HW2 18739

Due Date: Mar 20th Before Class

1 Introduction

In this homework[1], you will be implementing new techniques introduced in lectures on Neural Networks, and learn to train Convolutional Neural networks. You will understand the architecture of Convolutional Neural Networks and gain experience with training these models on benchmark datasets. There are 3 parts in this homework. In the first part, you will be revisiting your Keras implementation from last assignment and implementing convolutional structures to improve the accuracy by a great margin. In the second part, you will be implementing 3 specific structures/layers in CNN using `numpy`. In the third part, you will be training a CNN Model on the CIFAR–10 dataset.

2 Setup

In PART I and II and 5.2 & 5.3 of PART III, you can run your program in the notebook locally. However, it is recommended that you use GPU for 5.4 and 5.5 of PART III.

2.1 Python and Anaconda

In this homework, **you should only use Python 3.5 or 3.6**. Python 2 may not work with the code provided. Again, the use of Anaconda is highly recommended. If you have been using Python 2 and want to create a separate a environment for Python 3, please consult here.

2.2 Xsede Setup Guide

In this course, we are allocated to the Bridges server on PSC (Pittsburgh Super Computer) which has powerful GPU units. You probably have received an email about PSC and your resource allocation. You can see your group affiliation/resource usage in your portal when you log in through `https://portal.xsede.org/`. To use the Bridges server, you can consult the documentation here or follow the following steps, remember, you only need to use GPU for PART III 5.4 and 5.5:

[1]Based in part on material from Stanford CS231n `http://cs231n.github.io/`

1. You need to go to `https://apr.psc.edu/autopwdreset/autopwdreset.html` to set up a password for your PSC account. It is the password you need to `ssh` into the Bridges server and is a separate password from your Xsede account password.

2. After setting up the password, you can open up a terminal on a unix machine(use PuTTY for windows) and type command `ssh username@bridges.psc.xsede.org` to log into the system. For other ways to log in, please consult here.

3. For managing your file space, please consult here. It is recommended that you store your code files in `pylon2` and manage your software in the `home` directory. You each have 10 GB Physical storage. Please don't go over the limit by checking your usage using command `quota -s`. **Please don't run any program unrelated to this homework**.

4. For transferring your files from local to the server, please consult here.

5. In your home directory, please run `bash install_miniconda.sh` to download and install `miniconda`(A mini version of Anaconda). Then add miniconda to your environment by command `export PATH=$PATH:/home/username/miniconda3/bin/`(you need to do that every time you log on the server). Then create the software environment(preferable in your home directory instead of /pylon2)by running `conda env create -f environment.yml -n myenv`, which contains the softwares you need for tensorflow and GPUs. After all installations are complete, switch to that environment using command `source activate myenv`. Then load CUDA by `module load cuda`, you need to do this every time you log on to the server. For the `module` command and gpu usage, please consult here and here.

6. After you have set up your software environment, please carefully read `https://www.psc.edu/bridges/user-guide/running-jobs` to learn about running a GPU job. You want to use the partitions(GPU,GPU-shared...) with GPU configurations. It is recommended that you use one GPU and you won't need more than 2 GPUs for this homework. For example, you can try `interact -p GPU-shared --gres=gpu:p100:1 -t 04:00:00` to request a single P100 GPU with max time of 4 hours. If you don't want to wait while you run, please use the `sbatch` option instead. You might need to wait in a queue depending on the traffic and your request.

7. After starting a GPU session, you will see something like `[username@gpuXX ]$` on your screen. Please type `nvidia-smi` to check if your GPU is ready to go.

3 PART I. Keras Upgrade for MNIST (20pts)

In this part, you will be implementing more complicated/powerful networks on MNIST dataset on the Keras section from Homework I. You can run this part(`hw1_part1.ipynb`) on your laptop.

