DSA5207 – Assignment 2

Due: 04/04/2025

Name: ______________________________

Student Number: ______________________________

This written homework assignment tests your overall understanding of the material covered in the class so far. There are 2 types of questions: written and coding questions.

For written questions, please scan and upload your solutions to the Canvas.

The coding question is for you to apply the methods covered in class to a practical problem. Questions marked with "coding" next to the assigned points require a coding part in `submission.py`. Submit `submission.py`. You can use functions in `util.py`. However, please do not import additional libraries (e.g. `numpy`, `sklearn`) that are not mentioned in the assignment. You can run `test.py` to test your code but you don't need to submit it.

Question:	1	2	3	4	Total
Points:	5	8	12	15	40
Score:					

1. (5 points) **Multiple Choice Questions.**

   (a) Using tf-idf (term frequency, inverse document frequency) values for features in a unigram bag-of-words model should have an effect most similar to which of the following?

   ◯ Lowercasing the data
   ◯ Removing stopwords
   ◯ Increasing the learning rate
   ◯ Dropout regularization

   (b) Which of the following statements is INCORRECT?

   ◯ Recurrent neural networks can handle a sequence of arbitrary length, while feedforward neural networks can not
   ◯ Training recurrent neural networks is hard because of vanishing and exploding gradient problems
   ◯ Gradient clipping is an effective way of solving vanishing gradient problem
   ◯ RNNs have fewer parameters than LSTMs

   (c) Which of the following statement about Skip-gram is CORRECT?

   ◯ It predicts the center word from the surrounding context words
   ◯ When it comes to a small corpus, it has better performance than other methods
   ◯ It makes use of global co-occurrence statistics
   ◯ The final word vector for a word is the average or sum of the input vector v and output vector u corresponding to that word

   (d) Suppose a classifier predicts each possible class with equal probability. If there are 10 classes, what will the cross-entropy error be on a single example?

   ◯ $-\log(0.1)$
   ◯ $-\log(10)$
   ◯ $-0.1\log(1)$
   ◯ $-10\log(0.1)$

   (e) Which level of language cares about the literal meaning of a sentence?

   ◯ Pragmatics
   ◯ Morphology
   ◯ Syntax
   ◯ Semantics

2. **(N-gram Language Models)** In this problem, we will derive the MLE solution of n-gram language models. Recall that in n-gram language models, we assume that a token only depends on $n-1$ previous tokens, namely:

$$p(x_{1:m}) = \prod_{i=1}^{m} p(x_i \mid x_{i-n+1:i-1}) ,$$

   where $x_i \in \mathcal{V}$ and $x_{1:i}$ denotes a sequence of i tokens $x_1, x_2, \ldots, x_i$. Note that we assume all sequences are prepended with a special start token $*$ and appended with the stop token `STOP`, thus $x_i = *$ if $i < 1$ and $x_m = \texttt{STOP}$. We model the conditional distribution $p(x_i \mid x_{i-n+1:i-1})$ by a categorical distribution with parameters α:

$$p(w \mid c) = \alpha[w, c] \quad \text{for } w \in \mathcal{V}, c \in \mathcal{V}^{n-1} .$$

   Let $D = \{x^i_{1:m_i}\}_{i=1}^{N}$ be our training set of N sequences, each of length m_i.

   (a) (2 points) Write down the MLE objective for the n-gram model defined above. Note that we need to add the constraint that the conditional probabilities sum to one given each context.

(b) (2 points) Recall that the method of Lagrange multipliers allows us to solve an optimization problem with equality constraints by forming a Lagrangian function, which can be optimized without explicitly parameterizing in terms of the constraints.

Given an optimization problem to maximize $f(x)$ subject to the constraint $g(x) = 0$, we can express it in the form of the Langrangian, which can be written as $f(x) - \lambda g(x)$.

Write down the Langrangian $\mathcal{L}(\alpha, \lambda)$ for the MLE objective using the method of Lagrange multipliers.

(c) (4 points) Find the solution for α. Define $\text{count}(\cdot)$ to be a function which maps a sequence to its frequency in D. You can assume $\text{count}(c) > 0$ for $c \in \mathcal{V}^{n-1}$.

3. **(Skip-gram Model)** In this problem, we will gain some insights into the skip-gram model. In particular, we will show that it encourages the inner product between two word embeddings to be equal to their pointwise mutual information (PMI) (up to a constant).

Recall that to evaluate the objective function of the skip-gram model we need to enumerate over the entire vocabulary $\mathcal{V}$, which can be quite expensive in practice. Note that the reason we need to enumrate the vocabulary is to obtain a probability distribution of neighboring words.

Instead of modeling the distribution of words, we can model the distribution of an indicator: whether two words are neighbors. Let Y be the indicator random variable. We model it by a Bernoulli distribution

$$p_\theta(y = 1 \mid w, c) = \frac{1}{1 + e^{-u_c \cdot v_w}} ,$$

where $c \in \mathcal{V}$ is within a window centered at $w \in \mathcal{V}$, u_c, v_w represent embeddings of c and w, and θ is the model parameter, i.e. word embedding v's and context embedding u's.

What about the negative observations ($y = 0$)? A naive way is to consider all words that do not occur in the neighborhood of w, however, this is as expensive as our original objective. One solution is **negative sampling**, where we only consider a small number of non-neighboring words.

Let $D = \{(w, c)\}$ be the set of all word-neighbor pairs observed in the training set. For each pair (w, c), we generate k negative neighbors c_N by sampling from a distribution $p_N(\cdot)$ over the vocabulary.

