NLP3 - Lab2 – Group 9

Mathieu Rivier mathieu.rivier@epita.fr
Moustapha Diop moustapha.diop@epita.fr
Marius Dubosc marius.dubosc@epita.fr
Philippe Bernet philippe.bernet@epita.fr
Arthur Fan arthur.fan@epita.fr

January 2023

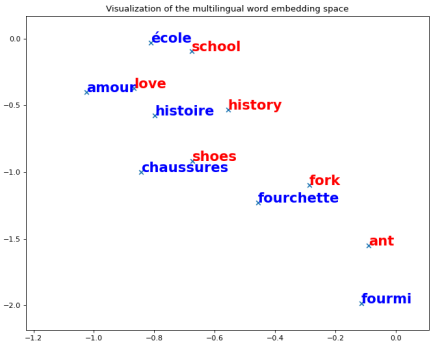

Figure 1: Visualization of the embeddings of French word to English after a PCA and applying MUSE alignment

Abstract

This is the final course of a series of three on Natural Language Processing (NLP). This report represents the second explanation hand-out out of two practicals and explains the attached Jupyter-notebook. This notebook covers language detection, semantic space rotation and attention exploration. Those will be treated in order and answer the questions from the notebook.

All the results mentioned in this report are reproducible in the above mentioned Jupyter-notebook.

Contents

1 Language Detection (24 points) – Guided coding

1.0.1 Try out a translation of a French sentence in Google Translate (or Bing Translate) to English.

We chose the following statement in French to give to Google Translate : "Le jour de l'an, on se réveille généralement tard car on a fait la fête toute la nuit". Google Translate outputs : "On New Year's Day, we usually wake up late because we've been partying all night" when the specified source language is French and the destination language is English. The translation is correct.

1.0.2 What happens if you select the wrong source language as follows?

To challenge our translator (Google Translate), we can change the source language. The source statement is the same as in 1.0.1.

Source	Translation
French	On New Year's Day, we usually wake up late because we've been partying all night
Italian	Le jour de l'an, on se réveille généralement tard car on a fait la fête toute la nuit
Spanish	Le jour de l'an, on se réveille **generally late** car on a fait la fête toute la nuit
German	Le jour de l'an, on se réveille généralement tard car on a fait la fete toute la nuit
Armenian	Le jour de l'an, on se réveille **generally** tard car on fait la fête toute la nuit
Finish	Le jour de l'an, on se réveille **generally** tard car on fait la fête toute la nuit

Languages are sorted according to their distance to French following a language tree.

On most languages, Google Translate does not do anything and propose to switch to a French source language. Google Translate set out how to translate some words (the bold one). This behaviour may come from the fact that in the source language the word is well identified. In Spanish, "généralement tard" is said "generalemente tarde", which is the same as in French without accents and with a 'e' that may be interpreted by Google Translate as a typo. However, in Armenian, "généralement" is pronounced "yndhanrapes" and in Finnish it is said "yleisesti", we have no idea how Google Translate figured out how to translate this specific word.

1.0.3 Explain in a few sentences what is happening in the backend.

To translate a piece of text, the service first breaks the text down into smaller units called "segments." It then uses machine learning algorithms to determine the most likely translation of each segment based on the context and previous translations. The service also uses other resources, such as dictionaries and pre-translated texts, to improve the accuracy of the translation.

1.1 Describe the language distribution of the dataset. What is the distribution of languages?

```
language

Arabic     1000 ; Chinese    1000
Dutch      1000 ; English    1000
Estonian   1000 ; French     1000
Hindi      1000 ; Indonesian 1000
Japanese   1000 ; Korean     1000
Latin      1000 ; Persian    1000
Portugese  1000 ; Pushto     1000
Romanian   1000 ; Russian    1000
Spanish    1000 ; Swedish    1000
Tamil      1000 ; Thai       1000
Turkish    1000 ; Urdu       1000
dtype: int64
```

Figure 2: Distribution of the dataset

The language distribution of the dataset is perfect, each language has an equal number of samples. This means that the detection should be equivalent on most languages.

