1 Word-Level Neural Bi-gram Language Model

1.a

Starting from our fundamental cross-entropy definition:

$$\mathrm{CE}(y, \hat{y}) = -\sum_i y_i \cdot \log(\hat{y}_i)$$

For a word-level model with softmax output, we can express $\hat{y}_i$ as:

$$\hat{y}_i = \mathrm{softmax}(\theta)_i = \frac{\exp(\theta_i)}{\sum_j \exp(\theta_j)}$$

Let's derive the gradient. For any output k (where y is the one-hot vector):

$$\frac{\partial \mathrm{CE}}{\partial \theta_k} = -\sum_i y_i \cdot \frac{\partial}{\partial \theta_k} \log(\hat{y}_i) \tag{1}$$

$$= -\sum_i y_i \cdot \frac{\partial}{\partial \theta_k} \log\left(\frac{\exp(\theta_i)}{\sum_j \exp(\theta_j)}\right) \tag{2}$$

$$= -\sum_i y_i \cdot (\delta_{ik} - \hat{y}_k) \tag{3}$$

$$= \hat{y}_k - y_k \tag{4}$$

Therefore, we can concisely express the gradient as:

$$\frac{\partial \mathrm{CE}}{\partial \theta} = \hat{y} - y$$

where y is the one-hot target vector and $\hat{y}$ is the predicted probability distribution.

1.b

Following the chain rule of differentiation, we can express the gradient as:

$$\frac{\partial J}{\partial x} = \frac{\partial J}{\partial (hW_2 + b_2)} \cdot \frac{\partial (hW_2 + b_2)}{\partial h} \cdot \frac{\partial \sigma}{\partial (xW_1 + b_1)} \cdot \frac{\partial (xW_1 + b_1)}{\partial x}$$

$$= (\hat{y} - y) \cdot W_2^\top \odot (h(1-h)) \cdot W_1^\top$$

This derivation shows the complete backward propagation of the gradient through the neural network layers, accounting for the weights (W_1, W_2), biases (b_1, b_2), and the activation function's derivative.

1.c

Code is attached.

1.d

After training the network for 40K iterations, we got 113.699 dev perplexity.

2 Generating Shakespeare Using a Character-level Language Model

2.a

Advantages of Character-Based Language Models:

Character-based models can handle any word, even unseen words or words that are not in the vocabulary. They have a smaller vocabulary compared to word-based models, which makes them more efficient in terms of memory. Additionally, they can capture subtleties like prefixes, suffixes, or spelling variations better than word-based models.

Advantages of Word-Based Language Models:

Word-based models are faster to train because they process fewer tokens for the same text length. They are also able to achieve better syntactic and semantic understanding, resulting in more coherent and meaningful text generation.

2.b Graph Plot

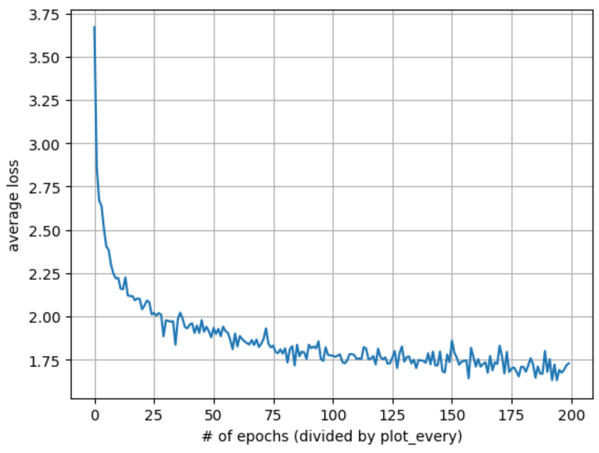

Figure 1: Average loss over epochs

3 Perplexity

3.a

$$
\begin{aligned}
2^{-\frac{1}{M}\sum_{i=1}^{M}\log_2 P(s_i|s_1,\ldots,s_{i-1})} &= \left(2^{\sum_{i=1}^{M}\log_2 P(s_i|s_1,\ldots,s_{i-1})}\right)^{-\frac{1}{M}} \\
&= \left(2^{\log_2 P(s_1)+\log_2 P(s_2|s_1)+\cdots+\log_2 P(s_M|s_1,\ldots,s_{M-1})}\right)^{-\frac{1}{M}} \\
&= \left(P(s_1)\cdot P(s_2 \mid s_1)\cdot\cdots\cdot P(s_M \mid s_1,\ldots,s_{M-1})\right)^{-\frac{1}{M}} \\
&= \left(e^{\ln P(s_1)+\ln P(s_2|s_1)+\cdots+\ln P(s_M|s_1,\ldots,s_{M-1})}\right)^{-\frac{1}{M}} \\
&= \left(e^{\sum_{i=1}^{M}\ln P(s_i|s_1,\ldots,s_{i-1})}\right)^{-\frac{1}{M}} \\
&= e^{-\frac{1}{M}\sum_{i=1}^{M}\ln P(s_i|s_1,\ldots,s_{i-1})}.
\end{aligned}
$$

