Mini-Project Report
Information Retrieval
Instructor: Venkatesh Vinayakarao
Chennai Mathematical Institute

Prateek Chandra Jha (MDS201922)
Saikrishna Ranganathan (MDS201928)

.

Submission Date: 13/12/2020

Introduction: Exploring the Mathematics of Attention Mechanism and Cutting Edge Transformers in Google Search and Information Retrieval

As recently as 15th October 2020, Google published a blog on how Artificial Intelligence is powering a more helpful Google[1]. Mr. Raghavan writes:

> *At the heart of Google Search is our ability to understand your query and rank relevant results for that query. We've invested deeply in language understanding research, and last year we introduced how BERT language understanding systems are helping to deliver more relevant results in Google Search. Today we're excited to share that BERT is now used in almost every query in English, helping you get higher quality results for your questions.*

In 2018, Google introduced and open-sourced a neural network-based technique for natural language processing (NLP) pre-training called Bidirectional Encoder Representations from Transformers[2], or BERT, in short. This technology enables anyone to train their own state-of-the-art question answering system. This breakthrough was the result of Google's research on transformers: models that process words in relation to all the other words in a sentence, rather than one-by-one in order. BERT models can therefore consider the full context of a word by looking at the words that come before and after it—particularly useful for understanding the intent behind search queries. In our IR course[3] we have seen a flavour of Probabilistic Information retrieval but the Attention mechanism[4] employed by the transformer models takes it up a notch. ***This project report explores the mathematics behind Attention mechanism, causal attention, bidirectional attention and multi-headed attention employed by Google Search's BERT models to explain the procedure of conducting a translation retrieval mechanism. The associated code can be found at this github repository:*** `https://github.com/prateekchandrajha/ir-mini-project`.

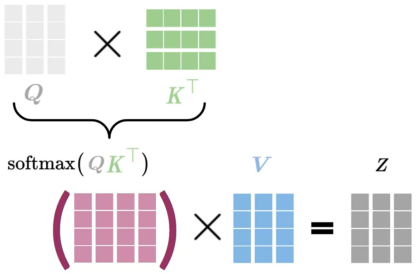

Figure 1: Dot-product Attention is the heart and soul of transformers. In general terms, the attention takes as inputs, queries, keys and values, which are matrices of embeddings. Dot-product Attention is composed by just two matrix multiplications and the softmax function.

[1]Prabhakar Raghavan (Senior Vice President, Search & Assistant, Geo, Ads, Commerce, Payments & NBU) writes on how AI is powering a more helpful Google. Available at: `https://blog.google/products/search/search-on/`

[2]Pandu Nayak (Google Fellow and Vice President, Google Search) writes on Understanding searches better than ever before using BERT. Available at: `https://blog.google/products/search/search-language-understanding-bert/`

[3]Available at: `http://vvtesh.co.in/teaching/IR-2020.html`

[4]Attention Is All You Need (Vaswani et al, 2017). Available at: `https://arxiv.org/abs/1706.03762`

What happens in Neural Machine Translation?

A Quick Note: This report does assume a fair level of awareness regarding the deep learning architectures-like seq2seq models, GRUs, LSTMs, why some of them work and why they don't. Apart from bare minimum and some common knowledge regarding the training of deep neural networks, nothing else is assumed on part of the reader. We provide enough intuition to bridge the gap and if unable to do so we provide enough references to consult.

In neural machine translation we use an encoder and a decoder, to translate from one language to another. In the associated code[5] we have used a WMT dataset for translating from English to German. To do this, we have uses a machine translation system that has LSTMs for both, encoding and decoding. The traditional Seq2Seq model was introduced by Google in 2014, and was a revelation at the time. Basically, it works by taking one sequence of items such as words, and us putting another sequence. The way it does this is by mapping variable length sequences to a fixed length memory. This feature is what made this model a powerhouse for machine translation, among other things. The inputs and outputs don't need to have matching lengths. You might recall the vanishing gradients problem from earlier in the specialization. And Seq2Seq models LSTMs and GRUs are typically used to avoid vanishing gradients.

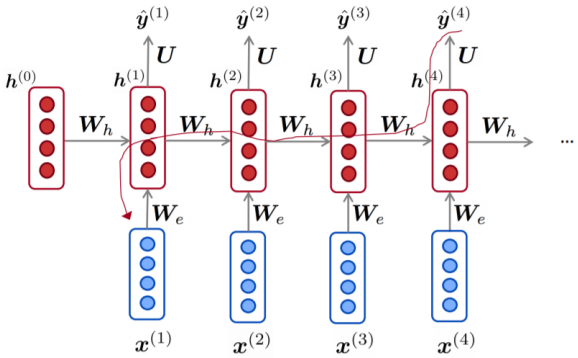

Figure 2: Gradients in neural networks can be seen as a measure of influence of the past on the future. How does the perturbation at time t affect predictions at $t + n$? As you can see that RED line in the figure demonstrating just that, it's the flow of information but what if the information fails to flow across the layers. The vanishing gradient problem can cause problems for simple RNN Language deep learning Models. When predicting the next word, information from many time steps in the past is not taken into consideration. So what's the solution?