1.2 Do the appropriate pre-processing to maximise the accuracy of language detection. What is your strategy?

The strategy is to clean the dataset, make all text lower case to maximise the detection capacity of the model that will be trained on the data. Further it is better to remove numbers that don't help as much in the detection of language. The specific characters such as '#' have also been stripped to reduce noise as much as possible. Finally, multiple spaces have been removed to prevent recognising them as important. The labels have been encoded into numbers using a `LabelEncoder`.

1.3 What would be the problem if your dataset was unbalanced?

With an unbalanced dataset, the trained model may overfit on the oversampled data. The detection prediction may include a bias and worsen the predictions.

1.4 What techniques could you use to solve that?

To overcome an unballanced dataset, many possibilities exists :

- oversampling and undersampling : In order to have the same amount of data for each class, it is possible to duplicated the undersampled data or remove data from the oversampled data.
- ensemble method : Bagging and Boosting, instead of doing a unique model classifying the input, it is possible to use a specific model for each class (here for each language), the model specialization would be to distinguish any language from the others with a certain probability for the sample to be the recognized language.
- weighted loss function : During the gradient descent, it is possible to artificially add a bias in the computation of the weights multiplying the gradient value.

1.5 Train a model of your choice and describe the accuracy across languages. Use an 80%, 20% train-test split. Performance is not key but explain thoroughly the process and the metric(s) you are tracking.

As can be seen in figure 3, the `MultinomialNB` model that was used for this question is very accurate. In fact, it has an **accuracy score of 0.93 on the testing set** that accounts of 20% of the total dataset.

Furthermore, figure 3 shows that the model is extremely consistant in its ability to accurately detect most languages. Below 90% score can only be seen for two languages: English and Swedish. Which appears surprising when knowing that both languages had the same number of samples as the other ones did. Through testing, it was noticed that two languages would always be detected less than the others. Although, they seem to arbitrarily change when re-trained.

```
Accuracy for each language:
Arabic    : 1.000000 ; Chinese   : 0.910256
Dutch     : 0.982533 ; English   : 0.713235
Estonian  : 0.984456 ; French    : 0.939394
Hindi     : 0.995146 ; Indonesian: 0.995215
Japanese  : 0.968085 ; Korean    : 0.994709
Latin     : 0.979487 ; Persian   : 0.994898
Portugese : 0.994709 ; Pushto    : 1.000000
Romanian  : 0.979695 ; Russian   : 0.985981
Spanish   : 0.975369 ; Swedish   : 0.493113
Tamil     : 1.000000 ; Thai      : 1.000000
Turkish   : 0.989950 ; Urdu      : 1.000000
```

Figure 3: Language results from `MultinomialNB` model

As mentioned above, the tracked metrics are both the general accuracy that gives an overall estimate of the language detection quality, as well as the accuracy per language, which provides a more refined and granural way of analysing the quality of predictions.

The preccess is to tokenise the dataset in order to get data that the `MultinomialNB` can process. It is then a matter of training the model, which is relatively fast about 2 minutes. A pre-processing has been performed to ensure the quality of the input data.

1.6 Train a `fasttext` model on Tatoeba parallel corpus and check that performance is good.

```
CPU times: user 1h 40min 15s, sys: 9.02 s,
↪  total: 1h 40min 24s
Wall time: 1h 40min 40s
```

Figure 4: Training time for `FastText` model with 5 epochs

In order to give the fasttext model the best results possible, it has been trained on the Tatoeba parallel corpus with five epochs. This training has taken nearly two hours as is visible in figure 4. Furthermore, safety checks have been conducted in both French and English to make sure the model was identifying simple sentences correctly.

```
Validation set accuracy: 0.9879
```

Figure 5: `valid.txt` accuracy

As can be seen in figure 5, the results on the validation set are more than satisfactory. The score obtained on the validation set is near perfect, and proves the quality of the `FastText` model.

1.7 Test your `FastText` model on the same dataset as in question 1.5. Compare with your custom model (make sure you use the exact same data for testing). How can you explain the difference in performance between the two models?

