COMP0090 19/20 Assignment 1: “Bag, not bag”

Pontus SAITO STENETORP
p.stenetorp@cs.ucl.ac.uk

1 Instructions

For this assignment the maximum number of points obtainable is 100, which maps one-to-one for the mark given for the assignment. The assignment are to be carried out in groups of *up to five students*, which may or may not be the same groups as those for future assignments. For the assignment you will use Colaboratory[1] and the Julia programming language.[2] *You are not to use any automatic differentiation packages* (such as Flux or KNet), you are however free to use any of the (sloppy) code provided in the lecture notebooks, packages that do not implement automatic differentiation, or to implement automatic differentiation on your own. The assignment is *due by Tuesday, November 19th 2019 at 12:00 (UTC+00:00)*.

Your submission *shall*:

1. Take the form of *a single cover page* PDF document, to which you should *append* a printout of your Colaboratory notebook (you can achieve this using for example PDFtk[3]). The final document is to be submitted via Moodle.
2. List all group members clearly on the cover page and provide their UCL e-mails.
3. Provide a hyperlink on the cover page to *a single* Colaboratory notebook, that *must not* be edited after the submission deadline (it may be wise to make a separate copy of your final notebook and name it something akin to “Submission for COMP0090 Assignment 1”). We expect the notebook to be executable and to have a total runtime of about an hour or two.
4. Describe briefly on the cover page (about a paragraph) which group member contributed to which part to the assignment. It is expected that each group member contributes towards at least one of the tasks.

Kindly report errors (even typos) and ask for clarifications when needed, this assignment is to be an exercise in deep learning, not mind reading. Corrections to this assignment post-release will be listed in Section 4.

[1] `https://colab.research.google.com`
[2] `https://julialang.org`
[3] `https://www.pdflabs.com/tools/pdftk-the-pdf-toolkit`

