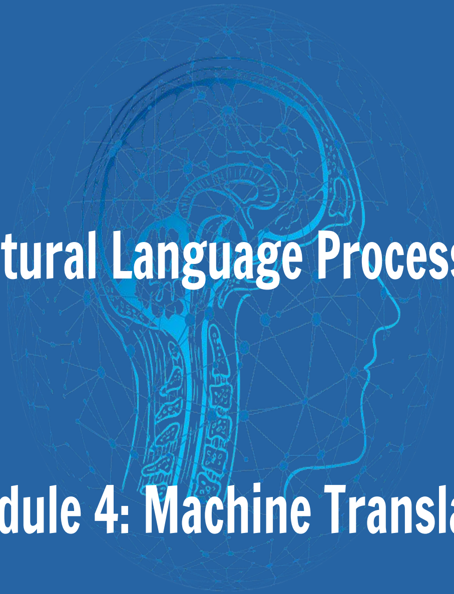
Imperial College
London

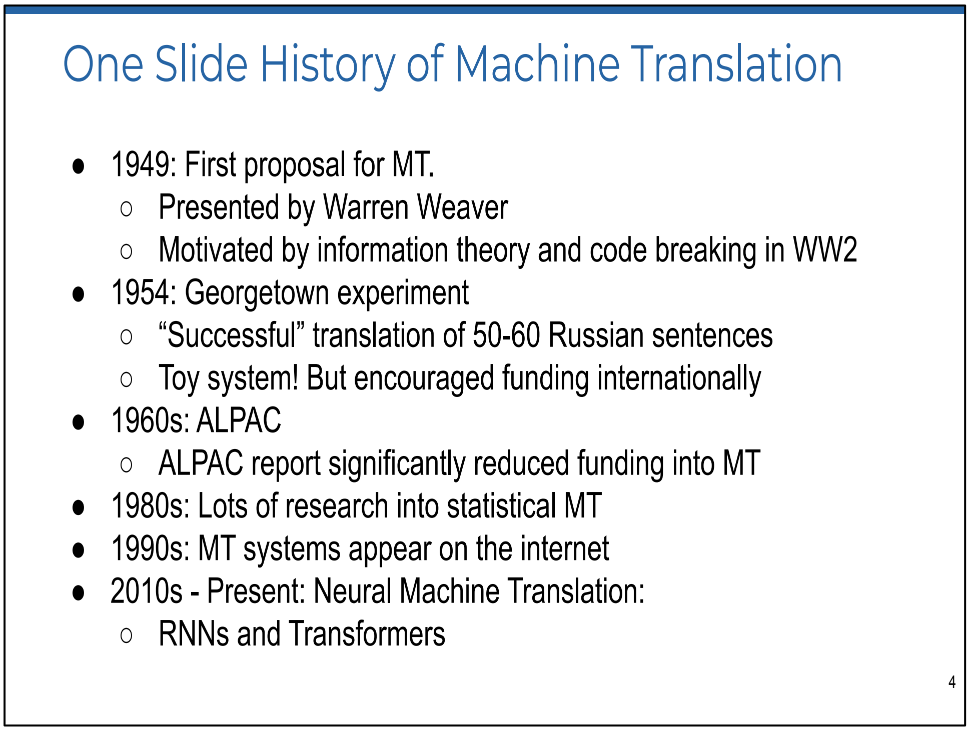


Natural Language Processing

Module 4: Machine Translation

Today...

Contents



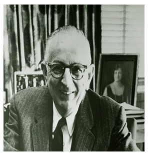
One Slide History of Machine Translation

- 1949: First proposal for MT.
    - Presented by Warren Weaver
    - Motivated by information theory and code breaking in WW2
- 1954: Georgetown experiment
    - “Successful” translation of 50-60 Russian sentences
    - Toy system! But encouraged funding internationally
- 1960s: ALPAC
    - ALPAC report significantly reduced funding into MT
- 1980s: Lots of research into statistical MT
- 1990s: MT systems appear on the internet
- 2010s - Present: Neural Machine Translation:
    - RNNs and Transformers

1949 – in the next slide we’ll take a look at a quote by Warren Weaver which will explain his information theory angle a bit more
1954 – The experiment promised to solve translation within 5-10 years! Well.. let’s just say that didn’t happen… Here we are 70 years later. It attracted a lot of funding (both within the US and internationally) which encouraged research into this field
1960s – Automatic MT was being used for readers to get a *rough* idea of topics being talked about. It assisted human translators in completing the translation of a document
1980s – Research in MT was largely coming out of Japan - motivated by their rising popularity in software and hardware
1990s – MT systems appear on the internet: AltaVista’s BabelFish and Google Translate. Research into SMT continues.

Encoder Decoder

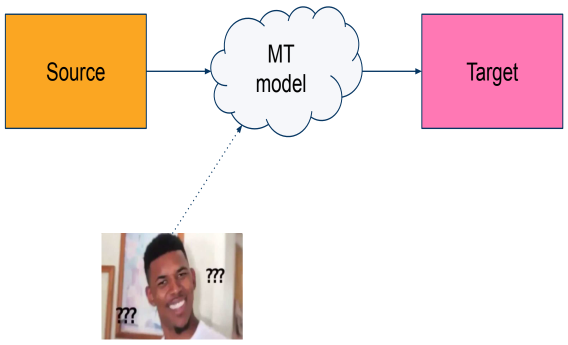
"...one naturally wonders if the problem of translation could conceivably be treated as a problem in cryptography. When I look at an article in Russian, I say: '*This is really written in English, but it has been coded in some strange symbols. I will now proceed to decode*'."

