Assignment 1 Report

Anay Kulkarni

Part 1A

The goal of these experiments was to evaluate the performance of different configurations of neural networks (NNs) for sentiment analysis, using GloVe embeddings and a Deep Averaging Network (DAN) architecture. Each network was tested across 50 epochs, and the results are presented based on test and train accuracy. We explored variations in loss functions, activation functions, optimizers, and the depth of the neural network.

Experimental Setup

- **Dataset**: Movie review sentences classified as positive (1) or negative (0).
- **Embedding**: GloVe embeddings with 300 dimensions, pre-trained.
- **Neural Network Architectures**:
  - 2 fully connected (FC) layers and 1 dropout layer
  - 3 fully connected layers and 2 dropout layers
- **Loss Functions**: Negative Log-Likelihood (NLL) and Cross-Entropy.
- **Activation Functions**: ReLU, Tanh, and Sigmoid.
- **Optimizers**: AdamW, Stochastic Gradient Descent (SGD) with momentum.
- **Batch Sizes**: 16 and 32.
- **Number of Epochs**: 50.

Experiment 1: Effect of Activation Functions and Loss Functions

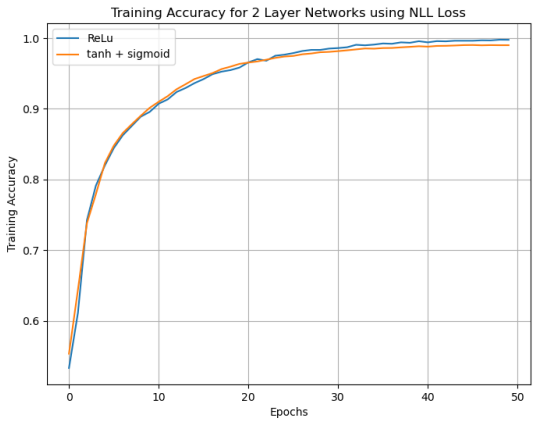

Figure 1: Training Accuracy

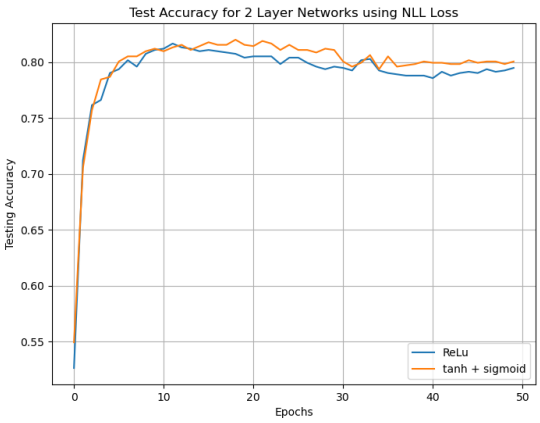

Figure 2: Test Accuracy

Table 1: Performance with Different Activation and Loss Functions

Epoch	**ReLU + NLL** (Train)	**ReLU + NLL** (Test)	**Tanh+Sigmoid + NLL** (Train)	**Tanh+Sigmoid + NLL** (Test)
5	0.820	0.790	0.823	0.787
10	0.896	0.811	0.901	0.812
15	0.936	0.810	0.942	0.814
20	0.958	0.804	0.964	0.815
30	0.985	0.796	0.981	0.811
50	0.998	0.795	0.990	0.800

Discussion:

- **ReLU with NLL** performed better initially, with a higher training accuracy after 50 epochs.
- **Tanh+Sigmoid with NLL** achieved better generalization (test accuracy 0.815) but had slower initial progress compared to ReLU.
- The overfitting trend is more evident with ReLU, as the training accuracy approached 99.8%, while test accuracy remained stagnant or slightly decreased after 20 epochs.

