Department of Computing

Imperial College of Science, Technology and Medicine

Neural Machine Translation

Attention and stuff

Author: Anton Zhitomirsky

Contents

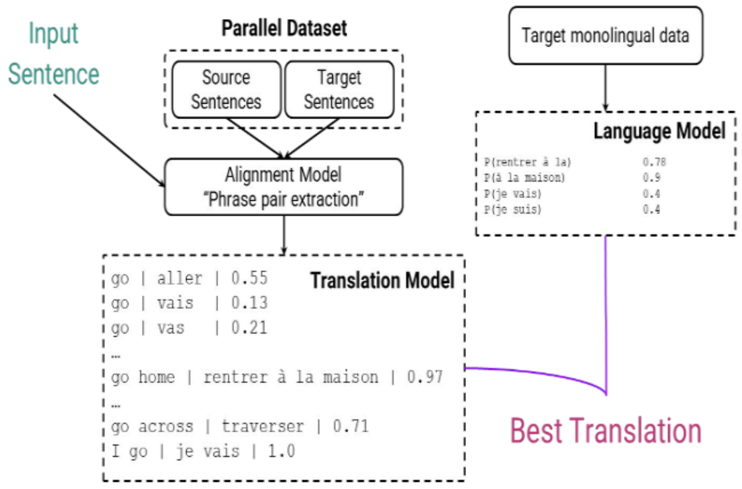
1 Machine Translation

At the highest level, given a document in source language, produce a translation in a target language. This motivates the encoder decoder models that we've seen.

2 Statistical Machine Translation

It is a pipeline of the models: Alignment model, Translation model, and Lanugage model.

Uses parallel corpa as its training set. A prallel corpa is that we have aligned sentences: i.e. an english sentence and its corresponding french sentence.



```
go | aller | 0.55
go | vais  | 0.13
go | vas   | 0.21
...
go home | rentrer à la maison | 0.97
...
go across | traverser | 0.71
I go | je vais | 1.0
```

```
P(rentrer à la)     0.78
P(à la maison)      0.9
P(je vais)          0.4
P(je suis)          0.4
```

- **Parallel dataset** - paired sentences between two different languages (we want to construct correlations between the phrase that happens in one language and translation into another.)
- **Alignment model** - responsible for extracting the phrase pairs
- **Translation model** - lookup table, what is the porbability of the word 'go' appearing in the contex of this target. statistics over large pairs of parallel texts can help identify parallel phrases
- **Corpus** of target monolongual corpus - this is used to get n-grams from the data (what is the probability of this n-gram given this data?)
- **Language model** - contains the probability of target language phrases

2.1 How to perform translations

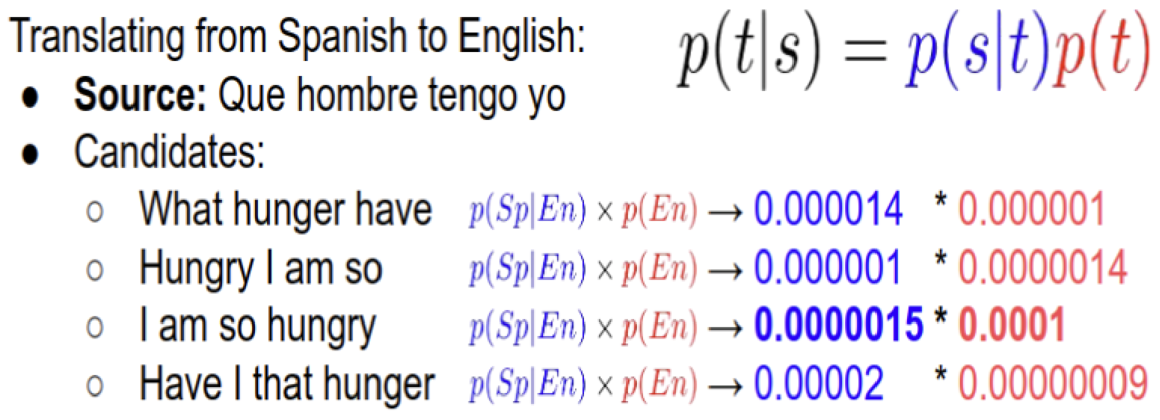
- The objective is to find $p(t|s)$; given a source sentence, s, we want target sentence, t. Language models can be trained on monolingual data[1] that gives us probability of a phrase occurring in that language.
- The translation model is the "objective" flipped. How likely is the source phrase going to occur given a cnadidate translation t. We will have multiple candidate translations.
- These probabilities/likelihoods are built from statistics of our bilingual parallel corpus.
- With this information we can apply Bayes rule to obtain $p(t|s)$.

$$p(t|s) = \frac{p(t)p(s|t)}{p(s)}$$

[1] "It contains texts in one language only" here

- p(s) is just a normalising factor. It doesn't contain t so is irrelevant to obtaining what we're interested in: the actual model
- The actual model is the argmax. Here - pick the phrase that has the highest probability of the product of the 2 terms $p(t)$ (language model) and $p(s|t)$ (translation model)

2.2 Example



Translating from Spanish to English:

$$p(t|s) = p(s|t)p(t)$$

- **Source:** Que hombre tengo yo
- Candidates:
  - What hunger have $p(Sp|En) \times p(En) \rightarrow 0.000014 * 0.000001$
  - Hungry I am so $p(Sp|En) \times p(En) \rightarrow 0.000001 * 0.0000014$
  - I am so hungry $p(Sp|En) \times p(En) \rightarrow$ **0.0000015 * 0.0001**
  - Have I that hunger $p(Sp|En) \times p(En) \rightarrow 0.00002 * 0.00000009$

The first part is to use the translation model, so given these target phrases which have been built by the alignment model and the probabilities which are in the translation modle, what is the most likely candidate, the most likely translation candidate for this given source sentence.

The second part is the lnaguage model itself, which will take each one of these candidates and compute how likley they are to actually appear in target data.

2.3 Downsides

- **Sentence Alignment**

  In parallel corpora single sentences in one language can be translated into several sentences in the other and vice versa. Long sentences may be broken up, short sentences may be merged. There are even some languages that use writing systems without clear indication of a sentence end (for example, Thai).

- **Word Alignment**

  Is about finding out which words align in a source-target sentence pair

  One of the problems presented is function words that have no clear equivalent in the target language. For example, when translating from English to German the sentence "John does not live here," the word "does" doesn't have a clear alignment in the translated sentence "John wohnt hier nicht."

- **Statistical anomalies**

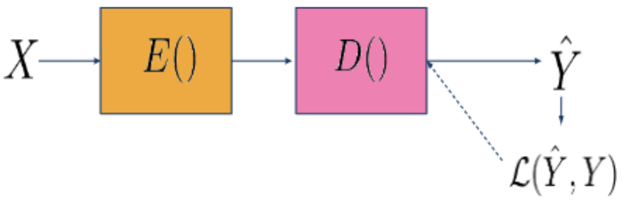
  Real-world training sets may override translations of, say, proper nouns. An example would be that "I took the train to Berlin" gets mis-translated as "I took the train to Paris" due to an abundance of "train to Paris" in the training set.

- **Idioms**

  Only in specific contexts do we want idioms to be translated. For example, using in some bilingual corpus (in the domain of Parliment), "hear" may almost invariably be translated to "Bravo!" since in Parliament "Hear, Hear!" becomes "Bravo!"

