**Introduction to Machine Learning** **Spring Semester**

Homework 4: May 9, 2023

*Due: May 24, 2023*

## Theory Questions

1. **(15 points) SVM with multiple classes.** One limitation of the standard SVM is that it can only handle binary classification. Here is one extension to handle multiple classes. Let $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ and now let $y_1, \ldots, y_n \in [K]$, where $[K] = \{1, 2, \ldots, K\}$. We will find a separate classifier $\mathbf{w}_j$ for each one of the classes $j \in [K]$, and we will focus on the case of no bias ($b = 0$). Define the following loss function (known as the *multiclass hinge-loss*):

$$\ell(\mathbf{w}_1, \ldots, \mathbf{w}_K, \mathbf{x}_i, y_i) = \max_{j \in [K]} (\mathbf{w}_j \cdot \mathbf{x}_i - \mathbf{w}_{y_i} \cdot \mathbf{x}_i + \mathbb{1}(j \neq y_i)),$$

   where $\mathbb{1}(\cdot)$ denotes the indicator function. Define the following multiclass SVM problem:

$$f(\mathbf{w}_1, \ldots, \mathbf{w}_K) = \frac{1}{n} \sum_{i=1}^{n} \ell(\mathbf{w}_1, \ldots, \mathbf{w}_K, \mathbf{x}_i, y_i)$$

   After learning all the $\mathbf{w}_j$, $j \in [K]$, classification of a new point $\mathbf{x}$ is done by $\arg\max_{j \in [K]} \mathbf{w}_j \cdot \mathbf{x}$. The rationale of the loss function is that we want the "score" of the true label, $\mathbf{w}_{y_i} \cdot \mathbf{x}_i$, to be larger by at least 1 than the "score" of each other label, $\mathbf{w}_j \cdot \mathbf{x}_i$. Therefore, we pay a loss if $\mathbf{w}_{y_i} \cdot \mathbf{x}_i - \mathbf{w}_j \cdot \mathbf{x}_i \leq 1$, for $j \neq y_i$.

   Consider the case where the data is linearly separable. Namely, there exists $\mathbf{w}_1^*, ..., \mathbf{w}_K^*$ such that $y_i = argmax_y \mathbf{w}_y^* \cdot \mathbf{x}_i$ for all $i$. Show that any minimizer of $f(\mathbf{w}_1, ..., \mathbf{w}_K)$ will have zero classification error.

2. **(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: $\max\{x_1, x_2\} = \frac{x_1 + x_2}{2} + \frac{|x_1 - x_2|}{2}$.)

3. **(15 points) Soft SVM with $\ell^2$ penalty.** Consider the following problem:

$$\min_{\boldsymbol{w}\in\mathbb{R}^d, b\in\mathbb{R}, \boldsymbol{\xi}\in\mathbb{R}^n} \frac{1}{2}\|\boldsymbol{w}\|^2 + \frac{C}{2}\sum_{i=1}^{n}\xi_i^2$$
$$\text{s.t. } y_i(\boldsymbol{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \quad \forall i = 1, \ldots, n$$

   (a) Show that a constraint of the form $\xi_i \geq 0$ will not change the problem. Meaning, show that these non-negativity constraints can be removed. That is, show that the optimal value of the objective will be the same whether or not these constraints are present.

   (b) What is the Lagrangian of this problem?

   (c) Minimize the Lagrangian with respect to $\boldsymbol{w}, b, \boldsymbol{\xi}$ by setting the derivative with respect to these variables to 0.

   (d) What is the dual problem?

4. **(15 points) Gradient of cross-entropy loss over softmax**. Let $\mathbf{y} \in \{0,1\}^d$ be a one-hot vector, i.e. $\mathbf{y}$ has a single entry which is 1 and the rest are 0. Consider the following loss function $\ell_{\mathbf{y}} : \mathbb{R}^d \to \mathbb{R}$:

$$\ell_{\mathbf{y}}(\mathbf{w}) = -\mathbf{y} \cdot \log\left(\text{softmax}(\mathbf{w})\right),$$

   where $\text{softmax}(\mathbf{w}) = \frac{e^{\mathbf{w}}}{\sum_{j=1}^{d} e^{w_j}}$, is the softmax function (see slide 7 of lecture #7). In the above notation, the exponent and logarithm function operate elementwise when given a vector as an input. The function $\ell_{\mathbf{y}}$ is known as the *cross-entropy loss*, and you will encounter it in the programming assignment.

   Prove that the gradient of $\ell_{\mathbf{y}}$ with respect to $\mathbf{w}$ is given by:

$$\nabla\ell_{\mathbf{y}}(\mathbf{w}) = \text{softmax}(\mathbf{w}) - \mathbf{y}$$

## 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`, `alexnet.ipynb`.
- The code for question 2 in the programming assignment should be submitted in `.ipynb` format (a Python notebook).

