Report Lab 2 - Advanced NLP(3)

paul.messeant | paul.renoux | corentin.pion | matthieu.schlienger | nikoloz.chaduneli

I. Language Detection

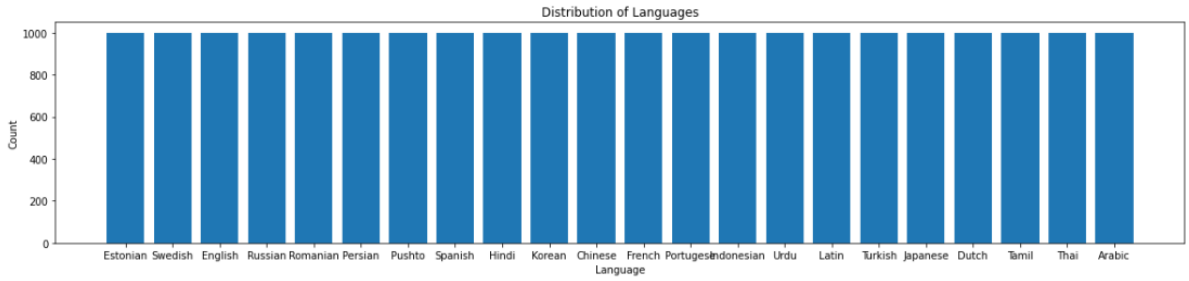
0. Try out a translation of a French sentence in Google Translate (or Bing Translate) to English. What happens if you select the wrong source language as follows? Explain in a few sentences what is happening in the backend.

When a user inputs text into Google Translate and selects the wrong source language, the system will still attempt to translate the text as if it were in the selected language. However, since the text does not match the chosen language model, the translation may not make sense or be very inaccurate, in most cases it is not even translated.

Google Translate uses a technique called "Statistical Machine Translation" (SMT) which is based on statistical models that are trained on large parallel corpora of text. The system uses these models to determine the probability of a particular word or phrase in the source language corresponding to a particular word or phrase in the target language.

When a user inputs text and selects a source language, the system uses the corresponding language model to analyze the text. If the user selects the wrong source language, the system will still use the selected model but it will not match the text, leading to poor translations because the model is not trained on that text, resulting in many mistakes, grammatical errors and mistranslations.

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



- It seems that the distribution of languages is perfectly balanced, there are 1000 text extracts per language, for 22 different languages.
- There is 22000 rows in the dataset containing text with differents encodings

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

There are several pre-processing techniques that can be used to improve the accuracy of language detection.

1. Text cleaning: Removing any special characters, numbers, and non-letters from the text can help to improve the accuracy of language detection. This is because special characters, numbers and non-letters can be present in any language, and so removing them can help to focus the analysis on the letters, which are more indicative of the language.
2. Text normalization: Converting all text to lowercase or uppercase can also help to improve the accuracy of language detection. This is because many languages have distinct letters or characters that only appear in uppercase or lowercase, and so normalizing the text can help to focus the analysis on these characters.
3. Encoding filter: Some languages use characters or scripts that are not used in other languages, such as Arabic or Chinese scripts. By removing these characters from the text, the analysis can be focused on the characters that are more likely to be indicative of the language. For instance, an encoding filter can be used to remove any non-unicode characters that are not used in any of the languages you are trying to detect. This can help to reduce noise in the detection process and improve the overall accuracy of the language detection.

```python
def pre_process(text):
    # Convert text to Unicode
    text = text.encode('utf-8').decode('utf-8')

    # Remove punctuation and special characters
    text = re.sub(r'[^\w\s]', '', text)

    # Convert to lowercase
    text = text.lower()

    return text
```

3. What would be the problem if your dataset was unbalanced?

An unbalanced dataset for a language recognition model can lead to the model being biased towards the majority class, overfitting to the majority class, inaccurate evaluation metrics and lack of diverse representation. This can lead to poor performance on minority languages and lack of ability of recognizing different languages.

