COMP0090 2020/21 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.

You are free to use *any programming language* and even mix programming languages between tasks. *You are not to use any automatic differentiation packages* (such as Flux, TensorFlow, PyTorch, etc.), 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 Thursday, November 5th 2020 at 16:00 (Europe/London)* and is to be submitted via Moodle. You shall use LaTeX[1] to produce *a single PDF document*. Use the following template as a starting point: `https://www.overleaf.com/read/gyphwtvffnxj`, but you are free to improve upon it as you see fit.

Describe briefly (about a paragraph) in your submission *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, but as with all group work *all members are collectively responsible for the submission in its entirety* and should confirm that they are familiar with all solutions prior to submission.

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://www.overleaf.com/learn/latex/Learn_LaTeX_in_30_minutes`

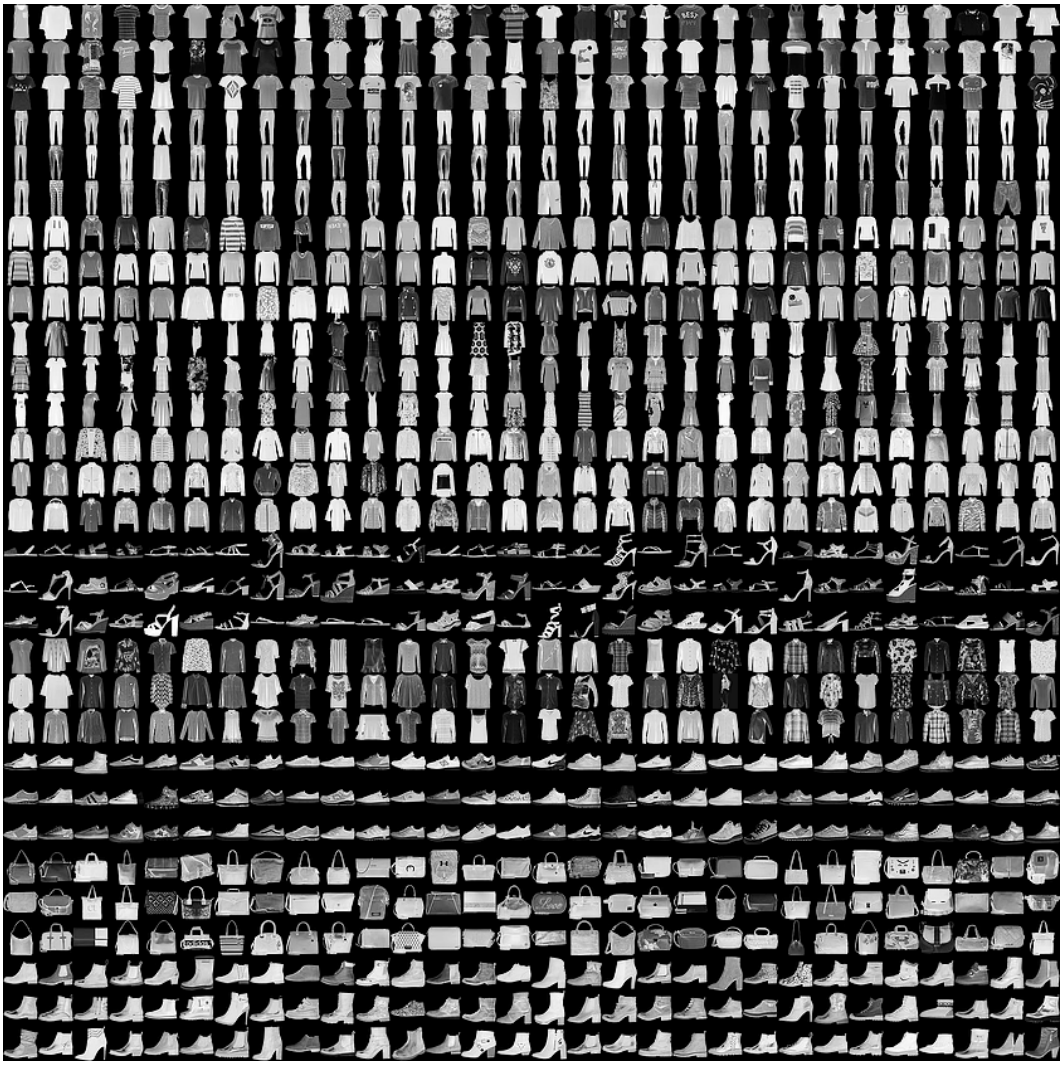

Figure 1: Examples from each of ten classes from Xiao et al. (2017), each class occupies three rows.

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[2] 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.

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

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

3 Tasks

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

Freund & Schapire (1999) introduced the voted-perceptron algorithm, 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. For this task we will consider a novel perceptron algorithm inspired by the voted perceptron and thus suffering from the same drawback in terms of memory complexity.

Data: $\mathcal{D} \coloneqq \{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_n}, y_n)\}$
Result: $v \coloneqq \{(\mathbf{w_1}, c_1), \ldots, (\mathbf{w_k}, c_k)\}$
$\mathbf{w_1} \coloneqq 0^d$;
$c_1 \coloneqq 0$;
$k \coloneqq 1$;
while *not converged* **do**
  for $i \coloneqq 1 \ldots, n$ **do**
    $\hat{y} \coloneqq \begin{cases} 1 & \mathbf{w_k} \cdot \mathbf{x_i} \geq 0 \\ 0 & \text{otherwise} \end{cases}$;
    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: Training algorithm with bias terms omitted.

Once trained, the algorithm performs predictions as follows (bias terms omitted):

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

To complete this task you are expected to:

1. Derive a variant of the algorithm above with memory complexity $\mathcal{O}(1)$ – informally, that the memory usage remains constant regardless of the number of misclassifications.
2. Prove that your efficient variant is functionally equivalent to the algorithm above.
3. Implement your efficient variant.
4. Train your model to convergence[3] 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.

[3] 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[4] 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.
4. Train your model to convergence[5] 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.

[4] 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.

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

3.3 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[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.
4. Train your model to convergence[7] using full-batch gradient descent on the training set.
5. Provide a plot of the 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.

3.4 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:
   (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, 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 and validation set for each epoch for your best model.
6. Provide a plot of the accuracy on the training and validation set for each epoch for your best model.
7. Provide the accuracy on the training and validation set for your best model, for the epoch on which you obtain your highest accuracy on the validation set.

4 Errata

26-10-2020
(a) Corrected Equation 1 to correspond to the Voted Perceptron.
(b) Corrected c_i initialisation to 0, as opposed to 1, in Algorithm 1.
(c) Removed references and requirement to evaluate on the test set in Sections 2 and 3.5, as this procedure caused confusion and marking concerns for a number of students.
(d) Removed duplicate sub-task in Section 3.3.
(e) Removed spurious left-over Colaboratory mentions.

27-10-2020
(a) Reverted Equation 1 to correspond to its previous state and corrected the writing to be unambiguous in regards to the given algorithm being novel.

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.