Experiment 2: Comparison Between NLL and Cross-Entropy Loss

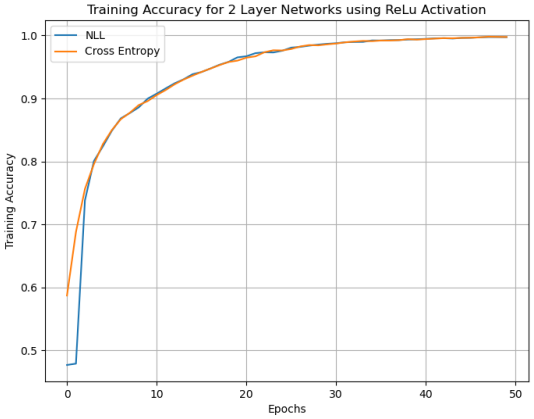

Figure 3: Training Accuracy

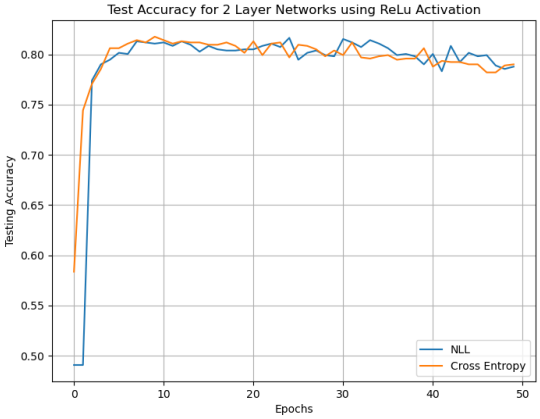

Figure 4: Test Accuracy

Table 2: NLL vs Cross-Entropy with ReLU Activation

Epoch	**ReLU + NLL** (Train)	**ReLU + NLL** (Test)	**ReLU + Cross Entropy** (Train)	**ReLU + Cross Entropy** (Test)
5	0.824	0.795	0.828	0.806
10	0.900	0.811	0.896	0.818
15	0.939	0.803	0.936	0.812
25	0.976	0.817	0.976	0.797
50	0.997	0.788	0.998	0.790

Discussion:

- Cross-Entropy performed marginally better in terms of test accuracy during the initial epochs (e.g., 0.806 at epoch 5), suggesting it was able to generalize better early on.
- However, the difference between Cross-Entropy and NLL became negligible as the epochs increased. Both functions showed slight overfitting.

Experiment 3: Effect of Depth in Neural Network

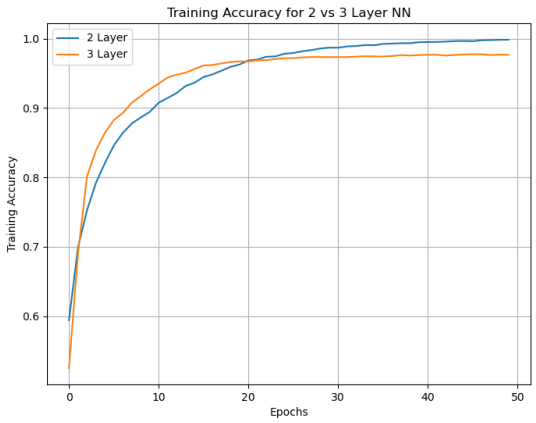

Figure 5: Training Accuracy

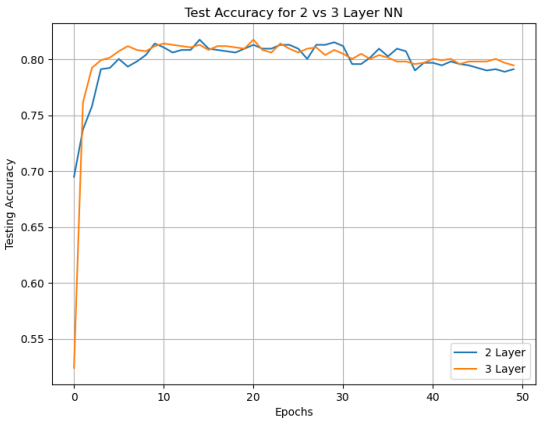

Figure 6: Test Accuracy

Table 3: 2-Layer vs. 3-Layer NN with Different Activation Functions

Epoch	**2-Layer (ReLU)** (Train)	**2-Layer (ReLU)** (Test)	**3-Layer (Tanh+Sigmoid)** (Train)	**3-Layer (Tanh+Sigmoid)** (Test)
5	0.821	0.792	0.864	0.802
10	0.895	0.814	0.927	0.812
20	0.963	0.810	0.967	0.810
30	0.987	0.815	0.973	0.808
50	0.999	0.791	0.977	0.795

Discussion:

- The **3-layer NN** showed more robust generalization during the first 20 epochs, consistently outperforming the 2-layer NN in test accuracy, particularly in the early epochs.
- However, the difference between the two networks diminished as the number of epochs increased.

Experiment 4: Hyperparameter Tuning (Batch Size and Optimizer)

Table 4: Effect of Batch Size and Optimizer on 3-Layer NN (with Tanh+Sigmoid)

Epoch	**Batch Size: 16 + AdamW** (Train)	**Batch Size: 16 + AdamW** (Test)	**Batch Size: 32 + SGD** (Train)	**Batch Size: 32 + SGD** (Test)
5	0.861	0.807	0.661	0.732
10	0.926	0.814	0.745	0.742
25	0.970	0.812	0.770	0.737
50	0.976	0.814	0.800	0.802

Discussion:

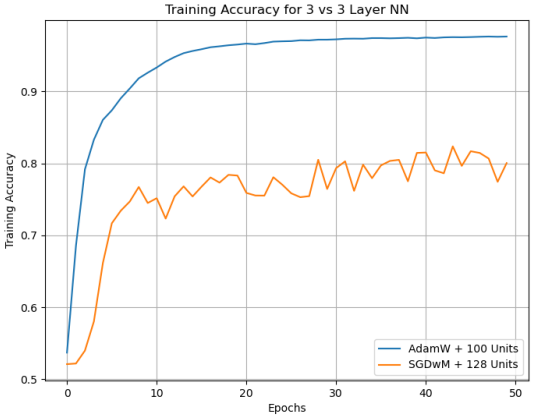

Figure 7: Training Accuracy

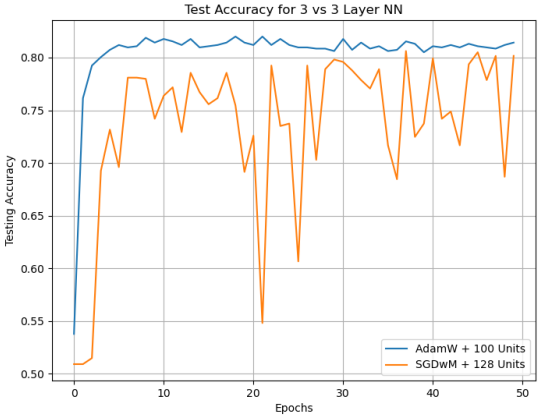

Figure 8: Test Accuracy

- **AdamW optimizer** with a smaller batch size of 16 performed better overall in terms of training and test accuracy.
- The **SGD optimizer** with momentum (0.9) and a batch size of 32 showed slower training progress but improved generalization after 50 epochs.

Conclusion

- **Activation Functions**: The combination of Tanh and Sigmoid performed slightly better than ReLU in terms of generalization, though ReLU enabled faster training.
- **Loss Functions**: Cross-Entropy and NLL had similar performance, with Cross-Entropy showing a slight advantage in early epochs.
- **Network Depth**: The 3-layer NN provided better generalization initially but did not provide significant improvement over the 2-layer NN after prolonged training.
- **Optimizers**: AdamW with a smaller batch size yielded the best overall performance, with steady and consistent improvements in both train and test accuracy.
- **Overall**: The best configuration was the 3 layer fully connected Neural Network using Negative Log Likelihood Loss function and AdamW (Adaptive Moment Estimation) with Weight Decay.
  - Learning Rate: 0.0001
  - Weight Decay: 0.01
  - Hidden Units: 100
  - Batch Size: 16
  - Glove Dimensions: 300

Additional Notes:

Impact of (Over) Preprocessing on Data

- The dataset underwent additional preprocessing steps, including stemming, converting text to lowercase, and removing stop words, using the NLTK library. While these are common preprocessing methods, they actually led to a decline in accuracy in this context. The highest accuracy obtained by the models was 74% on the preprocessed data. This drop in performance could be due to the nature of the reviews, which typically contain short sentences. Aggressive preprocessing, such as stop-word removal and stemming, may have overly simplified the data, reducing its complexity and leading to weaker test performance.

Part 1B

Effects of Training Embeddings from Scratch vs. Fine-Tuning Pre-trained GloVe Embeddings on Model Convergence and Performance

The experiment evaluates the impact of two approaches to word embeddings—fine-tuning pre-trained GloVe embeddings and training random embeddings from scratch—on the performance and convergence of a 3-layer neural network for sentiment classification. The comparison focuses on how these embedding strategies influence training speed, final accuracy, and generalization to unseen test data.

Experimental Setup:

1. **Model Architecture**:
   - 3-layer Neural Network (NN) with an embedding layer, three fully connected layers, two dropout layers, and softmax layer for output
2. **Embedding Types**:
   - **GloVe Embeddings**: Pre-trained on large corpora and fine-tuned during training.
   - **Random Embeddings**: Initialized with random weights and trained from scratch.
3. **Loss Function**: Negative Log-Likelihood (NLL)
4. **Activation Functions**: TanH and Sigmoid
5. **Optimizer**: Adam with weight decay
6. **Evaluation Metric**: Train and test accuracy over 100 epochs.

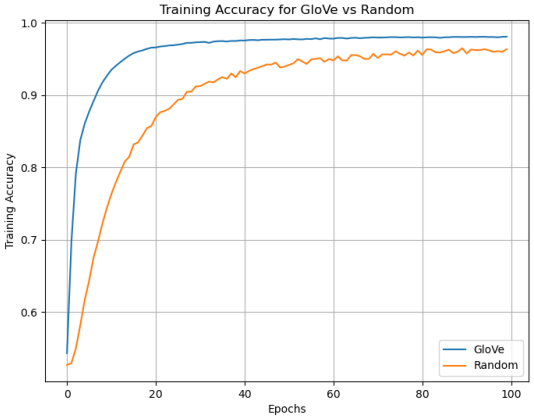

Figure 9: Training Accuracy

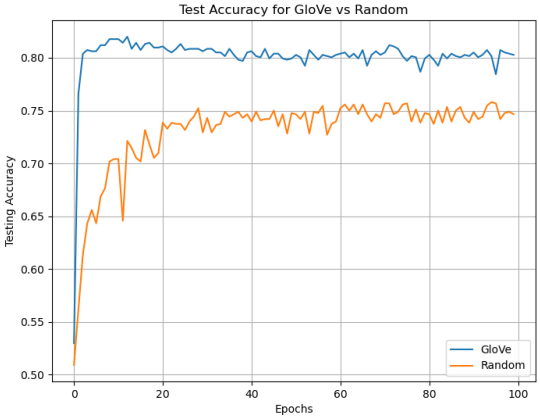

Figure 10: Test Accuracy

Results:

Epoch	Pre-trained GloVe (Train Accuracy)	Pre-trained GloVe (Test Accuracy)	Random Embeddings (Train Accuracy)	Random Embeddings (Test Accuracy)
5	0.861	0.806	0.617	0.656
10	0.927	0.818	0.745	0.704
15	0.955	0.814	0.815	0.705
20	0.966	0.810	0.857	0.710
25	0.969	0.813	0.887	0.737
30	0.973	0.806	0.912	0.729
50	0.977	0.799	0.940	0.748
75	0.980	0.802	0.961	0.756
100	0.981	0.803	0.964	0.747

Analysis:

1. **Convergence Speed**:
   - **GloVe Embeddings**: The model using fine-tuned GloVe embeddings reaches an 80% test accuracy within the first 5 epochs and achieves nearly 98% train accuracy by epoch 25. Test accuracy stabilizes around 80-81% early in training.
   - **Random Embeddings**: Training from scratch results in much slower convergence. At epoch 5, the test accuracy is 65.6%, significantly lagging behind GloVe's 80.6%. It takes until epoch 75 for random embeddings to achieve similar accuracy (75.6%).
2. **Final Accuracy**:
   - **GloVe Embeddings**: By epoch 100, the test accuracy fluctuates around 80.3%. This suggests strong generalization, as the model reaches a performance plateau early on, with minimal overfitting.
   - **Random Embeddings**: After 100 epochs, the random embeddings achieve a test accuracy of 74.7%. This indicates lower generalization compared to the GloVe embeddings, which could be attributed to the lack of prior knowledge in the random vectors.
3. **Generalization and Overfitting**:
   - **GloVe Embeddings**: The relatively close gap between training and test accuracies (around 18% by epoch 100) suggests that fine-tuning pre-trained GloVe embeddings helps maintain good generalization.
   - **Random Embeddings**: There is a larger gap between training and test accuracy, especially in later epochs (96.4% train accuracy vs. 74.7% test accuracy at epoch 100), signaling overfitting as the model learns more detailed but potentially less useful information.
4. **Training Time**:
   - The total training time for both models was similar (around 360 seconds), but **the GloVe model required fewer epochs to reach optimal test performance**. This indicates that GloVe embeddings, which come with pre-learned semantic relationships, allow for faster convergence despite similar computational costs.

Conclusion: Fine-tuning pre-trained GloVe embeddings significantly improves model convergence, resulting in faster achievement of high accuracy compared to training embeddings from scratch. Additionally, models trained with GloVe embeddings exhibit better generalization and lower overfitting, making them a preferable choice when available. On the other hand, training embeddings from scratch takes longer to converge and results in higher risk of overfitting, especially when training data is limited.

PART 2A

In this experiment, we compared the performance of a Deep Averaging Network (DAN) for sentiment analysis using two different tokenization strategies:

1. **Subword Tokenization** with Byte Pair Encoding (BPE)
2. **Word-level Tokenization**

The goal was to understand the impact of using subword tokenization in terms of model performance, training time, and vocabulary compression.

1. Experimental Setup

- **Model**: Deep Averaging Network (DAN)
- **Embedding**: Random embeddings for both subword and word-level tokens
- **Training Duration**: 100 epochs
- **Dataset**: Movie reviews (vocabulary sizes provided below)
- **Metrics**: Train accuracy, test accuracy, training time, evaluation time

2. Results

2.1 Vocabulary Comparison

Metric	Word-Level Vocabulary	BPE Merges	Subword-Level Vocabulary (Initial)	Subword-Level Vocabulary (After BPE)	Compression Ratio
Training Data Vocabulary Size	14,830	1500	65	1376	10.77

After 1500 BPE merges, the subword-level vocabulary size expanded significantly to 1376, achieving an overall compression ratio of 10.77 compared to the word-level vocabulary.

2.2. Training and Evaluation Time

Tokenization Method	Data Load Time (seconds)	Training Time (seconds)	Evaluation Time (seconds)
BPE Subword-Level	0.89	60.15	1.60
Word-Level	1.20	413.98	1.55

The BPE subword-level method had much faster training times (60.15 seconds) compared to the word-level tokenization (413.98 seconds), while evaluation times remained similar for both approaches.