3.1 Upgrade I: 2-layer Network

For the first model, you will be implementing a 2-layer Network. Inbetween the 1 layer network structure you had from HW1, you should add a hidden layer of 64 neurons and nonlinear Relu layer, with the following features:

- Weight initialization using multi-dimensional Gaussian normalized by its variance. Set Bias initialization to 0. (Feb.6's lecture slide pg.26). You find the Keras initializer API here.
- Use Rmsprop optimizer with learning rate 0.01 and decay(rho) of 0.99.

Note that it is also a good practice to normalize your data. Since this dataset is normalized when loaded, you don't have to do that. Report your accuracy on the test set, number of epochs and your batch size.

3.2 Upgrade II: CNN

In this section, you will be implementing a 2-layer Convolutional Neural Network. First, you need to reshape the input data into 28× 28 (since these are black-white images, there are no color channels) using `numpy.reshape()`. Your layer should have the following structures:

- Please also use weight initialization and same Rmsprop optimizer as in your first model.
- Layer 1 : A convolutional layer (Documentation) 32 neurons(filters) with size 5×5, stride of 1 and 0 zero padding. Then add Relu activation.
- Layer 2: A Maxpooling layer (Documentation) with size 2.
- Layer 3: Add a dropout layer(Documentation) which randomly churns 20% of the neurons.
- Layer 4: Connect the output with layer 3 with a fully connected layer with ReLu activation(similar to the one in the first model) with 128 neurons and then connect to the output via a softmax.

The first 2 layers are usually grouped together to perform "one round" of convolutional operation for images. In this model, we only have one such round. Typically the state–of–art models of image classification have much more such convolutional operations to deal with larger and more complicated images. After reporting your validation/test accuracy, batch size and epochs(For this model, you will find you model converging with much fewer epochs), please answer the following questions:

- What is the dimension of the output of each neuron of the first layer for a batch of images (size 28× 28× N)?

- If an image I has dimension (width, height, depth) of $W \times H \times D$ and is passed through a convolutional layer with N filters of filter size F, stride S and zero padding Z, what is the dimension of the filtered output for I?
- In this model, what is the dimension of the output of each neuron of the second layer(Maxpooling) for a batch of images (size 28$\times$ 28$\times$ N)?

3.3 Upgrade III: Your own Model

In this section, you are free to design your own model with your own parameters. Although the accuracy is already pretty high, you should achieve even better validation accuracy or faster convergence compared with Upgrade II. Remember, you can add or get rid of features/structures from your second model. Since this is a simple dataset, a lot of the benefits of some regularization techniques are not apparent, however, they will be really useful in training on a larger/more complicated dataset such as CIFAR–10 in PART III. However, please do try out different opimitizers/non-linear activation functions and see which works best. Please report the following:

- You best validation accuracy and the corresponding test accuracy, as well as batch size and number of epochs.
- A summary of your model architecture, including the hyper-parameters you chose for each layer.

4 PART II. Numpy implementations (25 pts)

Knowing that you can implement any a lot of common techniques/structures using power APIs such as Keras, you will be implementing some basic structures in CNN/DFN yourself in this part using only `numpy`. You will be completing 3 functions in `hw2_part2.ipynb` which includes Convolutional layer and Dropout layer and batch normalization. Each of these functions comes with some test code for you to test your implementation.

4.1 Understanding Convolution(10 pts)

In this section, you will be implementing a function which does the forward pass of a convolutional layer.
After you finish your function, you can test your code by running the testing cell which inputs two images into a convolutional layer with only 2 neurons(filters). The first neuron is already hard-coded, while you have to specify the values for the second neuron yourself:

- First Neuron: By looking at the filtered output, what does the filter do? Please provide your guess.

- Second Neuron: Fill in the values of the second neuron which converts an RGB channel image into a grey scale image by using Luma coding: $G = 0.299R + 0.587G + 0.114B$, where G is the converted greyscale image and R, G, B are the corresponding Red, Green and Blue channel of the images. (*Hint: if you want to just keep the red channel of the image and filter out the blue/green channels,or $G = 1R + 0G + 0B$, what would the filter look like?)*

If your code outputs result as you expected, then it means you have successfully implemented the forward pass of a convolutional layer. In most DL APIs, you can simply add the layer by specifying the dimensions/parameters without having to write the functions yourself. Also, in this example, we define two interpretable filters ourselves, however, in CNN, all filtered are initialized randomly so that they each learn specific things (by back–propagation) which are not always interpretable. We will delve into understanding how these neurons work in future lectures/assignments.

