# Programming Assignment 2

February 21, 2017

Please do it as early as possible, it may take some time to finish all.

## 1 Introduction

In this assignment, we will familiarize you with basic concepts of neural networks, word vectors, and their application to sentiment analysis. To finish this assignment, you need to know how word-embedding works, and some basic statical knowledge for gradient computation.

## 2 Setup

The attached starter code can help you get started. You might want to create a virtual environment for this assignment, which is recommended. It can make you not deal of the version problems of the software.

```
To set up a virtual environment, run the following:

  cd assignment2
  sudo pip install virtualenv      # This may already be installed
  virtualenv .env                  # Create a virtual environment
  source .env/bin/activate         # Activate the virtual environment
  pip install -r requirements.txt  # Install dependencies
  # Work on the assignment for a while ...
  deactivate                       # Exit the virtual environment

To install requirements (without a virtual environment):

  cd assignment1
  pip install -r requirements.txt  # Install dependencies

Download data:
  cd big_data/datasets
  ./get_datasets.sh
```

# 3 Problem detail

## 3.1 Softmax

The softmax function is often used in the final layer of neural networks, which are applied to classification problems. Such networks are commonly trained under a log loss (or cross-entropy) regime, giving a non-linear variant of multinomial logistic regression.

$$softmax(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

In file q1_softmax.py, you need to implement the softmax function and pass the test in the file, you can also add some test cases by yourself.

## 3.2 Neural Network Basics

In this part, you need to implement three files, q2_sigmoid.py, q2_gradcheck.py and q2_neural.py.

1. Derive the gradient of sigmoid function

   $$sigmoid(x)_i = \frac{1}{1+e^{-x}}$$

   You need to implement the sigmoid function and its gradient, remember that the input x can be any dimension (a number or a 2D array). Think about that the input f should be the sigmoid function value of your original input x, which means implementation should be based on the result of $sigmoid(x)$.

2. Gradient Check

   Remember the definition of the gradient

   $$f'(x_0) = \lim_{\varepsilon \to 0} \frac{f(x_0+\varepsilon) - f(x_0)}{\varepsilon}$$

   This definition can help us check our implementation for a gradient function. We can compare the result from our gradient function and the result calculated by this formula, usually $\varepsilon$ is set as $1e^{-4}$, and the threshold as $1e^{-5}$. Remember that the input is an array instead of only one number!

3. Simple Neural Network

   This part you need to implement a 2-layer network with its forward and backward propagation. In the forward propagation, you need to compute the output of each layer and compute the cost function, in the backward propagation, you need to BP all the gradients, and return all the gradients of the parameters. The output layer should not have the sigmoid function. Remember the cross entropy cost function is:

   $$CE(y, \hat{y}) = -\sum_i y_i log(\hat{y}_i)$$

## 3.3 Word2Vec

In this part we will implement the word2vec models and train your own word vectors with stochastic gradient descent (SGD). And you need to know skip-gram and CBOW.

1. SoftmaxCostAndGradient

   Given a predicted word vector $\hat{r}$, and the word prediction is make with the softmax function found in word2vec models

   $$\hat{y}_i = Pr(w_i|\hat{r}, u_{w_{1\ldots|v|}}) = \frac{exp(u_{w_i}^T \hat{r})}{\sum_{j=1}^{|v|} exp(u_{w_j}^T \hat{r})}$$

   where $w_j$ denotes the j-th word and $u_{w_j}$ (j=1,...,—V—) are the "output" word vectors for all words in the vocabulary. Assume cross entropy cost is applied to this prediction and word i is the expected word (the i-th element of the one-hot label vector is one), derive the gradients with respect to the output word vectors $u_{w_j}$ (including $u_{w_i}$). This result will be implemented in q3_word2vec.py

2. Negative sampling loss

   Repeat the same process for the negative sampling loss for the predicted work, remember that this cost function has 4 parameters:

   $$J_{neg-sample}(w_i, \hat{r}, u_{w_i}, u_{w1\ldots K}) = -log(\sigma(u_{w_i}^T \hat{r})) - \sum_{k=1}^{K} log(\sigma(-u_{w_k}^T \hat{r}))$$

3. Skip-gram and CBOW

   In this part you need to code the skip-gram part, the cost for a context is

   $$J_{skip-gram}(word_{i-C\ldots i+C}) = \sum_{-C\leq j\leq C, j\neq 0} F(w_{i+j}, v_{w_i})$$

   The CBOW is optional, if you implement it correctly you can earn some extra credits.

4. SGD optimizer

   Complete the implementation for SGD in q3_sgd.py. Test your implementation by running this file.

5. Run the code!

   Now we are going to load some real data and train word vectors with everything you just implemented! We are going to use the Stanford Sentiment Treebank dataset to train word vectors, and later apply them to a simple sentiment analysis task. Just run q3_run,py

## 3.4 Sentiment Analysis

f Now, with the word vectors you trained, we are going to perform a simple sentiment analysis. For each sentence in the Stanford Sentiment Treebank dataset, we are going to use the of all the word vectors in that sentence as its feature, and try to predict the sentiment level of the said sentence. The sentiment level of the phrases are represented as real values in the original dataset, here well just use five classes:

very negative, negative, neutral, positive, very positive

which are represented by 0 to 4 in the code, respectively. For this part, you will learn to train a softmax regressor with SGD, and perform train/dev validation to improve generalization of your regressor.

1. Implement a sentence featurizer and softmax regression. Fill in the implementation in q4_softmaxreg.py. Check your implementation just run this file.
2. Fill in the hyperparameter selection code in q4_sentiment.py to search for the optimal regularization parameter.
3. Plot the classification accuracy on the train and dev set with respect to the regularization value, using a logarithmic scale on the x-axis.