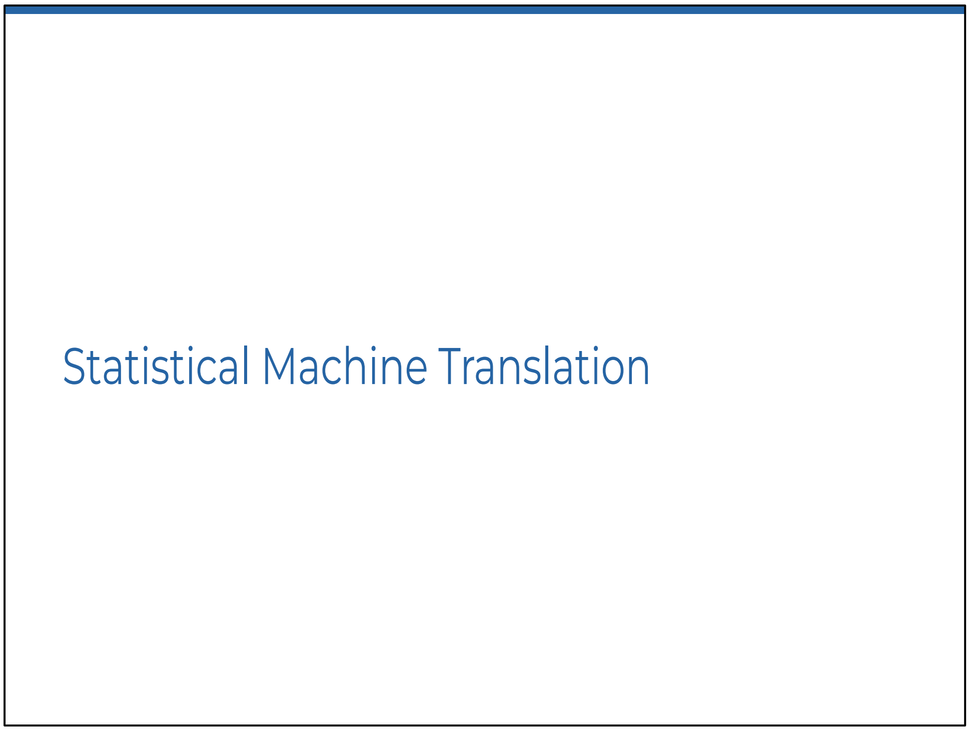

Warren Weaver, 1949
Pioneer of Machine Translation

-

The highest level overview

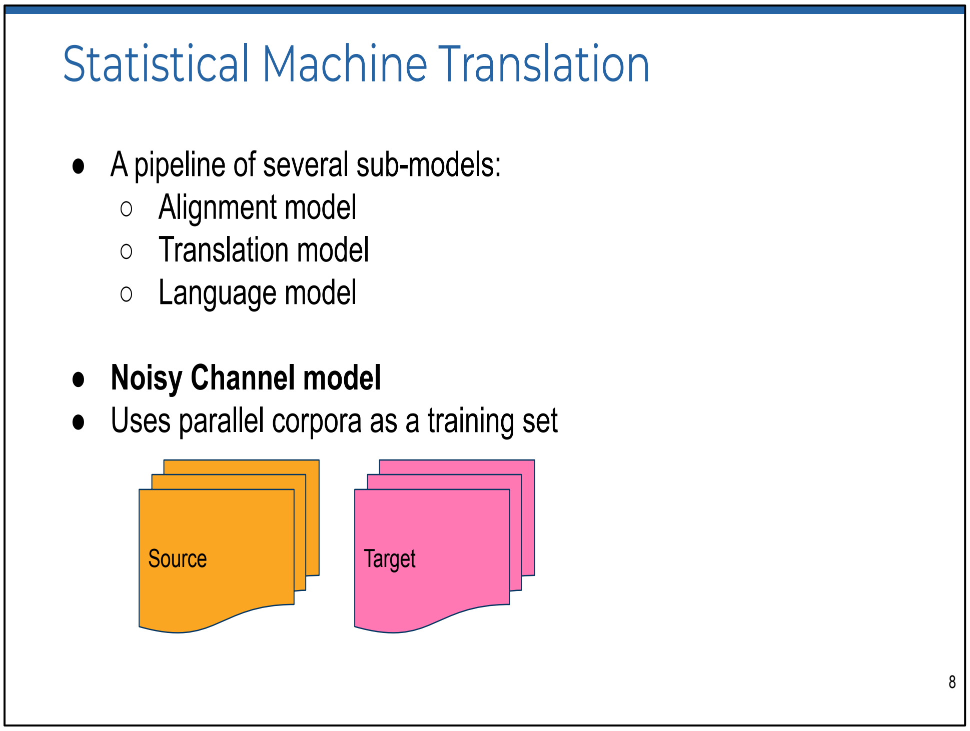
Given a document in **source** language,
produce a translation in **target** language





Statistical Machine Translation

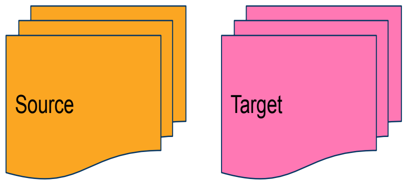
Running ETA: 5



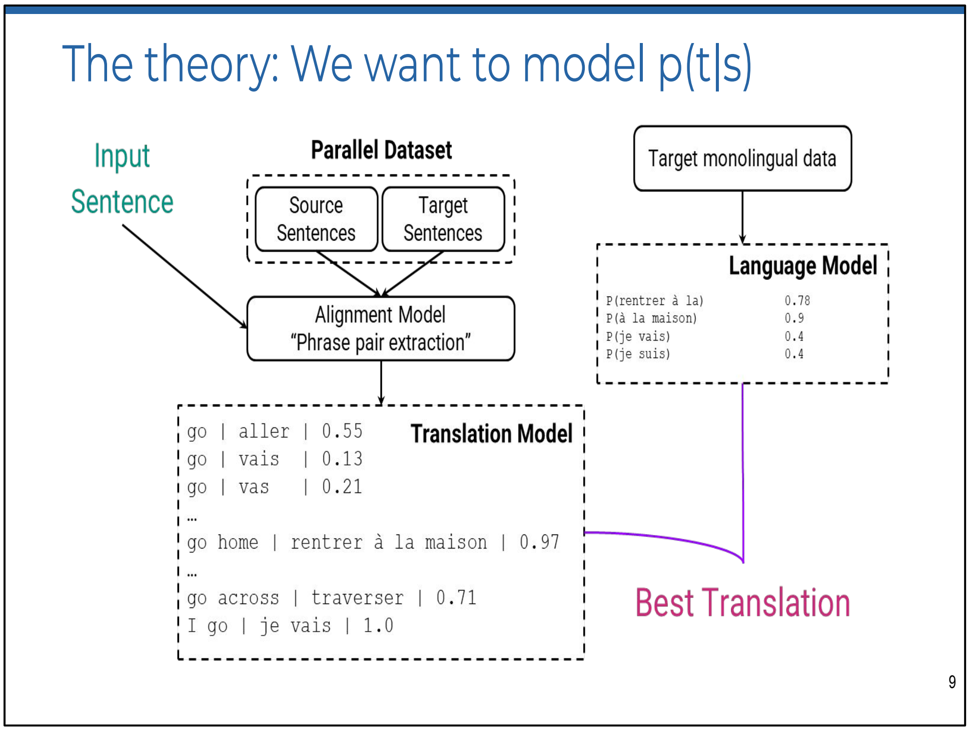
Statistical Machine Translation

- A pipeline of several sub-models:
    - Alignment model
    - Translation model
    - Language model

- **Noisy Channel model**
- Uses parallel corpora as a training set

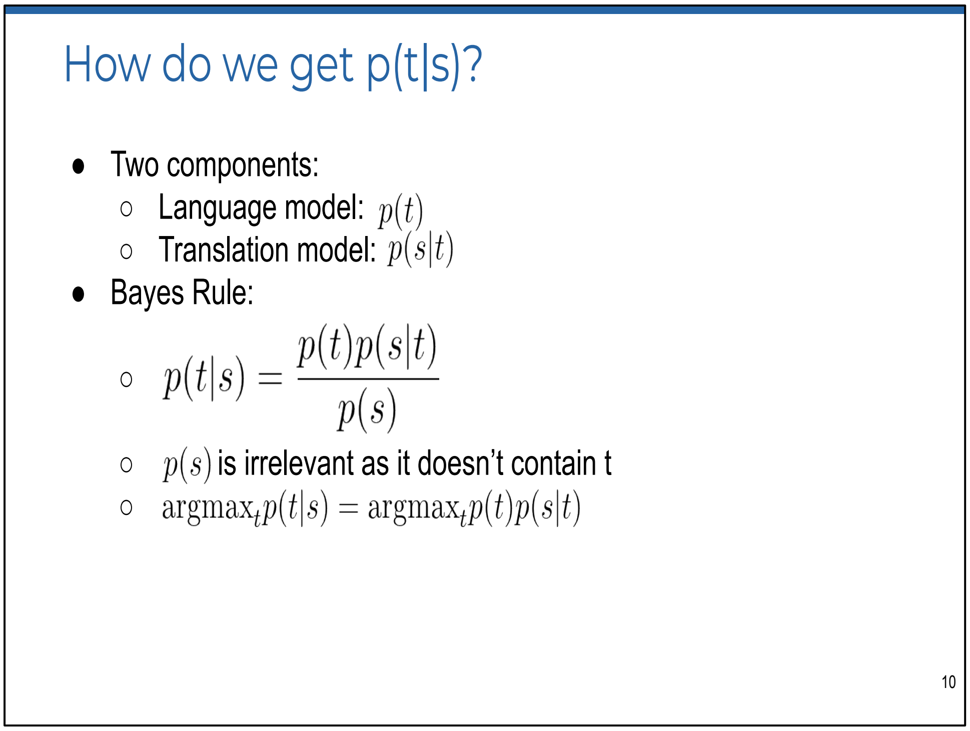


- Was the previous prominent approach before neural models. Is a well researched field with multiple different ways to achieve an output. Here we'll only be discussing the notion at a high level.
- Parallel corpus means we have aligned sentences: i.e. an english sentence and its corresponding french sentence



The theory: We want to model p(t|s)

- Given source sentence, s, we want target sentence, t:
- I.e. model p(t|s)
-
- Translation model contains phrases alongside their translation:
    - Alignment model is responsible for extracting the phrase pairs
    - Translation model is essentially a lookup table. We won't dive into it, but you can imagine that statistics over large pairs of parallel texts can help identify parallel phrases
- Language model contains the probability of target language phrases