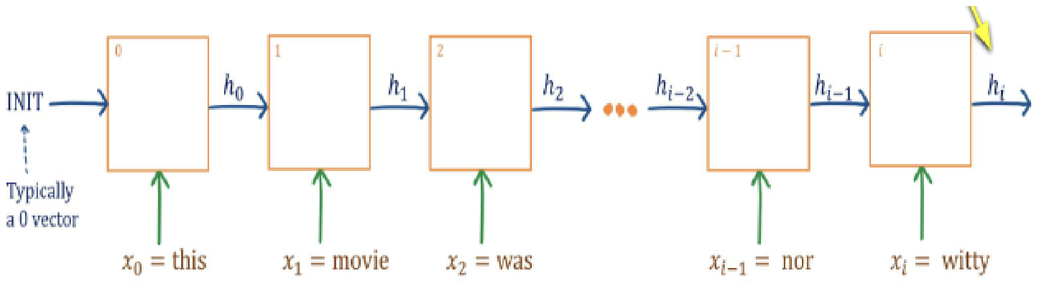
- **Out of vocabulary words**

3 Neural Machine Translation

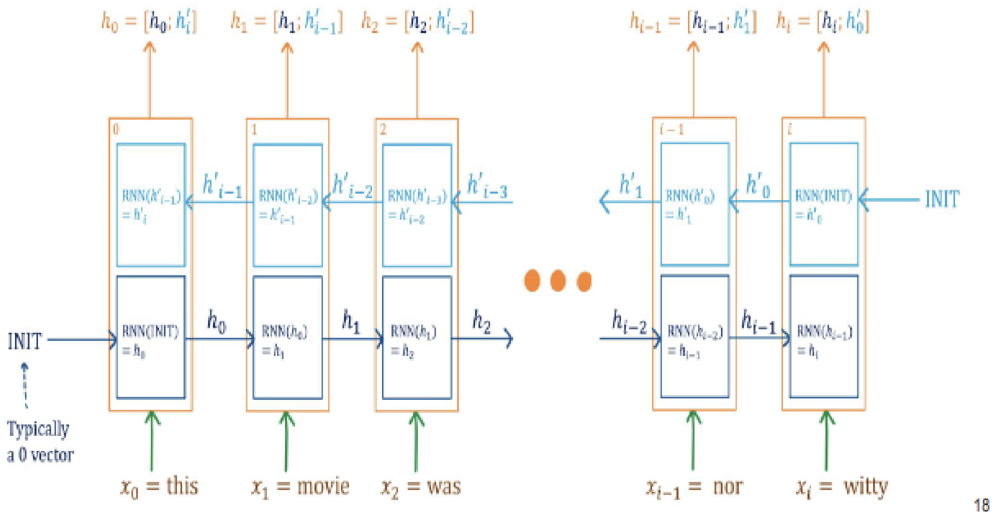


- Encoder function $E()$ - Represents the source as a latent encoding we cannot interpret
- Decoder function $D()$ - Generates a target from the latent encoding.
- Input sequence (source) $X = \{x_1, \ldots, x_i\}$
- Output sequence (target) $Y = \{y_1, \ldots, y_i\}$
- Loss over out outputs; our predicted output given our real output.

3.1 RNNs



(a) vanilla RNN. The hidden state represents the whole sequence

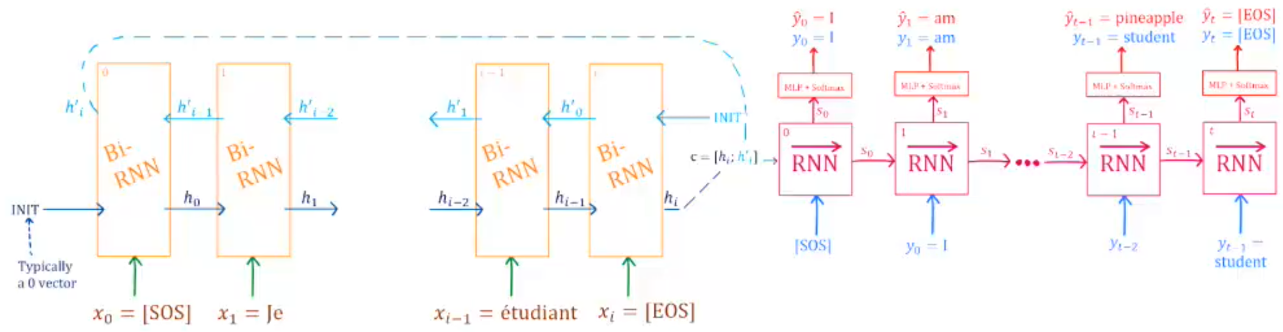


(b) bidirectional RNN

Using a BiRNN is appropriate for encoding the source as we're not trying to predict the source. We have access to the whole thing at inference time. Using a BiRNN, and reading information from both sides, allows us to gain a more information dense representation of our sequence

We can get a hint of representation for each word and hit a representation for each word. The inner representation of each word consists of the forward direction concatentated witht he backwards direction for that particular word or particular token concatenated together.

3.2 Naiive implementataion of machine translation



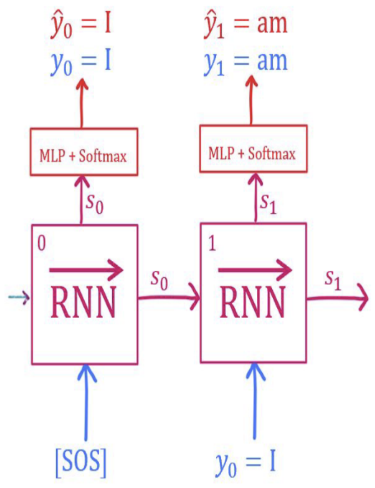
Given an input seqeunce we want to generate the output sequence. We get a c, the context vector, which we initialize our language model (the pink model, the decoder) which we initialize with the hidden state. The context comes from the encoder.

In the bidirectional RNN case, there are two ways that we can compute the context vector.

1. Simply take the lsat representations form the forward and backward direction and calculate them together
2. Average across each one of those representations adn theat can form our context vector (there is a leakage here)

The dimension of C if dimensionality of the model is d, then we either have to set s in R^{2d} or project c down to R^d.

3.3 Teacher Forcing



Decoder is auto-regressive only during inference

During training, we use teacher forcing. We are feeding the ground truth token ot the enxt deconder cell: S_t makes a prediciton, we calculate loss $\mathcal{L}(\hat{y}_t, y_t)$ then feed y_t to decoder cell at $t+1$.

We use teacher forcing with a **ratio**. I.e. we well teacher force ratio% of time. The ratio can be 100% (i.e. full teacher forcing), 50%, or you can even anneal it during training

If at timestamp t the model makes a wrong prediction $\hat{y}_t$ with standard autoregressive modelling, if we fed this back inot decoder at call $t+1$ our model has (potentially) incorrect context. It will use this incorrect context to decode $\hat{y}_{t+1}$; we will likley produce another incorrect word. This leads to an accumulation of errors, which is difficult to optimze over.

Teacher forcing creates a phenomenon called **Exposure Bias**.

Exposure Bias is when the data distributions the model is conditioned on vary between training and inference. In this case, if we set teacher forcing to 100%, then the model is never conditioned on its own predictions. So during inference, when the model uses the previously generated words as context, it is different than what it has seen in training.

3.4 Implementation

1. Minimse negative log likelihood loss:
$$-\sum_{t=1}^{T} \log p(\hat{y}_t | y_{<t}, c) \tag{3.4.1}$$
Here, we assume that we have a teacher forcing ratio of 100%. Also, $y_{<t}$ means all the tokens up to the t'th time step. With teacher ratio of less than 100%, this would be written as a joint of the previosuly decoded tokens (whether they're sampled or the ground truth).
2. c is the context vector output by the encoder
$$c = E(X) \tag{3.4.2}$$
3. $\hat{y}_t$ is the decoder output:
$$\hat{y}_t = \sigma(W_o \cdot D(y_{<t}; c)), \quad W_o \in \mathbb{R}^{d \times o} \tag{3.4.3}$$
Here, o is the output covabulary size. Sigma, is an activation funciton SOFTMAX becuase we're classifying over multiple things.
4. Encoder and decoder are connected... so perform BPTT as normal.

3.4.1 Why is this suboptimal

- We're tying to encode a long sequence in a fixed dimensional representation. For longer sequence lengths, the ability to retain all the source information in this vector diminishes.
- From the decoder side of things, as more words get decoded and the hidden state gets updated to incorporate the previously generated words, information from the context vector may start to diminish.. This is related to:
- vanishing gradients and the lack of ability to 'remember' things from earlier parts of the sequence

3.4.2 Fixes

- c is the entire context for the decoder, wheras the encoder can give us access ot more infromatino that just c (since we're using bidirectional encodings for each word, we know the context from both ends of the sentence) Thus if we could utilise these more fine-grained encodings, we wouldn't suffer from the loss of information that we get from a context vector.

