CSE 256 - Fall 2024 - PA1

Due: *Friday, Oct 18*
Answered by: Mathias Otnes

1 Deep Averaging Networks (DAN)

1.1 a) DAN implementation

In this assignment, I implemented the **Deep Averaging Network (DAN)** for sentiment classification, as outlined in the lecture. The task involves classifying movie reviews as positive or negative based on the Rotten Tomatoes dataset.

1.1.1 Model Architecture

Embedding Layer: This layer transforms the input list of input indicies into embeddings. By transforming the indicies to the glove embeddings or a randomly initialized embedding layer.

Averaging: For each sentence, the embeddings of the words are averaged to produce a single fixed-size vector.

Feedforward Layers: The averaged embeddings are passed through multiple fully connected (dense) layers. I experienced with different dimentions, but ended up with the set of dimentions; [100, 500, 2].

Regularization: To prevent overfitting, I added dropout layers between the fully connected layers. I used a dropout rate of 50%.

Loss Function: I used cross entropy as my loss function.

1.1.2 Results

For task **1a**, the goal was to implement the DAN and achieve at least 77% accuracy on the development set while keeping training time under five minutes. Using the glove embeddings I got the results presented in figure [1]. The training only took about 30 seconds on my computer, but it would be sufficient to stop after 5 epochs as the model started to overfit after this point. The perfomance was really good with minimal training, which indicates that the GloVe embeddings are good at catching the semantics of the words.

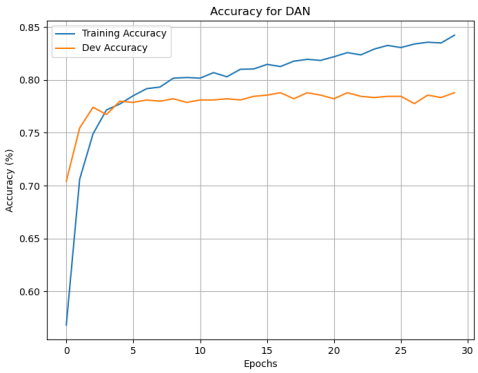

Figure 1: Train and dev accuracy during training using GloVe embeddings.

For task **1b**, I used a randomly initialized set og embeddings and trained them along with the rest of the network. The accuracy during training is visualized in figure [2] With these embeddings, the training took 10 epochs instead of 5 epochs before it reached its peak on development accuracy. It also started to overfit significantly compared to the fixed GloVe embeddings, which indicates that the sentiment labels are getting memorized directly into the embeddings. It also never reached the same peak development accuracy as with the GloVe embeddings which is most

likelly due to the embeddings being overfitted for the dataset. This could maybe be improved by implementing strict regularizations on the embeddings layer. It used almost twice the time to train compared to the GloVe model per epoch because of the increased number of parameters.

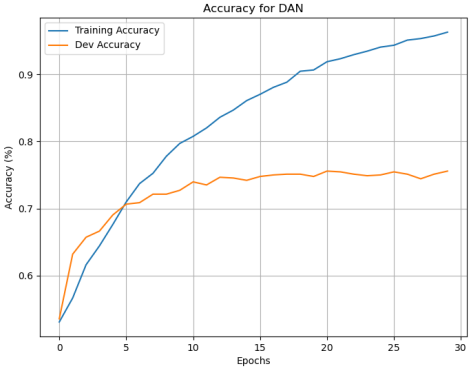

Figure 2: Train and dev accuracy during training using randomly initialized embeddings.

2 Byte-Pair Encoding (BPE)

I implemented BPE as suggested in the algorithm presented in [1]. The detailed implementation can be viewed in the code along with comments and explanations. I created a tokenizer in which used BPE on the training data to create a tokenizer function. This tokenizer were passed into the dataloaders so that the model had access to the tokenized version of the examples.

I experimented with different vocabulary sizes. Following is the results for the most interesting vocabulary sizes:

100 tokens: This is slightly more than the 65 unique ASCII characters in the dataset, meaning the BPE algorithm performed 35 merges. As a result, the vocabulary consists mainly of 1-2 character tokens. The DAN, therefore, averages over letter frequencies, which limits its ability to capture the complex sentiments that humans typically understand by recognizing entire words or phrases. Despite this, it achieved an accuracy of 65%, which is decent. However, the model is likely memorizing word frequencies instead of learning the underlying sentiment patterns.

