NLP
Assignment N.2

December 30, 2024

Bar, Alon	Ben Tzvi, Nehoray	Baron, Lia
205476013	206389538	036635803

1 Word-Level Neural Bi-gram Language Model

In this question, you will implement and train neural language model, and evaluate it on using perplexity.

a. Derive the gradient with respect to the input of a softmax function when cross entropy loss is used for evaluation, i.e., find the gradients with respect to the softmax input vector θ, when the prediction is made by $\hat{y} = \text{softmax}(\theta)$. Cross entropy and softmax are defined as:

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

$$\text{softmax}(\theta)_i = \frac{\exp(\theta_i)}{\sum_j \exp(\theta_j)}$$

The gold vector y is a one-hot vector, and the predicted vector $\hat{y}$ is a probability distribution over the output space.

Ans:

We aim to derive the gradient of the cross-entropy loss $CE(y, \hat{y})$ with respect to the input vector θ of the softmax function.

Cross-Entropy Loss:

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

Since y is a one-hot vector, only one element $y_k = 1$ is non-zero. Therefore, the loss simplifies to:

$$CE(y, \hat{y}) = -\log(\hat{y}_k)$$

where k is the index of the true class.

Softmax Function:

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

Using the chain rule, the derivative is:

$$\frac{\partial CE}{\partial \theta_i} = \frac{\partial CE}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial \theta_i}$$

From the definition of the cross-entropy loss:

$$\frac{\partial CE}{\partial \hat{y}_i} = -\frac{y_i}{\hat{y}_i}$$

The softmax function depends on all components of θ. The derivative is:

$$\frac{\partial \hat{y}_i}{\partial \theta_j} = \begin{cases} \hat{y}_i(1 - \hat{y}_i), & \text{if } i = j \\ -\hat{y}_i \hat{y}_j, & \text{if } i \neq j \end{cases}$$

This can be compactly written as:

$$\frac{\partial \hat{y}_i}{\partial \theta_j} = \hat{y}_i(\delta_{ij} - \hat{y}_j)$$

where δ_{ij} is the Kronecker delta.

Combining the two derivatives:

$$\frac{\partial CE}{\partial \theta_i} = \sum_j \frac{\partial CE}{\partial \hat{y}_j} \cdot \frac{\partial \hat{y}_j}{\partial \theta_i}$$

Substitute $\frac{\partial CE}{\partial \hat{y}_j} = -\frac{y_j}{\hat{y}_j}$ and $\frac{\partial \hat{y}_j}{\partial \theta_i} = \hat{y}_j(\delta_{ij} - \hat{y}_i)$:

$$\frac{\partial CE}{\partial \theta_i} = \sum_j \left(-\frac{y_j}{\hat{y}_j} \cdot \hat{y}_j(\delta_{ij} - \hat{y}_i) \right)$$

Simplify:

$$\frac{\partial CE}{\partial \theta_i} = -\sum_j y_j(\delta_{ij} - \hat{y}_i)$$

Since y is a one-hot vector with $y_k = 1$ and $y_j = 0$ for $j \neq k$, only the $j = k$ term survives:

$$\frac{\partial CE}{\partial \theta_i} = -y_k(\delta_{ik} - \hat{y}_i)$$

For $y_k = 1$, this becomes:

$$\frac{\partial CE}{\partial \theta_i} = \begin{cases} \hat{y}_i - 1, & \text{if } i = k \\ \hat{y}_i, & \text{if } i \neq k \end{cases}$$

Final Result: The gradient of the cross-entropy loss with respect to θ is:

$$\frac{\partial CE}{\partial \theta_i} = \hat{y}_i - y_i$$

where $\hat{y}$ is the softmax output, and y is the one-hot true label vector.

This result shows that the gradient is the difference between the predicted probabilities and the true labels.

b. Derive the gradients with respect to the input $\boldsymbol{x}$ in a one-hidden-layer neural network (i.e., find $\frac{\partial J}{\partial \boldsymbol{x}}$, where J is the cross entropy loss $\text{CE}(\boldsymbol{y}, \hat{\boldsymbol{y}})$). The neural network employs a sigmoid activation function for the hidden layer, and a softmax for the output layer. Assume a one-hot label vector $\boldsymbol{y}$ is used. The network is defined as:

$$\boldsymbol{h} = \sigma(\boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1), \quad \hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{h}\boldsymbol{W}_2 + \boldsymbol{b}_2).$$

The dimensions of the vectors and matrices are $\boldsymbol{x} \in \mathbb{R}^{1 \times D_x}$, $\boldsymbol{h} \in \mathbb{R}^{1 \times D_h}$, $\hat{\boldsymbol{y}} \in \mathbb{R}^{1 \times D_y}$, $\boldsymbol{y} \in \mathbb{R}^{1 \times D_y}$.

