Report for PA1

Part1

A specific implementation of DAN model includes an embedding layer, two linear layer and a softmax layer to output the result.

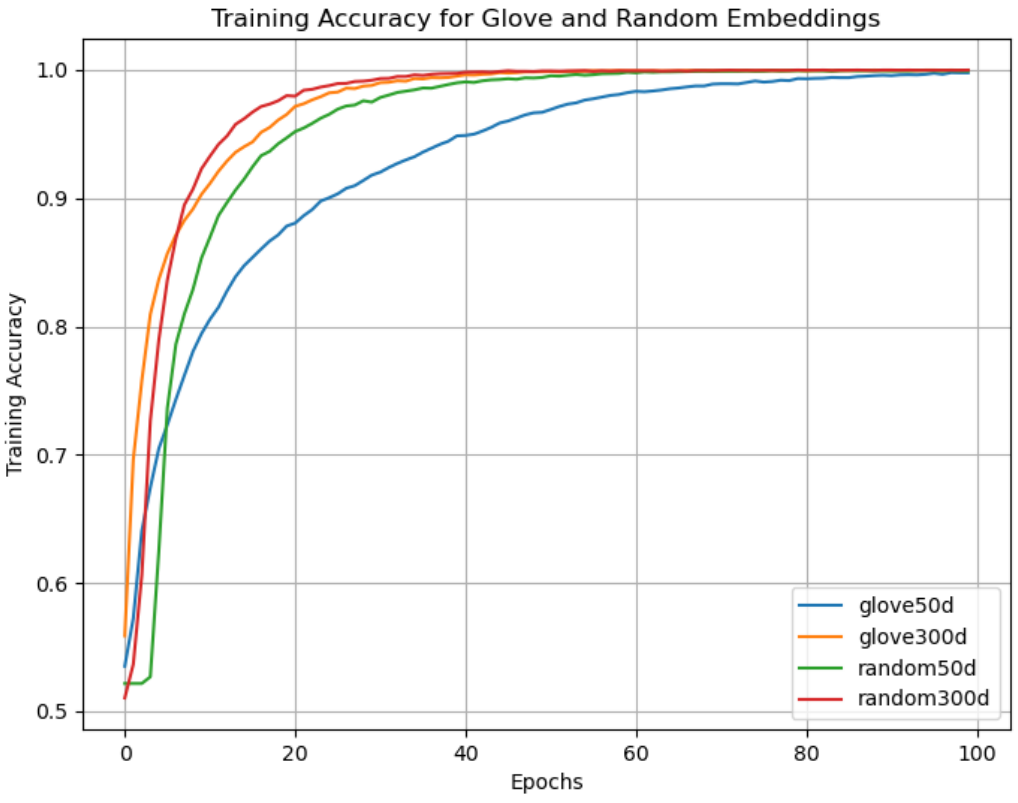
For part 1, the hyperparameters are as such:

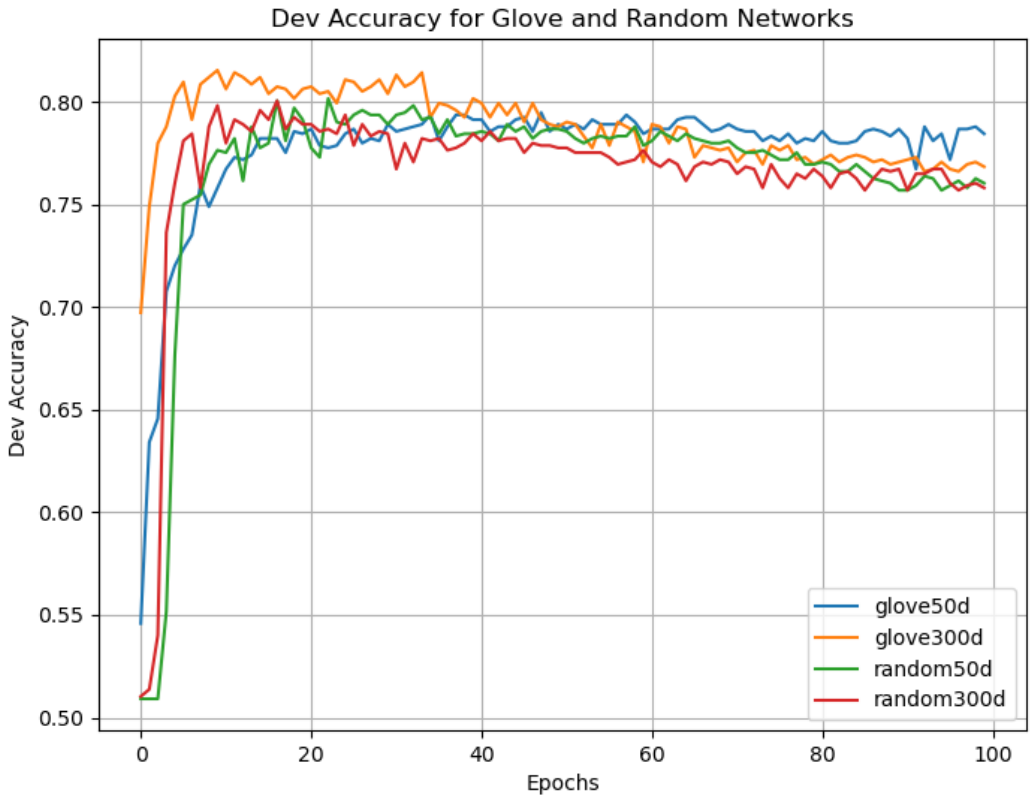
```
loss Function: NLLLoss
Optimizer: Adam
learning rate: 0.0001
hidden size: 100
```

The reason why I choose such a configuration is that it is the same as the BOW model does. In order to compare the performance of DAN model and BOW model, such hyperparameters should remain the same.

To explore the impact on model performance of adopting different embedding layers, I trained the model using four different embedding layer separately, which are glove.6B.50d, glove.6B.300d, random.50d, random.300d. The first two embeddings are loaded from the pretrained word embedding file from Glove, with embedding dimension of 50 and 300 separately. The last two embeddings are randomly initialized using a normal distribution.

The training and testing accuracy of four different layers are as below:





Embedding Name	Best Dev Accuracy
glove.50d	0.791
glove.300d	0.815
random.50d	0.791
random.300d	0.798

Training Stage

From the training figure, we can see that the training accuracy of the four models gradually converged to 1. The GloVe 300d model achieves the best performance on the training set, converging to nearly 100% accuracy the fastest. And the Random 50d model shows the slowest convergence and the lowest accuracy among all models.

We can conclude that

- Pre-trained embeddings (GloVe) provide a significant advantage during training, enabling faster and better convergence.
- Higher-dimensional embeddings (300D) improve performance compared to lower-dimensional ones, whether using GloVe or random initialization.
- Randomly initialized embeddings eventually reach high accuracy but require more epochs and time to converge.

Development Stage

From the development figure, we can see that GloVe 300d model performs well initially but exhibits some fluctuation in later epochs. Its accuracy remains around 80%. GloVe 50d model has the most stable performance on the dev set, with accuracy oscillating near 80% throughout the epochs. Random 300d converges quickly in the initial epochs, its accuracy fluctuates and tends to degrade slightly over time. Random 50d is similar to the random 300d model, this model shows more instability in performance compared to the GloVe models.

We can conclude that

- Pre-trained GloVe embeddings provide a more stable performance on the development set compared to random initialization.
- Higher-dimensional embeddings (300D) give a performance boost initially but might not generalize as well, leading to fluctuations in the later epochs.
- Random embeddings struggle with stability, especially on the dev set, indicating potential overfitting or difficulty in generalization.

Conclusion

Convergence Speed

- GloVe embeddings significantly speed up convergence during training compared to random initialization.
- Higher-dimensional embeddings (300D) improve convergence speed regardless of initialization.

