Programming Assignment 1

CSE 253: Deep Learning

Winter 2019

Instructions

Due Thursday, January 17^{th}:

1. Please hand in your assignment via gradescope. This has not been set up yet for this class, more detailed submission instructions will be posted later.

2. There are two components to this assignment: mathematical solutions/proofs with English explanations. These must be done individually, but also (as the next part) typeset using LaTeX or Word. The individual part does not need to be in NIPS format, but it must be typeset! **We will not be accepting any handwritten work.** This part will be submitted separately from the group report on gradescope.

3. For the programming assignment portion of the homework (Part II: 1, 2, and 3), we want your report written using LaTeX in NIPS format. All parts of the assignments must be typeset, including figures. You may also use Word if you so choose, and figures may be created with Excel or Python, so long as they are computer generated.

   Include an informative title (not "Programming Assignment 1"!), author list, and an abstract. The abstract should summarize briefly what you did, and the best percent correct you got on each problem. The report should have clear sections for the three programming parts, **as well as a fourth section where each team member says what their contribution to the project was.** The report should be well-organized with figures near where they are referenced, informative captions on figures, etc.

4. You are expected to use Python (usually, with NumPy). 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.

5. Using any off-the-shelf code is strictly prohibited.

6. For the "Problems to be solved individually" part, please do your own work. You can discuss the assignment with other classmates, as long as you follow the Gilligan's Island Rule (see below).

7. For the programming part (Logistic and Softmax regression), please work in pairs. In *extraordinary circumstances* (e.g., you have a highly contagious disease and are afraid of infecting your teammate), we will allow you to do it on your own. Please discuss your circumstances with your TA, who will then present your case to me.

8. 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

Problems to be solved and turned in individually

For this part we will *not* be accepting handwritten reports. Please use latex or word for your report. MathType is a handy tool for equations in Word. The free version (MathType Lite) has everything you need. This should be done individually, and each team member should turn in his or her own work separately.

1. **Problems from Bishop (20 points)** Work problems 1-4 (5 points each) on pages 28-30 of Bishop. *Note: In Equation 1.51, the argument of exp should be* $(-\epsilon^2/\sigma^2)$.

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

   Although we consider logistic regression to be a binary (two-category) classification technique, it is called "regression" because it is used to fit a continuous variable: the probability of the category, given the data. 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).

   Consider the following model parameterized by w:

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

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

   where we assume that x has been augmented by a leading 1 to represent the bias input. With the model so defined, we now define the Cross-Entropy cost function, equation 5, the quantity we want to minimize over our training examples:

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

   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, prove that this gradient is:

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

Part II
Programming Assignment

Requirements: Write your own Logistic Regression and Softmax Regression classifiers using Python, following the instructions below.
You **are allowed** to use the following Python libraries and packages: NumPy, PIL, matplotlib, os, and random.
You **are not allowed** to use any SciPy implementations or logistic or softmax regression or any other high-level machine learning packages (including, but not limited to TensorFlow, Py/Torch, Keras, etc.).

For this assignment, a "team" is defined as two people.

Background

Logistic Regression

See the explanation of Logistic Regression above. Recall the loss function for Logistic Regression is:

$$E(w) = -\sum_{n=1}^{N} \{t^n \ln y^n + (1 - t^n) \ln(1 - y^n)\}. \tag{5}$$

Here, t^n is the *target* or *teaching signal* for example n. One issue with this cost function is that it depends on the number of training examples. For reporting purposes in this assignment, a more convenient measure is the average error, so please use this when reporting results:

$$E(w) = -\frac{1}{N} \sum_{n=1}^{N} \{t^n \ln y^n + (1 - t^n) \ln(1 - y^n)\}. \tag{6}$$

Softmax Regression

Softmax regression is the generalization of logistic regression for multiple (c) classes. Now given an input x^n, softmax regression will output a vector y^n, where each element, y_k^n represents the probability that x^n is in class k.

