EEE 443/543: Neural Networks

Assignment 2

(Due 16 November 2020, 17:00PM)

Instructions:

1. Prepare a report (including your answers/plots) to be uploaded on Moodle.
2. The report should be typeset (no handwriting allowed except for lengthy derivations, which may be scanned and embedded into the report).
3. Show all steps of your work clearly.
4. Unclear presentation of results will be penalized heavily.
5. No partial credits for unjustified answers.
6. **Use of any toolbox or library for neural networks is prohibited**.
7. Return all Matlab/Python code that you wrote in a single `.m/.py` file.
8. Code should be commented, code for different HW questions should be clearly separated.
9. The code file should NOT return an error during runtime.
10. If the code returns an error at any point, the remaining part of your code will not be evaluated (i.e., 0 points).

Question	Points	Your Score
Q1	30	
Q2	40	
Q3	30	
TOTAL	100	

Question 1. [30 points]

In this question we will consdier a cat versus car detection problem on visual images. The data file `assign2_data1.h5` contains the variables `trainims` (training images) and `testims` (testing images) along with the ground truth labels `trainlbls` and `testlbls`. You will implement stochastic gradient descent on mini-batches.

Here, you will measure the error term as a function of epoch number, where an epoch is a single pass through all images in the training set. Two different error metrics will be calculated: the mean squared error and the mean classification error (percentage of correctly classfied images). Record the error metrics for each epoch separately for the training samples and the testing samples. Answer the questions below.

a) Using the backpropagation algorithm, design a multi-layer neural network with a single hidden layer. Assume a hyperbolic tangent activation function for all neurons. Experiment with different number of neurons N in the hidden layer, initialization of weight and bias terms, and mini-batch sample sizes. Assuming a learning rate of $\eta \in [0.1\ 0.5]$, select a particular set of parameters that work well. Using the selected parameters, run backpropagation until convergence. Plot the learning curves as a function of epoch number for training squared error, testing squared error, training classification error, and testing classsification error.

b) Describe how the squared-error and classification error metrics evolve over epochs for the training versus the testing sets? Is squared error an adequate predictor of classification error?

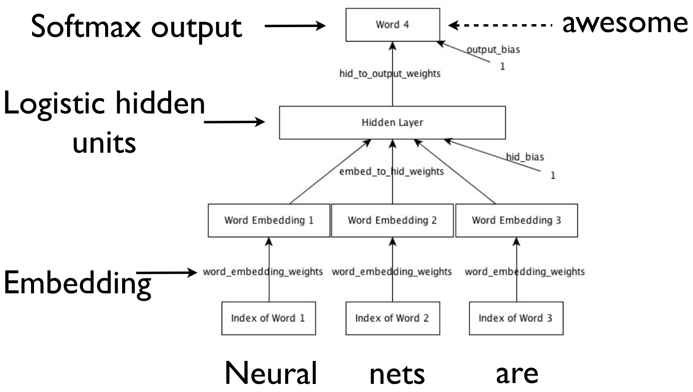
c) Train separate neural networks using substantially smaller and larger number of hidden-layer neurons (N_{low} and N_{high}). Plot the learning curves for all error metrics, overlaying the results for N_{low}, N_{high} and N^* prescribed in **part a**.

d) Design and train a separate network with two hidden layers. Assuming a learning rate of $\eta \in [0.1\ 0.5]$, select a particular set of parameters that work well. Plot the learning curves for all error metrics, and comparatively discuss the convergence behavior and classification performance of the two hidden-layer network with respect to the network in **part a**.

e) Assuming a momentum coefficient of $\alpha \in [0.1\ 0.5]$, retrain the neural network described in **part d**. Select a particular set of parameters that work well. Plot the learning curves for all error metrics, and comparatively discuss the convergence behavior with respect to **part d**.

Question 2. [40 points]

Neural network architectures can produce powerful computational models for natural language processing. Here, you will consider one particular model for examining sequences of words. The task is to predict the fourth word in sequence given the preceding trigram, e.g., trigram: 'Neural nets are', fourth word: 'awesome'. A database of articles were parsed to store sample fourgrams restricted to a vocabulary size of 250 words. The file `assign2_data2.h5` contains training samples for input and output (`trainx`, `traind`), for validation (`valx`, `vald`), and for testing (`testx`, `testd`). Using these samples, the following network should be trained via backpropagation:



The input layer has 3 neurons corresponding to the trigram entries. An embedding matrix R (250×D) is used to linearly map each single word onto a vector representation of length D. The same embedding matrix is used for each input word in the trigram, without considering the sequence order. The hidden layer uses a sigmoidal activation function on each of P hidden-layer neurons. The output layer predicts a separate response z_i for each of 250 vocabulary words, and the probability of each word is estimated via a soft-max operation $(o_i = \frac{e^{z_i}}{\sum_j e^{z_j}})$.

a) Assume the following parameters: a stochastic gradient descent algorithm, a mini-batch size of 200 samples, a learning rate of $\eta = 0.15$, a momentum rate of $\alpha = 0.85$, a maximum of 50 epochs, and weights and biases initialized as random Gaussian variables of std 0.01. If necessary, adjust these parameters to improve network performance. The algorithm should be stopped based on the cross-entropy error on the validation data. Experiment with different D and P values, (D,P) = (32,256), (16,128), (8,64) and discuss your results.

b) Pick some sample trigrams from the test data, and generate predictions for the fourth word via the trained neural network. Store the the predicted probability for each of the 250 words. For each of 5 sample trigrams, list the top 10 candidates for the fourth word. Are the network predictions sensible?

Question 3. [30 points]

The goal of this question is to introduce you fully-connected neural networks, and various optimization and regularization choices for these models. You will be experimenting with two Python demos, one on a fully-connected network model, and a second on dropout regularization on such a network. Download `demo_fnn.zip` from Moodle and unzip it. The demos are given as Jupyter Notebooks along with relevant code and data. The easiest way to install Jupyter with all Python and related dependencies is to install Anaconda. After that you should be able to run through demos in your browser easily. The point of these demos is that they take you through the training algorithms step by step, and you need to inspect the relevant snippets of code for each step to learn about implementation details.

a) The notebook `FullyConnectedNets.ipynb` contains demonstrations on a fully-connected network model. You need to run the demo till the end without any errors. You may have to debug the code in case of any fatal errors. You are supposed to convert the outputs of the completed demo to a PDF file, and attach it to the project report. You should also comment on your results and answer any inline questions that are provided in the notebooks.

b) The notebook `Dropout.ipynb` contains demonstrations of dropout regularization on a fully-connected network. You need to run the sdemo till the end without any errors. You are supposed to convert the outputs of the completed demo to a PDF file, and attach it to the project report. You should also comment on your results and answer any inline questions that are provided in the notebooks.