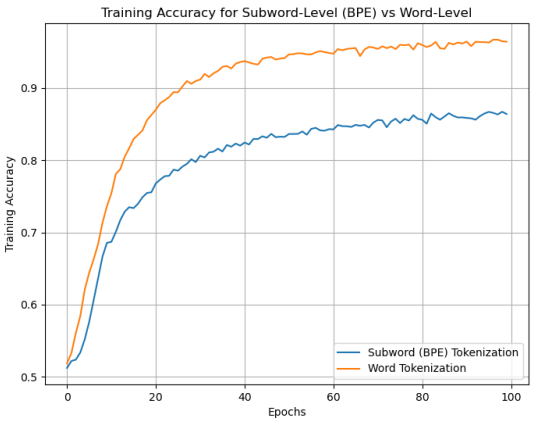

Figure 11: Training Accuracy

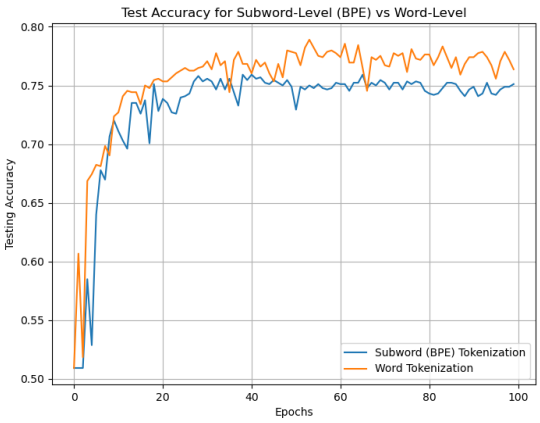

Figure 12: Test Accuracy

2.3 Model Performance Comparison (Accuracy)

Epoch	BPE Subword-Level Train Accuracy	Word-Level Train Accuracy	BPE Subword-Level Test Accuracy	Word-Level Test Accuracy
5	0.552	0.621	0.529	0.674
10	0.686	0.736	0.720	0.724
20	0.756	0.863	0.728	0.756
30	0.798	0.910	0.753	0.766

Epoch	BPE Subword-Level Train Accuracy	Word-Level Train Accuracy	BPE Subword-Level Test Accuracy	Word-Level Test Accuracy
50	0.832	0.942	0.749	0.779
70	0.852	0.956	0.755	0.775
100	0.864	0.964	0.751	0.764

This table shows both the training and test accuracy for each epoch, comparing the results for BPE subword-level tokenization and word-level tokenization.

Training and Test Accuracy

- **BPE Subword-Level Tokenization**:
  - The training accuracy started lower and increased steadily, reaching **86.4%** by epoch 100. However, the test accuracy plateaued around **75%** after reaching a peak of 75.5% at epoch 70.
  - This suggests that subword tokenization helps in gradually improving model performance, but it struggles with generalization beyond a certain point.
- **Word-Level Tokenization**:
  - Training accuracy improved faster, reaching **96.4%** by epoch 100. Test accuracy also peaked higher at **78.4%** at epoch 55 before slightly declining.
  - Word-level tokenization provided better overall generalization and faster convergence compared to the subword approach.

Time Efficiency

- BPE tokenization was significantly more **time-efficient** in training due to a much smaller effective vocabulary size. The training time for BPE was approximately **6.9 times** faster than word-level tokenization.

Compression

- The BPE algorithm achieved a **compression ratio of 10.78**, indicating that a much smaller subword vocabulary could represent the same set of words as a larger word-level vocabulary. This compression did not significantly harm the model's overall performance, though it did result in slightly lower test accuracy.

Generalization

- Word-level tokenization demonstrated better **generalization** as indicated by the higher test accuracy. Subword tokenization seemed to overfit more quickly as evidenced by the diminishing improvement in test accuracy after epoch 70.

3. Conclusion

- **BPE subword tokenization** is highly efficient and yields a significant reduction in the model's vocabulary size, speeding up training times. However, it leads to slightly lower generalization performance compared to word-level tokenization.
- **Word-level tokenization** offers better accuracy at the cost of longer training times, making it suitable when time is not a limiting factor and accuracy is more critical.

Evaluating the Effect of Subword Vocabulary Size on Neural Network Performance

Objective: This experiment aims to evaluate the impact of different subword vocabulary sizes generated using Byte Pair Encoding (BPE) on the performance of a 3-layer neural network, measured in terms of training time, evaluation time, and accuracy. The neural network was trained with randomly initialized weights for subword embeddings, and the model's performance was observed at various BPE subword vocabulary sizes.

