Programming Assignment 1

CSE 190: Neural Networks

Fall 2017

Instructions

Due on Tuesday, October 10th, 2016

1. Please submit your assignment on Vocareum. There are two components to this assignment: written homework (Problems 1,2, You will be writing a report in a conference paper format for this assignment, reporting your findings. While we won't enforce this, we prefer the report to be written using LaTeX or Word in NIPS format (NIPS is the top machine learning conference, and it is now dominated by deep nets - it will be good practice for you to write in that format!). You are free to choose an alternate format, but NIPS format is strongly recommended. The templates, both in **Word** and LaTeX are available from the 2015 NIPS format site.

2. You may use a language of your choice (Python and MATLAB are recommended). You also need to submit all of the source code files and a *readme.txt* file that includes detailed instructions on how to run your code.

   You should write clean code with consistent format, as well as explanatory comments, as this code may be reused in the future.

3. Using the MATLAB neural network toolbox or any off-the-shelf code is strictly prohibited.

4. Any form of copying, plagiarizing, grabbing code from the web, having someone else write your code for you, etc., is cheating. We expect you all to do your own work, and when you are on a team, to pull your weight. Team members who do not contribute will not receive the same scores as those who do. Discussions of course materials and homework solutions are encouraged, but you should write the final solutions alone. Books, notes, and Internet resources can be consulted, but not copied from. Working together on homework must follow the spirit of the **Gilligan's Island Rule** (Dymond, 1986): No notes can be made (or recording of any kind) during a discussion, and you must watch one hour of Gilligan's Island or something equally insipid before writing anything down. Suspected cheating has been and will be reported to the UCSD Academic Integrity office.

Part I

Homework problems to be solved individually, and turned in individually

1. **P**erceptrons (12 points)

   Recall the perceptron activation rule:

$$y = \begin{cases} 1 & \text{if } \sum_i^d w_i x_i \geqslant \theta \\ 0 & \text{else} \end{cases}$$

   (a) (2 pts) Assuming $d = 2$, derive the equation for the line that is the decision boundary

(b) (3 pts) prove that the distance from the decision boundary to the origin is given by:

$$l = \frac{w^T x}{\|w\|}$$

In class, we showed how to learn the "OR" function using the perceptron learning rule. Now we want to learn the "NAND" function using four patterns, as shown in Table 1.

Input	Output
0 0	1
0 1	1
1 0	1
1 1	0

Table 1: The "NAND" function

(c) (1 pt) Write down the perceptron learning rule as an update equation.

(d) (4pts) Draw the rest of this table, as we did in class, as the network learns NAND. Initialize w_1, w_2 and θ to be 0 and fix the learning rate to 1. Add one row for each randomly selected pattern (training example) for the perceptron to learn. Stop when the learning converges. Please present the resulting table and show the final learned weights and threshold. You may pick the "random" order to make the learning converge faster, if you can.

x_1	x_2	Net	Output	Teacher	w_1	w_2	Threshold θ $(= -w_0)$

(e) (2 pts) Is the solution unique? Why or why not?

2. **Logistic Regression (5 points)**

Logistic regression is a binary classification method. Intuitively, logistic regression can be conceptualized as a single neuron reading in a d-dimensional input vector $x \in \mathbb{R}^d$ and producing an output y between 0 and 1 that is the system's estimate of the conditional probability that the input is in some target category. The "neuron" is parameterized by a weight vector $w \in \mathbb{R}^{d+1}$, where w_0 represents the bias term (a weight from a unit that has a constant value of 1).

$$y = P(\mathcal{C}_1|x) = \frac{1}{1 + \exp(-w^\top x)} = g(w^\top x) \tag{1}$$

$$P(\mathcal{C}_0|x) = 1 - P(\mathcal{C}_1|x) = 1 - y, \tag{2}$$

where we assume that x has been augmented by a leading 1 to represent the bias input. With the parameterized model so defined, we now define the Cross Entropy cost function, equation 3, the quantity we want to minimize over our training examples:

$$E(w) = -\sum_{n=1}^{N} \{t^n \ln(y^n) + (1 - t^n) \ln(1 - y^n)\}. \tag{3}$$

Here, $t^n \in \{0, 1\}$ is the label or teaching signal for example n ($t^n = 1$ represents $x^n \in \mathcal{C}_1$). We minimize this cost function via gradient descent.