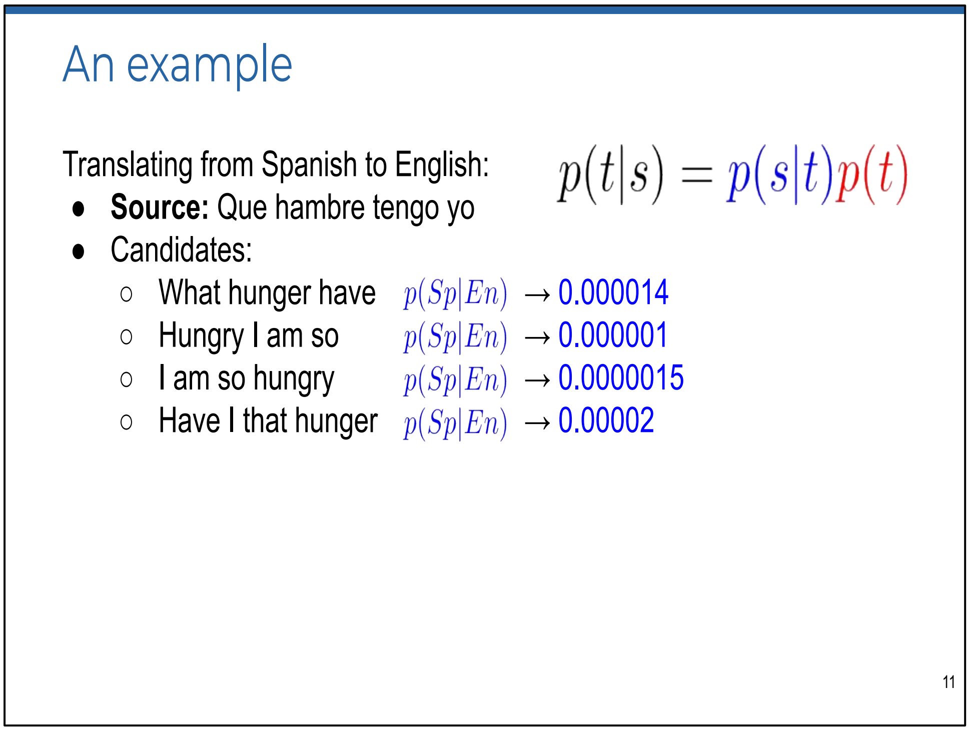


How do we get p(t|s)?

- Two components:
  - Language model: $p(t)$
  - Translation model: $p(s|t)$
- Bayes Rule:
  - $p(t|s) = \dfrac{p(t)p(s|t)}{p(s)}$
  - $p(s)$ is irrelevant as it doesn't contain t
  - $\text{argmax}_t p(t|s) = \text{argmax}_t p(t)p(s|t)$

10

- Language model can be trained on monolingual data and gives us probability of a phrase occurring in that language.
- Translation model is the "objective" flipped. Essentially we're saying: How likely is the source phrase going to occur given a candidate translation t. We will have multiple candidate translations (example on next slide)
  - These probabilities/likelihoods are built from statistics of our bilingual parallel corpus.
  - We're not going to dive into how this is created
- With this information, we can apply Bayes rule to obtain p(t|s)
  - 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 argmaxs. 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).
  -



An example

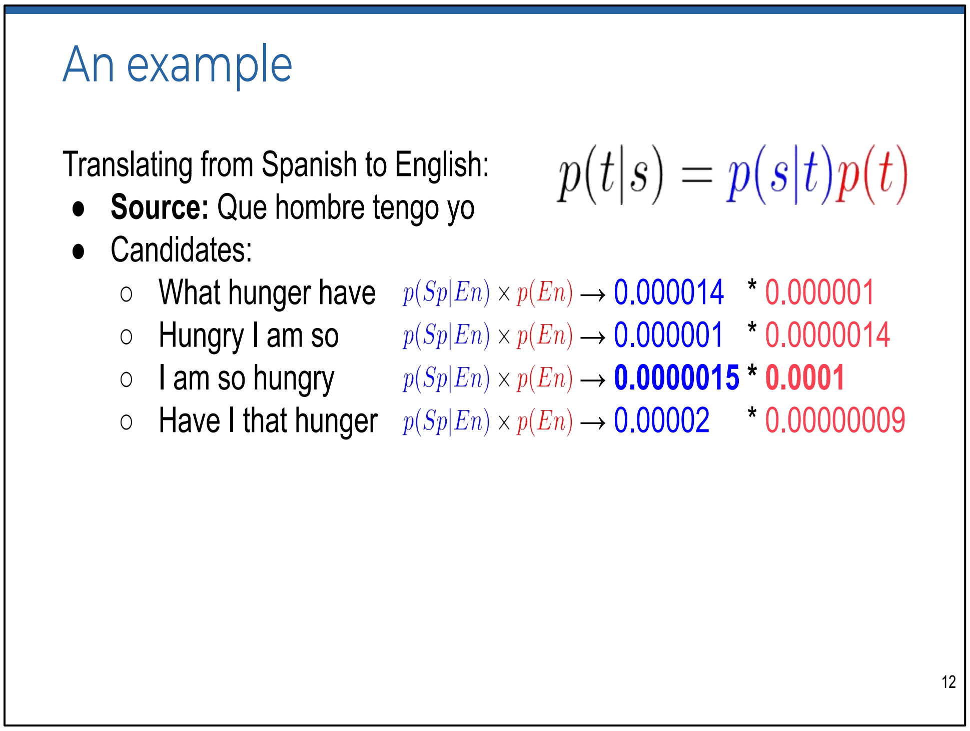
Translating from Spanish to English:

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

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

11

- Remember that translation model is asking p(s|t). In other words, how likely is the source sentence given a candidate target sentence



An example

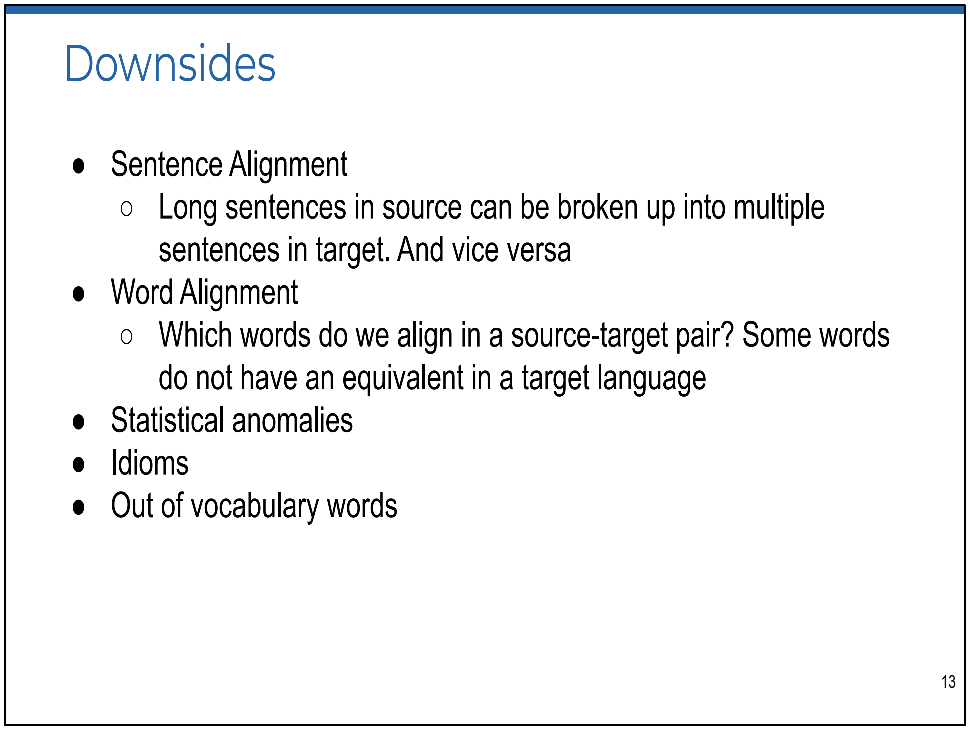
Translating from Spanish to English:

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

- **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

12

- Language model p(t) then says: for each of the candidate targets, how likely is that phrase to occur in the target language.
- Multiplying these probabilities gives us p(t|s)