$$y_k^n = \frac{exp(a_k^n)}{\sum_{k'} exp(a_{k'}^n)} \tag{7}$$

$$a_k^n = w_k^T x^n \tag{8}$$

Here, a_k^n is called the *net input* to output unit y_k. Equation 7 is called the *softmax activation function*, and it is a generalization of the logistic activation function. For softmax regression, we use a *one hot encoding* of the targets. That is, the targets are a c-dimensional vector, where the k^{th} element for example n (written t_k^n) is 1 if the input is from category k, and 0 otherwise. Note each output has its own weight vector w_k. With our model defined, we now define the *cross-entropy* cost function for multiple categories in Equation 9:

$$E = -\sum_{n} \sum_{k=1}^{c} t_k^n \ln y_k^n \tag{9}$$

Again, taking the average of this over the number of training examples normalizes this error over different training set sizes. Also averaging over the number of categories c makes it independent of the number of categories. Please take the average over both when reporting results. Surprisingly, it turns out that the learning rule for softmax regression is basically the same as the one for logistic regression! The gradient is:

$$-\frac{\partial E^n(w)}{\partial w_{jk}} = (t_k^n - y_k^n) x_j^n \tag{10}$$

where w_{jk} is the weight from the j^{th} input to the k^{th} output.

Assignment Instructions:

1. **Load and preprocess the data (5 points).**
   In this problem, we will use logistic regression to separate faces of different emotions (for example, discern happy faces from sad faces). We will use some of the faces from the California Facial Expressions (CAFE) dataset.
   For your convenience, we have provided a tarball of this dataset on the Piazza page under "Resources", as well as a basic data-loader to read in all the images in this directory: the images are stored as NumPy arrays in the list *images*, and their corresponding filenames (labels) are stored in *labels*. There is also a basic function to display an image (See the **dataloader.py** file for instructions to uncompress the tarball and install the dependencies NumPy and PIL if needed).

   The file name is in the format <subject number>_<[h,ht,m,s,f,a,d,n]<digit>.pgm The semantics are: h:happy, ht:happy with teeth, m:maudlin (sad), s:surprise, f:fear, a:anger, d:disgust, n:neutral. **Note:** You don't need the neutral faces, and you should just pick *one* of the happy faces for each subject - otherwise your training set will not be balanced between each emotion.

   (a) For each problem, apply PCA to the *training set* images. With the array of images, select the ones for the training set, and apply PCA to reduce the dimensionality. It is important to do this efficiently, as there are two methods, and using the SVD method is way too expensive. To find out how to do this efficiently, see this page. Do not use the method at the top of the page: scroll down to the one that starts "I posted my answer even though another answer has already been accepted;". Again, apply this only to the training set in each case. Then, for each principal component vector, scale it by the standard deviation (the square root of the eigenvalue for each component). This should result in data projections where each variable is 0 mean and unit standard deviation.

   The maximum number of principal components you can extract is the minimum of: 1) the dimensionality of the images and 2) the number of images used to compute them minus 1. Hence in our application, it will be the latter number. You then can use only the top k principal components to project the data onto, and then your regression will be on k-dimensional inputs. Try keeping three different numbers of components (e.g., when training on all emotions, try 10, 20, and 40; when training on only two emotions, try a much smaller number), and report your results.

   Note: In case the above instructions are too hard to follow (or the person's code I pointed you to doesn't work!), I will have the TAs do this shortly so that they can advise you on how to do it correctly.

   Once this is done, this transformed data is what you will use to train the logistic regression. For the test data, you project the images onto the principal components of the training data before testing.
   Question: Why is it important to only use the training data to compute the principal components? Explain (2 pts).

   (b) Make a figure showing six different emotions from this set of images (6 pictures) for the beginning of your report (1 pts).

   (c) Make a figure showing the first 6 eigenvectors, each plotted as an image. They should resemble ghostly-looking faces, if you have done the PCA correctly (1 pts).

2. **Logistic Regression (25 points)**
   Here, we build and evaluate classifiers on the data. We will experiment with discerning between two classes of data, and with multi-class classification.

   (a) **Implement Logistic Regression via Gradient Descent.** *(10 points)*
   Now, without using any high-level machine learning libraries, implement logistic regression. Here, you'll be using batch gradient descent, and will only need one logistic output unit. (Think about why we only need one if we're classifying two classes?) The most points will be given for clean, well-documented code.

   (b) **Evaluate on Happy vs Maudlin.** *(10 points)*
   We will use a split of 80% train, 10% holdout, and 10% test. This should be on a per-subject basis, so one subject's faces will be the holdout, one will be the test. The holdout set is not used for changing the weights - it is a stand-in for the test set. We use it to see how well the network generalizes and stop training when the error on it goes up. Since errors can oscillate, train for some maximum number of epochs, **for this problem, with so little data, let's use 10**, and keep the best weights.