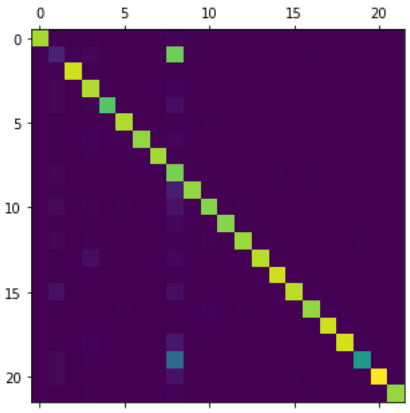
4. What techniques could you use to solve that?

To overcome these problems, techniques like oversampling or undersampling, synthetic data generation or collecting more data from minority languages can be used.

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.

```python
preprocX = X.apply(pre_process)

# preprocessing: tokenize text and build the vocabulary
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(preprocX)

# split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# train a logistic regression model
clf = LogisticRegression()
clf.fit(X_train, y_train)

# evaluate the model on test data
y_pred = clf.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred, average='micro'))
print("Recall:", recall_score(y_test, y_pred, average='micro'))

plt.matshow(confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.9034090909090909
Precision: 0.9034090909090909
Recall: 0.9034090909090909
<matplotlib.image.AxesImage at 0x7fb713da85d0>
```



The accuracy, the precision and the recall are actually really good. Logistic regression is a good choice for multiclass language detection problems because it can handle multiple classes using one-vs-all(OvA) or one-vs-rest strategy, it's a simple and interpretable algorithm that can handle a large number of features and can handle categorical, binary and numerical data, which is useful for text-based language detection problems. It is of course absolutely not the best algorithm because it is initially design for only two classes problems.

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

```
# your implementation goes here
model = fasttext.train_supervised(input="train.txt")
################################################

# @TODO: Save your model when trained
# model.save_model("langdetect.bin")
```

```
Read 89M words
Number of words:  4178805
Number of labels: 415
CPU times: user 1h, sys: 6.39 s, total: 1h 7s
Wall time: 1h 21s
Progress: 100.0% words/sec/thread:  124313 lr:  0.000000 avg.loss:  0.125459 ETA:   0h 0m 0s
```

```
# your implementation goes here
_, precision, _ = model.test("valid.txt")

print("Accuracy: ", precision)
################################################
```

```
Accuracy:  0.9545
```

The accuracy is really good with fastext.

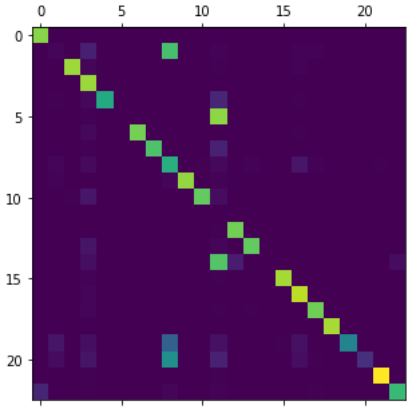
7. Test your fasttext model on the same dataset as in question 1-5. Compare with your custom model (make sure you use the exact same data for testing). How can you explain the difference in performance between the two models?

```
isolabels = {
    'Arabic' : "ara",
    'Chinese' : "cmn",
    'Dutch' : "nld",
    'English' : "eng",
    'Estonian' : "est",
    'French' : "fr",
    'Hindi' : "hin",
    'Indonesian' : "ind",
    'Japanese' : "jpn",
    'Korean' : "kor",
    'Latin' : "lat",
    'Persian' : "pes",
    'Portugese' : "por",
    'Pushto' : "pus",
    'Romanian' : "ron",
    'Russian' : "rus",
    'Spanish' : "spa",
    'Swedish' : "swe",
    'Tamil' : "tam",
    'Thai' : "tha",
    'Turkish' : "tur",
    'Urdu' : "urd"
}

X = data['Text']
preprocX = X.apply(pre_process)

# split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

file = open(filename, 'w')
for text, label in zip(X_test, y_test):
    file.write(f"__label__{isolabels[label]} {text}\n")
file.close()

_, precision, _ = model.test(filename)

print("Accuracy: ", precision)
```

```
Accuracy:  0.7897240723120837
```

```
##########################################
# your implementation goes here
defaultlabels = dict([reversed(i) for i in isolabels.items()])
y_pred = [defaultlabels.get(model.predict(X_test.iloc[i])[0][0][-3:], "None") for i in range(len(X_test))]

import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix

# confusion_matrix funnction a matrix containing the summary of predictions
plt.matshow(confusion_matrix(y_test, y_pred))

from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred, target_names=languages+["None"]))
##############################################
```

