TÜBİTAK Internship Report

First of all, before going into internship report, let me introduce myself and tell a bit about how the internship was and the thoughts about the community there. My name is Akif Faruk NANE studying at Bosporus University as a new sophomore student. I guess I have a really good background of computer science comparing to my peers. I always wanted to do crazy things and have been doing so whole my life. Starting the adventure with mathematics and shifted to the competitive programming, -you know- one always should find the best optimal solution of a problem. Beside those, I was also studying C# like 4-5 years in high school. So, after high school, I got interested in Artificial Intelligence specifically deep learning. My internship here was also on deep learning. Lately, I have been working writing heavily optimized numeric library with only C#, since .Net Core supports Simd instructions and catched C++ in terms of performance. I hope I can find someone who knows about these topics in Tübitak as well.

Let's get into the internship. I really liked the atmosphere and the environment in Bilgem. I've met many new people and luckily and especially the one whose name is Kadir working at Hisar lab who guided and some kind of mentored me. I had a few good conversations with him talking hours how we can achieve better language models and about humans and the business life etc. However, -beside the pros- the cons were the waking time which was really hard for me, I got accustomed after a week or so and I'm not sure now if working at Bilgem makes a great deal of benefits for me. I'm kind of a person that learning alone and whose interests could differ easily. Also, I'd like to mention that I have always been the one who is enjoying developing technologies that others can utilize and create products. So, I can say I don't like working on projects closer to the customers.

Okay, now let's get into the real stuff.

Akif Faruk NANE

Language models

Language models are used to predict the next upcoming word based on what's written before and also score the compatibility of the words in other words the likelihood probability of the sentence being created. The simplest model is N-gram models. N-gram is a sequence of N words creating a phrase. To illustrate, 2-gram is a sequence of 2 words like 'Good morning', or for 3-gram, it might be 'Studying computer science'. We use N-grams to create a language model by assigning probabilities for words.

N-gram

To determine the probability of a sentence created, first the probability of the words in the sentence should be determined. If the probability of words is known, it is easy to find the probability of a sentence by simply multiplying all probabilities of given words. Let's denote the probability of a word in a context of H as $P(W \mid H)$. It is simply calculated by using the count of phrases in the training data. The count of the phrase H combined with the word W divided by the count of the phrase H.

$$P(W \mid H) = C(W,H) \,/\, C(H)$$

$C(X)$ specifies the count of the phrase X in the training data. For a sentence consisting of words $(w_1, w_2, \ldots., w_n)$, the probability of word w_n is $C(w_1, w_2, \ldots, w_n) \,/\, C(w_1, w_2, \ldots, w_{n-1})$. So, it might be said the probability of 'Have a good day' is,

$$C(\text{'}Have\ a\ good\ day\text{'}) \,/\, C(\text{'}Have\ a\ good\text{'})$$

For a training set ('$Have\ a\ good\ day$','$Have\ a\ good\ time$','$Have\ a\ good\ breakfast$') its probability is $1/3$. Now, let's prove that the probability of a sentence equals the multiplication of the probability of words.

$$P(w_1, w_2, \ldots., w_n) = P(w_1)\, P(w_2 \mid w_1)\, P(w_3 \mid w_1, w_2) \ldots.. P(w_n \mid w_1, \ldots., w_{n-1})$$

- ⇨ $P(w_1) = C(w_1) \,/\, C(\,)$
- ⇨ $P(w_2 \mid w_1) = C(w_1, w_2) \,/\, C(w_1)$
- ⇨ $P(w_3 \mid w_1, w_2) = C(w_1, w_2, w_3) \,/\, C(w_1, w_2)$
- ⇨ .
- ⇨ $P(w_n \mid w_1, \ldots., w_{n-1}) = C(\,w_1, \ldots., w_n) \,/\, C(\,w_1, \ldots., w_{n-1})$

Multiply all the expressions, and we get $C(\,w_1, \ldots., w_n) \,/\, C(\,)$ meaning the count of W divided by the count of sentences in training set. So, H is blank here. $P(\,w_1, \ldots., w_n \mid H) = C(H, w_1, \ldots., w_n) \,/\, C(H)$ turns into $C(\,w_1, \ldots., w_n) \,/\, C(\,)$ meaning $P(\,w_1, \ldots., w_n)$.

For N-grams we generally calculate the likelihood probability. Probability of a word w_i is $P(w_i \mid H)$ where H is considered to corresponding last $n-1$ words. So the probability of a word turns into $P(w_i \mid w_{n-i+1}, w_{n-i+2}, \ldots, w_{i-1})$.

Why consider only last $n-1$ words to calculate the probability of n_{th} word? As everyone knows well that language is creative tool that humans use. So, what if someone create a sentence that never been created but likely to be created in some context H very meaningfully?