3.b

Neural Bi-gram Language Model:

Shakespeare Perplexity: 7.122318650322853

Wikipedia Perplexity:25.75261330172207

Character-level Language Model:

Shakespeare Perplexity: 7.459401319332429

Wikipedia Perplexity: 19.97325117242642

3.c

For the Character-level Language Model we can assume those result are due to the fact that he learns at the charachter level and also was trained on shakespear data which can give the advantage to more similiar text who is also contains non standard spelling and word forms but on the other hand the result on wikipedia was worse to the more complex and structured word level patterns. For the Bi-gram Model we would assume it struggled with Wikipedia and got even a worse score than the character-level language model because it is limited to a 2 word context, where in Wikipedia text usually contains long sentences with dependencies that span multiple words. Our assumption that the score is even lower then the Character-level Language Model to the the later ability to capture meanings of suffixes, prefixes even in complex modern words while not relying on find word pairs. Also we think the the Bi-gram model performs better on shakespear because the word pairs and sentence structures in shakespeares text are relatively predictable and consistent .

3.d

Neural Bi-gram Language Model: (After preprocessing)

Shakespeare Perplexity: 4.475288916573775

Wikipedia Perplexity: 2.536595085380771

Character-level Language Model: (After preprocessing)

Shakespeare Perplexity: 6.740292861033496

Wikipedia Perplexity: 16.24853261546454

During preprocessing, we primarily focused on "cleaning" the data. This included converting all text to lowercase for better generalization, removing all non-printable characters (which are likely irrelevant to the context), and eliminating extra spaces.

4 Deep Averaging Networks

4.a

YOUR ANSWER HERE

4.b

YOUR ANSWER HERE

4.c

YOUR ANSWER HERE

4.d

YOUR ANSWER HERE

4.e

YOUR ANSWER HERE

5 Attention Exploration

5.a 1.a

5.a.i

α can be interpreted as categorial probability distribution because:

1. $\alpha_i > 0$ for each $i \in [n]$ due to the $\exp(x)$ function properties.

2.
$$\sum_{i=1}^{n} \alpha_i = \sum_{i=1}^{n} \frac{\exp(k_i^\top q)}{\sum_{j=1}^{n} \exp(k_j^\top q)} = \frac{\sum_{i=1}^{n} \exp(k_i^\top q)}{\sum_{j=1}^{n} \exp(k_j^\top q)} = 1.$$

since α hold those two conditions it can be interpreted as categorial probability distribution.

5.a.ii

α_j is dependent on k_j and q, so in order to achieve $\alpha_j \gg \sum_{i \neq j} \alpha_i$, we would want $k_j \gg k_i$ for each $i \neq j$ in $[n]$.

5.a.iii

Since $\alpha_j \gg \sum_{i \neq j} \alpha_i$, we can presume $\alpha_j \approx 1$.

And since $c = \sum_{i=1}^{n} v_i \alpha_i \approx 1 \cdot v_j = v_j$, it means we will get c that is very close to v_j.

5.a.iv

It means that if the product between one of the key vectors and the query is very large compared to the other keys, then that query and the key are similar. This is because the output of the attention will be very close to the value associated with that key.

5.b

5.b.i

First, let's take a look at Ms:

$$\text{Ms} = M(v_a + v_b) = Mv_a + Mv_b = M(c_1 a_1 + c_2 a_2 + \cdots + c_n a_n) + M(d_1 b_1 + d_2 b_2 + \cdots + d_m b_m)$$

$$= MAc + MBd$$

where

$$c = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix}, \quad d = \begin{pmatrix} d_1 \\ \vdots \\ d_m \end{pmatrix}$$

Since $Ac = v_a$ and $Bd = v_b$, we can use the orthogonal properties of both bases and choose $M = AA^T$, yielding:

$$MAc = MBd = AA^T c + AA^T Bd = AIc + A^* 0d = Ac = v_a$$

5.b.ii

First, since $c \approx \frac{1}{2}(v_a + v_b)$, we get $\alpha_a = \alpha_b = \frac{1}{2}$. From equation (1a), we can deduce that this implies $k_a^T q + k_b^T q \gg k_i^T q$, for each $i \neq a, b \in [n]$.

Now, if we choose $q = \beta(k_a + k_b)$, with $\beta \gg 0$, we will get:

$$\alpha_i = \begin{cases} \frac{\exp(\beta)}{n-2+2\exp(b)} & \text{if } i = a \text{ or } i = b, \\ \frac{\exp(0)}{n-2+2\exp(\beta)} & \text{if } i \neq a, b. \end{cases}$$

Since $\beta \gg 0$, we have $\exp(\beta) \to \infty$, which means:

$$\alpha_i = \begin{cases} \frac{1}{2} & \text{if } i = a \text{ or } i = b, \\ 0 & \text{if } i \neq a, b. \end{cases}$$

as wanted.

5.c

YOUR ANSWER HERE

5.d

YOUR ANSWER HERE