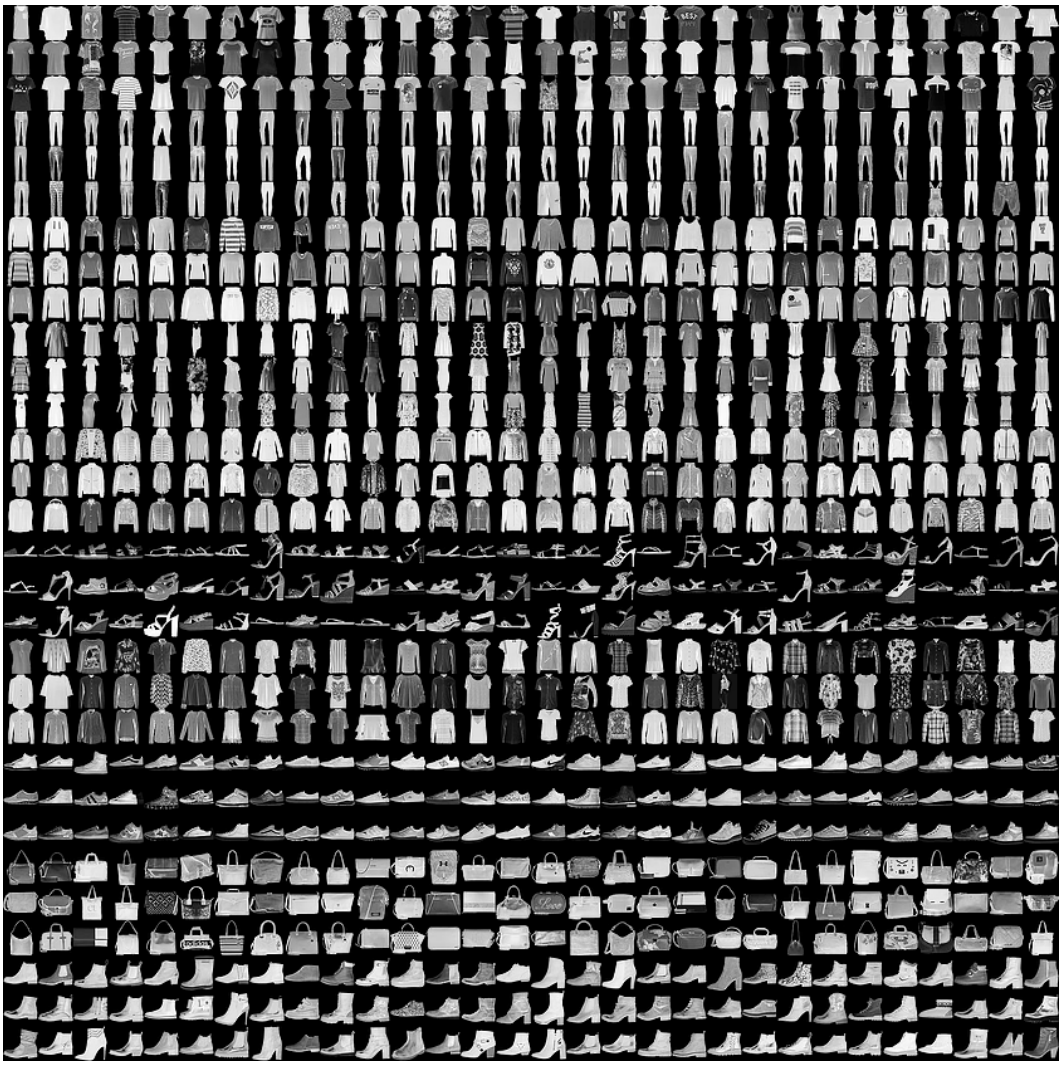

Figure 1: Examples from each of ten classes from Xiao et al. (2017), each class occupies three rows.

2 Data

A standard dataset in machine learning is the MNIST dataset of handwritten digits that has for many years been used to train and evaluate machine learning algorithms. However, even simple algorithms can perform relatively well on MNIST and its usage has increasingly come into question. As a response, alternative "drop-in" replacement datasets have been made available and we will use exactly such a dataset for all tasks in this assignment.

Zalando[4] is a German e-commerce company and that has released a dataset "Fashion-MNIST" (Xiao et al., 2017) which instead of digits from 0 to 10, contains 28-by-28 pixel, greyscale images of ten distinct fashion items (see Figure 1 for an example subset). The module organisers have prepared a subset of the dataset where the task is to classify a given image as to whether it depicts a bag or not. We have also created train, validation, and test splits, but it important to note that until the submission has passed the test data will consist solely of noise and that *your marks are in no way affected by your absolute performance on the hidden test set*.

A Colaboratory notebook loading the data can be found at: `https://colab.research.google.com/drive/1Tt4LTpLm_vjwkeOwrjNkNUAf1e49pF6I`.

[4] `https://www.zalando.de`

3 Tasks

3.1 A memory-efficient voted perceptron *(20 points)*

Freund & Schapire (1999) introduced the voted-perceptron algorithm (Algorithm 1), which scales the contribution to its weight vectors by a factor depending on how many correct classifications it has taken part in during training. They also demonstrated that despite its simplicity, the algorithm can be competitive with significantly more involved algorithms such as the support vector machine. However, a major drawback is that it has a memory complexity of $\mathcal{O}(k)$, where k is the number of times a sample is misclassified during training.

Data: $\mathcal{D} \coloneqq \{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_n}, y_n)\}$
Result: $v \coloneqq \{(\mathbf{w_1}, c_1), \ldots, (\mathbf{w_k}, c_k)\}$
$\mathbf{w_1} \coloneqq 0^d$;
$c_1 \coloneqq 1$;
$k \coloneqq 1$;
while *not converged* **do**
  for $i \coloneqq 1 \ldots, n$ **do**
    $\hat{y} \coloneqq \begin{cases} 1 & \mathbf{w_k} \cdot \mathbf{x_i} \geq 0 \\ 0 & else \end{cases}$;
    if $\hat{y} = y$ **then**
      $c_k \coloneqq c_k + 1$;
    else
      $\mathbf{w_{k+1}} \coloneqq \mathbf{w_k} + (y - \hat{y})\mathbf{x_i}$;
      $c_{k+1} \coloneqq 1$;
      $k \coloneqq k + 1$;
    end
  end
end

Algorithm 1: The voted-perceptron training algorithm with bias terms omitted.

