Homework 1

In this homework, we will learn how to implement backpropagation (or backprop) for “vanilla” neural networks (or Multi-Layer Perceptrons) and ConvNets.
You will begin by writing the forward and backward passes for different types of layers (including convolution and pooling), and then go on to train a shallow ConvNet on the CIFAR-10 dataset in Python. Next you’ll learn to use Caffe (http://caffe.berkeleyvision.org/), a BVLC (http://bvlc.eecs.berkeley.edu/) maintained, open-source deep learning framework, and replicate the experiments from before in Caffe.

This homework is divided into the following parts:

- Implement a neural network and train a ConvNet on CIFAR-10 in Python.
- Learn to use Caffe and replicate previous experiments in Caffe (2-layer NN, ConvNet on CIFAR-10).

Download the starter code here (https://github.com/batra-mlp-lab/VT-F15-ECE6504-HW1/archive/1.0.zip).

Part 1

Starter code for part 1 of the homework is available in the `1_cs231n` folder. Note that this is assignment 2 from the Stanford CS231n course (http://cs231n.github.io/assignment2/).

Setup

Dependencies are listed in the `requirements.txt` file. If working with Anaconda, they should all be installed already.

Download data.

```
cd 1_cs231n/cs231n/datasets
./get_datasets.sh
```

Compile the Cython extension. From the `cs231n` directory, run the following.

```
python setup.py build_ext --inplace
```

Q1.1: Two-layer Neural Network (10 points)

The IPython notebook `two_layer_net.ipynb` will walk you through implementing a two-layer neural network on CIFAR-10. You will write a hard-coded 2-layer neural network, implement its backward pass, and tune its hyperparameters.

Q1.2: Modular Neural Network (16 points)

The IPython notebook `layers.ipynb` will walk you through a modular neural network implementation. You will implement the forward and backward passes of many different layer types, including convolution and pooling layers.

Q1.3: ConvNet on CIFAR-10 (8 points)

The IPython notebook `convnet.ipynb` will walk you through the process of training a (shallow) convolutional neural network on CIFAR-10.

Deliverables

Zip the completed ipython notebooks and relevant files.

```
cd 1_cs231n
./collectSubmission.sh
```

Submit the generated zip file `1_cs231n.zip` .

Part 2

This part is similar to the first part except that you will now be using Caffe (http://caffe.berkeleyvision.org/) to implement the two-layer neural network and the convolutional neural network.

Note that you only need to define the various layers in the architecture file and the solver settings in the solver file.

You will be using existing layers and hence, this part does not involve any code as such.

- Layers and their definitions implemented in Caffe are here (http://caffe.berkeleyvision.org/tutorial/layers.html)
- Details about the solver file and methods are here (http://caffe.berkeleyvision.org/tutorial/solver.html)
- Caffe has many examples (https://github.com/BVLC/caffe/tree/master/examples) which are very helpful. It is in your `caffe/examples` folder.

The necessary files for this section are provided in the `2_caffe` directory.

Q2.1: Logistic Regression using Caffe (0 points)

The `logistic-regression.ipynb` notebook will walk you through implementing logistic regression using Caffe. This includes preparing the data and training. The `filter-viz.ipynb` will show you how to load a trained model into Caffe and visualize the learned weights.

There are no points for this question. It is provided to familiarize you with Caffe.

Q2.2: Two-layer Neural Network using Caffe (8 points)

By now, you have an idea of working with Caffe and may proceed to implementing a two-layer neural network. Go to the `2layernn` folder and complete the following:

- `2layernn.prototxt` : the architecture file
- `2layernn_solver.prototxt` : you may want to change the details like learning rate in the solver file
- Train the neural network using run_2layernn.sh - you will need make appropriate changes in path if necessary and the log file.
- Use `parse_log` to generate the `training loss vs iterations` and `validation accuracy vs iterations` as shown in `logistic-regression.ipynb` .
- Make suitable modifications in `filter-viz.ipynb` and save the weights of the first hidden layer.

Q2.3: ConvNet using Caffe (8 points)

Repeat the above steps for a convnet. Starter code is in `convnet` and remember to save the filters learned.

Deliverables

Zip the following files:

1. `training loss vs iterations` and `validation accuracy vs iterations` - resulting from both Q2.2 and Q2.3
2. Images of the visualized filters - from Q2.2 and Q2.3

Experiment (Extra credit: up to 10 points)

Experiment and try to get the best performance that you can on CIFAR-10 using a ConvNet.

- Filter size: Above we used 7x7; this makes pretty pictures but smaller filters may be more efficient
- Number of filters: Above we used 32 filters. Do more or fewer do better?
- Network depth: Some good architectures to try include:
    - [conv-relu-pool]xN - conv - relu - [affine]xM - [softmax or SVM]
    - [conv-relu-pool]XN - [affine]XM - [softmax or SVM]
    - [conv-relu-conv-relu-pool]xN - [affine]xM - [softmax or SVM]
- Alternative update steps: AdaGrad, AdaDelta

Deliverables

- Prototxt files (`train_val` , `solver` , `deploy`)
- Training log and `training loss vs iterations` plot
- List and describe all that you tried

References:

1. CS231n Convolutional Neural Networks for Visual Recognition (http://cs231n.stanford.edu/)