1. **Neural Networks (30 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 $0 \leq l \leq L-2$, 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 $\sigma$ 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.

```
>>> training_data, test_data = data.load(train_size=10000,test_size=5000)
>>> net = network.Network([784, 40, 10])
>>> net.SGD(training_data, epochs=30, mini_batch_size=10, learning_rate=0.1,
test_data=test_data)
```

(a) **(15 points)** Implement the back-propagation algorithm in the *backprop* function in the *Network* class. The function receives as input a 784 dimensional vector $\mathbf{x}$ and a one-hot vector $\mathbf{y}$. The function should return a tuple $(db, dw)$ such that $db$ contains a list of derivatives of $\ell_{(\mathbf{x},\mathbf{y})}$ with respect to the biases and $dw$ contains a list of derivatives with respect to the weights. The element $dw[i]$ (starting from 0) should contain the matrix $\frac{\partial \ell_{(\mathbf{x},\mathbf{y})}}{\partial W_{i+1}}$ and $db[i]$ should contain the vector $\frac{\partial \ell_{(\mathbf{x},\mathbf{y})}}{\partial \mathbf{b}_{i+1}}$.

(Hint: Use question 4 of the theoretical assignment to calculate $\frac{\partial \ell_{(\mathbf{x},\mathbf{y})}}{\partial \mathbf{v}_L}$ for the initial backpropagation step. You can also use the fact that the gradients of the loss with respect to the biases are given by $\frac{\partial \ell_{(\mathbf{x},\mathbf{y})}}{\partial \mathbf{b}_l} = \boldsymbol{\delta}_l \circ \sigma'(\mathbf{v}_l)$.)

You can use the *loss* function in the *Network* class to calculate $\ell_{(\mathbf{x},\mathbf{y})}(\mathcal{W})$.

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 $\{0.001, 0.01, 0.1, 1, 10, 100\}$, plot the *training* accuracy, *training* loss ($\ell(\mathcal{W})$) and *test* accuracy across epochs (3 plots: each contains the curves for all learning rates). For the test accuracy you can use the *one_label_accuracy* function, for the training accuracy use the *one_hot_accuracy* function and for the training loss you can use the *loss* function. All functions are in the *Network* class.

The test accuracy with leaning rate 0.1 in the final epoch should be above 80%.

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:

```
>>> training_data, test_data = data.load(train_size=50000,test_size=10000)
>>> net = network.Network([784, 40, 10])
>>> net.SGD(training_data, epochs=30, mini_batch_size=10, learning_rate=0.1,
test_data=test_data)
```

Do **not** calculate the training accuracy and training loss as in the previous section (this is time consuming). What is the test accuracy in the final epoch (should be above 90%)?

(d) **(10 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 96%. Any improvement over the previous clauses will give you 5 points. The maximum score for this homework set remains 100.

2. **Training a deep Convolutional Neural Network (15 points).** In this exercise you will train a deep convolutional neural network architecture called AlexNet [1], for image classification using PyTorch. Download the `alexnet.ipynb` file from Moodle, which contains code for training AlexNet to classify images from the dataset CIFAR10 (for information on the dataset, click here).

   - We **highly** recommend that you use Google Colab to run your code for this exercise, as it enables you to train your network with a GPU which considerably speeds up the training time. In order to use a GPU in Colab, click `Runtime -> Change runtime type` and select `GPU`. The code is already modified so that training will be performed with a GPU if it is available.

   (a) **(5 points)** Train the network **on 10% of the training set** (otherwise the training process would take a long time) once with the default parameters given in the code, and then again using the Adam optimizer instead of SGD (see the documentation in `torch.optim.Adam` for more information), with a learning rate of 0.00005. How is the training process affected by this change? Do you achieve better training loss? What about the validation accuracy?

   (b) **(5 points)** Again using Adam, train the network with a learning rate of 0.005 instead of the default 0.00005. How is the training process affected?

   (c) **(5 points)** Train the network using Adam and a step size of 0.00005, but change the architecture so that it doesn't include Batch Normalization or Dropout layers (simply comment out the relevant lines in the network's definition). How does the training loss evolve when compared to the one with BatchNorm + Dropout? Do these additions seem to accelerate the learning process?

   (d) **(5 points bonus)** Make whatever changes and tweaks in the network structure or other training parameters (except for adding more epochs - which would obviously improve the results) in order to obtain as good a test accuracy as you can get. **Train on the entire training set**. Take this opportunity to be creative, and try to learn more about common methods for training neural networks in practice. In order to get the 5 points bonus, you should get a test accuracy of at least 85%. Clearly state which changes or improvements you made in the training process.

[1] More info here.