Homework Set Two
ECE 271B - Winter 2019

ECE 271B
Department of Electrical and Computer Engineering
University of California, San Diego

Problem 1. In this problem, we will consider the limitations of the linear classifier.

a) Consider the set of binary classification problems, where the observation variable $\mathbf{X} = (X_1, X_2)$ is binary, i.e. $\mathbf{X} \in \{0,1\} \times \{0,1\}$, and non-degenerate, i.e. all possible configurations (X_1, X_2) have non-zero probability. Consider the decision function

$$h(\mathbf{x}) = \begin{cases} 1, & \mathbf{w}^T\mathbf{x} + b \geq 0 \\ 0, & \mathbf{w}^T\mathbf{x} + b < 0 \end{cases}$$

and let the class label $Y \in \{0,1\}$ be a deterministic function of $\mathbf{X}$, $Y = f(\mathbf{X})$. Show that there is at least one binary mapping

$$f : \{0,1\} \times \{0,1\} \to \{0,1\}$$

such that the observation $\mathbf{X}$ cannot be classified with zero probability of error with the decision function $h(\mathbf{x})$. Given that, by making $Y = f(\mathbf{X})$, it is easy to obtain zero probability of error on this problem, this observation is clearly problematic. Compute the smallest probability of error achievable with your function $f(\cdot)$, when all configurations of $\mathbf{X}$ are equally likely.

b) The extension of the linear classifier to problems with M classes is to define a set of functions

$$g_i(\mathbf{x}) = \mathbf{w}_i^T\mathbf{x} + b_i,\ i = 1, \ldots, M$$

and assign $\mathbf{x}$ to class i if $\mathbf{x} \in \mathcal{R}_i$, where

$$\mathcal{R}_i = \{\mathbf{x} \,|\, g_i(\mathbf{x}) \geq g_j(\mathbf{x}),\ \forall j \neq i\}.$$

Show that this classifier can only implement *convex* decision regions by showing that, if $\mathbf{x}_1 \in \mathcal{R}_i$ and $\mathbf{x}_2 \in \mathcal{R}_i$, then $\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2 \in \mathcal{R}_i$, $\forall\, 0 \leq \lambda \leq 1$.

Problem 2. Let $g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$ and consider the hyperplane $g(\mathbf{x}) = 0$.

a) Show that the Euclidean distance from a point $\mathbf{x}_a$ to the hyperplane is $|g(\mathbf{x}_a)|/||\mathbf{w}||$ by minimizing $||\mathbf{x} - \mathbf{x}_a||^2$ subject to $g(\mathbf{x}) = 0$.

b) Show that the projection of $\mathbf{x}_a$ onto the hyperplane is

$$\mathbf{x}_p = \mathbf{x}_a - \frac{g(\mathbf{x}_a)}{||\mathbf{w}||^2}\mathbf{w}.$$

Problem 3. Given a training set $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, the linear regression problem consists of finding the parameters $\mathbf{a} = (\mathbf{w}, b)$ of the function

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

which minimize the empirical risk

$$L(\mathbf{w}, b) = \sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i - b)^2.$$

a) Show that the optimal solution is of the form

$$\mathbf{a} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

and determine how the matrix $\mathbf{X}$ and vector $\mathbf{y}$ depend on the training patterns.

b) The matrix inversion above can be too expensive for some applications. Derive a stochastic gradient descent algorithm for this problem, i.e. an algorithm similar to the one we have studied in class for the Perceptron.

c) We have discussed in class that it is usually a good idea to penalize "complicated solutions." In the regression problem, this can be accomplished by minimizing the alternative empirical risk

$$L(\mathbf{w}, b) = \sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i - b)^2 + \lambda\mathbf{w}^T\mathbf{w},$$

resulting in what is usually called *ridge regression*. The second term favors solutions that make $\mathbf{w}$ close to zero, therefore penalizing those that have many degrees of freedom. Show that the optimal solution can be written as

$$\mathbf{w}^* = \sum_i \alpha_i \mathbf{x}_i$$

and use this to re-write $L(\mathbf{w}, b)$ in a form that only depends on the training patterns $\mathbf{x}_i$ through their dot-products. Solve the resulting problem to obtain the function

$$y = f(\mathbf{x})$$

in a form that only depends on the training patterns through their dot-products with $\mathbf{x}$.

Problem 4. We have seen in class that, for a training set $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, the Perceptron algorithm can be implemented as follows.

```
• set $\mathbf{w}_0 = \mathbf{0}, b_0 = 0, k = 0, R = \max_i ||\mathbf{x}_i||$
• repeat
    – for $i = 1, \ldots, n$
        ∗ if $y_i(\mathbf{w}_k^T \mathbf{x}_i + b_k) \leq 0$ then
            · set $\mathbf{w}_{k+1} = \mathbf{w}_k + \eta y_i \mathbf{x}_i$
            · set $b_{k+1} = b_k + \eta y_i R^2$
            · set $k = k + 1$
• until there are no classification errors
```

a) Is the learning rate η relevant? Why?

b) Show that an equivalent algorithm is

```
• set $\alpha = \mathbf{0}, b = 0, R = \max_i ||\mathbf{x}_i||$
• repeat
    – for $i = 1, \ldots, n$
        ∗ if $y_i(\sum_{j=1}^n \alpha_j y_j \mathbf{x}_j^T \mathbf{x}_i + b) \leq 0$ then
            · set $\alpha_i = \alpha_i + 1$
            · set $b = b + y_i R^2$
• until there are no classification errors
```

c) Can you give an interpretation to the parameters α_i? Which among the samples $\mathbf{x}_i$ are the hardest to classify?

d) One of the interesting properties of this implementation is that it only depends on the dot-products $\mathbf{x}_i^T \mathbf{x}_j$. Can you re-write the decision function

$$h(\mathbf{x}) = \text{sgn}(\mathbf{w}^T \mathbf{x} + b)$$

in this form?