- We may find out which source words are most important to look at for our current timestep of decoding.
- What if, during decoding, the current decoding timestep could look at all the source words, and fin dout which source words are most improtant to its current stage of decoding? Then we could 'pay attention' to those words more than the other words?

4 Attention

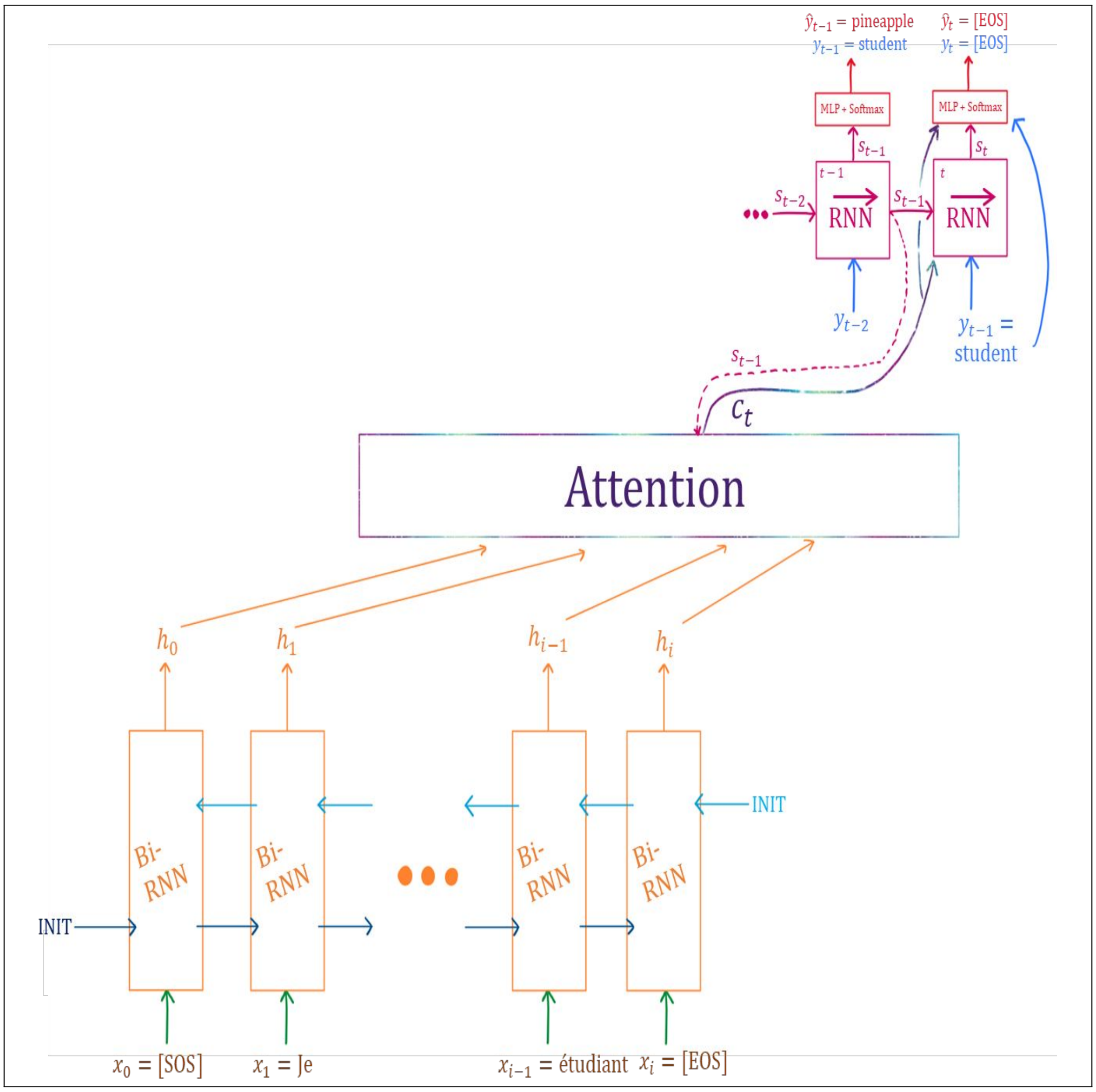
Attention enables weighting individual tokens in the input w.r.t something else. It allows us to look at individual tokens in the input and decide how much weighting a token should have with respect to the current timestep of decoding. The weighting does ont necessarily need ot be in respect to decoding.

4.1 Process

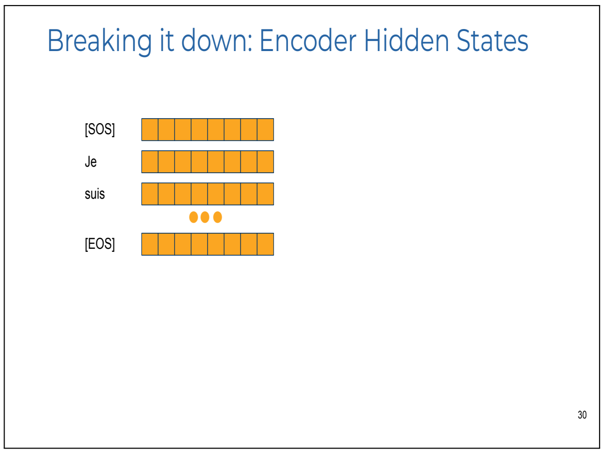
1. For every decoding timestamp we have, we now have a context vector for that decoding time step.
2. We have i hidden states, which are all the encoder's hidden states. We loop over them and multiply it by some α value which is a scalar between 0 and 1.

$$c_t = \sum_{i=1}^{I} \alpha_i h_i \tag{4.1.1}$$

3. We try to decode the y_t'th word and we have access all bi-directional states at every decoding timestep.
4. Before we decode y_t we're going to feed all our encoder hidden states AND the decoder hidden state s_{t-1} to our attention module.
5. The module will output a context vector, c_t.
6. c_t is a dynamic and contextualised representation. It uses the decoder hidden state information to try and figure out which source words are most important when for decododing y_t.
7. We send c_t to the t'th RNN step, alongside the previously generated token y_{t-1}.
8. One final change that the methodology introduced (not strictly related to attention itself), is that the output projection layer now laso takes in c_t and the word embeddings for y_{t-1} alongside s_t to predict the t'th word.
9. Within c_t the $\alpha_i h_i$ multiplication is a key point of attention. It applies a "mask" to each of our hidden states. Low energy values tend to 0 which measn that we do not need that words infromatino to decode the next word
10. Here, the alignment funciton is essentially a 2 layered MLP. The first layer combines the decoder hidden state with all our encoder hidden states, the second layer (v) uses this contextulaised representaiton to predict an energy score: i.e. unnormalised "importance" of each of teh source words.

