CS69000-DPL Spring 2020 Homework — Homework 3— Last update: February 10, 2020

https://www.cs.purdue.edu/homes/ribeirob/courses/Spring2020 Due: **6:00am**, Sunday, February 16th

Instructions and Policy: Each student should write up their own solutions independently, no copying of any form is allowed. You MUST to indicate the names of the people you discussed a problem with; ideally you should discuss with no more than two other people.
YOU MUST INCLUDE YOUR NAME IN THE HOMEWORK
You need to submit your answer in PDF. LaTeX is typesetting is encouraged but not required. Please write clearly and concisely - clarity and brevity will be rewarded. Refer to known facts as necessary.

Q0 (0pts correct answer, -1,000pts incorrect answer: (0,-1,000) pts): A correct answer to the following questions is worth 0pts. An incorrect answer is worth -1,000pts, which carries over to other homeworks and exams, and can result in an F grade in the course.

(1) Student interaction with other students / individuals:

  (a) I have copied part of my homework from another student or another person (plagiarism).

  (b) Yes, I discussed the homework with another person but came up with my own answers. Their name(s) is (are) ______________________________

  (c) No, I did not discuss the homework with anyone

(2) On using online resources:

  (a) I have copied one of my answers directly from a website (plagiarism).

  (b) I have used online resources to help me answer this question, but I came up with my own answers (you are allowed to use online resources as long as the answer is your own). Here is a list of the websites I have used in this homework:
  ____

  (c) I have not used any online resources except the ones provided in the course website.

1 Homework 2: Stochastic Gradient Descent and Adaptive Learning Rates

Learning Objectives: Let students understand Stochastic Gradient Descent (SGD) and optimization algorithms for adaptive learning rates.

Learning Outcomes: After you finish this homework, you should be capable of explaining and implementing SGD with minibatch from scratch and realize algorithms for adaptive learning rates.

1.1 Theoretical Part

Please answer the following questions **concisely**. All the answers, along with your name and email, should be clearly typed in some editing software, such as Latex or MS Word.

1. Minibatch Stochastic Gradient Descent (SGD) has been shown to give competitive results using deep neural networks in many tasks. Compared to the regular Gradient Descent (GD), the gradients computed by minibatch SGD are noisy. Can you use the Robbins-Monro stochastic approximation algorithm to explain why the noisy gradients need to be unbiased (i.e., in average, the noisy gradient is the true gradient). What would happen if the gradients had a bias?
2. In Robbins-Monro stochastic approximation the adaptive learning rate η_t must satisfy two conditions (a) $\sum_{t=1}^{\infty} \eta_t = \infty$ and (b) $\sum_{t=1}^{\infty} \eta_t^2 < \infty$. Explain what would happen with the approximation if condition (a) was not satisfied. Now explain what would happen with the approximation if condition (b) was not satisfied.
3. Give an intuitive definition of loss "flatness" in the parameter space that helps explain why the local minima found by SGD tends to generalize well in the test data.
4. Early stopping uses the validation data to decide which parameters we should keep during our SGD optimization. Explain why models obtained by early stopping tend to generalize better than models obtained by running a fixed number of epochs. Also explain why early stopping should never use the training data.
5. Give a mathematical reason to why CNNs are sensitive to image rotations. Describe the role of max-pooling on a CNN and why it should be applied over small image patches.
6. Explain why min-pooling is never applied to CNNs with ReLU activations.