	precision	recall	f1-score	support
Estonian	0.89	1.00	0.94	193
Swedish	0.10	0.02	0.03	197
English	1.00	0.97	0.98	208
Russian	0.64	1.00	0.78	200
Romanian	1.00	0.82	0.90	176
Persian	0.00	0.00	0.00	196
Pushto	1.00	0.97	0.98	192
Spanish	1.00	0.88	0.94	192
Hindi	0.29	0.81	0.43	182
Korean	1.00	0.97	0.98	204
Chinese	0.99	0.89	0.94	202
French	0.00	0.00	0.00	0
Portugese	0.91	0.99	0.95	186
Indonesian	0.98	0.91	0.95	196
Urdu	0.00	0.00	0.00	207
Latin	1.00	0.99	0.99	206
Turkish	0.84	0.99	0.91	215
Japanese	0.97	0.97	0.97	190
Dutch	1.00	1.00	1.00	204
Tamil	1.00	0.47	0.64	224
Thai	1.00	0.16	0.28	203
Arabic	1.00	0.99	0.99	237
None	0.95	0.84	0.89	190
accuracy			0.75	4400
macro avg	0.76	0.72	0.72	4400
weighted avg	0.80	0.75	0.75	4400



We can see that the scores are a bit worse than the first method, the scores are very estrogenic. Indeed, there is few languages with really bad understanding (swedish, persian, hindi, urdu). It seems that there is an issue about the French language because it is not recognized by fasttext.

This scores could be understood by the training which is very different. Fasttext is trained to recognize a lot more different languages, we have only 22 of them, the preprocessing should be a bit different to prepare the data for fasttext according to the documentation.

8. Compute your performance metrics yourself and compare with sklearn.

```python
def precision(y_test, y_pred):
    n_classes = len(set(y_test))
    true_positives = [0.00001 for _ in range(n_classes)]
    false_positives = [0.00001 for _ in range(n_classes)]
    for i in range(len(y_test)):
        if y_pred[i] == y_test[i]:
            true_positives[languages.index(y_pred[i])] += 1
        else:
            if (y_pred[i] != 'None'):
                false_positives[languages.index(y_pred[i])] += 1
    precision_list = [true_positives[i]/(true_positives[i]+false_positives[i]) for i in range(n_classes)]
    return sum(precision_list)/n_classes

print("Precision: ", precision(y_test.tolist(), y_pred))

from sklearn.metrics import precision_score

print(f"Sklearn Precision: {precision_score(y_test, y_pred, average='micro')}")
```

```
Precision:  0.8436329278476098
Sklearn Precision: 0.7545454545454545
```

```python
def recall(y_test, y_pred):
    n_classes = len(set(y_test))
    true_positives = [0 for _ in range(n_classes)]
    false_negatives = [0 for _ in range(n_classes)]
    for i in range(len(y_test)):
        if y_pred[i] == y_test[i]:
            true_positives[languages.index(y_test[i])] += 1
        else:
            false_negatives[languages.index(y_test[i])] += 1
    recall_list = [true_positives[i]/(true_positives[i]+false_negatives[i]) for i in range(n_classes)]
    return sum(recall_list)/n_classes

print(recall(y_test.tolist(), y_pred))

from sklearn.metrics import recall_score

print(f"Sklearn Recall: {recall_score(y_test, y_pred, average='micro')}")
```

```
0.7557135909809894
Sklearn Recall: 0.7545454545454545
```

On the first hand, the precision score is not exactly the same with both methods, it is probably due to the non-recognition of the language in certain cases.

On the other hand, the recall is quite good.

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

We should train the fasttext model only on our languages to improve the model performance.

We could also increase the amount of training data, tuning the model's hyperparameters or maybe use pre-trained embeddings.

10. Which method would you use for language detection and why?

The choice of the language detection model is highly dependent on the problem. In the case of a detection with a lot of different languages, fasttext is probably the most suitable because it is trained for. If you are dealing with a case with few different languages, other algorithms could do the job very well as they are much lighter, logistic regression should be good if there is only two languages, multinomial algorithms, k-Nearest Neighbors, Decision Trees could work well for more.