Experimental Setup

1. **Initial Vocabulary Sizes:**
   - **Word-level vocabulary size (before BPE):** 14,830
   - **Initial subword vocabulary size (before BPE):** 65
   - **BPE merges**: Carried out over a range of 500 to 5000, with step up of 500.
2. **Metrics Tracked:**
   - **Training Time** (in seconds)
   - **Evaluation Time** (in seconds)
   - **Peak Train Accuracy**
   - **Peak Test Accuracy**

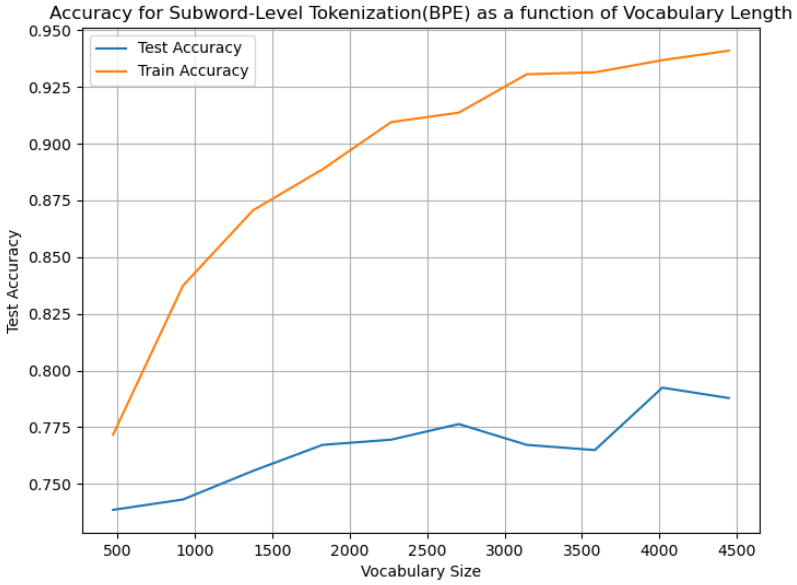

Figure 13: Model performance over varying vocabulary sizes

Results

BPE Merges	Subword Vocabulary Size	Compression Ratio	Training Time (s)	Evaluation Time (s)	Peak Train Accuracy	Peak Test Accuracy
500	472	31.42	46.58	1.75	0.7717	0.7385
1000	924	16.05	51.80	1.68	0.8374	0.7431
1500	1376	10.78	64.82	1.81	0.8707	0.7557
2000	1825	8.13	85.12	1.99	0.8887	0.7672

BPE Merges	Subword Vocabulary Size	Compression Ratio	Training Time (s)	Evaluation Time (s)	Peak Train Accuracy	Peak Test Accuracy
2500	2268	6.54	90.70	1.95	0.9095	0.7695
3000	2705	5.48	109.60	2.22	0.9137	0.7764
3500	3143	4.72	119.10	2.12	0.9306	0.7672
4000	3583	4.14	191.92	2.06	0.9315	0.7649
4500	4019	3.69	192.60	3.28	0.9368	0.7924
5000	4448	3.33	188.66	2.81	0.9410	0.7878

Discussion

1. **Training Time:**
   Training time increases with the size of the subword vocabulary. For smaller vocabulary sizes (e.g., 472), the training completes quickly (46.58 seconds), while for the larger vocabulary sizes (e.g., 4448), the training time exceeds 3 minutes (188.66 seconds). This is expected, as larger vocabularies result in larger embedding layers that take longer to process.

2. **Evaluation Time:**
   Evaluation time also increases as the subword vocabulary size grows. This is partly due to the overhead introduced by the larger embedding size, which needs to be processed during evaluation. A large vocabulary size (4019) saw the longest evaluation time at 3.28 seconds, but interestingly, there was a slight reduction at the 4448 vocabulary size. This is not consistent with the rest of the data and the outlier can be attributed to random chance or some constraints resulting from other processes consuming CPU resources. Since, both training and evaluation times should grow with vocabulary size.

3. **Accuracy Trends:**

   - **Train Accuracy:** Peak train accuracy improves as the vocabulary size increases, plateauing around the higher vocabulary sizes (e.g., 4448). This indicates that the model is able to capture more fine-grained features with a larger subword vocabulary, leading to improved training performance.
   - **Test Accuracy:** Test accuracy generally improves with increasing vocabulary size, reaching a maximum of 0.7924 at a vocabulary size of 4019. However, further increases to the vocabulary size (e.g., 4448) lead to a slight decline (0.7878), which may indicate overfitting as the model becomes too specialized to the training data.

Conclusion

This experiment reveals a clear trade-off between vocabulary size and performance. Smaller vocabularies lead to faster training and evaluation times, but they cap the model's ability to learn and generalize effectively. Larger vocabularies enhance the model's accuracy, particularly on the test set, but at the cost of increased computational expense. Based on this experiment, a vocabulary size between 2268 and 4019 seems to offer the best balance between performance and computational cost, with the peak test accuracy being achieved at a vocabulary size of 4019.

PART 3

Q1)

In the skip-gram model, given a center word, the objective is to predict the surrounding context words within a window size kk. Here, k = 1k=1, meaning the model considers one word to the left and one word to the right as context words.