To do so, we need to derive the gradient of the cost function with respect to the parameters w_j. Assuming we use the logistic activation function g as in equation 1, show that this gradient is:

$$-\frac{\partial E(w)}{\partial w_j} = \sum_{n=1}^{N} (t^n - y^n) x_j^n \tag{4}$$

Show work.

3. **Softmax Regression (7 points)**

Softmax regression is the generalization of logistic regression for multiple classes. Given an input x^n, softmax regression outputs a vector, $(y_1^n, y_2^n, ..., y_c^n)$ where y_i^n represents $P(\mathcal{C}_i|x^n)$, the probability that input x^n is in category $\mathcal{C}_i$. The model, which is a generalization of the logistic, is:

$$y_i = P(\mathcal{C}_i|x^n) = \frac{exp(w_i^\top x^n)}{\sum_{k=1}^{c} exp(w_k^\top x^n)} \tag{5}$$

which clearly is positive for each y_i, and sums to 1 over all of the y_i's. Our cost function is the cross-entropy loss for $c > 2$ categories (often called "the softmax loss" these days):

$$E = -\sum_{n=1}^{N} \sum_{k=1}^{c} t_k^n \ln y_k^n \tag{6}$$

Here, we are using a 1 out-of-c encoding (also known as the "one hot" encoding) that is, in a 10-class situation (e.g., 10 digits, to pick a random example), if the category is $k = 5$, then $t_5 = (0, 0, 0, 0, 0, 1, 0, 0, 0, 0)$. ($t_0$ represents the digit 0).

For the cross entropy cost function, equation 6, show that the gradient is:

$$-\frac{\partial E(w)}{\partial w_{jk}} = \sum_{n=1}^{N} (t_k^n - y_k^n) x_j^n \tag{7}$$

Show work. Here, w_{jk} is the weight from input x_j to unit y_k of the softmax layer.

Hint: Recall your logarithm rules, such as $\log(\frac{a}{b}) = \log(a) - \log(b)$. Additionally, note that the error function sums over all our classes, specifically, from $j = 1$ to $j = c$. Which terms of this sum will be affected by the gradient with respect to w_{jk}?

Part II

Team Programming Assignment

1. **Perceptrons** (15 points).
In this problem, we will use a perceptron classifier to identify two types of flower species from their features. We will use a modified version of the popular UCI Iris dataset, provided in hw1_iris.tar.gz under "resources" in piazza. Unpack the tarball, and in the file *iris_train.data* in the iris folder is the data. Each row of the file is an input-output pattern. There are 5 columns in each row, with the first four numbers representing the following input variables: sepal length in cm; sepal width in cm; petal length in cm; and petal width in cm. The last column is the class label, either Iris-setosa or Iris-versicolor.

   (a) Z-score each attribute (feature) of data by the equation:

   $$\tilde{x_i}^n = (x_i^n - \mu_{x_i})/\sigma_{x_i}$$

   where μ_{x_i} and σ_{x_i} are the mean and standard deviation of the i^{th} input variable, respectively, over the training set. That is, the z-scoring is on a per-column basis. This transformed data is what you will use to train the perceptron. Keep track of μ_{x_i} and σ_{x_i} and use the same numbers to z-score the test data (we can't necessarily assume that we will see all of the test data at once, so we have to assume it comes from the same distribution). You will find that z-scoring is a common pre-processing step to normalize input variables. Can you explain why it is useful when training a perceptron? (*2 points*)

   (b) Using the MATLAB function scatter(X,Y) (or the equivalent in the language you are using), make six plots, one for each pair of variables (i.e., sepal length vs. sepal width, sepal width vs. petal width, etc.). Are the classes linearly separable in each of these feature subspaces? Why or why not? (*3 points*)

   (c) Implement a perceptron and the perceptron learning rule, and train it to classify the training data *iris_train.data*, using all four features. Make a plot of the error rate (percent of examples that are misclassified) as a function of training epochs (one *epoch* is a complete pass through the data). You are free to choose the stopping criterion when necessary (i.e., a patience parameter or some test that the error is no longer going down), since the data might not be linearly separable. In your report, be sure to document how you did this, what your learning rate was, and how you initialized the weights and bias. Your well-documented code should be submitted for your team on Vocareum. (*4 points*)