11. Given a sentence with $N1$ tokens in English and $N2$ token in French, what would be your strategy to assign a language to such sentence?

If the two values N1 and N2 are far apart, then we could argue that the larger number is more likely to be the represented language. On the other hand, if the values are very close, one strategy to assign a language to a sentence with a mix of English and French tokens could be to create a custom model with a large dataset of labeled sentences in both English and French, train a multi-class classifier that can predict the language of a given sentence, and use it for out sentence. If the classifier is not confident in its prediction, we could use a second level of analysis such as tokenizing the sentence, and analyze each token in isolation and decide the language based on frequency of languages or by using N-Gram based detection.

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

A multilingual architecture, such as a multi-language transformer model like the mBART or XLM-Roberta, would be more robust to handling multiple languages in a single sentence than a single-language model. These models are pre-trained on a large corpus of text from multiple languages, so they have the ability to understand and generate text in multiple languages.

However, these models are usually trained to be best at handling one or a few languages at a time. They are able to handle multiple languages in the same sentence, but it is still not a trivial task and the performance may be not as good as using them to only one language. The model will predict the most likely language based on the input and make a decision based on that, but it can still have mistakes, specially if the text is written in a code-switching manner.

When it comes to assigning a specific language to a sentence with a mix of languages, a multilingual model could make the decision based on the probability scores for each language, but the performance may not be as good as a specialized model that was trained specifically for that task.

II. Rotate two semantic spaces

1. Explain in a few sentences how MUSE1 is doing the alignment of the semantic spaces in the supervised way.

MUSE (Multilingual Unsupervised and Supervised Embeddings) is a tool for creating cross-lingual word embeddings, which are mathematical representations of words in a multi-dimensional space such that words with similar meanings are located in close proximity to one another. MUSE uses a supervised approach to aligning the semantic spaces of multiple languages. This is done by using a dictionary to establish correspondences between words in different languages and then training embeddings for each language on the same input data. The embeddings are then aligned by using the dictionary correspondences to find the best linear transformation that maps one embedding space to the other, so that the nearest neighbors of a word in one language are close to its translations in the other languages.

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

The main limitation of the supervised approach used by MUSE for aligning semantic spaces is that it requires a pre-existing dictionary that contains correspondences between words in different languages. The quality and coverage of the dictionary can have a significant impact on the quality of the aligned embeddings. If the dictionary is incomplete or contains errors, the alignments will also be incomplete or incorrect. Additionally, the alignment can only be performed between the languages that are included in the dictionary, so if you are working with an under-resourced language or a new language that has no pre-existing dictionary then the supervised approach would not be possible.
Another limitation is that the approach relies on a linear transformation of the embeddings, which can limit the ability of the model to capture the complex relationships between words across different languages. In particular, it might not generalize well for low-resource language or distant languages with different structures,
Finally, the approach would not be able to discover new relationships or representations that are not present in the training data and dictionary.

