Introduction to Machine Learning **Spring Semester**

Homework 3: July 2, 2024

Due: July 23, 2024

Theory Questions

Remark: Throughout this exercise, when we write a norm $\|\cdot\|$ we refer to the ℓ_2-norm.

1. **(20 points) Convex functions.**

   (a) Let $f : \mathbb{R}^n \to \mathbb{R}$ a convex function, $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$. Show that, $g(\mathbf{x}) = f(A\mathbf{x} + b)$ is convex.

   (b) Consider m convex functions $f_1(\mathbf{x}), \ldots, f_m(\mathbf{x})$, where $f_i : \mathbb{R}^d \to \mathbb{R}$. Now define a new function $g(\mathbf{x}) = \max_i f_i(\mathbf{x})$. Prove that $g(\mathbf{x})$ is a convex function. (Note that from (a) and (b) you can conclude that the hinge loss over linear classifiers is convex.)

   (c) Prove that $f : \mathbb{R}^n \to \mathbb{R}$ defined by $f(\mathbf{x}) = \|\mathbf{x}\|_2^2$ is a convex function. (**Hint:** use the convexity definition which uses the gradient.)

   (d) Let $A \in \mathbb{R}^{n\times n}$ be a PSD matrix (recall the definition of PSD matrix from HW1). Use (a) and (c) to show that $f(\mathbf{x}) = \mathbf{x}^T A\mathbf{x}$ is a convex function.

   (e) Let $\ell_{\log} : \mathbb{R} \to \mathbb{R}$ be the log loss, defined by

   $$\ell_{\log}(z) = \log_2\left(1 + e^{-z}\right)$$

   Show that $\ell_{\log}$ is convex, and conclude that the function $f : \mathbb{R}^d \to \mathbb{R}$ defined by $f(\mathbf{w}) = \ell_{\log}(y\mathbf{w}\cdot\mathbf{x})$ is convex with respect to $\mathbf{w}$.

   (**Hint:** You can use the fact that a twice-differentiable function $f : \mathbb{R} \to \mathbb{R}$ is convex if and only if $f''(z) \geq 0$ for all $z \in \mathbb{R}$.)

2. **(15 points) Log loss with linearly separable data.** Consider a setup of learning linear classifiers over a dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ where $\mathbf{x}_i \in \mathbb{R}^d, y_i \in \{-1, +1\}$ for all i. Assume the data is linearly separable, i.e., given a training set there exists $\mathbf{w}^* \in \mathbb{R}^d$ such that $y_i\mathbf{w}^*\cdot\mathbf{x}_i > 0$ for all i (note the strict inequality; assume no bias for simplicity). Recall the definition of the *log loss* given in lecture #4:

   $$\ell_{\log}(r) = \log_2\left(1 + e^{-r}\right).$$

   We would like to show that in the linearly separable case, minimizing the log loss over the training data will yield a classifier with optimal zero-one loss.

   (a) Let $L(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^n \ell_{\log}(y_i\mathbf{w}\cdot\mathbf{x}_i)$. Show that for every $\epsilon > 0$ there exists $\mathbf{w}_\epsilon \in \mathbb{R}^d$ for which $L(\mathbf{w}_\epsilon) < \epsilon$. (**Hint:** Consider the log loss of $c\mathbf{w}^\star$ for some constant $c > 0$. What happens to the log loss as $c \to \infty$?).

(b) Prove that there exists $\epsilon > 0$ small enough such that any $\mathbf{w}_\epsilon$ for which $L(\mathbf{w}_\epsilon) < \epsilon$ achieves optimal zero-one loss, that is, $sign(\mathbf{w}_\epsilon \cdot \mathbf{x}_i) = y_i$ for all i.

3. **(15 points) Gradient Descent on Smooth Functions.** We say that a continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ is β-smooth if for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$

$$f(\mathbf{y}) \le f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y} - \mathbf{x}) + \frac{\beta}{2}\|\mathbf{x} - \mathbf{y}\|^2$$

In words, β-smoothness of a function f means that at every point $\mathbf{x}$, f is upper bounded by a qaudratic function which coincides with f at $\mathbf{x}$.

