Assignment 1

RELEASE DATE: 01/28/2026

DUE DATE: 02/10/2026 11:59pm on Gradescope

LaTeX Template: https://www.overleaf.com/read/tsrvwjgzwjrw#942964

Name: First-Name Last-Name UIN: 000000000

This assignment consists of two parts: a writing section and a programming section. For the writing section, please use the provided LaTeX template to prepare your solutions and remember to fill in your name and UIN. For the programming section, please follow the instructions carefully.

Discussions with others on course materials and assignment solutions are encouraged, and the use of AI tools as assistance is permitted. However, you must ensure that ***the final solutions are written in your own words****. It is your responsibility to avoid excessive similarity to others' work. Additionally, please clearly* ***indicate any parts where AI tools were used*** *as assistance.*

If you have any question, please send an email to csce638-ta-26s@list.tamu.edu

1 Deriving Derivatives [20pts]

1.1 Sigmoid Function [6pts]

In the class, we introduced the *sigmoid function*

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

(a) [3pts] Please show that the derivative of $\sigma(t)$ can be calculated in the following way

$$\sigma(t)' = \sigma(t) \cdot (1 - \sigma(t))$$

Solution:

Please enter your solution here.

(b) [3pts] Please show that

$$(1 - \sigma(t))' = -\sigma(t) \cdot (1 - \sigma(t))$$

Solution:

Please enter your solution here.

1.2 Multiclass Logistic Regression [7pts]

In the class, we introduced the multiclass logistic regression as follows. Given an example $(\mathbf{x}, y)$,

$$z_c = \mathbf{w}_c \cdot \mathbf{x} + b_c$$

$$P(y = c|\mathbf{x}) = \text{softmax}(z_c) = \frac{\exp(\mathbf{w}_c \cdot \mathbf{x} + b_c)}{\sum_{i=1}^{K} \exp(\mathbf{w}_i \cdot \mathbf{x} + b_i)}$$

where

- K is the number of classes
- $\mathbf{x}$ is the input feature vector
- y is the ground truth label for $\mathbf{x}$
- $\mathbf{w}$ is the weight parameter with $\mathbf{w}_c$ is the linear weight vector for class c

The *cross entropy loss* for classification is

$$\mathcal{L}(\mathbf{w}) = -\sum_{i=1}^{K} y_i \log P(y = i|\mathbf{x})$$

where $y_i = 1$ if y belongs to class i, otherwise $y_i = 0$.

(a) [6pts] Show that

$$\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}_c} = \begin{cases} (P(y = c|\mathbf{x}) - 1) \cdot \mathbf{x} & \text{If } y \text{ belongs to class } c \\ P(y = c|\mathbf{x}) \cdot \mathbf{x} & \text{If } y \text{ does not belong to class } c \end{cases}$$

Solution:

Please enter your solution here.

(b) [1pts] Based on (a), explain why the gradient encourages $\mathbf{w}_c$ to make more accurate predictions.

Solution:

Please enter your solution here.

1.3 Context Word Vectors in Skip-Gram [7pts]

In the class, we introduced the idea of using skip-gram to learn word vectors. Specifically, we have two sets of vectors $\mathbf{u}_w$ and $\mathbf{v}_w$

$$\mathbf{u}_w : \text{the vector when } w \text{ is the center word}$$

$$\mathbf{v}_w : \text{the vector when } w \text{ is the context word}$$

Given a center word x and a context word c, the probability of predict c based on x is defined as

$$P(c|x) = \frac{\exp(\mathbf{u}_x \cdot \mathbf{v}_c)}{\sum_{k \in \mathcal{V}} \exp(\mathbf{u}_x \cdot \mathbf{v}_k)}$$

where $\mathcal{V}$ is the vocabulary set. The objective function of skip-gram is minimizing *negative log likelihood*

$$y = -\log P(c|x) = -\log \left(\frac{\exp(\mathbf{u}_x \cdot \mathbf{v}_c)}{\sum_{k \in \mathcal{V}} \exp(\mathbf{u}_x \cdot \mathbf{v}_k)} \right)$$

