Deep Learning (IST, 2021-22)

Homework 1

André Martins, Francisco Melo, Ben Peters

Deadline: Friday, January 7, 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).
Please submit **a single zip file** in Fenix under your group's name.

Question 1

In this exercise, you will show that the binary and multinomial logistic losses are convex.

1. (5 points) The sigmoid activation function is $\sigma(z) = 1/(1+e^{-z})$, where $z \in \mathbb{R}$. Show that its derivative can be expressed as $\sigma'(z) = \sigma(z)(1-\sigma(z))$.

2. (5 points) Consider a binary classification problem with $y \in \{\pm 1\}$. A binary logistic regression model defines $P(y = +1 \mid x; \boldsymbol{w}, b) = \sigma(z)$ with $z = \boldsymbol{w}^\top \boldsymbol{\phi}(x) + b$. The binary logistic loss is

$$L(z; y) = \begin{cases} -\log \sigma(z) & \text{if } y = +1 \\ -\log(1-\sigma(z)) & \text{if } y = -1 \end{cases} = -\frac{1+y}{2} \log \sigma(z) - \frac{1-y}{2} \log(1-\sigma(z)), \quad (1)$$

   where y is the gold label. Assume $y = +1$ and compute the first and second derivatives of $L'(z; y = +1)$ and $L''(z; y = +1)$ with respect to z. Show that the binary logistic loss is convex as a function of z. *Hint: Use the result from the previous question.*

3. (5 points) Let us now turn to the multi-class case, where $y \in \{1, \ldots, K\}$ with $K \geq 2$. Let $\boldsymbol{z} \in \mathbb{R}^K$. The softmax transformation is a a function from $\mathbb{R}^K$ to $\mathbb{R}^K$ defined as

$$[\text{softmax}(\boldsymbol{z})]_j = \frac{\exp(z_j)}{\sum_{k=1}^K \exp(z_k)}$$

   for $j = 1, \ldots, K$ (the entries of $\boldsymbol{z}$ are called logits or scores). Compute the Jacobian matrix of the softmax transformation on point $\boldsymbol{z}$ – this is the K-by-K matrix whose (j,k)-th entry is $\frac{\partial [\text{softmax}(\boldsymbol{z})]_j}{\partial z_k}$.

4. (5 points) The multinomial logistic loss is defined as $L(\boldsymbol{z}; y = j) = -\log[\text{softmax}(\boldsymbol{z})]_j$. Compute the gradient and Hessian of this loss with respect to $\boldsymbol{z}$ and show that this loss is convex with respect to $\boldsymbol{z}$.

   Hint: Use again the result from the previous question.

5. (5 points) Show that in a linear model where $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{\phi}(x) + \boldsymbol{b}$, the multinomial logistic loss is also convex with respect to the model parameters $(\boldsymbol{W}, \boldsymbol{b})$, and therefore a local minimum is a also global minimum. Is this also true in general when $\boldsymbol{z}$ is not a linear function of the model parameters?

Hint: use the fact that the composition of an affine map $\boldsymbol{g} : \mathbb{R}^n \to \mathbb{R}^m$ with a convex function $f : \mathbb{R}^m \to \mathbb{R}$ is convex, i.e., if $\boldsymbol{g}(\boldsymbol{u}) = \boldsymbol{A}\boldsymbol{u} + \boldsymbol{c}$ and f is convex in $\mathbb{R}^m$, then $(f \circ \boldsymbol{g})(\boldsymbol{u}) = f(\boldsymbol{g}(\boldsymbol{u})) = f(\boldsymbol{A}\boldsymbol{u} + \boldsymbol{c})$ is convex in $\mathbb{R}^m$.

Question 2

Regression of house prices with linear models and neural networks.

1. (5 points) Consider the squared error loss

$$L(\boldsymbol{z}; \boldsymbol{y}) = \frac{1}{2}\|\boldsymbol{z} - \boldsymbol{y}\|_2^2 = \frac{1}{2}(\boldsymbol{z} - \boldsymbol{y})^\top(\boldsymbol{z} - \boldsymbol{y}),$$

   where $\boldsymbol{z} = \boldsymbol{W}^\top \boldsymbol{\phi}(x) + \boldsymbol{b}$. Show that L is convex with respect to $(\boldsymbol{W}, \boldsymbol{b})$.