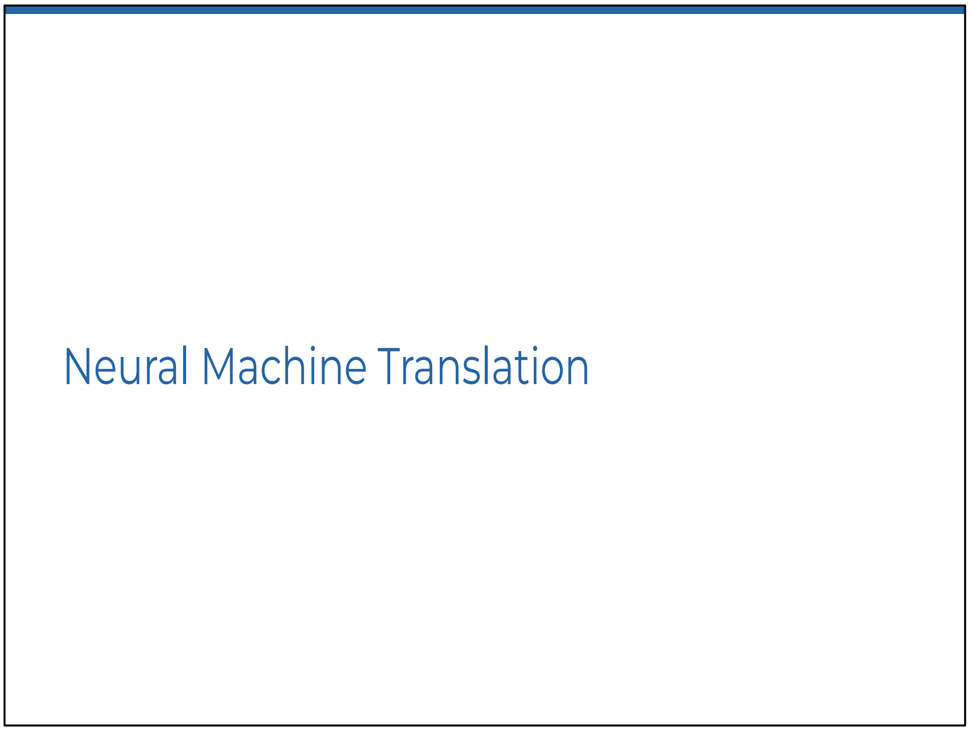


Downsides

- Sentence Alignment
  - Long sentences in source can be broken up into multiple sentences in target. And vice versa
- Word Alignment
  - Which words do we align in a source-target pair? Some words do not have an equivalent in a target language
- Statistical anomalies
- Idioms
- Out of vocabulary words

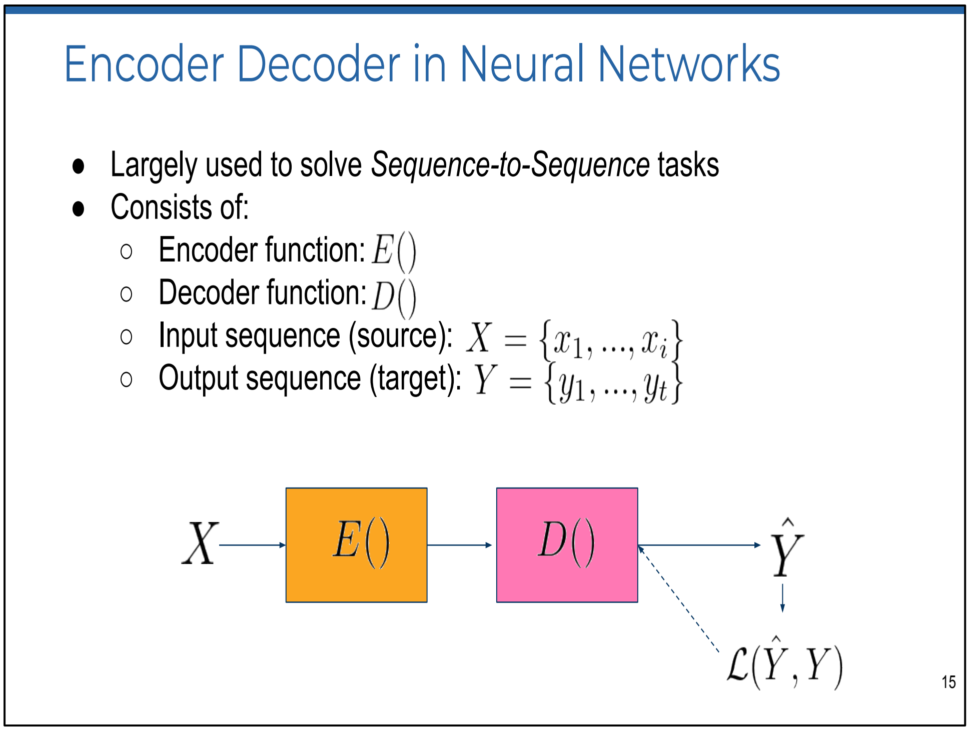
13

- 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!”.



Neural Machine Translation

Running ETA: 13

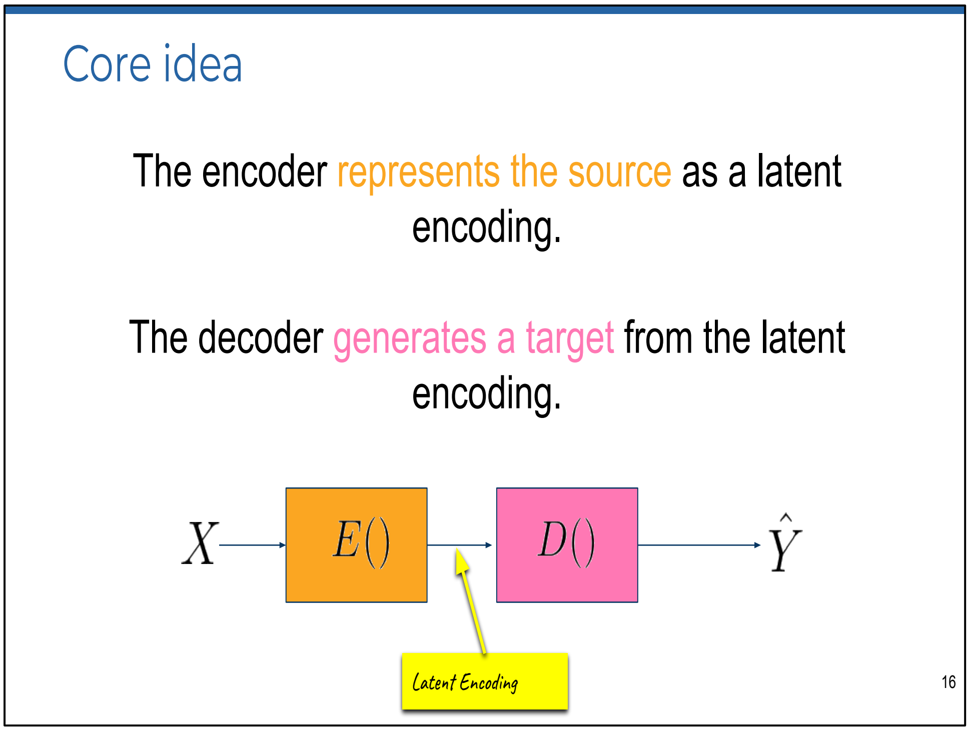


Encoder Decoder in Neural Networks

- Largely used to solve *Sequence-to-Sequence* tasks
- Consists of:
  - Encoder function: $E()$
  - Decoder function: $D()$
  - Input sequence (source): $X = \{x_1, ..., x_i\}$
  - Output sequence (target): $Y = \{y_1, ..., y_t\}$

$$\begin{tikzcd} X \arrow[r] & E() \arrow[r] & D() \arrow[r] & \hat{Y} \arrow[d] \\ & & & \mathcal{L}(\hat{Y}, Y) \arrow[ul, dashed] \end{tikzcd}$$

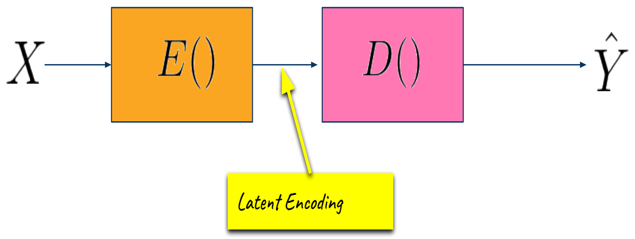
- Sequence to sequence tasks: Summarization, Generative Question Answering



Core idea

The encoder represents the source as a latent encoding.