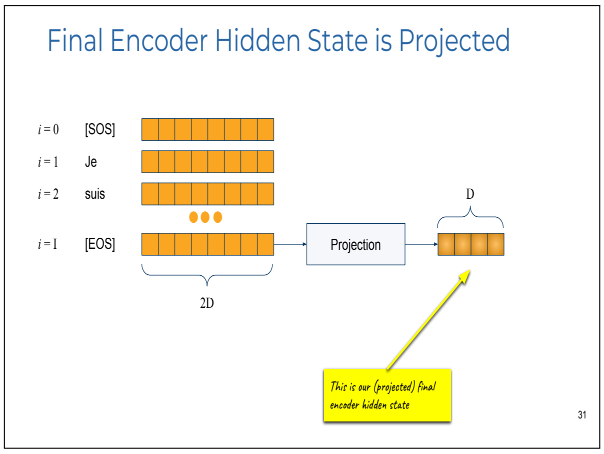


4.1.1 Example



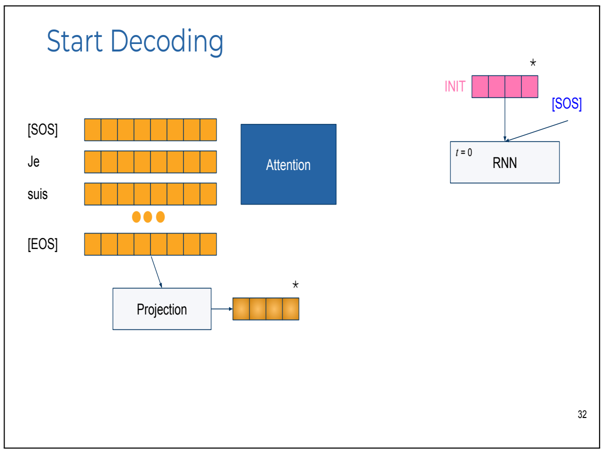
Breaking it down: Encoder Hidden States

(c) These are our bidirectionally encoded encoder hidden states where for each word we have some d dimensional representation of the word

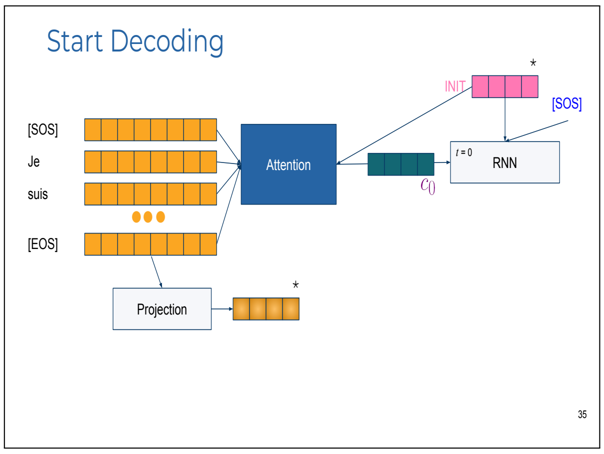


Final Encoder Hidden State is Projected

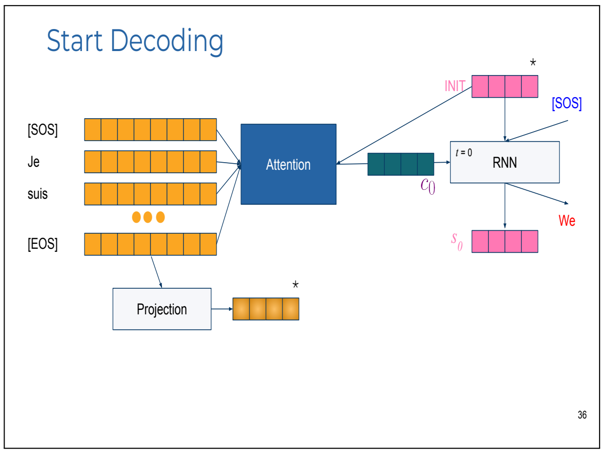
(d) We have I words, and each word is encoded in $2D$ dimensions. Where D is the model dimensionality. It is 2D because we have forward and backwards representations. We project the last hidden state to obtain a D dimensional vector



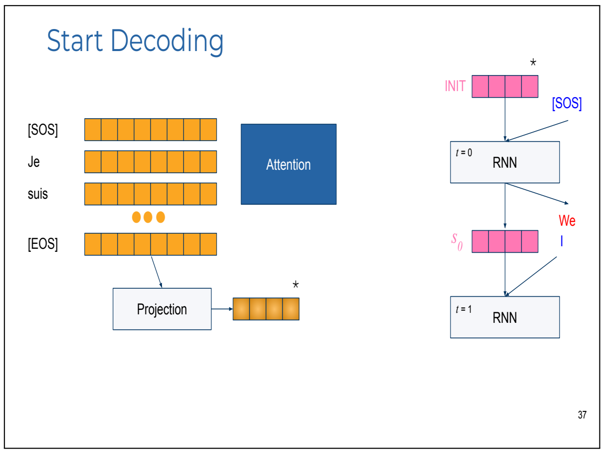
(e) To feed C into our decoder, we double the dimensionality that is in our encoder o or have a projection layer to reduce the dimensionality into d dimensions. Then the decoding process can begin. The decoder RNN has inputs INIT vector (the projected version of the output of the encoder), and the SOS token (start of sequence).



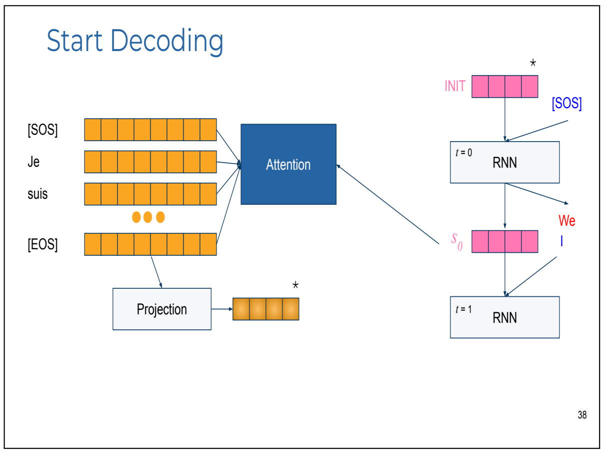
(f) Before we generate the next prediced word, we run the attention mechanis. The attention mechanism takes ALL the encoder hidden states as input, and teh decoder hidden state being inputted to the RNN to decode the current word. The attention is performed over these inputs, and the module outputs a context vector specific to this decoding time step.



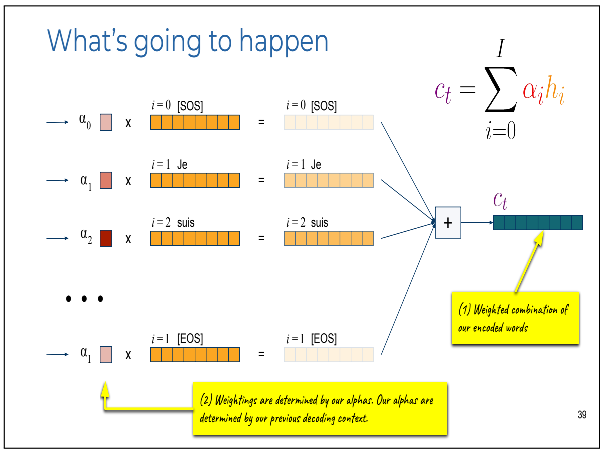
(g) It gives us 2 outputs: our decoder hidden state vector (s_0), and the prediction first word



(h) We then start decoding the next word, if using teacher forcing, we feed the ground truth word as input



(i) Again, we run the attention process. We take in ALL encoder hidden states as input, and the previously outputted decoder hidden state


What's going to happen
$c_t = \sum_{i=0}^{I} \alpha_i h_i$
α_0 x $i=0$ [SOS] = $i=0$ [SOS]
α_1 x $i=1$ Je = $i=1$ Je
α_2 x $i=2$ suis = $i=2$ suis
• • •
α_I x $i=I$ [EOS] = $i=I$ [EOS]
+
c_t
(1) Weighted combination of our encoded words
(2) Weightings are determined by our alphas. Our alphas are determined by our previous decoding context.
39

(j) Attention is going to give us α values for EACH encoded word. These will be between 0 and 1. We're going to scale each encoded word by α and sum them all together to obtain a context vector.

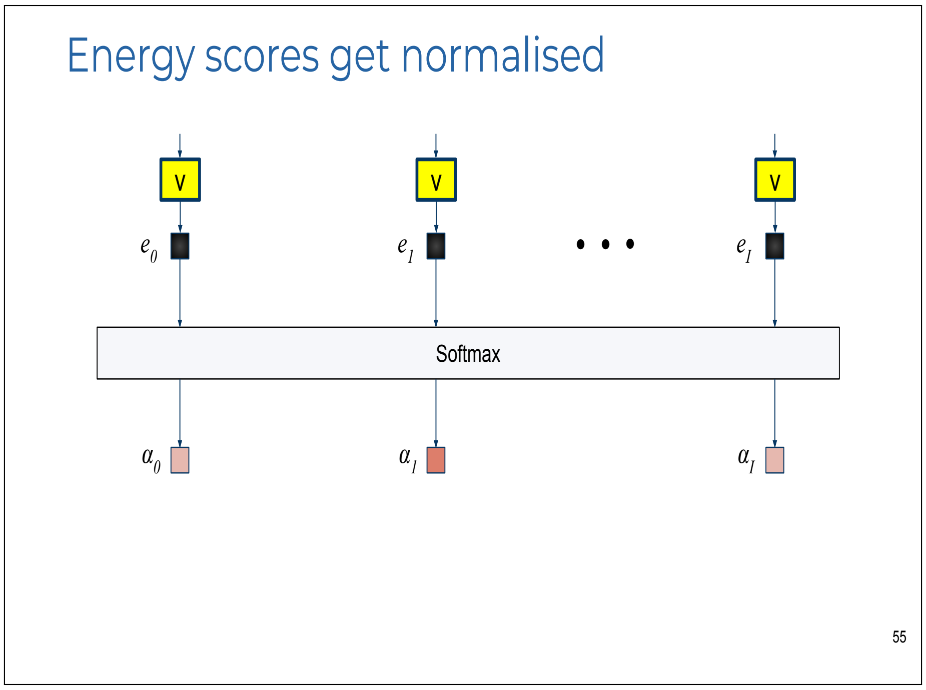
4.2 How to obtain energy scores?

For every source word, i, we're going to obtain a value:

$$\underbrace{e_i}_{1} = \underbrace{a}_{2}(\underbrace{s_{t-1}}_{3}, \underbrace{h_i}_{4}) = \underbrace{v^T}_{5} \overbrace{\tanh}^{9}(\underbrace{W}_{6} s_{t-1} \overbrace{+}^{8} \underbrace{U}_{7} h_i) \tag{4.2.1}$$

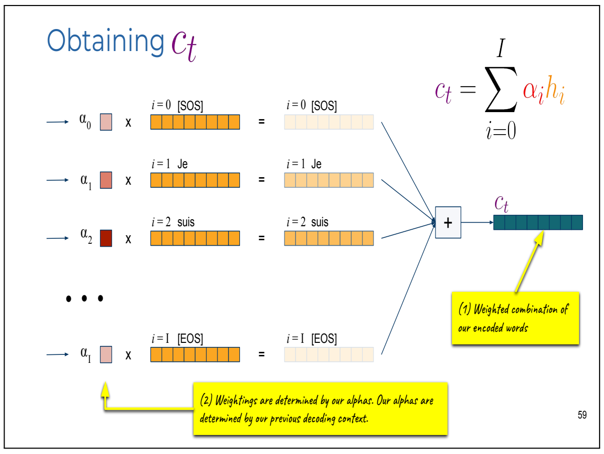
1. $e_i \in \mathbb{R}^1$. The unnormalized energy source for each source word i. "How important is this word to the current decoding step?"
2. A learnt neural network
3. $s_{t-1} \in \mathbb{R}^{D\times 1}$ Previous decoder hidden state
4. $h_i \in \mathbb{R}^{2D\times 1}$ Encoder hidden state for the i'th word.
5. $v^T \in \mathbb{R}^{D\times 1} \wedge v \in \mathbb{R}^{1\times D}$. v is responsible for creating the energy score.
6. $W \in \mathbb{R}^{D\times D}$. similarly to U, we project s_{t-1} via W which also results in a D-dimensional vector
7. $U \in \mathbb{R}^{D\times 2D}$. We project/transform h_i to a D-dimensional vector via a learned weights matrix U.
8. The plus is important here because it mixes the infromation from the decoder and the hidden states together
9. Here, $\tanh$ is just a non-linearity we're applying to it. We could have used another activation function if we wanted to.

The energy scores then get normalised.

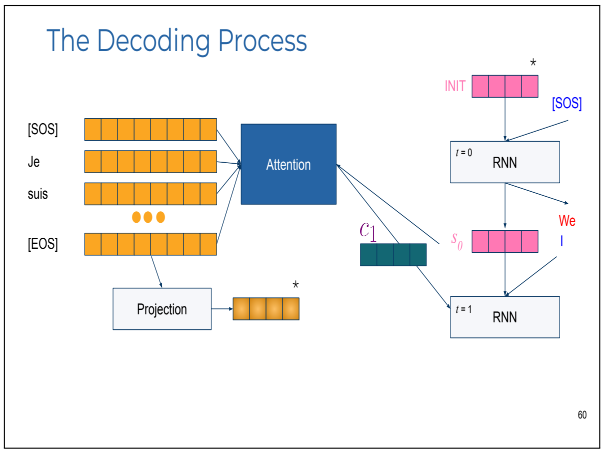


We now have all our energy scores, but they are currently unnormalised. We normalize them by sending all the enrgy scores to a softmax function. This gives us α values between 0 and 1. Therefore, they all sum to 1.

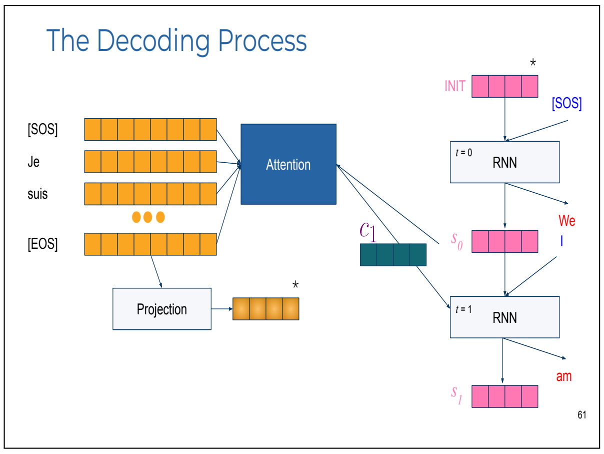
4.2.1 Example continued


Obtaining c_t
$c_t = \sum_{i=0}^{I} \alpha_i h_i$
α_0 x $i=0$ [SOS] = $i=0$ [SOS]
α_1 x $i=1$ Je = $i=1$ Je
α_2 x $i=2$ suis = $i=2$ suis
α_I x $i=I$ [EOS] = $i=I$ [EOS]
+
c_t
(1) Weighted combination of our encoded words
(2) Weightings are determined by our alphas. Our alphas are determined by our previous decoding context.
59

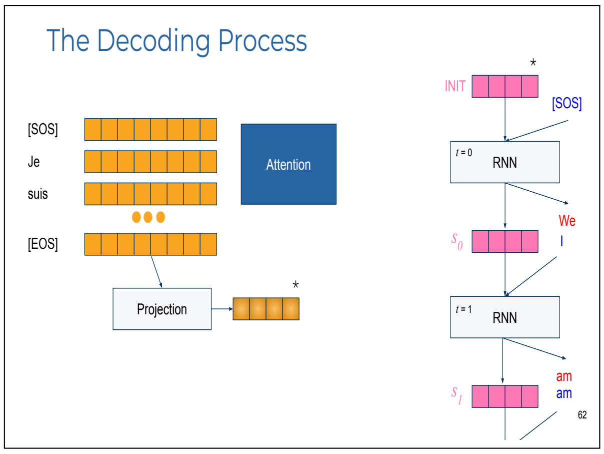
(k) Add all together to obtain c_t - our context vector



(l) Continuing from where we were in our diagram, we then output the context vector for the $t = 1$ word (c1). This forms our 3rd input to the current RNN step (alongside s_0 and an input word)