Sentence	Center Word	Context Word	Training Pair
“the dog”	the	dog	(the, dog)
“the dog”	dog	the	(dog, the)
“the cat”	the	cat	(the, cat)
“the cat”	cat	the	(cat, the)
“a dog”	a	dog	(a, dog)
“a dog”	dog	a	(dog, a)

(3a) Optimal Probabilities for the Training Set

Given a dataset D consisting of word-context pairs (x, y), where $x$is the center word and y is the context word, the goal is to maximize the likelihood of the observed data under the skip-gram model. The likelihood $L(\theta)$ of the observed data D under the model is the product of the probabilities of all word-context pairs in the dataset. If we have N word-context pairs, this is expressed as:

$$L(\theta) = \prod_{(x,y)\in D} P(y \mid x; \theta)$$

Where:

- $P(y \mid x; \theta)$ is the probability of the context word y given the center word x, parameterized by θ (the model parameters, which include the word and context embeddings).
- D is the set of observed word-context pairs in the training data.

To simplify the optimization process, we usually work with the **log-likelihood** instead of the likelihood itself. Taking the logarithm of the likelihood function turns the product into a sum:

$$\log L(\theta) = \sum_{(x,y)\in D} \log P(y \mid x; \theta)$$

The objective is to find the parameters θ that maximize this log-likelihood. We know that the likelihood is maximized by having the model match the empirical distribution of outcomes observed in the data.

Let $P_{\text{emp}}(y \mid x)$ be the empirical probability of seeing context word y given center word x. This empirical distribution is based on the relative frequency of observing each pair (x, y) in the data:

$$P_{\text{emp}}(y \mid x) = \frac{\text{Count}(x, y)}{\text{Count}(x)}$$

Where:

- $\text{Count}(x, y)$ is the number of times the pair (x, y) occurs in the data.
- $\text{Count}(x)$ is the number of times the center word x appears in the data.

The training data includes two context words for "the" ("dog" and "cat"). Therefore,

$$P_{\text{emp}}(dog \mid the) = \frac{\text{Count}(the, dog)}{\text{Count}(the)} = \frac{1}{2}$$

$$P_{\text{emp}}(cat \mid the) = \frac{\text{Count}(the, cat)}{\text{Count}(the)} = \frac{1}{2}$$

(3b) Nearly Optimal Word Vector for "the"

We have,

$$P(y \mid x) = \frac{e^{v_x \cdot c_y}}{\sum_{y'} e^{v_x \cdot c_{y'}}}$$

We want to calculate $\mathbf{v}_{the}$, such that,

$$P(y \mid the) = \frac{e^{v_{the} \cdot c_y}}{\sum_{y'} e^{v_{the} \cdot c_{y'}}}$$

$$P(y \mid the) = \frac{e^{\mathbf{v}_{the} \cdot \mathbf{c}_y}}{e^{\mathbf{v}_{the} \cdot \mathbf{c}_{\text{cat}}} + e^{\mathbf{v}_{the} \cdot \mathbf{c}_{\text{dog}}}}$$

To find a nearly optimal word vector for "the," we need to achieve probabilities close to $P(\text{dog} \mid \text{the}) = \frac{1}{2}$ and $P(\text{cat} \mid \text{the}) = \frac{1}{2}$ using the softmax function.

$$P(dog \mid the) = \frac{e^{\mathbf{v}_{the} \cdot \mathbf{c}_{dog}}}{e^{\mathbf{v}_{the} \cdot \mathbf{c}_{\text{cat}}} + e^{\mathbf{v}_{the} \cdot \mathbf{c}_{\text{dog}}}}$$

$$\frac{1}{2} = \frac{e^{\mathbf{v}_{the} \cdot \mathbf{c}_{dog}}}{e^{\mathbf{v}_{the} \cdot \mathbf{c}_{\text{cat}}} + e^{\mathbf{v}_{the} \cdot \mathbf{c}_{\text{dog}}}}$$

Solving this equation, we get,

$$e^{\mathbf{v}_{the} \cdot \mathbf{c}_{\text{cat}}} = e^{\mathbf{v}_{the} \cdot \mathbf{c}_{dog}}$$

i.e., a trivial solution that satisfies the above equation can be obtained by simply setting the dot products to zero.

$$\mathbf{v}_{the} \cdot \mathbf{c}_{dog} \approx \mathbf{v}_{the} \cdot \mathbf{c}_{cat} \approx 0$$

We know, the context embedding vectors for "dog" and "cat" are:
$\mathbf{c}_{\text{dog}} = \mathbf{c}_{\text{cat}} = (0, 1)$

