Introduction to Machine Learning **Fall Semester**

Homework 5: December 21, 2022

Due: January 4, 2023

Theory Questions

1. **(10 points) Suboptimality of ID3.** Solve exercise 2 in chapter 18 in the course book: Understanding Machine Learning: From Theory to Algorithms.

2. **(20 points) Properties of KL divergence**. Recall the definition of the KL-divergence from slide 15 in recitation 8.

   (a) Show that the KL-divergence is always non-negative.
   (Hint: Consider the convex function $f(y) = y \log y$, and use Jensen's inequality which states that for any distribution q, $\mathbb{E}_{z\sim q}[f(z)] \geq f\left(\mathbb{E}_{z\sim q}[z]\right)$).

   (b) Let p_1, p_2, q_1, q_2 be distributions over $\mathcal{X}$ such that p_1 is independent of p_2 and q_1 is independent of q_2. Denote the product distributions $p = p_1 \times p_2$ and $q = q_1 \times q_2$ over $\mathcal{X}^2$ (i.e. $p(x_1, x_2) = p_1(x_1)p_2(x_2)$ for any $x_1, x_2 \in \mathcal{X}$ and similarly for q). Prove the following:

$$D_{KL}(p, q) = D_{KL}(p_1, q_1) + D_{KL}(p_2, q_2).$$

3. **(20 points) AdaBoost.** Let $x_1, \ldots, x_m \in \mathbb{R}^d$ and $y_1, \ldots, y_m \in \{-1, 1\}$ its labels. We run the AdaBoost algorithm as given in the lecture, and we are in iteration t. Assume that $\epsilon_t > 0$.

   (a) Show that the error of the current hypothesis relative to the new distribution is exactly $1/2$, that is:

$$\Pr_{x\sim D_{t+1}}[h_t(x) \neq y] = \frac{1}{2}.$$

   (b) Show that AdaBoost will not pick the same hypothesis twice consecutively; that is $h_{t+1} \neq h_t$.

4. **(20 points) Sufficient Condition for Weak Learnability.** Let $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ be a training set and let $\mathcal{H}$ be a hypothesis class. Assume that there exists $\gamma > 0$, hypotheses $h_1, \ldots, h_k \in \mathcal{H}$ and coefficients $a_1, \ldots, a_k \geq 0$, $\sum_{i=1}^{k} a_i = 1$ for which the following holds:

$$y_i \sum_{j=1}^{k} a_j h_j(x_i) \geq \gamma \tag{1}$$

   for all $(x_i, y_i) \in S$.

   (a) Show that for any distribution D over S there exists $1 \leq j \leq k$ such that

$$\Pr_{i\sim D}[h_j(x_i) \neq y_i] \leq \frac{1}{2} - \frac{\gamma}{2}.$$

   (Hint: Take expectation of both sides of inequality (1) with respect to D.)
   Remark: Note that the condition above is sufficient for *empirical* weak learnability, the condition defined in lecture #9 for the Adaboost analysis.

(b) Let $S = \{(x_1, y_1), \ldots, (x_n, y_n)\} \subseteq \mathbb{R}^d \times \{-1, 1\}$ be a training set that is realized by a d-dimensional hyper-rectangle classifier, i.e., there exists a d dimensional hyper-rectangle $[b_1, c_1] \times \cdots \times [b_d, c_d]$ which contains all of the positive points in S and doesn't contain the negative points in S. Let $\mathcal{H}$ be the class of decision stumps of the form

$$h(x) = \begin{cases} 1 & x_j \leq \theta \\ -1 & x_j > \theta \end{cases}, \quad h(x) = \begin{cases} 1 & x_j \geq \theta \\ -1 & x_j < \theta \end{cases},$$

for $1 \leq j \leq d$ and $\theta \in \mathbb{R} \cup \{\infty, -\infty\}$ (for $\theta \in \{\infty, -\infty\}$ we get constant hypotheses which predict always 1 or always -1). Show that there exist $\gamma > 0$, $k > 0$, hypotheses $h_1, \ldots, h_k \in \mathcal{H}$ and $a_1, \ldots, a_k \geq 0$ with $\sum_{i=1}^{k} a_i = 1$, such that the condition in inequality (1) holds for the training set S and hypothesis class $\mathcal{H}$. This implies that $\mathcal{H}$ is empirically weak learnable w.r.t. data realizable by a d-dimensional hyper-rectangle.
(Hint: Set $k = 4d - 1$, $a_i = \frac{1}{4d-1}$ and let $2d - 1$ of the hypotheses be constant.)

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 (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 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.

```
>>> 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 $\frac{\partial \ell_{(\mathbf{x},\mathbf{y})}}{\partial \mathbf{b}_{i+1}}$.
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.