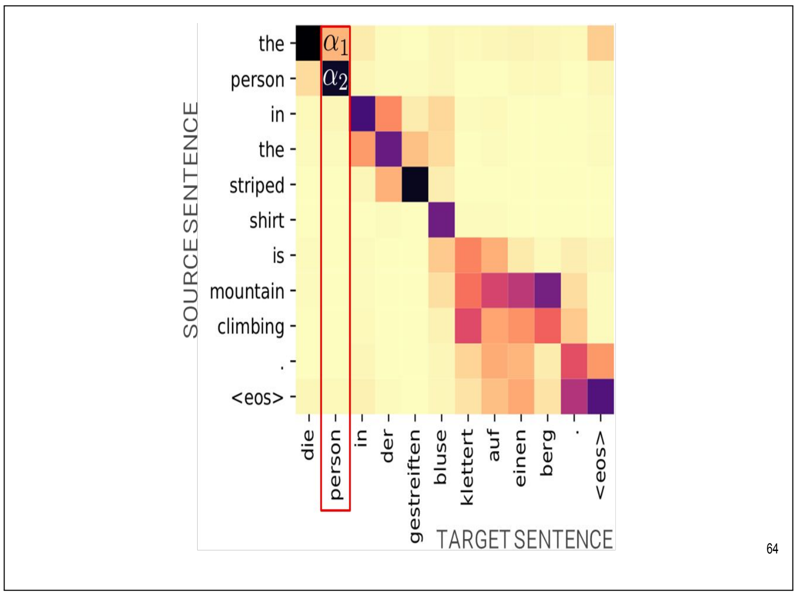


(m) The RNN uses these 3 inputs to output decoder hidden state s_t and a prediction (am)



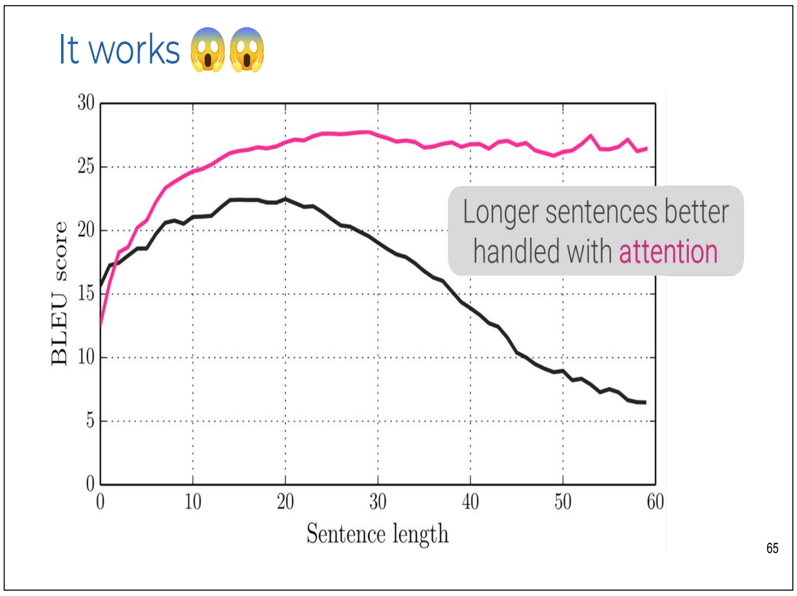
(n) Repeat the process until we hit our max length

4.3 Visualising the attention



We can visualise our attention values to see which source words were looked at most when decoding a target word - I.e. when we're decoding the word person, in German, we're looking strongly at "person" in English, and also "the".

4.4 Conclusion



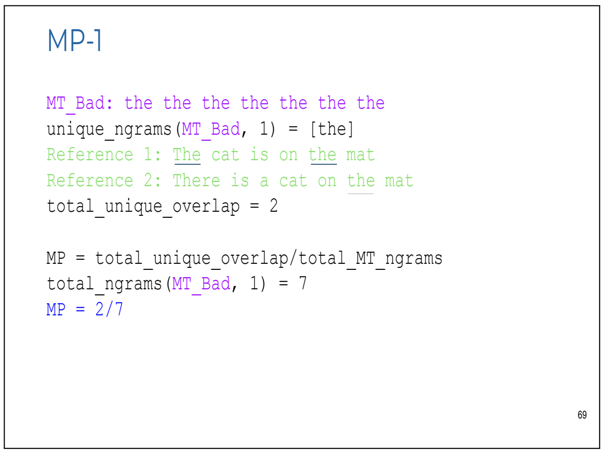
We see that longer sentences are better handled with attention. We have dynamic represetntaion for each time tsep that we're decoding.

5 Evlauation metrics of MT/NLG systems

- Human evaluation (best but expensive)
- Automatic evaluation: proxies to gaugae how effective the machine generated output is
  - Fast/proxy mesaures for evaluation
  - Often based on count statistics of n-grams

5.1 BLEU

BLEU reports a modified precision metric for each level of n-gram. Credit is assigned as the maximum aamount of tiems eqhc unique n-gram appears over all the references. Multiple references are recommended (not a strict requirement).



MP-1

```
MT_Bad: the the the the the the the
unique_ngrams(MT_Bad, 1) = [the]
Reference 1: The cat is on the mat
Reference 2: There is a cat on the mat
total_unique_overlap = 2

MP = total_unique_overlap/total_MT_ngrams
total_ngrams(MT_Bad, 1) = 7
MP = 2/7
```

69

First calculate the number of unique n-grams in our Hypothesis. ModifiedPrecision-1 means uni-grams. Then we look at the n-gram overlap bewteen the unique ngrams in our Hypothesis and all n-grams in our References. We work out the number of unique overlaps beteween the unique ngrams and all the Refs (there are two instances of the in the first reference, and 1 in the second). Then we obtain our modified-precision scores: the total unique overlap divided by the number of n-grams in our Hyp.

5.1.1 Modified Precision Score

$$\text{Modified Precision score } p_n = \frac{\text{Total Unique Overlap}_n}{\text{Total n-grams}} \tag{5.1.1}$$

Here, we have an example sentence in french "Le chat est sur le tapis" with references (shown on the example) and two outputs, MT_Bad and MT_Ok.

```
MT_Ok: the cat the cat on the mat
unique_ngrams(MT_Ok, 1) = ?
Reference 1: The cat is on the mat
Reference 2: There is a cat on the mat
total_unique_overlap = ?

MP = total_unique_overlap/total_MT_ngrams
total_ngrams(MT_Ok, 1) = ?
MP = ?
```

(o) MP-1: Question, answer below:

```
MT_Ok: the cat the cat on the mat
unique_ngrams(MT_Ok, 2) = ?
Reference 1: The cat is on the mat
Reference 2: There is a cat on the mat
total_unique_overlap = ?

MP = total_unique_overlap/total_MT_ngrams
total_ngrams(MT_Ok, 2) = ?
MP = ?
```

(p) MP-2: Question, answer below:

```
MT_Ok: the cat the cat on the mat
unique_ngrams(MT_Ok, 1) = [the, cat, on, mat]
Reference 1: The cat is on the mat
Reference 2: There is a cat on the mat
total_unique_overlap = 5

MP = total_unique_overlap/total_MT_ngrams
total_ngrams(MT_Ok, 1) = 7
MP = 5/7
```

(q) "What are the unique n-grams that exist (in the `MT_OK`)? We have 'the', 'cat', 'on', 'mat'. Then we work out the maximum number of times that there is an overlap betwen each one of our unique n-grams and we see: 'the', 'cat', 'on', 'the' (we see 'the' twice in our references) and 'mat'. Then, the total number of machine translated n-grams is 7 so the modified precision socre is 5"

```
MT_Ok: the cat the cat on the mat
unique_ngrams(MT_Ok, 2) = [the cat, cat the, cat on,
on the, the mat]
Reference 1: The cat is on the mat
Reference 2: There is a cat on the mat
total_unique_overlap = 4

MP = total_unique_overlap/total_MT_ngrams
total_ngrams(MT_Ok, 2) = 6
MP = 4/6
```

(r) `total_unique_overlap` measures the **UNIQUE** occurrences, not only the intersection between the two reference sentences. Here, underlined in bold black are the unique occurrences from the unique_ngrams list. We do not include the end of sentence reference here because we are testing the produced output. EOS is somethind done within the model representation. `total_ngrams` is not unique, there are 6 in the `MT_OK`.