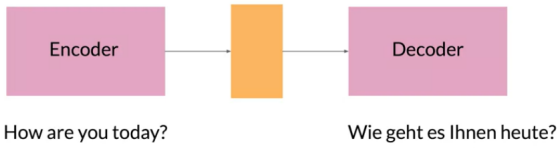

Figure 3: An encoder decoder takes in a hidden state and a string of words, such as a single sentence. The encoder takes the inputs one step at a time, collects information for that piece of inputs, then moves it forward. The orange rectangle represents the encoders final hidden states, which tries to capture all the information collected from each input step, before feeding it to the decoder. This final hidden state provides the initial states for the decoder to begin predicting the sequence.

One major limitation of the traditional Seq2Seq model, is what's referred to as the information

[5]Available at: `https://github.com/prateekchandrajha/ir-mini-project/blob/main/IR_translation_retrieval_attention_mechanism.ipynb`

bottleneck. One might have already begun to imagine what can happen in the case of a long sequence. As individual inputs begin stacking up inside the encoders final hidden states, because Seq2Seq uses a fixed length memory, longer sequences become problematic. Another issue surfaces as the later input steps in the sequence are given more importance which you can see illustrated in Figure 4 below, with the last word of the sentence today. This results in lower model performance as sequence size increases,

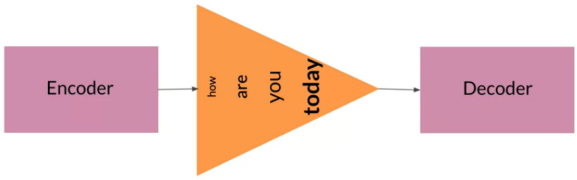

Figure 4: All of this results in a model that performs incredibly well for shorter sequences. And not so well for longer, more complex ones. So the power of Seq2Seq which lies in its ability to lets inputs and outputs be different sizes, becomes its weakness when the input itself is a large size. Because the encoder hidden states is of a fixed size, and longer inputs become bottlenecked on their way to the decoder.

and that's no good. So the issue with having one fixed size encoder hidden states, is that it struggles to compress longer sequences. And ends up throttling itself and punishing the decoder, which only wants to make a good prediction. How can you be nicer to the decoder? You could hold onto each word vector with this individual information, instead of trying to smash it all into one big vector of hidden states (See Figure 5 below). But this model still has obvious flaws with memory and context. How could you

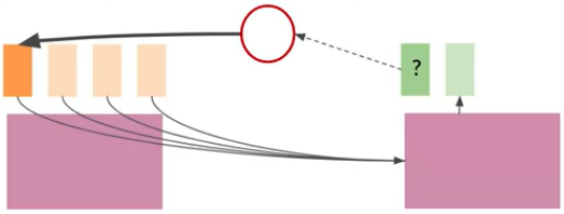

Figure 5: If we can provide the information specific to each input word, we give the model a way to focus its **ATTENTION** in the right place/region at each step. This is a very important intuition which has changed the world of information retrieval using deep learning architectures.

build a time and memory efficient model that predicts accurately from a long sequence? This becomes possible if the model has a way to select and focus on the likeliest words each step. You can think of this as giving the model a new layer to process this information. If you provide the information specific to each input word, you give the model a way to focus its attention[6] in the right place at each step. In the next sections, we provide a conceptual idea of what this new attention layer is doing and why did we need it. This section provided an overview of neural machine translation, and also a rough intuition of what does attention look like. Simply put, we have got to know which words the model is focusing on when translating from one language to another language.

[6]Attention is all you need (2017). A Vaswani, N Shazeer, N Parmar, J Uszkoreit, L Jones, AN Gomez

Why is Alignment so important in Information Retrieval?

Before we entered the era of end to end neural machine translation, alignments were very critical when translating one language to another language. Alignment is still widely used today for word sense disambiguation or word sense discovery. If our model needs to be able to focus in the right place (remember intuition behind attention), so it can choose the right output to predict, it makes a lot of sense that you would want the words of the inputs to align with the words of the output. The concept of word alignment is not a new one, actually, it used to be critically important to classic methods of statistical translation. It's still widely used today for translating languages, and word sense discovery, and disambiguation (See Figure 6 below).

Figure 6: For word sense disambiguation, let's say you have the word bank over here, which could mean either a financial institution or a riverbank. What you can do with this is translate the word into another language. And based on the interpretation of this word in the other language, you'll be able to tell which definition is meant.

For our associated code on neural machine translation with attention, we have achieved word alignments by using a system that retrieves information for each piece of inputs and scores it[7]. Let's take a look at how words align in a sentence, where each word is not exactly the same when translating from English to German.

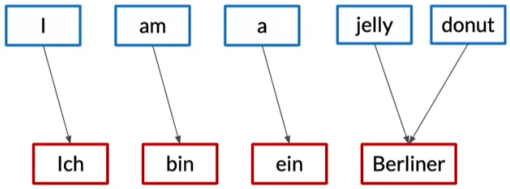

Figure 7: This example over here is from a famous speech given by John F Kennedy in Berlin during the Cold War, which caused a lot of confusion. It's because the term Berliner can denote either a citizen of Berlin or a Jelly donut. You can read more about this historic cultural misunderstanding over here: `https://www.theatlantic.com/magazine/archive/2013/08/the-real-meaning-of-ich-bin-ein-berliner/309500/`

After seeing Figure 7 one can notice how the English translation has more words than the German version (Think Alignment). When performing word alignment, your model needs to be able to identify relationships among the words in order to make accurate predictions in case the words are out of order or not exact translations. In a model that has a vector for each input, there needs to be a way to focus more attention in the right places. Many languages don't translate exactly into another language. To be able to align the words correctly, you need to add a layer to help the decoder understand which inputs are more important for each prediction. Enter the attention layer which performs a series of calculations that assigns some inputs more weights than the others (Please see Figure 8 below). Now we discuss what exactly happens inside the attention layer of the Neural Machine Translation (NMT) model to see what it's doing. The key idea is relatively pretty straightforward so let's delineate what's going on:

- First, get all of the available hidden states ready for the encoder and do the same for the first hidden states of the decoder. In this simplified example (please see Figure 9 while reading ahead), there are two encoder hidden states and one decoder hidden states.