To make the comparisons fair, the `MultinomialNB` model in section 1.5 and the `FastText` model in this section have the same exact data to run on. The results are further reproducible, since a random state has been set upon generation. in section 1.5 of the attached colab notebook, a train-test-split has been run on the indices of the common dataset. The data is then collected through a custom function that gets the values and labels. In this way the comparisons between both models are fair.

On this data-set the `FastText` model lacks behind compared to the `MultinomialNB` model seen in section 1.5. **It has a test set accuracy of 0.80** tested on the same exact samples as the previous model as mentioned above. It thus trailing 13% behind the previous model. Even though it scored better on its own validation set.

A comparison of the results from figure 3 and figure 6 shows the differences and gives meaning to the accuracies. In order to be able to compare both models though, a conversion of the labels from the `FastText` had to be made. Finding the right language codes and converting them was made easier by the Tatoeba page referencing the different languages and codes it contains, with the number of sentences per language. The language code are iso-639 and were translated to the language name with an external library. This enables us to have the following figure 6:

The accuracy on most languages are slightly lower with the FastText model, with a few languages really dropping in accuracy. Although for Pushto, Chinese, Tamil, and Thai those predictions are quite mediocre. While solving the 'Portugese' problem was simple (i.e adding a u: 'Portuguese'), the others were not as simple. In fact,

```
Accuracy for each language:
Arabic    : 0.980198 ; Chinese   : 0.004975
Dutch     : 0.969565 ; English   : 1.000000
Estonian  : 0.810000 ; French    : 0.989362
Hindi     : 0.985577 ; Indonesian: 0.868545
Japanese  : 0.845361 ; Korean    : 0.957895
Latin     : 0.885714 ; Persian   : 0.994898
Portugese : 0.958763 ; Pushto    : 0.000000
Romanian  : 0.969543 ; Russian   : 0.995305
Spanish   : 0.984925 ; Swedish   : 1.000000
Tamil     : 0.474747 ; Thai      : 0.117347
Turkish   : 0.974874 ; Urdu      : 0.866995
```

Figure 6: Language results from `FastText` model trained on the tatoeba corpus

Chinese is very ambiguous as there is no one Chinese language, but a multitude. In the Tatoeba corpus alone there are eight different Chinese languages. For the purpose of this test, all of them were considered Chinese, trying each language separately before adding them together. In the end the detection of Chinese language on the testing set remains nonexistent. The 'Pushto' or 'Pashto' language on the other end simple had little to no sentences in the Tatoeba corpus: 60 which does not represent enough for a meaningful training.

The main difference that may explain the disparities in result of the two models, is that the MultinomialNB was trained on the final comparison dataset with the right labels. On the other hand, the FastText model was trained on a different dataset, with much more data, but also more languages. Some of the predictions that the FastText model has made on the testing set contained languages that were not present in this one creating more possibilities to make mistakes.

It is important to highlight the impressive language detection capabilities of the MultinomialNB model, since it takes about 2 minutes to train and has better testing performance than the FastText model in this specific case.

1.8 Compute your performance metrics yourself and compare with sklearn. [code]

The metrics do the same thing and have the same exact results.

1.9 How could you improve the fasttext model performance from the previous question? Explain in a few sentences.

There are several ways of improving the performance of a FastText model:

The most simple one being increasing the size of the training data. Although in this case, this is not a problem or something that should be considered first.

It is also possible to fine tune the model's hyperparameters such as the learning rate or the number of epochs. Those have significant impact on a model's performance.

Another solution could be to use pre-trained word embeddings as input to the model. This would give additional information about a words meaning and would thus possibly improve performance.

If everything else has been done, other more complex solutions are available. Using an ensemble of multiple FastText models, and transfer learning can also help improve a model's performance.

1.10 Which method would you use for language detection and why?

Which method to use really depends on the data and needs. While the methods manually explored worked better than fasttext, it also detects much less languages and was trained on less data, meaning that in a real life scenario or with different data, it might not have performed as well. Furthermore, this model was very, very ram-ivor, it consumed ram like if it was nothing, the main problem being to tokenize large

amounts of data, and storing them in the ram while training, or testing.