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

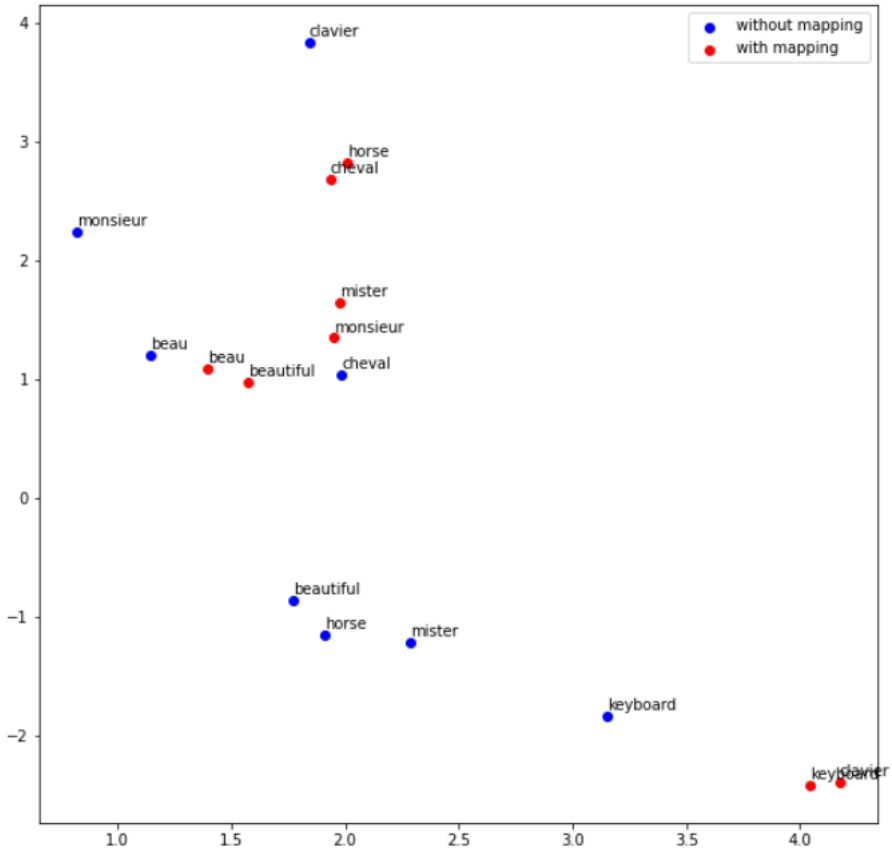
In a domain-specific field such as a tech company, one way to align semantic spaces across languages would be to create a specialized dictionary of terms that are specific to that field. This dictionary would contain correspondences between words and phrases in different languages that are commonly used within the company. This approach is similar to the one used by MUSE, but the specialized dictionary would be created specifically for the company and would reflect the specific terms and jargon used within the field.

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

```
[8] !python unsupervised.py --src_lang fr --tgt_lang en --src_emb data/wiki.fr.vec --tgt_emb data/wiki.en.vec --n_epochs 1 --exp_name "fr_en" --exp_path "./" --export ""
```

As we saw previously, supervised approach is limited because of the pre-existing dictionary required. So we used an unsupervised method with one epoch which is in off because several epochs is too long. We choose french and english fasttext Wikipedia embeddings for the source and the target embeddings.

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



We are going to visualize few words by mapping them. Before mapping, we can identify two words clusters, an English one and a French one. After mapping, words from both languages with the same meanings are regrouped. With this observation we can verify that the alignement works.

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.

To find the translation of a word you have to map the source language words embeddings and the target language words embeddings. Then look for the nearest target word embedding of the source word embedding and choose it as the translation.

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

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?

Aligning semantic spaces at the sentence level, as opposed to the word level, can be more challenging because sentences can convey more complex meanings and may include idiomatic expressions, figurative language, and other nuances that are difficult to capture in a simple linear transformation.

One way to improve the alignment at the sentence level is to use additional information to disambiguate the meaning of sentences. For example, using additional context from the surrounding sentences or external knowledge base, can help disambiguate the meaning of sentences and improve the alignment.

9. Someone, in your company, asked you to do sentiment analysis on their dataset. Given a set of sentences {*s*1, … , *sN*}, where *si* can be written in any language, what architecture would you use to have a vector representation of *si*? Motivate in 2-3 bullet points.

For my favorite colleague i would use an architecture with a pre-trained language model such as BERT, GPT-2, RoBERTa because they are trained on a massive amount of text data and can be fine-tuned on a specific task such as sentiment analysis. (We can also use fasttext or any other RNN or CNN specialized in this task.)

- Transfer Learning: PLMs are trained on a massive amount of text data, which means they have already learned a lot of general information about the structure of language. This information can be used to improve performance on the sentiment analysis task by fine-tuning the PLM on a smaller dataset of labeled sentiment data.
- Better performance: they have been shown to achieve state-of-the-art performance on a wide variety of natural language processing tasks, including sentiment analysis. Using a pre-trained model can lead to improved accuracy and precision.
- Handling Out-of-Vocabulary words: PLM like BERT are designed to handle words not seen in the training data through their subword tokenization technique which is capable of handling infrequent and rare words effectively. This is useful in sentiment analysis task as one of the common problem is to classify the sentiment of a product review with reference to the product. So, PLM should be able to handle the words which refers to the product and might not be in the training dataset.