   (d) With your trained perceptron, classify the test data in the file *iris_test.data* and report the percent correct (this is simply the number of correctly classified test data points divided by the total number of data points). (*2 points*)

   (e) What happens if you raise/lower your learning rate (e.g., 0.25, 0.5, 1, 2)? Visualize the effects of varying this parameter by making plots as in part c above, and report the test performance in a table. Explain your results. (*4 points*)

2. **Logistic and Softmax Regression (25 points)**
In this problem, we will train systems to categorize handwritten digits into 10 classes, using data from the MNIST dataset, available on Yann LeCun's website: MNIST Database. Please download the four files found there, these will be used for this problem. To reduce computation time, we will create a subset of these files. Extract the first 20,000 images and labels from the training sets, and the first 2,000 images and labels from the test sets. So for training now, $N = 20,000$, and for testing, $N = 2,000$.

   Read and process the data. Read in the data from the files. Each image is 28×28 pixels, so now the dimensionality of the input is $d = 784$. Turn each image into a vector, so that $x \in \mathbb{R}^{784}$, where each vector component represents the greyscale intensity of that pixel. For such large datasets with high-dimensional data, it is typical *not* to z-score the 784 variables - we don't compute the average and variance over 20,000 examples of each one! For each image, append a '1' to the beginning of each x-vector; this is the input from a unit that is always 1 and will be the constant x_0 that is multiplied times the (learned) bias, w_0.

   (a) **Logistic Regression via Gradient Descent.** *(10 points)*
   Now, using the gradient derived for Logistic Regression cross entropy loss, apply batch gradient descent to classify each $x \in \mathcal{R}^{785}$. We can still think of this as a "1 hot" encoding, i.e., the target vector will

Algorithm 1 Two Approaches to Gradient Descent

procedure BATCH GRADIENT DESCENT
  $w \leftarrow 0$
  for t = 0, . . . , m **do**
    $w_{t+1} = w_t - \alpha \sum_{i=1}^{n} \nabla E(w)$
  return w

procedure STOCHASTIC GRADIENT DESCENT
  $w \leftarrow 0$
  for t = 0, . . . , m **do**
    randomize the order of the indices into the training set
    for n=1, . . . , N **do**
      $w_{t+1} = w_t - \alpha \nabla E^n(w)$
  return w

be, for example, $t^n = (0, 1, ..., 0)$ for an image of a "1", and $t^n = (0, 0, 1, ..., 0)$ for an image of a "2", etc. That is, the *network* is the same between this problem and the version using softmax, we just use a different activation function, i.e.,the logistic, instead of the softmax.

i. Plot the error over training epochs. If the error blows up, your learning rate is too high.

ii. Report the test accuracy. In a case like this, we naturally find the maximum output, say y_i^n for input pattern x^n and choose the i^th category as the system's choice.

(b) **Softmax Regression via Gradient Descent.** *(10 points)* Now, train the same network using the gradient derived for Softmax Regression. The only difference between these two networks is the activation function.

i. Plot the training accuracy on the training set vs. number of iterations of gradient descent. Is there a difference in convergence rate between logistic and softmax regression?

ii. Report the test accuracy on the test set. Is there a difference from the logistic network?

(c) **Batch versus stochastic gradient descent. (5pts)**

i. Using your softmax network, implement the stochastic gradient descent algorithm above. Note you now have another `for` loop inside the "epoch" `for` loop. To randomize the patterns, use an indirect indexing method, i.e., have an integer array of N indices, and permute the order of that array. Let's call this array P. So initially, $P[i] = i$. Then on each epoch, you simply permute the integers in P. Let's call the $N \times d$ matrix of input patterns $Input$. For each epoch, as i goes from 1 to N on the innermost loop, we train on the 784 dimensional input $Input[P[i]]$,instead of $Input[i]$.

ii. Plot the error over training epochs. Is stochastic gradient descent faster, in terms of minimizing the error in fewer epochs? Explain your result.

3. **Individual Contributions to the Project**
Don't forget to include a paragraph at the end of your report, one per team member, describing what that team member's contribution to the project was.