2. **Ames housing dataset.** In this question, you will use the Ames housing dataset. The dataset lists a large number of properties in the city of Ames, Iowa. Each property is described by a large number of features and has an associated "price tag". Your goal is to build a predictor for the price of a property given the features describing it. The Ames housing dataset is similar to other existing datasets (such as the Boston or California datasets), but featuring much richer features. It is also more complex to handle: it contains missing data and both numerical and categorical features. For this task we have prepared the dataset beforehand, filling in missing values and encoding the categorical features using the "one-hot" encoding scheme. The data is included in the file `ames.npz`.

   You will train a linear regression model to predict the price of the different properties using the features already in the provided data. **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).**

   Skeleton code For this question, you are recommended (but not required) to use the skeleton scripts `hw1-q2.py` and `utils.py`, and the environment `environment.yml`, included in the attached zip file.

   (a) (10 points) Implement the `update_weights` method of the `LinearRegression` class in `hw1-q2.py`. Train the model for 150 epochs using stochastic gradient descent with a learning rate of 0.001. Report the performance on the training and test set. Plot the accuracy on both sets as a function of the epoch number. Explain the observed difference between the performance on the training and test sets as the number of epochs increases. Also report the distance between your weight vector and the weight vector computed analytically as a function of the epoch number (you will need to compute the analytic solution yourself by implementing the `solve_analytically` function). You can run the code with the command

```
python hw1-q2.py linear_regression
```

   (b) (10 points) You will now train a neural network model for the same regression problem. **Without using any neural network toolkit**, implement a feed-forward network with a single hidden layer to solve the regression problem above, including the gradient back-propagation algorithm which is needed to train the model. You can achieve this by implementing the `__init__()`, `update_weight`, and `predict` methods of the provided `NeuralRegression` class. Use 150 hidden units, a ReLU activation function for the hidden layer, and a mean squared error loss in the output. Train the model with stochastic gradient descent with a learning rate of 0.001 for 150 epochs. Initialize biases

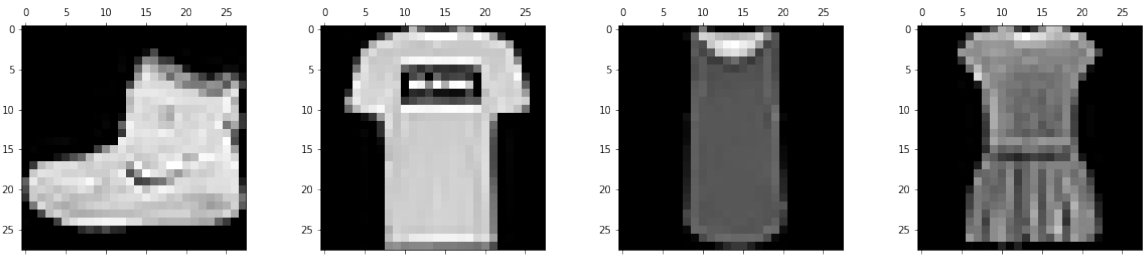

Figure 1: Examples of images from the FashionMNIST dataset.

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`). Report the performance on the training and test set. Plot the accuracy in both sets as a function of the epoch number. Compare the observed difference between the performance in the training and test sets with that observed in the linear model. You can do this with the command

```
python hw1-q2.py nn
```

Question 3

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 Fashion-MNIST dataset. 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-q3.py`).

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

```
python download_fashion_mnist.py.
```

1. (a) (5 points) Implement the `update_weights` method of the `Perceptron` class in `hw1-q3.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-q3.py perceptron
   ```

   (b) (5 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-q3.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).

   (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-q3.py mlp
```

Question 4

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-q4.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: $\{0.001, 0.01, 0.1\}$.

   For the best configuration, report it 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. (10 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`.
   - The optimizer: `SGD` and `Adam`.

Number of Epochs	20
Learning Rate	0.01
Hidden Size	200
Dropout	0.3
Batch Size	1
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. (5 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.)