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

1. Identify the domain specific context: Before starting the sentiment analysis, it is important to identify the domain specific context in which the analysis will be conducted. For example, the context could be customer reviews of a hotel or restaurant, or comments on a news article about a specific topic.
2. Collect and preprocess the data: Collect a large amount of data in multiple languages that is relevant to the domain specific context. Preprocess the data by cleaning and normalizing it, removing any irrelevant information and formatting it in a way that can be easily analyzed.
3. Use machine learning techniques: Use machine learning techniques such as natural language processing and deep learning to analyze the data. These techniques can be trained on a large amount of labeled data to accurately identify the sentiment of the text.
4. Use language-specific models: Use language-specific models for each language in the analysis. For example, if the analysis includes both English and Spanish, use a model trained on English data for the English text and a model trained on Spanish data for the Spanish text.
5. Evaluate and refine the models: Evaluate the accuracy of the models on a separate test dataset and refine the models as needed. Repeat the process of collecting, preprocessing, analyzing and refining the models until the desired level of accuracy is reached.

III. Attention Exploration

1. Copying in attention

$\sum_{i=1} \alpha_i v_i = v_j \implies \alpha_i = \{1 \text{ if i =j and } 0 \text{ else}\} \implies k_i = -N(1, 1..)_d \; i \neq j, N \in \mathbb{N}$. Bigger the N, better the approximation.

2. An average of two

$\sum_i \alpha_i v_i = \frac{1}{2}(v_a + v_b) \implies \alpha_i = \frac{1}{2}$ for i = a or b and 0 else.
Since $k_i \perp k_j$, we set $q = k_a + k_b$ so that $k_i^T q = 0 \; \forall i \neq a, b$ and we multiply the result by a large number N so that $\alpha_a = \alpha_b \approx \frac{1}{2}$

3. Drawbacks of single-headed attention

i.

Covariances being close to zero, $k_i \approx \mu_i$.
So we set $q = \mu_a + \mu_b \implies k_i^T q \approx \mu_i^T q = 0 \; \forall i \neq a, b$
as the mean vectors are orthogonal. Again, multiplying with a large N (q' = Nq) inside the softmax yealds a

ii.

$$\|k_a\| \ll \|\mu_a\| \implies c \approx \frac{\exp(k_a^T q)}{\exp(k_a^T q) + \exp(k_b^T q)} v_a + \frac{\exp(k_b^T q)}{\exp(k_a^T q) + \exp(k_b^T q)} v_b = (1) \frac{\exp(\epsilon * N)}{\exp(\epsilon * N) + \exp(N)} v_a + (2) \cdot$$

for a small ϵ and a large N. (1) negligible with respect to (2).

Similarly, $\|k_a\| \gg \|\mu_a\| \implies c \approx v_a$ as (2) becomes negligible w/r (1)

4. Benefits of multi-headed attention

i.

We need $c_a \approx v_a$ and $c_b \approx v_b$. We can achieve this by setting $q_1 = N * \mu_a$ and $q_2 = N * \mu_b$. We get $c_1 \approx \frac{\exp(N\mu_a^T q_1)}{\exp(N\mu_a^T q_1)+n-1} v_a = \frac{\exp(N)}{\exp(N)+n-1} v_a \approx v_a$ and $c_2 = \approx \frac{\exp(N\mu_b^T q_1)}{\exp(N\mu_b^T q_1)+n-1} v_a = \frac{\exp(N)}{\exp(N)+n-1} v_b \approx v_b$.

ii.

Looking back at the two cases for the norm of k_a from question 3.ii. there was an exp(N) term in the denom that was much larger than the $\exp(\epsilon N)$ term which in term made the fraction negligible.
We won't have this problem here with c_1 as the q_1 doesn't have the μ_b part.
No matter the magnitude of k_a, c will always be approximately $\frac{1}{2}(v_a + v_b)$.

IV. Neural Machine Translation

1. Why might it be important to model our Cherokee-to-English NMT problem at the subword-level vs. the whole word-level?

In polysynthetic languages, like Cherokee, words can be extremely long and expressive, containing a large amount of grammatical information encoded into a single word. These word structures can be complex to capture and generalize in a word-level model, as there is limited number of examples for each word in the training data.

