Introduction to Deep Learning for Computer Vision
Assignment 3: Simple Linear Classifier II

Due: January 31, 2019

Abstract

This assignment is the second part of the pure `Python` pipeline for training a simple linear classifier, without the help of Deep Learning libraries such as PyTorch. For this part, we will implement the gradient and the training loop, as well as the cross validation process.

Please read the instructions carefully.

1 Instructions

1.1 Submission package

Within the homework assignment package, there should be a `submission-package.zip`, which contains the directory structure and empty files for you to get started. Please edit the contents within the file for code, and then create a `zip` archive with the file name `submission-package.zip`, and submit it. **Do not use other archive formats such as `rar` or `tar.gz`.** Also, submit the report in `pdf`, **as a separate file alongside** the archive with the codes.

All assignments should be submitted electronically. Hand written reports are **not** accepted. You can, however, include scanned pages in your report. For example, if you are not comfortable with writing equations, you can include a scanned copy.

1.2 Assignment Report

For this assignment, all execution results should be summarised in an assignment report. Reports should be in `pdf` format. Though not mandatory, students are encouraged to submit their reports written in LaTeX. In the assignment package, you should have been given an empty skeleton file for you to get started.

However, it is **not required** for you to explain your code in the reports. Reports are for discussing results. You **should** however, provide comments in your code, well enough to be understood by directly reading the code.

1.3 Code

All assignments should be in `Python` 3. Codes that fail to run on `Python` 3 will receive 20% deduction on the final score. In other words, do **not** use `Python` 2.7.

For this assignment, you should **not** need to create additional files. Fill in the skeleton files in the submission package. Do **not** change the name of these scripts.

It is **strongly encouraged** to follow `PEP8`. It makes your code much more readable, and less room for mistakes. There are many open source tools available to automatically do this for you.

1.4 Delayed submission

In case you think you will not meet the deadline due to network speed or any other reasons, you can send an email with the `SHA-256` hash of your `.zip` archive first, and then submit your assignment through email later on. This will **not** be considered as a delay.

Delayed submissions are subject to 20% degradation per day. For example, an assignment submitted 1 minute after the deadline will receive 80% of the entire mark, even if it was perfect.

Submission will close 24 hours after the original deadline, as the solution for the assignment will be released to be used for the next assignment.

1.5 Use of open source code

Any library under any type of open source license is allowed for use, given full attribution. This attribution should include the name of the original author, the source from which the code was obtained, and indicate terms of the license. Note that using copyrighted material without an appropriate license is not permitted. Short snippets of code on public websites such as StackOverflow may be used without an explicit license, but proper attribution should be given even in such case. This means that if you embed a snippet into your own code, you should properly cite it through the comments, and also embed the full citation in a LICENSES file. However, if you include a full, unmodified source, which already contains the license within the source file, this is unnecessary. Please note that without proper attribution, *it will be considered plagiarism.*

In addition, as the assignments are intended for you to learn, (1) if the external code implements the core objective of the task, no points will be given; (2) code from other CSC486B/CSC586B students will count as plagiarism. To be more clear on (1), with the assignment, we will release a `requirements.txt` file, that you can use with `pip` to setup your environment. On top, you are also allowed to use `OpenCV3.X`, which we will not include in `requirements.txt` as `OpenCV` installation depends on your own framework.

2 Implementing linear classifiers (Continued)

2.1 Multiclass SVM (0 points)

This part has already been done for you, and have been explained in class. Implementation is provided in `utils/linear_svm.py`. Do use this example later on when you compare it with multinomial logistic regression.

2.2 Multinomial logistic regression (cross entropy)

2.2.1 Gradient function: `model_grad` (20 points)

Derive the gradients and implement it accordingly. **The derivation should go into your report**—see later section for exact things to put in report. Also, be sure to ensure numerical stability as above where needed.

Hint When you derive the gradient, you should end up with a form that includes only the "probabilities", `x`, and `y`. Also, if you introduce a "target" value for these probabilities, you can re-write the loss function as,

$$L_i = -\log \frac{\sum_j t_{ij} e^{s_{ij}}}{\sum_j e^{s_{ij}}} \quad , \tag{1}$$

where t_{ij} is 1 if $j = y_i$, and 0 if $j \neq y_i$. Well actually, if we denote the probabilities as $p_{ij} = \frac{e^{s_{ij}}}{\sum_k e^{s_{ik}}}$, the above equation becomes even more simpler. You can write it only with p_{ij} and t_{ij}. Here, note that p_{ij} is nothing but a standard `softmax` operation, and its derivative can easily be found all over the web. You'll still have to write the derivation in the report.

Useful functions

- `numpy.zeros`
- `numpy.arange`
- `numpy.reshape`
- `numpy.mean`
- `len`

3 The training and cross validation loop

With the functions defined above, we should now be able to implement the training pipeline, that is, the main script and the functions that we will run to see how our classifier performs.

3.1 Parsing command line arguments (0 points)

Note that we use `argparse` for obtaining command line arguments. See `config.py` for more details. You are free to add any additional arguments, but do **not** remove any of the arguments there.

3.2 Implement `train` (30 points)

Implement the `train` function in `solution.py` according to the function definitions and the pseudo code written in the comments.

While implementing the `train`, note that we use `compute_loss` and `compute_grad` to use the proper module in a lazy import fashion. This becomes quite handy if you later want to write a deep learning framework because you want to test various configurations in an automated fashion.

Hints A large portion of the code is already written. You may take them out and change them if you want, but they are there to help you out. Do try to study why they are there and where they can be used.

Useful functions

- `numpy.random.rand`
- `numpy.zeros`
- `numpy.mean`

3.3 Implement `main` (30 points)

In `solution.py`, `main`, according to the comments. Note that we have two modes for running the main function, defined by `--cross_validate`.

- `numpy.random.permutation`
- `numpy.max`

4 Reporting

As analyzing the results we get is the final goal, reporting is also a critical element in research. For this assignment, you will have to report the following mandatory items.

4.1 Abstract (5 points)

Write a brief abstract summarizing the assignment, as well as your discoveries.

4.2 Derivation of the gradient for cross entropy (5 points)

Provide derivation of the gradient for multinomial logistic regression (cross entropy).

4.3 Training curve (5 points)

Report your loss curve and the accuracy curve for the `HOG` feature option. Do **not** use cross validation (too slow!). Also report the test performance of the models with best validation accuracy.

Note that example results were provided to you in Lecture 9.

4.4 Cross validation results (5 points)

Report the cross validation results for at least two different hyper-parameter setups for the `HOG` feature. You can do more, but it is nor required. The current model has mainly two hyper-paremeters that can be tuned, that is, the learning rate and the regularization strength.

Hints You can also write an additional loop containing `main` to do hyperparameter comparison automatically. While this is out of the scope of this assignment evaluation, it might be a good idea to do that. If implemented, you just have to run your experiments, and come back later to check the results.