DAT410 Module 4 Assignment 4 – Group 26

Yahui Wu (MPMOB) (15 hrs)
yahuiw@chalmers.se
Personal number: 000617-3918
Tianshuo Xiao (MPMOB) (15 hrs)
tianshuo@chalmers.se
Personal number: 000922-7950

February 14, 2023

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

Yahui Wu

a) Similar to the development of automatic translation, the technology of speech recognition went through different stages with different approaches to achieve in history. In the early 1950s, three researchers from Bell Labs built a system to recognize digits in English. The system located the formants in the power spectrum of each utterance. In the 60s and 70s, the speech recognition system was most built aimed to recognize isolated speech of words. In the 1980s, continuous speech recognition became the hot topic. Instead of recognizing speech based on a standard reference, hidden markov model was used by researchers. Meanwhile, neural networks were introduced as a potential helpful method. In the late 1980s, neural networks had been used in many aspects of speech recognition. Neural networks make fewer explicit assumptions about feature statistical properties than HMMs and have several qualities making them attractive recognition models for speech recognition. However, traditional neural networks have some limitations in handling continuous speech. In the 2010s, with the development of machine learning and appearance of more advanced neural networks, the limitation could be solved by using deep neural networks (DNN) and the variant of DNNs which is RNNs. Another very interesting application of AI is AI playing chess which could be the earliest AI problems. In the earlier days, the system used Minmax algorithm which was based on exhaustive-attack method to list all the possible situations and choose the optimal step combined with $\alpha - \beta$ pruning which aimed to simplify the calculation as the game went on. Besides brute-force method to list all the possibilities exhaustively, the introduction of neural networks could be a more efficient way for today's AI system to solve game problem. The well-known AI system AlphaGO combined with Monte Carlo tree search (MCTS), reinforcement learning (RL) and deep neural networks (DNN) to imitate human players.
b) In the interlingual machine translation method, the system could find a intermediate representation of the source text and it is unified for all the worlds languages. The system translates the source language by translating the intermediate language to the target language instead of doing the direct translation. Neural networks encode the source sentence to a specific set of features, and another one can decode them back to the target text we need and the features just play a very similar role as interlingua in the interlingual machine translation.
c) The modern translation solutions may perform well when translating everyday phrases. However, they could have rare words issues. For example, when there is a technical term that is rarely used in everyday life, the modern systems could cause a failure. By contrast, rule-based machine translation has high morphological accuracy, reproducibility of results and the ability to tune it to the subject area so it is more suitable for some special professional tasks like the weather report translation, diplomatic documents translation (it requires the names of countries and areas to be translated accurately and in a fixed form) and the product manual translation.

Tianshuo Xiao

a.This article discusses the history of machine translation, include Rule-Based Machine Translation (RBMT), Example-based Machine Translation (EBMT), Statistical Machine Translation (SMT) and Neural Machine Translation (NMT). With continuous improvement of algorithms, the accuracy of translation has increased, and many real time translators have emerged consequently, such as Google Translate. In other AI problems, we can see a similar broad approach. In the image recognition problem, it has gone through three stages: text recognition, digital image processing and recognition, and object recognition. Traditional text recognition algorithms consist of three steps: image pre-processing, character segmentation and character recognition. These algorithms need to model a specific scene, which will fail once the scene changes, and the algorithms are less robust. When we face complex text backgrounds and scene changes, deep learning-based algorithms have better performance. For example, in the CTC method, the text sequences are not aligned and can be recognized more quickly.