[7]Scoring Mechanism behind BLEU coded over here: `https://github.com/prateekchandrajha/ir-mini-project/blob/main/IR_project_Bleu_Score_Implementation.ipynb`

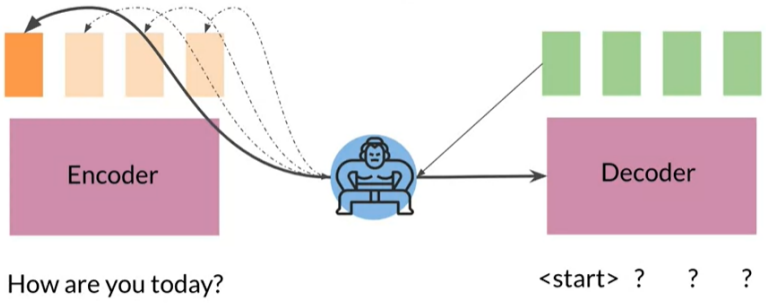

Figure 8: We can see here that the hidden state shown in dark orange has a heavier line, which indicates that it's been given a higher attention score. This means that the next word in the decoder's output will be strongly influenced by this encoders hidden states. Now that you know what word alignment is, the goal of attention and alignment is computed by giving each word vector a score.

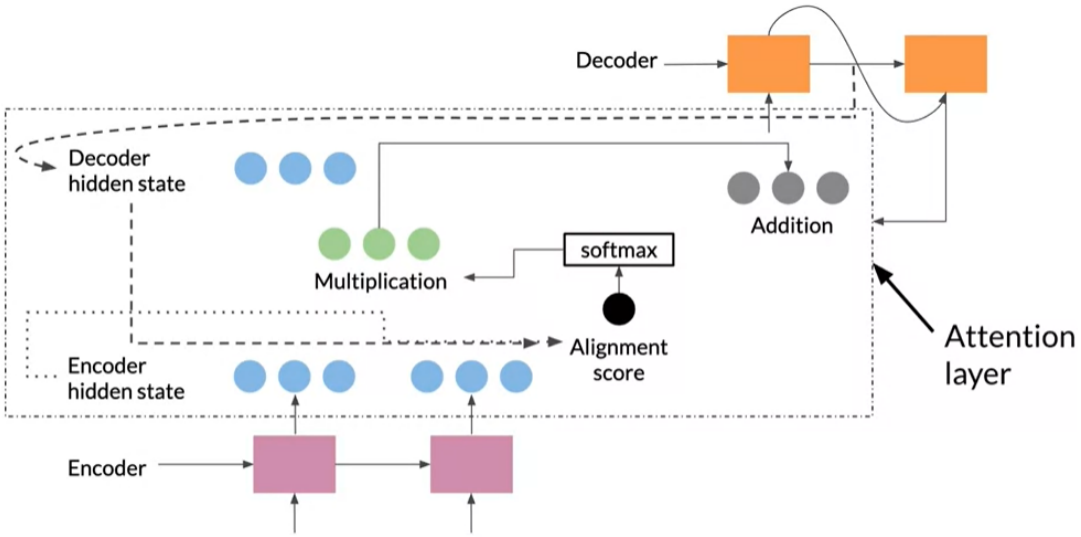

Figure 9: Mathematical Underbelly of Attention Scoring Mechanism

- Next, we score each of the encoder hidden states by getting its dot product between each encoder state and decoder hidden states. If one of the scores is higher than the others, it means that this hidden state will have more influence than the others on the output.
- Then we will run scores through softmax, so each score is transformed to a number between 0 and 1, this gives us our attention distribution. We take each encoder hidden state, and multiply it by its softmax score, which is a number between 0 and 1, this results in the alignments vector.
- Now we simply add up everything in the alignments vector to arrive at what's called the context vector. This is what we feed into the decoder, so whatever is happening over here in Figure 9 can be distilled to a few mathematical operations that are scoring words based on their importance.

In this section we established the importance of alignments in information retrieval, and saw how attention makes use of this concept so brilliantly. In the next section, we delve a bit deeper into understanding the attention mechanism and what's actually contained in those hidden states we have referring to for long now.

Exploiting The Concept of Attention For Information Retrieval

In this section we shall delve a bit deeper into how attention works by creating some components for each input that are useful for quick information retrieval. Firstly though, we thought it best to provide some intuition for information retrieval since that's what our course is about, and then move on to discussing the components used for calculating attention weights, called keys, queries, and values. So before getting deeper into attention territory, here's some intuition for how information retrieval works in the context of attention (Please see Figure 10 and its caption below).

Figure 10: Say we have lost our keys and have spent a lot of time going over each meter of our room looking for them. This is time-consuming and inefficient if we don't have a good starting point. Wisely, you ask your mom to help you find them. She thinks for a moment weighting up all the possibilities for where the keys could be, based on what she already knows about where they usually are and our lax & lethargic behaviour. Then, she tells us the likeliest place to look. Lo & Behold, she's right, there are your keys in one of those suggested locations. And that in nutshell is what attention is doing. Taking a query, selecting the place where the highest likelihood to look for the key, then retrieving the right key.