4096 tokens: A 4096-token BPE vocabulary balances word-level tokenization with subword representation. This larger size captures more complete words, which is crucial for sentiment analysis, while subword tokens improve generalization over rare words. This approach minimizes over-segmentation while retaining flexibility for less frequent terms, enhancing sentiment classification. It ended up converging at a 75% accuracy on the development set, which is very similar to what I acheived with the randomized embeddings.

12986 tokens: In this vocabulary, almost every word in the dataset was assigned its own token. This makes the tokenization quite similar to a randomized embedding model over the 14,923 unique words, with the difference that not all unique words are included and subwords are also represented. When compared to randomized embeddings, it performed slightly better, likely because the subword tokens allow the model to generalize better across unseen or rare words. This gives the model an advantage in handling words not fully present in the training data, while still benefiting from tokenizing frequent words efficiently. The results are visualized in figure [3].

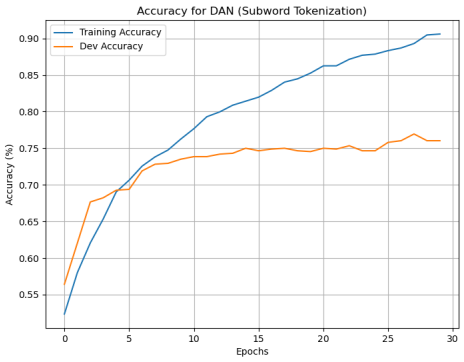

Figure 3: Train and dev accuracy during training using BPE tokenization and randomized embeddings.

3 Understanding Skip-Gram

Q1 - 3a

Using the training data we get the following pairs of skipgrams:

- $(x = \text{the}, y = \text{dog})$
- $(x = \text{dog}, y = \text{the})$
- $(x = \text{the}, y = \text{cat})$
- $(x = \text{cat}, y = \text{the})$
- $(x = \text{a}, y = \text{dog})$
- $(x = \text{dog}, y = \text{a})$

To find the conditional probability that maximizes the data likelyhood, we can use the aproach suggested in the hint of choosing the model that matches the empirical distribution of the data. Given that $y = \text{the}$, we have the remaining skip-grams:

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

In which it's easy to see that the probability that maximizes this data is $P(y = \text{dog}|x = \text{the}) = 0.5$ and $P(y = \text{cat}|x = \text{the}) = 0.5$.

Q1 - 3b

For this task, we are tasked with finding a vector $\boldsymbol{v}_{\text{the}}$ such that it gives nearly optimal probabilities for $P(\text{dog}|\text{the})$ and $P(\text{cat}|\text{the})$, both of which should be close to 0.5. The skip-gram equation is given by:

$$P(y|x) = \frac{\exp(\boldsymbol{v_x} \cdot \boldsymbol{c_y})}{\sum_{y'} \exp(\boldsymbol{v_x} \cdot \boldsymbol{c_{y'}})}$$

Where:

$\boldsymbol{v_x}$ is the vector for the center word x (here, "the"), and $\boldsymbol{c_y}$ is the vector for the context word y (here, "dog", "cat", "the", "a").

From the problem statement, the context vectors are:

$$\boldsymbol{c}_{\text{dog}} = \boldsymbol{c}_{\text{cat}} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad \boldsymbol{c}_{\text{the}} = \boldsymbol{c}_{\text{a}} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

We are asked to find $\boldsymbol{v}_{\text{the}}$ such that:

$$P(\text{dog}|\text{the}) \approx P(\text{cat}|\text{the}) \approx 0.5$$

Because the embeddings for "dog" and "cat" are identical, we will have $P(\text{dog}|\text{the}) = P(\text{cat}|\text{the})$ for any $\boldsymbol{v}_{\text{the}}$. To ensure these probabilities are as close to 0.5 as possible, we need $P(\text{the}|\text{the})$ and $P(\text{a}|\text{the})$ to be as close to zero as possible.