There are many kinds of algorithms for image recognition. The first is the SVM algorithm. the SVM model uses a set of techniques to create an algorithm that will determine whether an image corresponds to a target object or not. Depending on the set data set, the SVM model is trained to classify the hyperplane into several categories. In this process, objects are placed in the hyperplan based on pixel values, and their positions are predicted based on the category separation learned from the training phase. By the later development of image recognition through neural networks, the main principle is that we can interpret one layer to analyze the color, another shape, the next texture of the object, etc. At the end of the process, the superposition of all layers makes the prediction possible.
In addition, voice recognition system. It has gone through three phases, Template Matching (DTW), Statistical Modeling (GMM-HMM) and Deep Learning (DNN-HMM, E2E). In template matching, continuous speech recognition is performed using a dynamic tracking phoneme approach. In the 1980s, statistical models were gradually investigated for speech recognition, with linguistic models represented by n-gram and acoustic models represented by HMM. In the 1990s, ASR was relatively mature, and the mainstream HMM-GMM was widely used. Nowadays, deep learning, based on end-to-end speech recognition solutions, gradually became the main point, the CTC algorithm became a classical method.
b.Rule-based machine translation includes Direct Machine Translation, Transfer-based Machine Translation, and Interlingual Machine Translation. It divides the text into words, translates them, corrects the morphology slightly, and harmonizes the grammar (or determines the grammatical structure of the sentences first). Interlingual Machine Translation converts the source text into an intermediate representation and then into target language.
The basic modelling framework for neural network-based machine translation is the end-to-end sequence generation model, a framework and method for transforming input sequences to output sequences. In the RNN model, we get the input source language word sequence by word separation at first, each word is represented by a word vector to get the appropriate word vector sequence next, and then use a forward RNN neural network to get its forward encoded representation. Then we use a reverse RNN to get its reverse encoding representation. Finally, we splice the forward and reverse encoding representations, and then use an attention mechanism to predict which word needs to be translated at which moment, and through continuous prediction and translation, we can get the translation of the target language.
There are similarities between the two systems. Both systems split whole sentences into word pairs to make predictions. In addition, both these translation systems use mediators. In an interlingual machine translation system, it converts the translated language into an intermediate language before translating into the target language. And in the state-of-the-art neural systems, they extract these specific features of the language and transform them into intermediate vectors before decoding them into the target language.
c.The "old-school" rule-based solution with its morphological accuracy (it does not confuse words, for example), repeatability of results (all translators get the same result) and the ability to adapt it to the subject area is suitable for translating legal texts or carrying out daily translations of weather forecasts. This is more convenient and less costly than a neural network translation system.

Implementation

(a) Warmup

```
import numpy as np
import pandas as pd
import string
from collections import Counter
import re

#read files
#english
de_en_en = open('europarl-v7.de-en.lc.en','r', encoding = "utf-8").read()
fr_en_en = open('europarl-v7.fr-en.lc.en','r',encoding = "utf-8").read()
sv_en_en = open('europarl-v7.sv-en.lc.en','r',encoding = "utf-8").read()
all_en = de_en_en + " " + fr_en_en + " " + sv_en_en + " "

#other language
de_en_de = open('europarl-v7.de-en.lc.de','r',encoding = "utf-8").read()
fe_en_fr = open('europarl-v7.fr-en.lc.fr','r',encoding = "utf-8").read()
sv_en_sv = open('europarl-v7.sv-en.lc.sv','r',encoding = "utf-8").read()
all_other = de_en_de + " " + fe_en_fr + " " + sv_en_sv + " "

#removing special symbols
def clean_file(name):
    punctuation = string.punctuation
    for i in punctuation:
        name = name.replace(i,'')
    return name
#removing special symbols
def clean_file2(textname):
    xxx = re.sub(r'[^\w]', ' ',textname)
    return xxx

#find most frequent words
def most_frequent_words(file_name, num):
    #file = file_name.replace(",","")
    #file = file.replace(".","")
    words = Counter(file_name.split()).most_common(num)
    print(words)
#Germany
de_clean = clean_file(de_en_de)
de = most_frequent_words(de_clean,10)
#French
fr_clean = clean_file(fe_en_fr)
fr = most_frequent_words(fr_clean,10)
```

Listing 1: Warmup

Germany
die, der, und, in, zu, den, wir, da, ich, das

```
de_clean2 = clean_file2(de_en_de)
de2 = most_frequent_words(de_clean,10)
```

```
[('die', 10521), ('der', 9374), ('und', 7028), ('in', 4175), ('zu', 3169), ('den', 2976), ('wir', 2863), ('daÃŸ', 2738), ('ich', 2670), ('das', 2669)]
```

