Introduction to Machine Learning **Spring Semester**

Homework 4: April 6, 2022

Due: April 27, 2022

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*):

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

   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 $y_i = argmax_y \mathbf{w}_y^* \cdot \mathbf{x}_i$. Show that any minimizer of $f(\mathbf{w}_1, ..., \mathbf{w}_K)$ will have zero classification error.

2. **(10 points)** Consider the soft-SVM problem with seperable data:

$$\begin{aligned} &\min_{\mathbf{w}, \xi} 0.5\|\mathbf{w}\|^2 + C \sum_{i=1}^{n} \xi_i \\ s.t\ \forall i: \quad &y_i \mathbf{w} \cdot \mathbf{x}_i \geq 1 - \xi_i \\ &\xi_i \geq 0 \end{aligned}$$

   Let $\mathbf{w}^\star$ be the solution of **hard SVM**. Show that if $C \geq \|\mathbf{w}^\star\|^2$ then the solution of soft SVM separates the data. (Hint: Show that in the optimal solution $(\mathbf{w}', \boldsymbol{\xi}')$ of the soft SVM problem, the sum of ξ_i''s is bounded by some constant smaller than 1).

3. **(15 points) Separability using polynomial kernel.** Let $x_1, ..., x_n \in \mathbb{R}$ be distinct real numbers, and let $q \geq n$ be an integer. Show that when using a polynomial kernel, $K(x, x') = (1 + xx')^q$, hard SVM achieves zero training error. Use the following fact: Given distinct values $\alpha_1, ..., \alpha_n$, the Vandermonde matrix defined by,

$$\begin{pmatrix} 1 & \alpha_1^1 & ... & \alpha_1^q \\ 1 & \alpha_2^1 & ... & \alpha_2^q \\ \vdots & & & \\ 1 & \alpha_n^1 & ... & \alpha_n^q \end{pmatrix},$$

is of rank n. (Hint: use the lemma from slide 16 in recitation 6).

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

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

In this exercise you will show how to implement the maximum function $f(x_1, x_2) = \max\{x_1, x_2\}$ using a neural network with one hidden layer and ReLU activations.

(a) Prove the following equalities:
- $x = \max\{x, 0\} - \max\{-x, 0\}$
- $|x| = \max\{x, 0\} + \max\{-x, 0\}$
- $\max\{x_1, x_2\} = \frac{x_1+x_2}{2} + \frac{|x_1-x_2|}{2}$

(b) Use (a) to show that it is possible to implement the maximum function using a ReLU network with one hidden layer which has 4 neurons, using ReLU activations (you can assume that there is no activation after the last layer). Explicitly write the parameters (weights) of the neural network you obtain.

5. **(10 points) Implementing boolean functions using ReLU networks.** Consider n boolean input variables, i.e. $x_1, \ldots, x_n \in \{0, 1\}$. Your goal is to construct a neural network with ReLU activations, which implements the AND function:

$$f(x_1, \ldots, x_n) = x_1 \wedge x_2 \wedge \ldots \wedge x_n$$

Show how to construct a ReLU network with a constant number of layers which implements f. Describe the network's structure and parameters.

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: `svm.py`, `backprop_main.py`, `backprop_data.py`, `backprop_network.py`.

1. **(15 points) SVM.** In this exercise, we will explore different polynomial kernel degrees for SVM. We will use an existing implementation of SVM: the SVC class from sklearn.svm. This class solves soft-margin SVM problem. In the file `svm.py` you will find the method `plot_results` which gets following as an input:

   - A list of fitted estimators. That is, each element of the list is a return value of the fit method of the SVM model.
   - A list of names that corresponds to the models above.
   - Data points in $\mathbb{R}^2$: a numpy array of shape $n \times 2$, where n is the number of data points.
   - A numpy array of lables in $\{-1, 1\}$ for the above data points.

   The method plots the data points and the classifiers' prediction.

   (a) **(5 points)** The code you are given generates 200 samples (100 samples for each class) which are classified by a circle centered around the origin. Train three soft SVM models with regularization parameter $C = 10$, using linear kernel, **homogeneous** polynomial kernel of degree 2 and **homogeneous** polynomial kernel of degree 3. Plot your results using the methods above. Which of the models fits the data well? Explain the phenomena you see in the plots.

   (b) **(5 points)** Repeat clause (a) above but now with **non-homogeneous** polynomial kernel. Do the results change? Explain.

   (c) **(5 points)** Perturb the labels in the following manner: Change each negative label to a positive one with probability 0.1. Train a soft-SVM model with a polynomial kernel of degree 2 (non-homogeneous), and another model using RBF kernel with $\gamma = 10$. Which of the two models seems to generalize better on the noisy data? What happens if you change γ? Submit your plots and explain the phenomena you see.

2. **Neural Networks (25 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 $n_0 = 784$. Then for $0 \leq l \leq L-2$, define

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

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

and

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

$$\mathbf{z}_L = \frac{e^{\mathbf{v}^L}}{\sum_i e^{\mathbf{v}_i^L}}$$

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 $\mathbf{z}_L(\mathbf{x}; \mathcal{W}) \in \mathbb{R}^{10}$ ($n_L = 10$).

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 log-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) `skeleton_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 `skeleton_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) **(10 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 $\frac{\partial \ell_{(\mathbf{x},\mathbf{y})}}{\partial \mathbf{b}_{i+1}}$.
You can use the *loss* function in the *Network* class to calculate $\ell_{(\mathbf{x},\mathbf{y})}(\mathcal{W})$.

(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 plot: 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 to the previous clauses will give you 5 points. The maximum score for this homework set remains 100.