(a) [6pts] For each word k in $\mathcal{V}$, show that

$$\frac{\partial y}{\partial \mathbf{v}_k} = \begin{cases} (P(k|x) - 1) \cdot \mathbf{u}_x & \text{If } k = c \\ P(k|x) \cdot \mathbf{u}_x & \text{If } k \neq c \end{cases}$$

Solution:

Please enter your solution here.

(b) [1pts] Based on (a), explain why the gradient encourages $\mathbf{v}_k$ to capture more accurate word similarity.

Solution:

Please enter your solution here.

2 N-Gram Features and Linear Classifier [15pts]

Consider the following training examples for a binary sentiment classification.

Label	Sentence
+	`<s> I like apples </s>`
−	`<s> I hate apples </s>`
−	`<s> I do not like apples </s>`
+	`<s> I do not hate apples </s>`
−	`<s> grapes are bad </s>`

where `<s>` and `</s>` are special tokens that represent the start and the end of a sentence, respectively.

(a) [2pts] Let's consider bag-of-words (unigrams) to construct vector representations for sentences. Each sentence can be represented by a 11-dimensional *binary* vector $\mathbf{x}$, where the dimensions in the vector correspond to the following order:

`I`, `like`, `hate`, `apples`, `grapes`, `do`, `not`, `are`, `bad`, `<s>`, `</s>`.

Please list the vectors for all training examples above.

Solution:

Please enter your solution here.

(b) [3pts] Let's consider logistic regression to classify sentences with the following probability

$$P(y = +|\mathbf{x}) = \text{sigmoid}(\mathbf{w} \cdot \mathbf{x} + b)$$

and the following prediction function

$$y = \begin{cases} + & \text{If } P(y = +|\mathbf{x}) \geq 0.5 \\ - & \text{If } P(y = +|\mathbf{x}) < 0.5 \end{cases}$$

Is it possible to construct a set of parameters $(\mathbf{w}, b)$ that can make correct predictions for all training examples? If yes, please construct the parameters $(\mathbf{w}, b)$ and calculate the probability $P(y = +|\mathbf{x})$ for all training examples. If no, please explain the reason.

Solution:

Please enter your solution here.

(c) [2pts] Let's consider bag-of-words (unigrams) and bigrams to construct vector representations for sentences. Each sentence can be represented by a 25-dimensional *binary* vector $\mathbf{x}$, where the dimensions in the vector correspond to the following order:

`I`, `like`, `hate`, `apples`, `grapes`, `do`, `not`, `are`, `bad`, `<s>`, `</s>`, `<s> I`, `I like`, `I hate`, `like apples`, `hate apples`, `apples </s>`, `I do`, `do not`, `not like`, `not hate`, `<s> grapes`, `grapes are`, `are bad`, `bad </s>`.

Please list the vectors for all training examples above.

Solution:

Please enter your solution here.

(d) [3pts] Similar to (b), let's consider logistic regression to classify sentences. Is it possible to construct a set of parameters $(\mathbf{w}, b)$ that can make correct predictions for all training examples? If yes, please construct the parameters $(\mathbf{w}, b)$ and calculate the probability $P(y = +|\mathbf{x})$ for all training examples. If no, please explain the reason.

Solution:

Please enter your solution here.

(e) [2pts] Following (d), let's consider the following two testing examples.

Label	Sentence
+	`<s> I like grapes </s>`
−	`<s> I hate grapes </s>`

Is it possible to construct a set of parameters $(\mathbf{w}, b)$ that can make correct predictions for all training examples, while making correct predictions for all *testing* examples at the same time? If yes, please construct the parameters $(\mathbf{w}, b)$ and calculate the probability $P(y = +|\mathbf{x})$ for all testing examples. If no, please explain the reason.