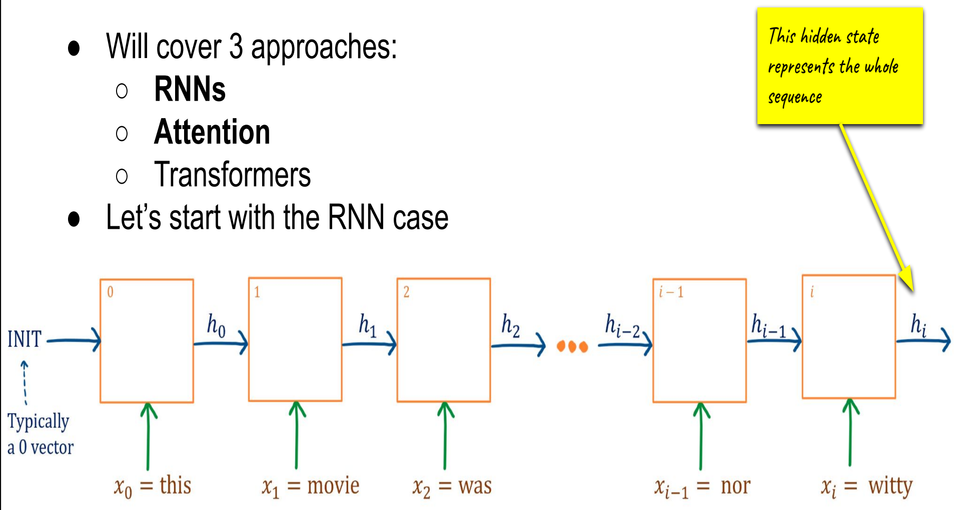
The decoder generates a target from the latent encoding.



- Think of a decoder as a language model that you looked at last week

Ok... but how?

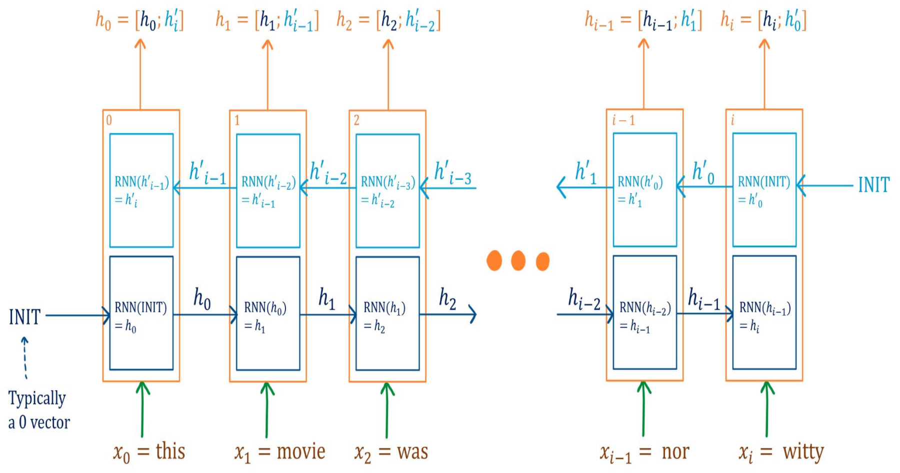
- Will cover 3 approaches:
  - **RNNs**
  - **Attention**
  - Transformers
- Let's start with the RNN case



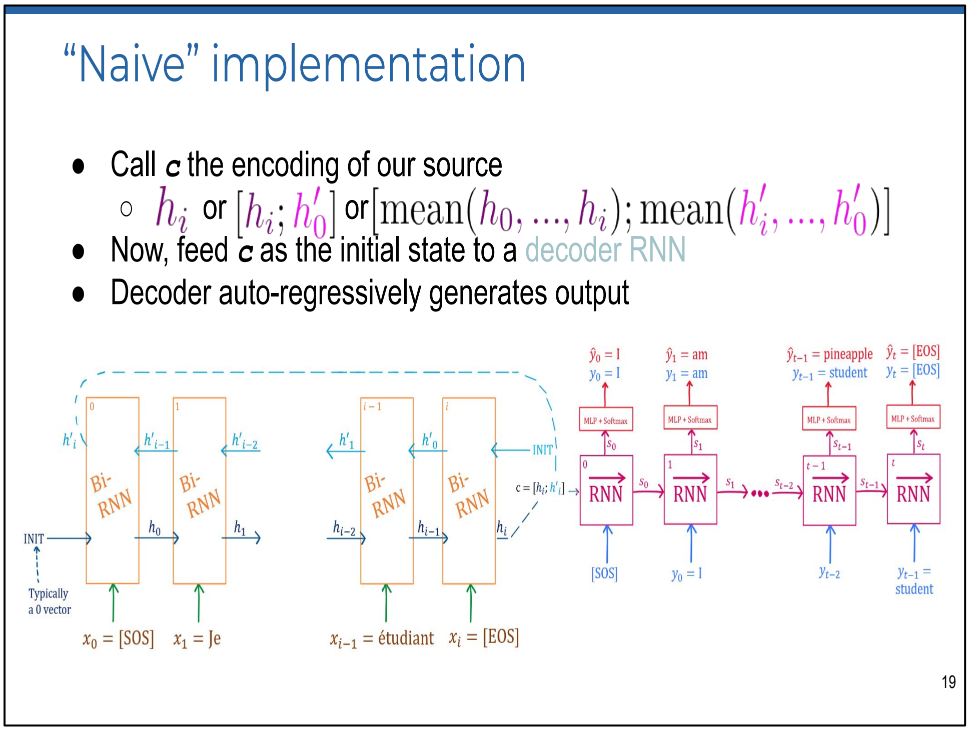
-

BiDirectional RNN

- More informative representations

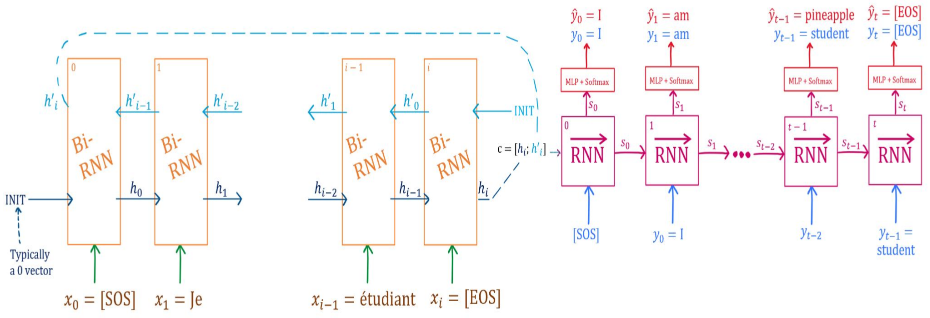


- Ask audience: What is a BiRNN?
  - Ans: Has an RNN that processes a sequence forward in time. And a different RNN that processes it backwards in time
- Using a BiRNN is appropriate for encoding the source as we're not trying to predict the source. We have access to the whole thing at inference time. Using a BiRNN, and reading information from both sides, allows us to gain a more information dense representation of our sequence.
- Different ways to use a BiRNN:
  - In some tasks we want access to hidden states for each word
  - Or we can get a whole sequence vector by concatenating [h_i ; h'_0]



