CENG463 Assignment 3

Deadline: 15/06/2020

Neural Networks

In this assignment, you will implement a feedforward neural network for named entity recognition (NER) with a single named entity class PERSON. You will derive and implement the word embedding layer, the feedforward neural network and the corresponding backpropagation training algorithm.

1. Model Overview

   The neural network has 3 layers: an input layer, a hidden layer and an output layer. Recall that our inputs are the word vectors which are also model parameters that we will optimize. Our top layer will be a logistic regression classifier. The final cost function (also often called loss function) will be the binary cross-entropy error, which is similar to the general cross entropy but for only two classes.
   The idea of the model is that in order to classify each word, you take as input that word's vector representation and the vector representations of the words in its context. These vectors constitute your *features*. These features are the input to a neural network which is a function that will first transform them into a *hidden* vector and then use that vector to predict a probability of how likely the window's center word is of a certain class.

2. Word vectors and context windows

   Assume you have the following sequence of words in your training set

   George is happy about the election outcome.

   Each word is associated with a label y defining that word as either PERSON or not any named entity 0 . We define PERSON to be class 1 $(y = 1)$ and O to be class 0. Each word is also associated with an index into a vocabulary, so the word George might have index 444.

   All word vectors are n-dimensional and saved in a large matrix $L \in R^{nxV}$, where V is the size of the vocabulary. We will provide you with an initial set of word vectors that have been trained with an unsupervised method. The vectors are $n = 50$ dimensional.

   You will have to load these vectors into your Python program and save them into a matrix along with their indices and the words they represent.

Assume that at the first location of the sentence we have vector x_1, which was retrieved via the index of the word George. Similarly, we can represent the whole sentence as a list of vectors $(x_1, x_2, ..., x_8)$. When we want to classify the first and last word and include their context, we will need special beginning and end tokens, which we define as $< s >$ and $< /s >$ respectively. If we use a context size of C, we will have to pad each input of the training corpus with C many of these special tokens. For simplicity, let us use $C = 1$. We define the vector corresponding to the begin padding as x_s and for the end padding as x_e. Hence, we will get the following sequence of vectors:

$(x_s, x_1, x_2, ..., x_8, x_e)$.

For this training corpus, we have the following windows that we will give as input to the neural network:

$[x_s, x_1, x_2], [x_1, x_2, x_3], [x_2, x_3, x_4], ..., [x_6, x_7, x_8], [x_7, x_8, x_e])$

Each window is associated with the label of the center word. So for this sequence, we get the labels $(1, 0, 0, 0, 0, 0, 0, 0)$. Note that the model just sees each window as a separate training instance.

3. Definition of feedforward network function

   Each window's vectors are given as input to a single neural network layer (called a hidden layer) which has dimensionality H. This layer is used as features for a logistic regression classifier which will return the probability that the center word belongs to either of the two classes. For the first window, the feed-forward equations are as follows:

   $z = Wx + b^{(1)}$

   $a = f(z)$

   $h = g(U^T a + b^{(2)})$

   The final prediction h is x_1's probability of being a PERSON. Let us define all the involved notation: the model parameters W, U and the model functions f, g. We have the following neural network parameters: $W \in R^{HxCn}$. For our case with a window size of $C = 3$ and if we have a reasonable hidden layer size such as $H = 100$, we get $W^{100x150}$. The bias of the hidden layer is $b(1) \in R^{Hx1}$ and the bias of the logistic regression $b^{(2)}$ is a scalar. The parameters for the logistic regression weights then have to be $U \in R^{Hx1}$. Note that you could also add a single 1 to a a and use $U \in R^{(H+1)x1}$, in other words including the bias term $b^{(2)}$ inside U.

   The nonlinearity function f can be *tanh*.

   As the function g needs to return a probability so we will always use the sigmoid function:

   $g(z) = sigmoid(z) = \frac{1}{1+e^{-z}}$

Backpropagation Training

In this part, you will implement the backpropagation algorithm to compute the gradient for the neural network cost function. Once you have computed the gradient,

you will be able to train the neural network by minimizing the cost function $J(\theta)$ using a simple optimization technique called stochastic gradient descent (SGD).

Recall that the intuition behind the backpropagation algorithm is as follows. Given a training example $(x^{(i)}, y^{(i)})$, we will first run a forward pass to compute all the activations throughout the network, including the output value of the hypothesis $h_\theta(x)$. Once we have the prediction, we will compute the binary cross-entropy error. Then, for each node j in the hidden layer, we would like to compute an error term δ_j that measures how much that node was "responsible" for the cross-entropy error.