Deep Learning (IST, 2024-25)

Homework 1

André Martins, Chrysoula Zerva, Mário Figueiredo,
Duarte Alves, Pedro Santos, Duarte Almeida, Francisco Silva, Adrian Herta

Deadline: Monday, December 16, 2024.

Please turn in the answers to the questions below in a PDF file, together with the code you implemented to solve them (when applicable).

IMPORTANT: Please write 1 paragraph indicating clearly what was the contribution of each member of the group in this project. A penalization of 10 points will be applied if this information is missing.

Please submit **a single zip file** in Fenix under your group's name.

Question 1 (35 points)

Landscape classification with linear classifiers and neural networks. In this exercise, you will implement a linear classifier to classify images of landscapes, using a preprocessed version of the Intel Image Classification dataset. The preprocessed dataset contains 17,034 RGB images of size 48x48 of different landscapes, under 6 categories. The set of labels is {0: buildings; 1: forest; 2: glacier; 3: mountain, 4: sea; 5: street}. The dataset was split into training, validation and test sets with sizes 12,630, 1,404, and 3,000 respectively. **Please do not use any machine learning library such as `scikit-learn` or similar for this exercise; just plain linear algebra (the `numpy` library is fine).** The python skeleton code is provided in `hw1-q1.py`.

In order to complete this exercise, you will need to download the preprocessed Intel Image Classification dataset that can be found in:

`https://drive.google.com/file/d/16AzJIrmra4qWdY7wU9N1ew4bwadmNt4j`

1. In the first part of the exercise, we will implement the Perceptron algorithm.
   (a) (7 points) Implement the `update_weights` method of the `Perceptron` class in `hw1-q1.py`. Then, train the perceptron for 100 epochs on the training set and report its performance on the training, validation and test sets. Plot the train and validation accuracies as a function of the epoch number. You can do this with the command

   ```
   python hw1-q1.py perceptron -epochs 100
   ```

2. In the second part of this exercise, we will implement a logistic regression classifier, using stochastic gradient descent as the training algorithm. In particular, you will implement two versions of the logistic regression classifier: (i) a non-regularized version of logistic regression; and (ii) an ℓ_2-regularized version of logistic regression.

(a) (7 points) We will start with the logistic regression **without ℓ_2 regularization**. For this, implement the `update_weights` method of the `LogisticRegression` class. In this exercise, you can ignore the `l2_penalty` argument. Train the classifier for 100 epochs with a learning rate of 0.001 using the command.

```
python hw1-q1.py logistic_regression -epochs 100
```

(b) (5 points) Now, modify the **update_weights** function to support ℓ_2 regularization, where the strength of the regularization is defined by the `l2_penalty` argument. Note that when `l2_penalty` is zero we should recover the non-regularized version of the classifier. Train the model with a `l2_penalty` of 0.01 and report the final test accuracies as well as plot the train and validation accuracies over the epochs. Comment on the differences on the train and validation accuracies obtained with and without regularization. For training the classifier, you can use the command:

```
python hw1-q1.py logistic_regression -epochs 100 -l2_penalty 0.01
```

(c) (3 points) Finally, we take a closer look at how regularization impacts the values of the weights of the logistic regression classifier. For this, report the ℓ_2-norm of the weights (i.e., $\|\boldsymbol{W}\|_F = \sqrt{\sum_{i,j} W_{ij}^2}$, where $\boldsymbol{W}$ denotes the weights of the logistic regression classifier) of both the non-regularized and regularized versions of the logistic regression classifiers as a function of the number of epochs and comment on the obtained values.

(d) (3 points) What would you expect to be different in the values of the weights at the end of training if you were to use ℓ_1 regularization instead of ℓ_2 regularization? You do **not** need to implement a logistic regression classifier with ℓ_1 regularization, just briefly comment on the expected differences.

3. Now, you will implement a multi-layer perceptron (a feed-forward neural network), again using as input the original feature representation (i.e., simple independent pixel values).

   (a) (10 points) **Without using any neural network toolkit,** implement a multi-layer perceptron with a single hidden layer to solve this problem, including the gradient backpropagation algorithm needed to train the model. Use 100 hidden units, a `relu` activation function for the hidden layers, and a multinomial logistic loss (also called cross-entropy) in the output layer. Do not forget to include bias terms in your hidden units. Train the model for 20 epochs with stochastic gradient descent with a learning rate of 0.001. Initialize biases with zero vectors and values in weight matrices with $w_{ij} \sim \mathcal{N}(\mu, \sigma^2)$ with $\mu = 0.1$ and $\sigma^2 = 0.1^2$ (hint: use `numpy.random.normal`). Run your code with the base command, adding the necessary arguments

   ```
   python hw1-q1.py mlp
   ```

   Report the final test accuracy and include the plots of the train loss and train and validation accuracies as a function of the epoch number.

Question 2 (35 points)

Landscape classification with an automatic differentiation toolkit. In Question 1, you implemented gradient backpropagation manually. Now, you will implement the same classification system using a deep learning framework with automatic differentiation. Skeleton code for PyTorch is provided in `hw1-q2.py`. If you prefer a different framework, you may use it instead.

1. (10 points) Implement a linear model using logistic regression and stochastic gradient descent for training. Use a `batch_size` of 32 and set the ℓ_2 regularization (`l2_decay`) parameter to 0.01. Train your model for 100 `epochs` and tune the `learning_rate` by evaluating the following values: $\{0.00001, 0.001, 0.1\}$.