Although, in the end it really depends what the priority is and the dataset being used. The FastText model is generally considered to be fast and efficient, as it can process large amounts of text data quickly. However, it is also less accurate than some other models, such as the Multinomial Naive Bayes (MultinomialNB) classifier, which is often used for text classification tasks such as language detection. The MultinomialNB classifier is based on the assumption that the features in the data are conditionally independent given the class label, and is known for its good performance and simplicity.

In general, if speed is the primary concern, the FastText model is a good choice, while if accuracy is more important, the MultinomialNB classifier may be a better choice.

1.11 Given a sentence with N_1 tokens in English and N_2 token in French, what would be your strategy to assign a language to such sentence?

There are multiple takes as to how to assign a language to a sentence with tokens in multiple languages. Such strategies include:

Identifying the language of each token individually. Meaning that each token would have a identified language. The model can then assign a language to the entire sentence based on the identified languages of individual tokens. For example: use the language of the majority of tokens. Meaning that the sentence gets classified as English if most of its tokens are in English.

Another strategy is to base the prediction on contextual information. If the sentence is part of a larger document to which the language is known, this can help classify the sentence's language.

Ultimatly which strategy is chosen depends on the context and specific case. Although, basing a case on a similar example to the one studied in this paper, the first option would be the best, since there are no context or document surrounding each sentence.

1.12 Would a multilingual architecture be robust to multiple languages in a single sentence? Elaborate your answer accordingly.

It is possible for a multilingual architecture to be robust to multiple languages in a single sentence, but it depends on the specific architecture and amount of training data it has been given.

A multilingual architecture is designed to handle multiple languages and is trained on data from multiple languages. This type of architecture can be robust to multiple languages in a single sentence if it has sufficiently been trained on data that includes multiples languages in a sentence and if it has been designed with the ability to identify and differentiate between the different languages in a given sentence.

However, if a multilingual architecture has not been adequately trained on data that includes examples of multiple languages in a single sentence, or if it has not been designed to handle this type of input, it may struggle to accurately process and understand the multiple languages in a single sentence.

In general, the more training data an architecture has been given and the more sophisticated its design, the more likely it is to be robust to multiple languages in a single sentence.

2 Rotate two semantic spaces (23 points) – Not guided coding

2.1 Explain in a few sentences how MUSE is doing the alignment of the semantic spaces in the supervised way

In the supervised way, MUSE[1] uses a trained bilingual dictionary (using identical character

[1] `https://github.com/facebookresearch/MUSE`

strings) as anchor points to learn a mapping from the source to the target . The mapping is performed using a *Procrustes* alignment, which is a technique to align two sets of data doing rotation, translation, and scaling. First, sets are centred to remove the translation of the alignment. Then, optimal rotation and scale factors are found using a Single Value Decomposition[2] (SVD) or an iterative optimization process (such as a gradient descent). In the context of MUSE, it is not specified in the article [1] which of the previously cited techniques is used to compute the factors.

2.2 What is the limit of doing that alignment based on the approach taken in the supervised way?

The main limit of the supervised learning of alignment in MUSE is that the alignment is based on anchor points and bilingual dictionaries. As a consequence, without dictionaries or not enough data in it, the supervised way cannot be used or will not result in a good alignment. Furthermore, MUSE use as anchors points the identical string, assuming that two identical words in different language share the same meaning. It is not always true, and even less true for words with multiple meaning following the context. For example, in English the word "bass" may refer : to the instrument, to a low frequency sound, or to the lowest part in music. Even if the lexical field is the same, the meaning is quite different and the alignment may lead to mistakes. As an example for word having the same string but different meanings : in Italian and Spanish the word "salsa" exists but respectively refers to the sauce (same as in French) and to the dance.

The supervised way for the alignment of languages using MUSE has to be performed on high-quality dictionaries.