Generalization

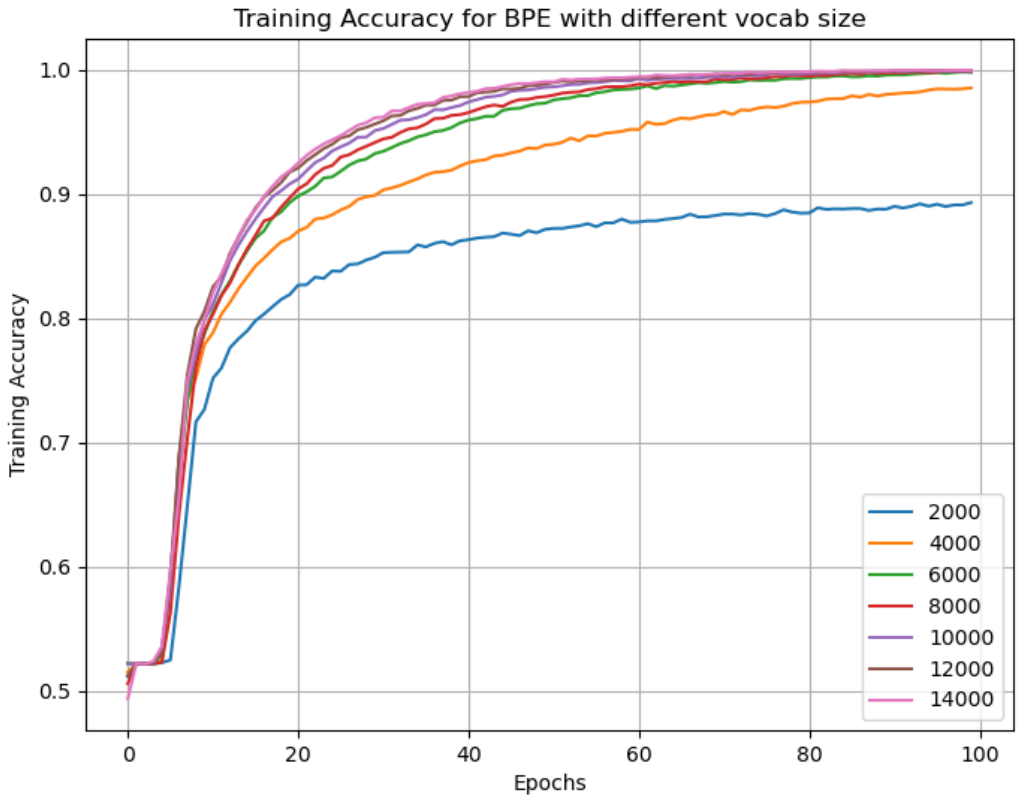
- While both GloVe and random embeddings achieve reasonable performance, GloVe embeddings offer more stability on the development set.
- Randomly initialized embeddings, though capable of achieving high accuracy on the training set, show more variability on the dev set, possibly due to overfitting.