Now, we start peeking under the hood to see attention in action. The attention mechanism uses encoded representations of both the input (or the encoder hidden states) and the outputs (or the decoder hidden states). The keys and values are pairs. Both of dimension N, where N is the input sequence length and comes from the encoder hidden states. Keys and values have their own respective matrices, but the matrices have the same shape and are often the same matrix. While the queries come from the

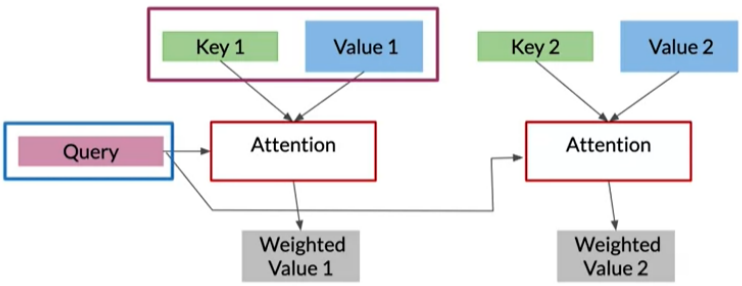

Figure 11: Peeking inside the Attention Layer

decoder hidden states, both the key value pair and the query enter the attention layer from their places on opposite ends of the model (See Figure 11 above). And once they're inside, the dot product of the

querying and the key is calculated. This is essentially a measure of similarity between them, and the dot product of similar vectors tends to have a higher value. The weighted sum given to each value is determined by the probability that the key matches the query. Probability as you might recall, can be determined by running the attention weights through the softmax, so they are transformed to fit a distribution of numbers between zero and one. Then, the query is mapped to the next key value pair and so on and so forth. This is the so called scaled dot product attention i.e. $Attention = Softmax\left(\mathrm{QK^{\wedge}T}\right)\mathrm{V}$. Now, let's visualize how the attention weights look in matrix form (please see Figure 12 and the caption

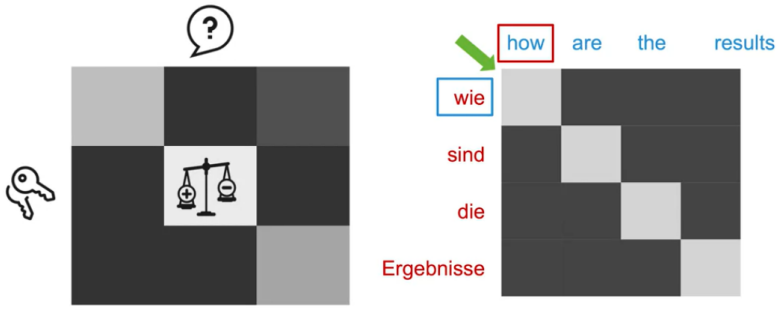

Figure 12: You have all the inputs or words of one query Q as columns and the words of the keys (K) as the rows. That value score that we just mentioned as the weighted sum is indicated here in a lighter shade of gray to show the highest score and therefore the corresponding match.

above). Figure 12 also features an example (on the right side of the image) that shows where the model is looking when translating between English and German, as we've implemented in our code. In this rather straightforward example, attention is first translating the English word how, into the German word wie. When attention is translating, the word are, is looking at the word sind, and so on.

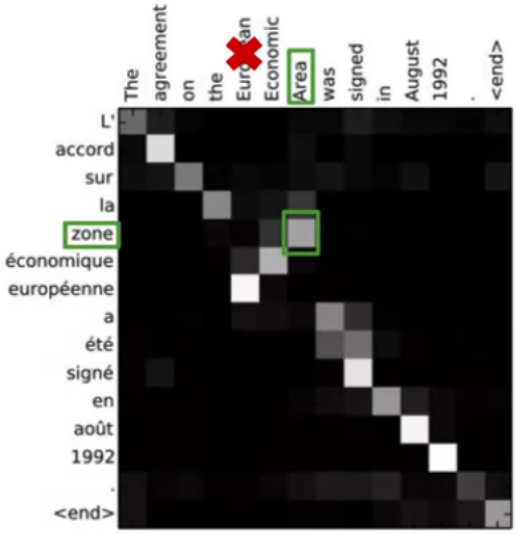

Figure 13: You can see that the first four tokens, "The agreement on the", are pretty straightforward in translating, but then the grammatical structure between French and English changes. Now instead of looking at the corresponding fifth token to translate the French word zone, the attention knows to look further down at the eighth token, which corresponds to the English word "area". This is the essence of information retrieval using attention. It's amazing what a little matrix multiplication can do in making retrieval easier.

An important thing to keep in mind is that the model should be flexible enough to connect each English word with its relevant German word, even if they do not appear in the same position in their respective sentences. In other words, it should be flexible enough to handle differences in grammar and word ordering in different languages. So, even though in this example in Figure 12, the word how is the first word in the English sentence and the word wie is also the first word in the German sentence, the attention model should still be able to find a connection between the two words, even if the grammar in one language requires a different word ordering than the other language does. In the matrix, the lighter square shows where the model is actually looking when making the translation of that word. We will shortly see how to build an attention matrix like this one. We will also learn how to properly interpret the attention matrix and how to use it to make the word predictions as we have done in our code.