Solution:

Please enter your solution here.

(f) [3pts] Similar to (d), is it possible to construct a set of parameters $(\mathbf{w}, b)$ that can make correct

predictions for all training examples, while making *incorrect* predictions for all *testing* examples at the same time? If yes, please construct the parameters $(\mathbf{w}, b)$ and calculate the probability $P(y = +|\mathbf{x})$ for all testing examples. If no, please explain the reason.

Solution:

Please enter your solution here.

3 Language Modeling [16pts]

Consider the following corpus of text

```
<s> I like raspberries </s>
<s> You like all berries </s>
<s> I hate sour fruits </s>
<s> You like all sweet fruits </s>
<s> I like chocolate covered raspberries </s>
```

where `<s>` and `</s>` are special tokens that represent the start and the end of a sentence, respectively. In the following questions, consider splitting sentences into tokens by whitespace only.

3.1 Unigram Language Model [8pts]

(a) [4pts] Calculate the unigram probabilities for this corpus, including the detailed steps. The results should be displayed in a table format. You can learn some basics about Tables here.

Example table format:

Unigram	Probability
$P($`<s>`$)$	...
$P($`I`$)$	...
...	...

Solution:

Please enter your solution here.

(b) [2pts] Using the estimate from (a), calculate the *probabilities* for the following sentences.

```
<s> You like all berries </s>
```

```
<s> You hate sour fruits </s>
```

Solution:

Please enter your solution here.

(c) [2pts] Using the estimate from (a), calculate the *perplexity* for the following corpus.

```
<s> I like all sweet fruits </s>
<s> You like raspberries </s>
```

Solution:

Please enter your solution here.

3.2 Bigram Language Model [8pts]

(a) [4pts] Consider the same corpus at the beginning. Calculate the *all non-zero* bigram probabilities for this corpus, including the detailed steps. The results should be displayed in a table format.

Example table format:

Unigram	Probability
$P(\texttt{I}\vert\texttt{<s>})$	...
$P(\texttt{like}\vert\texttt{I})$	...
...	...

Solution:

Please enter your solution here.

(b) [2pts] Using the estimate from (a), calculate the *probabilities* for the following sentences.

```
<s> You like all berries </s>
```

```
<s> You hate sour fruits </s>
```

Solution:

Please enter your solution here.

(c) [2pts] Using the estimate from (a), calculate the *perplexity* for the following corpus.

```
<s> I like all sweet fruits </s>
<s> You like raspberries </s>
```

Solution:

Please enter your solution here.

4 Word Embeddings (Programming) [18pts]

`CSCE638-S26-HW1-4.ipynb`: Colab Notebook

`glove.6B.50d.txt`: Data

Please open the above Colab Notebook with Google Colab. **Remember to use your *@tamu.edu* email to access the Colab Notebook.** Copy the Colab Notebook to your Google drive and make the changes. The notebook has marked blocks where you need to code:

```
### ========= TODO : START ========= ###
```

```
...
### ========= TODO :  END  ========= ###
```

Please copy and paste your code (between `TODO:START` and `TODO:END`) **as part of the solution.** You can use the Minted package for code highlighting. Here is one example:

```
def hello_world():
    print("Hello World!")
```

4.1 Loading GloVe Embeddings [4pts]

Please follow the instructions and implement the related parts to load the GloVe word embeddings. Each word should map to a 50-dimensional vector. Copy and paste your code (between `TODO:START` and `TODO:END`) as well as the output of `test_glove_loading`.

Solution:

Please enter your solution here.

4.2 Compute Word Similarity [4pts]

Let's consider the cosine similarity. Given two vectors $\mathbf{x}$ and $\mathbf{y}$, the cosine similarity is

$$\frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|}$$

Please follow the instructions and implement the related parts to compute word similarity. Copy and paste your code (between `TODO:START` and `TODO:END`) as well as the output of `test_word_similarity`.

