Outline

- Introduction to Neural Machine Translation
- Seq2Seq model and its shortcomings
- Solution for the information bottleneck

Neural Machine Translation

How are you today? → Wie geht es Ihnen heute?

Seq2Seq model

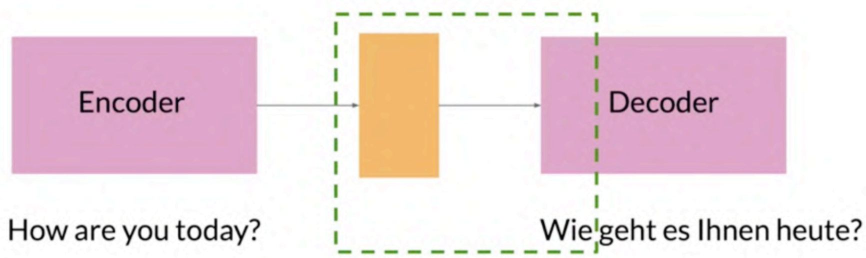
- Introduced by Google in 2014
- Maps variable-length sequences to fixed-length memory
- LSTMs and GRUs are typically used to overcome the vanishing gradient problem

Seq2Seq model

An encoder/decoder looks like this. It takes in a hidden state and a string of words such as a single sentence. The encoder takes the input one step at a time, collects information for that piece of input, then moves it forward.

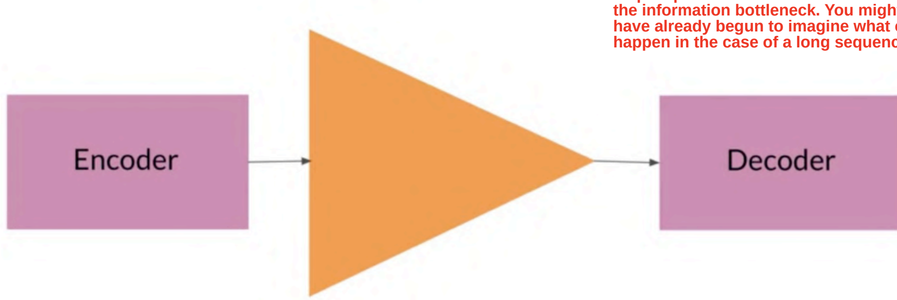
Seq2Seq model

The orange rectange represents encoder's final hidden state 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 state for the decoder to begin predicting the sequence.



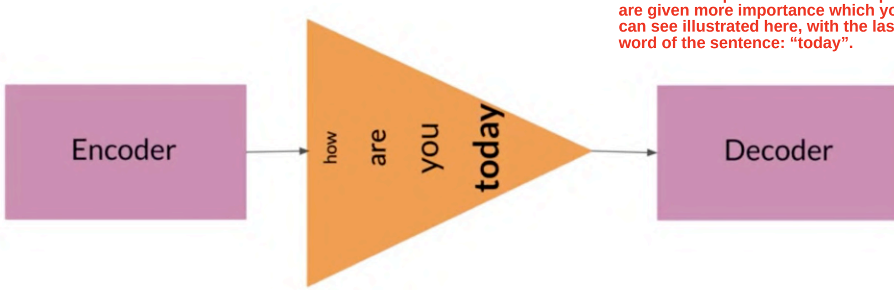
The information bottleneck

One major limitation of the the traditional seq2seq model is what is referred to as the information bottleneck. You might have already begun to imagine what can happen in the case of a long sequence.



The information bottleneck

As individual inputs begin stacking up inside the encoder's final hidden state because seq2seq uses a fixed length memory. Longer sequences become problemmatic. Another issue surfaces as the later inputs that's in the sequence are given more importance which you can see illustrated here, with the last word of the sentence: “today”.



All of this results in a model that performs incredibly well for shorter sequences and not so well for longer, more complex sequences

---

Seq2Seq shortcomings

- Variable-length sentences + fixed-length memory =

The power of seq2seq which lies in its ability to let inputs and outputs be difference sizes becomes its weakeness when the input itself is of a large size becuase the encoder hidden state is of a fixed size and longer inputs become bottlenecked on their way to the decoder

This results in lower model performance as sequence size increases

- As sequence size increases, model performance decreases

The issue with having one fixed size encoder hidden state is that it struggles to compress longer sequencce and ends up throttling itself and punishing the decoder who only wants to make good predictions.

You can be nicer to the decoder by holding onto each word vector with its individual information instead of trying to smush it all into one big vector. This model still have obvious flaws with memory and context. How could you build a time and memory efficient model that predicts accurately from a long sequence?

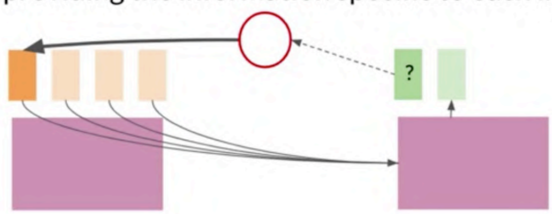
One vector per word

How are you today?

Wie geht es Ihnen heute?

Solution: focus attention in the right place

- Prevent sequence overload by giving the model a way to focus on the **likeliest** words at each step
- Do this by providing the information specific to each input word



This becomes possible if the model has a way to select and focus on the likeliest word at 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 in the right place at each step.

Motivation for alignment

Before end to end neural machine translation alignments was very critical when translating one language to another language. Alignment is still widely used today for word sense disambiguation or word sense discovery.

If your model needs to be able to focus in the right place, so it's can choose the right output to predict. Then it makes a lot of sense that you would want the words of the inputs to align what 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.

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

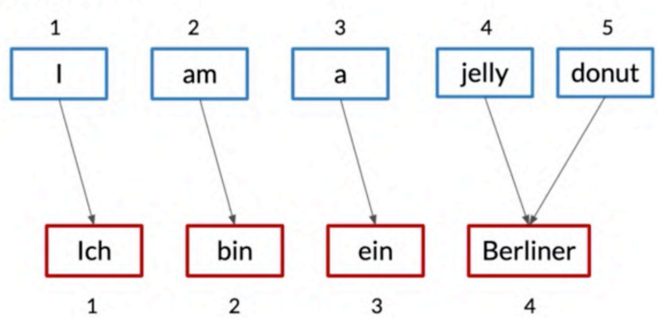
Correctly aligned words are the goal:

- Translating from one language to another
- Word sense discovery and disambiguation

- Achieve alignment with a system for retrieving information step by step and scoring it

---

Word alignment

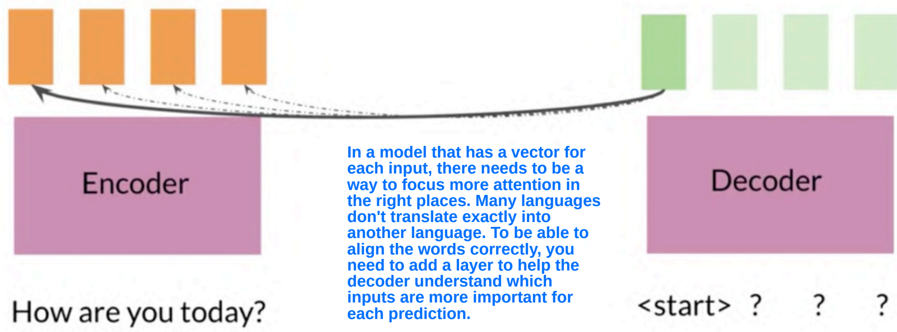


Let's take a look at how words align any sentence, where each word is not exactly the same when translating from English to German.

So the example given here is from a famous speech John F Kennedy gave in Berlin during the Cold War, which caused some confusion. Because the term Berliner can denote either a citizen of Berlin or a Jelly donut. For all my word nerds out there, look for an optional reading about this historic misunderstanding In the course material.

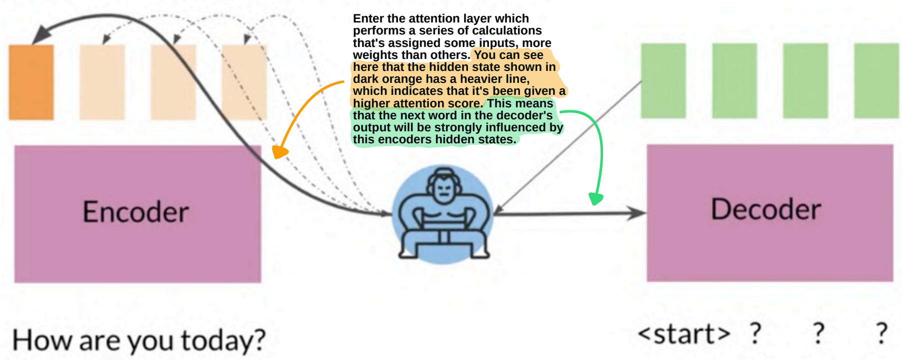
But I digress, notice how the English translation has more words than the German version. When performing word alignment, your model needs to be able to identify relationships among the words in order to make accurate predictions in case the words are out of order or not exact translations.

Which word to pay more attention to?