5.1.2 BLEU score

Measures precision of n-gram matches (typically up to 4-grams). Scaled by a Brevity Penalty (BP), it is used to mostly encourage the hypothesis to be of a simlar length to the reference.

$$\texttt{BLEU-4} = \text{BP}(\prod_{n}^{4} p_n)^{\frac{1}{4}}, \quad \text{BP} = \min(1, \frac{\text{MT Output Length}}{\text{Reference Length}}) \tag{5.1.2}$$

5.1.3 Downside

A shortcoming of BLEU is that it focuses a lot on the precision between Hyp and Ref, but not the recall.

5.2 Chr-F & TER

5.2.1 Chr-F

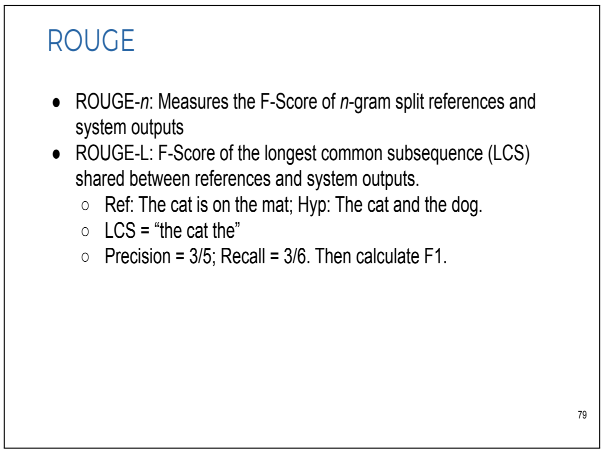
Chr-F is a character n-gram F_β score.

- Balances **character precision**
  - percentage of n-grams in the hypothesis which have a acounterpart in the reference
- and **character recall**
  - percentage of character n-grams in the reference which are also present in the hypothesis.

5.2.2 TER: Translation Error Rate

Minimum # of edits required to change a hypothesis into one of the references. TER is performed at the word level, and the "edits" can be a: Shift, Insertion, Substitution and Deletion. TER balances these edits to build a metric

5.3 ROUGE



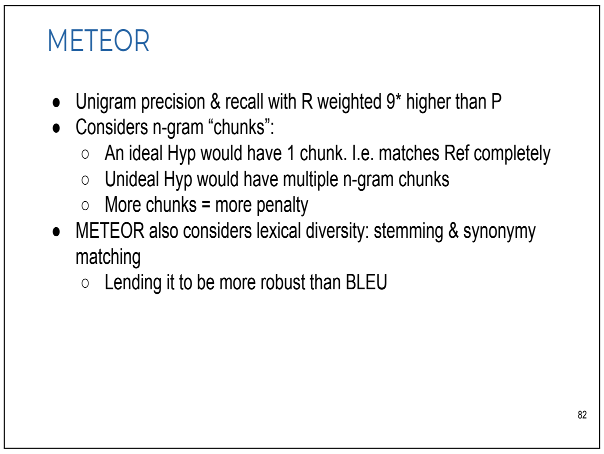
ROUGE

- ROUGE-*n*: Measures the F-Score of *n*-gram split references and system outputs
- ROUGE-L: F-Score of the longest common subsequence (LCS) shared between references and system outputs.
  - Ref: The cat is on the mat; Hyp: The cat and the dog.
  - LCS = "the cat the"
  - Precision = 3/5; Recall = 3/6. Then calculate F1.

79

- ROUGE balances both precision and recall via the F-Score.
- Though originally a translation metric, ROUGE is more common in captioning/summarization literature than translation. Obviously it can be used for translation though.
- ROUGE-n measures the F-score of n-gram split references and system outputs
- ROUGE-L is the F-score of the LCS between the references and system outputs. The subsequence does not have to be consecutive
- For example. . . given this Ref and Hyp:
- Other variants of ROUGE such as ROUGE-S, ROUGE-W which incorporate skip grams and/or weighting

5.4 METEOR



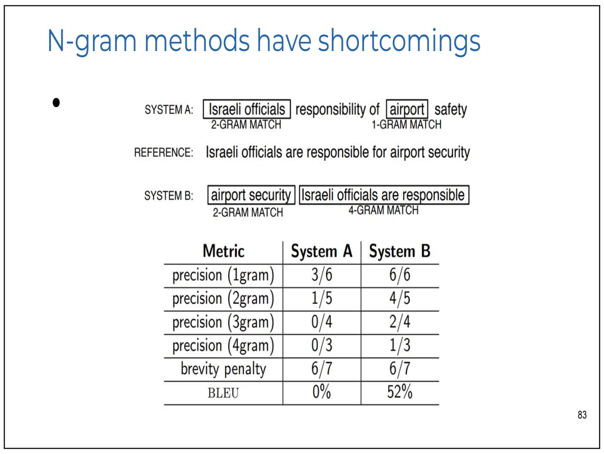
METEOR

- Unigram precision & recall with R weighted 9* higher than P
- Considers n-gram "chunks":
  - An ideal Hyp would have 1 chunk. I.e. matches Ref completely
  - Unideal Hyp would have multiple n-gram chunks
  - More chunks = more penalty
- METEOR also considers lexical diversity: stemming & synonymy matching
  - Lending it to be more robust than BLEU

82

- METEOR is more modern and a better metric than BLEU, though for some reason the NMT community still adopts BLEU a lot more frequently. You will find METEOR in other generation tasks such as summarization and captioning

5.5 N-gram methods have shortcomings



N-gram methods have shortcomings

SYSTEM A: Israeli officials responsibility of airport safety
2-GRAM MATCH 1-GRAM MATCH

REFERENCE: Israeli officials are responsible for airport security

SYSTEM B: airport security Israeli officials are responsible
2-GRAM MATCH 4-GRAM MATCH

Metric	System A	System B
precision (1gram)	3/6	6/6
precision (2gram)	1/5	4/5
precision (3gram)	0/4	2/4
precision (4gram)	0/3	1/3
brevity penalty	6/7	6/7
BLEU	0%	52%

83

- To humans, system A makes more sense, as it catches more meaning behind the meaning of the reference
- This n-gram based method cannot capture the context behind sentences.
- Other metrics might be slightly more robust than this, but fundamentally n-gram methods cannot take into account valid candidates/-machine outputs which may use synonyms instead of a word in the reference.

5.6 BertScore

Not count based. Bert is a trained language model that can give oyu contextual representations of tokens/words. It's not used so much for translation, but still a good metric for most sequence-to-sequence tasks. The biggest drawback is now not the n-gram matching, rather, the scores can vary if evaluated against different BERT models.

BertScore

- Computes pairwise cosine similarity for each token in the candidate with each token in the reference sentence

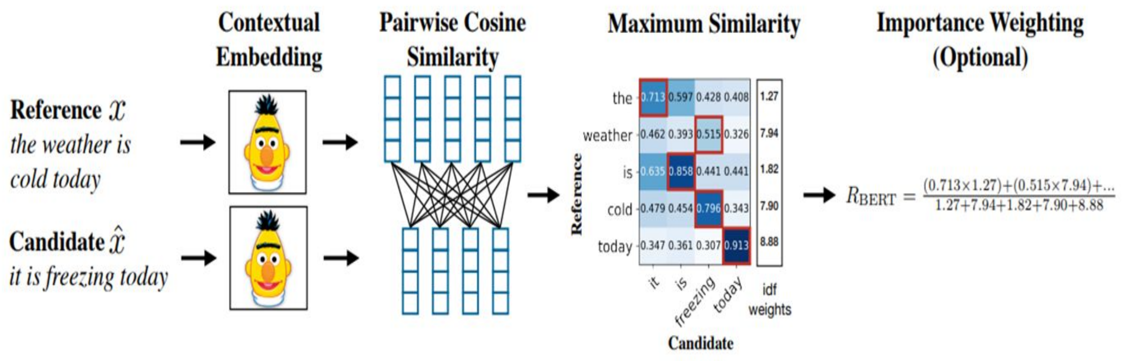


Figure 1: Illustration of the computation of the recall metric R_{BERT}. Given the reference x and candidate $\hat{x}$, we compute BERT embeddings and pairwise cosine similarity. We highlight the greedy matching in red, and include the optional idf importance weighting.