Let us define $\boldsymbol{v}_{\text{the}}$ as:

$$\boldsymbol{v}_{\text{the}} = \begin{pmatrix} \boldsymbol{v_1} \\ \boldsymbol{v_2} \end{pmatrix}$$

We compute the dot products:

$$\begin{aligned} \boldsymbol{v}_{\text{the}} \cdot \boldsymbol{c}_{\text{dog}} &= \boldsymbol{v_1} \cdot 0 + \boldsymbol{v_2} \cdot 1 = \boldsymbol{v_2} \\ \boldsymbol{v}_{\text{the}} \cdot \boldsymbol{c}_{\text{cat}} &= \boldsymbol{v_1} \cdot 0 + \boldsymbol{v_2} \cdot 1 = \boldsymbol{v_2} \\ \boldsymbol{v}_{\text{the}} \cdot \boldsymbol{c}_{\text{the}} &= \boldsymbol{v_1} \cdot 1 + \boldsymbol{v_2} \cdot 0 = \boldsymbol{v_1} \\ \boldsymbol{v}_{\text{the}} \cdot \boldsymbol{c}_{\text{a}} &= \boldsymbol{v_1} \cdot 1 + \boldsymbol{v_2} \cdot 0 = \boldsymbol{v_1} \end{aligned}$$

The exponentials are:

$$\begin{aligned} \exp(\boldsymbol{v}_{\text{the}} \cdot \boldsymbol{c}_{\text{dog}}) &= \exp(\boldsymbol{v_2}) \\ \exp(\boldsymbol{v}_{\text{the}} \cdot \boldsymbol{c}_{\text{cat}}) &= \exp(\boldsymbol{v_2}) \\ \exp(\boldsymbol{v}_{\text{the}} \cdot \boldsymbol{c}_{\text{the}}) &= \exp(\boldsymbol{v_1}) \\ \exp(\boldsymbol{v}_{\text{the}} \cdot \boldsymbol{c}_{\text{a}}) &= \exp(\boldsymbol{v_1}) \end{aligned}$$

The normalization constant is:

$$\sum_{y'} \exp(\boldsymbol{v}_{\text{the}} \cdot \boldsymbol{c_{y'}}) = 2\exp(\boldsymbol{v_1}) + 2\exp(\boldsymbol{v_2})$$

Therefore, the probabilities are:

$$P(\text{dog}|\text{the}) = \frac{\exp(\boldsymbol{v_2})}{2\exp(\boldsymbol{v_1}) + 2\exp(\boldsymbol{v_2})}$$
$$P(\text{cat}|\text{the}) = \frac{\exp(\boldsymbol{v_2})}{2\exp(\boldsymbol{v_1}) + 2\exp(\boldsymbol{v_2})}$$
$$P(\text{the}|\text{the}) = \frac{\exp(\boldsymbol{v_1})}{2\exp(\boldsymbol{v_1}) + 2\exp(\boldsymbol{v_2})}$$
$$P(\text{a}|\text{the}) = \frac{\exp(\boldsymbol{v_1})}{2\exp(\boldsymbol{v_1}) + 2\exp(\boldsymbol{v_2})}$$

To achieve $P(\text{dog}|\text{the}) \approx 0.5$ and $P(\text{the}|\text{the}) \approx 0$, we need $\exp(\boldsymbol{v_2}) \gg \exp(\boldsymbol{v_1})$. We can accomplish this by setting $\boldsymbol{v_2} = 0$ and choosing a large negative value for $\boldsymbol{v_1}$. Let's choose $\boldsymbol{v_1} = -10$:

$$\boldsymbol{v}_{\text{the}} = \begin{pmatrix} -10 \\ 0 \end{pmatrix}$$

Now, compute the exponentials:

$$\exp(\boldsymbol{v_1}) = \exp(-10) \approx 4.539 \times 10^{-5}$$
$$\exp(\boldsymbol{v_2}) = \exp(0) = 1$$

The normalization constant becomes:

$$2\exp(-10) + 2\exp(0) \approx 0 + 2 = 2$$

Thus, the probabilities are:

$$P(\text{dog}|\text{the}) \approx \frac{1}{2}$$
$$P(\text{cat}|\text{the}) \approx \frac{1}{2}$$
$$P(\text{the}|\text{the}) \approx 2.269 \times 10^{-5}$$
$$P(\text{a}|\text{the}) \approx 2.269 \times 10^{-5}$$

These probabilities satisfy our requirements of having a margin < 0.01 from the desired probability.

