Introduction to Deep Learning for Computer Vision
Assignment 2: Simple Linear Classifier

Due: February 9, 2018

Abstract

This assignment is aimed at creating a pure `Python` pipeline for training a simple linear classifier, without the help of Deep Learning libraries such as TensorFlow. It is intended to make you practice what you learned in lectures by implementing them. Large portion of the implementation is already done for you to make it easier.

Instructions are provided first for a reason.
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 replace the `pdf` file with your report. You can then simply make a `zip` archive and submit it as your submission.

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

Even for programming assignments, all results should be summarised in a 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. Likewise, an assignment that was submitted one day and 1 minute after the deadline will receive 60%.

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.

Please note that the requirements.txt has been updated from assignment1

2 Preparing the input data

In this part of the assignment, we will focus on the early stages of the pipeline. We will download the dataset, and extract three types of features. Later, we will try all of them out to figure out which is indeed the best feature to use for our task.

2.1 Downloading and preparing CIFAR10 (0 points)

We've already discussed in Lecture 8 about the input pipeline. Please see Lecture Slides 8 for details on how to obtain CIFAR10 data.

2.2 Preparing the raw RGB data (0 points)

RGB data from CIFAR 10 is in the order of Channel, Height, and Width, or CHW ("channels first") for short. For many image processing libraries, we need to swap them so that they are in the form of Height, Width, and Channel order, or HWC ("channels last").

Especially, later on, when you use Tensorflow, by default tensorflow uses NHWC, where N is the sample dimension. So you want to get used to this format. For some libraries that are specialized for GPUs, it is also common to have NCHW, which is the format that CIFAR10 is in, so it really depends on your library of choice. Anyways, let's stick to NHWC for now.

`load_data` function is already provided for you in `utils/cifar10.py`. Please **do** have a look at it before you proceed any further, so that you understand what is going on.

2.3 Extracting color histograms in HSV space (10 points)

In `utils/features.py`, implement the function `extract_h_histogram` that takes a color image, converts it to HSV and uses the Hue value only to create the histogram with 16 bins. Again, the follow the function definitions in the skeleton file.

When extracting the Hue histogram, you **do not have to** use the weighted vote strategy discussed during the lecture. A simple histogram is sufficient.

Hints Converting color spaces are in many cases already implemented in your library. This is also true for the histogram generation. You probably don't need to do all that yourself. In addition, according to how you load the data, that is, which library you use, you will have either `RGB` or `BGR`. Be careful which space you use for conversion.

Useful functions

- `skimage.color.rgb2hsv`
- `numpy.histogram`
- `numpy.linspace`

2.4 Extracting histogram of oriented gradients (0 points)

This part should have already been done for you in the assignment package as an example. See `extract_hog` in `utils/features.py`.

2.5 Pre-processing (5 points)

Implement the `normalize` function according to the function specs in `utils/preprocess.py`. After you extract features, you might want to use them in combination, which we will probably will do at some point, but not for this assignment. However, to be prepared and to make this good habit stick, we will normalize the input data so that it is zero mean and its min/max is within -1 and 1.

Hints With `Python`, you can compare with the operator `==` or `is`. Especially when comparing with `None` the behavior is very different. Your code will probably run fine with either of them for this assignment, but things could get nasty if you are not careful.

Useful functions

- `numpy.mean`
- `numpy.abs`
- `numpy.max`

3 Implementing linear classifiers

3.1 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.

3.2 Multinomial logistic regression (cross entropy)

Similar to the SVM case, in `utils/logistic_regression.py`, implement the function that computes the loss, that is `model_loss`, and the function `model_grad` that computes the parameter updates. The prediction function `model_predict` is actually identical to the SVM case, so that one is already done for you.

3.2.1 Loss function: `model_loss` (5 points)

Implementing this function should be straightforward. Implement the multinomial logistic regression loss from Lecture 6, which is defined as follows:

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

where i is the sample index. Or equivalently,

$$L_i = -\log \frac{e^{s_{iy_i}-C_i}}{\sum_j e^{s_{ij}-C_i}} \quad , \tag{2}$$

where $C_i = \max_j s_{ij}$ to ensure numerical stability.

Hint For loops in `Python` are nasty. See how it's done in the SVM case and try doing it in a similar way. In fact, it should even be simpler than how it's done for SVM. Also for the `loss_c`, you want to probably save the "probabilities" for being each class for all samples, that is, the one that goes in the log. This, `x`, and `y` should be the only things that you need when you compute the gradient later on.

Useful functions

- `numpy.max`
- `numpy.sum`
- `numpy.exp`

3.2.2 Gradient function: `model_grad` (20 points)

Derive the gradients and implement it accordingly. 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,

$$L_i = -\log \frac{\sum_j t_{ij} e^{s_{ij}}}{\sum_j e^{s_{ij}}} \quad , \tag{3}$$

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`

4 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.

4.1 Parsing command line arguments (0 points)

From now on, we will 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.

4.2 Implement `predict` (5 points)

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

4.3 Implement `train` (20 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`, use `compute_loss` and `compute_grad` to use the proper module in a lazy import fashion. These two function are already implemented for you.

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.

4.4 Create train/validation splits (5 points)

In `solution.py`, `main`, pseudo code is written for the generation of train and validations splits to do cross validation. Implement this.

4.5 Perform cross validation (10 points)

After the generation of the train/validation splits, train your linear classifier using these splits to get an average performance of your classifier given the current set of hyper parameters and design choices. Again, implement this according to the pseudo code in `main`.

5 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.

5.1 Abstract (5 points)

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

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

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

5.3 Cross validation results (10 points)

Report how your cross validation experiments went. Discuss how the trend was, for example for different values of the regularizor strength, for each loss. Discuss also which feature gave you the best results, and possible pros and cons. Finally, discuss which setup was gave you the best results.

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. Since then, you just have to run your experiments, and come back later to check the results.