1.2 Programming Part

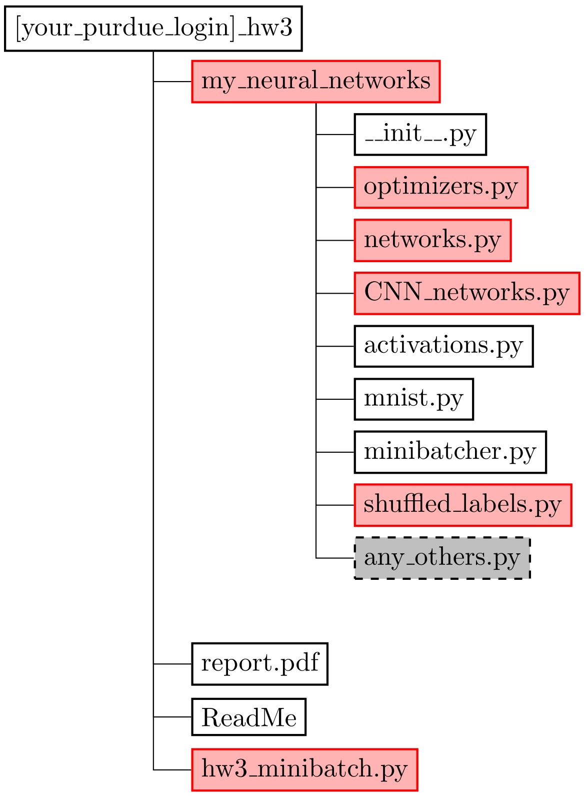
In this part, you are going to implement multiple (1) gradient descent variants and (2) regularization methods. This HW is built on top of HW2. In case that you did not work out HW2, we provide the skeleton code, which is basically the solution of HW2, so that you can start from a solid base. If you are confident of your own HW2 implementation, you are welcome to use it, but you need to make sure that it works well with the new executable provided. The rule of thumbs is that you can do any changes in the ”my_neural_networks,” but need to **keep the main executable, such as hw3_minibatch.py, untouched**.

Skeleton Package: A skeleton package is available on Piazza with the execution script hw3_minibatch.py You should be able to download it and use the folder structure provided, like the HW2.

1.2.1 HW Overview

You are going to add a few new components to the previous HW2 package. **optimizers.py** is a module that contains four learning algorithms (optimizers) that you will explore. Those optimizers should be called in **networks.py** for updating network parameters. Moreover, in **networks.py**, you will add L2 regularization. So spend most of your efforts on improving these two modules.

Again, we plot the folder structure that you should use:



- **[your_purdue_login]_hw3**: the top-level folder that contains all the files required in this homework. You should replace the file name with your name and follow the naming convention.
- **report.pdf**: Your written solutions to all the homework questions, including theoretical and programming parts. Should be submitted in pdf format.
- **ReadMe**: Your ReadMe should begin with a couple of **example commands**, e.g., ”python hw3.py data”, used to generate the outputs you report. TA would replicate your results with the commands provided here. More detailed options, usages and designs of your program can be followed. You can also list any concerns that you think TA should know while running your program. Note that put the information that you think it's more important at the top. Moreover, the file should be written in pure text format that can be displayed with Linux ”less” command.
- **hw3_minibatch.py**: The main executable to run training with minibatch SGD.
- **my_neural_networks**: Your Python neural network package. The package name in this homework is **my_neural_networks**, which should NOT be changed while submitting it. Two modules should be at least included:

  Two modules that you are going to develop:

  - **optimizers.py**
  - **networks.py**

The detail will be provided in the task descriptions. All other modules are just there for your convenience. It is not requried to use them, but exploring that will be a good practice of re-using code. Again, you are welcome to architect the package in your own favorite. For instance, adding another module, called utils.py, to facilitate your implementation.

We also recommend that you explore **minibatcher.py**, because you could use it to creates minibatches. You are also welcome to do it completely by yourself.

1.2.2 Data: MNIST

You are going to conduct a simple classification task, called MNIST (`http://yann.lecun.com/exdb/mnist/`). It classifies images of hand-written digits (0-9). Each example thus is a 28×28 image.

- The full dataset contains 60k training examples and 10k testing examples.
- We provide a data loader (read_images(.) and read_labels(.) in **my_neural_networks/mnist.py**) that will automatically download the data.

1.2.3 Gradient Descent Variants

Gradient Descent does its job decently when the dataset is small. However, that is not the case we see in Deep Learning. Multiple variants of learning algorithms have been proposed to realize training with huge amount of data. We've learned several in the class. Now it's time to test your understanding about them.

