Deep Learning (IST, 2023-24)

Homework 1

André Martins, Francisco Melo, João Santinha, Duarte Alves, Margarida Campos

Deadline: Friday, December 15, 2023.

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)

Medical image classification with linear classifiers and neural networks. In this exercise, you will implement a linear classifier for a simple medical image classification problem, using the OCTMNIST dataset. The dataset contains 109,309 valid optical coherence tomography (OCT) images for retinal diseases comprised of 4 diagnosis categories - multi-class classification task. The original training set was split with a ratio of 9:1 into training and validation set, and use its source validation set as the test set. The source images are gray-scale with sizes varying between (384–1,536)×(277–512). Images were center-cropped with a window size of length of the short edge and resize them into 1×28×28. **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).** Python skeleton code is provided (`hw1-q1.py`).

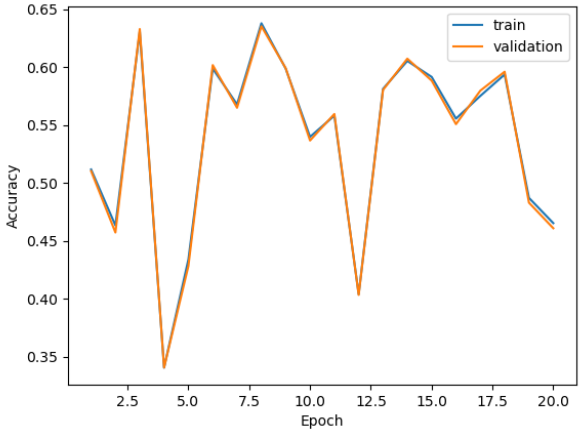
In order to complete this exercise, you will need to download the OCTMNIST dataset. You can do this by running the following command in the homework directory:

```
python download_octmnist.py
```

1. (a) (7 points) Implement the `update_weights` method of the `Perceptron` class in `hw1-q1.py`. Then train 20 epochs of the perceptron 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
   ```

   Solution: Final training accuracy was 0.4654. Final validation accuracy was 0.4610. Final test accuracy was 0.3422.



(b) (13 points) Repeat the same exercise using logistic regression instead (without regularization), using stochastic gradient descent as your training algorithm. Report the final test accuracies and compare, based on the plots of the train and validation accuracies, the models obtained using two different learning rates of $\eta = 0.01$ and $\eta = 0.001$. This can be solved by implementing the `update_weights` method in the `LogisticRegression` class. You can do this with the command

```
python hw1-q1.py logistic_regression -epochs 50 -learning_rate 0.01
```

Solution: Final test accuracy was 0.4556 using an $\eta = 0.01$ and 0.6068 for $\eta = 0.001$. The train and validation accuracies were more stable for $\eta = 0.001$ as seen in Figure 1 and test accuracy was also higher.

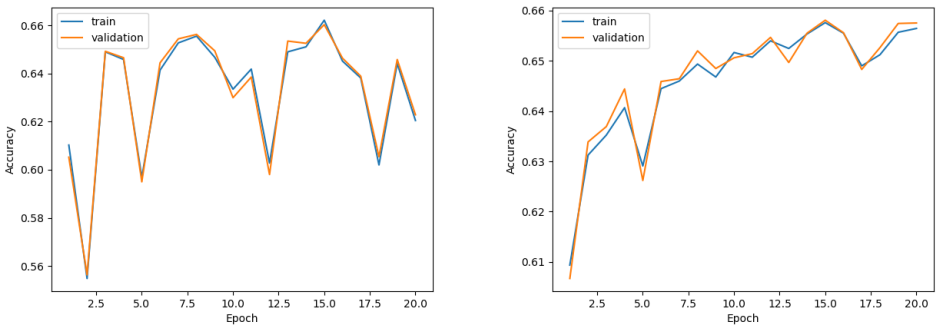


Figure 1: Logistic regression train and validation accuracy plots as a function of the epoch number for $\eta = \{0.01; 0.001\}$. On the left: train and validation accuracies obtained with a $\eta = 0.01$. On the right: train and validation accuracies obtained with a $\eta = 0.001$.

2. 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) (5 points) Comment the following claim: "A logistic regression model using pixel values as features is not as expressive as a multi-layer perceptron using `relu` activations. However, training a logistic regression model is easier because it is a convex optimization problem." Is this true of false? Justify.

Solution: The claim is correct. Logistic regression is a linear model and therefore its decisions cannot depend on interaction among pixels, which is a strong limitation. Multi-layer perceptron with non-linear activations is able to learn such interactions

through its internal representations. However, logistic regression leads to a convex optimization problem, unlike multi-layer perceptron, which is non-convex and therefore depends on the initialization.

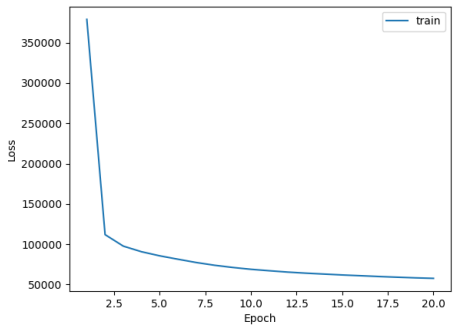
(b) (10 points) **Without using any neural network toolkit,** implement a multi-layer perceptron with a single hidden layer to solve this problem, including the gradient backpropagation algorithm which is needed to train the model. Use 200 hidden units, a `relu` activation function for the hidden layers, and a multinomial logistic loss (also called cross-entropy) in the output layer (eventhough we are dealing with a binary classification this will allow you to use the same code for a multi-class problem). Don't 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

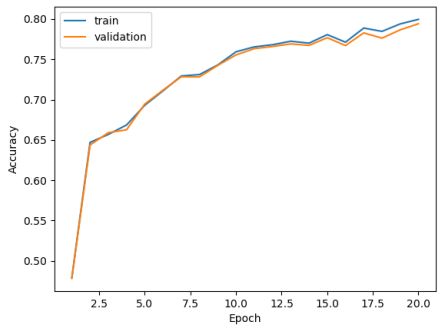
```
python hw1-q1.py mlp
```

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

Solution: The final test accuracy was 0.7675.

The train loss and train and validation accuracies as a function of the epoch number are shown below





Question 2 (35 points)

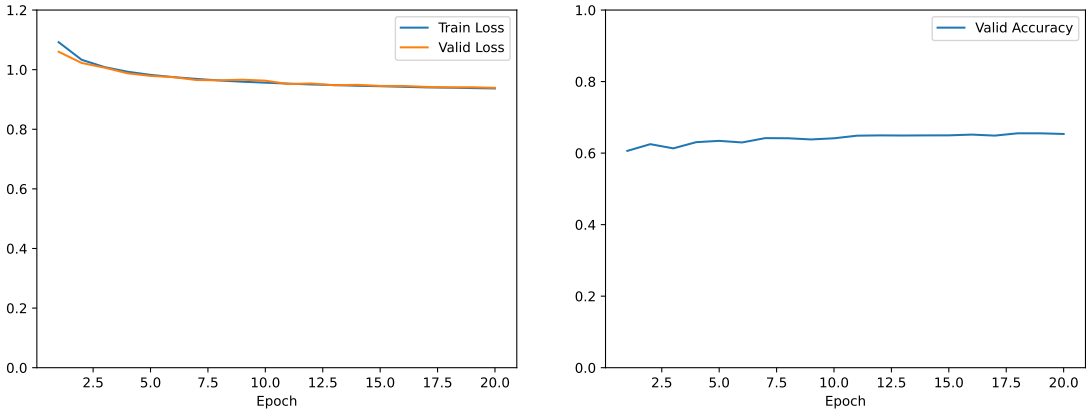
Medical image classification with an autodiff toolkit. In the previous question, you had to write gradient backpropagation by hand. This time, you will implement the same system using a deep learning framework with automatic differentiation. Pytorch skeleton code is provided (`hw1-q2.py`) but if you feel more comfortable with a different framework, you are free to use it instead.

1. (10 points) Implement a linear model with logistic regression, using stochastic gradient descent as your training algorithm (use a batch size of 16). Train your model for 20 epochs and tune the learning rate on your validation data, using the following values: $\{0.001, 0.01, 0.1\}$.

   Report the best configuration (in terms of final validation accuracy) and plot two things: the training loss and the validation accuracy, both as a function of the epoch number. Report the final accuracy on the test set.

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

Solution: The best configuration was obtained with a $lr = 0.01$ the accuracies were 0.6417 on the validation set and 0.6314 on the test set.



For $lr = 0.001$ the accuracies were 0.5764 on the validation set and 0.6030 on the test set.

For $lr = 0.1$ the accuracies were 0.4504 on the validation set and 0.3365 on the test set.

2. (25 points) Implement a feed-forward neural network using dropout regularization. Make sure to include all the hyperparameters and training/model design choices shown in Table 1. Use the values presented in the table as default.

Number of Epochs	20
Learning Rate	0.1
Hidden Size	200
Number of Layers	2
Dropout	0.0
Batch Size	16
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) Compare the performance of your model with batch sizes 16 and 1024 with the remaining hyperparameters at their default value. Plot the train and validation losses for both, report the best test accuracy and comment on the differences in both performance and time of execution.

You can time the execution of a program using `time`:

```
time python hw1.py mlp
```

Solution:

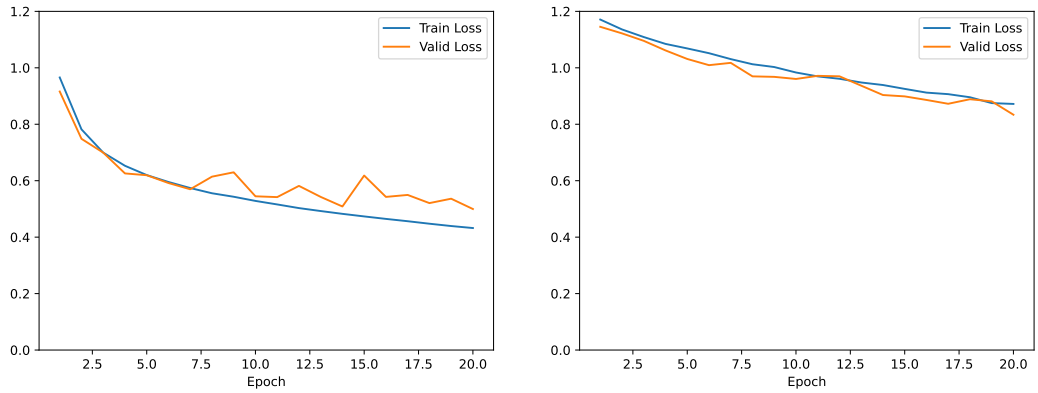


Figure 2: Train and validation losses for batch sizes 16 (left) and 1024 (right).

Best test accuracy for batch size 16: ≈ 0.76. Training time is over two times higher for batch size 16. The model with batches of size 16 does 64 times more updates than the one with batch size 16 - this leads to the faster computation time but resulst in lower performance for the same number of epochs since the model converges slower.

(b) (8 points) Train the model with learning rates: 1,0.1,0.01 and 0.001 with the remaining hyperparameters at their default value. Plot the train and validation losses for the best and worst configurations in terms of validation accuracy, report the best test accuracy and comment on the differences in performance.

Solution:

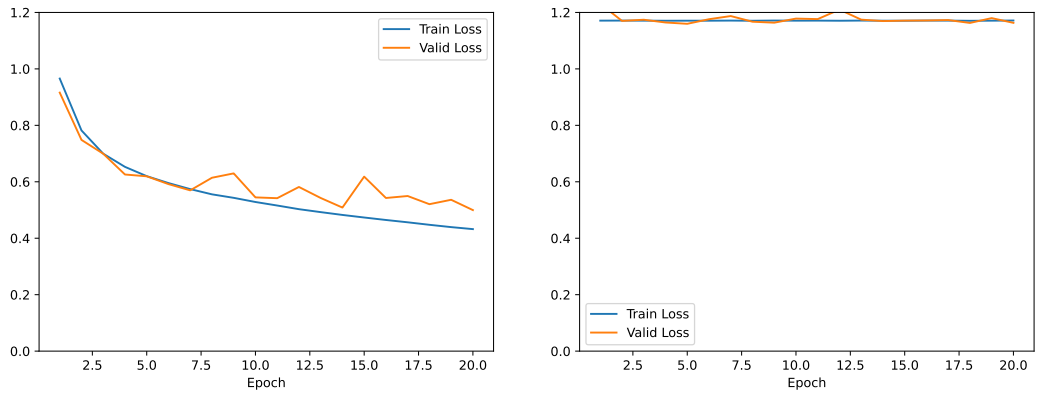


Figure 3: Train and validation losses for learning rates 0.01 (left) and 1 (right).

Best test accuracy for learning rate 0.1: ≈ 0.82. Similar results are obtained for 0.1 and 0.01. A learning rate of 1 yields the worst results, as it is too large and the model is not able to converge towards the minimum; a learning rate of 0.001 leads the model to converge too slowly, hence leading to a lower performance for the same number of epochs.

(c) (9 points) Using a batch size of 256 run the default model for 150 epochs. Is there overfitting? Train two similar models with the following changes: one with the L2 regularization parameter set to 0.0001 and the other with a dropout probability of 0.2. Plot the train and validation losses for the best and worst configuration in terms of validation accuracy, report the best test accuracy and comment on the differences of both techniques.

Solution:

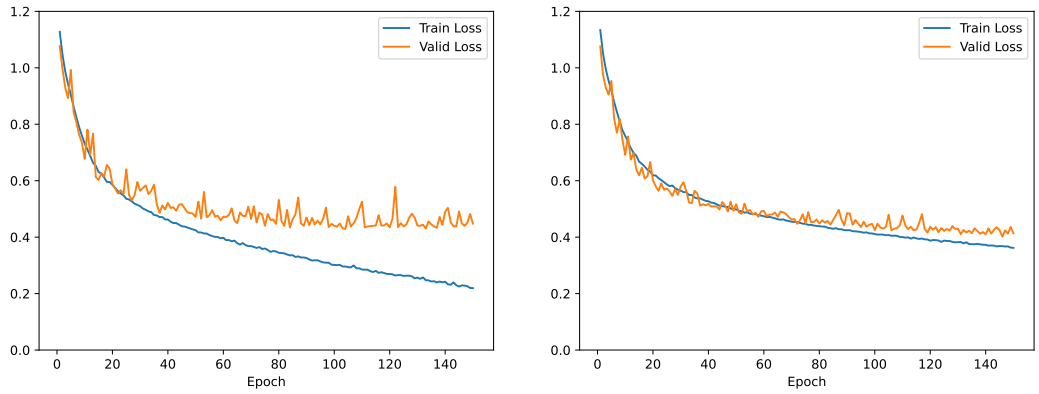


Figure 4: Train and validation losses without regularization (left) and with dropout regularization with probability 0.2 (right).

Best test accuracy with dropout regularization: ≈ 0.79. One can see the model starts to overfit without regularization. We can see that by applying L2 regularization - applying a penalization to the norm of the weights - the training loss does not decrease as much but validation loss decreases. With dropout regularization - randomly selecting 20% of the neurons not to be used - the training is even more regularized and we get a higher train loss but better performance on both validation and test sets.

The dropout tehcnique works by changing the network itself, instead of the loss function. It is used not only as regularization but also to increase model robustness.

Question 3 (30 points)

1. In this exercise, you will design a multilayer perceptron to compute a Boolean function of D variables, $f : \{-1, +1\}^D \to \{-1, +1\}$, defined as:

$$f(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \sum_{i=1}^{D} x_i \in [A, B], \\ -1 & \text{otherwise,} \end{cases} \tag{1}$$

where A and B are integers such that $-D \leq A \leq B \leq D$.

(a) (5 points) Show that the function above cannot generally be computed with a single perceptron. *Hint: think of a simple counter-example.*

Solution: When $D = 2$ and $A = B = 0$ this becomes the XOR problem, which is not linearly separable and therefore the function cannot be computed with a single perceptron.