Problem 5. In this problem, we will design a neural network for image classification. In all questions, you will use the training set contained in the directory `MNISTtrain` (60,000 examples) and the test set in the directory `MNISTtest` (10,000). To read the data, you should use the script `readMNIST.m` (use readDigits=60,000 or readDigits=10,000, respectively, and offset=0). This returns two matrices. The first (`imgs`) is a matrix with $n \in \{10000, 60000\}$ rows, where each row is a 28×28 image of a digit, vectorized into a 784-dimensional row vector. The second (`labels`) is a matrix with $n \in \{10000, 60000\}$ rows, where each row contains the class label for the corresponding image in `imgs`. In all problems, you should initialize the weights as samples from independent Gaussian random variables of zero mean and unit variance. Note that the network should have a bias term. In what follows, we assume homogeneous coordinates.

a) We start by designing a **single layer** neural network. For input $\mathbf{x}^n$, the activation of unit k is given by

$$a_k^n = \mathbf{w}_k^T \mathbf{x}^n.$$

This is then fed to the softmax non-linearity

$$y_k^n = \frac{\exp(a_k^n)}{\Sigma_j \exp(a_j^n)},$$

which produces an estimate $y_k^n = P_{Z|X}(k|\mathbf{x}^n)$ that $\mathbf{x}^n$ belongs to class $Z = k$. Given a training set $\{(\mathbf{x}^n, t^n)\}$, where t^n is the class label for example $\mathbf{x}^n$, the network is trained to minimize the cross-entropy cost

$$E(\mathbf{w}) = -\sum_n \sum_{k=1}^{c} t_k^n \ln y_k^n,$$

where $\mathbf{w}$ is the vector of all weights. As usual, we use backpropagation, which reduces to the weight updates

$$w_{jk}^{(t+1)} = w_{jk}^{(t)} - \eta \frac{\partial E^n(\mathbf{w})}{\partial w_{jk}},$$

where $E^n(\mathbf{w})$ is the term of the cost due to example $\mathbf{x}^n$.

i) Show that the gradient is

$$-\frac{\partial E^n(\mathbf{w})}{\partial w_{jk}} = (t_k^n - y_k^n) x_j^n.$$

ii) To actually learn the network, we use what is called *batch* learning. This consists of using all data to update the weight of each unit, i.e.

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \sum_{n=1}^{N} \nabla E^n(\mathbf{w}).$$

Implement this learning algorithm using $\eta = 10^{-5}$. Make a plot of the train and test set error, as a function of the iteration t. Report the final training and test errors.

b) We next consider a network of **two layers**. Assume that the first layer has H units, all with non-linearity $\sigma(\cdot)$. Again, the output layer uses the softmax non-linearity and the cost function is the cross-entropy.

i) Derive the equations for backpropagation by *stochastic gradient descent* for this network. Note that this involves a different set of derivatives for the first and second layer.

ii) Assuming a sigmoid non-linearity

$$\sigma(u) = \frac{1}{1+e^{-u}},$$

implement the *batch* training procedure to train the two layer network. Again, use $\eta = 10^{-5}$, make a plot of the train and test set error, as a function of the iteration t, and report the final training and test errors. Repeat the procedure for $H \in \{10, 20, 50\}$. Compare the performance of the three networks.

iii) We next compare the sigmoid and ReLU non-linearities. However, if you attempt to repeat the procedure of the previous question with the ReLU, you will see that the weights will explode. To avoid this, we introduce weight decay, i.e. a cost of the form

$$E = -\sum_n \sum_{k=1}^{c} t_k^n \ln y_k^n + \frac{\lambda}{\eta} ||\mathbf{w}||^2.$$

Train a two-layer network with sigmoid hidden units and another with ReLU hidden units. Again, use the softmax on the output layer. In these experiments, set $\lambda = 0.001$ and $\eta = 2 \times 10^{-6}$. Make a plot of the training and test set error as a function of the iteration t, and report the final training and test errors. Repeat the procedure for $H \in \{10, 20, 50\}$. Compare the performance of the six networks. Which non-linearity achieves better performance and faster network convergence?

iv) We next consider the importance of the regularization weight. Repeat **iii)** with $\lambda = 0.0001$. What can you say about the importance of regularization?

c) Repeat **a) ii)** and **b) iii)** using stochastic gradient descent instead of the batch approach. Comment on the differences. (Note: in this case, use learning rate $\eta = 10^{-2}$ for the sigmoid network and $\eta = 2 \times 10^{-3}$ for the ReLU network. Do not use weight decay.)