Task 1a: Minibatch Stochastic Gradient Descent (SGD)

This is a warm-up task. You are going adapt your HW2 from full-batch to minibatch SGD. Most of the codes are given. Only the minibatch training requires some of your efforts.

In Minibatch SGD, the gradients come from minibatches:

$$L(W) = \frac{1}{N_b} \sum_{i=1}^{N_b} L_i(x_i, y_i, W) \tag{1}$$

$$\nabla_W L(W) = \frac{1}{N_b} \sum_{i=1}^{N_b} \nabla_W L_i(x_i, y_i, W), \tag{2}$$

where N_b is the number of examples in one batch. It is similar to the full-batch case, but now you only feed a subset of the training data and update the weights according to the gradients calculated from this subset.

Related Modules:

- hw3_minibatch.py
- my_neural_networks/optimizers.py
- my_neural_networks/networks.py

Action Items:

1. Go through all the related modules. Specifically, you should understand the **networks.BasicNeuralNetwork** well. Note that the "train_one_epoch" method is renamed as "train_one_batch" to accommodate our purpose, but does the same thing: consider all the input examples and update the weights.
2. You should notice that we move the weight updates to the **optimizers.py**. The **optimizers.SGDOptimizer** is fully implemented. You should look into it and understand its interactions with the network. Inside **optimizers.py** you will find a basic skeleton for your implementation. You are required to fill in the missing parts of the code, denoted with ⟨fill in⟩.
3. Now, move to **hw3_minibatch.py**. This file basically has the same structure as the executable in HW2. We only did minor changes to adapt to the library.
   (a) In this file, **implement the training part using minibatch SGD** (You should see the comments that ask you to fill in your code).
   (b) There is a "–minibatch_size" command-line option that specifies that minibatch size. And you should use the "networks.BasicNeuralNetwork.train_one_batch(.)" for training one batch.
   (c) Once you finish the implementation, run your code with the command "python hw3_minibatch.py -n 20000 -m 300 -v data" (supposed your corpus is in a folder called "data"), which specifies that minibatch size is 300 and in default the maximum number of epochs is 100.
4. Report your results by filling up Table 1 for **losses, training accuracies, and testing accuracies** (You should have 3 tables). Use the batch sizes and learning rates given and report the results in the final epoch. Describe your observations in the report.
5. The hw3_minibatch.py can output two plots: "Loss vs. Epochs" and "Accuracies vs. Epochs," if you collect the results correctly in the code. Make use of those plots for debugging and analyzing the models.
6. Running the program without specifying the batch size (remove the "–minibatch 300" option) gives you the regular Gradient Descent (GD). Compare the results of using regular GD and minibatch SGD, and state your observations in the report.

BatchSize \ LearningRate	1e-3	1e-4	1e-5
100			
500			
3000			
5000			

Table 1: The results should be reported.

Task 1b: Adaptive Learning Rate Algorithms

You should have learned the adaptive learning rate algorithms in the lecture. In addition to SGD, here we are going to implement several more: **Momentum, Nesterov Momentum, and Adam**.

Note that you can still report the results with full-batch training if you skipped the minibatch SGD implementation in Task 1a.

Since each algorithm might have minor versions, here we define the exact one we want you to implement:

- **SGD**: (You already have it. Just for reference). For each parameter x_t at the iteration t, you update it with:

$$x_{t+1} = x_t - \alpha \nabla f(x_t), \tag{3}$$

  where α is the learning rate.

- **Momentum**: for each parameter x_t at the iteration t, and the corresponding velocity v_t, we have

$$v_{t+1} = \rho v_t + \alpha \nabla f(x_t) \tag{4}$$
$$x_{t+1} = x_t - v_{t+1}, \tag{5}$$

  where α is the learning rate, ρ is the hyperparameter for the momentum rate. A common default option for ρ is 0.9.