Figure 1: The most frequently appearing words in German

French
apos, de, la, et, l, le, les, , des, que

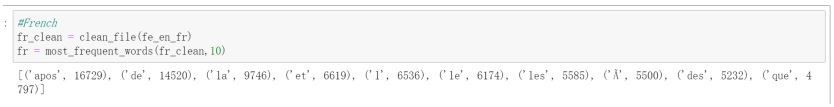

Figure 2: The most frequently appearing words in German

```
 # probability of 'speaker' and 'zebra'
all_texts = all_en + " " + all_other + " "
all_texts_clean = clean_file(all_texts)
all_texts_counters = Counter(all_texts_clean.split())
speaker = all_texts_counters["speaker"]/sum(all_texts_counters.values())
zebra = all_texts_counters["zebra"]/sum(all_texts_counters.values())
print(f"Probability of speaker: {speaker}")
print(f"Probability of zebra: {zebra}")
```

Listing 2: Language modeling

Probability of speaker: 2.13493846201621e-05
Probability of zebra: 0.0

(b) Language modeling

```
def find_bigram(lang):
    ''''
    find the bigram and unigram and count how many times for each to appear in the
    English text
    ''''
    bigram = []
    unigram = []
    for i in range(len(lang)-1):
        bigram.append((lang[i],lang[i+1]))
    for i in range(len(lang)):
        unigram.append(lang[i])

    count_bigram = dict(Counter(bigram))
    count_unigram = dict(Counter(unigram))
    return bigram, count_bigram, count_unigram # bigram(list),  count_bigram(dict),
    count_unigram(dict)

def probability(sentence, lang):
    '''
    calculate the probability of each bigram and unigram of one individual input
    sentence
    '''
    bigrams, count_bigrams, count_unigram = find_bigram(lang)
    pro_bigram = {} # dict to store the probability of each bigram
    sentence_split = sentence.split()
    bi_sen = [] # list to store bigrams of the input sentence
    prob = 1
    for bigram in list(count_bigrams.keys()):
        pro_bigram[bigram] = count_bigrams[bigram]/count_unigram[bigram[0]]

    for i in range(len(sentence_split)-1):
        bi_sen.append((sentence_split[i],sentence_split[i+1]))
        if bi_sen[i] in pro_bigram:
            prob *= pro_bigram[bi_sen[i]]
        else:
            # if the bigram does not exist in the bigrams of the English text, we
    consider it appearing 0.1 times
```

```
            prob *= 0.1/sum(count_bigrams.values())
    return prob
```

Listing 3: Language modeling

What happens if you try to compute the probability of a sentence that contains a word that did not appear in the training texts?
If the word did not appear in the training text, the probability will be very low.
What happens if your sentence is very long (e.g. 100 words or more)?
If the sentence is particularly long, the probability would be lower. We can use a bigger n value in the n-gram model instead of using the bigram model.

(c) Translation modeling

In translation model, we use the source language and English word alignments for its translation. This is because given a sentence in the source language, the machine translation system needs to determine the most likely translation in the target language. Each source language word depends on the English word it is aligned to, and we use a word translation probability $P(F|E)$ can make sure we have the right content.

```
# load the English file and French file
En_file = open('./dat410_europarl/europarl-v7.fr-en.lc.en',encoding = 'utf-8').read()
En_file = En_file.splitlines()
Fr_file = open('./dat410_europarl/europarl-v7.fr-en.lc.fr',encoding = 'utf-8').read()
Fr_file = Fr_file.splitlines()
En_file
sentence_pair = [] # list to store parallel sentence pair
for i in range(len(En_file)):
    sentence_pair.append([Fr_file[i],En_file[i]])
word_pair = []
# split each sentence pair into individual word
for i in range(len( sentence_pair)):
    En_word = re.split('[^\w+]', sentence_pair[i][1])
    En_word = [x for x in En_word if x]
    En_word.append('null')

    Fr_word = re.split('[^\w+]', sentence_pair[i][0])
    Fr_word = [x for x in Fr_word if x]

    word_pair.append([Fr_word,En_word])

def EM_ENtoFR(integ, word_pair):
'''
translation model to find the alignment
'''
    # initialization
    trans = {}
    for i in range(len(word_pair)):
        for en_word in word_pair[i][1]:
            for fr_word in word_pair[i][0]:
                trans[en_word,fr_word] = random.random()

    trans_total = {}
    delta = {}
    for t in range(integ):
        soft_cnt = {}
        soft_cntuni = {}
        for k in range(len(word_pair)):
            for fr_word in word_pair[k][0]:
                soft_cntuni[fr_word] = 0
```

```
                for en_word in word_pair[k][1]:
                    soft_cnt[(en_word,fr_word)] = 0

        for k in range(len(word_pair)):
            for en_word in word_pair[k][1]:
                trans_total[en_word] = 0
                for fr_word in word_pair[k][0]:
                    trans_total[en_word]+= trans[(en_word,fr_word)]
                for fr_word in word_pair[k][0]:
                    delta[(en_word,fr_word)] = trans[(en_word,fr_word)]/trans_total[
    en_word]
                    soft_cnt[(en_word,fr_word)] +=  delta[(en_word,fr_word)]
                    soft_cntuni[fr_word] += delta[(en_word,fr_word)]

        for k in range(len(word_pair)):
            for en_word in word_pair[k][1]:
                for fr_word in word_pair[k][0]:
                    trans[(en_word,fr_word)] = soft_cnt[(en_word,fr_word)]/soft_cntuni
    [fr_word]
    return trans

def trans_ENword(word,t,word_pair):
    '''
    function to find the most possible 10 words in French given an English word
    '''
    trans = EM_ENtoFR(t,word_pair)
    word_list = {}
    word_topten = []
    for pair in trans.keys():
        if pair[0] == word:
            word_list[pair[1]] = trans[pair]
    word_top = sorted(word_list.items(), key = lambda x:x[1], reverse = True)

    for i in range(10):
        word_topten.append(word_top[i][0])

    return word_topten

trans_ENword('european',3,word_pair)
```