(a) (3 points) Let L be the learning objective that maximizes the log-likelihood of D and the *expected* log-likelihood of the negative examples. Note that C_N is the random variable in the expectation. Show that

$$L = \max_\theta \sum_{(w,c)\in D} \log \frac{1}{1 + e^{-u_c \cdot v_w}} + k \sum_{(w,\cdot)\in D} \sum_{c_N \in \mathcal{V}} p_N(c_N) \log \frac{1}{1 + e^{u_{c_N} \cdot v_w}} \tag{1}$$

(b) (2 points) Let

$$\sigma(\alpha) = \frac{1}{1 + e^{-\alpha}} .$$

Show that

$$\frac{d}{d\alpha} \log \sigma(\alpha) = \frac{1}{1 + e^{\alpha}} .$$

(c) (3 points) Let's consider one pair $(w = b, c = a)$. Let $\alpha = u_a \cdot v_b$. Show that the optimal solution of α is

$$\alpha^* = \log \frac{A}{B} ,$$

where

$$A = \sum_{(w,c)\in D} \mathbb{1}[w = b, c = a] \tag{2}$$

$$B = k \sum_{(b,\cdot)\in D} p_N(a) \ . \tag{3}$$

(d) (2 points) Suppose the distribution of negative words $p_N(w)$ for $w \in \mathcal{V}$ is a categorical distribution given by

$$p_N(w) = \frac{\text{count}(w, \cdot, D)}{|D|} \ ,$$

where $\text{count}(w, \cdot, D) = \sum_{c\in\mathcal{V}} \text{count}(w, c, D)$. Note that $p_N(w)$ is simply the fraction of unigram w in D. Show that

$$\alpha^* = \text{PMI}(b, a) - \log k \ ,$$

where the PMI score of word co-occurence in D is defined as

$$\text{PMI}(b, a) \stackrel{\text{def}}{=} \log \frac{\text{count}(b, a, D)|D|}{\text{count}(b, \cdot, D)\text{count}(a, \cdot, D)} \ .$$

(e) (2 points) Let's denote the unigram model defined above by $p_{\text{unigram}}(w)$. In practice, we often prefer to use $p_N(w) \propto p^{\beta}_{\text{unigram}}(w)$ where $\beta \in [0, +\infty]$. Describe the desired range of β and explain why.

4. **(Conditional Random Fields)** In this problem, you will implement inference algorithms for the CRF model and compare different sequence prediction models on synthetic data. You may want to go over the `mxnet_tutorial.ipynb` first before you start.

   Environment setup: Follow instructions in `README.md` to set up the environment for running the code.

   (a) (2 points) To get started, take a look at the function `generate_dataset_identity` in `util.py` and the class `UnigramModel` in `model.py`. Given $x = (x_1, \ldots x_n)$ where $x_i \in \mathcal{V}$, the model makes an independent prediction at each step using only input at that step, i.e. $p(y_i \mid x_i)$. Run `python test.py unigram` to train a `UnigramModel`. It outputs the average hamming loss in the end. Let $y = (y_1, \ldots, y_n)$ be the gold labels and $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_n)$ be the predicted labels, take a look at `hamming_loss` in `submission.py` and write down the loss function.

   (b) (2 points) Take a look at the `RNNModel` in `model.py`. It uses a bi-directional LSTM to encode x and makes independent predictions for each y_i. This time let's use the dataset generated by `generate_dataset_rnn`. Compare the result by running `python test.py unigram -data rnn` and `python test.py rnn -data rnn`. Which model has a lower error rate? Explain your findings.

   (c) (4 points) [coding] Next, we are going to add a CRF layer on top of the RNN model (see `CRFRNNModel` in `model.py`). Here we use the autograd function in MXNet to compute gradient for us, so we only need to implement the forward pass (the counterpart of the forward algorithm). Take a look at `crf_loss`. The main challenge here is to compute the normalizer which sums over all possible sequences:

$$\begin{aligned} \text{normalizer} &= \sum_{y\in\mathcal{Y}^n} \exp\left[s(y)\right] \\ &= \sum_{y\in\mathcal{Y}^n} \exp\left[\sum_{i=1}^{n} u(y_i) + \sum_{i=2}^{n} b(y_i, y_{i-1})\right] \end{aligned}$$

where u and b are scores from the CRFRNNModel. Note that here we assume $y_1 = *$ (the start symbol). Implement compute_normalizer using the logsumexp function in util.py. Your result must match bruteforce_normalizer. [**HINT:** You can compute all sums using array operations. np.expand_dims is very helpful here.]

See submission.py. No written submission.

(d) (4 points) [coding] During inference, we will use Viterbi decoding to find

$$\arg\max_{y \in \mathcal{Y}^n} s(y)$$

where $s(y) = \sum_{i=1}^{n} u(y_i) + \sum_{i=2}^{n} b(y_i, y_{i-1})$. Implement viterbi_decode. Your result must match bruteforce_decode. [**HINT:** You can compute all sums using array operations. np.expand_dims is very helpful here.]

See submission.py. No written submission.

(e) (3 points) [coding] We are ready to test the CRFRNN model now. Use the HMM data (take a look at generate_dataset_hmm in util.py) and compare it with the RNN model by running `python test.py rnn -data hmm` and `python test.py crfrnn -data hmm`. Compare the results. [**NOTE:** This is an open-ended question. Discuss any findings you have is fine, e.g. runtime, error rate, convergence rate etc.]