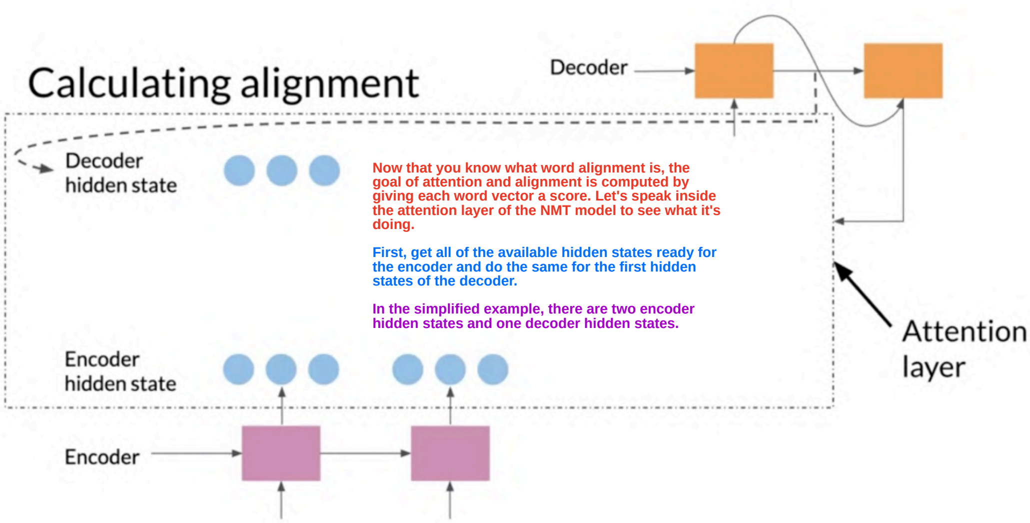
In a model that has a vector for each input, there needs to be a way to focus more attention in the right places. Many languages don't translate exactly into another language. To be able to align the words correctly, you need to add a layer to help the decoder understand which inputs are more important for each prediction.

Give some inputs more weight!



Enter the attention layer which performs a series of calculations that's assigned some inputs, more weights than others. You 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.

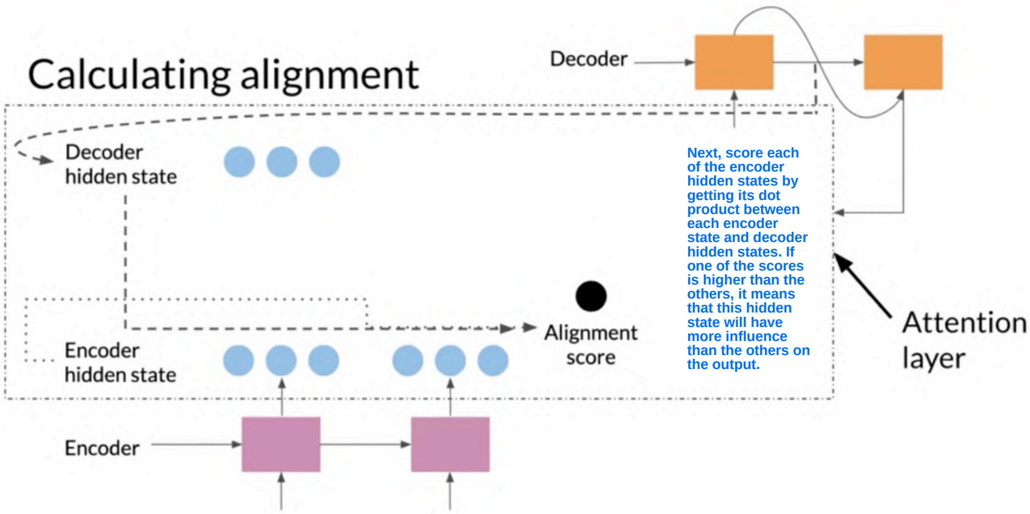
Calculating alignment

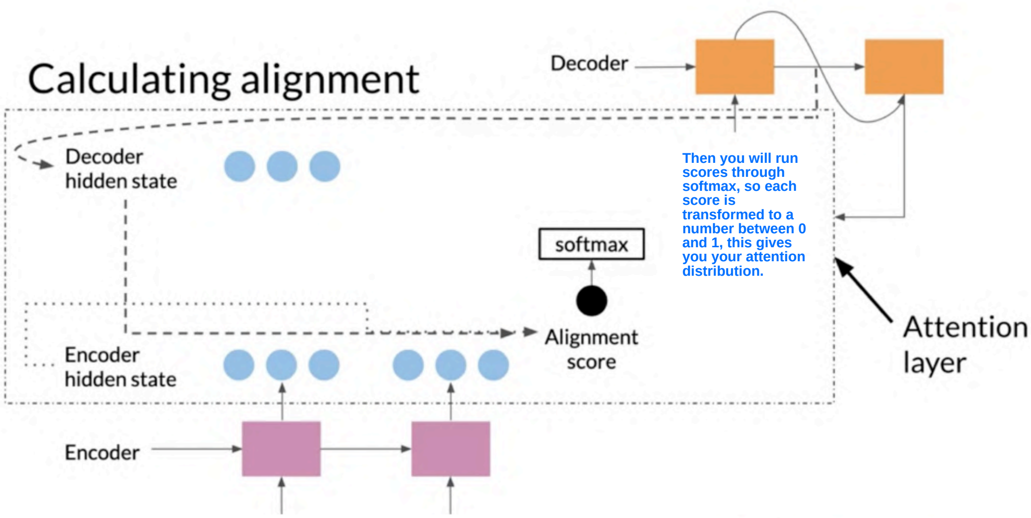


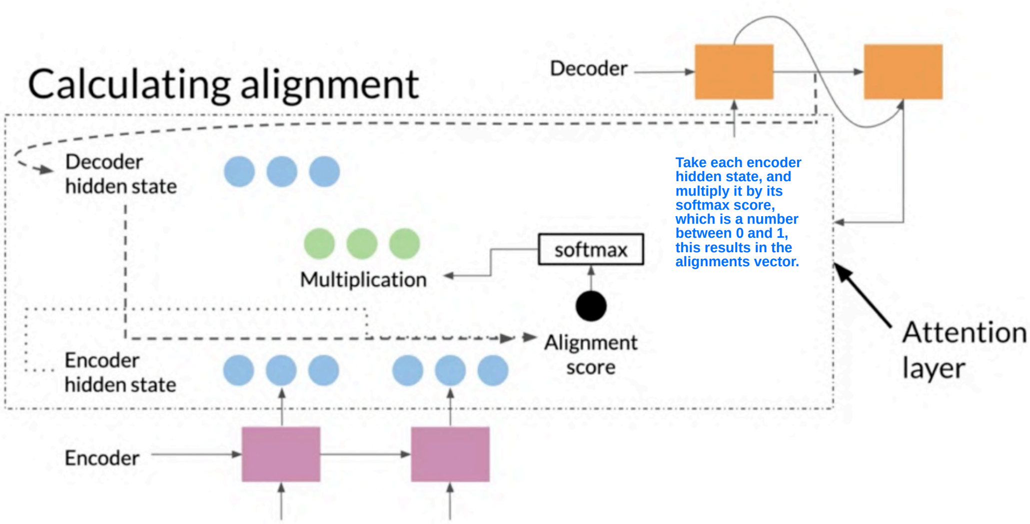
Now that you know what word alignment is, the goal of attention and alignment is computed by giving each word vector a score. Let's speak inside the attention layer of the NMT model to see what it's doing.

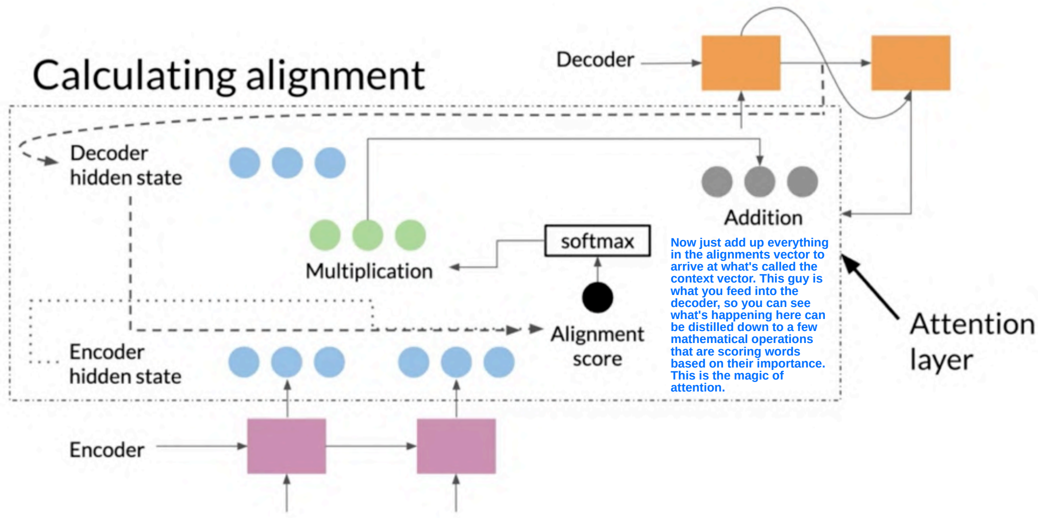
First, get all of the available hidden states ready for the encoder and do the same for the first hidden states of the decoder.

In the simplified example, there are two encoder hidden states and one decoder hidden states.


Calculating alignment
Decoder
Decoder hidden state
Encoder hidden state
Alignment score
Encoder
Attention layer
Next, 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.


Calculating alignment
Decoder
Decoder hidden state
softmax
Encoder hidden state
Alignment score
Encoder
Attention layer
Then you will run scores through softmax, so each score is transformed to a number between 0 and 1, this gives you your attention distribution.

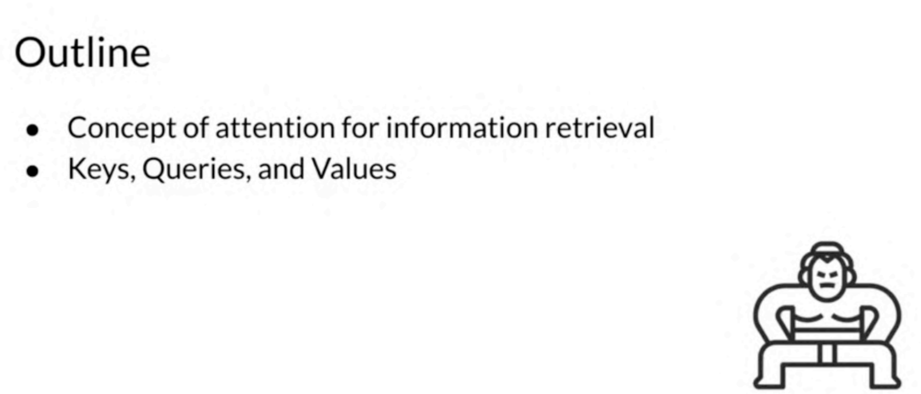

Calculating alignment
Decoder
Decoder hidden state
Multiplication
softmax
Encoder hidden state
Alignment score
Encoder
Attention layer
Take each encoder hidden state, and multiply it by its softmax score, which is a number between 0 and 1, this results in the alignments vector.



