CS7643: Deep Learning
Assignment 2

Instructor: Zsolt Kira

Deadline: 8:00am September 30th, 2024, ET

- This assignment is due on the date/time posted on canvas. We will have a 48-hour grace period for this assignment. However, no questions regarding the assignment are answered during the grace period in any form.
- Discussion is encouraged, but each student must write his/her own answers and explicitly mention any collaborators.
- Each student is expected to respect and follow the GT Honor Code. **We will apply anti-cheating software to check for plagiarism**. Anyone who is flagged by the software will automatically receive 0 for the homework and be reported to OSI.
- Please **do not change the filenames and function definitions** in the skeleton code provided, as this will cause the test scripts to fail and you will receive no points in those failed tests. You may also **NOT** import additional modules other than what is already used in the provided code.
- It is your responsibility to make sure that all code and other deliverables are in the correct format and that your submission compiles and runs. We will not manually check your code (this is not feasible given the class size). Thus, **non-runnable code in our test environment will directly lead to a score of 0**. Also, your entire programming parts will **NOT** be graded and given a 0 score if your code prints out anything that is not asked in each question.

Theory Problem Set

You must show your work for full credit

1. The convolution layer inside of a CNN is intrinsically an *affine transformation*: A vector is received as input and is multiplied with a matrix to produce an output (to which a bias vector is usually added before passing the result through a nonlinearity). This operation can be represented as $y = Ax$, in which A describes the affine transformation.

   We will first revisit the convolution layer as discussed in the class. Consider a convolution layer with a 3x3 kernel W, operated on a single input channel X, represented as:

   $$W = \begin{bmatrix} w_{(0,0)}, w_{(0,1)}, w_{(0,2)} \\ w_{(1,0)}, w_{(1,1)}, w_{(1,2)} \\ w_{(2,0)}, w_{(2,1)}, w_{(2,2)} \end{bmatrix}, X = \begin{bmatrix} x_{(0,0)}, x_{(0,1)}, x_{(0,2)} \\ x_{(1,0)}, x_{(1,1)}, x_{(1,2)} \\ x_{(2,0)}, x_{(2,1)}, x_{(2,2)} \end{bmatrix} \tag{1}$$

   Using this example, let us work out a **stride-4** convolution layer, with **zero padding size of 2** and let the output be Y. Consider 'flattening' the input tensor X in row-major order as:

   $$x = \left[x_{(0,0)}, x_{(0,1)}, x_{(0,2)}, ..., x_{(2,0)}, x_{(2,1)}, x_{(2,2)}\right]^\top \tag{2}$$

   Note this x does not and should not include any padding!

   Write down this convolution of W and X as a matrix operation A such that: $y = Ax$. This is similar to the im2col operation in the lectures but slightly different. Output y is also flattened in row-major order. NOTE: For this problem we are referring to a convolution in the deep learning context (i.e. do not flip the kernel). Note that the flattened x does not contain any padding.

   (a) What are the dimensions of A?

   (b) Write down the entries in this matrix A.

2. Consider a specific 2 hidden layer ReLU network with inputs $x \in R$, 1 dimensional outputs, and 2 neurons per hidden layer. This function is given by

   $$h(x) = W^{(3)} \max\{0, W^{(2)} \max\{0, W^{(1)}x + b^{(1)}\} + b^{(2)}\} + b^{(3)} \tag{3}$$

where the max is element-wise, with weights:

$$W^{(1)} = \begin{bmatrix} 1.5 \\ 0.5 \end{bmatrix} \tag{4}$$

$$b^{(1)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{5}$$

$$W^{(2)} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \tag{6}$$

$$b^{(2)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{7}$$

$$W^{(3)} = \begin{bmatrix} 1 & 1 \end{bmatrix} \tag{8}$$

$$b^{(3)} = -1 \tag{9}$$

An interesting property of networks with piece-wise linear activations like the ReLU is that on the whole they compute piece-wise linear functions. At each of the following points $x = x_o$, determine the value of the new weight $W \in R$ and bias $b \in R$ such that $\frac{dh(x)}{dx}|_{x=x_o} = W$ and $Wx_o + b = h(x_o)$.

$$x_o = 2 \tag{10}$$

$$x_o = -1 \tag{11}$$

$$x_o = 1 \tag{12}$$

3. The ReLU function sometimes has problems where it creates “dead-neurons” due to producing 0 gradients when the activation is less than 0. In practice however, this does not seem to be a big problem. Why is this?

