Module 4 - Natural Language Processing:

Group: 12
Anton Claesson, 971104-3217, MPDSC, canton@student.chalmers.se, 22~h
Lukas Martinsson : 980203-9678, MPDSC, malukas@student.chalmers.se, ~20h

We hereby declare that we have both actively participated in solving every exercise. All solutions are entirely our own work, without having taken part of other solutions

Reading and reflection: Anton

a) Naturally, there would be several similar related problems in NLP that would have been following the development of the automatic translation problem in terms of different approaches applied to them.
However in non-NLP terms the most comparable domain is computer vision which is another “holy grail” of AI research, in the pursuit to build machines that could “mimic” the human vision perception system. For example, fingerprint identification and face detection/identification are problems that have seen many kinds of approaches and a similar development (rule based and statistical approaches to neural networks) throughout the years.

b) The Interlingual Machine Translation approach is in many ways similar to embeddings that modern neural systems use, with the most prominent difference being that neural systems automatically create these intermediate representations.

c) Old school rule-based solutions are more interpretable and understandable, i.e. one can predict and explain exactly what the system is doing. Moreover they don’t rely on large amounts of data to work. So in cases where explainability and predictability of the system is key and we don’t have access to vast amounts of data, rule based approaches might be favorable. One example might be when building a system for translating programming code snippets in one language to another. For example it is probably favorable to utilize a rule-based approach for converting for-loop syntax in Java to Python.

Reading and reflection: Lukas

a) Although it has probably not had as diverse approaches as NLP I would assume recommender systems have advanced quite a bit since their start. Since only by a few simple categories one can make a basic recommender system and right now we have both content and collaborative filtering, which can also be used in tandem. Another more complicated problem is in regards to computer vision, however seeing as I do not have adequate knowledge this would be just a guess.

b) Whilst rather different in how to work they still identify some of the same structures in sentences, the difference being that state-of-the-art neural systems sees these as patterns whilst rule-based have been told this is the structure of the language.
c) The problem with modern solutions is that they are so complex, which means understanding the model's reasoning for choosing a specific translation is extremely hard. When utilizing more old school solutions it is easier to understand their reasoning. Hence when starting learning about NLP it is highly advantageous to understand the simpler models that a human can analyze first. Also since neutral systems require a much larger dataset and take a lot longer to train it makes simpler models more practical for the purpose of learning.

Implementation

We choose to translate Swedish into English.

Warmup

In order to count the 10 most frequent words in each language we used a Counter as adviced. These are the results:

- Swedish: ['.', 'att', ',', 'och', 'i', 'det', 'som', 'för', 'av', 'är']
- English: ['the', ',', '.', 'of', 'to', 'and', 'in', 'is', 'that', 'a']

Next the probability of a certain word is calculated as the number of occurrences of that word divided by the total occurrences of all words.

- 'speaker' - count: 10, probability: 0.0000355
- ‘zebra' - count: 0, probability: 0.0

Language modeling

In order to implement the Bigram model we first utilized a defaultdict containing Counters. So for each bigram ‘word1 word2’ we can obtain the count by indexing the defaultdict as [word1][word2]. Next, we converted these counts to probabilities stored in a defaultdict containing a defaultdict, where the default value returned would be zero. But in order to avoid later division-by-zero errors we use a small number to represent minimum probabilities rather than zero (sys.float_info.min = 2.2250738585072014e-308).

This default value handles the case where we use an out-of-vocabulary word. Moreover, since we are multiplying probabilities of each word in a sentence to compute the total sentence probability, when the sentence is long we are multiplying very small products which might lead to underflow. To avoid this, instead of multiplication we add the log probabilities of the words in the sentence together and use that instead. Examples:

[“i would like”] log prob: -2.956, prob: 0.052

[“i would like ”*50] log prob: -34859.24, prob: 0.0 (note: underflow)

Translation modeling

Let's denote the target language we wish to translate to as *t* (English) and source language we wish to translate from as *s* (Swedish).

- *If our goal is to translate from some language (s) into English (t), why does our conditional probability seem to be written backwards? Why don't we estimate p(t|s) instead?*

To answer this question here is a screenshot from our notebook (as it is hard to do equations in google docs):

From Bayes rule we know that

$$p(t|s) \propto_t p(s|t)p(t)$$

so our objective can be written as

$$t^* = \arg\max_{t \in T} p(t|s) = \arg\max_{t \in T} p(s|t)p(t)$$

This is suitable since we can use a language model *p(t)* trained from arbitrary target language text to consider grammar and fluency, while at the same time *p(s|t)* will handle the translation probabilities. Hence this divides the main problem, which is determining *p(t|s)* into two smaller subproblems. This division of problems also makes further analysis into how to improve the model easier, since it allows us to find if the eventual shortcomings that originates in *p(t)* which concerns the grammar and fluency and *p(s|t)*, the translation probability.