84

Performs a pairwise cosine similarity between each word in the reference and each word in the candidate.

If you have a sequence to sequence task then bert score is the metric you should be using. So if you have summarization, audio transcription, generative question answering then BERT score is a good score to use.

6 Inference

During training, we used teacher forcing. How we don't have access to the ground truth. During inference we can either perform:

6.1 Greedy decoding

Outputs the most likely word at each time step (i.e. an argmax). It is fast, but doesn't look into the future. But as we decode the rest of the sentence, it might be worse than we thought. If we were able to see what the future candidates might be, we might be able to predict a better word for the current time step.

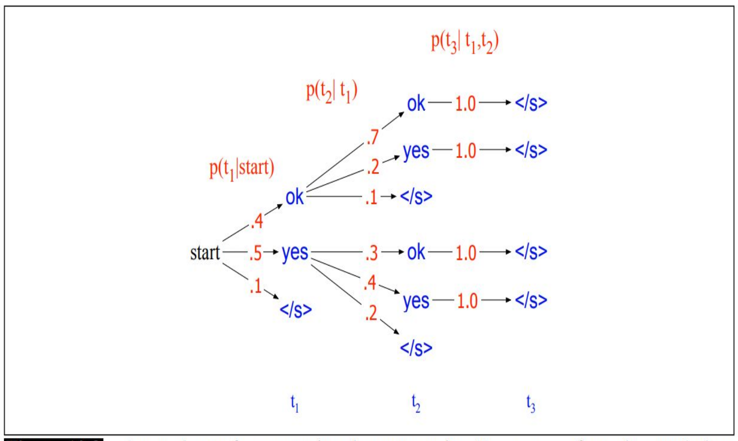
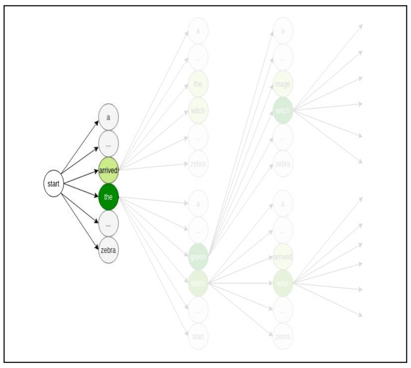


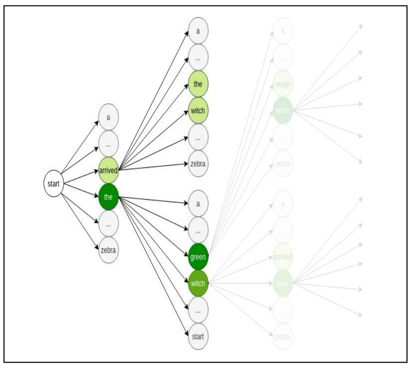
Figure 10.8 A search tree for generating the target string $T = t_1, t_2, \ldots$ from the vocabulary $V = \{\text{yes}, \text{ok}, \text{<s>}\}$, showing the probability of generating each token from that state. Greedy search would choose *yes* at the first time step followed by *yes*, instead of the globally most probable sequence *ok ok*.

6.2 Beam search

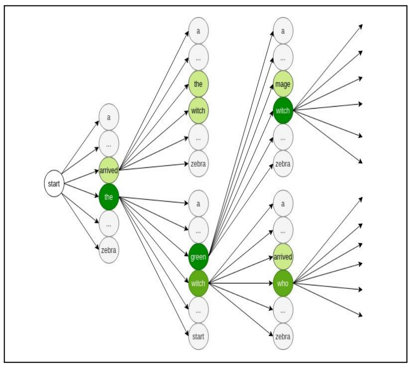
Instead of choosing the best token to generate at each time-step we keep k possible tokens at each step. In practice, k is normally between 5-10. Note that we will end up with k hypothesis at the end of decoding. Decoding finishes when we hit an EOS token. For example, here is $k = 2$.



(s) $t = 0$ 'arrived' and 'the' are the 2 most likley words



(t) $t = 1$ for each word, decode the next k. We end up with the green nodes. Then, prune all but the top-k



(u) $t = 2$ repeat. Now we have the next green, and lets predict 2 times steps into the future and keep track of the 2 tokens that give the highest likelihood for those time steps.

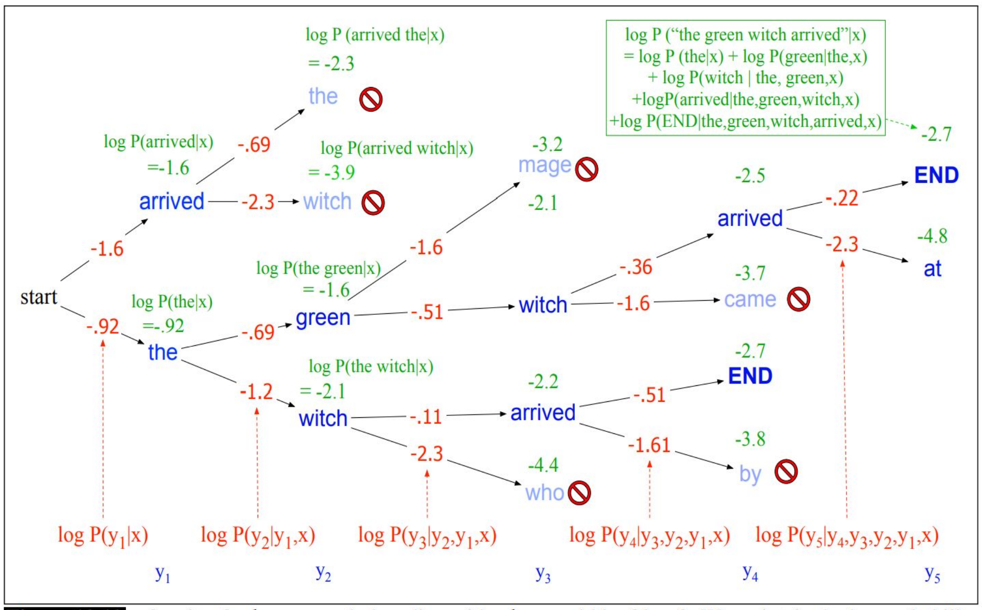


Figure 10.10 Scoring for beam search decoding with a beam width of $k = 2$. We maintain the log probability of each hypothesis in the beam by incrementally adding the logprob of generating each next token. Only the top k paths are extended to the next step.

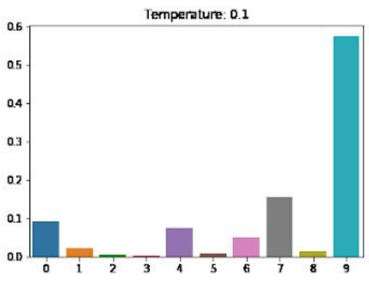
6.3 Temperature sampling

Introduces stochasticity into the decoder step. This is a problem with the two above; they are deterministic.

Temperature sampling

- One "problem" with the above methods: They are deterministic
- Could sample instead of argmax… but softmax can be peaky
- Temperature sampling can help:
  - 1. Divide logits by T. Run softmax.
  - 2. Multinomial sample over softmax probabilities

$$\frac{e^{\frac{y_i}{T}}}{\Sigma_j^N e^{\frac{y_j}{T}}}$$



92

7 Tricks — how to improve performance of models

7.1 Data Augmentation

7.1.1 Backtranslation

Lets translate the source into one language and translate it back. The hope is, that once translated back the semantics is the same but syntactically it may be different.

- Train/use a separate target $\rightarrow$ source model
- Translate target monolingual corpora to source using the model
- Train the actual source $\rightarrow$ target model with the additional data
- Ideally changes the syntax of source while keeping the same semantics. Downsides: adds noise to the source

7.1.2 Synonym replacement

- Use dictionary & syntax trees to find appropriate synonyms
- Use word embeddings & nearest neighbors to find synonyms.

7.1.3 Batching, PAdding and Sequence Length

- Group similar length sentences together in the batch
- train your model on simpler/smaller sequence lengths first.