Q2 - 3c

With a window size $k = 1$, the skip-gram model considers the words immediately before and after the target word as context. For the given sentences, we extract the (center word, context word) pairs as follows:

"the dog" - (the, dog), (dog, the)
"the cat" - (the, cat), (cat, the)
"a dog" - (a, dog), (dog, a)
"a cat" - (a, cat), (cat, a)

Which is all our pairs for our training data.

Q2 - 3d

We aim to find word vectors $\boldsymbol{v_x}$ and context vectors $\boldsymbol{c_y}$ such that the model's predicted probabilities $P(y|x)$ match the empirical probabilities from the training data.

Empirical Probabilities:

From the training examples, the empirical conditional probabilities are:

- For center words "the" and "a":

$$P(\text{dog}|x) = P(\text{cat}|x) = \frac{1}{2}, \quad x \in \{\text{the}, \text{a}\}$$

- For center words "dog" and "cat":

$$P(\text{the}|x) = P(\text{a}|x) = \frac{1}{2}, \quad x \in \{\text{dog}, \text{cat}\}$$

We can reckognize that this is similar to problem **3b)**, on which we've already found a solution to the word vector $\boldsymbol{v}_{\text{the}}$. Because our empirical data and context vectors are identical for $\boldsymbol{v}_{\text{the}}$ and $\boldsymbol{v}_{\text{a}}$, and similarly with $\boldsymbol{v}_{\text{dog}}$ and $\boldsymbol{v}_{\text{cat}}$, we can use the word vectors:

$$\boldsymbol{v}_{\text{the}} = \boldsymbol{v}_{\text{a}} = \begin{pmatrix} -10 \\ 0 \end{pmatrix}, \quad \boldsymbol{v}_{\text{dog}} = \boldsymbol{v}_{\text{cat}} = \begin{pmatrix} 0 \\ -10 \end{pmatrix}$$

Verification of Probabilities:

For center words $x \in \{\text{the}, \text{a}\}$ we have the word vector:

$$\boldsymbol{v_x} = \begin{pmatrix} -10 \\ 0 \end{pmatrix}$$

This will give us the following exponentials:

$$\boldsymbol{v_x} \cdot \boldsymbol{c_y} = \begin{cases} -10 \cdot 0 + 0 \cdot 1 = 0 & \text{if } y \in \{\text{dog}, \text{cat}\} \\ -10 \cdot 1 + 0 \cdot 0 = -10 & \text{if } y \in \{\text{the}, \text{a}\} \end{cases}$$

$$\exp(\boldsymbol{v_x} \cdot \boldsymbol{c_y}) = \begin{cases} \exp(0) = 1 & \text{if } y \in \{\text{dog}, \text{cat}\} \\ \exp(-10) \approx 0 & \text{if } y \in \{\text{the}, \text{a}\} \end{cases}$$

And the normalization constant will still be the same:

$$2 \times \exp(0) + 2 \times \exp(-10) \approx 2$$

This yields probabilities very close to the empirical data:

$$P(y|x) = \frac{\exp(\boldsymbol{v_x} \cdot \boldsymbol{c_y})}{2} = \begin{cases} \frac{1}{2} & \text{if } y \in \{\text{dog}, \text{cat}\} \\ \approx 0 & \text{if } y \in \{\text{the}, \text{a}\} \end{cases}$$

For center words $x \in \{\text{dog}, \text{cat}\}$ we have the word vector:

$$\boldsymbol{v_x} = \begin{pmatrix} 0 \\ -10 \end{pmatrix}$$

In which we calculated the probabilities in **3b)** to be

$$P(y|x) = \frac{\exp(\boldsymbol{v_x} \cdot \boldsymbol{c_y})}{2} = \begin{cases} \frac{1}{2} & \text{if } y \in \{\text{the}, \text{a}\} \\ \approx 0 & \text{if } y \in \{\text{dog}, \text{cat}\} \end{cases}$$

References

[1] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In Katrin Erk and Noah A. Smith, editors, *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics.