Deep Learning (IST, 2022-23)

Homework 1

André Martins, Ben Peters, Margarida Campos

Deadline: Friday, December 23, 2022.

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)

Image classification with linear classifiers and neural networks. In this exercise, you will implement a linear classifier for a simple image classification problem, using the Kuzushiji-MNIST dataset, which contains handwritten cursive images of 10 characters from the Hiragana writing system (used for Japanese). Examples of images in this dataset are shown in Figure 1. **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 Kuzushiji-MNIST dataset. You can do this by running the following command in the homework directory:

```
python download_kuzushiji_mnist.py.
```

1. (a) (10 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 validation and test set. Plot the accuracies as a function of the epoch number. You can do this with the command

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

   (b) (10 points) Repeat the same exercise using logistic regression instead (without regularization), using stochastic gradient descent as your training algorithm. Set a fixed learning rate $\eta = 0.001$. 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
   ```

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).

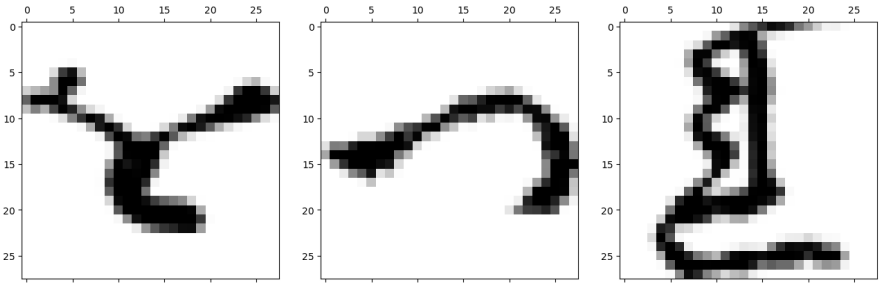

Figure 1: Examples of images from the Kuzushiji-MNIST dataset.

(a) (5 points) Justify briefly why multi-layer perceptrons with non-linear activations are more expressive than the simple perceptron implemented above, and what kind of limitations they overcome for this particular task. Is this still the case if the activation function of the multi-layer perceptron is linear?

(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. Don't forget to include bias terms in your hidden units. Train the model 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 command

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

Question 2 (35 points)

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 1). 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.

2. (15 points) Implement a feed-forward neural network with a single layer, 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. Tune each of these hyperparameters while leaving the remaining at their default value:

   - The learning rate: $\{0.001, 0.01, 0.1\}$.

- The hidden size: $\{100, 200\}$.
- The dropout probability: $\{0.3, 0.5\}$.
- The activation function: `relu` and `tanh`.

Number of Epochs	20
Learning Rate	0.01
Hidden Size	100
Dropout	0.3
Batch Size	16
Activation	ReLU
Optimizer	SGD

Table 1: Default hyperparameters.

Report your best configuration, make similar plots as in the previous question, and report the final test accuracy.

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

3. (10 points) Using the same hyperparameters as in Table 1, increase the model to 2 and 3 layers. Report your best configuration, make similar plots as in the previous question, and report the final test accuracy. (Note: in the real world, you would need to do hyperparameter tuning for the different network architectures, but this is not required for this assignment.)

Question 3 (30 points)

Multi-layer perceptron with quadratic activations. In this exercise, we will consider a feed-forward neural network with a single hidden layer and a quadratic activation function, $g(z) = z^2$. We will see 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 $\hat{y} = \boldsymbol{v}^\top \boldsymbol{h}$, where $\boldsymbol{h} \in \mathbb{R}^K$ are internal representations, given by $\boldsymbol{h} = \boldsymbol{g}(\boldsymbol{W}\boldsymbol{x})$, $\boldsymbol{x} \in \mathbb{R}^D$ is a vector of input variables, and $\Theta = (\boldsymbol{W}, \boldsymbol{v}) \in \mathbb{R}^{K \times D} \times \mathbb{R}^K$ 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 $\boldsymbol{\phi} : \mathbb{R}^D \to \mathbb{R}^{\frac{D(D+1)}{2}}$ independent of Θ and $\boldsymbol{A}_\Theta \in \mathbb{R}^{K \times \frac{D(D+1)}{2}}$. 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 also 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 $\boldsymbol{c}_\Theta \in \mathbb{R}^{\frac{D(D+1)}{2}}$. Does this mean this is a linear model in terms of the original parameters Θ?

3. (10 points) Assume $K \geq D$. Show that for any real vector $\boldsymbol{c} \in \mathbb{R}^{\frac{D(D+1)}{2}}$ there is a choice of the original parameters $\Theta = (\boldsymbol{W}, \boldsymbol{v})$ such that $\boldsymbol{c}_\Theta = \boldsymbol{c}$. That is, **we can equivalently parametrize the model with $\boldsymbol{c}_\Theta$ instead of Θ.** Does this mean this is a linear model in terms of $\boldsymbol{c}_\Theta$? Show that an equivalent parametrization might not exist if $K < D$.

4. (5 points) Suppose we are given training data $\mathcal{D} = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$ with $N > \frac{D(D+1)}{2}$ and where all input vectors $\{\boldsymbol{x}_n\}$ are linearly independent, and that we want to minimize the squared loss

$$L(\boldsymbol{c}_\Theta; \mathcal{D}) = \frac{1}{2} \sum_{n=1}^{N} (\hat{y}_n(\boldsymbol{x}_n; \boldsymbol{c}_\Theta) - y_n)^2.$$

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?