CSCE 636: Deep Learning (Fall 2020)
Assignment #4

Due 11:59PM on 11/24/2020

---

1. You need to submit (1) a report in PDF and (2) your code files, both to eCampus.
2. Your PDF report should include (1) answers to the non-programming part, and (2) results and analysis of the programming part. For the programming part, your PDF report should at least include the results you obtained, for example the accuracy, training curves, parameters, etc. You should also analyze your results as needed.
3. Please put all your files (PDF report and code files) into a compressed file named "HW#_FirstName_LastName.zip"
4. Unlimited number of submissions are allowed on eCampus and the latest one will be timed and graded.
5. Please read and follow submission instructions. No exception will be made to accommodate incorrectly submitted files/reports.
6. All students are highly encouraged to typeset their reports using Word or LaTeX. In case you decide to hand-write, please make sure your answers are clearly readable in scanned PDF.
7. Only write your code between the following lines. Do not modify other parts.

   ### YOUR CODE HERE

   ### END YOUR CODE

---

1. (100 points)(Coding Task) In this assignment, you will implement a recurrent neural network (RNN) for language modeling using Pytorch. The task is to predict word x_{t+1} given words $x_1, \ldots, x_t$:

$$P(x_{t+1} = v_j | x_t, \ldots, x_1)$$

   where v_j is the j-th word in the vocabulary. The file"utils.py" gives an example of how to generate the vocabulary. You can read it if interested. With the vocabulary, we can transform a word x_i into a one-hot vector.

   Our RNN model is, for $t = 1, \ldots, n-1$:

$$\begin{aligned} e^{(t)} &= x^{(t)} L, \\ h^{(t)} &= \text{sigmoid}(h^{(t-1)} H + e^{(t)} I + b_1), \\ \hat{y}^{(t)} &= \text{softmax}(h^{(t)} U + b_2), \\ \bar{P}(x_{t+1} &= v_j | x_t, \ldots, x_1) = \hat{y}_j^{(t)}. \end{aligned}$$

   where the first line actually corresponds to a word embedding lookup operation. $h^{(0)}$ is the initial hidden state, $\hat{y}^{(t)} \in \mathbb{R}^{|V|}$ and its j-th entry is $\hat{y}_j^{(t)}$.

   Training parameters θ in this model are:

- L – embedding matrix which transforms words in the vocabulary into lower dimensional word embedding.
- H – hidden transformation matrix.
- I – input transformation matrix which takes word embedding as input.
- U – output transformation matrix which projects hidden state into prediction vector.
- b_1 – bias for recurrent layer.
- b_2 – bias for projection layer.

(a) (10 points) Let the dimension of word embedding as d, the size of vocabulary as $|V|$, the number of hidden units as D, please provide the size of each training parameter above.

(b) (30 points) To train the model, we use *cross-entropy* loss. For time step t (note that for language model, we have loss for every time step), we have:

$$E^{(t)}(\theta) = CE(y^{(t)}, \hat{y}^{(t)}) = -\sum_{j=1}^{|V|} y_j^{(t)} \log(\hat{y}_j^{(t)}),$$

where $y^{(t)}$ is the one-hot vector corresponding to the target word, which here is equal to $x^{(t+1)}$. For a sequence, we sum and average the loss of every time step.

PyTorch does not require the implementation of back-propagation, but you should know the details. **Compute the following gradients at a single time step t:**

$$\frac{\partial E^{(t)}}{\partial U}, \frac{\partial E^{(t)}}{\partial b_2}, \frac{\partial E^{(t)}}{\partial I}\bigg|_{(t)}, \frac{\partial E^{(t)}}{\partial H}\bigg|_{(t)}, \frac{\partial E^{(t)}}{\partial b_1}\bigg|_{(t)}, \frac{\partial E^{(t)}}{\partial h^{(t-1)}}$$

where $|_{(t)}$ denotes the gradient with respect to time step t only. Note that we have weight sharing in recurrent layer, so in practice, the back-propagation would update the parameters according to the gradients across all time steps. Hint:

$$\frac{\partial E^{(t)}}{\partial U} = (h^{(t)})^T(\hat{y}^{(t)} - y^{(t)})$$

Make sure you understand this hint and you can use it directly in your answer.

(c) (10 points) To evaluate a language model, we use *perplexity*, which is defined as the inverse probability of the target word according to the model prediction $\bar{P}$:

$$PP^{(t)}(y^{(T)}, \hat{y}^{(t)}) = \frac{1}{\bar{P}(x_{t+1}^{pred} = x_{t+1}|x_t, \ldots, x_1)} = \frac{1}{\sum_{j=1}^{|V|} y_j^{(t)} \hat{y}_j^{(t)}}.$$

Show the relationship between *cross-entropy* and *perplexity*.

(d) (50 points) Read the starting code very carefully and implement the above model by completing RNNLM.py. You only need to code some parts related to the RNN model, such as the architecture, loss function. Follow the instructions in the starting code to understand which parts need to be filled in. When you are done, run python RNNLM.py. The starting code supports GPU computation and also provides guidelines on how to support CPU. It is very straightforward but note that CPU computation is slow! You should change the hyperparameters to explore the best configurations. Submit the code

with your best hyperparameters. Also report the best testing perplexity score. This model is a *generative* model. At the end of the run, it uses the trained language model to generate sentences with the start words you provide. Be creative and report any results that you think are interesting. Please summarize your results and observations.

2. (30 points) As introduced in class, the attention mechanism can be written into:

$$\text{Attention}(Q, K, V) = \text{softmax}(QK^T)V.$$

By adding linear transformations on Q, K, and V, it turns into:

$$\text{Attention}(QW^Q, KW^K, VW^V) = \text{softmax}\left(QW^Q(KW^K)^T\right)VW^V.$$

Here, we set $Q \in \mathbb{R}^{n\times d}$, $W^Q \in \mathbb{R}^{d\times d}$, $K \in \mathbb{R}^{n\times d}$, $W^K \in \mathbb{R}^{d\times d}$, $V \in \mathbb{R}^{n\times d}$, $W^V \in \mathbb{R}^{d\times d}$.

We've also covered the multi-head attention:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(head_1, \ldots, head_h),$$

where

$$head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V),\ i = 1, \ldots, h.$$

Here, Q, K, V are the same as defined above. We set $W_i^Q \in \mathbb{R}^{d\times \frac{d}{h}}$, $W_i^K \in \mathbb{R}^{d\times \frac{d}{h}}$, $W_i^V \in \mathbb{R}^{d\times \frac{d}{h}}$.

(a) (15 points) Compute and compare the number of parameters between the single-head and multi-head attention.

(b) (15 points) Compute and compare the amount of computation between the single-head and multi-head attention, including the softmax step. Use the big-O notation to show your results.

(Hint1: For (b), what we talked about in class may not be precise.)

(Hint2: Quoted from the paper (`https://arxiv.org/pdf/1706.03762.pdf`), "Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.")

3. (20 points) For deep learning on graph data, we have learned GCNs, which aggregate information from neighboring nodes. The layer-wise forward-propagation operation of GCNs can be expressed as

$$X^{l+1} = \sigma(AX^lW^l),$$

where X^l and X^{l+1} are the input and output matrices of layer l, respectively, and A is the adjacency matrix.

(a) (10 points) One limitation of this simple model is that multiplication with A means each center node sums up feature vectors of all neighboring nodes but not the node itself. How to fix this limitation with a simple modification on A?

(b) (10 points) Another limitation is that A is not normalized. The multiplication of A will change the scale of feature vectors. Suppose we want to normalize A such that all rows sum to one. How to fix this limitation with a simple modification on A?