- **Nesterov Momentum**: for each parameter x_t at the iteration t, and the corresponding velocity v_t, we have

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t) \tag{6}$$
$$x_{t+1} = x_t + v_{t+1}, \tag{7}$$

  where α is the learning rate, ρ is the hyperparameter for the momentum rate. A common default option for ρ is 0.9.

- **Adam**: for each parameter x_t at the iteration t, we have

$$m_1 = \beta_1 * m_1 + (1-\beta_1)\nabla f(x_t) \tag{8}$$

$$m_2 = \beta_2 * m_2 + (1-\beta_2)(\nabla f(x_t))^2 \tag{9}$$

$$u_1 = \frac{m_1}{1-\beta_1^t} \tag{10}$$

$$u_2 = \frac{m_2}{1-\beta_2^t} \tag{11}$$

$$x_{t+1} = x_t - \alpha\frac{u_1}{(\sqrt{u_2}+\epsilon)}, \tag{12}$$

  where α is the learning rate, m_1 and m_2 are the first and second moments, u_1 and u_2 are the first and second moments' bias correction, and β_1, β_2, and ϵ are hypterparameters. A set of common choices of the parameters are $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-5}$, and $\alpha = 1e-4$. We initialize $m_1 = m_2 = 0$.

Related Modules:

- my_neural_networks/optimizers.py

Action Items:

1. The corresponding class prototypes for the three algorithms are given in the skeleton. You should finish each of them.
2. There is a "–optimizer" command-line option in hw3_minibatch.py. So you can test your optimizers individually with the following commands:
   - "python hw3_minibatch.py -n 20000 -m 300 –optimizer sgd data"
   - "python hw3_minibatch.py -n 20000 -m 300 –optimizer momentum data"
   - "python hw3_minibatch.py -n 20000 -m 300 –optimizer nesterov data"
   - "python hw3_minibatch.py -n 20000 -m 300 –optimizer adam data"
3. Plot "Loss vs. Epochs" and "Accuracies vs. Epochs" for each algorithm (include SGD) and attach them to the report. You should have 8 plots in total.
4. Describe your observations from the plots in the report. Make algorithm comparisons.

1.2.4 Regularization

You will be implementing two common regularization methods in neural networks and analyze their difference.

Note that you can still report the results with full-batch training if you skipped the minibatch SGD implementation in Task 1a.

Task 2a: L2-Regularization

The first method is L2-Regularization. It is a general method that not only works for neural networks but also for many other parameterized learning models. It has a hyperparameter for determining the weight of the L2 term, i.e., it is the coefficient of the L2 term. Let's call it λ. In this task, you should implement the L2-regularization and conduct hyperparameter tuning for λ. Think about what should be added to the gradients when you have a L2-Regularization term in the objective function.

Here we give the mathematical definition of the L2-regularized objective function that you should look at:

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W) + \lambda R(W) \tag{13}$$

$$R(W) = \sum_k \sum_j W_{k,j}^2, \tag{14}$$

where λ is a hyperparameter to determine the weight of the regularization part. You should think about how to change the gradients according to the added regularization term.

Related Modules:

- my_neural_networks/networks.py

Action Items:

1. There is a command-line option "–l2_lambda", which specifies the coefficient of the L2 term. In defaut, it is zero, which means no regularization.
2. Add L2-Regularization to your **BasicNeuralNetwork** with respect to the command-line option. You need to make changes for all the methods to support L2-regularization, especially for the "train_one_epoch(.)" and "loss(.)".
3. Test your network with the following commands:
   - "python hw3_minibatch.py -n 20000 -m 300 –l2_lambda 1 data"
   - "python hw3_minibatch.py -n 20000 -m 300 –l2_lambda 0.1 data"
   - "python hw3_minibatch.py -n 20000 -m 300 –l2_lambda 0.01 data"
4. Plot "Loss vs. Epochs" and "Accuracies vs. Epochs" for each lambda value, and include them in the report. You should have 6 plots.
5. Describe your observations from the plots in the report. Make comparisons.

Task 3: Implement a CNN

In this task we will implement a CNN and its task to better understand its inner workings.