(Optional): The Real Meaning of Ich Bin ein Berliner

Here's an article from The Atlantic that discusses the famous JFK speech containing the words "Ich bin ein Berliner," the example you saw in the Alignment video.

https://www.theatlantic.com/magazine/archive/2013/08/the-real-meaning-of-ich-bin-ein-berliner/309500/ (Putnam, 2013)



Information retrieval

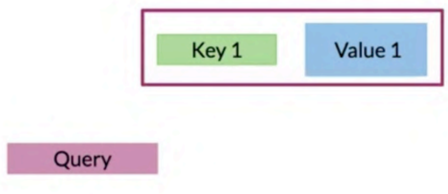
Say you're looking for your keys.

You ask your mom to help you find them.

She weighs the possibilities based on where the keys usually are, then tells you the most likely place.

This is what Attention is doing: using your query to look in the right place, and find the key.

Inside the Attention Layer

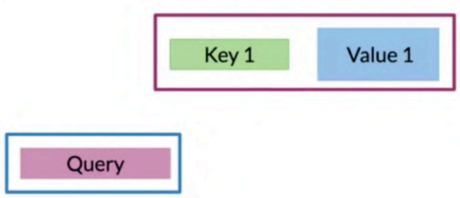


The attention mechanism uses encoded representations of both the input or the encoder hidden states and the outputs or the decoder hidden states.

The keys and values are pairs. Both of dimension N, where N is the input sequence length and comes from the encoder hidden states.

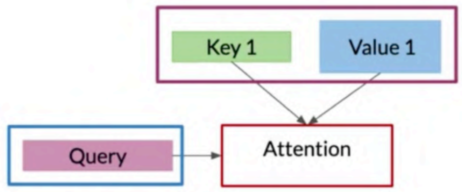
Keys and values have their own respective matrices, but the matrices have the same shape and are often the same.

Inside the Attention Layer



While the queries come from the decoder hidden states

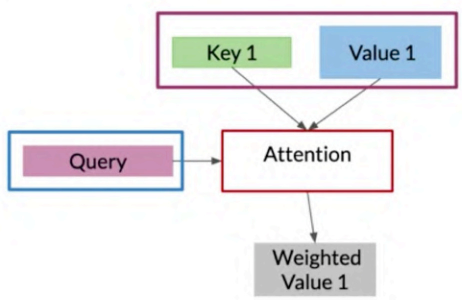
Inside the Attention Layer



Both the key value pair and the query enter the attention layer from their places on opposite ends of the model. And once they're inside, the dot product of the querying and the key is calculated.

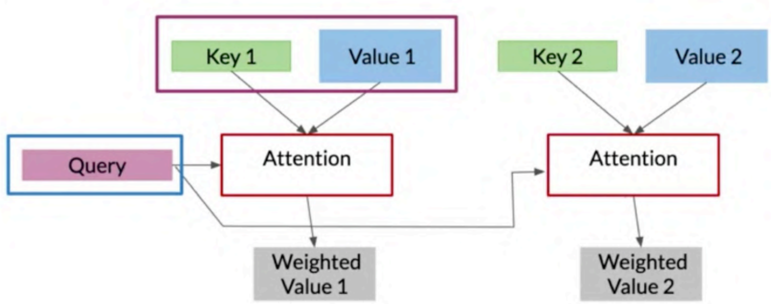
This is essentially a measure of similarity between them, and the dot product of similar vectors tends to have a higher value.

Inside the Attention Layer



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 wait through the softmax, so there transform to fit a distribution of numbers between zero and one.

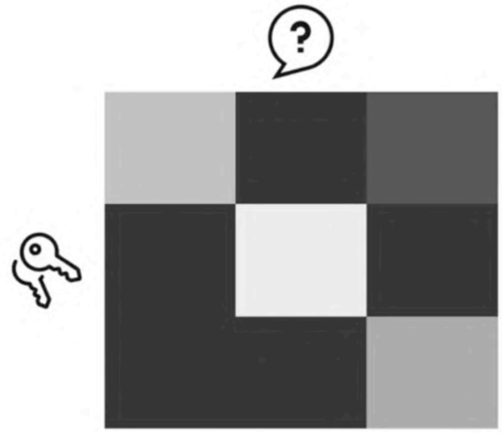
Inside the Attention Layer



Then, the query is mapped to the next key value pair and so on and so forth. This so is called scale dot product attention.

Attention

Keys and queries = 1 matrix with the words of one query (Q) as columns and the words of the keys (K) as the rows



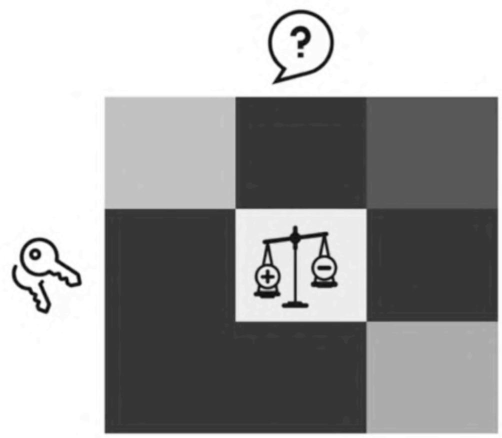
Let's visualize how the attention weights look in matrix form. You have all the inputs or words of 1, query Q is columns and the words of the keys (K) as the rows.

Attention

Keys and queries = 1 matrix with the words of one query (Q) as columns and the words of the keys (K) as the rows

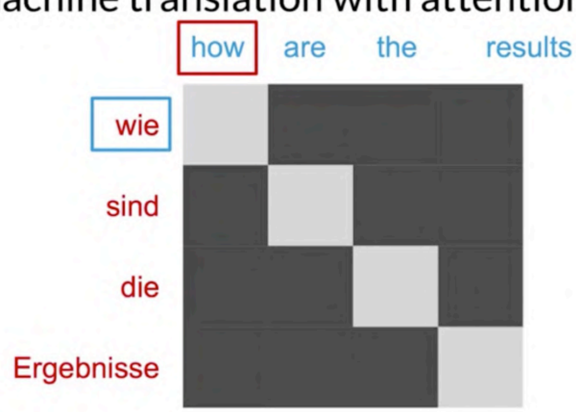
Value score (V) assigned based on the closeness of the match

Attention = *Softmax*(QK^T)V



That's value score that you'll remember as the weighted sum from the previous slide is indicated here in a lighter shade of gray to show the highest score and therefore the match

Neural machine translation with attention

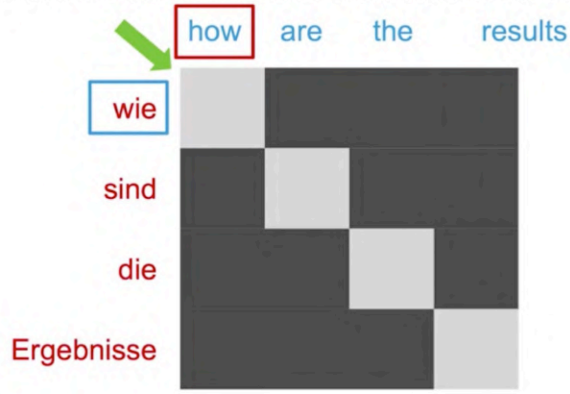


Here's an example that shows where the model is looking when translating between English and German, as you'll be doing in your assignments. In this rather straightforward example, attention is first translating the English word how, into the German word wie. When attention is translating, the word are, is looking at the word sind, and so on.

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 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. And one language requires a different word ordering, then the other language does.

Neural machine translation with attention



In the matrix, the lighter square shows where the model is actually looking when making the translation of that word. I'll show you how to build an attention matrix like this one. You will learn how to properly interpret the attention matrix and how to use it to make the word predictions.

Flexible attention

For languages with different grammar structures, attention still looks at the correct token between them

In a situation like the one I just mentioned, where the grammar of foreign language requires a difference word order than the other, the attention is so flexible enough to find the connection. The first four tokens, the agreements on the, are pretty straightforward.

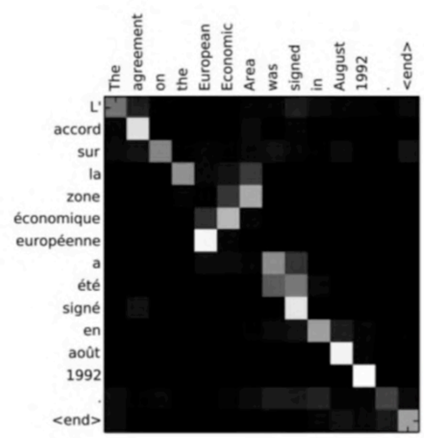


image © (Bahdanau et al., 2015)

Flexible attention

For languages with different grammar structures, attention still looks at the correct token between them

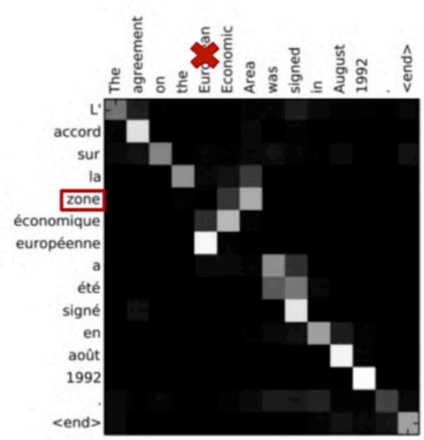


image © (Bahdanau et al., 2015)

Flexible attention

For languages with different grammar structures, attention still looks at the correct token between them

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, glorious and necessary. It's pretty amazing, was a little matrix multiplication can do.