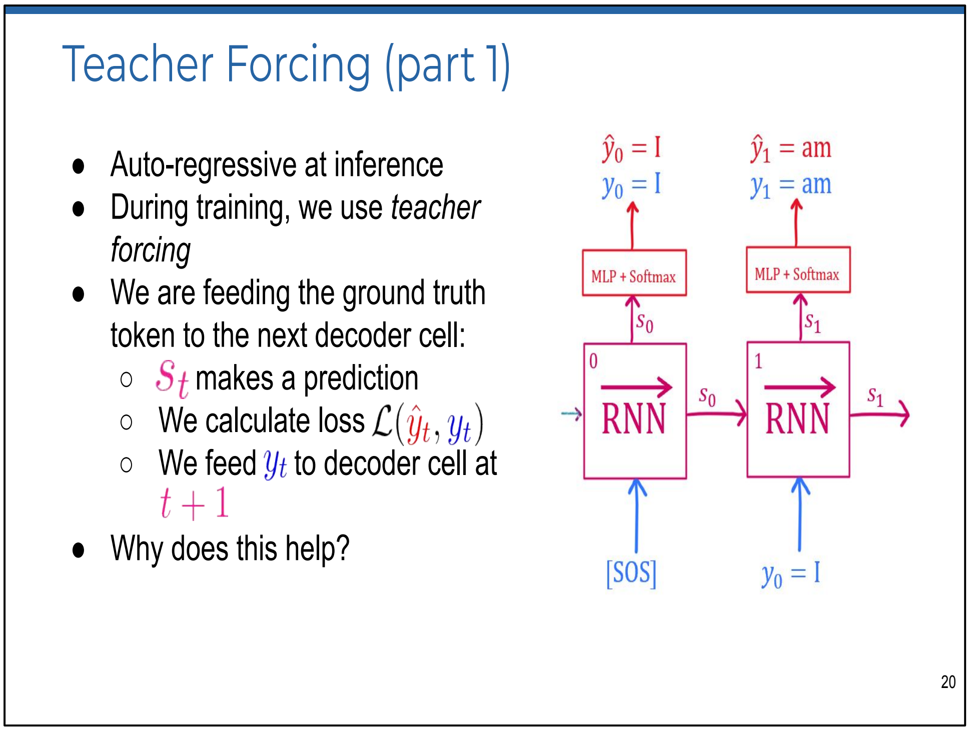
"Naive" implementation

- Call ***c*** the encoding of our source
  - h_i or $[h_i; h'_0]$ or $[\mathrm{mean}(h_0, ..., h_i); \mathrm{mean}(h'_i, ..., h'_0)]$
- Now, feed ***c*** as the initial state to a decoder RNN
- Decoder auto-regressively generates output



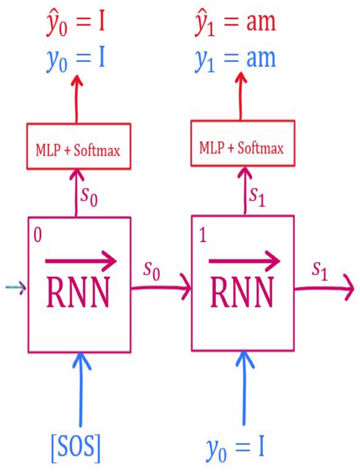
- Well.. a couple of ways to actually get that sequence vector: Can also use the mean of the hidden states concatenated together as our representations
- Though we will just use = [h_i; h'_0]

- Think of the decoder as a typical language model (that you looked at last week). The only difference is that we initialise it with our c vector instead of a zero-vector
- Ask class: if the dimensionality of our model is d, what is the dimensionality of c if we incorporate both forwards and backward direction in it?
  - Practically we either have to set s in R^2d or project c down to R^d



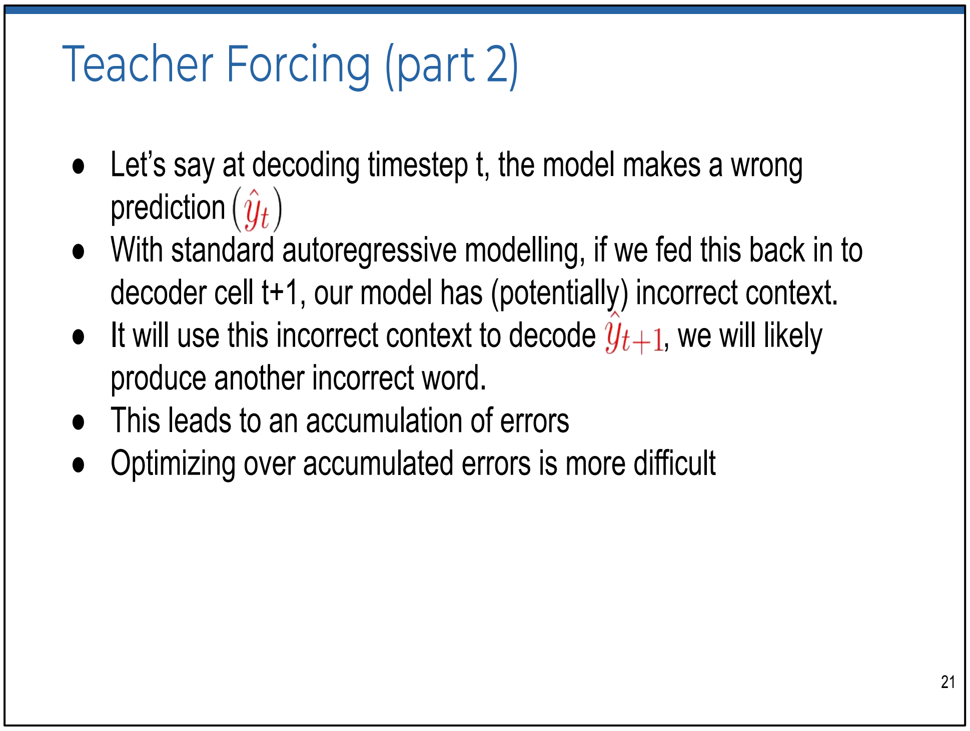
Teacher Forcing (part 1)

- Auto-regressive at inference
- During training, we use *teacher forcing*
- We are feeding the ground truth token to the next decoder cell:
  - s_t makes a prediction
  - We calculate loss $\mathcal{L}(\hat{y}_t, y_t)$
  - We feed y_t to decoder cell at $t+1$
- Why does this help?



20

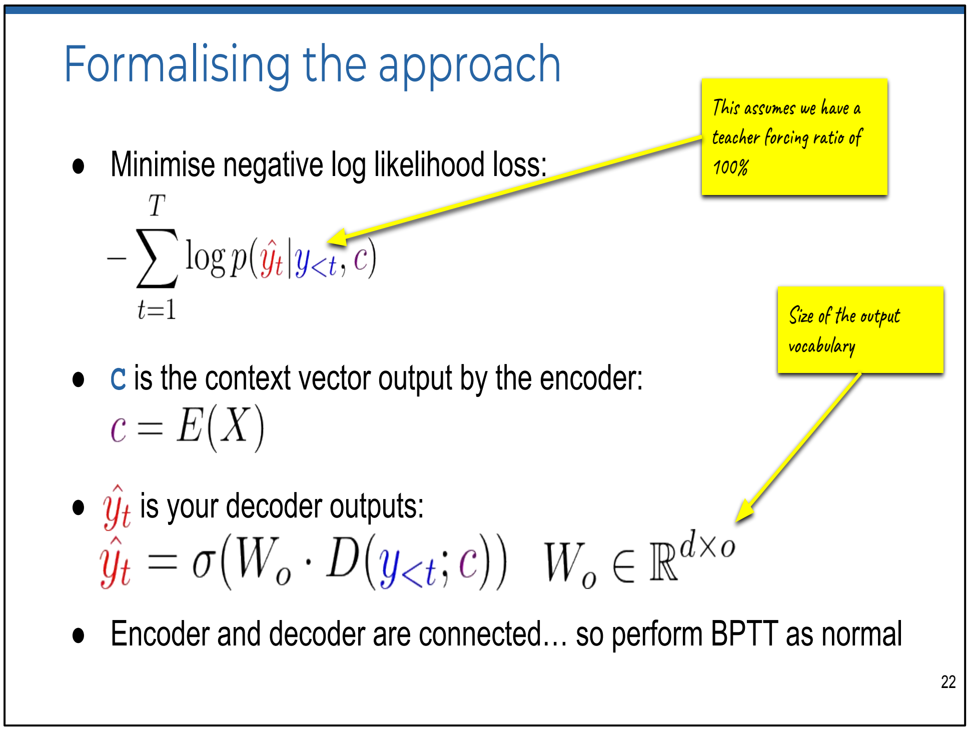
- Decoder is auto-regressive only during inference
- During training, we use teacher forcing with a *ratio*. I.e. we well teacher force ratio% of time. The ratio can be 100% (i.e. full teacher forcing), 50%, or you can even anneal it during training.
-



Teacher Forcing (part 2)

- Let's say at decoding timestep t, the model makes a wrong prediction $(\hat{y}_t)$
- With standard autoregressive modelling, if we fed this back in to decoder cell t+1, our model has (potentially) incorrect context.
- It will use this incorrect context to decode $\hat{y}_{t+1}$, we will likely produce another incorrect word.
- This leads to an accumulation of errors
- Optimizing over accumulated errors is more difficult

- 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
- It's unclear how much of an issue exposure bias actually is. There are multiple papers on either side of the argument.



