Deep Structured Learning (IST, Fall 2020)

Homework 1

Instructor: André Martins
TA: Marcos Treviso

Deadline: Wednesday, October 28, 2020.

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 email your solutions in **electronic format** (a single zip file) with the subject "Homework 1" to:

`deep-structured-learning-instructors@googlegroups.com`

Hard copies will not be accepted.

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 $(\boldsymbol{W}, b)$, 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?

Question 2

Optical character recognition with linear classifiers and neural networks. In this exercise, you will implement a linear classifier for a simple image classification problem. **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).**

Download the OCR dataset from `http://ai.stanford.edu/~btaskar/ocr`. This dataset contains binary image representations of 52,152 alphabetical characters `a`–`z` (the characters are grouped together to form English words, but this structure will be ignored in this exercise). The task is to take each image representation as input (with 16x8 pixels) and to predict as output the correct character in `a`–`z` (i.e., a multi-class classification problem with 26 classes). The dataset is organized into 10 folds: folds 0–7 are for training (41,679 examples), 8 is for validation (5,331 examples), and 9 is for testing (5,142 examples). The evaluation metric is the fraction of characters correctly classified.

Skeleton code For this question, you are recommended but not required to use the skeleton script `hw1-q2.py`, which has been provided to you on the course webpage. In order to use it, make sure it is located in the same directory as the `letter.data` file from the OCR dataset. The script requires `Python3`, `numpy`, and `matplotlib`.

1. In the first part of the exercise, we will use as a feature representation the binary pixel values.
   (a) (5 points) Do you think this is a good choice of feature representation? Justify.
   (b) (10 points) Implement the `update_weights` method of the `Perceptron` class in `hw1-q2.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-q2.py perceptron
   ```

   (c) (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-q2.py logistic_regression
   ```

2. Let us now do some feature engineering.
   (a) (10 points) Can you think of a better feature representation? Come up with one and train the perceptron there. You can implement your feature representation in the function `custom_features` and then run

   ```
   python hw1-q2.py perceptron -custom_features
   ```

   Suggestion: instead of individual pixel binary values $\phi_i(\boldsymbol{x}) = x_i$ (where i indexes a pixel position), use as features all pixel pairwise combinations, $\phi_{ij}(\boldsymbol{x}) = x_i x_j$.
   (b) (5 points (bonus)) Repeat the previous exercise using logistic regression instead (with $\eta = 0.001$ and without regularization).

3. In the previous part, you might have noticed that feature engineering can be tedious. 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) Explain why multi-layer perceptrons can learn internal representations and avoid manual feature engineering.

   (b) (15 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 later. Don't forget to include bias terms in your hidden units. Train the model with stochastic gradient descent tuning the learning rate in $\{0.001, 0.01, 0.1\}$ in the validation set.

Question 3

OCR with an autodiff toolkit. In the previous question, you had to wrote gradient backpropagation by hand. This time, you will implement the same system using a deep learning framework with automatic differentiation. A skeleton code for PyTorch is provided (`hw1-q3.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 in 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 in 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`.
   - 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

Tabela 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 (bonus)) 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.