function between labels and your prediction and other metrics if it is necessary (for example accuracy).

That's all. Good luck!