Listing 4: Translation modeling

Output:
['européennes', 'européen', 'européens', 'union', 'européenne', 'risquées', 'prison', 'pic', 'labour', 'aérienne']

(d) Decoding

```
def trans_FRword(word,t,word_pair):
    '''
    find the most possible English word given a French word
    '''
    trans = EM_ENtoFR(t,word_pair)
    word_list = {}
    for pair in trans.keys():
        if pair[1] == word:
            word_list[pair[0]] = trans[pair]
    word_top = sorted(word_list.items(), key = lambda x:x[1], reverse = True)
    word_top = word_top[0][0]

    return word_top
```

```
def trans_sentence(sentence, t, word_pair, words):
    '''
    translate the given French sentence
    '''
    sen_trans = [] # store the words in the input sentence
    sen_split = sentence.split() # split the sentence
    for word in sen_split:
        sen_trans.append(trans_FRword(word,t,word_pair))
    if 'null' in sen_trans:
        # remove "null" in the word list
        sen_trans = sen_trans.remove('null')
    tmp_list = itertools.permutations(sen_trans)
    permu_word = [] # store the permutations of the given word list
    for res in tmp_list:
        permu_word.append(list(res))

    for sub_sen in permu_word:
        # find in what sequence the sentence will have the highest probability
        sub_sen = ' '.join(sub_sen)
        sen_permu[sub_sen] = probability(sub_sen,words)
    sen_top = sorted(sen_permu.items(), key = lambda x:x[1], reverse = True)
    return sen_top[0][0]
trans_sentence('une p tition a d  j     t   introduite',20,word_pair,words)

trans_sentence('je suis un  tudiant',10,word_pair,words)
```

Listing 5: Decoding

Output:
'people has already been introduced a'
'i am a darker'

Answers:
By running the code above, we could find that the translation model took the most likely corresponding word in English as the output by using EM algorithm for each word in the French sentence "une pétition a déjà été introduite". Then, it generated the permutations of combinations of the output English words. The language model found which permutation of these translated words were the best one with highest probability and took this permutation as the final output. Similarly, taking the second French sentence "je suis un étudiant" as an example, the translation model found the words list ['i','am','a','darker'] and generated the permutations ['I','am','a','darker'], ['i','a','am','darker'], ['am','i','a','darker']...etc. Then the language model calculated the probability of each permutation and took "i am a darker" which had the highest probability as the final result. However, there are some morphological accuracy issues in both cases. The French word "pétition" means petition in English and "étudiant" means student. The system translated those words to "people" and "darker" separately which have completely different meanings compared with the source words.