Algorithm 1 Two Approaches to Gradient Descent

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

procedure STOCHASTIC GRADIENT DESCENT
    $w \leftarrow 0$
    for t = 0, . . . , T' **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

Repeat this 10 times: Use each subject as the test set once, and for each test subject, pick a random subject to be the holdout (don't use the same one over and over), and when reporting error or accuracy (see below) report the average over the 10 subjects. **Don't forget to encode your labels:** let's have $t = 1$ mean "Happy" and $t = 0$ be "Sad". Hence, $y > 0.5$ means the network thinks the input is "Happy" and "Sad" otherwise. *(5 points)*

i. Plot the error (cross-entropy loss, Equation 5) over the 10 training epochs (an "epoch" is one pass through all of the training data) averaged over the ten runs for the training set, along with the holdout loss. Include the standard deviation over the ten runs in your plots for 2, 4, 8, and 10 epochs. If the error blows up, your learning rate is too high.
Use early stopping based on the holdout set error. Since the error can go up and down, this is a little tricky: So, save the weights any time the holdout set error is less than it has been before. If it goes back up again, you still have the best weights.
Make sure that your graph is well-labeled (i.e., x and y axes labeled with what they are) with a key, and with a figure caption that clearly states what is being shown in the graph. This should be the case for any graph you plot.

ii. Report the test accuracy after training. Here, by "accuracy", we mean the percent correct when choosing the category according to the rule above (> 0.5 = "Happy"). Again, this should be an average over the 10 runs with the standard deviation in parentheses (e.g., like this: 94.4% (1.2)).

iii. Repeat the above for 3 different learning rates (i.e., 3 total). Try and find one that's too high, one that's too small, and one that's just right. Plot the training set error (cross-entropy loss) for the three different learning rates on one plot, including the standard deviation as above. For your report above, use the best learning rate you found.

(c) **Evaluate on Afraid vs Surprised.** *(5 points)*

i. Plot the cross-entropy loss over training epochs averaged over the ten runs. Use the best learning rate you found above.

ii. Report the test accuracy averaged over the 10 runs.

iii. How does this differ from what we observed above? Why do you think that is?

3. **Implement Softmax Regression via Gradient Descent.** *(20 points)*
Now, we'll modify our network to classify all six emotion categories. To achieve multi-class classification, we'll need more output units, and use the gradient derived for Softmax Regression. The primary mathematical difference in this section is the activation function, which is the one in Equation 7. Also, you will be using one-hot encoding of the targets here. When choosing the category for evaluating percent correct, you just choose the maximum output (as in Lecture 1).

(a) **Evaluate your network on all six emotions.** *(10 points)*
Again, here, just choose one of the happy faces for each person so that there is the same amount of data for each emotion category. What do you think would happen if we used all of the happy faces? (You can try it if you like). Follow the same instructions as for Happy vs. Sad - repeat this ten times using a different subject for the test set each time, use early stopping with a hold-out set, etc. **for this**

problem, we anticipate it will take a little longer to train, so use 50 epochs, plotting the standard deviation at 10, 20, 30, 40, and 50 epochs.

i. Plot the loss (Equation 9) on the training set as well as the loss on the holdout set vs. number of iterations of gradient descent . 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. Again, plot the average over 10 runs.

ii. Create a table that is a 6×6 confusion matrix for this data based on the test set results. The emotions should be listed along the top and left side of the matrix. The confusion matrix will have 36 entries, C_{ij}, where i is the correct emotion and j is the emotion chosen by the network, and C_{ij} is the percent time that j was chosen for i. Hence the rows should add to 100%. A perfect system will have 100% down the diagonal and 0's everywhere else. Your system will most likely not be perfect!

(b) **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 input $Input[P[i]]$,instead of $Input[i]$.

ii. Plot the training set loss over training epochs, with one curve for the batch mode above, and one for stochastic gradient descent. Is stochastic gradient descent faster, in terms of minimizing the error in fewer epochs? Explain your result.

(c) **Visualize the weights (5 points).**
We can visualize the weights by taking the weights and multiplying them times the eigenvector they are from, and then adding together all of those images (i.e., the eigenvectors treated as images) together. Using one of your trained networks, transform the learned weights for each emotion into an image in this fashion. You will need to rescale the resulting pixels into the range 0-255 for this purpose. That is, simply linearly scale them so the minimum pixel value for each emotion is 0 and the maximum is 255. (This will be different for each emotion). Then, visualize these as an image. You should see a ghostly-looking image of happiness, etc. Explain why you end up seeing this. Include the six images in your report as a well-labeled figure.

(d) **Extra Credit (5 points).**
Use the softmax to classify the faces into the 10 identities, and visualize the weights as above.

4. **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.