The dimensions of the parameters are $\boldsymbol{W}_1 \in \mathbb{R}^{D_x \times D_h}$, $\boldsymbol{W}_2 \in \mathbb{R}^{D_h \times D_y}$, $\boldsymbol{b}_1 \in \mathbb{R}^{1 \times D_h}$, $\boldsymbol{b}_2 \in \mathbb{R}^{1 \times D_y}$.

Ans:

The cross-entropy loss gradient with respect to $\boldsymbol{u}$ (the input to the softmax) is:

$$\frac{\partial J}{\partial \boldsymbol{u}} = \hat{\boldsymbol{y}} - \boldsymbol{y}$$

The relationship between $\boldsymbol{h}$ and $\boldsymbol{u}$ is linear: $\boldsymbol{u} = \boldsymbol{h}\boldsymbol{W}_2 + \boldsymbol{b}_2$. Using the chain rule:

$$\frac{\partial J}{\partial \boldsymbol{h}} = \frac{\partial J}{\partial \boldsymbol{u}} \cdot \frac{\partial \boldsymbol{u}}{\partial \boldsymbol{h}}$$

Since $\boldsymbol{u} = \boldsymbol{h}\boldsymbol{W}_2 + \boldsymbol{b}_2$:

$$\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{h}} = \boldsymbol{W}_2^\top$$

Thus:

$$\frac{\partial J}{\partial \boldsymbol{h}} = (\hat{\boldsymbol{y}} - \boldsymbol{y})\boldsymbol{W}_2^\top$$

The hidden layer output $\boldsymbol{h}$ depends on $\boldsymbol{z}$ through the sigmoid activation: $\boldsymbol{h} = \sigma(\boldsymbol{z})$. The derivative of the sigmoid function is:

$$\sigma'(z) = \sigma(z)(1 - \sigma(z)) = h \odot (1 - h)$$

where $\odot$ denotes elementwise multiplication. Using the chain rule:

$$\frac{\partial J}{\partial \boldsymbol{z}} = \frac{\partial J}{\partial \boldsymbol{h}} \cdot \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}$$

Substituting $\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} = \text{diag}(\boldsymbol{h} \odot (1 - \boldsymbol{h}))$ (a diagonal matrix), we get:

$$\frac{\partial J}{\partial \boldsymbol{z}} = \frac{\partial J}{\partial \boldsymbol{h}} \odot (\boldsymbol{h} \odot (1 - \boldsymbol{h}))$$

Explicitly:

$$\frac{\partial J}{\partial \boldsymbol{z}} = \left[(\hat{\boldsymbol{y}} - \boldsymbol{y})\boldsymbol{W}_2^\top\right] \odot (\boldsymbol{h} \odot (1 - \boldsymbol{h}))$$

The relationship between $\boldsymbol{x}$ and $\boldsymbol{z}$ is linear: $\boldsymbol{z} = \boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1$. Using the chain rule:

$$\frac{\partial J}{\partial \boldsymbol{x}} = \frac{\partial J}{\partial \boldsymbol{z}} \cdot \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}$$

Since $\boldsymbol{z} = \boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1$:

$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \boldsymbol{W}_1^\top$$

Thus:

$$\frac{\partial J}{\partial \boldsymbol{x}} = \frac{\partial J}{\partial \boldsymbol{z}} \cdot \boldsymbol{W}_1^\top$$

Substituting $\frac{\partial J}{\partial \boldsymbol{z}}$:

$$\frac{\partial J}{\partial \boldsymbol{x}} = \left[\left[(\hat{\boldsymbol{y}} - \boldsymbol{y})\boldsymbol{W}_2^\top\right] \odot (\boldsymbol{h} \odot (1 - \boldsymbol{h}))\right] \boldsymbol{W}_1^\top$$

Final Gradient: The gradient of the loss J with respect to $\boldsymbol{x}$ is:

$$\frac{\partial J}{\partial \boldsymbol{x}} = \left[\left[(\hat{\boldsymbol{y}} - \boldsymbol{y})\boldsymbol{W}_2^\top\right] \odot (\boldsymbol{h} \odot (1 - \boldsymbol{h}))\right] \boldsymbol{W}_1^\top$$

This expresses the full backpropagation process through the softmax output layer, the hidden sigmoid layer, and the input layer.

c. Implement the forward and backward passes for a neural network with one sigmoid hidden layer. Fill in your implementation in q1c neural.py. Sanity check your implementation with python q1c neural.py.