Assumptions:
1.We set the initialize $t(f|e)$ parameters to a random number. If this value of a certain pair of words is too large, it can affect the accuracy of the translation and make it difficult to converge with fewer iterations.
2. In this assignment, we translate English to French and French to English assuming a mutual process, i.e.$p(f|e) = p(e|f)$. Because we are training from a corpus, this can lead to inaccurate translation results when there are multiple meanings of a word.
3.When we make a translation, we use the word with the highest probability of matching in the corpus. This assumption may be made in the translation process without considering the original meaning of the sentence.

Problem:
In our language model, finding the English sentence with the highest probability requires calculating the probability corresponding to each possible word, which becomes lower with the longer sentences. It makes the computation of longer matching sentences intractable. Our language model sometimes assigns high probabilities to ungrammatical or nonsensical sentences, which makes it a challenge to find the sentence that best captures the intended meaning.

Discussion

(a). For the evaluation of the machine translation system, we evaluate it in terms of a manual procedure and an automatic procedure.
According to manual procedure, we can invite experts in linguistics to translate the sentences and compare them with the machine-translated results. The Manual procedure is more specific in its evaluation of translations and matches the preferences of human understanding of the language. However, it is time-consuming and costly.
According to automatic procedure, we can select a certain sentence to translate and find the sentence from the corresponding English corpus, and then make a word-by-word correspondence between the two sentences. This evaluation can save time and the sentences are more grammatical and structural. However, if there is a lack of corresponding words or fewer words, such as the presence of NULL in English, it may make the accuracy lower.
(b). The current online translation services such as Google translate are using recurrent neural networks (RNN) which could memorize the prior word when translating. The regular RNN was then upgraded to bi-directional, where the translator considered not only words before the source word, but also the next word. Moreover, the RNN with LSTM could store the translation context for a long time. The RNN could be a supervised neural networks method which means the training-set contains a sentence in the source language always has a corresponding sentence in the target language. In the training set, *ta* accompanied by *arvutiprogrammeerija* or *arst* in Estonian could be translated to "he" in English for most of the cases and LSTM units will store such a situation. When the translation system processes the new added sentence in Estonian where *ta* is accompanied by *arvutiprogrammeerija* or *arst*, the RNN could be affected by the storage in LSTM units and consider the following word *arvutiprogrammeerija* or *arst*. Therefore, the system tends to translate *ta* to "he". Similarly, because of LSTM units and the function of taking the next word into account in the bi-directional RNN algorithm, the system could be more tend to translate *ta* to "she" with the following word *lapsehoidja* or *meditsiinide*.
We believe it's a nice feature of a neural machine translation system using the RNN since it considers the real situation where the majority of doctors are male and most of nurses are female. It shows the intelligence of neural machine translation to some extent.
(c). When translating sentences with more frequent occurrences of the same words, the RNN recurrent neural network will remember the previous results. The correct translation of "bat" was selected under the first two sentences because it had been trained on a large corpus of text data, including examples of the use of the word "bat" in context. This enables the model to learn the appropriate meaning of the word in different languages and to select the correct translation based on the context in which it is used.
In the third sentence, the system created a new meaningless word because it had likely not seen a similar context before and it was generating a response based on its training data. This happens when the model encounters an incomplete sentence or an ambiguous input, generating a combination of words. The system tries to generate a meaningful response based on the information it has been trained on, but sometimes the result can be incorrect. Therefore in the algorithm we have to ensure both that the word alignment in meaning and that the grammar should fit the sentence structure.

References

[1] Euclidean metric versus Manhattan distance. AliceWanderAI-CSDN Blog_The difference between Euclidean metric and Manhattan distance. (n.d.). Retrieved February 7, 2023, from https://blog.csdn.net/NXHYD/article/details/103887404

[2] Verma, J. (2022, August 3). Easy ways to normalize data in python. DigitalOcean. Retrieved February 7, 2023, from https://www.digitalocean.com/community/tutorials/normalize-data-in-python