Once trained, the algorithm performs predictions as follows:

$$f(\mathbf{x}) = \begin{cases} 1 & [\sum_{i:=1}^{k} c_i(\mathbf{w_i} \cdot \mathbf{x})] \geq 0 \\ 0 & else \end{cases} \tag{1}$$

To complete this task you are expected to:

1. Derive a variant of the voted perceptron with memory complexity $\mathcal{O}(1)$.
2. Prove that your efficient variant is functionally equivalent to the voted perceptron.
3. Implement your efficient variant in your Colaboratory notebook.
4. Train your model to convergence[5] on the training set.
5. Provide a plot of the accuracy on the training set for each epoch.
6. Provide the accuracy on the training and validation set, for the epoch on which you obtain your highest accuracy on the validation set.

[5] When you observe a lack of improvement on the training set over several epochs after having seen a large amount of initial improvement.

3.2 Mean squared-loss logistic regression *(20 points)*

In our lectures we have covered using the log-likelihood loss function for logistic regression, but it is of course possible to use a wide range of loss functions. Consider for example the mean squared loss below which is common for regression tasks, but can also be applied for binary classification:

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} \frac{(y_i - \hat{y}_i)^2}{2} \tag{2}$$

To complete this task you are expected to:

1. Derive the analytical gradients for the model parameters given the square loss.
2. Provide code verifying that your analytical gradients are correct using finite difference[6] for two or more samples from the training data (you are free to use an external package that calculates the finite difference).
3. Implement the algorithm in your Colaboratory notebook.
4. Train your model to convergence[7] using full-batch gradient descent on the training set.
5. Provide a plot of the square loss on the training set for each epoch.
6. Provide a plot of the accuracy on the training set for each epoch.
7. Provide the accuracy on the training and validation set, for the epoch on which you obtain your highest accuracy on the validation set.

[6] Keep in mind that the precision of your floating point type will be a major factor when using finite difference, it is thus common to use `Float64` and avoid using the GPU when verifying gradients this way.

[7] When you observe a lack of improvement on the training set over several epochs after having seen a large amount of initial improvement.

3.3 Three-layer multi-layer perceptron *(20 points)*

In our lectures we have introduced and implemented a two-layer multi-layer perceptron (MLP), a natural next step would be a three-layer MLP:

$$W_1 \in \mathbb{R}^{h_1 \times 28^2} \tag{3}$$

$$W_2 \in \mathbb{R}^{h_2 \times h_1} \tag{4}$$

$$W_3 \in \mathbb{R}^{1 \times h_2} \tag{5}$$

$$b_1 \in \mathbb{R}^{h_1} \tag{6}$$

$$b_2 \in \mathbb{R}^{h_2} \tag{7}$$

$$b_3 \in \mathbb{R} \tag{8}$$

$$f(x) = \sigma(W_3\sigma(W_2\sigma(W_1x + b_1) + b_2) + b_3) \tag{9}$$

To complete this task you are expected to:

1. Assuming the log-likelihood loss, derive the analytical gradients for the model parameters.
2. Provide code verifying that your analytical gradients are correct using finite difference[8] for two or more samples from the training data (you are free to use an external package that calculates the finite difference).
3. Provide code verifying that your analytical gradients are correct using finite difference for two or more samples from the training data (you are free to use an external package that calculates the finite difference).
4. Implement the algorithm in your Colaboratory notebook.
5. Train your model to convergence[9] using full-batch gradient descent on the training set.
6. Provide a plot of the loss on the training set for each epoch.
7. Provide a plot of the accuracy on the training set for each epoch.
8. Provide the accuracy on the training and validation set, for the epoch on which you obtain your highest accuracy on the validation set.

[8] Keep in mind that the precision of your floating point type will be a major factor when using finite difference, it is thus common to use `Float64` and avoid using the GPU when verifying gradients this way.

[9] When you observe a lack of improvement on the training set over several epochs after having seen a large amount of initial improvement.

3.4 Hyperparameter tuning *(20 points)*

For any deep learning (or machine learning for that matter) model you will be faced with the task of determining a set of "good" hyperparameters given a validation loss.