Ans: Done in the attached python file.

d. GloVe (Global Vectors) embeddings are a type of word embeddings that represent words as vectors in a high-dimensional space, based on the co-occurrence statistics of words in a corpus. They are related to the skip-gram embeddings you saw in class in that they both aim to capture the semantic and syntactic relationships between words, but GloVe embeddings incorporate global corpus-level information in addition to local context information. In this section you will be using GloVe embeddings to represent the vocabulary. Use the neural network to implement a bigram language model in q1d neural lm.py.

Use GloVe embeddings to represent the vocabulary

(data/ lm/vocab.embeddings.glove.txt). Implement the lm wrapper function, that is used by sgd to sample the gradient, and the eval neural lm function that is used for model evaluation. Report the dev perplexity in your written solution. Don't forget to include saved params 40000.npy in your submission zip!

Ans: dev perplexity : 113.31395087517392

2 Generating Shakespeare Using a Character-level Language Model

a. Can you think of an advantage that a character-based language model could have over a word-based language model?

Ans: 1. Handling Unknown Words: Character-based models are not limited to a fixed vocabulary and can handle unknown words, such as slang, abbreviations, or typos, more effectively than word-based models.

2. Memory Efficiency: The vocabulary size in a character-based model is much smaller , reducing memory requirements compared to a word-based model, which might need to store tens of thousands of word embeddings.

3. Learning Subword Features: Character-based models can learn patterns in prefixes, suffixes, and roots, allowing them to understand and generate morphologically complex words that are not explicitly present in the training data.

And what about the other way around: can you think of an advantage a word-based language model could have over a character-based language model? (Add your answer to the final submission pdf).

Ans:

1. Context Representation: Words inherently encode semantic meaning, whereas characters must be assembled into meaningful words, making word-based models better suited for capturing higher-level semantic relationships directly.

2. Efficiency: Word-based models process fewer tokens compared to character-based models, making them computationally more efficient. For example, attention mechanisms scale as $O(l^2)$, where l is the sequence length, which is typically shorter for word-level models.

3. Simpler training: Word-based models can focus directly on sentence-level or context-level understanding without the need to first construct words from characters, simplifying the learning process for tasks requiring high-level abstraction.

b. Plotting the the losses that were computed during training can provide a further indication that the network was indeed learning (Add your plot to the final submission pdf).

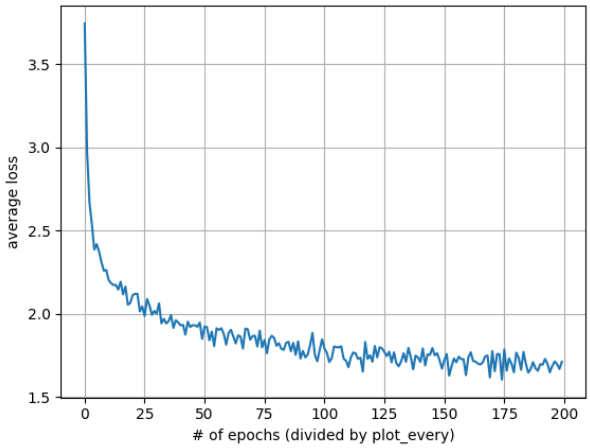

Figure 1: Average loss plot

3 Perplexity

a. Show that perplexity calculated using the natural logarithm ln(x) is equal to perplexity calculated using log2(x). i.e:

$2^{-\frac{1}{M}\sum_{i=1}^{M} log_2 p(s_i|s_1,...,s_{i-1})} = e^{-\frac{1}{M}\sum_{i=1}^{M} lnp(s_i|s_1,...,s_{i-1})}$

Ans:

Let us look at t as the base for the exponential and the log:

$t^{-\frac{1}{M}\sum_{i=1}^{M} log_t p(s_i|s_1,...,s_{i-1})} = (t^{log_t(p(s_1|s_0))}...t^{log_t(p(s_M|s_1,...,s_{M-1}))})^{\frac{-1}{M}}$

$= (p(s_1|s_0)...p(s_M|s_1,...,s_{M-1}))^{\frac{-1}{M}}$

So we see that the term does not depend on t anyway (could be 2, or e).

b. In this section you will be computing the perplexity of your previous trained models on two different passages. Please provide your results in the PDF file, as well as attach the code to your code files. The two different passages appear in the .zip file you've got. Their names are:

shakespeareforperplexity.txt which contains a subset from the Shakespeare dataset,

wikipediaforperplexity.txt which contains a certain passage from Wikipedia.

Please compute the perplexity of the bi-gram LM, and the model from section 3, on both these passages. Use the models with their own tokenizer, without any additional prepossessing.

Ans:

q1 Shakespeare perplexity : 5.770625170390492

q1 Wikipedia perplexity : 24.694984109600412

q2 Shakespeare Perplexity: 6.897611961776458

q2 Wikipedia Perplexity: 15.843863708201395

c. Try to explain the results you've got. Particularly, why there might be large gaps in perplexity, while looking on different passages.

Ans:

• **Q1 model:** The perplexity difference stems from how the tokenizer processes the input.

By mistakenly **treating entire sentences as single "UNKNOWN" tokens**, the model effectively predicts sentences as indivisible units, ignoring their internal structure.

This can artificially lower perplexity compared to tokenizing individual words because the model is not penalized for missing finer-grained predictions within the sentence. Instead of needing to predict each word based on context—a much harder task—it only has to predict the broad, abstract notion of the next "sentence token."

For Shakespeare's text, this treatment results in lower perplexity (5.771) because its sentences are often rhythmic, structured, and follow consistent patterns that make them more predictable as whole units. On the other hand, Wikipedia text is dense with factual details, proper nouns, and complex sentence structures, making its sentences harder to predict even as single units, leading to higher perplexity (24.695).

If the tokenizer functioned as intended—breaking down sentences into individual tokens—perplexity would likely rise for both texts, as the model would need to handle the intricacies of word-by-word prediction, especially with Shakespeare's archaic phrasing and Wikipedia's detailed information. By collapsing entire sentences into "UNKNOWN," the model sidesteps much of the prediction challenge, favoring texts with more regularity and consistent structure, like Shakespeare, while struggling with more variable and information-dense content, like Wikipedia.

• **Q2 model:** The perplexity difference between the Wikipedia input and the Shakespeare input occurs because the model was trained on Shakespearean-style text, making it more familiar with the vocabulary, syntax, and structure of the Shakespeare passage. As a result, it predicts character sequences in Shakespearean text more accurately, leading to lower perplexity. In contrast, the Wikipedia text uses modern vocabulary and technical terms that the model is less familiar with, leading to higher uncertainty. Additionally, the structure of Wikipedia is more complex and differs from Shakespeare's style, making predictions less reliable and increasing perplexity.

d. Implement a prepossessing step to improve the results. Explain in the pdf your solution.

Ans:

After pre-processing, we got the following results:

• **Q1 model:** After the preprocessing step we added, which tokenizes sentences into individual words and punctuation marks, placing each on a separate line, and uses blank lines to separate sentences, the model's tokenizer can now properly process the input as individual tokens rather than entire sentences.

This change significantly increases perplexity for both texts because the model must predict each word or punctuation mark based on its context, a far more granular and challenging task than predicting entire sentences - More components adding to the loss.

For Wikipedia, the modern, factual language and structured syntax make individual words more predictable, resulting in a lower perplexity (74.888). Shakespeare's text, however, with its archaic vocabulary, poetic structure, and less conventional syntax, introduces greater variability and unpredictability at the word level, leading to a higher perplexity (94.191). The preprocessing ensures proper tokenization but also exposes the model to the full complexity of the text, amplifying the differences in predictability between the two writing styles.

• **Q2 model:** In the preprocessing step, we applied several transformations to the input text, including converting it to lowercase, removing numbers and non-alphabetic characters, removing extra whitespaces, and replacing line breaks with spaces. These changes helped standardize the text and removed elements that could have caused noise for the model, such as numbers and punctuation. The reduction in perplexity for Wikipedia (from 20.844 to 10.928) can be attributed to the fact that the preprocessing removed modern terms, symbols, and formatting, making the text more consistent with the Shakespearean style the model was trained on.

However, the slight increase in perplexity for Shakespeare (from 7.268 to 7.825) suggests that the model might have been slightly less adept at handling the uniformity introduced by the preprocessing, especially if it lost some subtle stylistic features of the Shakespearean text.

While preprocessing helped the model handle Wikipedia text better by aligning it closer to its training data, it may have made Shakespeare text a bit too simplified, slightly increasing its perplexity.

4 Deep Averaging Networks

(e) For your best model, sample 5 examples from the evaluation set that the model classified incorrectly and for each example try to explain why the model classified it incorrectly

Ans:

Text 0 (False Negative)

- The text contains mixed sentiments:
- Positive phrases: "entertaining," "hot girls posing in their bikinis," and "recommend it."
- Negative phrases: "boring," "off-putting," and "wouldn't recommend it to women wanting to go into the modelling business."

Text 1 (False Positive)

- The text begins with some positive expectations: "looking forward to Man of the Year," "pretty good movie," and "liked it."
- However, the review quickly shifts to negative sentiment: "huge disappointment," "barely made through it," and "wouldn't recommend this movie."
- The model may have weighted the initial positive phrases more heavily than the subsequent negative ones, leading to misclassification.

Text 2 (False Positive but almost neutral)

- The text critiques a movie harshly: "unfunny," "very poor example of filmmaking," and "nosedived."
- Despite the negative tone, the model output is nearly neutral, with a slight positive tilt.
- This could be due to phrases like "short is a perfect example" being misinterpreted as positive.

Text 3 (False Positive)

- Positive-sounding phrases like "awesome scenery" and "stunning cinematography" could have led the model to assign a positive sentiment.

- Negative phrases, such as "characters are a crashing bore" and "story itself is totally tame," were likely overshadowed by positive keywords.

Text 4 (False Negative but almost neutral)

- The text praises the movie as "not half bad," "good B-horror movie fun," and "badass," but also acknowledges typical horror movie flaws like "nothing new" and "cliché."
- The balanced sentiment confused the model, leading it to predict a slightly negative sentiment instead of positive.

5 Attention Exploration

Multi-head self-attention is the core modeling component of Transformers. In this question, we'll get some practice working with the self-attention equations, and motivate why multi-headed self-attention can be preferable to single-headed self-attention.

Recall that attention can be viewed as an operation on a query vector $q \in \mathbb{R}^d$, a set of value vectors $v_1, ..., v_n, v_i \in \mathbb{R}^d$, and a set of key vectors $k_1, ..., k_n, k_i \in \mathbb{R}^d$, specified as follows:

$$c = \sum_{i=1}^{n} v_i \alpha_i \tag{1}$$

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

with $\alpha = \{\alpha_1, \ldots, \alpha_n\}$ termed the "attention weights." Observe that the output $c \in \mathbb{R}^d$ is an average over the value vectors weighted with respect to α.

5.1 Copying in Attention

One advantage of attention is that it's particularly easy to "copy" a value vector to the output c. In this problem, we'll motivate why this is the case.

a. **Explain** why α can be interpreted as a categorical probability distribution.

Ans:

The following prerequisites for a categorical probability distribution are: 1. $n > 0$ number of categories.

2. Non-negativity: $\alpha_i = \frac{\exp(k_i^\top q)}{\sum_{j=1}^n \exp(k_j^\top q)}$ the nominator and dominator are exponential $\rightarrow \alpha_i > 0$

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

The vector $\alpha = \{\alpha_1, \ldots, \alpha_n\}$ satisfies the axioms of a categorical probability distribution

b. The distribution α is typically relatively "diffuse"; the probability mass is spread out between many different α_i. However, this is not always the case. **Describe** (in one sentence) under what conditions the categorical distribution α puts almost all of its weight on some α_j, where $j \in \{1, \ldots, n\}$ (i.e., $\alpha_j \gg \sum_{i \neq j} \alpha_i$). What must be true about the query q and/or the keys $\{k_1, \ldots, k_n\}$?

Ans:

The dot product $k_j^\top q$ is significantly larger than $k_i^\top q$ for all $i \neq j$, meaning the query q is highly similar to the key k_j compared to all other keys

c. Under the conditions you gave in (b), **describe** the output c.

Ans:

We can approximate $\alpha_j \approx 1 \rightarrow c \approx v_j$

d. **Explain** (in two sentences or fewer) what your answer to (b) and (c) means intuitively.

Ans:

If one of all key vectors is very similar or almost identical to the given query, the attention weight will be put almost exclusively on its value. Therefore, the output equals that value and in that sense we "copied" the keys value.

5.2 An average of two

Instead of focusing on just one vector v_j, a Transformer model might want to incorporate information from multiple source vectors. Consider the case where we instead want to incorporate information from two vectors v_a and v_b, with corresponding key vectors k_a and k_b.

a. How should we combine two d-dimensional vectors v_a, v_b into one output vector c in a way that preserves information from both vectors? In machine learning, one common way to do so is to take the average: $c = \frac{1}{2}(v_a + v_b)$. It might seem hard to extract information about the original vectors v_a and v_b from the resulting c, but under certain conditions one can do so. In this problem, we'll see why this is the case.

Suppose that although we don't know v_a or v_b, we do know that v_a lies in a subspace A formed by the m basis vectors $a_1, a_2, ..., a_m$, while v_b lies in a subspace B formed by the p basis vectors $b_1, b_2, ..., b_p$. (This means that any v_a can be expressed as a linear combination of its basis vectors, as can v_b. All basis vectors have norm 1 and are orthogonal to each other.) Additionally, suppose that the two subspaces are orthogonal; i.e. $a_j^T b_k = 0$ for all j, k. Using the basis vectors $a_1, a_2, ..., a_m$, construct a matrix M such that for arbitrary vectors $v_a \in A$ and $v_b \in B$, we can use M to extract v_a from the sum vector $s = v_a + v_b$. In other words, we want to construct M such that for any $v_a, v_b, M_s = v_a$. Show that $M_s = v_a$ holds for your M.

Hint: Given that the vectors $a_1, a_2, ..., a_m$ are both orthogonal and form a basis for v_a,we know that there exist some $c_1, c_2, ..., c_m$ such that $v_a = c_1a_1 + c_2a_2 + ... + c_ma_m$. Can you create a vector of these weights c?

Ans:

$s = v_a + v_b, Ms = v_a \rightarrow M(v_a + v_b) = v_a$ (def.)

We need to find M such as $Mv_a = v_a, Mv_b = v_b$

We will use $M = AA^T$

$v_a = c_1a_1 + c_2a_2 + ... + c_ma_m$ for some $c_1, c_2, ..., c_m$

(vectors $a_1, a_2, ..., a_m$ are both orthogonal and form a basis for v_a)

$Mv_a = AA^Tv_a = \sum_{j=1}^m a_j a_j^\top (\sum_{k=1}^m c_k a_k) = \sum_{j=1}^m \sum_{k=1}^m c_k a_j (a_j^\top a_k)$

$= \sum_{j=1}^m c_k a_k = v_a$ $\{a_1, a_2, \ldots, a_m\}$ are orthonormal $\rightarrow a_j^\top a_k = \begin{cases} 1, & \text{if } j = k \\ 0, & \text{if } j \neq k \end{cases}$

$Mv_b = AA^Tv_b = \sum_{j=1}^m a_j a_j^\top v_b = 0$ (A and B Are Orthogonal$\rightarrow a_j^\top b_k = 0$)

Combined we get $Ms = Mv_a + Mv_b = v_a + 0 = v_a$

b. As before, let v_a and v_b be two value vectors corresponding to key vectors k_a and k_b, respectively. Assume that (1) all key vectors are orthogonal, so $k_i^T k_j = 0$ for all $i \neq j$; and (2) all key vectors have norm 1. **Find an expression** for a query vector q such that $c \approx \frac{1}{2}(v_a + v_b)$, and justify your answer.

Ans:

$c \approx \frac{1}{2}(v_a + v_b) \rightarrow \alpha_a, \alpha_b \approx \frac{1}{2}$

$k_a^T q \approx k_b^T q \gg k_j^T q \forall j \neq a, b$ (α is categorical probability distribution)

We will use $q = \lambda(k_a + k_b)$ when $\lambda \gg 0$ and analyze $k_j^T q \forall j$

$k_j^T \lambda(k_a + k_b) = \lambda(k_j^T k_a + k_j^T k_b)$

$j \neq a, b : k_j^T k_a = 0, k_j^T k_b = 0 \rightarrow k_j^T q = 0$ (key vectors are orthogonal)

$j = a : k_a^T k_a = 1, k_a^T k_b = 0 \rightarrow k_j^T q = \lambda$ (key vectors are orthogonal)

$j = b : k_b^T k_a = 0, k_a^T k_b = 1 \rightarrow k_j^T q = \lambda$ (key vectors are orthogonal)

$\alpha_a = \alpha_b = \frac{exp(\lambda)}{2exp(\lambda)} = 0.5$ (α def.)

5.3 Drawbacks of single-headed attention

In the previous part, we saw how it was possible for a single-headed attention to focus equally on two values. The same concept could easily be extended to any subset of values. In this question we'll see why it's not a practical solution. Consider a set of key vectors $k_1, ..., k_n$ that are now randomly sampled, $k_i \sim \mathcal{N}(\mu_i, \Sigma_i)$, where the means $\mu_i \in \mathbb{R}^d$ are known to you, but the covariances Σ_i are unknown. Further, assume that the means μ_i are all perpendicular; $\mu_i^T \mu_j = 0$ if $i \neq j$, and unit norm, $||\mu|| = 1$.

a. Assume that the covariance matrices are $\Sigma_i = \alpha I \forall i \in 1, 2, ..., n$, for vanishingly small α. Design a query q in terms of the μ_i such that as before, $c \approx \frac{1}{2}(v_a + v_b)$, and provide a brief argument as to why it works.

Ans:

$k_i \sim \mathcal{N}(\mu_i, \Sigma_i) \rightarrow k_i \sim \mathcal{N}(\mu_i, \alpha_i I)$ ($\Sigma_i = \alpha I$)

$k_i = \mu_i + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, \alpha I) \rightarrow k_i \approx \mu_i$ (α is vanishingly small)

$k_a \approx \mu_a, k_b \approx \mu_b$

$q = \lambda(\mu_a + \mu_b)$ (Same proof as 5.2 b)

b. Though single-headed attention is resistant to small perturbations in the keys, some types of larger perturbations may pose a bigger issue. Specifically, in some cases, one key vector k_a may be larger or smaller in norm than the others, while still pointing in the same direction as μ_a. As an example, let us consider a covariance for item a as $\Sigma_a = \alpha I + \frac{1}{2}(\mu_a \mu_a^T)$ for vanishingly small α (as shown in the figure). This causes k_a

to point in roughly the same direction as μ_a, but with large variances in magnitude. Further, let $\Sigma_i = \alpha I$ for all $i \neq a$.

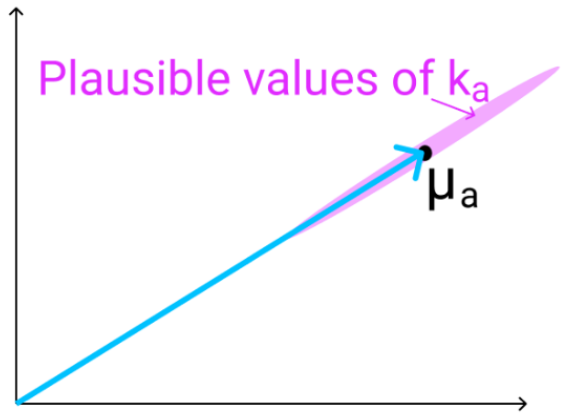

Figure 2: The vector μ_a (shown here in 2D as an example), with the range of possible values of k_a shown in red. As mentioned previously, k_a points in roughly the same direction as μ_a, but may have larger or smaller magnitude.

When you sample $k_1, ..., k_n$ multiple times, and use the q vector that you defined in part i., what do you expect the vector c will look like qualitatively for different samples? Think about how it differs from part (i) and how c's variance would be affected.

Ans:

$k_a \sim \mathcal{N}(\mu_a, \Sigma_a) \rightarrow k_a \sim \mathcal{N}(\mu_a, \alpha I + \frac{1}{2}(\mu_a \mu_a^T))$ $\quad (\Sigma_a = \alpha I + \frac{1}{2}(\mu_a \mu_a^T))$

$k_a = \mu_a + \epsilon_a, \quad \epsilon_a \sim \mathcal{N}(0, \alpha I + 0.5) \rightarrow \epsilon_a = \gamma \mu_a, \quad \gamma \sim \mathcal{N}(0, 0.5)$ (α is vanishingly small)