Paper Review

In this section, you must choose **one** of the papers below and complete the following:

1. provide a short review of the paper
2. answer paper-specific questions

Guidelines: Please restrict your reviews to no more than 350 words and answers to questions to no more than 350 words per question. The review part (1) should include answers to the following:

3. What is the main contribution of this paper? In other words, briefly summarize its key insights. What are some strengths and weaknesses of this paper?

4. What is your personal takeaway from this paper? This could be expressed either in terms of your perceived novelty of this paper compared to others you've read in the field, potential future directions of research in the area that the authors haven't addressed, or anything else that struck you as being noteworthy.

Paper Choice 1:

The first of our paper reviews for this module comes from recent work that discusses *biases* induced by the convolutional neural network. It turns out that CNNs and humans have different biases, but there might be methods to force the network to learn biases that better reflect ones that humans have. The paper can be viewed here.

Questions for this paper:

- Why would we care about the biases of the neural network, even if it can do well on the training set? Should we desire the network to have the same biases as humans or not? Why or why not?

- Why would training on stylized images change the bias of the network, and why might this bias generalize better to datasets with corruptions?

Paper Choice 2:

The second paper is an empirical study of transfer learning, specifically looking at which task (e.g. semantic segmentation, edge detection, depth estimation, etc.) is more transferable to another task. The paper is here and webpage/visualization are here.

Questions for this paper:

- Do the task pairs with stronger arrows (better transfer) make sense in terms of why they would transfer better? Pick one positive pair (with good transfer) and one negative pair (with bad transfer) and conjecture why it might be the case. Note that there are several types of features in deep learning, including low-level (e.g. edges), mid-level (components), and high-level (abstract concepts and classification layer) that you might reason about.

- What does this say in terms of practical usage of deep learning across tasks? How might we use this information to guess where to transfer from if we have a new target task?

Coding: Implement and train a network on CIFAR-10

Overview

Convolutional Neural Networks (CNNs) are one of the major advancements in computer vision over the past decade. In this assignment, you will complete a simple CNN architecture from scratch and learn how to implement CNNs with PyTorch, one of the most commonly used deep learning frameworks. You will also run different experiments on imbalanced datasets to evaluate your model and techniques to deal with imbalanced data.

Python and dependencies

In this assignment, we will work with **Python 3**. If you do not have a python distribution installed yet, we recommend installing Anaconda (or miniconda) with Python 3. We provide `environment.yaml` (present under *./part1-convnet*) which contains a list of libraries needed to set the environment for this assignment. You can use it to create a copy of conda environment. Refer to the users' manual for more details.

```
$ conda env create -f environment.yaml
```

Please note that our environment does **NOT have PyTorch Installation** for you because you may use CPU/GPU or a different version of CUDA. To install PyTorch, please refer to the official documentation and select the options based on your local OS.

If you already have your own Python development environment, please refer to this file to find necessary libraries, which are used to set the same coding/grading environment.

Jupyter Notebook

We recommend completing this assignment using the jupyter notebook provided. To launch,

```
$ jupyter notebook <name-of-notebook>
```

If you do not have access to GPU or Apple Silicon, we also suggest using google colab for part 2 of the assignment. To do this, simply upload the assignment folder to google drive and then open up the notebook file.

Code Test

There are two ways (steps) that you can test your implementation:

1. Python Unit Tests: Some public unit tests are provided in the `tests/` in the assignment repository. You can test each part of your implementation with these test cases by:

   ```
   $ python -m unittest tests.<name_of_tests>
   ```

   Or simply run the corresponding cells in the jupyter notebook. However, passing all local tests neither means your code is free of bugs nor guarantees that you will receive full credits for the coding section. Your code will be graded by GradeScope Autograder(see below for more details). There will be additional tests on GradeScope which are not present in your local unit tests.

2. Gradescope Autograder: You may also submit your code to gradescope for testing. Gradescope would only reveal the results of public tests. Your final grades of the coding section are based on both public and hidden private tests. However, we do not recommend using Gradescope as your primary testing method during development because the private test cases will **NOT** be available to you at any time.

1 Implementing CNN from Scratch

You will work in *./part1-convnet* for this part of the assignment. **Note that vectorization is not a requirement for this part of the assignment.**

1.1 Module Implementation

You will now learn how to build CNN from scratch. Typically, a convolutional neural network is composed of several different modules and these modules work together to make the network effective. For each module, you will implement a forward pass (computing forwarding results) and a backward pass (computing gradients). Therefore, your tasks are as follows:

(a) Follow the instructions in the code to complete each module in *./modules*. Specifically, modules to be implemented are 2D convolution, 2D Max Pooling, ReLU, and Linear. These will be the building blocks of the full network. The file *./modules/conv_classifier.py* ties each of the aforementioned modules together and is the subject of the next section.

1.2 Network Implementation

After finishing each module, it's time to put things together to form a real convolutional neural network. Your task is:

(a) Follow the instructions in the code to complete a CNN network in *./modules/conv_classifier.py*. The network is constructed by a list of module definitions **in order** and should handle both forward and backward communication between modules.

1.3 Optimizer

You have implemented a simple SGD optimizer in assignment-1. In practice, it is common to use a momentum term in SGD for better convergence. Specifically, we introduce a new velocity term v_t and the update rule is as follows:

$$\begin{aligned} v_t &= \beta v_{t-1} - \eta \frac{\partial L}{\partial w} \\ w &= w + v_t \end{aligned}$$

where β denotes the momentum coefficient and η denotes the learning rate

(a) Follow the instructions in the code to complete SGD with momentum in *./optimizer/sgd.py*. Hint: you will need to store and use the velocity from the previous iteration of SGD to compute the new gradient for the current iteration. Feel free to add member variable(s) to achieve this.

You might have noticed that the training process of your implementation can be extremely slow. Therefore, we only want to deliberately overfit the model with a small portion of data to verify whether the model is learning something or not.

Under the Experiments section in the jupyter notebook, we will train the CNN with only 50 samples in CIFAR-10 dataset. The script will make a plot on the training data only and **be sure to include the plot in your report**. Your final accuracy should be slightly under 0.9 with the given network in the script.

2 PyTorch

You will work in *./part2-pytorch* for this part of the assignment. The train function in *solver.py* contains the major logic of the code and you can run it by

```
solver = Solver(<args>)
solver.train()
```

Easiest way is to run the code snippet under the Pytorch Model section of the jupyter notebook.

2.1 Training

The first thing of working with PyTorch is to get yourself familiarized with the basic training step of PyTorch.

1. Read the PyTorch tutorial

2. Complete *_compute_loss_update_params* function in *solver.py.*

2.2 PyTorch Model

You will now implement some actual networks with PyTorch. We provide some starter files for you in *./models.* The models for you to implement are as follows:

1. Two-Layer Network. This is the same network you have implemented from scratch in assignment 1. You will build the model with two fully connected layers and a sigmoid activation function in between the two layers. Please implement the model as instructed in *./models/twolayer.py*

2. Vanilla Convolutional Neural Network. You will build the model with a convolution layer, a ReLU activation, a max-pooling layer, followed by a fully connected layer for classification. Your convolution layer should use **32 output channels**, a **kernel size of 7** with **stride 1** and **zero padding**. You max-pooling should use a **kernel size of 2** and **stride of 2**. The fully connected layer should have **10 output features**. Please implement the model as instructed in *./models/cnn.py*

3. Your Own Network. You are now free to build your own model. Notice that it's okay for you to borrow some insights from existing well-known networks, however, directly using those networks as-is is **NOT** allowed. In other words, you have to build your model from scratch, which also means using any sort of pre-trained weights is also **NOT** allowed. Please implement your model in *./models/my_model.py*

We provide you configuration files for these three models respectively. For Two-Layer Network and Vanilla CNN, you need to train the model without modifying the configuration file. The script automatically saves the weights of the best model at the end of training. We will evaluate your implementation by loading your model weights and evaluating the model on CIFAR-10 test data. You should expect the accuracy of Two-Layer Network and Vanilla CNN to be around 0.3 and 0.4 respectively.

For your own network, you are free to tune any hyper-parameters to obtain better accuracy. Your final accuracy must be above 0.5 to receive at least **partial credit**. Please refer to the GradeScope auto-test results for the requirement of full credits. **Try to keep your submission under 100mb or GradeScope may not accept it.** All in all, please make sure the checkpoints of each model are saved into *./checkpoints*.

3 Data Wrangling

So far we have worked with well-balanced datasets (samples of each class are evenly distributed). However, in practice, datasets are often not balanced. In this section, you will explore the limitation of standard training strategy on this type of dataset. This being an exploration, it is up to you to design experiments or tests to validate these methods are correct and effective.

You will work with an unbalanced version of CIFAR-10 in this section, and you should use the ResNet-32 model in *./models/resnet.py*.

3.1 Class-Balanced Focal Loss

You will implement one possible solution to the imbalance problem: Class-Balanced Focal Loss. this CVPR-19 paper: Class-Balanced Loss Based on Effective Number of Samples. You may also refer to the original paper of Focal Loss for more details if you are interested. Please implement CB Focal Loss in *./losses/focal_loss.py*.

NOTE: the CVPR-19 paper uses Sigmoid CB focal loss (section 4). Softmax CB focal loss is not described in the paper, but it is easy to derive from the mentioned papers. You must implement the softmax version to pass the tests.

Hint: Make sure you are using torch operations throughout your focal loss implementation otherwise the torch computation graph will not be built properly.

Now, follow the instructions in the report template to obtain the best results possible for Resnet with regular CE loss (you may need to perform extra hyperparameter tuning). Then experiment with the beta parameter. Finally, obtain the best possible results for Resnet using focal loss. You are welcome to change hyperparameters in the config file as necessary. Make sure you set the imbalance config parameter as appropriate.

4 Deliverables

4.1 Code Submission

4.1.1 Part-1 ConvNet

Simply run the last block in the colab notebook to generate the assignment submission. The zip file should contain modules/ and optimizer/. Be sure to upload the zip file (uncompressed) to gradescope for proper grading.

4.1.2 Part-2 PyTorch

Simply run the last block in the colab notebook to generate the assignment submission. The zip file should contain: configs/, losses/, checkpoints/, models/, and solver.py. **Make sure you include all configs used for the experiments for focal loss in case we want to verify your results in your report**.

4.2 Write-up

Please follow the report template in the starter folder to complete your write-up. You will need to explain your design of the network from Section 2.2 and explain your experimental results from the data wrangling section.

Note: Explanations should go into *why* things work the way they do with proper deep learning theory. If you need more than one slide for a question, you are free to create new slides **right after the given one**.

You will need to export your report in **pdf** format and submit under ”Assignment 2 Writeup” in Gradescope. **When submitting to Gradescope, make sure you select ALL corresponding slides for each question. Failing to do so will result in -1 point for each incorrectly tagged question, with future assignments having a more severe penalty.**