[2] `https://en.wikipedia.org/wiki/Singular_value_decomposition`

2.3 How can we align two semantic spaces in a domain specific field, e.g., in a tech company?

To align specific semantic spaces in a specific domain, multiple techniques may be used :

- use a domain specific bilingual dictionary and perform a supervised alignment using MUSE if the dictionary is filled enough.
- use unsupervised alignment using parallel texts (same meaning in the texts but in different languages).
- use a specific word-embedding algorithm for the specific domain.

2.4 Align the French space and the English space together, with the method of your choice.

We used MUSE to align the French space to the English space. To do so, we clone the MUSE repository. We download the data using the script in the folder *data*.

```
python supervised.py --src_lang fr
↪  --tgt_lang en --src_emb
↪  data/wiki.fr.vec --tgt_emb
↪  data/wiki.en.vec --n_refinement 5
↪  --dico_train "identical_char"
```

Figure 7: Alignment of the French space to the English space using MUSE

In figure 7, we use *supervised.py* to train the MUSE model from source language *fr* to target language *en* using the embeddings of *wiki*. The anchor points are the *identical_char*.

see code in the attached notebook.

2.5 Visualize the output on a few words of your choice. Comment on the performance on the alignment based on the output.

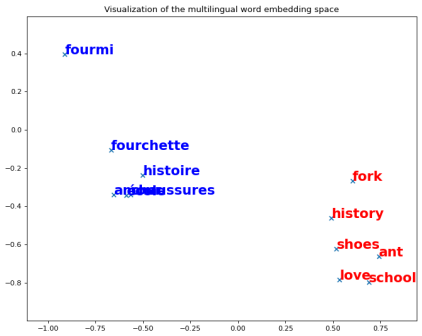

Figure 8: Visualization of the embeddings of French word to English after a PCA

Using the file *demo.ipynb* in MUSE repository. We can display on a 2D space, obtained using a PCA on the embeddings, some chosen words. In figure 8, we can see the disposition in the space without the rotation of the semantic spaces. We can observe that words with similar meaning such as *fork* and *fourchette* are not close to each other. Meanwhile, in figure 9, each French word is close to its translation in English. They also seem to be closer to their translation than to any other word (it will be verified in the next section). This is because the PCA has been performed on the embedding after the alignment.

2.6 How can you find the translation of a word with this approach? Explain your method and the distance metric you choose in a few sentences.

It is possible to compute the nearest embedded word after the alignment. The goal is to get the embeddings, id2word, and word2id dictionaries for both the src and target data sets. Then the `get_nn` provides the closest K neighbours to the word passed as parameter. This is easily done thanks to MUSE library.

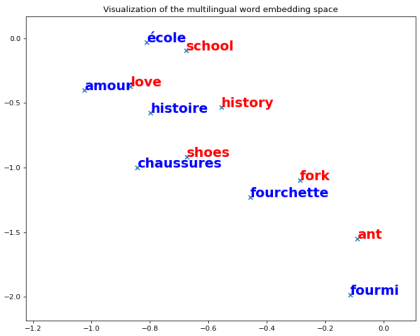

Figure 9: Visualization of the embeddings of French word to English after a PCA after applying MUSE alignment

Word/ neighbours	**'école'**	**'amour'**	**'fourchette'**
neighbour 1	0.7419 - school	0.7452 - love	0.4701 - estimate
neighbour 2	0.6330 - teacher	0.6369 - unrequited	0.4663 - chucking
neighbour 3	0.6008 - schools	0.6360 - longing	0.4635 - measuringworth
neighbour 4	0.6005 - teachers	0.6081 - lover	0.4630 - profit/loss
neighbour 5	0.5764 - teacherage	0.6065 - affection	0.4588 - recalculation

Table 1: Closest 5 neighbours for the words 'école', 'amour' and 'fourchette'.

As table 1 shows, the neighbours are actually very good. Even 'fourchette' that may appear at first glance wrong is good indead 'fourchette' also means 'estimate' or 'range'.

