CSC2515 Winter 2015
Assignment 2
Due: March 9, 2015, at START of class

February 18, 2015

There are several questions on this assignment; only the last question involves coding. Please turn in your write-up at the beginning of class on March 9th (or submit it electronically to CDF), but do not attach your code. For your implementation, submit it electronically to CDF, with all files compressed in *A2.zip* or *A2.tar.gz*. Please also specify the assignment name *A2*.

Late assignments will have 25% subtracted from the total out of which they are graded for each day or part of a day that they are late. They will be more than one day late if they are not submitted within 24 hours of the due date/time.

1 Learning a Mixture of Multinomials (15 points)

In this question you will derive gradient updates to perform learning in a *mixture-of-multinomial distributions models*.

Imagine that variable x is D-dimensional vector distributed according to a mixture-of-K-discrete-multinomials:

$$p(\mathbf{x}|\boldsymbol{\pi}, \boldsymbol{\mu}) = \sum_{k=1}^{K} \pi_k p(\mathbf{x}|\boldsymbol{\mu}_k)$$

where

$$p(\mathbf{x}|\boldsymbol{\mu}_k) = \prod_{i=1}^{D} \boldsymbol{\mu}_{ki}^{x_i}$$

is a discrete multinomial with parameter $\boldsymbol{\mu}_k$ and $\boldsymbol{\pi}$ represents the mixture proportions.

Imagine that your task is to maximize the likelihood of the data given your model parameters using gradient descent. We ask you to derive the updates for both the mixing proportions, $\boldsymbol{\pi}$ as well as the parameters of the multinomials $\boldsymbol{\mu}$. In both cases you must be careful that your updated variables will still meet the constraints:

- $\pi_k \geq 0, \ \forall k$ must be positive and $\sum_k \pi_k = 1$
- $\mu_{ki} \geq 0, \ \forall k, \forall i$, and $\sum_i \mu_{ki} = 1$.

That is, gradient descent typically performs unconstrained optimization. However in this case the variables are constrained and you must re-parametrize the variables such that their resulting updates will be sensible to the constraints.

2 Neural networks for speech recognition

Neural networks are now an integral part of most modern speech recognizers (for example Google's Voice Search System uses a Neural Network for Acoustic Modelling). For this question we have created a rudimentary speech recognition system that uses neural networks that predict speech sound units (called *phonemes*) for transcribing audio signals. Your goal is to train the neural networks.

The training data includes frames of audio from 400 utterances. We have pre-processed these utterances so that audio recordings have been converted to a sequence of vectors of 23 dimensions, representing Mel log filter bank coefficients [1]. Each frame of the utterance has been associated with a target representing a phoneme label. The targets are 1 of 44 different label classes (see file *phones.txt* for the list of these 44 phones and *phone_code.txt* for examples of words containing these phones). The validation data similarly include 200 utterances and corresponding data and targets. The test data does not provide targets, but the transcriptions are provided, for the purpose of scoring the speech recognition results.

For this assignment we will be implementing forward propagation, back propagation and stochastic gradient descent with momentum. We have provided template code, that we expect you to extend for this assignment. The changes to this code will be graded, along with the responses to the questions below. Once you have made the required changes, you can train neural networks on the data we have provided.

[1] You do not need to know anything about these coefficients for the purpose of this assignment. However, for completeness sake - Mel Log filter banks are a low resolution spectral representation motivated by the human perceptual apparatus, that has been found very useful for speech recognition

Note about Notation: We will refer to learning rate as η and momentum as m in this question. A n-hidden layer neural network we refer to a neural network with n layers of hidden units. Thus a softmax function is a 0-layer network, and a 1-hidden layer network actually has two sets of weights. In speech technology, the validation set is generally referred to as the development or 'dev' set.

Here are the details of the code. We have provided both matlab and python stubs (3 and 4 below, respectively). You only need to use one of them.

The assignment archive contains the following folders:

1. data. This folder contains files that have the data in them. The training, validation and test files are *train.mat*, *dev.mat* and *test.mat* respectively. The transcriptions for the test utterances are in file *test_trans.txt*. The file *transition_counts.txt* tabulates the number of times phoneme j follows phoneme i in the entire database, in entry $\{i, j\}$. The file *reduced.mat* is a reduced set of data to test your code and gradients, etc, before running the actual experiments below on the real data.

2. decoder. This folder contains the two programs that are needed to score the accuracy of the speech recognizer you have trained. The first of these is the speech recognizer which uses your neural network predictions. This is in *decoder.py*. The second of these is the program that computes the accuracy of the transcriptions, by comparing them to the known transcriptions. This is in the file *compute_per.py*. They are both python programs - but running them does not require python knowledge. Even if you implemented your neural network in matlab, you should still be able to use this.

