Practical Introduction to Neural Networks
Homework 2
Due: May 1, 2019

Overview

Before starting with this homework please make sure that you have TensorFlow and Jupyter installed on your laptop. In addition, **GPU** is needed for training CNN assignments. If you don't have available GPU, you could use the Compute Engine of Google Cloud Platform. Please check the setup guide on canvas. For each problem in this homework we have provided a Jupyter notebook template with which you can get started.

Templates are available at the course Github Repository:
`https://github.com/shlizee/PracticalIntroductionNN`

Please submit Jupyter notebooks with your code (and outputs) by the due date to your Github repository and place a link to your repository in the canvas assignment.

Problem 1: Implement A Neural Network From Scratch

Implement a single hidden layer neural network from scratch (using Numpy). The network will be trained on the MNIST dataset (You can still acquire the data from TensorFlow as before.) This Problem does not have **.ipynb** template. In the following parts, you are free to use any comfortable tool in Python to implement the neural network (Classes, functions, brute-force for loops etc.) but the plots and discussions should be displayed in a **single .ipynb** file. Please indicate the corresponding parts using comments or markdown cells in your **.py** or **.ipynb** files. Please use the 'ReLu' as the activation function of the hidden layer, softmax function as the activation function of the output layer, and cross entropy for computing the cost function. Choose the values of other hyper-parameters as they fit the problem.

Part a:

Define and implement the necessary components of your computation graph (training/testing inputs, outputs, layers, learning rate, cost function).

Part b:

Implement forward propagation and compute the cost.

Part c:

Compute gradients of your cost function.

Part d:

Implement back-propagation. (Use Gradient Descent update rule).

Part e:

Plot the cost versus training iterations using different mini-batch sizes: $16, 64, 256, 1024$. Record the test accuracy in percentage and total training time you spent in seconds.

Part f:

Implement Adam Optimizer (Use $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$) and repeat **Part e**.

Part g:

Implement the same architecture in TensorFlow, repeat **Part d and e**, compare the cost, test accuracy and training time against your Numpy Neural Network.

Problem 2: CIFAR-10 dataset with Fully Connected NN

For MNIST data it is pretty easy to get high accuracy with little effort. In this problem, you will be working with more interesting dataset: CIFAR-10. The dataset consists of 60000, 32×32 **color images** in 10 classes, with 6000 images per class. The dataset is divided into five training batches and one test batch, each with 10000 images. Please check the details and download the **CIFAR-10 python version** from: `https://www.cs.toronto.edu/~kriz/cifar.html`

You will find the following files: data_batch_1,...,data_batch_5, as well as test_batch. Each of these files is a Python "Pickle" object. You can follow the instruction on the website to open such files which returns a dictionary with keys: 'data' and 'labels'(use encoding='latin1' instead of encoding='bytes').

Part a:

Implement a set of necessary functions to prepare the data for training. In particular,

- Define functions to load the training and testing data from the pickle files
- Perform min-max normalization of the data features
- Perform one-hot encoding on your training and testing data labels
- Separate the validation set from the training set

Save the pre-processed data into new pickle files for later usage. You will implement these functions in **load_cifar_template.py**. Once you are done, use **HW2_2a_template.ipynb** to verify.

Part b:

Implement a fully connected neural network to classify the CIFAR-10 dataset in TensorFlow. You are free to use any concept taught in class (initialization, regularization, dropout, batch normalization, etc.). You should be able to reach at least 50% test accuracy within 10 training epochs (Do continue to train further and examine training and testing performance). Implement the model using **HW2_2b_template.ipynb**. Explain your insights and the training procedure. Support your investigation with plots.

Problem 3: LeNet-5 CNN

Part a: MNIST dataset

Implement the LeNet-5 model in TensorFlow using **HW2_3a_template.ipynb**. You should be able to reach 99% test accuracy for MNIST data. (Note: MNIST image size is 28×28, remember to pad images with 0s to get 32×32 image size before training).

Part b: CIFAR-10 dataset

Train the LeNet-5 CNN on CIFAR-10 dataset. You should be able to reach at least 60% test accuracy after 10 training epochs. Implement the model using **HW2_3b_template.ipynb**

Problem 4: AlexNet CNN

In this problem, you will reproduce the AlexNet CNN to solve Cats and Dogs classification problem. Download the dataset from: `https://www.kaggle.com/c/dogs-vs-cats/data` (Note: to downlaod the data from Kaggle you may be required to sign up for an account with Kaggle). There are two zip files (train.zip and test1.zip). Train.zip contains 25000 training images (12500 for cats and dogs respectively) and test1.zip contains 12500 images without labels.

Part a:

As for CIFAR-10 data-set, implement the necessary functions to prepare for training, validation and testing data. For details, check **HW2_4a_template.ipynb**.

Part b:

Implement the AlexNet CNN in TensorFlow and perform training for binary classification: dog or a cat. You should be able to reach at least 85% **validation accuracy** within 10 epochs. Plot the first 10 test images with a title that shows the probability of the predicted class. Use **HW2_4b_template.ipynb** to implement the model.