Next, we implemented IBM model 1 and the EM-algorithm. Again, we utilized defaultdicts and nested defaultdicts to represent the different probabilities and counts that are updated by the algorithm.

We print the top 10 most likely Swedish words the word “european” could be translated into at every iteration, by checking *t(s|t)* for every Swedish word.

We see that there is an improvement of the translation at each iteration. Here we provide the probabilities after the first and fifth iteration:

Iteration number and comment	**Top-10 word probabilities**
Iteration 1: After one iteration the translations seem pretty random. Only Swedish world that is somewhat similar to “european” is ‘europagrupp’, the 10th most likely word.	'lugnas' : 0.125 'flygsäkerheten' : 0.100 'sovjetunionen' : 0.100 'hörd' : 0.100 'skadat' : 0.091' ‘enhetsakten' : 0.091 'kärnpunkt' : 0.091 'attraktivt' : 0.083 'skikt' : 0.077 'europagrupp' : 0.077
Iteration 5: After five iterations the translations have become quite good. The top two most likely words can be considered to be direct translations to the word “european”.	'europeisk' : 0.225 'europeiska' : 0.211 'europaparlamentet' : 0.175 'lugnas' : 0.151 'europeiskt' : 0.118 'europaparlamentets' : 0.115 'sovjetunionen' : 0.114 'hörd' : 0.107 'csu' : 0.105 'unionen' : 0.100

Decoding

Finding the English sentence that has the highest probability in our model is a problem that has shown to be NP-complete; i.e. it is in a set of hard problems for which no polynomial time algorithm has been found. The reason being that the search space of possible translated sentences is combinatorial.

At first, we tried searching for decoding strategies and found that it is common to use stack decoding or beam search algorithms, which are both heuristic methods based on keeping a set of viable candidates of translation. However we were already out of time for this week and we therefore opted for a much simpler approach. We made the following simplifying assumptions:

1) The sentence to translate can only include words already in the corpus. One might for example also translate out of vocabulary words to an “unknown” word but we simply choose to avoid this problem.

2) Markov property: the next word depends only on the current word. Then *P(S|T),* the probability of translating to sentence *S* given sentence *T*, might be approximated by

$$P(S|T) \approx P(s_1|t_1) \cdot P(s_2|t_2) \cdot \ldots \cdot P(s_n|t_n)$$

Note that this assumption assumes that all our translations are the same length as the source sentence, an obvious drawback.

3) We assume that *P(s|t)* can be described with our IBM 1 model and *P(t)* can be described by our Bigram language model.

The algorithm we use for decoding works as follows:

1) For each word s_i in the source sentence (Swedish)
2) Find the most probable (English) target word t_i given s_i and choose this as the translation of this word. This gives $P(s_i|t_i)$.
3) Repeating 1-2 until the sentence is translated, we obtain *P(S|T)* from the equation above using the Markov assumption. We can then calculate *P(T)* using our bigram language model. The total probability (up to a constant) of the generated translation is then *P(S|T)P(T).*

Unfortunately, this evaluates only a single translation. One might however imagine improving this by picking a set of candidate translated words instead of just the best one, and thus creating a set of candidate translated sentences. By then calculating the total probability of all these candidates and picking the highest-probability one we might obtain better performance.

Due to time reasons and complexity of this we only implement this without considering multiple candidates. Here is an example sentence, it's translation and the probability of this translation:

Swedish: "*den socialistiska gruppen har begärt ett uttalande från kommissionen om dess strategiska mål för de fem kommande åren .*"
English: "*the socialist group have requested a statement from commission on its strategic objective for the five next years .*"
Log probability: -2206.35

The most obvious problem with this approach is that the language model is in essence only being used to rank the candidate translations. We are a bit unsure about how one could also utilize the language model in the process of picking the candidate translations. The solution would probably require us to look ahead at least for the next word such that our bigram language model could be used. We did not dive into the details of the more sophisticated

decoding algorithms mentioned (stack decoding, beam search) but it seems likely as this is something these algorithms have accounted for.