(b) (15 points) Show that the function above can be computed with a multilayer perceptron with **a single hidden layer with two hidden units** and hard threshold activations $g: \mathbb{R} \to \{-1, +1\}$ with

$$g(z) = \text{sign}(z) = \begin{cases} +1 & \text{if } z \geq 0, \\ -1 & \text{otherwise,} \end{cases} \tag{2}$$

and where **all the weights and biases are integers** (positive, negative, or zero). Provide all the weights and biases of such network, and ensure that the resulting network is robust to infinitesimal perturbation of the inputs, i.e., the resulting function $h : \mathbb{R}^D \to \mathbb{R}$ should be such that $\lim_{t \to 0} h(\boldsymbol{x} + t\boldsymbol{v}) = h(\boldsymbol{x}) = f(\boldsymbol{x})$ for any $\boldsymbol{x} \in \{-1, +1\}^D$ and $\boldsymbol{v} \in \mathbb{R}^D$.

Solution: There might be multiple solutions, but we will solve this exercise by (1) designing one hidden unit to be active iff $\sum_i x_i \geq A - \frac{1}{2}$, (2) designing another hidden unit to be active iff $\sum_i x_i \leq B + \frac{1}{2}$, and (3) designing an output unit that does the conjunction of the outputs of the previous two units.

- Step (1) can be achieved by choosing $w_{1i}^{(1)} = 1$ for each i and $b_1^{(1)} = -A + \frac{1}{2}$. Since we want integer parameters – and noting that multiplying the argument of the sign function by a positive constant does not change the output – we can equivalently choose $w_{1i}^{(1)} = 2$ for each i and $b_1^{(1)} = -2A + 1$.
- Step (2) can be achieved by choosing $w_{2i}^{(1)} = -1$ for each i and $b_2^{(1)} = B + \frac{1}{2}$, or equivalently $w_{2i}^{(1)} = -2$ for each i and $b_2^{(1)} = 2B + 1$.
- Step (3) can be achieved by choosing $w_1^{(2)} = 1$, $w_2^{(2)} = 1$, and $b^{(2)} = -1$.

(c) (10 points) Repeat the previous exercise if the hidden units use rectified linear unit activations instead. (You will get partial credit if you solve for $D = 2$ and $A = B = 0$ only.)

Solution: With ReLUs on the hidden units, the previous network parameters do not work since step (3) as above is trickier – the output would need to be positive iff the two inputs are both positive, but just using their sum does not allow distinguishing this case from one where one input is positive and large and the second input is zero. But it is possible to solve the problem as follows.

- We can set $w_{1i}^{(1)} = 2$ for each i and $b_1^{(1)} = -2B - 1$. The output of this unit after the ReLU activation will be 0 if $\sum_i x_i \leq B + \frac{1}{2}$ and it will be ≥ 0 otherwise. For $\sum_i x_i \geq B + 1$ the output will be ≥ 1.
- Then we can set $w_{2i}^{(1)} = -2$ for each i and $b_2^{(1)} = 2A - 1$. The output of this unit after the ReLU activation will be 0 if $\sum_i x_i \geq A - \frac{1}{2}$ and it will be ≥ 0 otherwise. For $\sum_i x_i \leq A - 1$ the output will be ≥ 1.
- Finally, the output unit should output a non-negative number if the two inputs are zero, and a negative number if at least one of them is > 0. This is ensured by weights $w_1^{(2)} = -2$, $w_2^{(2)} = -2$, and $b^{(2)} = 1$.

Note: This solution is robust to noisy inputs (which was requested in the question) and also to noise in the hidden activations (which was not explicitly requested). Other solutions exist, of course. For example, it also works to use same hidden layer parameters as in previous question and $w_1^{(2)} = -1$, $w_2^{(2)} = -1$, and $b^{(2)} = 2$, without robustness to noise in the hidden activations.