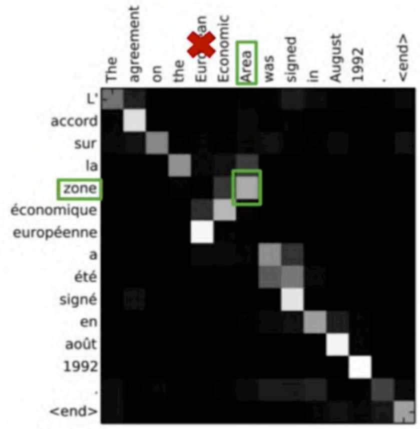


image © (Bahdanau et al., 2015)

Summary

- Attention is an added layer that lets a model focus on what's important
- Queries, Values, and Keys are used for information retrieval inside the Attention layer
- This flexible system finds matches even between languages with very different grammatical structures

Data in machine translation

English	German
I am hungry!	Ich habe Hunger.
...	...
I watched the soccer game.	Ich habe das Fußballspiel gesehen.

Attention! (no pun intended) Assignment dataset is not as squeaky-clean as this example and contains some Spanish translations.

Machine translation setup

State-of-the-art uses pre-trained vectors

Otherwise, represent words with a one-hot vector to create the input

Keep track of index mappings with word2ind and ind2word dictionaries

Use start-of and end-of sequence tokens:

<SOS>  <EOS>

Preparing to Translate to German

This is English sentence and then the tokenized version of the English sentence, you can see that it has an index of 4546 for the word both after the initial tokenization just add the EOS token shown here as one, and pad with zeros.

You can see the one at the end of the sequence. Then a series of zeros that's if you were to count, would be equal to the number of unpaired tokens in the sequence.

In other words, until the size of the longest sentence aloud is reached. These tokens will later be used to produce a matrix for the embedding vectors.

ENGLISH SENTENCE:

Both the ballpoint and the mechanical pencil in the series are equipped with a special mechanism: when the twist mechanism is activated, the lead is pushed forward.

TOKENIZED VERSION OF THE ENGLISH SENTENCE:

[4546 4 11358 362 8 4 23326 20104 1745 8210 9641 5 6
4 3103 31 2767 30 13 914 4797 64 196 4 22474 5 4797 16
24864 86 2 4 1060 16 6413 1138 3 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

English to German

Now let's go to the German translation of that sequence. Along with the tokenized version of the German translation. Notice that one is the end of token here two. It's also followed with a series of zeros. This are also equal to the number of unpaired tokens in the sequence.

This is the general setup process you would use in the real world, but I'll give you a pre-built package that will take care of this process. All you have to do is load this package and give it the sequences, and it'll give you a corresponding tokenized version of each sentence.

GERMAN TRANSLATION:

Der Kugelschreiber und der Drehbleistift der Serie sind mit einem besonderen Mechanismus ausgestattet: Bei Betätigung der Drehmechanik wird die Schreibmine nach vorne geschoben.

TOKENIZED VERSION OF THE GERMAN TRANSLATION:

[149 3892 5280 14774 2418 12 11 9883 6959 7298 15157 5 11 8453 75
39 114 5324 10565 2520 64 752 12954 26538 147 11 9883 23326 20104
300 78 10 21150 10166 126 14566 5 23850 1171 3 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

Outline

- Teacher forcing
- Model for NMT with attention

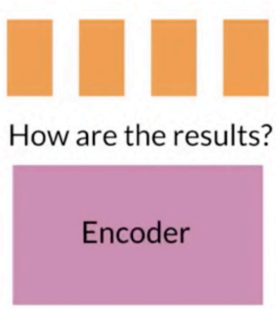
How to know predictions are correct?

Teacher forcing allows the model to "check its work" at each step

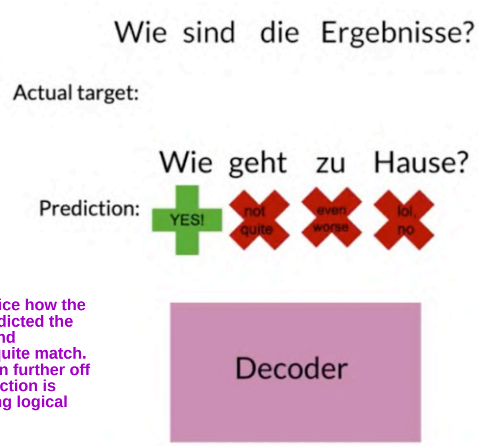
Or, compare its prediction against the real output during training

Result: Faster, more accurate training

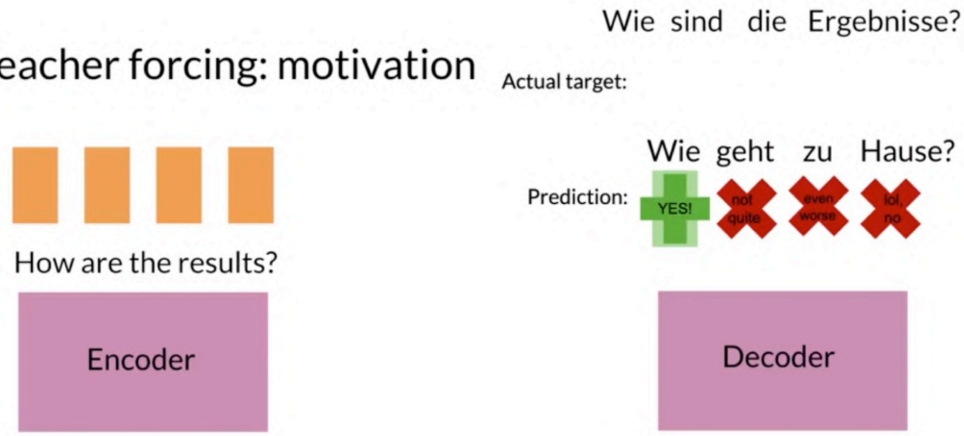
Teacher forcing: motivation



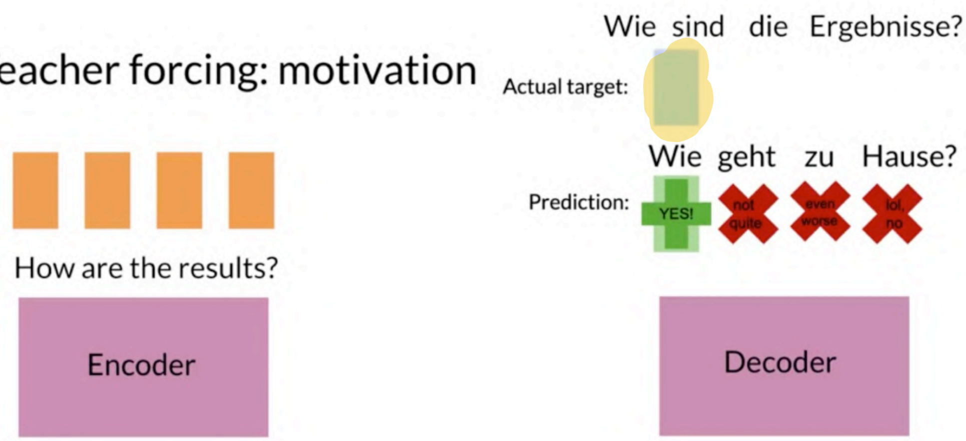
In this example, notice how the model correctly predicted the token. But the second prediction doesn't quite match. The third one is even further off and the fourth prediction is quite far from making logical sense in German