Discussion

a) Firstly, in regards to what makes a translation “good”. Depending on what weight one puts in the word “good” the answer can differ. If “good” is closer to “adequate” then “perfect” then it would probably be a translation which is grammatically somewhat correct and can convey the general message. However if one defines it closer to “perfect” it would probably be that the grammar and structure firstly is flawless. Secondly, and this would be much harder for the translation is to convey the correct “feeling/meaning” behind the word. There are several synonyms for a feeling and the proper word depends on the context. So a “perfect” translation would take this into account, so that it for example knew if it was correct to use formal words or informal, words that convey more emotion or more passively formulated words. Hence a “good” translation would probably lay somewhere between these two definitions, which is kind of interesting since the interpretation of the word good and the interpretation of words themselves determines the answer to both the question and the word itself.

- One manual procedure would be to let a human, preferably a linguistics expert evaluate the translations that the model makes. Either the expert can then give binary answers, that the translation is good or bad or give a broader range of answers. For example, whether the translation was perfect, adequate, bad or wrong. In the latter case if for example an expert would evaluate 100 words then each perfect score could result in 1%, each adequate in 0.75%, each bad in 0.25%, and each wrong in 0% addition to the total accuracy score. However, since a lot more than a 100 words would have to be evaluated for the model to get a fair assessment of its accuracy this would be expensive and time consuming. Furthermore, since it is a human who is evaluating the results it would be hard to replicate it with the same result if one wants to compare it to other models.
  Another interesting idé would be to use a test similar to the Turing test. Let a human get two translations, one from the machine and one from another individual and let he/she guess which one is from the AI. However this test does not really test how accurate the model is but instead how close the model's translation is to an actual human, which depending on the context might not be as relevant if one wants to compare different models.
- A more automatic approach would be to compare the model to either already established translation models (however this would probably be sub-optimal) or to already translated sentences that can be found for example on instruction manuals in different languages.The evaluation criteria here could

be similar to the 4 choice above, perfect, adequate, bad, or wrong. How those would be defined is harder to define than for the manual procedure however the bigram distribution would probably play a large part when analyzing how "correct" the translated sentence is.

b) Neural translation models (such as the one google translate uses) mirrors patterns and bias in the real world since it is trained on large quantities of text from the real world. Therefore, it is likely that the doctor or programmer occupations are associated with the masculine pronoun and vice versa for the other occupations. Additionally, since the English language does not have a gender neutral word it would be impossible for it to be correct all the time without additional context.
   The ability of these models to learn these types of patterns can be a feature in many cases, although in this case it might not be a desirable one. Hence, since the model's results depend on how it is designed the question that would be better to ask is "Do you consider this feature appropriate or not? However it would be interesting to see if in the translation to Swedish which has the word "hen" as a gender neutral word the translation model takes such words into account or not.

c) The problem here is that the english word "bat" has numerous swedish direct translations, so the correct one needs to be inferred from the context. It seems as if the model is able to correctly identify the correct word based on the context in the first two sentences; perhaps the usage of the word "bat" in similar contexts is relatively common in the training data. On the other hand, when the model fails to infer the correct word from the context, we get a nonsensical output which perhaps seemed as the most plausible one according to the model. The context could be confusing since the swedish word for baseball bat ("slagträ") might be linked to "wood" and therefore in turn "forest", but at the same time the animal "bat" ("fladdermus") would also be highly linked to be living somewhere, maybe even in the forest. This could lead to a confusion about which word is appropriate, and where a nonsensical mixture of words might be the best choice the model can make when translating.

Individual Summary and Reflection: Anton

Lecture summary (NLP)

The lecture first gave an overview to the history of the NLP domain and some classic problems within it. In particular, NLP can generally be divided into four common categories; categorization tasks, tagging tasks, parsing tasks and generation tasks. The lecturer talked a bit about how "traditional" NLP solutions in which a ML pipeline is used compares to modern neural based end-to-end systems, as well as some pros and cons of both approaches.

We then focused on the problem of machine translation, “the holy grail” of NLP and some history of this problem such as rule-based machine translation and the notion of interlinguism. Then a bit about statistical machine learning and modern neural translation. The rest of the lecture covered the statistical approach with IBM model 1 in more detail as this was what we were supposed to implement in the assignment, as well as the bigram language model.