Formalising the approach

This assumes we have a teacher forcing ratio of 100%

- Minimise negative log likelihood loss:

$$-\sum_{t=1}^{T} \log p(\hat{y}_t | y_{<t}, c)$$

Size of the output vocabulary

- c is the context vector output by the encoder:

$$c = E(X)$$

- $\hat{y}_t$ is your decoder outputs:

$$\hat{y}_t = \sigma(W_o \cdot D(y_{<t}; c)) \quad W_o \in \mathbb{R}^{d \times o}$$

- Encoder and decoder are connected… so perform BPTT as normal

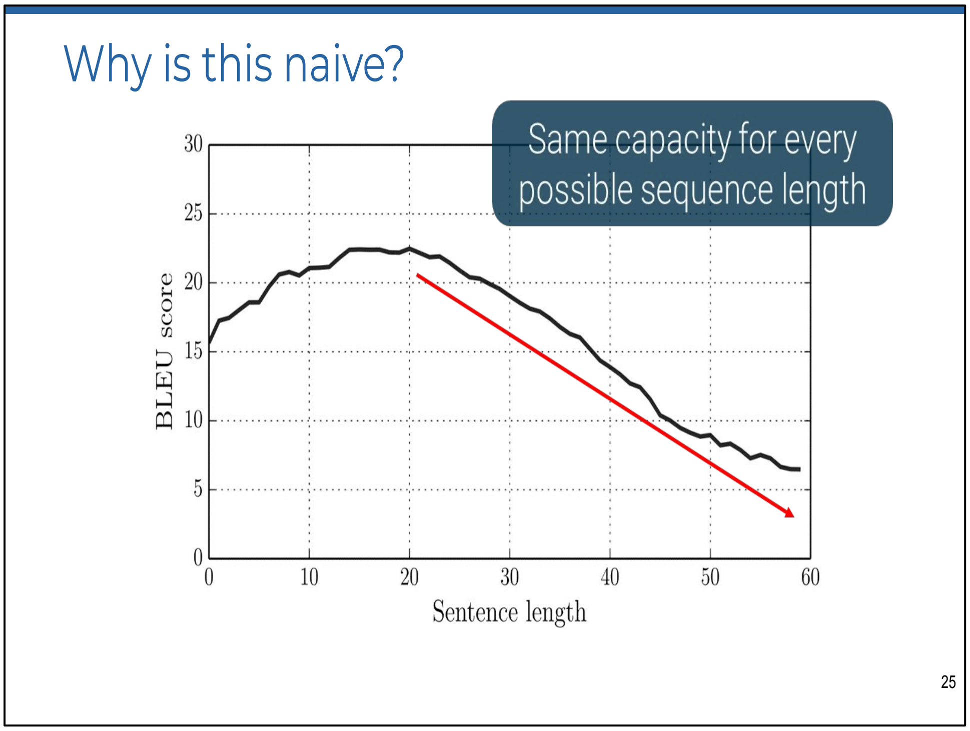
- y_<t means all the tokens up to the t'th timestep. With the notation I've put here, it assumes a teacher forcing ratio of 100%. If you didn't use a teacher forcing ratio of 100%, then you could write this as a joint of your previously decoded tokens (whether they're sampled or the ground truth).
- o is your output vocabulary size. What activation function does sigma take on? (A: Softmax)

CODE

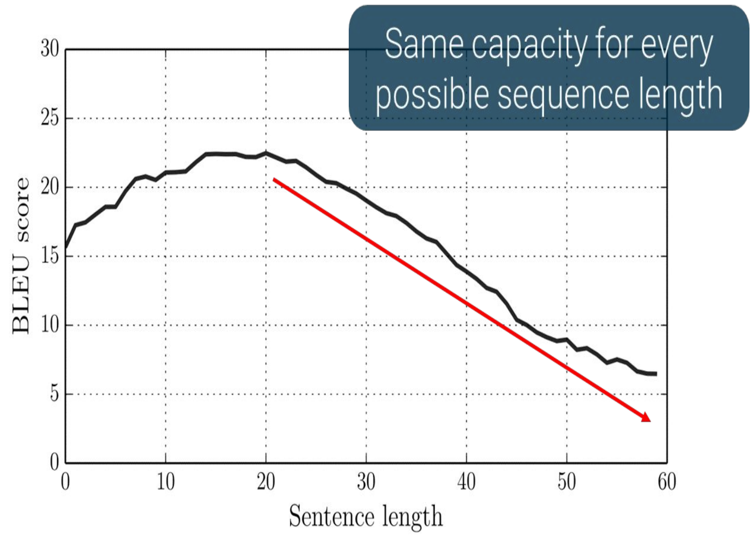
- Running ETA: 21

Why is this naive?

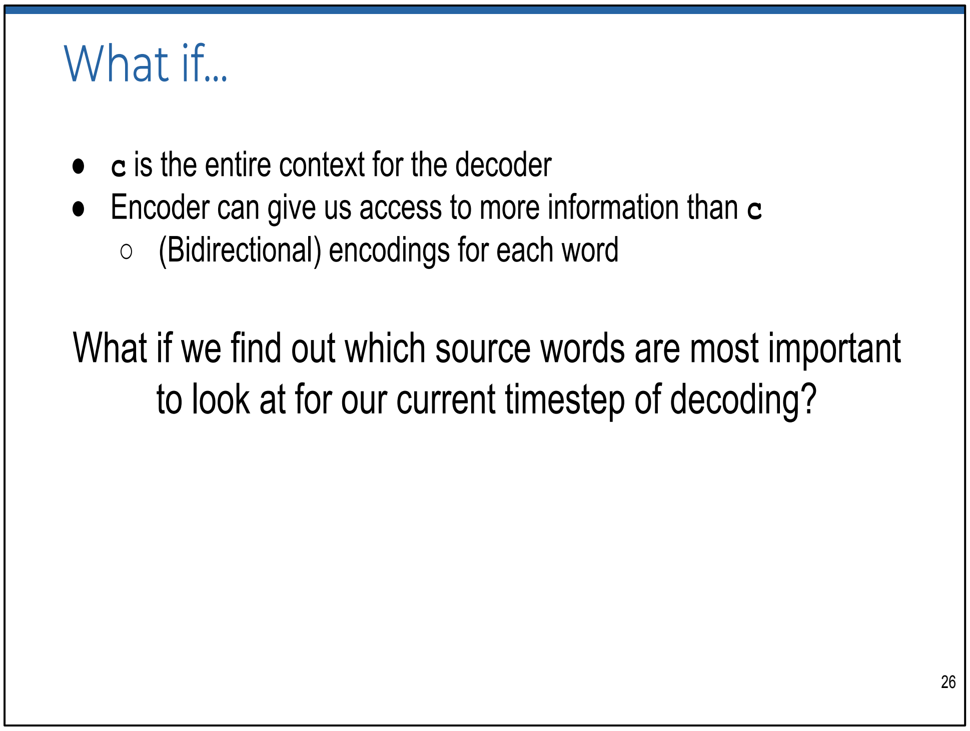
- Ask students about what the drawbacks of this implementation are. Guide them to long sequences in a fixed-dimensional representation



Why is this naive?