Please also report your findings.

Solution:

Please enter your solution here.

4.3 Word Analogy [5pts]

Let's study the following word analogy test.

$$\text{word A} \ : \ \text{word B} \ \approx \text{word C} \ : \ ?$$

If we get the word vectors $\mathbf{w}_a, \mathbf{w}_b, \mathbf{w}_c$ corresponding to word A, word B, and word C, it is equivalent to solve the following optimization problem

$$v^* = \arg \max_{v \in \mathcal{V}, v \neq \text{word C}} \text{CosineSimilarity}(\mathbf{w}_v, \mathbf{w}_b - \mathbf{w}_a + \mathbf{w}_c)$$

Please follow the instructions and implement the related parts to study word analogy. Copy and paste your code (between `TODO:START` and `TODO:END`) as well as the output of `test_word_analogy`.

Please also report your findings.

Solution:

Please enter your solution here.

4.4 Visualizing Word Embeddings [5pts]

To visualize word embeddings, we first consider principal component analysis (PCA) to reduce the dimension of word vectors from 50 to 2. Then, we use `matplotlib` to generate the visualization. You can learn how to compute PCA with `sklearn` here.

Please follow the instructions and implement the related parts to study word analogy. Copy and paste your code (between `TODO:START` and `TODO:END`) as well as the output of `visualize`.

Please also report your findings.

Solution:

Please enter your solution here.

5 Text Classification with Neural Networks (Programming) [31pts]

We will follow a step-by-step pipeline to construct a simple deep neural network classifier based on bag-of-words embeddings. We will consider `PyTorch`, a popular python framework that is widely used for building and training neural networks. You can learn some basics about `PyTorch` here.

`CSCE638-S25-HW1-5.ipynb`: Colab Notebook

`glove.6B.50d.txt`: Data

`HW1-5_train.json`: Data

`HW1-5_valid.json`: Data

`HW1-5_test.json`: Data

Please open the above Colab Notebook with Google Colab. **Remember to use your *@tamu.edu* email to access the Colab Notebook.** Copy the Colab Notebook to your Google drive and make the changes. The notebook has marked blocks where you need to code:

```
### ========= TODO : START ========= ###
...
### ========= TODO :  END  ========= ###
```

Please copy and paste your code (between `TODO:START` and `TODO:END`) **as part of the solution.** You can use the Minted package for code highlighting. Here is one example:

```
def hello_world():
    print("Hello World!")
```

5.1 Loading GloVe Embeddings [1pts]

Similar to problem 4.1. Please follow the instructions and implement the related parts to load the GloVe embeddings. Copy and paste your code (between `TODO:START` and `TODO:END`) as well as the output of `test_glove_loading`.

Solution:

Please enter your solution here.

5.2 Loading Data and Tokenization [4pts]

Please download `train.json`, `valid.json`, and `test.json` from the links above and set the correct path. We consider a subset of AG News dataset. It's a dataset for 4-class classification. The input will be a short description of a news article, and the output will be 0/1/2/3, which corresponds to 4 different topics World/Sports/Business/Tech.

We are going to use the NLTK package for tokenization. You can learn more about how to use the `word_tokenize` function here.

Please follow the instructions and implement the related parts to load and tokenize data. Copy and paste your code (between `TODO:START` and `TODO:END`) as well as the output of `test_tokenization`.

Solution:

Please enter your solution here.

5.3 Preparing Bag-of-Words Vectors [4pts]

We consider bag-of-words embeddings as the feature vector. We use the average GloVe embeddings of all tokenized words, and ignore any word that is not in the GloVe vocabulary. That means each example will map to a 50-dimensional vector.

Please follow the instructions and implement the related parts to prepare bag-of-words vectors. Copy and paste your code (between `TODO:START` and `TODO:END`) as well as the output of `test_bow_vector`.