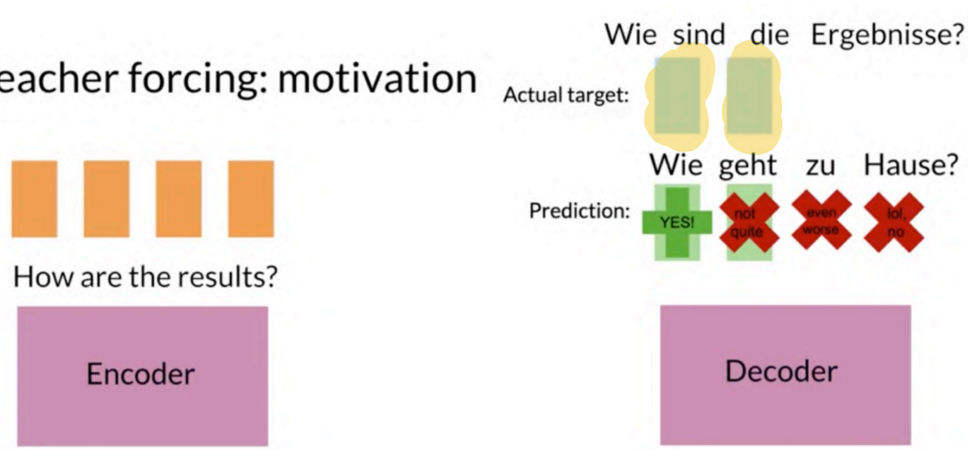
Teacher forcing: motivation

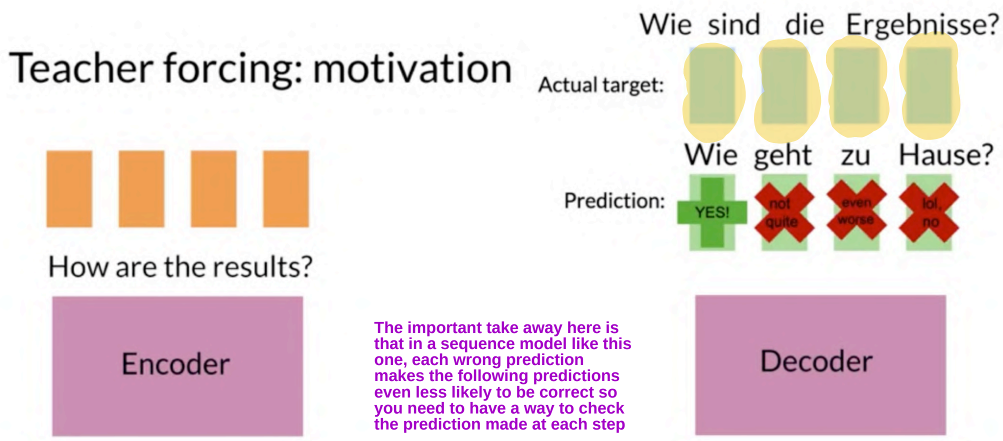


Teacher forcing: motivation



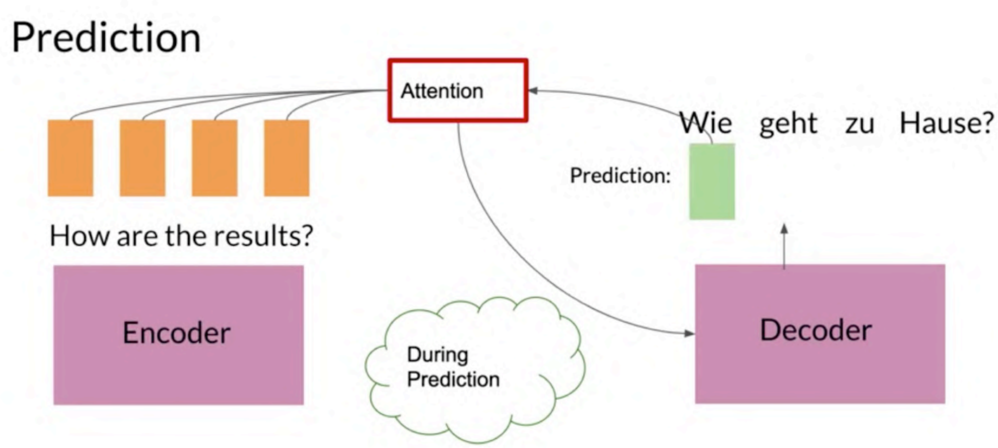
Teacher forcing: motivation

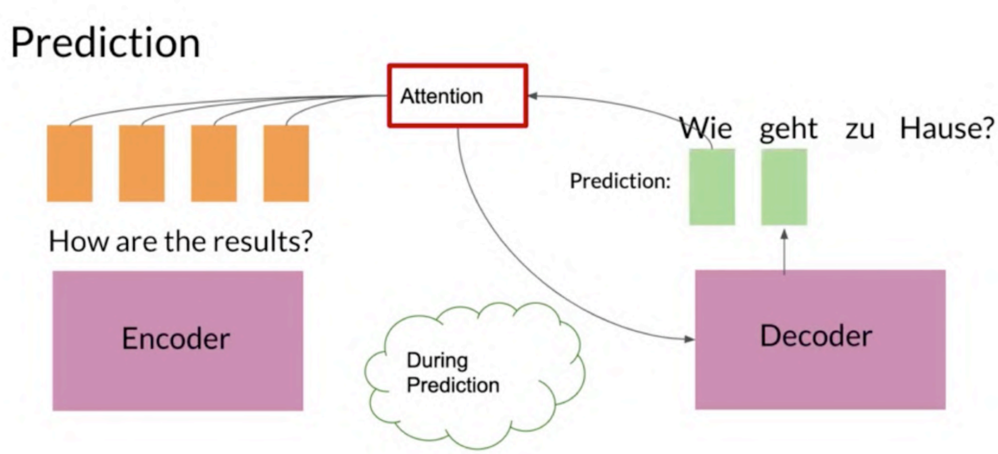


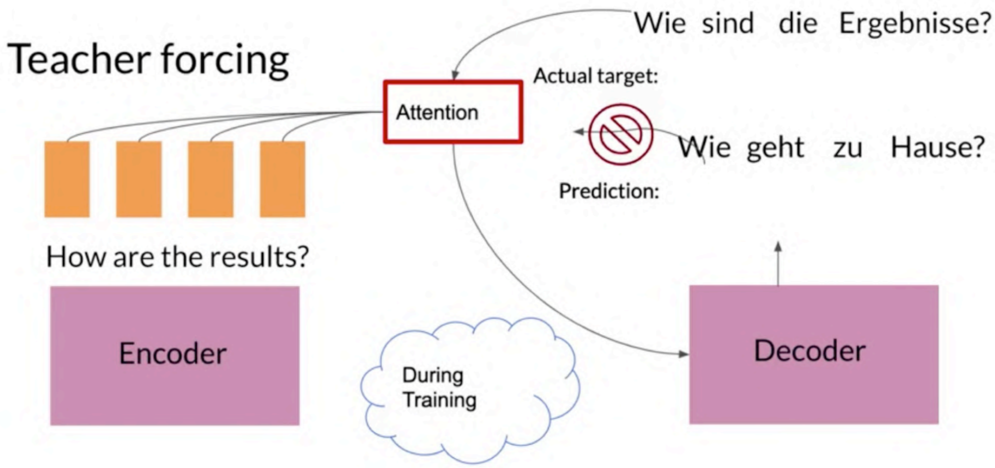


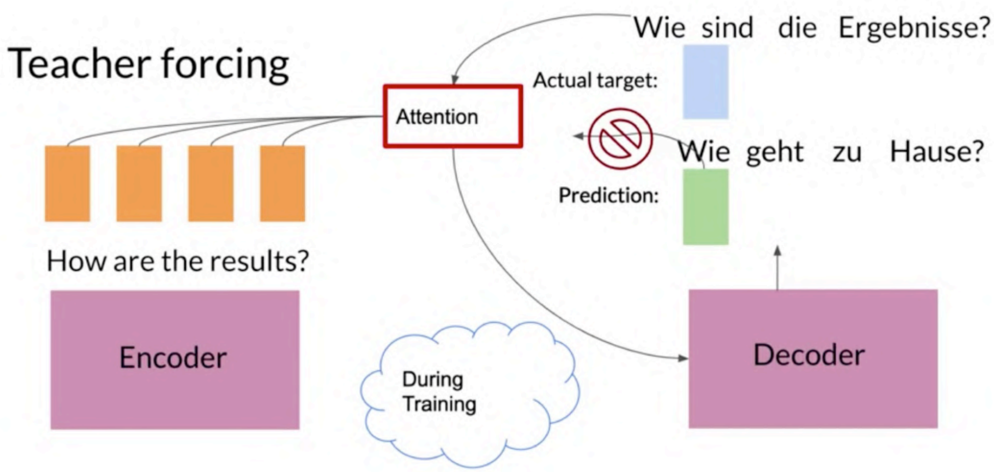
The important take away here is that in a sequence model like this one, each wrong prediction makes the following predictions even less likely to be correct so you need to have a way to check the prediction made at each step

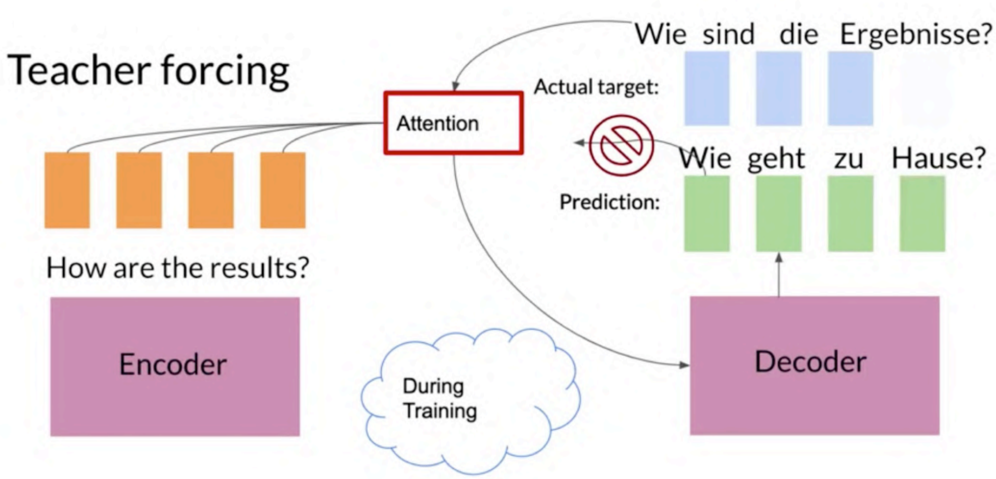
The T1 rectangle shown



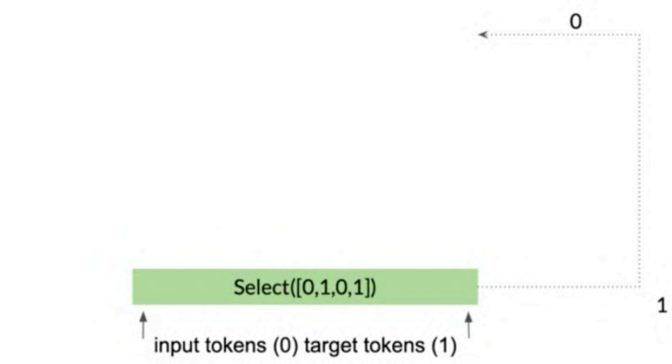



Teacher forcing
Wie sind die Ergebnisse?
Actual target:
Attention
Wie geht zu Hause?
Prediction:
How are the results?
Encoder
During Training
Decoder


Teacher forcing
Wie sind die Ergebnisse?
Actual target:
Attention
Wie geht zu Hause?
Prediction:
How are the results?
Encoder
During Training
Decoder