As for the distance that was used, it is defined this way: $distance = 1 - similarity$. An argmax is then ran to recover the best possible neighbours for each word.

2.7 Apply your approach and comment on the performance of the translation.

The data used to test the preciseness of the translations is MUSE en-fr test samples which contains english words next to their french translation. A function was built on top to enable testing both $K = 5$ and $K = 1$.

```
CPU times: user 17min 19s, sys: 3min 31s,
↪  total: 20min 51s
Wall time: 17min 30s
0.692
```

Figure 10: Top 5 neighbours accuracy on a 1000 samples from the above mentioned dataset

in figure 10 the accuracy of translation taking into account the top 5 results is 69.2% which is not ground breaking but at the same time impressing knowing the model was trained fast.

```
times: user 17min 26s, sys: 3min 33s,
↪  total: 21min
Wall time: 17min 37s
0.498
```

Figure 11: Top 1 neighbours accuracy on a 1000 samples from the above mentioned dataset

While still good, the direct performance remains low with 49.8% accuracy with a top1 neighbour translation. This can partly be explained by the fact that some of the translations contain plurals and some singulars, which in our testing would be counted as false, even though the translation is correct. The actual accuracy is thus probably marginally better.

2.8 What is the limit of aligning two spaces at a sentence level? What do you suggest to improve the alignment, at a sentence level?

Doing a translation at a sentence level requires taking attention to the context to translate words. In fact, multiple words during a translation. As an example, "we love Paris" will be translated in French as "Nous aimons Paris", where "nous" needed "we", "aimons" needed both "we" and "love", and "Paris" needed "Paris". Furthermore, source and target languages may not share the same sentence structure. As an exemple, English sentences are formed as Subject Verb Object (SOV) while other languages such as Latin are SVO.

To best perform the translation of a sentence, we should consider the use of transformers, which can take attention to the entire statement during the translation and to the previously translated words.

2.9 Someone, in your company, asked you to do sentiment analysis on their dataset. Given a set of sentences $s_1, \ldots, s_N$, where s_i can be written in any language, what architecture would you use to have a vector representation of s_i?

A possible architecture to perform sentiment analysis given a set of sentences is the following. The architecture for representing sentences as vectors for sentiment analysis would involve the following components:

1. **Pre-trained mBERT model**: (transformer-based architecture), it can handle a variety of NLP tasks and handles different languages well.

2. **Tokenizer**: Convert the input sentences into a format the mBERT model can take as input. Tokenizing each sentence into individual words with specific token IDs.

3. **Attention masks**: Guide the BERT model which tokens to focus on and which to ignore (padding).
4. **Fine-tuning layer**: Dense layer with a softmax activation and the number of output neurons set to the number of classes in the target task.
5. **Classifier**: After the fine-tuning, use the encoded vectors and sentiment labels to train a classifier. this helps predict new sentences with sentiments as output. This classifier can be: a simple feed-forward NN with a hidden layer, or be more complex depending on the available resources.

All these components work together to encode the input sentences into fixed-length vectors and classify the sentiment of sentences. The Tokenizer and mBERT model enable to use a vector representation of S_i.

2.10 How would you do sentiment analysis across multiple languages in a domain specific context? Justify your approach step by step.

Performing sentiment analysis across multiple languages in a domain-specific context can be challenging due to the differences in word usage, grammar, and idiomatic expressions across languages. Although, there are several way to palliate those challenges, here's one of them:

1. Collect a text dataset in multiple languages that caters to the specific domain of interest. Includes labels and text samples.
2. Preprocess the dataset.
3. Translate the text samples in the dataset to a common language. This step ensures that the text samples can be processed using the same techniques regardless of the original language.
4. Use the tokenizer provided by the mBERT model pre-trained and mentioned in the previous question.
5. Create an attention mask to focus on the important tokens.
6. run the tokenized masked text through mBERT to obtain a vector for each text sample.
7. fine-tune the model with the right number of classes to predict the sentiment of the text samples. To improve the performance, different models can be trained on different languages, and then use ensemble methods like majority voting, weighted voting, etc. to combine the results of different models into a single prediction.