In a situation like the one we just mentioned, where the grammar of foreign language requires a difference word order than the other, the attention is flexible enough to find the connection between various word orderings and contextual perspectives (please see Figure 13 above). In this section, we showed how attention is a layer of calculations that let your model focus on the most important parts of the sequence for each step. Queries, values, and keys are representations of the encoder and decoder hidden states. And they're used to retrieve information inside the attention layer by calculating similarity between the decoder queries and the encoder key value pairs. This is so flexible that it can even find matches between languages with very different grammatical structures or linguistic alphabets. In the next section, let's build upon this and cover the nuts & bolts of the neural machine translation and show exactly how the setup looks like for training and building such a system. We shall also have a look at the dataset we have used for the project code and also talk about the steps required for preprocessing our data sets.

How to Build & Train such a Machine Translation System?

In this section we will see how to train our neural machine translation system- with attention. We will learn about training concepts like teacher forcing, and also see which type of cost function is being used when training such translation models. By this point in this report, we are well aware of how attention works in a seq2seq model by saving the information from each time step in its own hidden states and using the decoder's previous prediction to give more or less weights to each of the encoder hidden states. In the last section, we have seen how specially weighted values are tailored to each subsequent German prediction in Eng-to-German translation. But how do we really know and assess that these predictions aren't wacky or a total waste? This is where teacher forcing comes in. Teacher forcing allows your model to use the ground truth or the actual outputs from your decoder to compare its predictions during training. This yields faster training with the added benefits of higher accuracy. Let's take a closer look at

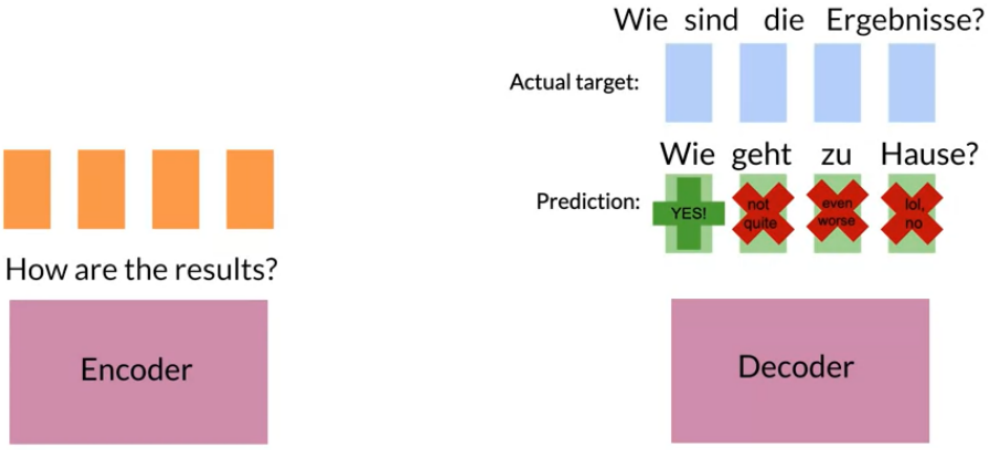

Figure 14: Motivation behind Teacher Forcing: In this example we notice how the model correctly predicted the token at the start of the sequence. But the second prediction doesn't quite match. The third one is even further off. The fourth predicted token is quite far from making logical sense in German.

why teacher forcing is necessary in a seq2seq model. The important takeaway here, is that in a sequence model like this one, each wrong prediction makes the following predictions even less likely to be correct. We need to have a way to check the prediction made at each step. This is how the process works during prediction:

- The orange rectangles (see Figure 14 above) shown here is processed through attention, in order to predict the green rectangle, and so on and so forth for the next predictions. An important takeaway of this concept is that during training, the predicted outputs are not being used for predicting the next predicted green rectangle. Instead, the actual outputs, or ground-truth, is the input to the decoder for each time step until the end of the sequence is reached.
- Without teacher forcing, models can be slow to reach convergence, if they manage to reach it at all. Teacher forcing is not without its issues and is still an area of active research[8].

This is the part where we put together everything we have seen so far in this report. We have used this knowledge to try and train our very own neural machine translation model with attention[9].

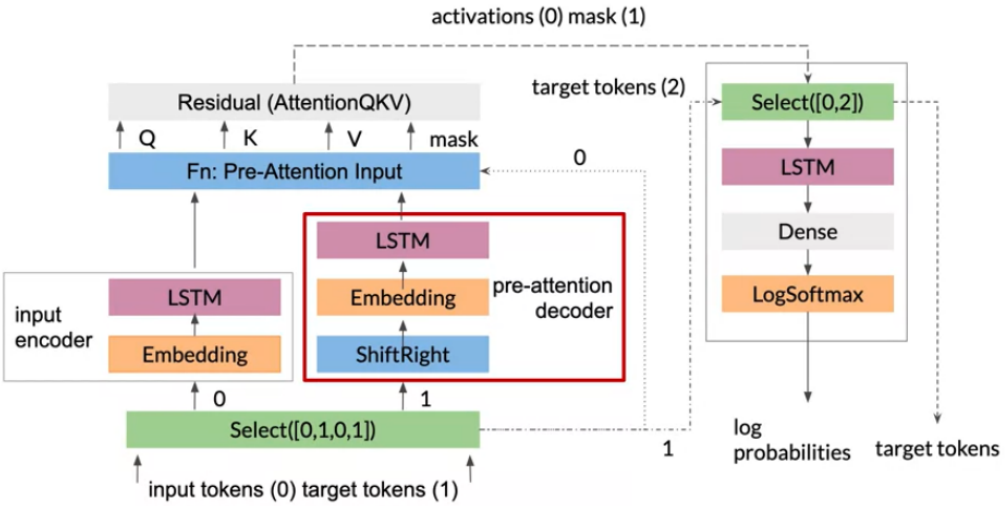

Figure 15: The Nuts & Bolts of Training a NMT Retrieval Model

Let's delineate each step outlined in Figure 15 with its respective subtlety and nuances:

- The initial select makes two copies. Each of the input tokens represented by zero and the target tokens represented by one. Remember that here the input is English tokens, and the target is German tokens. One copy of the input tokens are fed into the inputs encoder to be transformed into the key and value vectors. While a copy of the target tokens goes into the pre-attention decoder.
- It is important to note here, that the pre-attention decoder is not the decoder we have shown earlier, which produces the decoded outputs. The pre-attention decoder is transforming the prediction targets into a different vector space called the query vector. That's going to calculate the relative weights to give each input weight. The pre-attention decoder takes the target tokens and shifts them one place to the right. This is where the teacher forcing takes place. Every token will be shifted one place to the right, and in start of a sentence token, will be a sign to the beginning of each sequence.
- Next, the inputs and targets are converted to embeddings or initial representations of the words. Now that we have our query, key and value vectors, we can prepare them for the attention layer.

[8]If you'd like to know more about Teacher Forcing, including pros/cons and a list of further resources. Available at: `https://towardsdatascience.com/what-is-teacher-forcing-3da6217fed1c` (Wong, 2019)

[9]Available at: `https://github.com/prateekchandrajha/ir-mini-project`

- We will also apply a padding mask to help determine the padding tokens. The mask is used after the computation of the Q, K transpose, just before computing the softmax. The where operator that you see in our code will convert the zero-padding tokens to negative one billion, which will become approximately zero when computing the softmax. That's how padding works.
- Now, everything is ready for the attention, inside which all the calculations that assign weights happen.
- The residual block adds the queries generated in the pre-attention decoder to the results of the attention layer. The attention layer then outputs its activations along with the mask that was created earlier. It's time to drop the mask before running everything through the decoder, which is what the second Select is doing. It takes the activations from the attention layer or the zero, and the second copy of the target tokens, or the two which we remember from way back at the beginning. These are the true targets which the decoder needs to compare against the predictions.
- Then run everything through a dense layer or a simple linear layer with your targets vocab size. This gives your output the right size.
- Finally, we will take the outputs and run it through LogSoftmax, which is what transforms the attention weights to a probability distribution between zero and one. These last four steps (including this one) comprise our decoder.
- The true target tokens are still hanging out here, and we'll pass them down along with the log probabilities to be matched against the predictions.

There we have it, the NMT retrieval model that we have tried to build in our code, and the intuition behind the many steps required to train such a large scale model. Now, we know what's actually happening at each part of the NMT retrieval model. In the next section, we go through the widely used evaluation mechanisms for such probabilistic translation cum retrieval models.

How to evaluate such Machine Translation/Retrieval Systems?

BLEU: Bilingual Evaluation Understudy

The BLEU score, which stands for a Bilingual Evaluation Understudy, is an algorithm that was developed to solve some of the most difficult problems in NLP, including Machine Translation. The BLEU score was proposed by Kishore Papineni, et al in their 2002 paper titled BLEU: a Method for Automatic Evaluation of Machine Translation[10]. It evaluates the quality of machine-translated text by comparing a candidate text translation to one or more of the reference translations. Usually, the closer the BLEU score is to one, the better your model is. The closer to zero, the worse it is. With that said, what is BLEU score and why is this an important metric? To get a BLEU score, the candidates and the references are compared based on an average of uni, bi, try or even four-gram precision. Mathematically, we can express the BLEU score as:

$$BLEU = BP\Big(\prod_{i=1}^{4} precision_i\Big)^{(1/4)}$$

with the Brevity Penalty and precision defined as:

$$BP = min\Big(1, e^{(1-(ref/cand))}\Big)$$

$$precision_i = \frac{\sum_{snt \in cand} \sum_{i \in snt} min\Big(m^i_{cand}, m^i_{ref}\Big)}{w^i_t}$$

[10] Available at: `https://www.aclweb.org/anthology/P02-1040.pdf`

where:

- m^i_{cand}, is the count of i-gram in candidate matching the reference translation.
- m^i_{ref}, is the count of i-gram in the reference translation.
- w^i_t, is the total number of i-grams in candidate translation.

Using the BLEU score as an evaluation metrics has some caveats. For one, it doesn't consider the semantic meaning of the words. It also doesn't consider the structure of the sentence. Imagine getting this translation, "Ate I was hungry because." If the reference sentence is, I ate because I was hungry, this would actually get a perfect BLEU score and this is highly problematic part of BLEU. Still BLEU score is the most widely adapted evaluation metric for machine translation.

ROUGE: Recall Oriented Understudy for Gisting Evaluation

Another similar method for evaluation of such systems is the ROUGE score which calculates precision and recall for machine texts by counting the n-gram overlap between the machine texts and a reference text.