3. code_python. This folder contains the python stubs needed for this assignment. *nn_train.py* contains the neural network implementation. It uses the implementation for neural network layers that is provided in *nn_layers.py*. These files need to be changed to provide a neural network implementation. The file *train_nnet.py* contains wrapper code that loads the data and calls the neural network implemented in *nn_train.py*. It also needs to be modified, to implement stopping criterion, etc. The final file it contains is *create_predictions.py*. This needs to be run on the test data using the model you learned. This creates the predictions you need for decoding (see below). You need to change the file so that the preprocessing you apply in training the neural network is the same as the one you apply during predictions.

4. code_matlab. This folder contains the matlab stubs needed for this assignment. Because of the way matlab 7.1 (installed on cdf) handles classes we had to separate out each of the classes into a different folder. *nn* is the folder containing the nn class (in *nn.m*) and its member functions (the other files in the same folder). Similarly, *@layer* folder contains the implementation of class layer and its functions. Analogous to the python code, the neural net class *nn* (in folder *@nn*) relies on the *layer* class (in

folder *@layer*). The functions in these folders need to be changed to provide a neural network implementation. The file *train_nnet.m* in the main folder contains wrapper code that loads the data and calls the neural network implemented in *@nn*. It also needs to be modified to implement stopping criterion, etc. The file *create_preds.m* needs to be run on the test data using the model you learned. This creates the predictions you need for decoding (see below). You need to change the file so that the preprocessing you apply in training the neural network is the same as the one you apply during predictions.

Here are the details of the assignment.

Please complete the code for forward and backward propagation in *nnet_layers.py* and *nnet_train.py* (find the aptly named functions for these). Implement stochastic gradient descent in function called *apply_gradients*. Provide snippets of changed code with your main document.

Please modify *train_nnet.py* so as to implement a learning schedule with a reasonable learning rate and other parameters. Also remember to control for overfitting (see part 2 below).Hand in code for this file in the main document.

Also, please hand in your answers to the following questions.

1. **Pre-processing Inputs.** Correct pre-processing of your data can often make the difference between a neural network that trains fast and well and one that gets nowhere. How would you preprocess the provided data in such a way that it results in much faster learning ? (Hint: You can use procedures we have already implemented in the data interface). Train on the training data with a Neural Network with 1 hidden layer with 100 units (using $\eta = 0.03$ and $m = 0.9$), for three epochs with and without preprocessing and report the classification error (referred to as Frame Error Rate). Experiment with η and m to see if you can find values that lead to better learning in 3 epochs. Use this pre-processing for the rest of the question.

2. **Controlling Overfitting.** Train a 1-layer neural network with 100 hidden units for 30 epochs using learning parameters from previous question. Plot the training and validation set classification error as a function of the epoch. Explain what you see in the curve. Based on what you see in the curve, suggest a strategy for selecting the best neural network parameters over a run. Implement this strategy in file *train_nnet.py*, and provide a hard copy of the implementation. What classification error do you get on the development set ?

3. **# of units.** Using the procedure developed in the last question, and the same learning rate parameters, train a 1-layer neural network with 100, 300 and 500 units. What trend do you observe on the development set result for each of these settings, using the final, selected parameters after training ?

4. **Assessing impact of depth.** Using 300 hidden units per layer, train a 2-layer network. How do these numbers compare to the results from a 1-layer network with 300 units, above ? Why do you think this is the case ?

5. **Decoding Results.** The neural network models we have trained so far only predict the class of phone sounds from acoustic frames. However, in a speech recognition system, the frame predictions are only the first step in transcription. These predictions have to be converted into a reasonable sequence of words by the recognizer. In this part, we will use predictions of phone categories over an utterance, from a neural network, to do speech recognition over the test data. The transcription from the recognizer will be compared to the known transcription. Typically transcriptions are compared on a word level, and a Word Error Rate (WER) is reported. For this question we are looking at a reduced problem, and instead of comparing words, we are comparing phoneme sequences. This helps us avoid the issue of using pronounciation dictionaries that map words to phoneme sounds. The error is thus called the Phone Error Rate (PER).

   First run *create_predictions.py* to create a prediction file over test data (in *test.mat*). This file outputs the log of the probabilities of the different phoneme classes, for each acoustic frame of an utterance. Run, e.g. :

   python create_predictions.py data/test.mat run_name/model.mat predictions.mat

   The predictions above are independent predictions over the sequence of frames for each utterance. As such, there is no smoothing in the predictions. Speech recognizers use pronounciation dictionaries - which map words to sequence of phone sounds - and language models - which score sequence of words - to impose hard and soft temporal constraints on these predictions. With these additional constraints, a best path through state space of phonemes over the utterance, results in a reasonably good transcription.