On the other hand, representing these words as sequences of subword units allows the model to effectively learn the underlying structure of the language and generate more fluent translations. Additionally, representing the language at the subword level also mitigates the issues with out of vocabulary words, as the embeddings for subwords can be shared across multiple words.

Additionally, subword representations often work better in low-resource settings, which is often the case when working with endangered or minority languages.

2. Character-level and subword embeddings are often smaller than whole word embeddings. In 1-2 sentences, explain one reason why this might be the case

One reason why character-level and subword embeddings are often smaller than whole word embeddings is because they typically use a smaller vocabulary. Instead of representing every word in the language with a unique embedding, subword embeddings share embeddings for common subword units, which reduces the dimensionality of the embedding matrix. This in turns allows for a smaller overall embeddings size.

3. One challenge of training successful NMT models is lack of language data, particularly for resource-scarce languages like Cherokee. One way of addressing this challenge is with multilingual training, where we train our NMT on multiple languages (including Cherokee). You can read more about multilingual training here. How does multilingual training help in improving NMT performance with low-resource languages?

Multilingual training can be useful for improving NMT performance with low-resource languages, such as Cherokee, because it allows the model to leverage information from other languages to learn better representations for the target language.

When training an NMT model on multiple languages, the model learns to identify and utilize common patterns across languages, which can help to improve its ability to understand and generate text in the target language, even when it has limited training data. This way, the model can generalize better across languages and use the knowledge of other languages to learn better representations for the low-resource language.

The encoder-decoder architecture of NMT models is often language agnostic, meaning that it doesn't require specific features like stem, declensions etc. This property of the NMT models allows the model to learn from multiple languages, which in turn, can improve its performance on low-resource languages.

4. Here we present three examples of errors we found in the outputs of our NMT model (which is the same as the one you just trained). The errors are underlined in the NMT translation sentence.

For each example of a source sentence, reference (i.e., 'gold') English translation, and NMT (i.e., 'model') English translation, please:
1. Provide possible reason(s) why the model may have made the error (either due to a specific linguistic construct or a specific model limitation).
2. Describe one possible way we might alter the NMT system to fix the observed error. There are more than one possible fixes for an error. For example, it could be tweaking the size of the hidden layers or changing the attention mechanism.

Source Translation: *Yona utsesdo ustiyegv anitsilvsgi digvtanv uwoduisdei.*

Reference Translation: *Fern had a crown of daisies in her hair.*

NMT Translation: *Fern had her hair with her hair.*

The first error may be due to the model not understanding an metaphor like "crown of daisies" for this context. The best solution to resolve this issue would be to get more data to train our model.. It can be hard for our language since it's not really used but it could impove our resultat for other

Source Sentence: *Ulihelisdi nigalisda.*

Reference Translation: *She is very excited.*

NMT Translation: *It's joy.*

In this exemple, the model could not comprehend the subject in the sentence. Maybe our model is just to simple to understand a language as complex as the cherokee. Like the first one we could add more data or improve our model by adding more hidden layer so our model could comprehend the grammar of our language

Source Sentence: *Tsesdi hana yitsadawoesdi usdi atsadi!*

Reference Translation: *Don't swim there, Littlefish!*

NMT Translation: *Don't know how a small fish!*

This error is due to the fact that the model can't see that "Littlefish" is a proper name.

5. Please compute the BLEU scores for c1 and c2. Let $\lambda i = 0.5$ for $i \in \{1, 2\}$ and $\lambda i = 0$ for $i \in \{3, 4\}$ (this means we ignore 3-grams and 4-grams, i.e., don't compute p3 or p4). When computing BLEU scores, show your working. Which of the two NMT translations is considered the better translation according to the BLEU Score? Do you agree that it is the better translation?