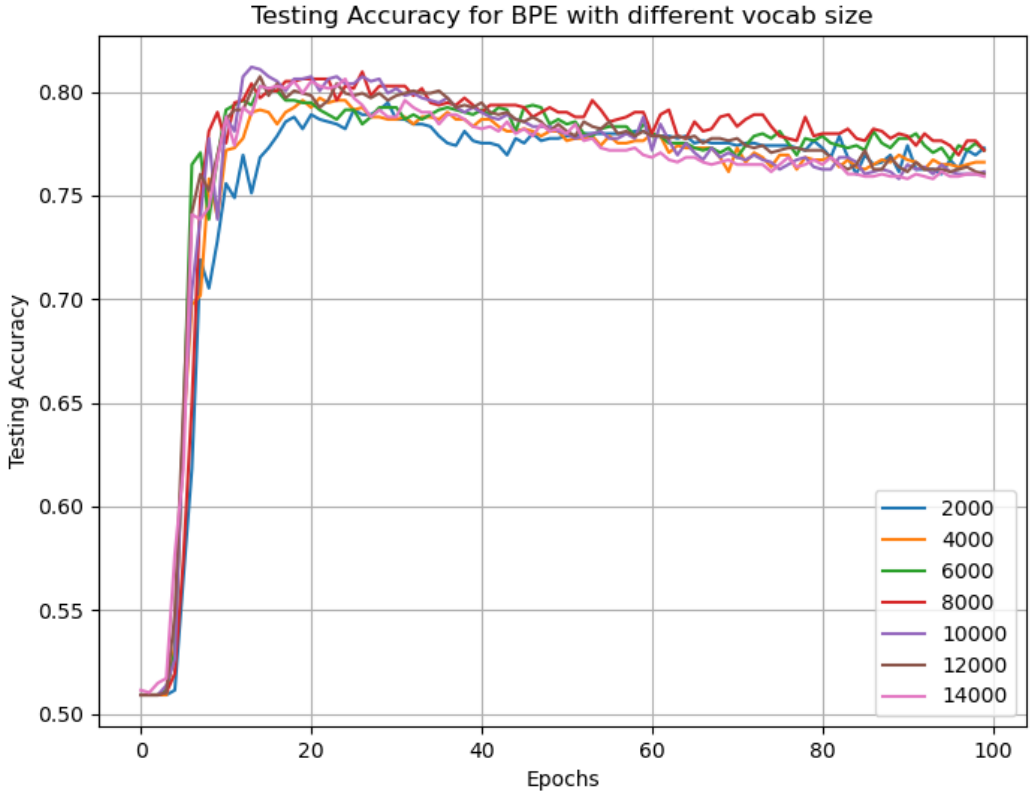
Impact of Embedding Dimension

- 300D embeddings tend to improve performance but may introduce some instability on unseen data.
- 50D embeddings show more consistent performance on the development set, particularly with GloVe.

Part2

For BPE tokenizer, I directly used the tokenizer library provided by HuggingFace. To explore the effect of using BPE tokenizers of different word sizes on the results, I set the vocab size to be 2000, 4000, 6000, 8000, 10000, 12000, 14000. Also, the dimension of the embedding layer is 50.





Vocab size	Best Dev Accuracy
2000	0.795
4000	0.796

Vocab size	Best Dev Accuracy
6000	0.795
8000	0.806
10000	0.806
12000	0.799
14000	0.799

Training Stage

From the training accuracy figure, we can observe that the models with larger vocabularies, particularly those with 8000 or more tokens, converge to nearly 100% accuracy on the training set. The models with vocab sizes between 10000 and 14000 achieve the fastest convergence and the highest training accuracy, while the 2000-token model shows the slowest convergence and the lowest final accuracy.

We can conclude that:

- Larger vocabularies enable models to capture more information, leading to faster and better convergence on the training set.
- Models with smaller vocabularies (2000 tokens) require more epochs to converge and fail to reach the same level of accuracy as larger-vocabulary models, indicating reduced expressiveness.
- Vocabulary sizes beyond 10000 yield marginal improvements, suggesting a diminishing return on performance beyond a certain point.
- Smaller vocabularies perform reasonably well but trade off some training accuracy for simpler, faster models with fewer parameters.

Development Stage

From the testing accuracy figure, we notice that all models perform similarly on the test set, with accuracies ranging between 0.75 and 0.80 after convergence. However, models with smaller vocab sizes such as 4000 and 6000 tokens achieve relatively stable performance throughout the epochs, while larger vocab models show slight drops over time, likely due to overfitting. Also, compared to word-level tokenizer with same dimension, BPE do improve the accuracy.

We can conclude that:

- Smaller vocabularies (4000-6000) provide more stable generalization, showing comparable performance to larger vocab models on the test set with less risk of overfitting.
- Larger vocabularies (e.g., 10000 or more) may overfit the training data, leading to slight performance degradation during testing.
- Selecting a balanced vocabulary size between 6000 and 10000 achieves a good trade-off between training accuracy and generalization ability on unseen data.
- Using BPE as the tokenizer can achieve better dev accuracy than using word-level tokenizer.

Conclusion

Convergence Speed

- Larger vocabulary sizes (e.g., 8000 and above) significantly improve convergence speed, achieving nearly 100% accuracy faster during training.
- Smaller vocabularies (e.g., 2000 tokens) require more epochs to converge and fail to reach the same level of training accuracy as larger vocab sizes.