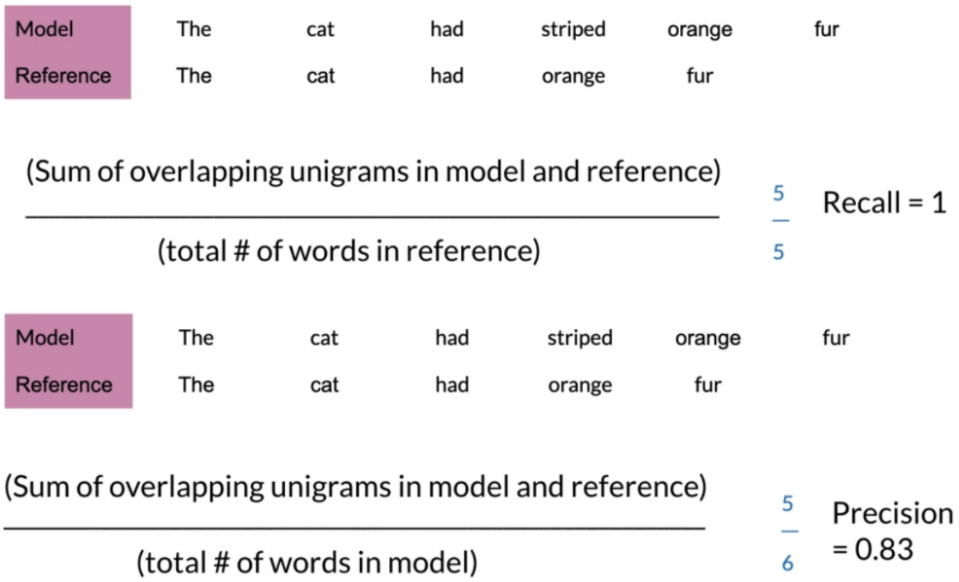

Figure 16: Calculation of ROUGE Precision & Recall for Machine Texts

How do we construct the final translations after training such a NMT model? What Retrieval methods work the best?

In this final section, we see more or less two ways that will allow us to construct the final translated sentence. The first approach is known as greedy decoding, and the second approach is known as random sampling. We will see the pros, and cons of each method. These methods are not full-proof and a bit nuanced e.g. when choosing the word with the highest probability at every time, that does not necessarily generate the best sequence of translations. We will explore why in the upcoming parts. So let's dive deeper into a few methods for sampling, and decoding, as well as a discussion of an important hyperparameter in sampling called the temperature. Firstly, let us remind ourselves of where our model is in the process when undertaking sampling, and when decoding comes into play (See Figure 17 and caption below).

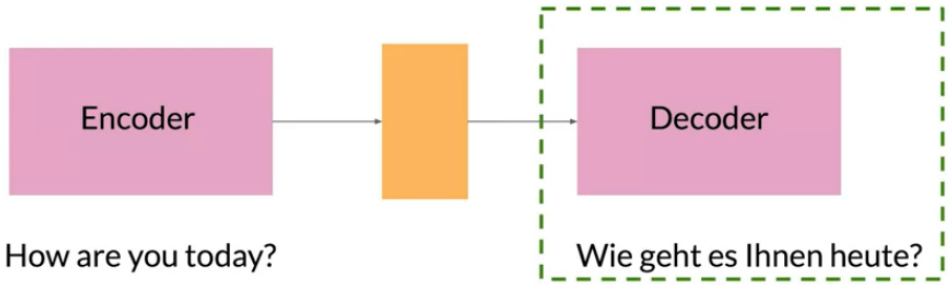

Figure 17: After all the necessary calculations have been performed on the encoder hidden states, and our model is ready to predict the next token, how will we choose to do it? With the most probable token or by taking a sample from a distribution? It's a bit tricky as in what to choose at this step.

Greedy Decoding

Greedy decoding is the simplest way to decode the model's predictions as it selects the most probable word at every step. However, this approach has limitations. When you consider the highest probability for each prediction, and concatenate all predicted tokens for the output sequence as the greedy decoder does, you can end up with a situation where the output instead of I am hungry, gives you I am, am, am, am (See Figure 18 below). We can see how this could be a problem but not in all the cases. For shorter

Figure 18: Probabilistic retrieval can fail to meet our intuitive expectations

sequences this should work just fine but if we have many other words to consider from, then knowing what's coming up next might help us better in predicting the next sequence.

Random Sampling

Another option is known as random sampling. What random sampling does is provide probabilities for each word, and sample accordingly for the next outputs. One of the problems with this is that it could be a little bit too random. A solution for this is to assign more weight to the words with higher probability, and less weight to the others. We will shortly see a method for doing this.

Temperature

In sampling, temperature is a parameter you can adjust to allow for more or less randomness in your predictions. It's measured on a scale of 0-1, indicating low to high randomness. We need our model to make careful, and safe decisions about what to output so set temperature lower, and get the prediction equivalent of a very confident but rather boring person seated next to you at dinner. If you feel like taking more of a gamble, set your temperature a bit higher. This has the effect of making your network more excited, and you may get some pretty fun predictions. On the other hand, there will probably be a lot more mistakes.