First of all, we have implemented a blue function that calculate our score

```python
def bleu(candidate, references, n, weights):
    """
    Calculates the BLEU score for a candidate translation and a list of reference translations
    candidate : string : the candidate translation
    references : list of strings : the reference translations
    n : int : the maximum n-gram length to use in the calculation
    weights: list of float: the weight distribution for different n-grams
    """
    precisions = []
    for i in range(1, n+1):
        candidate_ngrams = Counter(ngrams(candidate, i))
        reference_ngrams = [Counter(ngrams(ref, i)) for ref in references]
        overlap_counts = [min(candidate_ngrams[candidate_n_gram], ref[candidate_n_gram]) for ref in reference_ngrams for candidate_n_gram in candidate_ngrams]
        precisions.append(sum(overlap_counts)/len(candidate))
    precisions = [min(1, p) for p in precisions]
    # Brevity Penalty
    reference_length = min(len(ref) for ref in references)
    candidate_length = len(candidate)
    if candidate_length > reference_length:
        bp = 1
    else:
        bp = math.exp(1 - reference_length / candidate_length)
    print("P0 =",precisions[0])
    print("P1 =",precisions[1])
    print("len(c) =",candidate_length)
    print("len(r) =", reference_length)
    print("BP =", bp)
    return bp * math.exp(sum(w*math.log(p) for w, p in zip(weights, precisions))/sum(weights))
```

Conserning our result, we achieved this:

```python
candidate1 = " the love can always do"
candidate2 = " love can make anything possible"
references = [ "love can always find a way" , "love makes anything possible" ]
weights = [0.5, 0.5, 0, 0]
n = 4
score1 = bleu(candidate1, references, n, weights)
print(score1)
print("----------------------")
score2 = bleu(candidate2, references, n, weights)
print(score2)
```

```
P0 = 1
P1 = 1
len(c) = 23
len(r) = 26
BP = 0.8777137332821824
0.8777137332821824
----------------------
P0 = 1
P1 = 1
len(c) = 32
len(r) = 26
BP = 1
1.0
```

Both sentences have a good score but the 2nd candidates have a perfect score. Actually it could be logix since this sentence look a lot like “love makes anything possible”

ii. Our hard drive was corrupted, and we lost Reference Translation r2. Please recompute BLEU scores for c1 and c2, this time with respect to r1 only. Which of the two NMT translations now receives the higher BLEU score? Do you agree that it is the better translation?

Now we can see that the first candidate has a better result, like the last question the first candidate seems similar.

```
candidate1 = " the love can always do"
candidate2 = " love can make anything possible"
references = [ "love can always find a way"  ] #, "love makes anything possible"
weights = [0.5, 0.5, 0, 0]
n = 4
score1 = bleu(candidate1, references, n, weights)
print(score1)
print("----------------------")
score2 = bleu(candidate2, references, n, weights)
print(score2)
```

```
PO = 0.8260869565217391
P1 = 0.6521739130434783
len(c) = 23
len(r) = 26
BP = 0.8777137332821824
0.6442397056412241
----------------------
PO = 0.59375
P1 = 0.3125
len(c) = 32
len(r) = 26
BP = 1
0.4307515235028194
```

We could agree with this result

iii. Due to data availability, NMT systems are often evaluated with respect to only a single reference translation. Please explain (in a few sentences) why this may be problematic.

Evaluating machine translation systems using a single reference translation can be problematic because it may not fully reflect the variability and nuances of human-produced translations. In many cases, there may be multiple valid translations for a given source sentence, and using only one reference translation may lead to an incomplete or skewed understanding of a system's capabilities. Additionally, evaluating against a single reference may also lead to models being over-fit to that particular translation, reducing its ability to generalize to other translations.

iv. List two advantages and two disadvantages of BLEU, compared to human evaluation, as an evaluation metric for Machine Translation.

Advantages of BLEU compared to human evaluation as an evaluation metric for machine translation:

1. Speed and cost: BLEU is an automated metric, which can be calculated quickly and inexpensively, whereas human evaluation is a time-consuming and expensive process.
2. Objectivity: BLEU aims to remove the subjectivity of human evaluation by providing a numerical score that can be easily compared across different systems and runs.

Disadvantages of BLEU compared to human evaluation as an evaluation metric for machine translation:

1. Limited scope: BLEU primarily measures the degree of lexical and grammatical similarity between the machine-generated and reference translations, but it does not take into account other factors such as meaning, fluency, or adequacy of the translations.
2. Over-reliance on word-for-word matching: BLEU scores are based on word n-gram overlaps between the machine-generated and reference translations, which may not always be an accurate measure of the quality of a translation. This can lead to the model being over-fit on the reference, therefore a good BLEU score does not necessary mean a good machine translation.