CE7455: Deep Learning for Natural LanguageProcessing

Assignment # 1

Submission Due: 21 Feb, 2020 (before 6:30 pm)
Submission: Send it to ce7455.assignments@gmail.com with subject "A1-YourStudentID"".

1 Question One [10 marks]

Linear regression and logistic regression are examples of **generalized linear models or GLMs**. The output layer of a neural network can be modeled as a GLM as shown in the lecture. It was also mentioned that for the GLMs, the gradients of the loss with respect to the parameters are

$$\nabla_{\mathbf{w}_k} \mathcal{L}(W) = (\hat{y}_k - y_k)\mathbf{z} \tag{1}$$

where $\mathbf{z}$ is the input vector to the GLM layer, W is the matrix containing the associated weight vectors, and $\hat{y}_k$ is the prediction at the k-th output unit (e.g., for k-th class).

In this exercise, you have to prove Eq. 1, when the GLM is a multi-class logistic regression (or MaxEnt) defined by the following equation:

$$p(y = k|\mathbf{z}, W) = \hat{y}_k = \frac{\exp(\mathbf{w}_k^T \mathbf{z})}{\sum_{k'=1}^{K} \exp(\mathbf{w}_{k'}^T \mathbf{z})} \tag{2}$$

where K is the total number of classes. Show your derivation.

2 Question Two [40 marks]

A language model can predict the probability of the next word in the sequence based on the words already observed in the sequence. Neural networks are a preferred method for developing such language models because they can use a large context of observed words and use a distributed representation (as opposed to symbolic representation) of words. In this exercise, you will learn how to develop a neural language model in Pytorch. In particular, you will learn

(a) How to prepare texts for developing a word-based language model.

(b) How to design and fit/train a neural language model.

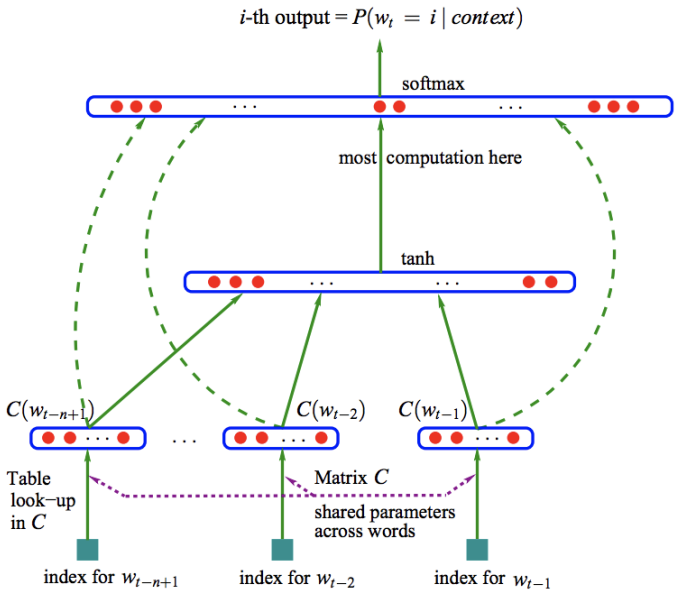

Figure 1: Neural architecture: $f(i, w_{t-1}, \cdots, w_{t-n+1}) = g(i, C(w_{t-1}), \cdots, C(w_{t-n+1}))$ where g is the neural network and $C(i)$ is the i-th word feature vector.

Figure 1: A Neural Probabilistic Language Model by [1]

(c) How to use the learned language model to generate new texts.

(d) How to evaluate word vectors.

The dataset for this exercise will be the wikitext-2 dataset. You can use this code as your codebase and build on top of it. Please do the following:

(i) Please download the dataset and the code. The dataset should have three files: *train*, *test*, and *valid*. The code should have basic prepossessing (see data.py) and data loader (see main.py) that you can use for your work. Try to run the code.

(ii) You should understand the preprocessing and data laoding functions.

(iii) Write a class *class FNNModel(nn.Module)* similar to *class RNNModel(nn.Module)*. The FNNModel class should implement a language model with a feed-forward network architecture. For your reference, RNNModel implements a recurrent network architecture, more specifically, Long Short-Term Memory (LSTM) that you will learn later in the course.

The FNN model should have an architecture as shown in Figure 1. This is indeed the first neural language model [1]. The neural model learns the distributed representation of each word (embedding look-up matrix C) and the probability function of a sequence as a function of the distributed representations. It has a hidden layer with tanh activation and the output layer is a Softmax layer. The output of the model for each input of $(n-1)$ previous word indices are the probabilities of the $|V|$ words in the vocabulary.

(iv) Train the model with any of SGD variants (Adam, RMSProp, Adagrad).

(v) Show the perplexity score on the test set. You should select your best model based on the perplexity score on the *valid* set.

(vi) Do steps (iv)-(v) again, but now with sharing the input (look-up matrix) and output layer embeddings (final layer weights).

(vii) Adapt generate.py so that you can generate texts using your language model (FNNModel).

(viii) In your opinion, which computation/operation is the most expensive one in inference or forward pass? Can you think of ways to improve this? If yes, please mention.

(ix) Notice that the model also learns word vectors (input and output layer embeddings) as a byproduct. One way to evaluate the trained word vectors is to measure the cosine similarity between pairs of words, and then report the correlation with the similarity scores given by humans. For this exercise, use the dataset available here and report the Spearman correlation for the input embeddings. Exclude any pair if it is not in the embedding matrix.

3 Question Three [0 marks]

To help me in future planning with the assignment exercises, please mention how much time you spent on each of the questions.

References

[1] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March 2003.