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a β-smooth and non-negative function (i.e., $f(\mathbf{x}) \ge 0$ for all $\mathbf{x} \in \mathbb{R}^n$). Consider the (non-stochastic) gradient descent algorithm applied on f with constant step size $\eta > 0$:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \nabla f(\mathbf{x}_t)$$

Assume that gradient descent is initialized at some point $\mathbf{x}_0$. Show that if $\eta < \frac{2}{\beta}$ then

$$\lim_{t \to \infty} \|\nabla f(\mathbf{x}_t)\| = 0$$

(Hint: Use the smoothness definition with points $\mathbf{x}_{t+1}$ and $\mathbf{x}_t$ to show that $\sum_{t=0}^{\infty} \|\nabla f(\mathbf{x}_t)\|^2 < \infty$ and recall that for a sequence $a_n \ge 0$, $\sum_{n=1}^{\infty} a_n < \infty$ implies $\lim_{n\to\infty} a_n = 0$. Note that f is not assumed to be convex!)

4. **(10 points) Expressivity of ReLU networks.** Consider the ReLU activation function:

$$h(x) = \max\{x, 0\}$$

Show that the maximum function $f(x_1, x_2) = \max\{x_1, x_2\}$ can be implemented using a neural network with one hidden layer and ReLU activations. You can assume that there is no activation after the last layer.

(Hint: It is possible to implement the function using a hidden layer with 4 neurons. You may find the following identity useful: $\max\{x_1, x_2\} = \frac{x_1+x_2}{2} + \frac{|x_1-x_2|}{2}$.)

Programming Assignment

Submission guidelines:

- Download the supplied files from Moodle. Written solutions, plots and any other non-code parts should be included in the written solution submission.
- Your code should be written in Python 3.
- Your code submission should include these files: `backprop_main.py`, `backprop_data.py`, `backprop_network.py`.

1. **Neural Networks (40 points).** In this exercise we will implement the back-propagation algorithm for training a neural network. We will work with the MNIST data set that consists of 60000 28x28 gray scale images with values of 0 to 1 in each pixel (0 - white, 1 - black). The optimization problem we consider is of a neural network with ReLU activations and the cross entropy loss. Namely, let $\mathbf{x} \in \mathbb{R}^d$ be the input to the network (in our case $d = 784$) and denote $\mathbf{z}_0 = \mathbf{x}$ and $k_0 = 784$. Then for $0 \le l \le L-2$, define

$$\mathbf{v}_{l+1} = W_{l+1}\mathbf{z}_l + \mathbf{b}_{l+1}$$

$$\mathbf{z}_{l+1} = \sigma(\mathbf{v}_{l+1}) \in \mathbb{R}^{k_{l+1}}$$

and

$$\mathbf{v}_L = W_L \mathbf{z}_{L-1} + \mathbf{b}_L$$

$$\mathbf{z}_L = \frac{e^{\mathbf{v}_L}}{\sum_i e^{v_{L,i}}}$$

where σ is the ReLU function applied element-wise on a vector (recall the ReLU function $\sigma(x) = \max\{0, x\}$) and $W_{l+1} \in \mathbb{R}^{k_{l+1} \times k_l}$, $\mathbf{b}_{l+1} \in \mathbb{R}^{k_{l+1}}$ (k_l is the number of neurons in layer l). Denote by $\mathcal{W}$ the set of all parameters of the network. Then the output of the network (after the softmax) on an input $\mathbf{x}$ is given by $\mathbf{z}_L(\mathbf{x}; \mathcal{W}) \in \mathbb{R}^{10}$ ($k_L = 10$).