$k_a = (1 + \gamma)\mu_a, \quad \gamma \sim \mathcal{N}(0, 0.5)$

$$\alpha_a = \frac{exp([(1+\gamma)\mu_a]^T \lambda(\mu_a + \mu_b))}{exp([(1+\gamma)\mu_a]^T \lambda(\mu_a + \mu_b) + exp(\mu_b^T(\lambda(\mu_a + \mu_b))} = \frac{exp(\lambda(1+\gamma))}{exp(\lambda(1+\gamma)) + exp(\lambda)}$$

$$\alpha_b = \frac{exp(\mu_b^T \lambda(\mu_a + \mu_b))}{exp([(1+\gamma)\mu_a]^T \lambda(\mu_a + \mu_b) + exp(\mu_b^T(\lambda(\mu_a + \mu_b))} = \frac{exp(\lambda)}{exp(\lambda(1+\gamma) + exp(\lambda)}$$

$\gamma \approx 0.5 \rightarrow \alpha_a = 1, \alpha_b = 0$ $\gamma \approx 0 \rightarrow \alpha_a = \alpha_b$ $\gamma \approx -0.5 \rightarrow \alpha_a = 0, \alpha_b = 1$

When sampling $\{k_1, \ldots, k_n\}$ multiple times:

1. Qualitatively: The attention output c becomes less stable and fluctuates between being dominated by v_a or v_b (when k_a is large or small) and being a balanced average of $v_a$and v_b (when k_a is closer to its mean).

2.Variance: The variance of c is significantly higher compared to part (i), due to the large variability in k_a norm caused by the second term ($\frac{1}{2}(\mu_a\mu_a^\top)$) in Σ_a.

5.4 Benefits of multi-headed attention

Now we'll see some of the power of multi-headed attention. We'll consider a simple version of multi-headed attention which is identical to single-headed self-attention as we've presented it in this homework, except two query vectors (q_1 and q_2) are defined, which leads to a pair of vectors (c_1 and c_2), each the output of single-headed attention given its respective query vector. The final output of the multi-headed attention is their average, $\frac{1}{2}(c_1 + c_2)$. As in question 1(3), consider a set of key vectors $k_1, ..., k_n$ that are randomly sampled, $k_i \sim \mathcal{N}(\mu_i, \Sigma_i)$, where the means μ_i are known to you, but the covariances Σ_i are unknown. Also as before, assume that the means μ_i are mutually orthogonal; $\mu_i^T\mu_j = 0$ if $i \neq j$, and unit norm, $||\mu_i|| = 1$.

a. Assume that the covariance matrices are $\Sigma_i = \alpha_i I$, for vanishingly small α. Design q_1 and q_2 such that c is approximately equal to $\frac{1}{2}(v_a + v_b)$. Note that q_1 and q_2 should have different expressions.

Ans:

$q_1 = \lambda\mu_1, q_2 = \lambda\mu_2$

$$\mu_i^\top \mu_a \approx \begin{cases} \lambda, & \text{if } i = a, \\ 0, & \text{otherwise.} \end{cases}$$

$$k_i^\top q_1 \approx \begin{cases} \lambda, & \text{if } i = a, \\ 0, & \text{otherwise.} \end{cases}$$

$c_1 = \sum_{i=1}^n v_i \frac{\exp(k_i^\top q)}{\sum_{j=1}^n \exp(k_j^\top q)} = v_a$

$c_2 = \sum_{i=1}^n v_i \frac{\exp(k_i^\top q)}{\sum_{j=1}^n \exp(k_j^\top q)} = v_b$

$c = \frac{1}{2}(c_1 + c_2) = \frac{1}{2}(v_a + v_b)$

b. Assume that the covariance matrices are $\Sigma_a = \alpha I + \frac{1}{2}\mu_a\mu_a^T$ for vanishingly small α, and $\Sigma_i = \alpha I$ for all $i \neq a$. Take the query vectors q_1 and q_2 that you designed in part i. What, qualitatively, do you expect the output c to look like across different samples of the key vectors? Explain briefly in terms of variance in c_1 and c_2. You can ignore cases in which $k_a^T q_i < 0$.

Ans:

$c_2 = v_b$ (Same proof as 5.4 a)

$k_a = (1+\gamma)\mu_a, \quad \gamma \sim \mathcal{N}(0, 0.5)$ (Same proof as 5.3 b)

$c_1 = \sum_{i=1}^{n} v_i \frac{\exp(k_i^\top q)}{\sum_{j=1}^{n} \exp(k_j^\top q)} = v_a$

$$k_i^\top q_1 \approx \begin{cases} \lambda(1+\gamma), & \text{if } i = a, \\ 0, & \text{otherwise.} \end{cases}$$

$c = \frac{1}{2}(c_1 + c_2) = \frac{1}{2}(v_a + v_b)$