4.2 Dropout Implementation (5 pts)

Please implement function `dropout()` to implement a forward pass for a dropout layer, which randomly churns neurons in a network. Since a dropout layer behaves differently during training and testing, make sure to implement the operation for both modes. Once you have done so, run the cell immediately below to test your implementation.

4.3 Batch Normalization (10 pts)

For this section,it is recommended that you read the batch-normalization paper before you complete `batch_normalization_foward()` and run the tests. At training time, a batch normalization layer uses a minibatch of data to estimate the mean and standard deviation of each feature. These estimated means and standard deviations are then used to center and normalize the features of the minibatch. A running average of these means and standard deviations is kept during training, and at test time these running averages are used to center and normalize features.

5 PART III. CIFAR–10 Training in Tensorflow (45 pts)

In this part, you will train a CNN on a benchmark dataset. You should run 5.1 through 5.3 on you laptop and 5.4 & 5.5 on the Bridges server. To get the dataset, run `bash get_dataset.sh`.

5.1 Learning Resources

TensorFlow has many excellent tutorials available, including those from Google themselves. Otherwise, the code in `hw3_part3.ipynb` will walk you through much of what you need to do to train models in TensorFlow.

5.2 Model 1: A simple model (0 pts)

In this part, you don't have to write any code or use the remote server. You should study the code/documentation and run the cell on your laptop. **Make sure you understand the code before you proceed.**

5.3 Model 2: A specific model(10 pts)

In this section, we're going to specify a model for you to construct. Again, you don't have to use your the GPU yet. The goal here isn't to get good performance (that'll be next), but instead to get comfortable with understanding the TensorFlow documentation and configuring your own model. Using the code provided in model1 as guidance, and using the following TensorFlow documentation, specify a model with the following architecture:

- 7x7 Convolutional Layer with 32 filters and stride of 1
- ReLU Activation Layer
- Spatial Batch Normalization Layer (trainable parameters, with scale and centering)
- 2x2 Max Pooling layer with a stride of 2
- Affine layer with 1024 output units
- ReLU Activation Layer
- Affine layer from 1024 input units to 10 outputs

After you finish your function, please use the cell immediately below in the notebook to check if you are doing the right thing.

5.4 GPU (5 pts)

Starting from this session, you are going to have to log onto bridge server to use the GPU. Please refer to the setup section above to set up your server and transfer your code files. You don't have to transfer your data file since it might be slower compared to downloading it again directly on the cloud. Start an interactive Python shell and follow the 5 steps in `hw2_part3_gpu.py`.

1. Load the data.
2. Test you GPU device. If there is no GPU, we get a Python exception and have to rebuild our graph.

3. Set up an **RMSprop optimizer** (using a 1e-3 learning rate) and a **cross-entropy loss** function. See the TensorFlow documentation for more information:

    - Layers, Activations, Loss functions `https://www.tensorflow.org/api_guides/python/nn`
    - Optimizers: `https://www.tensorflow.org/api_guides/python/train#Optimizers`

4. Train your model. Now that you've seen how to define a model and do a single forward pass of some data through it, this piece of code will through how you'd actually train one whole epoch over your training data (using the `model2()` you created).

5. Test your model,You should see a loss of 1.4 to 2.0 and an accuracy of 0.4 to 0.55. There will be some variation due to random seeds and differences in initialization.

6. Copy your output of running `hw2_part3_gpu.py` to the corresponding cell in `hw2_part3.ipynb`.

5.5 Your Own Model (30 pts)

In this section, you are going to implement your own model. Fill in the code in `hw2_part3_mymodel.py` and run it on GPU server. It's your job to experiment with architectures, hyperparameters, loss functions, and optimizers to train a model that achieves $\geq 70\%$ **accuracy** on the validation set of CIFAR–10. You can use the `run_model()` function to train and test your model.

5.5.1 Things to try

- Filter size: In model 2 we used 7x7; this makes pretty pictures but smaller filters may be more efficient

- Number of filters: In model 2 we used 32 filters. Do more or fewer do better?

- Pooling vs Strided Convolution: Do you use max pooling or just stride convolutions?

- Batch normalization: Try adding spatial batch normalization after convolution layers and vanilla batch normalization after affine layers. Do your networks train faster?

- Network architecture: The network above has two layers of trainable parameters. Can you do better with a deep network? Good architectures to try include:

    – (conv-relu-pool)$\times$N $\rightarrow$ (affine)$\times$M $\rightarrow$ (softmax or SVM)

  - (conv-relu-conv-relu-pool)×N → (affine)×M → (softmax or SVM)
  - (batchnorm-relu-conv)×N → (affine)×M → (softmax or SVM)

- Use TensorFlow Scope: Use TensorFlow scope and/or tf.layers to make it easier to write deeper networks. See this tutorial for how to use `tf.layers`.
- Regularization: Add l2 weight regularization, or perhaps use Dropout.

5.5.2 Training Tips

For each network architecture that you try, you should tune the learning rate and regularization strength. When doing this there are a couple important things to keep in mind:

- If the parameters are working well, you should see improvement within a few hundred iterations
- Remember the coarse-to-fine approach for hyperparameter tuning: start by testing a large range of hyperparameters for just a few training iterations to find the combinations of parameters that are working at all.
- Once you have found some sets of parameters that seem to work, search more finely around these parameters. You may need to train for more epochs.
- You should use the validation set for hyperparameter search, and we'll save the test set for evaluating your architecture on the best parameters as selected by the validation set.
- Don't stick to the book! Feel free to try something novel.

5.5.3 What We Expect

At the very least, you should be able to train a ConvNet that gets $\geq 70\%$ **validation accuracy**. This is just a lower bound - if you are careful it should be possible to get accuracies much higher than that!

5.5.4 Report your work/findings

Please copy the result of running `hw2_part3_mymodel.py` to the corresponding cell in `hw2_part3.ipynb`, also include the GPU configuration you used for training. In a separate cell you should write an explanation of what you did, any additional features that you implemented, and any visualizations or graphs that you make in the process of training and evaluating your network. If you implement certain structures, or choose certain parameters, please provide **justifications for your choices**. You can either provide your intuition in the justification, or state some empirical evidence , for example, you can say something like "by changing parameter a from 1 to 0.1, my validation accuracy increased from X to X". Or you can list out some of your top architectures and compare them. Please also provide your insights/findings as you are doing the training/validation, for example, you might find doing certain things speed things up a lot without compromising your accuracy. If you have written any additional code to for these findings, please include them in `hw2_part3_mymodel.py`.

5.5.5 Report Your Test Accuracy

Report your test accuracy in the final cell, (with the model that provides you with the best validation accuracy) after you have done everything above. In other words, your test data should only be used once and you should never adjust your model based on your test data. Compare your validation accuracy against your test accuracy.

5.5.6 Grading

Of the 30 pts in this section, 10 pts will be based on your test result/ how your validation/test accuracy compared to the rest of the class. 10 pts will based on your report, and 10 pts will be based on your coding.

Have fun and happy training!

Submission

You have to submit the files according to the following procedures:

1. Rename all the ipynb files (hw2_partx.ipynb) as ⟨your_andrew_id⟩_hw2_partx.ipynb.
   and rename all the python files(hw2_part3_XXX.py) as ⟨your_andrew_id⟩_hw2_part3_XXX.py
2. Make sure that you have run all your program in all the cells before you submit so that the results/plots can be seen by simply opening the file. In other words, the grader won't have to run the program to see your results.
3. Please comment your code.
4. Please cite all your references in the corresponding `ipynb` file.

Put these 6 files into ⟨your_andrew_id⟩_HW2 folder, zip the folder into ⟨your_andrew_id⟩_HW2.zip and submit the zip file on Canvas.