Assume we have an MNIST training data set $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$ where $\mathbf{x}_i \in \mathbb{R}^{784}$ is the 28x28 image given in vectorized form and $\mathbf{y}_i \in \mathbb{R}^{10}$ is a one-hot label, e.g., $(0, 0, 1, 0, 0, 0, 0, 0, 0, 0)$ is the label for an image containing the digit 2. Define the cross entropy loss on a single example $(\mathbf{x}, \mathbf{y})$, $\ell_{(\mathbf{x},\mathbf{y})}(W) = -\mathbf{y} \cdot \log \mathbf{z}_L(\mathbf{x}; \mathcal{W})$ where the logarithm is applied element-wise on the vector $\mathbf{z}_L(\mathbf{x}; \mathcal{W})$. The loss we want to minimize is then

$$\ell(\mathcal{W}) = \frac{1}{n}\sum_{i=1}^{n} \ell_{(\mathbf{x}_i,\mathbf{y}_i)}(\mathcal{W}) = \frac{1}{n}\sum_{i=1}^{n} -\mathbf{y}_i \cdot \log \mathbf{z}_L(\mathbf{x}_i; \mathcal{W})$$

The code for this exercise is given in the backprop.zip file in moodle. The code consists of the following:

(a) `backprop_data.py`: Loads the MNIST data.

(b) `backprop_network.py`: Code for creating and training a neural network.

(c) `backprop_main.py`: Example of loading data, training a neural network and evaluating on the test set.

(d) `mnist.pkl.gz`: MNIST data set.

The code in `backprop_network.py` contains the functionality of the training procedure except the code for back-propagation which is missing.

Here is an example of training a one-hidden layer neural network with 40 hidden neurons on a randomly chosen training set of size 10000. The evaluation is performed on a randomly chosen test set of size 5000. It trains for 30 epochs with mini-batch size 10 and constant learning rate 0.1.

```
>>> x_train, y_train, x_test, y_test = load_as_matrix_with_labels(10000, 5000)
>>> net = network.Network([784, 40, 10])
>>> net.train(x_train, y_train, epochs=30, batch_size=10, learning_rate=0.1,
x_test=x_test, y_test=y_test)
```

(a) **(15 points)** Implement the back-propagation algorithm in the *Network* class. You should implement the missing code in `backprop_network.py`, specifically, you should implement the following functions: *backpropagation*, *forward_propagation*, *cross_entropy_derivative*, *relu*, *relu_derivative*. You may add helper functions as you see fit.

There are several functions in `backprop_network.py` which you should implement, carefully go over the skeleton code to see the functions you should implement yourself and other functions you can use as helper functions.

(b) **(10 points)** Train a one-hidden layer neural network as in the example given above (e.g., training set of size 10000, one hidden layer of size 40). For each learning rate in $\{0.001, 0.01, 0.1, 1, 10\}$, plot the *training* accuracy, *training* loss ($\ell(\mathcal{W})$) and *test* accuracy across epochs (3 plots: each contains the curves for all learning rates). Note that the SGD function calculates and returns the loss and accuracy for both the training and test sets across epochs.

The test accuracy with leaning rate 0.1 in the final epoch should be above 93%.

What happens when the learning rate is too small or too large? Explain the phenomenon.

(c) **(5 points)** Now train the network on the whole training set and test on the whole test set. What is the test accuracy on the test set in the final epoch (should be above 95%)?

(d) **(10 points)** Using a step size of 0.1, train a linear multiclass classifier (that is, a network with no hidden layer) on the entire dataset, for 30 epochs. Plot the training and test accuracy across epochs. The trained classifier is characterized by a weight matrix $W \in \mathbb{R}^{10,784}$ (and a bias vector, which we ignore). Each row of W can be reshaped into a 28 x 28 image and plotted using the following `matplotlib.pyplot` function: `imshow(reshape(W[i], (28, 28)), interpolation='nearest')`. Use this to plot each of the 10 rows of W as an image. What do you see? Explain.

(e) **(5 points bonus)** Explore different structures and parameters and try to improve the test accuracy. Use the whole test set. In order to get full credit you should get accuracy of more than 97%. Any improvement over the previous clauses will give you 5 points. The maximum score for this homework set remains 100.