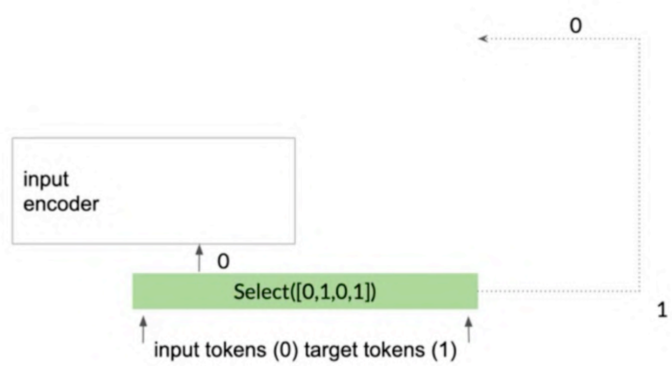

Teacher forcing
Wie sind die Ergebnisse?
Actual target:
Attention
Wie geht zu Hause?
Prediction:
How are the results?
Encoder
During Training
Decoder

Training NMT



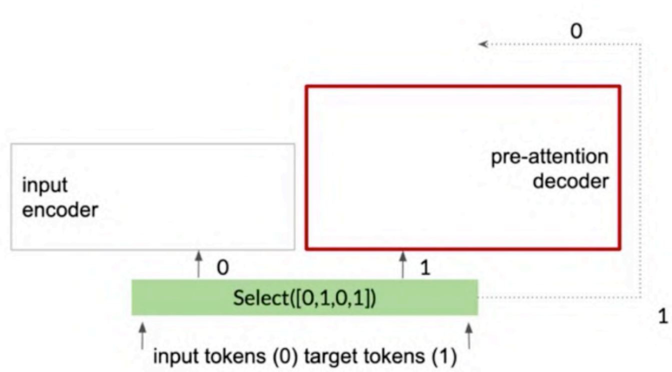
Let's put together everything we've seen so far. The initial select makes two copies each of the inputs tokens represented by 0 and the target tokens represented by 1. Remember that here the input is English tokens and the target is German tokens.

Training NMT



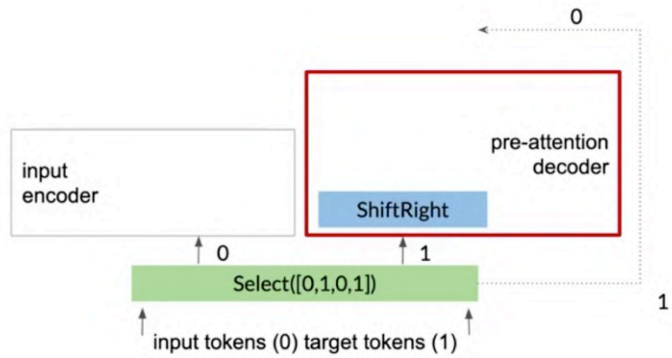
One copy of the input tokens are fed into the inputs encoder to be transformed into the key and value vectors.

Training NMT



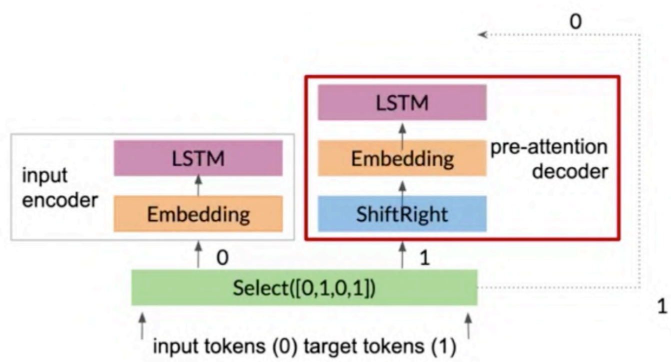
While a copy of the target tokens goes into the pre-attention decoder. Important note here, the pre-attention decoder is not the decoder you were shown earlier which produces a decoded output. The pre-attention decoder is transforming the prediction target into a different vector space called the query vector so that it can calculate the relative weight to give each input word.

Training NMT



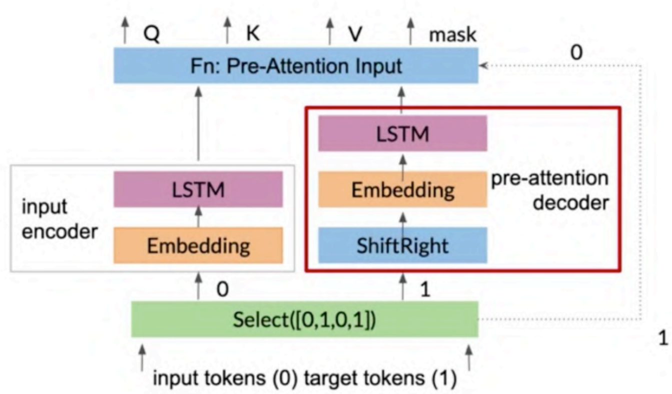
The pre-attention decoder takes the target tokens and shifts them one place to the right. This is where the teacher forcing takes place. Every token will be shifted one place to the right and it starts with a sentence token that will be assigned to the beginning of each sequence.

Training NMT

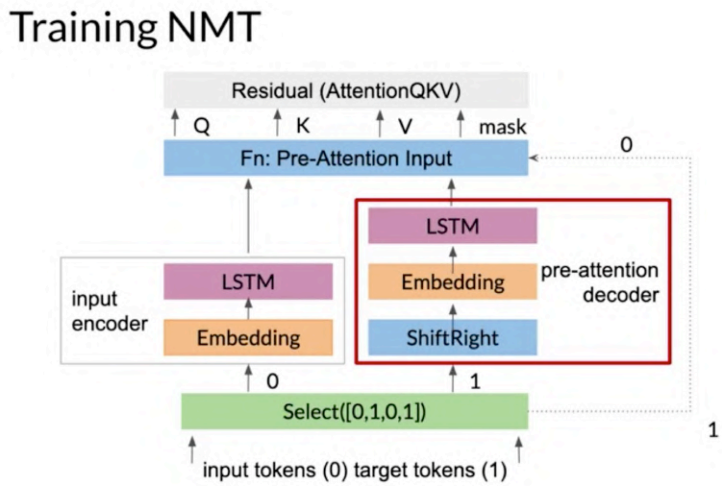


Next, the inputs and targets are converted to embeddings or initial representations of the words.

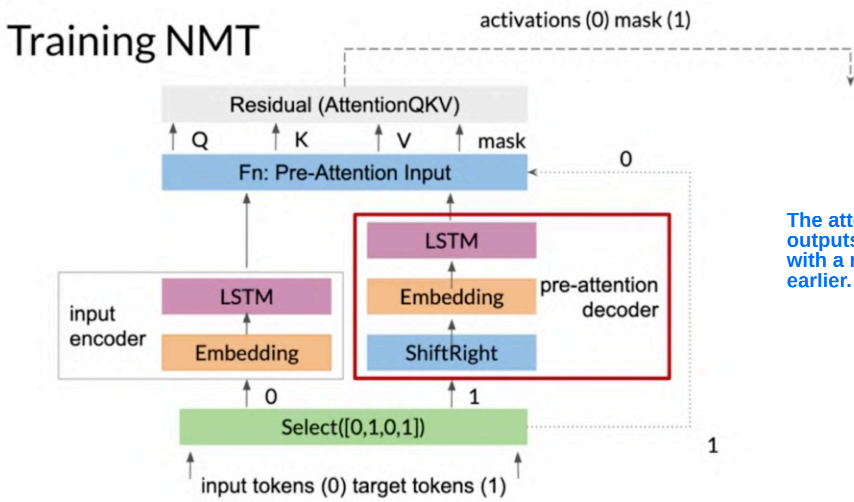
Training NMT



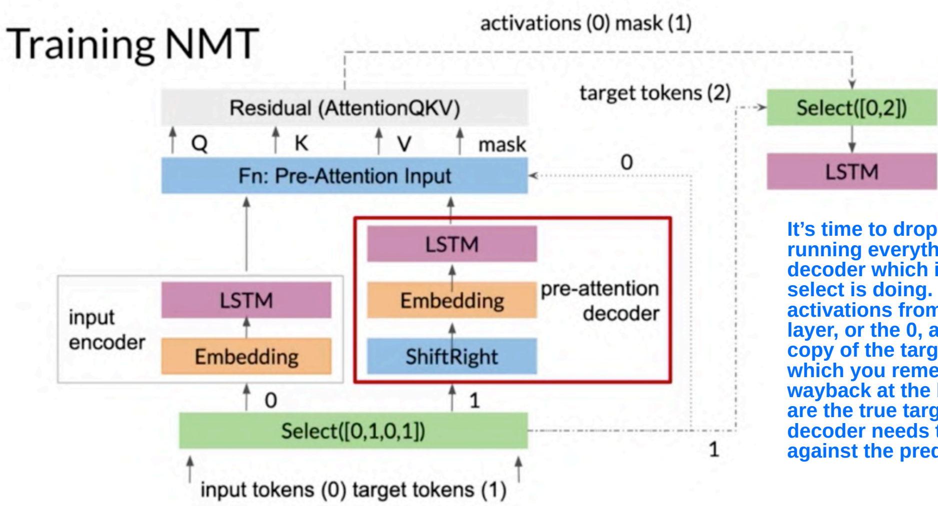
Now that you have your query key and value factors, you can prepare them for the attention layer. You'll also apply a padding mask to help determine the padding tokens. The mask is used after the computation of the QK transpose just before computing the softmax. The where operator in your programming assignments will convert the zero padding tokens to negative one billion which will become approximately 0 when computing the softmax. So that's how padding works



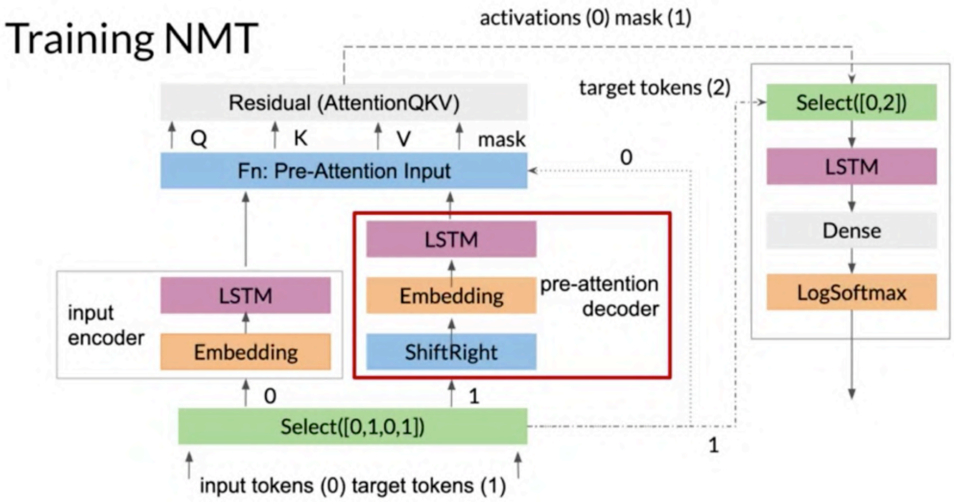
Now everything is ready for the attention. Inside, where all of the calculations that assign weights happen, the residual block adds the queries generated in the pre-attention decoder to the results of the attention layer.