Identify the learning rate that achieved the highest validation accuracy, reporting corresponding test and validation accuracy. Plot the training and validation losses for each configuration, and compare the differences between the validation loss curves, explaining these differences by relating to the learning rate values.

In the provided skeleton code, you will need to implement the `train_batch()` function as well as the `__init__()` and `forward()` methods of the `LogisticRegression` class.

2. (25 points) In this exercise, you will implement a feed-forward neural network. Your implementation should follow the hyperparameters and training/model design choices detailed in Table 1, which should be used by default.

Number of Epochs	200
Learning Rate	0.002
Hidden Size	200
Number of Layers	2
Dropout	0.3
Batch Size	64
Activation	ReLU
L2 Regularization	0.0
Optimizer	SGD

Table 1: Default hyperparameters.

In the skeleton code, you will need to implement the class `FeedforwardNetwork`'s `__init__()` and `forward()` methods.

(a) (8 points) Train 2 models: one using the default hyperparameters and another with `batch_size` 512 and the remaining hyperparameters at their default value. Plot the train and validation losses for both, report the test and validation accuracy for both and explain the differences in both performance and time of execution.

(b) (9 points) Train the model setting `dropout` to each value in $\{0.01, 0.25, 0.5\}$ while keeping all other hyperparameters at their default values. Report the final validation and test accuracies and plot the training and validation losses for the three configurations. Analyze and explain the results.

(c) (8 points) Using a `batch_size` of 1024, train the default model while setting the `momentum` parameter to each value in $\{0.0; 0.9\}$ (use the `-momentum` flag). For the two configurations, plot the train and validation losses and report the test and validation accuracies. Explain the differences in performance.

Question 3 (30 points)

Multi-layer perceptron with activations $g(z) = z(1-z)$. In this exercise, we will consider a feed-forward neural network with a single hidden layer and the activation function $g(z) = z(1-z)$. We will see that, under some assumptions, this choice of activation, unlike other popular activation functions such as tanh, sigmoid, or relu, can be tackled as a linear model via a reparametrization.

We assume a univariate regression task, where the predicted output $\hat{y} \in \mathbb{R}$ is given by $\hat{y} = \boldsymbol{v}^\top \boldsymbol{h} + v_0$, where $\boldsymbol{h} \in \mathbb{R}^K$ are internal representations, given by $\boldsymbol{h} = \boldsymbol{g}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$, $\boldsymbol{x} \in \mathbb{R}^D$ is a vector of input variables, and $\Theta = (\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{v}, v_0) \in \mathbb{R}^{K\times D} \times \mathbb{R}^K \times \mathbb{R}^K \times \mathbb{R}$ are the model parameters.

1. (10 points) Show that we can write $\boldsymbol{h} = \boldsymbol{A}_\Theta \boldsymbol{\phi}(\boldsymbol{x})$ for a certain feature transformation $\boldsymbol{\phi} : \mathbb{R}^D \to \mathbb{R}^{\frac{(D+1)(D+2)}{2}}$ independent of Θ, where $\phi_1(\boldsymbol{x}) = 1$ (i.e., the first feature is a constant feature), and $\boldsymbol{A}_\Theta \in \mathbb{R}^{K \times \frac{(D+1)(D+2)}{2}}$. That is, $\boldsymbol{h}$ is a **linear transformation** of $\boldsymbol{\phi}(\boldsymbol{x})$. Determine the mapping $\boldsymbol{\phi}$ and the matrix $\boldsymbol{A}_\Theta$.

2. (5 points) Based on the previous claim, show that $\hat{y}$ is a linear transformation of $\boldsymbol{\phi}(\boldsymbol{x})$, i.e., we can write $\hat{y}(\boldsymbol{x}; \boldsymbol{c}_\Theta) = \boldsymbol{c}_\Theta^\top \boldsymbol{\phi}(\boldsymbol{x})$ for some $\boldsymbol{c}_\Theta \in \mathbb{R}^{\frac{(D+1)(D+2)}{2}}$. Determine $\boldsymbol{c}_\Theta$. Does this mean this is a linear model in terms of the original parameters Θ? (Note: you can answer this question even if you have not solved the previous exercise.)

3. (10 points) Assume $K \geq D$. Show that for any real vector $\boldsymbol{c} \in \mathbb{R}^{\frac{(D+1)(D+2)}{2}}$ and any $\epsilon > 0$ there is a choice of the original parameters $\Theta = (\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{v}, v_0)$ such that $\|\boldsymbol{c}_\Theta - \boldsymbol{c}\| < \epsilon$. That is, up to ϵ-precision **we can equivalently parametrize the model with $\boldsymbol{c}$ instead of Θ.** Describe a procedure to obtain Θ from $\boldsymbol{c}$. (Hint: use orthogonal decomposition of a certain matrix assumed non-singular. Use the fact that any matrix is ϵ-close to a non-singular matrix.)

4. (5 points) Suppose we are given training data $\mathcal{D} = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$ with $N > \frac{(D+1)(D+2)}{2}$ and where all input vectors $\{\boldsymbol{x}_n\}$ are linearly independent, and that we want to minimize the squared loss
$$L(\boldsymbol{c}_\Theta; \mathcal{D}) = \frac{1}{2} \sum_{n=1}^N (\hat{y}(\boldsymbol{x}_n; \boldsymbol{c}_\Theta) - y_n)^2.$$
Can we find a closed form solution $\hat{\boldsymbol{c}}_\Theta$? Comment on the fact that global minimization is usually intractable for feedforward neural networks – what makes our problem special? (Note: you can answer this question even if you have not solved the previous exercises.)