Beam Search Decoding

Previously, we have seen the greedy decoding algorithm which just selects one best candidate as an input sequence for each time stamp. The model has already encoded the input sequence, and used the previous time steps translation to calculate how much attention to give each of the input words. Now it's using the decoder to predict the next translated word. Now choosing just one best candidate might be suitable for the current time step but when we construct the full sentence, it maybe a sub-optimal choice. Beam search decoding is a more exploratory alternative for decoding that uses a type of restricted breadth-first search (BFS) to build a search stream. Instead of offering a single best output like in greedy decoding, beam search selects multiple options based on conditional probability. The search restriction we mentioned a moment ago is the beam width parameter B, which limits the number of branching paths based on a number that we choose, such as let us say 3. Then at each time step, the beam search selects B number of best alternatives with the highest probability as the most likely choice for the time step. Once you have these B possibilities, you can choose the one with the highest probability. This is a beam search decoding, which doesn't look only at the next output but instead applies a beam width parameter to select several possible options. Let's take a look at an example sequence where the beam width parameter is set to three. The beam width parameter is a defining feature of beam search, and it controls the number of beam searching through the sequence of probabilities. Setting this parameter works in an intuitive way. A larger beam width will give you better model performance but slower decoding speed. This is a very simple example but you can see for yourself how beam search makes

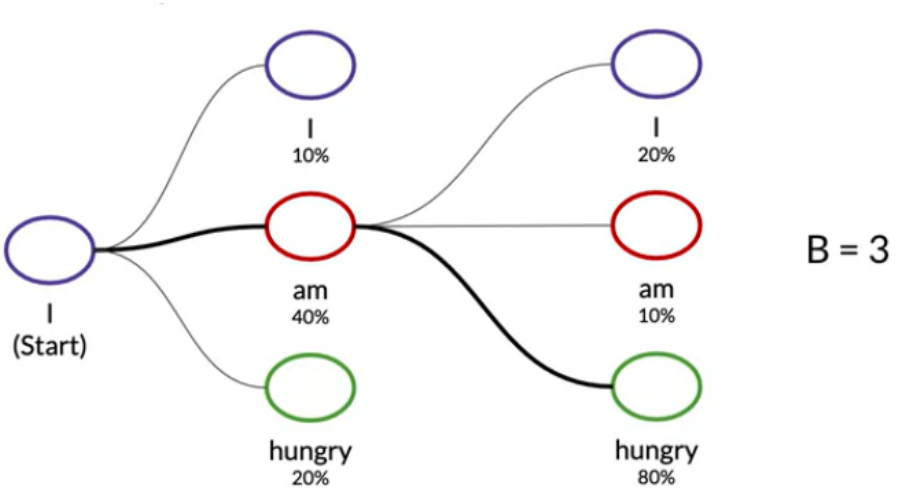

Figure 19: Provided with the first token I, and the beam width parameter of three i.e. $B = 3$, beam search assigns conditional probabilities to each of several options for the next word in the sequence. The highest probability is the one that will be chosen for each time step, and the other options will be pruned. It's determined that "am" is the most likely next token in the sequence with a probability of 40 percent. For the third, and final time step, beam search identifies hungry as the most likely token with probability around 80 percent. Does this sentence construction make more sense than any of the other options?

a more powerful alternative to greedy decoding for machine translation of longer sequences. However,

beam search decoding runs into issues where the model learns a distribution that isn't useful or accurate in reality. It can also use single tokens in a problematic way, especially for unclean corpora. Imagine having training data that is not clean e.g. from a speech corpus. If you have the filler word "Uhm" which appears as a translation in every sentence with one percent probability that single element can throw off your entire translation. Imagine now that you have 11 good translations of Vereinigten Staaten, which is German for the United States. These could be USA, US, US of A etc, compared to your German inputs. So in total we have 11^2, at least good translations, each with the same probability because they're all equal. The most probable one is the filler word, Uhm, and that because $1/11^2$ is less than 0.01% so that ends up being the most probable outcome. There are still other alternatives to consider.

Minimum Bayes Risk (MBR)

Earlier we encountered random sampling as a way to choose a probable token, and the issues with that very simple implementation. But if you go a little further with that, say, by generating 30 samples, and comparing them all against one another to see which one performs the best, you'll see quite a bit of improvement in your decoding. This is called Minimum Bayes Risk decoding or MBR for short. Implementing MBR is pretty straightforward:

- Begin by generating several random samples
- Then we compare each sample against all its mates, and assign similarity score for each comparison. ROUGE is a good one that you may recall from last section on evaluation of such systems
- Finally, choose the sample with the highest similarity, which is sometimes referred to as the golden one

Now let us enumerate the steps for implementing MBR on a small set of 4 samples (to illustrate what we have done in our code with sizes up north of 30 samples):

- First, calculate the similarity between the first and the second sample
- Then for the first and the third sample
- Then calculate again for the first and the fourth sample
- Then take the average of those three similarity scores. It is more common to use a weighted average when we compute MBR
- Then you'll repeat this process for the other three samples in your set
- At last, the top performer out of all of them will be chosen, and that's it for MBR

We have implemented this one in our code along with a greedy decoder. Let's recap this final section to understand the retrieval methods used for creating final translations. Now we are aware that the beam search uses a combination of conditional probabilities, and a beam width parameter to offer more options for inputs at each time step. An alternative to beam search, the Minimum Bayes Risk, takes several samples, and compares them against themselves, then chooses the best performer. This can give us a more contextually accurate translation.

Associated Project Code, Demos & Notebooks

Please visit my github repository at `https://github.com/prateekchandrajha/ir-mini-project/` in order to go through the associated code and jupyter notebook demos for various concepts (See the README file on github). I've tried to learn over the timeframe of this project. This was a fairly complex retrieval model to build from end-to-end and I hope the reader of this report draws a clear high-level picture of what goes into designing such attention models. I'd take this opportunity to thank Prof. Venkatesh Vinayakarao[11] who motivated all the fundamental concepts which serve as the foundations for such large-scale systems.

[11] Information Retrieval Course Resources. Available at: `https://vvtesh.github.io/teaching/IR-2020.html`