Related Modules:

- hw3_minibatch.py
- my_neural_networks/optimizers.py
- (to create) my_neural_networks/CNN_networks.py
- (to create) my_neural_networks/shuffled_labels.py

Action Items:

1. (in the code) Follow the the pytorch implementation provided at the end of lecture `https://www.cs.purdue.edu/homes/ribeirob/courses/Spring2020/lectures/05/optimization.html` to create **CNN_networks.py** using the default parameters given in the lecture.
   Optimize using the standard SGD. Modify **hw3_minibatch.py** so it will ran the CNN code with parameter `--CNN`
   (in the PDF) Describe the neural network architecture and its hyper-parameters: layers, type of layer, number of neurons on each layer, the activation functions used, and how the layers connect.

2. For a $k \times k$ filter, the CNN considers $k \times k$ image patches. These image patches overlap according to stride, which is by how much each block must be separated horizontally and vertically. If k is not a multiple of the image height of width, we will need padding (increase image height (width) by adding a row (column) of zeros). Modify these filters to (a) 3×3 with stride 3, and (b) 14×14 with stride 1. In the PDF, show the test accuracy of the classifier over training and test data for items (a) and (b). (you do not need to include your code). Discuss your findings. Specifically, what could be the issue of having (a) 3×3 filters with stride 3 and (b) 14×14 filters?
   Hint: Zero-pad as necessary to keep the input and output of the convolutional layers of the same dimensions.

3. Deep neural networks generalization performance is not related to the network's inability to overfit the data. Rather, it is related to the solutions found by SGD. In this part of the homework we will be testing that hypothesis. Please use **100 epochs** in the following questions. Please implement this part of the code in a separate file **shuffled_labels.py**.

   (a) Using the provided code, show a plot (in the PDF) with two curves: the training accuracy and validation accuracy, with the x-axis as the number of epochs.

   (b) Now consider an alternative task: randomly shuffle the target labels, so that the image (hand-written digit) and its label are unrelated. Show a plot with two curves: the training accuracy and validation accuracy, with the x-axis as the number of epochs. What Modify **hw3_minibatch.py** so it will ran this shuffle-label CNN experiment with parameter `--shuffleCNN`. What can you conclude about the ability of this neural network to overfit the data? Would you say that the inductive bias of a CNN can naturally "understand images" and will try to classify all hand-written "0"s as the same digit?

(c) Using the approach to interpolate the initial (random) and final parameters seen in class `https://www.cs.purdue.edu/homes/ribeirob/courses/Spring2020/lectures/05/optimization.html`
show the “flatness” plot of the original task and the task with shuffled labels over the training data. Can you conclude anything from these plots about the possible generalization performance of the two models?

Submission Instructions

Please read the instructions carefully. Failed to follow any part might incur some score deductions.

Naming convention: [firstname]_[lastname]_hw3

All your submitting files, including a report, a ReadMe, and codes, should be included in one folder. The folder should be named with the above naming convention. For example, if my first name is ”Bruno” and my last name is ”Ribeiro”, then for Homework 3, my file name should be “bruno_ribeiro_hw3”.

Tar your folder: [firstname]_[lastname]_hw3.tar.gz

Remove any unnecessary files in your folder, such as training datasets. Make sure your folder structured as the tree shown in Overview section. Compress your folder with the the command: **tar czvf bruno_ribeiro_hw3.tar.gz czvf bruno_ribeiro_hw3** .

Submit: **TURNIN INSTRUCTIONS**

Please submit your compressed file on **data.cs.purdue.edu** by turnin command line, e.g. **”turnin -c cs690dl -p hw3 bruno_ribeiro_hw3.tar.gz”**. Please make sure you didn’t use any library/source explicitly forbidden to use. If such library/source code is used, you will get 0 pt for the coding part of the assignment. If your code doesn’t run on scholar.rcac.purdue.edu, then even if it compiles in another computer, your code will still be considered not-running and the respective part of the assignment will receive 0 pt.

1.3 Grading Breakdown

- Theoretical: 30%
- Programming: 70%