To complete this task you are expected to:

1. Starting with the two-layer MLP implementation we used in class, implement the following in your Colaboratory notebook:
   (a) (Full-)batch gradient descent *without* momentum
   (b) Stochastic gradient descent *without* momentum
   (c) Mini-batch gradient descent *without* momentum
   (d) (Full-)batch gradient descent *with* momentum
   (e) Stochastic gradient descent *with* momentum
   (f) Mini-batch gradient descent *with* momentum
2. Attempt to find "good" values for the following hyperparameters:
   (a) Learning rate
   (b) Batch size
   (c) Momentum coefficient
3. List the hyperparameter values you have evaluated and describe briefly (a single paragraph) your strategy and how you ultimately arrived at your "good" hyperparameters.
4. Provide a plot of the loss on the training and validation set for each epoch for your "good" hyperparameters.
5. Provide a plot of the accuracy on the training and validation set for each epoch for your "good" hyperparameters.
6. Provide the accuracy on the training and validation set for your "good" hyperparameters, for the epoch on which you obtain your highest accuracy on the validation set.

3.5 Model shootout *(20 points)*

To prove a models viability for a dataset, it is common to first establish the performance of a "weaker" model and show that performance can be improved using a more complex model.

To complete this task you are expected to:

1. Implement a "vanilla" perceptron model (as introduced in Lesson 4) in your Colaboratory notebook, we will refer to this as our *baseline* model.
2. Train your baseline model on the training set and use the validation set to determine when to stop training.
3. Consider the models we have covered so far and make a reasonable attempt to outperform the baseline model on the validation set.
4. Briefly describe in one or three paragraphs how you approached the task and what your final best model is, also listing its hyperparameters.
5. Provide a plot of the loss on the training, validation, and *test* set for each epoch for your best model.
6. Provide a plot of the accuracy on the training, validation, and *test* set for each epoch for your best model.
7. Provide the accuracy on the training, validation, and *test* set for your best model, for the epoch on which you obtain your highest accuracy on the validation set.

As you may have noted, we are asking you to evaluate your model on a test set that consists of random noise; in isolation this makes little sense. However, after the submission deadline we will replace the random noise test set with our hidden test set to compile an anonymous leader board to be presented at a future lecture. If you want a team name and/or your own names to be listed on this leader board, please state so explicitly in the PDF document that you submit – if no explicit preference is given, we will assume that you want to remain anonymous to your fellow students. We again want to emphasise that *your marks are in no way affected by your absolute performance on the hidden test set*. You are simply, in a way, taking part in *graduate student descent*...

4 Errata

04-11-2019 Corrected three typos in Algorithm 1 and clarified the writing regarding k, thank you Marco CAROBENE and Simon LEHNERER for the feedback.

05-11-2019 Removed an erroneous mention of "square loss" from Section 3.3, thank you Rahul TRADA for spotting this.

12-11-2019 Removed three erroneous transposes from Equation 9, thank you Song WAN JING for spotting this. Several students gave feedback at the tutorial session yesterday, which was forwarded to me through Teaching Assistant Max BARTOLO. This led to several improvements: added details regarding expected notebook runtime, better wording regarding the expected "final" metrics, and clarifications regarding the application of finite difference. A big thank you to the students who delivered this feedback.

18-11-2019 Removed the use of the term "significantly" in Section 3.5 as it could unintentionally imply statistical significance, it has now been clarified that what we expect is an attempt to outperform the baseline. The use of the term "'vanilla' perceptron" in Section 3.5 was rather unfortunate as it could refer to a single logistic regression layer, what was intended was the very first perceptron we introduced in Lesson 4. If you have already implemented something that could reasonably be interpreted to be a "vanilla" perceptron, do not worry, we will treat it as perfectly acceptable as a baseline for the purpose of this submission. Added a comment regarding the relation between floating point type and error for finite difference in Sections 3.2 and 3.3. Clarified that by "convergence", we mean that the training accuracy or loss has showed shown initial progress, but then not made any large improvement for a great number of epochs. While this is a very loose usage of the term, it is the one commonly used by machine learning researchers and practitioners. A big thank you to all the students that gave feedback.

References

Yoav Freund and Robert E. Schapire. Large Margin Classification Using the Perceptron Algorithm. *Machine Learning*, 37(3):277–296, Dec 1999. ISSN 1573-0565. doi: 10.1023/A:1007662407062.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms, 2017.