EEE598: Statistical Machine Learning: From Theory to Practice

Instructor: Dr. Lalitha Sankar
Homework Assignment #5
Total points: 100

Due Date and Time: 12/4/2022 11:59 PM

Student Name: _____ ASU Id: _____

Instructions:

(1) Every problem should begin on a new page (use the \newpage command).

(2) Page numbers for each problem should be entered in Gradescope when uploading HW

(3) Furthermore, **page numbers for every sub-problem within a problem should also be entered in Gradescope**.

(4) Points are given for steps and work – show all work and steps and clarifications/arguments for the steps

(5) Some rules on Python code files:

(i) Filename convention: For every problem, the corresponding Python filename should follow the following nomenclature: ⟨FirstInitialLastName-HWxx-Problem#Subproblem#⟩. For example, for HW3, problem 7a, my Python file should have the name: LSankar-HW3-7a.xxx

(ii) All subproblems can be in one file but the FILENAME has to be clarifying as in (i) above.

(iii) All Python files generated for an HW can be zipped and uploaded but again, if filenames are misleading then points will be deducted.

(iv) All code will also be checked for complete overlap with other student's code.

(v) **Every file should be independently executable**.

(6) Enjoy!

Some instructions on how to submit solutions on Gradescope and Canvas:

(1) **All plots have to be included in the PDF file uploaded to Gradescope**. It is not the TA's or grader's job to find it on the canvas submission. Plots not in the PDF submission on Gradescope will NOT graded.

(2) **Plots without legend on what you are plotting them for will lead to points being deducted**. Keep in mind you have several plots and it is important for us to know what these plots correspond to.

(3) **Canvas submission should be restricted to only CODE**.

(4) **Please include a copy of your code in your Gradescope submission** – this can make grading easier for us as a first run (before we run your code using the canvas upload). This is particularly helpful if we find that your basic code is correct and your results are correct.

Additional Notes:

a. Please review all homework guidance above before submitting to Gradescope.

b. Please provide succinct answers along with succinct reasoning for all your answers. Points may be deducted if long answers demonstrate a lack of clarity.

c. Similarly, when discussing the experimental results, concisely create tables and/or figures when appropriate to organize the experimental results. In other words, **all your explanations, tables, and figures for any particular part of a question must be grouped together**.

d. **Late homework submissions will not be accepted after the deadline.**

Deep learning architectures

1. In this problem we will explore different deep learning architectures for a classification task using the CIFAR-10 dataset. Go to `http://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html` and complete the following tutorials

- *What is PyTorch?*
- *Autograd: automatic differentiation*
- *Neural Networks*
- *Training a classifier*

The final tutorial will leave you with a network for classifying the CIFAR-10 dataset, which is where this problem starts. Just following these tutorials could take a number of hours but they are excellent, so start early. After completing them, you should be familiar with tensors, two-dimensional convolutions (`nn.Conv2d`) and fully connected layers (`nn.Linear`), ReLu non-linearities (`F.relu`), pooling (`nn.MaxPool2d`), and tensor reshaping (`view`); if there is any doubt of their inputs/outputs or whether the layers include an offset or not, consult the API `http://pytorch.org/docs/master/`.

A few preliminaries:

- Using a GPU may considerably speed up computations but it is not necessary for these small networks (one can get away with using one's laptop).
- Conceptually, each network maps an image $x^{in} \in \mathbb{R}^{32\times32\times3}$ (3 channels for RGB) to an output layer $x^{out} \in \mathbb{R}^{10}$ where the image's class label is predicted as $\arg\max_{i=0,1,\ldots,9} x_i^{out}$. An error occurs if the predicted label differs from its true label.
- In this problem, the network is trained via cross-entropy loss, the same loss we used for multi-class logistic regression. Specifically, for an input image and label pair (x^{in}, c) where $c \in \{0, 1, \ldots, 9\}$, if the network's output layer is $x^{out} \in \mathbb{R}^{10}$, the loss is $-\log(\frac{\exp(x_c^{out})}{\sum_{c'=0}^{9}\exp(x_{c'}^{out})})$.
- For computational efficiency reasons, this particular network considers *mini-batches* of images per training step meaning the network actually maps $B = 4$ images per feed-forward so that $\widetilde{x}^{in} \in \mathbb{R}^{B\times32\times32\times3}$ and $\widetilde{x}^{out} \in \mathbb{R}^{B\times10}$. This is ignored in the network descriptions below but it is something to be aware of.
- The cross-entropy loss is a convex loss; however, when composed with the hypothesis class consisting of neural networks, the resulting function is not convex. This means that the optimization method may converge to different *local minima* based on different hyperparameters of the optimization procedure (e.g., stepsize). Usually one can find a good setting for these hyperparameters by just observing the relative progress of training over the first epoch or two (how fast is it decreasing) but you are warned that early progress is not necessarily indicative of the final convergence value (you may converge quickly to a poor local minimum whereas a different step size could have poor early performance but converge to a better final value).
- The training method used in this example uses a form of stochastic gradient descent (SGD) that relies on a technique called *momentum*; this technique incorporates scaled versions of previous gradients into the current descent direction[1]. Practically speaking, momentum is another optimization hyperparameter in addition to the step size. If this bothers you, you can obtain all the same results using regular stochastic gradient descent.
- We will not be using a validation set for this exercise. Hyperparameters like network architecture and step size should be chosen based on the performance on the test set. This is very bad practice for all the reasons we have discussed over the semester, but we aim to make this exercise as simple as possible.
- You should modify the training code such that at the end of each epoch (one pass over the training data) you compute and print the training and test classification accuracy (you may find the running calculation that the code initially uses useful to calculate the training accuracy).
- While one would usually train a network for hundreds of epochs for it to converge, this can be prohibitively time consuming so feel free to train your networks for just a dozen or so epochs.