Reflection on the previous module

Yahui Wu

In last module, we applied what we have learned to implement a air pollution prediction system. Our task was to use k-means to classify the Chinese cities with different labels. To design such a system, we used fit, predict, score approach. In the "fit" function, we aimed to find the mapping which could label each centroid. The centroid with label "1" means the PM value is high while "0" means the PM is low. However, there was a distributional shift issue in the training set and test set. Viewing the training set which contained the data of Beijing and Shenyang, the air quality of these two cities seemed to be worse than that of Shanghai and Guangzhou whose data was taken as the test set. Given the same outputs, we could get the same input in training set and test set with different output distributions. Meanwhile, we could gave same outputs for same inputs with different input distributions. Guangzhou and Shanghai are in southern China where the air quality cold be better than cities in northern China which means the data was biased. We would like to have more expanded training set that contains northern cities as well as southern cities. Additionally, we found that our system was better at predicting cities with low PM. In other words, it could be possible for the system to get a higher score on test set than training set. The unusual issue happened because k-means tended to label the centroid as "0" since the majority of data was labeled as "0" whereas there were still many data points labeled as "1" in that cluster.
The data needs pre-process before we design the system. However, some features of the instances are difficult to standardize. For example, the wind direction of each instance was described as discrete values thus each wind direction was labeled as "0" or "1". To standardize the feature of wind, we could measure the components of the strength of wind in each direction.

Tianshuo Xiao

In last module, we learn about the AI Tools. In this module, we mainly learned how to build classifiers by fit, predict, and score. In our last assignment, we manually built a K-Means classifier to predict the air quality of other cities.
K-Means is an unsupervised learning algorithm. The core of this algorithm is clustering. It divides a large number of unknown labeled datasets into multiple categories according to the intrinsic similarity of the data, so that the data within the categories are more similar and the data between the categories are less similar. In our last assignment, we divided the data into k clusters by K-means in scikit-learn, and then brought k into our own set of classifiers. There are some differences between unsupervised learning and supervised learning. The training set for supervised learning requires that the input and output (features and targets) be included. The targets in the training set are artificially labeled. Supervised learning is the most common classification problem, where the training samples are trained to get an optimal model (the model belongs to a certain set of functions, and optimal means the best under a certain evaluation criterion), and then the model is used to map all the inputs to the corresponding outputs, and a simple judgment is made on the outputs to achieve the purpose of classification. In our last assignment, we manually labeled the classified cities after classification (PM_HIGH=0 or 1).
In discussion, we mainly considered about distributional shift between cities, label shift and imbalanced data. We also need to consider about choice of metric. In our assignment, we use accuracy score to judge training set and testing set accuracy. The accuracy score calculates the ratio of the number of correctly classified predictions to the total number of predictions. However, for unbalanced data sets, the accuracy score is not accurate. For all weather qualities, the category PM_HIGH=0 is the majority class. When the number of samples in the majority class (PM_HIGH=0) far exceeds that of the other classes (PM_HIGH=1), if accuracy is used to evaluate the goodness of the classifier, then even if the model performance is poor (e.g., PM_HIGH=0 is predicted regardless of the input weather quality).

The accuracy metric measures the global sample prediction. If we use F1-score evaluation metric, we can plot the confusion matrix and calculate Precision and Recall for each class separately, which can predict the accuracy more appropriately.
In conclusion, we need to consider in more depth the assumptions made in the machine learning process and its scoring methods. AI Tools is an integral part of machine learning and I need to continue to study its algorithms and make improvements to apply them to my own tasks.