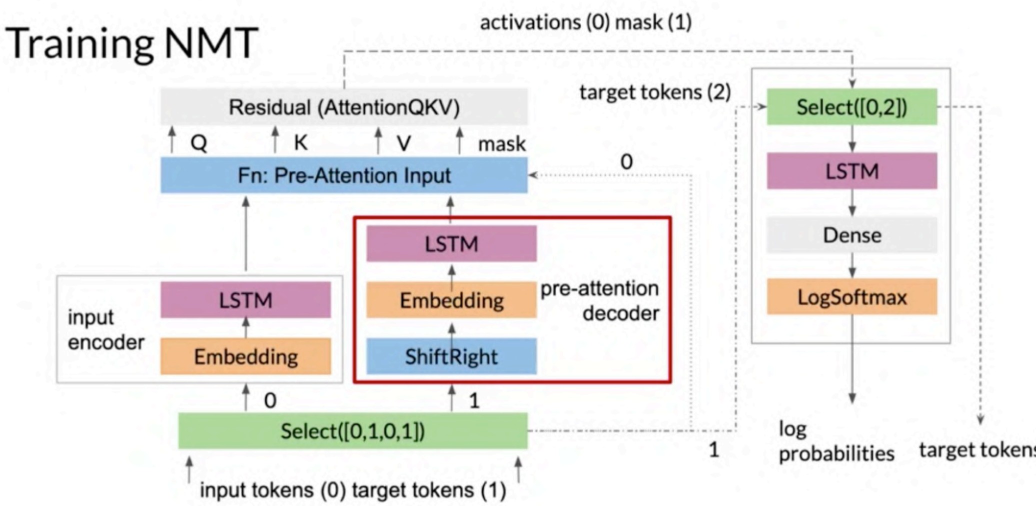
The attention layer then outputs it’s activations along with a mask that was created earlier.



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 0, and the second copy of the target tokens, or the 2, which you remember from wayback 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 target vocab size. This gives your output the right size.



Finally, you'll take the outputs and run it through log softmax which is what transforms the attention weights to a distribution between 0 and 1.Those last four steps comprise your decoder.

The true target tokens are still hanging out here and will pass down along with the log probabilities to be matched against the predictions.

(Optional) What is Teacher Forcing?

Here's an additional resource from the blog Towards Data Science if you'd like to know more about Teacher Forcing, including pros/cons and a list of further resources.

https://towardsdatascience.com/what-is-teacher-forcing-3da6217fed1c (Wong, 2019)

The BLEU score, which stands for a Bilingual Evaluation Understudy. It's an algorithm that was developed to solve some of the most difficult problems in NLP, including Machine Translation. It's evaluates the quality of machine-translated text by comparing a candidate texts translation to one or more 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?

BLEU Score

Stands for Bilingual Evaluation Understudy

Evaluates the quality of machine-translated text by comparing "candidate" text to one or more "reference" translations.

Scores: the closer to 1, the better, and vice versa:

BLEU Score

To get a BLEU score, the candidates and the references are usually based on an average of uni, bi, try or even four-gram precision. To demonstrate, I'll use uni-grams as an example.

Candidate	I	I	am	I	I
Reference 1	Younes	said	I	am	hungry
Reference 2	He	said	I	am	hungry

BLEU Score

Let's say that you have a candidate sequence composed of I, I, am, I, I. This is what your model outputs at this step.

Candidate	I	I	am	I	I
Reference 1	Younes	said	I	am	hungry
Reference 2	He	said	I	am	hungry

Then you have a reference sequence one, which contains the words Younes, I am hungry. The second reference sequence contains the words, he said, I am hungry.

BLEU Score

Candidate	I	I	am	I	I
Reference 1	Younes	said	I	am	hungry
Reference 2	He	said	I	am	hungry

To get the BLEU score count how many words and the candidates also appear in the references. The candidate you can see I, I, I, I appeared four times. Then am also appeared once. Each word, I and am, appears once in Reference 1, and Reference 2.

BLEU Score

Candidate	I	I	am	I	I
Reference 1	Younes	said	I	am	hungry
Reference 2	He	said	I	am	hungry

How many words in the candidate column appear in the reference translations?

I appears once in each reference. You're going to write M underscore W or max words is equal to one, and clip the count as one.

Then sum over the unique uni-gram counts in the candidates. Meaning you sum over the unique counts in these candidates and you divide by the total number of words in the candidates.

BLEU Score

Candidate	I	I	am	I	I
Reference 1	Younes	said	I	am	hungry
Reference 2	He	said	I	am	hungry

"I" appears at most once in both, so clip to one: $m_w = 1$

$$\frac{\text{(Sum over unique n-gram counts in the candidate)}}{\text{(total \# of words in candidate)}}$$

Well, for the first one you have I and am, so two. Then you divide by the total number of words in the candidates, which is five. Two out of five. This is your BLEU score.

BLEU Score

	2			divide by total (5) = 2/5	
Candidate	$\boxed{\text{I}}$	I	$\boxed{\text{am}}$	I	I
Reference 1	Younes	said	I	am	hungry
Reference 2	He	said	I	am	hungry

"I" appears at most once in both, so clip to one: $m_w = 1$

$$\frac{\text{(Sum over unique n-gram counts in the candidate)}}{\text{(total \# of words in candidate)}}$$

BLEU score is great, but...

Consider the following:

- BLEU doesn't consider semantic meaning
- BLEU doesn't consider sentence structure:

"Ate I was hungry because!"

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. BLEU score is the most widely adapted evaluation metric for machine translation. But you should be aware of these drawbacks before you begin using it.

ROUGE

Recall-Oriented Understudy for Gisting Evaluation

Evaluates quality of machine text

Measures precision and recall between generated text and human-created text

It stands for Recall Oriented Understudy for Gisting Evaluation, which is a mouthful. But let's you know right off the bat that it's more recall-oriented by default. This means that it's placing more importance on how much of the human created reference appears in the machine translation.

ROUGE was originally developed to evaluate the quality of machine summarized texts, but is useful for evaluating machine translation as well. It works by comparing the machine texts or system texts against the reference texts, which is often created by a human.

The ROUGE score calculates precision, and recall for a machine texts by counting the n-gram overlap between the machine texts and a reference texts. Recall that's an n-gram, is a list of words that appear next to each other in a sentence where the order matters. If you have the word, "I baked a pie," a uni-gram can be the word baked, and then bi-gram can be the two words a pie. Next, I'll show you an example of how this works with uni-grams.

ROUGE evaluation

The ROUGE family of metrics focuses on the n-gram overlap between system translated texts and the reference. By system translated text, I'm referring to a model that's being trained to do the prediction. By reference, I'm referring to the ideal correct sentence that I want the model to predict. I mentioned earlier that ROUGE is primarily recall-oriented by default.

What I meant by recall on a high level, is that if you look at all of the words in the reference, which is the cats had orange fur. How many of the reference words gets predicted by the model?

The second part of the equation is precision, which you can think of as answering this question. Of all the words that the model predicted, how many of them are words that we want the model to predict.

Model	The	cat	had	striped	orange	fur
Reference	The	cat	had	orange	fur	

Recall = How much of the reference text is the system text capturing?

Precision = How much of the model text was relevant?

Recall in ROUGE

To calculate the recall for your model translated text. For each word in the true reference sentence, "The cats had orange fur," counts how many of them are also predicted by the model? The, appears in the model prediction. Cats, appears in the prediction as well, had, appears, orange also appears, fur, also appears. In this case, all five of the reference words are also predicted by the model.

For the example system texts, the cat had many orange fur and the reference texts, the cats had orange fur. For the example system texts, the cats had many orange fur and the reference texts, the cats had orange fur. You can see that there are a total of five overlapping uni-grams and five total words in the reference. This would give you a recall of one, a high score.

If your model wanted to have a high recall score, it could just guess hundreds of thousands of words, and it would have a good chance of guessing all the words in the true reference sentence. But what does that actually tell you? This is where precision comes in.

Model	The	cat	had	striped	orange	fur
Reference	The	cat	had	orange	fur	

$$\frac{\text{(Sum of overlapping unigrams in model and reference)}}{\text{(total \# of words in reference)}} \qquad \frac{5}{5} \qquad \text{Recall} = 1$$

Precision in ROUGE

To calculate precision, look at all the words that are predicted by your model. In this example, the cats had striped orange fur. How many of these predicted words actually show up in the correct sentence? Which is represented by the reference sentence? Over here, the, appears in the reference, cat, appears in the reference, had, appears in the reference. Striped, was predicted by the model, but does not appear in the reference sentence. Orange, appears in the reference, and so does fur.