[1] See `http://www.cs.toronto.edu/~hinton/absps/momentum.pdf` for the deep learning perspective on this method.

You will construct a number of different network architectures and compare their performance on the CIFAR-10 dataset. For all, it is highly recommended that you copy and modify the existing (working) network you are left with at the end of the tutorial *Training a classifier*. For all of the following perform a hyperparameter selection (manually by hand, random search, etc.) using the test set, report the hyperparameters you found, and plot the training and test classification accuracy as a function of iteration (one plot per network).

Here are the network architectures you will construct and compare.

(a) *[30 points]* Fully connected output, 0 hidden layers (logistic regression): we begin with the simplest network possible that has no hidden layers and simply linearly maps the input layer to the output layer. That is, conceptually it could be written as

$$x^{out} = W\text{vec}(x^{in}) + b$$

where $x^{out} \in \mathbb{R}^{10}$, $x^{in} \in \mathbb{R}^{32\times32\times3}$, $W \in \mathbb{R}^{10\times3072}$, $b \in \mathbb{R}^{10}$ where $3072 = 32 \cdot 32 \cdot 3$. For a tensor $x \in \mathbb{R}^{a\times b\times c}$, we let $\text{vec}(x) \in \mathbb{R}^{abc}$ be the reshaped form of the tensor into a vector (in an arbitrary but consistent pattern).

(b) *[30 points]* Fully connected output, 1 fully connected hidden layer: we will have one hidden layer denoted as $x^{hidden} \in \mathbb{R}^M$ where M will be a hyperparameter you choose (M could be in the hundreds). The nonlinearity applied to the hidden layer will be the relu ($\text{relu}(x) = \max\{0, x\}$, elementwise). Conceptually, one could write this network as

$$x^{out} = W_2\text{relu}(W_1\text{vec}(x^{in}) + b_1) + b_2$$

where $W_1 \in \mathbb{R}^{M\times3072}$, $b_1 \in \mathbb{R}^M$, $W_2 \in \mathbb{R}^{10\times M}$, $b_2 \in \mathbb{R}^{10}$.

(c) *[40 points]* Fully connected output, 1 convolutional layer with max-pool: for a convolutional layer W_1 with individual filters of size $p \times p \times 3$ and output size M (reasonable choices are $M = 100$, $p = 5$) we have that $\text{Conv2d}(x^{in}, W_1) \in \mathbb{R}^{(33-p)\times(33-p)\times M}$. Each convolution will have its own offset applied to each of the output pixels of the convolution; we denote this as $\text{Conv2d}(x^{in}, W) + b_1$ where b_1 is parameterized in $\mathbb{R}^M$. We will then apply a relu (relu doesn't change the tensor shape) and pool. If we use a max-pool of size N (a reasonable choice is $N = 14$ to pool to 2×2 with $p = 5$) we have that $\text{MaxPool}(\text{relu}(\text{Conv2d}(x^{in}, W_1)+b_1)) \in \mathbb{R}^{\lfloor\frac{33-p}{N}\rfloor\times\lfloor\frac{33-p}{N}\rfloor\times M}$. We will then apply a fully connected layer to the output to get a final network given as

$$x^{output} = W_2\text{vec}(\text{MaxPool}(\text{relu}(\text{Conv2d}(x^{input}, W_1) + b_1))) + b_2$$

where $W_2 \in \mathbb{R}^{10\times M(\lfloor\frac{33-p}{N}\rfloor)^2}$, $b_2 \in \mathbb{R}^{10}$. The parameters M, p, N (in addition to the step size and momentum) are all hyperparameters.

(d) (Extra credit: *[10 points]*) Returning to the original network you were left with at the end of the tutorial *Training a classifier*, tune the different hyperparameters (number of convolutional filters, filter sizes, dimensionality of the fully connected layers, stepsize, etc.) and train for many epochs to achieve a *test accuracy* of at least 87%.

The number of hyperparameters to tune in the last exercise combined with the slow training times will hopefully give you a taste of how difficult it is to construct good performing networks. It should be emphasized that the networks we constructed are **tiny**; typical networks have dozens of layers, each with hyperparameters to tune. Additional hyperparameters you are welcome to play with if you are so interested: replacing relu $\max\{0, x\}$ with a sigmoid $1/(1 + e^{-x})$, max-pool with average-pool, and experimenting with batch-normalization or dropout.