Do we say, the sentence does not show up in the training set, so the probability must be zero? It would be ridiculous. So instead of finding whole phrase, the large meaningfully coming together structure in the training set, we look for the smaller meaningful phrases that might more likely to be in the training set and that combine the main sentence which we want to determine the probability. This is why n-gram models are used. It is not logical to search one big phrase in data. No matter how big data is, there will be always unseen sentences. Maybe right now I am writing one of those, who knows…

Let's name it as **zero situation** if we don't have the phrase in our data, which means $C(W) = 0$. There are smoothing techniques to overcome zero situations. However, we will not give point to those.

If you are to analyze the number $\boldsymbol{n}$, what should $\boldsymbol{n}$ be, what if $\boldsymbol{n}$ is big number or a small one, what would you say?

The less n becomes:

- ⇨ The less data is required. Because there are less combinations of words comparing to greater $\boldsymbol{n}$s.
- ⇨ The less zero situation there might be.
- ⇨ The less meaningful, but the more creative sentences could be predicted or estimated. The semantic meanings and the combinations are correct and likely trustable for those in training sets created by human beings. The shorter the phrases in the training set becomes, the smaller meanings that they hold. Forming new sentences with those combinations might not be meaningful sometimes. This shows that they are more creative when n is smaller.

Perplexity

Perplexity is a metric likely considered to be the average error score of each word.

$$PP(w) = \sqrt[L]{\prod_{i=0}^{L} \frac{1}{p(w_i|w_1 \cdot \cdots w_{i-1})}}$$

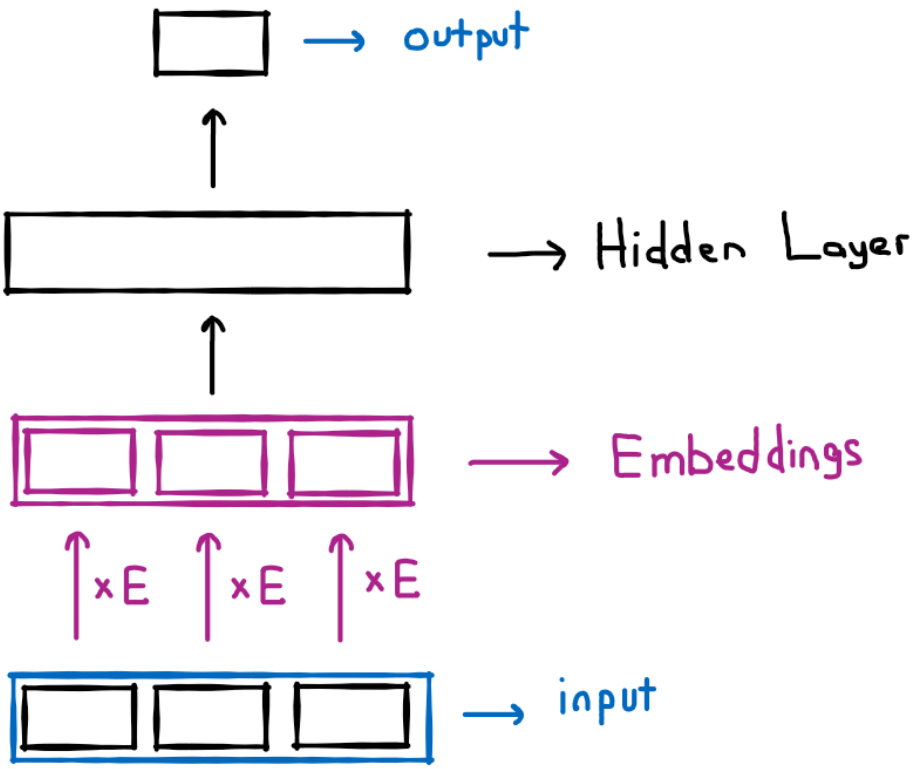
The n-gram version would be:

$$PP(w) = \sqrt[L]{\prod_{i=0}^{L} \frac{1}{p(w_i|w_{i-n-1} \cdot \cdots w_{i-1})}}$$

Feed Forward Neural Language Model

The simplest n-gram model with neural networks would be a feed forward neural LM. The model takes $n-1$ words as input and gives the probabilistic distribution of what the next word should be as output.

The scheme of an example model is shown below.



The words at the input are passed to the network as **one-hot vectors**. The one-hot vector of w_i is an array whose element i is one and the rest are zeros. Assume the **dictionary size is T**. The input consists of 3 words in the example which corresponds to $3T$ numbers in one-hot coded state.

Each one-hot vector is passed to up-forward layer being multiplied by matrix **E** which is the abbreviation of **Embedding Matrix**. It can be said that embedding matrices are word to vector converters. It simply takes a one-hot vector and converts it into a meaningful another vector. By doing so, it allows us to feed the meaning of the words into the network.

The output has T numbers every of which are between zero and one and each one corresponds the probability of being the next word predicted.

There are many ways to create Embedding Matrices. Pretrained embeddings can be used in neural networks. It can also be trained during the training process as well. Needless to say, it's up to you whether the Embedding Matrix should be fixed, trainable or semi-trainable. You can mix the previous meanings in the Embedding Matrix with the new meanings required for the problem that you are dealing with.