This approach utilizes the power of pre-trained multilingual models to encode the text into fixed-length vectors, regardless of the original language of the text. Translating the text to a common language and fine-tuning the model on domain-specific data allows the model to understand the specific context and sentiment of the text. Using ensemble methods can further improve the performance. This approach can be fine-tuned and improved by experimenting with different architectures and models and using other techniques such as transfer learning.

3 Attention Exploration

3.1 Describe (in one sentence) what properties of the inputs to the attention operation would result in the output c being approximately equal to v_j for some $jj \in \{1, ..., n\}$. Specifically, what must be true about the query q, the values $\{v_1, ..., v_n\}$ and/or the keys $\{k_1, ..., k_n\}$

In order for $c = \sum_{i=1}^{n} \alpha_i v_i \approx v_j$, we must have, $\forall i \neq j$:

$$\begin{aligned} \alpha_i &\approx 0 \\ \Leftrightarrow \frac{\exp k_i^T q}{\sum_{i=1}^{n} \exp k_i^T q} &\approx 0 \\ \Leftrightarrow k_j^T q &\gg k_i^T q \end{aligned}$$

This is equivalent to say that the query must be as collinear in the same direction ("similar") to k_j as possible, and as collinear in opposite direction ("dissimilar") or orthogonal ("unrelated") to other keys as possible. Indeed, if the query is very similar to the k_j key and very dissimilar or unrelated to other keys, the attention output c could be almost identical to the corresponding *value* v_j, as if it was copied.

3.2 An average of two: Consider a set of key vectors $\{k_1, ..., k_n\}$, where all key vectors are perpendicular, that is $k_i \perp k_j$ for all $i \neq j$ Let $\|k_i\| = 1$ for all i. Let $\{v_1, ..., v_n\}$ be a set of arbitrary value vectors. Let $v_a, v_b \in \{v_1, ..., v_n\}$ be two of the value vectors. Give an expression for a query vector q such that the output c is approximately equal to the average of v_a and v_b, that is, $\frac{1}{2}(v_a + v_b)$

We want $c = \sum_{i=1}^{n} \alpha_i v_i = \frac{1}{2}v_a + \frac{1}{2}v_b$

$$\Leftrightarrow \alpha_a \approx \alpha_b \approx \frac{1}{2}$$

$$\Leftrightarrow \exp(k_a^T q) \approx \exp(k_a^T q) \gg \sum_{i=1}^{n} \exp(k_i^T q), \forall i \notin \{a, b\}$$

We can take $q = k_a + k_b$. Since we know that all key vectors are perpendicular, $k_i^T q = 0, \forall i \notin \{a, b\}$.

But this would result to :

$$\alpha_a = \alpha_b \approx \frac{\exp(1)}{2\exp(1) + (n-2)} \approx \frac{2.7}{5.4 + (n-2)}$$

If n is large, α_a and α_b will be significantly smaller than $\frac{1}{2}$. Therefore, we need $\|q\|$ to be a large scalar : $q = \beta(k_a + k_b)$, where $\beta \gg 0$ (large scalar multiple).

We could also substract the other keys in the linear combination, so that their scalar product with q is negative, making them less impactful in the softmax : $q = \beta(k_a + k_b - \sum_{i \notin \{a,b\}}^{n} {}_{i=1,} \; k_i)$

3.3 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 $\{k_1, \ldots, k_n\}$ that are now 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. Further, assume that the means μ_i are all perpendicular; $\mu_i^\top \mu_j = 0$ if $i \neq j$, and unit norm, $\|\mu_i\| = 1$.

3.3.1 Assume that the covariance matrices are $\Sigma_i = \alpha I$ 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.

Since α is *vanishingly small*, $k_i \approx \mu_i$. Therefore, we can keep our expression $q = \beta(k_a + k_b)$, where $\beta \gg 0$.