Let, $\mathbf{v}_{\text{the}} = (a, b)$

$\mathbf{v}_{the} \cdot \mathbf{c}_{dog} = (a, b) \cdot (0, 1)^T = 0$

i.e., any vector of the form, $\mathbf{v}_{the} = (a, 0)$, where aa is a real number, could be an optimal vector for the word "the" in our dataset. Therefore, we choose $\mathbf{v}_{the} = (1, 0)$
However, in practice, the value of the vectors would be randomly initialized, and the optimization algorithm will converge the weights of $\mathbf{v}_{the}$ to a value that approximately takes the form $(weight_{random}, 0)$

(3c) Training Examples Derived from These Sentences

Sentence	Center Word	Context Word	Training Pair
"the dog"	the	dog	(the, dog)
"the dog"	dog	the	(dog, the)
"the cat"	the	cat	(the, cat)

Sentence	Center Word	Context Word	Training Pair
"the cat"	cat	the	(cat, the)
"a dog"	a	dog	(a, dog)
"a dog"	dog	a	(dog, a)
"a cat"	a	cat	(a, cat)
"a cat"	cat	a	(cat, a)

3d: A Set of Both Word and Context Vectors that Nearly Optimizes the Skip-Gram Objective

Let's calculate the probabilities $P(y \mid x)$ using the formula:

$$P(y \mid x) = \frac{\text{count}(x, y)}{\text{count}(x)}$$

Where:

- $\text{count}(x, y)$ is the number of times y appears as a context word for x
- $\text{count}(x)$ is the total number of times x appears as a center word across all training pairs.

Table of Probabilities:

Probability P(y|x)	Count(x, y)	Count(x)	Probability Value
P(dog|the)	1	2	0.5
P(cat|the)	1	2	0.5
P(the|dog)	1	2	0.5
P(a|dog)	1	2	0.5
P(the|cat)	1	2	0.5
P(a|cat)	1	2	0.5
P(dog|a)	1	2	0.5
P(cat|a)	1	2	0.5

Assume some trivial vector representations for center words,
$\mathbf{v}_{\text{the}} = (0, 1)$
$\mathbf{v}_{\text{dog}} = (1, 0)$
$\mathbf{v}_{\text{cat}} = (1, 0)$
$\mathbf{v}_{\text{a}} = (0, 1)$

Similar to the solution from (3b), we can calculate for $P(y \mid the)$,

$$\frac{1}{2} = \frac{e^{\mathbf{v}_{the} \cdot \mathbf{c}_{dog}}}{e^{\mathbf{v}_{the} \cdot \mathbf{c}_{\text{cat}}} + e^{\mathbf{v}_{the} \cdot \mathbf{c}_{\text{dog}}}}$$

We get,

$$e^{\mathbf{v}_{the} \cdot \mathbf{c}_{\text{cat}}} = e^{\mathbf{v}_{the} \cdot \mathbf{c}_{dog}}$$

We get,

$$\mathbf{v}_{the} \cdot \mathbf{c}_{cat} \approx \mathbf{v}_{the} \cdot \mathbf{c}_{dog} \approx 0$$

Repeating these steps, for $P(y \mid a)$, $P(y \mid cat)$, and $P(y \mid dog)$ we get,

$\mathbf{v}_{cat} \cdot \mathbf{c}_{the} = 0$ and $\mathbf{v}_{cat} \cdot \mathbf{c}_{a} = 0$
and,

$\mathbf{v}_{dog} \cdot \mathbf{c}_{the} = 0$ and $\mathbf{v}_{dog} \cdot \mathbf{c}_{a} = 0$
and,
$\mathbf{v}_{the} \cdot \mathbf{c}_{dog} = 0$ and $\mathbf{v}_{the} \cdot \mathbf{c}_{cat} = 0$
and,
$\mathbf{v}_{a} \cdot \mathbf{c}_{dog} = 0$ and $\mathbf{v}_{a} \cdot \mathbf{c}_{cat} = 0$

Since, $\mathbf{v}_{cat} = (1, 0)$ and $\mathbf{v}_{dog} = (1, 0)$,
and
$\mathbf{v}_{the} = (0, 1)$ and $\mathbf{v}_{a} = (0, 1)$

The solutions that satisfy the above equations give us the following final word vectors:

- $\mathbf{v}_{\text{the}} = (0, 1)$
- $\mathbf{v}_{\text{dog}} = (1, 0)$
- $\mathbf{v}_{\text{cat}} = (1, 0)$
- $\mathbf{v}_{\text{a}} = (0, 1)$

and context vectors,

- $\mathbf{c}_{\text{the}} = (0, 1)$
- $\mathbf{c}_{\text{dog}} = (1, 0)$
- $\mathbf{c}_{\text{cat}} = (1, 0)$
- $\mathbf{c}_{\text{a}} = (0, 1)$

References

1. The code uses supplementary Python functions and classes supplied in the PA1 zip folder.
2. The code uses the BPE Algorithm referenced in the lecture slides.
3. The (LaTeX) formatting for this report was prepared with the help of ChatGPT.
4. The report content was revised for better grammar and sentence structure using Grammarly.