Out of the six words predicted by the model, five of them appear in the reference sentence. This means that your model has a precision of five divided by six, or roughly 83 percent of the words were relevant.

Model	The	cat	had	striped	orange	fur
Reference	The	cat	had	orange	fur	

$$\frac{\text{(Sum of overlapping unigrams in model and reference)}}{\text{(total \# of words in model)}} \qquad \frac{5}{6} \qquad \text{Precision} = 0.83$$

Problems in ROUGE

- Doesn't take themes or concepts into consideration (i.e., a low ROUGE score doesn't necessarily mean the translation is bad)

Model	I	am	a	fruit-filled	pastry
Reference	I	am	a	jelly	donut

There are a few considerations to be aware of when using a ROUGE score. For one, it focuses on comparing n-gram counts to a yield score, which doesn't allow for meaningful evaluation of topics. What this means is that it can only count word overlap as a measure of similarity and misses any broader contexts that words are describing.

For example, if two sentences being compared were, I am a fruit-field pastry, and I'm a jelly donut. ROUGE would have no way of understanding that the two sentences actually mean the same thing. Because of this limitation, it's can take a similar or synonymous concepts into consideration when the score is computed.

A low ROUGE score may not reflect that a model translated text actually captured all the same relevant content as the reference texts just because it's had a large difference in n-gram overlap.

But ROUGE scores are still very useful for evaluation of machine translations and summaries. These are just a couple of caveats to keep in mind as you start taking your own ROUGE scores.

Summary

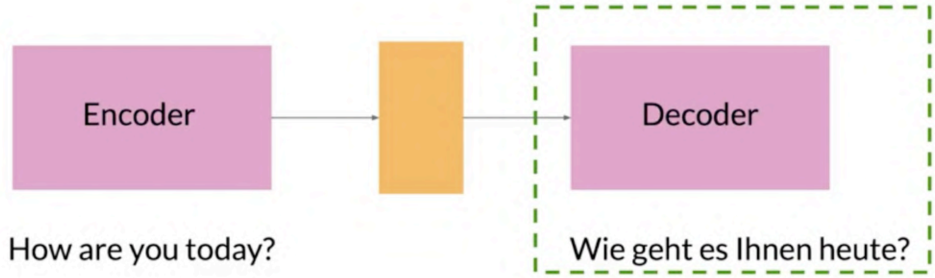
- BLEU score compares "candidate" against "references" using an n-gram average
- BLEU doesn't consider meaning or structure
- ROUGE measures machine-generated text against an "ideal" reference

Outline

- Random sampling
- Temperature in sampling
- Greedy decoding
- Beam search
- Minimum Bayes' risk (MBR)

Seq2Seq model

First, here's a reminder of where your model is in the process when sampling and decoding comes into play. After all the necessary calculations have been peformed on the encoder hidden states, and your model is ready to predict the next token, how will you choose to do it? With the most probable token or taking a sample of from a distribution?



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...". You can see how this could be problematic. But, not in all cases. For shorter sequences, it can be fine. But if you have many other words to consider, then knowing what's coming up next might help you better predict the next sequence.

Selects the most probable word at each step

But the best word at each step may not be the best for longer sequences...

Ich habe Hunger.

I am _hungry_.

I am, am, am, am...

Random sampling

Another option is random sampling. What random sampling does is provide probabilities for each word and sample accordingly for the next output. One of the problems with this is that this can be a little bit too random. A solution to this is to assign more weight to the word with higher porbability and less weight to the others.

am	full	hungry	I	the
0.05	0.3	0.15	0.25	0.25

Often a little too random for accurate translation!

Solution: Assign more weight to more probable words, and less weight to less probable words.

Temperature

In sampling, temperature is a parameter allowing for more or less randomness in predictions

Lower temperature setting = More confident, conservative network

Higher temperature setting = More excited, random network (and more mistakes)

Previously you've seen the greedy decoding algorithm which selects one best candidate as an input sequence for each timestamp. The model has already encoded the input sequence and used the previous timestep's translation to calculate how much attention to give each of the input's words. Now it's using the decoder to predict the next translated word. Not choosing just one best candidate might be suitable for the current timestep, but when we construct the full sentence it may be a suboptimal search.

Beamsearch decoding is the more exploratory alternative for decoding that uses a type of restrictived BFS to build a search tree. Instead of offering a single best output like in greedy decoding, beamsearch selects multiple options based on codnitional probability. The search restriction mentioned a moment ago is the beamwidth parameter, B, which limits the number of branching paths based on a number that you choose. Then at each timestep, the beamsearch selects B number of best alternatives with the highest probability as the most likely choice for the timestep. Once you have these B possibilities, you can choose the one with the highest probability. This is a beam search deccoding which doesn't look only at the next output, but instead applies a beamwidth parameter to select several possible options.

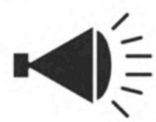
Beam search decoding

A broader, more exploratory decoding alternative

Selects multiple options for the best input based on conditional probability

Number of options depends on a predetermined beam width parameter B

Selects B number of best alternatives at each time step

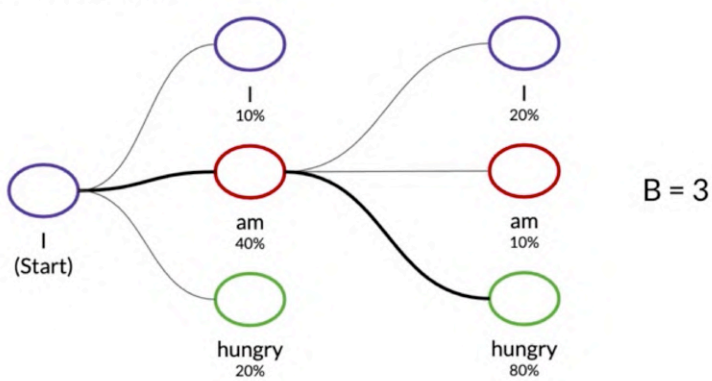


Beam search example

Let's take a look at an example where the beamwidth parameter, B, is 3. The beamwidth parameter is a defining feature of beamsearch which controls the number of beams searching through the sequence of probabilities. Setting the parameter works in an intuitive way. A larger beamwidth will give better model performance but slower decoding speed.

Provided with the first token, i, and the beamwidth parameter, B, of 3, beamsearch assigns conditional probabilites to each of several options for the next word in the sequence. The highest probability is the one that will be chosen for each timestep and the other options will be pruned.

It's determined that "am" is the most likely next token in the seuqnece with a probability of 40%. For the third and final timestep, beamsearch determines "hungry" as the most likely token with probability around 80%. Does this sentence contruction make more sense than any of the other options? This is a very simple example, but you can see for yourself how beamsearch makes a more powerful alterntive to greedy decoding for machine translation of longer sequences



Problems with beam search

Since the model learns a distribution, that tends to carry more weight than single tokens

Can cause translation problems, i.e. in a speech corpus that hasn't been cleaned

However, beamsearch decoding runs into issues where the model learns a distribution that isn't useful or accurate in relatity. It can use single tokens in a problematic way, espeically for uncleaned corporea. Imagine having training data that is not cleaned, for example from a speech corpus. If you have the filler word "um" which appears as a translation in every sentence with 1% probability, that single element can throw off the 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, U.S. of A, ect compared to your German inputs. So in total, you have 11 * 11 at least good translations each with the same probability because they are all equal. So the most probable one is the filler word "um" instead because 1/(11^2) < 0.01%. So that ends up being the most probable outcome which isn't great.

Problems with beam search

"Ich mag die Vereinigten Staaten, weil die Vereinigten Staaten groß sind."

Even with 11 good English translations of "Vereinigten Staaten," but a ~1% probability of the non-word "Uhm" occurring, you might get this as a translation:

"I like the United States, because the Uhm is big. "

Even with 11^2 good translations, the most probable one will still be "Uhm."

Earlier you encountered random sampling as a way to choose a probable token and the issues with that very simple implementation. If you go a little further with that, say by generating 30 examples and comparing them all against one another to see which one performs the best, you will see quite a bit of improvement in your decoding. This is called Minimum Bayes Risk decoding (MBR). Implementing MBR is pretty straightforward, begin by generating several random samples, then compare each sample to all it's mates and assign a similiarity score for each comparison. Rouge is a good one you may recall from earlier. Finally, choose the sample with the highest similiarity which is sometimes referred to as the "golden one".

Minimum Bayes Risk (MBR)

Compares many samples against one another. To implement MBR:

- Generate several random samples
- Compare each sample against all the others and assign a similarity score (such as ROUGE!)
- Select the sample with the highest similarity: the golden one ✨

Example: MBR Sampling

To generate the scores for 4 samples:

1. Calculate similarity score between sample 1 and sample 2
2. Calculate similarity score between sample 1 and sample 3
3. Calculate similarity score between sample 1 and sample 4
4. Average the score of the first 3 steps (Usually a weighted average)
5. Repeat until all samples have overall scores

Summary

- Beam search uses conditional probabilities and the beam width parameter
- MBR (Minimum Bayes Risk)takes several samples and compares them against each other to find the golden one ✨
- Go forth to the coding assignment!

References

This course drew from the following resources:

- Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer (Raffel et al, 2019)

- Reformer: The Efficient Transformer (Kitaev et al, 2020)

- Attention Is All You Need (Vaswani et al, 2017)

- Deep contextualized word representations (Peters et al, 2018)

- The Illustrated Transformer (Alammar, 2018)

- The Illustrated GPT-2 (Visualizing Transformer Language Models) (Alammar, 2019)

- BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Devlin et al, 2018)

- How GPT3 Works - Visualizations and Animations (Alammar, 2020)

That's awesome as well as the discussion of an important hyper parametric call temperature first here's a reminder for your model is in the process when Sam necessary calculations have been performed on the encoder hidden states how will you choose to do it token a sample from a distribution let's discuss a few of the methods available to you and coding is the simplest way to do models predictions as it's Alex the most probable word at every step