Lecture summary (AI Tools follow-up)

In this lecture we first discussed some of the questions asked and problems encountered during the module and potential solutions for how to combat these. For example the problem of distributional shift was further explained and clarified. We also talked a bit about the implementation of the assignment.
In general the discussions seemed to be focused on problems related to the “technical debt” which had been one of the talking points during the module, i.e. problems that require special consideration in order to avoid unpredictable and expensive maintenance costs down the line.
Next, the lecture covered deployment of AI models and talked about some useful tools, strategies and commonly occurring problems when going from research use to production use. For example the importance of code and data versioning control software. Other important points to consider include creating backwards compatible AI systems and how unit tests, metrics, “explainable AI methods” and experiment tracking is useful to make sure our AI systems are functioning.
Finally, the lecture covered a bit about what hardware, and that this depends a lot on what the application is. For example when one uses CPU clusters or GPU clusters.

Reflection of previous module (AI Tools)

The AI tools module talked a lot about the practical problems of creating AI systems that are often overlooked when just considering the research use and development of the system. As a student with limited knowledge of implementing real world AI systems, it is easy to fall into the trap of forgetting the “technical debt” that could occur in the form of expensive maintenance costs after deploying a model without considering these production-time problems. The main take-away for me here has been that I should be aware of these risks and take measures for avoiding them during development. The module also covered some common tools, strategies and frameworks to developing different kinds of ML systems, but I was quite familiar with most of these since earlier so the thing I felt gave me the most new insight and perspective was the idea of hidden technical debt, as it was something I had not considered as much previously but I now realize is very important.

Individual Summary and Reflection: Lukas

Lecture summary (NLP)

The lecture covered useful areas to apply NLP to, among them were spam filters, machine translation as well as spell and grammar checkers. The lecture further covered the difficulties of NLP models. Firstly, words themselves are discrete in the sense that the words have no natural ordering based on the word itself, secondly they are varied since there are so many words in total, and several of which are synonyms and lastly that when you train your model on a dataset most words only occur a few times. Continuing on a history of NLP advancement and how to progress from using linear models to neural models which are used today was explained. Lastly IBM models (which we used in the assignment) were explained in depth. How through Bayes rule we can reformulat the goals of maximizing $P(E|F)$ to $P(E)\ P(f|E)$. This is better since it enables us to easier train a model and can divide the problems into two smaller subproblems. Then a bigram model was explained and why using the Markov property for a bigram model is crucial.

Lecture summary (AI Tools follow-up)

The follow up lecture firstly covered many of the idés that and problems arose during the assignment and discussion as well as patterns that many students missed during their explanation. For example if possible its highly effective to use math notation to convey one's points since it is more precise then words. Then distribution shifts were explained in depth and what assumptions one might have, such assuming the same joint distribution or same labeling function. The second part covered the deployment of AI and how research use differs compared to production usage. Furthermore, since AI and machine learning models are code, it still has to be treated like normal software code to some extent. For example, it needs maintenance and carefully created structures, for example through the usage of modularity. Then the primary reason for why AI needs to be maintenanced and changed, which has to do with two main things. Either when changes occur to the models pipeline or if the training data needs to change. Lastly the importance of data and model version was covered and how crucial it is, since as written above it is still a type of software.

Reflection of previous module (AI Tools)

There were three main points that I learned during the previous week's assignment, preprocessing of data, the importance of structure in ML learning and technical debt in ML.

- Firstly, in the previous week's assignment we were supposed to write a model that could predict if the PM was high or not. Among the features there were cyclical values that were presented as continuous discrete values, 1-4 for each of the seasons. This illustrated perfectly how data, whilst presented in an easy to read format is always not the best one to use for a ML model. At the same time the cardinal direction could have been saved in a similar version, data wise, however they were saved as different features with binary values instead. This subtle way of demonstrating two different ways data can be presented made me realize the importance of understanding preprocessing and the importance of analyzing each feature. Also something equally

important, but for a different reason is that when preprocessing one could also look at how the feature correlates to the output and through it try and determine its value for the model.

- Secondly, since our previous tasks in ML courses have been structured in a very straightforward fashion, do task a) then b) and so forth, with steps a usually being some sort of preprocessing and later steps fitting and evaluating. Hence this module refreshed the importance of writing codes in a class/function format and not just a long line of code. This is one of the major drawbacks of using jupyter notebooks in my mind, or at least in my case. Since it is so user friendly, it enables one to easily break coding principles without really thinking about it.
- Thirdly, what technical debt is for ML models and AI in general. This is not a reflection in the same way as the above point since it only required one to read the paper, whereas the other points are more of my own reflections. Nevertheless it is an aspect of ML I did not know about until this module