Solution:

Please enter your solution here.

5.4 Preparing Dataset and DataLoader [2pts]

We will use `torch.utils.data.Dataset` and `torch.utils.data.DataLoader` to compile the data for training. You can learn more about them here.

Please follow the instructions and implement the related parts to prepare Dataset and DataLoader. Copy and paste your code (between `TODO:START` and `TODO:END`).

Solution:

Please enter your solution here.

5.5 Preparing Model [4pts]

We will consider a two-layer neural network

$$y_c = \text{Softmax}(\mathbf{W}_2^\top \varphi(\mathbf{W}_1^\top \mathbf{x}))$$

where $\mathbf{W}_1$ is a 50×100 wight matrix, $\mathbf{W}_2$ is a 100×4 wight matrix, and φ is the ReLU activation function. You can learn how to use `torch.nn.Linear` and other `PyTorch` function to build a model here.

Please follow the instructions and implement the related parts to build the model. Copy and paste your code (between `TODO:START` and `TODO:END`) as well as the output of `test_model`.

Solution:

Please enter your solution here.

5.6 Evaluating Accuracy [4pts]

Please follow the instructions and implement the related parts to evaluate the model accuracy for validation set. You can check the `test_loop` in this tutorial for reference. Copy and paste your code (between `TODO:START` and `TODO:END`) as well as the output of `test_evaluate_acc`. Since we have not started the training yet, it is possible if the accuracy is low now.

Solution:

Please enter your solution here.

5.7 Training [6pts]

We are going to use the Adam optimizer to train the model with learning rate 0.01. We will set the loss function to the CrossEntropyLoss. During training, we will evaluate the current model's performance on the validation set in each iteration, and save the best checkpoint for future testing.

Please follow the instructions and implement the related parts to train the model. You can check the `train_loop` in this tutorial for reference. Copy and paste your code (between `TODO:START` and `TODO:END`) as well as the cell output. If you implement everything correctly, after running the notebook cell, you should get outputs similar to the following (the numbers can be different).

```
Epoch [1/100], Loss: 1.3878025674819947, Valid Acc: 0.295
Epoch [2/100], Loss: 1.3806762647628785, Valid Acc: 0.365
Epoch [3/100], Loss: 1.3604189085960388, Valid Acc: 0.3525
Epoch [4/100], Loss: 1.3338648283481598, Valid Acc: 0.43
```

```
Epoch [5/100], Loss: 1.3066564297676087, Valid Acc: 0.58
Epoch [6/100], Loss: 1.286200487613678, Valid Acc: 0.545
...
```

Solution:

Please enter your solution here.

5.8 Testing [6pts]

The last step is load the best check point and evaluate the testing performance. You can learn more about saving and loading `PyTorch` models here.

Please follow the instructions and implement the related parts to test performance. Copy and paste your code (between `TODO:START` and `TODO:END`) as well as the final accuracy on the test set.

During grading, **we will replace the test file with another hidden file to test the performance of your model**. Please make sure that there won't be any runtime errors.

Solution:

Please enter your solution here.

Submission Instructions

You have to upload two files to Gradescope:

- `report.pdf`: The `.pdf` file generated by the LaTeX template. Please remember to **annotate the correct page for each question** on Gradescope. Failure to do so may result in a grade penalty.
- `programming.zip`: A `.zip` file contains the following:
  - `problem4.py`: Please export the Colab Notebook to a `.py` file by clicking "File" → "Download" → "Download .py"
  - `problem4.ipynb`: Please execute and export the Colab Notebook to a `.ipynb` file by clicking "File" → "Download" → "Download .ipynb"
  - `problem5.py`: Please export the Colab Notebook to a `.py` file by clicking "File" → "Download" → "Download .py"
  - `problem5.ipynb`: Please execute and export the Colab Notebook to a `.ipynb` file by clicking "File" → "Download" → "Download .ipynb"