3.3.2 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 $\Sigma_a = \alpha I + \frac{1}{2}(\mu_a \mu_a^\top)$ for vanishingly small α. Further, let $\Sigma_i = \alpha I$ for all $i \neq a$. When you sample $\{k_1, \ldots, k_n\}$ multiple times, and use the q vector that you defined in part i., what qualitatively do you expect the vector c will look like for different samples?

We know that $\mu_a = 1$, and that α is vanishingly small. Therefore :

$\Sigma_a = \alpha I + \frac{1}{2}(\mu_a \mu_a^\top) = \frac{1}{2}$

Therefore, $k_a = \epsilon \mu_a$, with $\epsilon \sim N(1, \frac{1}{2})$.

We know that $k_i^T q = 0, \forall i \notin \{a, b\}$ and $k_b^T q = \beta \gg 0$. $k_a^T q$ is equal to $\epsilon\beta \gg 0$. We can now give an expression of c :

$$
\begin{aligned}
c &\approx \frac{\exp(k_a^T q)}{\exp(k_a^T q) + \exp(k_b^T q)} v_a + \frac{\exp(k_b^T q)}{\exp(k_a^T q) + \exp(k_b^T q)} v_b \\
&= \frac{\exp(\epsilon\beta)}{\exp(\epsilon\beta) + \exp(\beta)} v_a + \frac{\exp(\beta)}{\exp(\epsilon\beta) + \exp(\beta)} v_b \\
&= \frac{1}{1 + \exp((1-\epsilon)\beta)} v_a + \frac{1}{\exp((\epsilon-1)\beta) + 1} v_b
\end{aligned}
$$

3.4 Now we'll see some of the power of multi-headed attention. We'll consider a simple version of multi-headed attention

multi-headed attention 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(c), consider a set of key vectors $\{k_1, \ldots, k_n\}$ 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; $\mu_i^\top \mu_j = 0$ if $i \neq j$, and unit norm, $\|\mu_i\| = 1$.

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

Since α is *vanishingly small*, $k_i \approx \mu_i$. Therefore, $q_1 = q_2 = \beta(k_a + k_b)$, where $\beta \gg 0$.

3.4.2 Assume that the covariance matrices are $\Sigma_a = \alpha I + \frac{1}{2}(\mu_a \mu_a^\top)$ 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? Please briefly explain why. You can ignore cases in which $q_i^T k_a < 0$.

We know that $\mu_a = 1$, and that α is vanishingly small. Therefore :

$\Sigma_a = \alpha I + \frac{1}{2}(\mu_a \mu_a^\top) = \frac{1}{2}$

Therefore, $k_a = \epsilon \mu_a$, with $\epsilon \sim N(1, \frac{1}{2})$.

We know that $k_j^T q_i = 0, \forall j \notin \{a, b\}$ and $i \in$ 1,2 and $k_b^T q_i = \beta \gg 0$. $k_a^T q$ is equal to $\epsilon\beta \gg 0$. We can now give an expression of each c :

$$
\begin{aligned}
c &\approx \frac{\exp(k_a^T q_i)}{\exp(k_a^T q_i) + \exp(k_b^T q_i)} v_a + \frac{\exp(k_b^T q_i)}{\exp(k_a^T q_i) + \exp(k_b^T q_i)} v_b \\
&= \frac{\exp(\epsilon\beta)}{\exp(\epsilon\beta) + \exp(\beta)} v_a + \frac{\exp(\beta)}{\exp(\epsilon\beta) + \exp(\beta)} v_b \\
&= \frac{1}{1 + \exp((1-\epsilon)\beta)} v_a + \frac{1}{\exp((\epsilon-1)\beta) + 1} v_b
\end{aligned}
$$

By having multiple attention head, we reduce the impact of the noise (ϵ will tend towards its expectation, μ_a). Therefore, the result of the attention head (the average of all c) will tend towards $\frac{1}{2}(v_a + v_b)$ as we add attention head (with an additional computational cost).

References

[1] CONNEAU, A., LAMPLE, G., RANZATO, M., DENOYER, L., AND JÉGOU, H. Word translation without parallel data. *arXiv preprint arXiv:1710.04087* (2017).