Generalization

- Moderate vocabularies (between 6000 and 10000 tokens) provide the best trade-off between training performance and stability on the test set.
- Larger vocabularies (above 10000 tokens) may suffer from overfitting, showing slightly lower and less stable test accuracy.
- Smaller vocabularies (e.g., 2000 tokens) generalize well but have a slower convergence rate and lower training accuracy.

Impact of Vocabulary Size

- Larger vocabularies (e.g., 10000+ tokens) improve training performance but may introduce instability on unseen data due to overfitting.
- Smaller vocabularies (e.g., 4000-6000 tokens) show more consistent performance on the test set, offering better generalization at the cost of slightly slower convergence.

Impact of BPE Tokenizer

- BPE tokenizer can help model of same dimension achieve better dev accuracy than word-level tokenizer.

Part3

Q1: Training with Specific Sentences

Consider the sentences:

1. `"the dog"`
2. `"the cat"`
3. `"a dog"`

With a window size of (k = 1), the training examples are:

- From `"the dog"`:

$$(x = \text{"the"}, y = \text{"dog"}) and (x = \text{"dog"}, y = \text{"the"})$$

.

- From `"the cat"`:

$$(x = \text{"the"}, y = \text{"cat"}) and (x = \text{"cat"}, y = \text{"the"})$$

- From `"a dog"`:

$$(x = \text{”a”}, y = \text{”dog”}) and (x = \text{”dog”}, y = \text{”a”}).$$

The Skip-Gram objective is to maximize the log-likelihood of this training data:

$$\sum_{(x,y)} \log P(y \mid x)$$

3a) Finding the Optimal Probabilities

We need the set of probabilities that maximize the likelihood.

- Observed pairs:
  - "the" occurs with "dog" and "cat".

Since each occurs once with `"the"`, the optimal empirical probabilities are:

$$P(\text{”dog”} \mid \text{”the”}) = 0.5, \quad P(\text{”cat”} \mid \text{”the”}) = 0.5$$

This follows the principle of maximum likelihood estimation (similar to estimating probabilities from biased coin flips).

3b) Achieving These Probabilities with Vectors

To achieve probabilities of 0.5 for both "dog" and "cat", the dot products between the word vector for `"the"` and the context vectors for `"dog"` and `"cat"` must be equal.

Assume:

$$\mathbf{c}_{\text{dog}} = (0, 1)$$

$$\mathbf{c}_{\text{cat}} = (0, 1)$$

A suitable vector for `"the"` would be:

$$\mathbf{v}_{\text{the}} = (k, 1)$$

With this setup, the dot products are:

$$\mathbf{v}_{\text{the}} \cdot \mathbf{c}_{\text{dog}} = 1, \quad \mathbf{v}_{\text{the}} \cdot \mathbf{c}_{\text{cat}} = 1$$

Thus, the Softmax will output approximately equal probabilities for both `"dog"` and `"cat"`, matching the empirical distribution.

Q2: Additional Sentences

Consider the additional sentences:

1. `"the dog"`
2. `"the cat"`
3. `"a dog"`
4. `"a cat"`

3c) Training Examples

With (d = 2), the training examples are:

$$(x = \text{"the"}, y = \text{"dog"})$$

$$(x = \text{"the"}, y = \text{"cat"})$$

$$(x = \text{"a"}, y = \text{"dog"})$$

$$(x = \text{"a"}, y = \text{"cat"})$$

3d) Optimizing Word and Context Vectors

Assign the following vectors to optimize the objective:

$$\mathbf{v}_{\text{the}} = (1, 0)$$

$$\mathbf{v}_{\text{a}} = (1, 0)$$

$$\mathbf{c}_{\text{dog}} = (0, 1)$$

$$\mathbf{c}_{\text{cat}} = (0, 1)$$

The dot products become:

$$\mathbf{v}_{\text{the}} \cdot \mathbf{c}_{\text{dog}} = 0, \quad \mathbf{v}_{\text{the}} \cdot \mathbf{c}_{\text{cat}} = 0$$

The Softmax function will produce equal probabilities for all context words, aligning with the empirical distribution.