   For this question, we have built a very simple language model, which tabulates the probabilities of transitions between one sound to the next from one frame to its neighbor. This is in the file: *transition_counts.txt*. Let's take the predictions above and find the best sequence of phonemes that these predictions represent, when combined with our 'language' model (a process known as decoding):

   python decode.py data/phones.txt data/transition_counts.txt predictions.mat pred_trans.txt

   Take a look at *pred_trans.txt*. It should have one line per test set utterance. This is the result of the speech recognition. To make sense of the transcription, you can look at the phone codes in file: *phone_code.txt*. The actual transcriptions are in the file: *test_trans.txt*. We are now going to compare these transcriptions and get the Phone Error Rate (PER) using the following command:

   python compute_per.py pred_trans.txt test_trans.txt

   Please report the PER's you achieved for each of the experiments above.

3 k-Nearest Neighbors

Note: *For this question use the file: knn_subset.mat.*

The k-nearest neighbors (KNN) algorithm is often highly competitive as a classifier despite being conceptually perhaps one of the simplest in machine learning. In this question, we ask you to experiment with KNN on a subset of the data above, and then, in writing, extend KNN to be truly non-parametric.

First, implement the KNN algorithm using an appropriate distance metric and use it to classify the test set of the filterbanks data. Run KNN for $k \in \{1, 3, 5\}$. The Matlab function 'bsxfun' will help to significantly speed up any loops over data examples (but its not available on the matlab versions on cdf).

Label noise is a prevalent problem in machine learning. Assume that ten percent of your training set labels are noise. For example, assume someone sabotaged your data by running the following Matlab code:

```
inds = randperm(4400);
train_targets(inds(1:440)) = randi(44,440,1);
```

In python you can achieve the same by running:

```
indices_to_corrupt = permutation(4400)[:440]
targets[indices_to_corrupt,0] = random.multinomial(1, [1/44.]*44,
                  size=indices_to_corrupt.size).argmax(axis=1)
```

Repeat the KNN classification for $k \in \{1, 3, 5\}$ after corrupting 10% of the labels. Include the following in your report:

1. Report the test set error (percent of examples that were misclassified) for $k \in \{1, 3, 5\}$.
2. Report the test set error for $k \in \{1, 3, 5\}$ after corrupting 10% of the labels.
3. Explain, in one paragraph or less, the discrepancy between the results you observe before and after adding label noise. How does the choice of k affect the different experiments?
4. The choice of k is somewhat arbitrary, but important to the KNN algorithm. One possible solution to this problem is to take into account the distances of the neighbors to the query point in their contribution to the classifier. Describe how you could modify k-nearest neighbors to do this, both in words and equations. Can you use this approach to remove the dependence on the parameter k? Describe situations in which you expect this method to do well and not do well.

4 Principal Components Analysis (20 points)

Principal components analysis (PCA) is one of the most widely used of the large class of continuous valued latent variable models. In this question, we ask you to examine the use of PCA to perform dimensionality reduction on the dataset from before. Then you will use PCA as a pre-processing step to your k-nearest neighbors algorithm from that assignment.

1. PCA involves learning a linear projection from a reduced dimensionality space to the data such that squared reconstruction error is minimized. Write down the probabilistic interpretation of this loss function and explain how maximum likelihood in this model corresponds to minimizing squared error.

2. Explain *briefly* what the optimal weights for PCA correspond to and how one can estimate this global optimum. Two sentences should be enough.

3. Now compute the optimal weights for PCA on the data from before. You may use the Matlab functions *eig*, *svd* and *cov* or Python functions *numpy.linalg.eig*, *numpy.linalg.svd* and *numpy.cov* A nice property of the transformation learned by PCA is that it is orthogonal. This allows us to project the data into the latent space by simply using the transpose of the weights. Project the training and test data into the first 10 principal components of the training data[2]. Now perform the k-nearest neighbors algorithm for $k = 1$ in this lower dimensional space and report the test classification error. Repeat this process for 5, 10 and 20 principal components and report the test error for each.

4. Explain in a paragraph or less the results you observed and compare this to your results from previous questions. Is PCA helping? Why yes/no? When do you expect it to help?

[2]To get a better intuition for what the PCA projection will looks like we also suggest that you look at the projection of your data in 2 dimensions.