Summary of lectures

Yahui Wu

Natural language processing
February 7, 2023
In the last class, we discussed about natural language processing (NLP). In everyday life, NLP is of widely used such as spam filters, spellcheckers and grammar checkers. The data being used by NLP is discrete, structured, diverse and sparse. To process this data and do the translation well, there are several tasks to be done in a NLP algorithm. The first task is to figure out what category of the data belongs to. The second task is to tag or label the part of speech for each word in a certain sentence. Then, we need to figure out the structure of this sentence. The last task is to generate the target sentence based on the previous steps. The development of NLP went through several stages. In the early days, rule-based methods are the dominated method in NLP. In the early 80s, NLP research started to focus on data-driven. In the 1990s, lots of linear models from machine learning were used and these are the main trend until neural models were introduced in 2015. Neural models are more natural in transfer learning and are end-to-end solutions which are more feasible. However, neural models are expensive, hard to tweak and costly. Therefor, as a very easy and fast method, linear model is still of widely used. Using rule-based machine translation (MT) and interlinguas could be a solution but impractical for open-ended MT. By contrast, data-driven solution such as statistical MT is a better method. Statistical MT is trained on parallel example texts which could be collected from many different sources. One of the most famous model of statistical MT is the IBM models. In this class, we focused more on IBM model 1. Basically, the model was built based on $E^* = \underset{E}{argmax}\, P(E|F)$. Then we rewrite this into $E^* = \underset{E}{argmax}\, P(E)P(F|E)$ where $P(F|E)$ is the translation model to make sure the content we have is correct and $P(E)$ is the language model which is responsible for the fluency of the translation and to train the language model, we just need English text. In the language model, we would like to know what is the most possible word to appear given a certain word. To find the next word with most probability, the Markov assumption could be used here. According to the assumption, the next word only depends on the current word. Thus we could obtain the probability of a certain bigram using MLE: $P(w_n|w_{n-1}) = \frac{count(w_n, w_{n-1})}{count(w_{n-1})}$. To estimate probabilities of the word translation, the alighment probabilities were introduced. For instance, to translate the French word chat in such a French sentence "le chat noir" in the context of a parallel English sentence "the black cat", we choose the word in the target sentence with the highest probability. To achieve this, we could use Expectation-Maximization (EM) algorithm. In the EM algorithm, we compute "soft-count" $c(f \rightarrow e)$ and $c(e)$ based on posterior word alignment probabilities. Then, we re-estimate $t(f|e)$ based on the "soft counts". Fist, we assume the current $t(f|e)$ are some raandom values. In each integration, we add the $c(f \rightarrow e)$ to that from the last integration and set $t(f|e) = \frac{c(e,f)}{c(e)}$. We do this several times to find the translation.

Tianshuo Xiao

Natural language processing
February 7, 2023
This lecture introduce the filed of NPL with some particular machine learning models and how to deal with language into AI problems. The most important thing is that transfer them into probabilistic models in AI.
NPL is central in AI research and applications, which can process the spoken and written language. Also, there are some applications such as spam filters, spellcheckers and grammar checkers, machine translation systems, etc. The language is different from other data. Each sentence has its own structure, and the structure varies in different language systems. We need to combine discrete and continuous

to unite languages and deal with different languages. For example, map the frequency of word occurrences in a language and use machine learning for training. Then we can create a classifier that assigns labels to keywords, such as spam or not spam. For a sentence, we perform sequential input and then perform sequential output as a subclass of data structures of numbers, i.e., structured output, and finally generate the translated text.

For the history of NLP development, in the beginning different machine learning models were generally used, which is actually a statistical model to calculate the frequency of word appearance. As science developed, we could split the document, identify the grammar, and find the subject-predicate object in the sentence. Then we can use the algorithm of neural network, train the model and then reuse it, for a new system to train and gain experience from the old system.

When we want to translate the meaning of an item, they imagine a picture of the item in their mind, and through mental representation then output the target language. For machine translation, the input interlanguage. So, we build some kind of data structure to represent the encoded information. The IBM Model is based on a word alignment algorithm for translation. Suppose any English sentence e and a French sentence f. Define the probability of translating f into e as $Pr(e|f)$ with the normalization condition, $\sum_e Pr(e|f) = 1$, so that the problem of translating f into e becomes solving as $\hat{\boldsymbol{e}} = \operatorname{argmax} \Pr(\boldsymbol{e} \mid \boldsymbol{f})$. In this model, P(E) is the language model, which reflects the degree of "E is like a sentence", i.e. fluency; $P(F|E)$is the translation model, which reflects the degree of "F is like E", i.e. fidelity; using the two models together is better than using the translation model separately.Among them, we have to solve the modeling and parameter estimation of the language model P(E), the modeling and parameter estimation of the translation model $P(F|E)$ and the decoding (search) algorithm. Through Bayes we know that $E^* = \arg\max_E P(E \mid F) = \arg\max_E P(E)P(F \mid E)$. Another language model is Markov assumption, the probability of the whole sequence is $P(w1, ..., wn) = P(wn|wn-1)....P(w2|w1)P(w1| < START >)$

In translation model, crucial idea in it word alignments. The main idea is using maximum likelihood estimate: $T_{MLE}(chat|cat) = \frac{c(cat-->chat)}{c(cat)}$. But when alignments are not available we are supposed to use ExpectationMaximization algorithm to the rescue. First we compute word alignment probabilities, then update our soft counts (label). Finally, we can get translation results.