- All sequences are being represented by a d-dimensional vector. 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



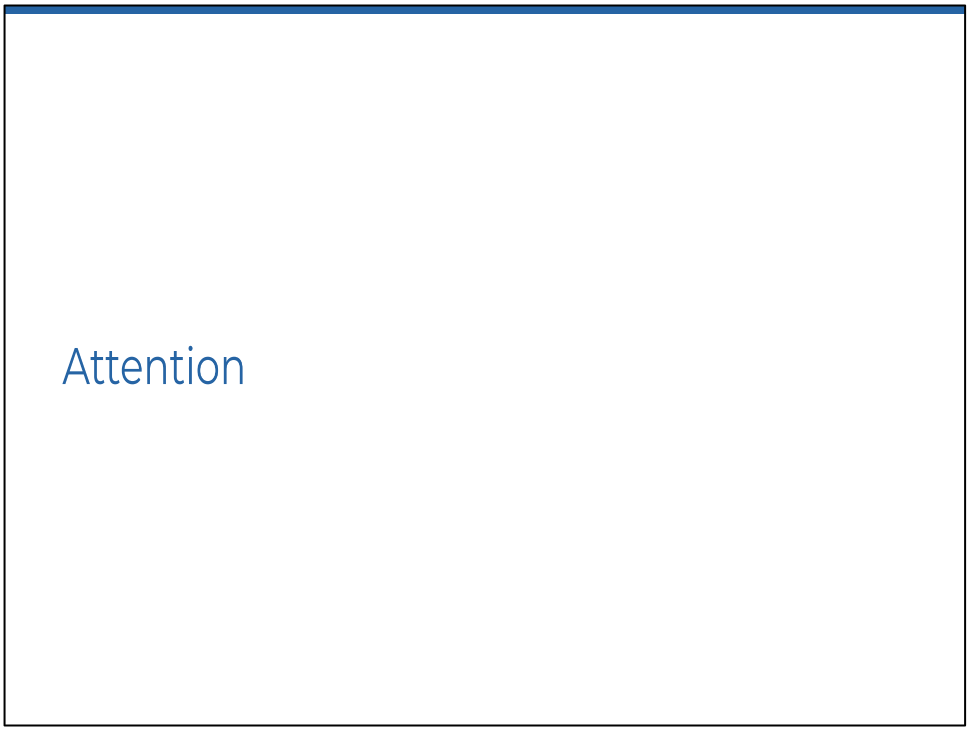
What if...

- **c** is the entire context for the decoder
- Encoder can give us access to more information than **c**
  - (Bidirectional) encodings for each word

What if we find out which source words are most important to look at for our current timestep of decoding?

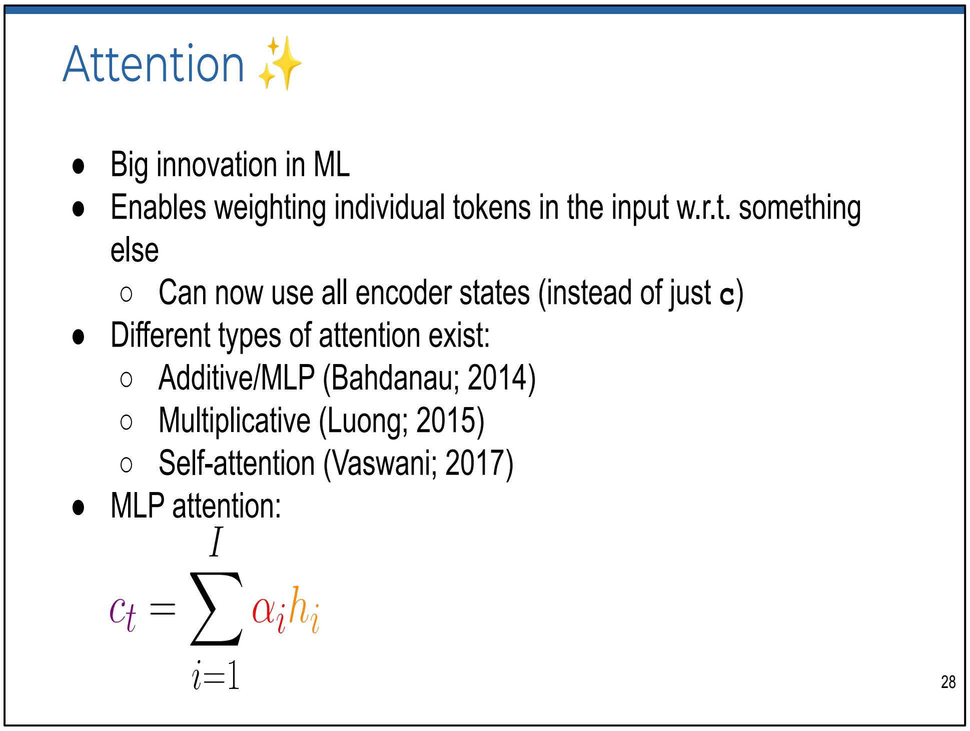
26

- Currently we are relying on c as the entire context for the decoder. This has the drawbacks I just mentioned in the previous slide.
- However, the encoder can give us access to context encodings of each word. For example, the word x_i has a bidirectional contextual representation. If we could utilise the these more fine-grained encodings, we wouldn't suffer from the loss of information that we get from a context vector
- We can't trivially feed in a matrix as the initial state to our decoder though… so we have to think of something slightly more sophisticated
- What if, during decoding, the current decoding timestep could look at all the source words, and find out which source words are most important to its current stage of decoding? Then, could we 'pay attention' to those words more than the other words?



Attention

Running ETA: 38



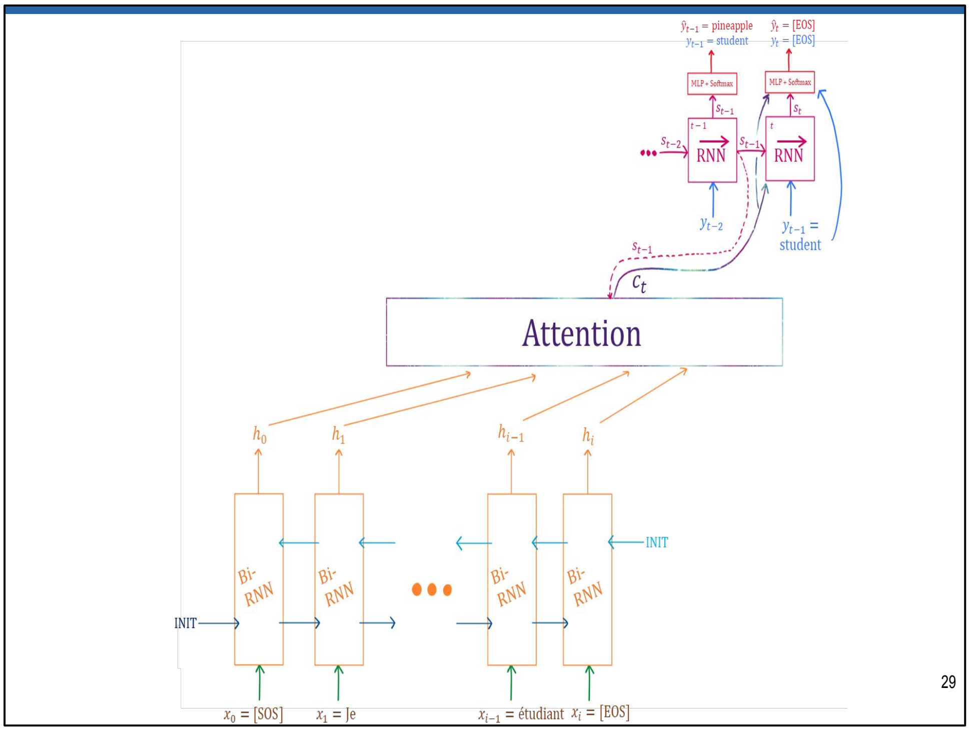
Attention ✨

- Big innovation in ML
- Enables weighting individual tokens in the input w.r.t. something else
  - Can now use all encoder states (instead of just **c**)
- Different types of attention exist:
  - Additive/MLP (Bahdanau; 2014)
  - Multiplicative (Luong; 2015)
  - Self-attention (Vaswani; 2017)
- MLP attention:

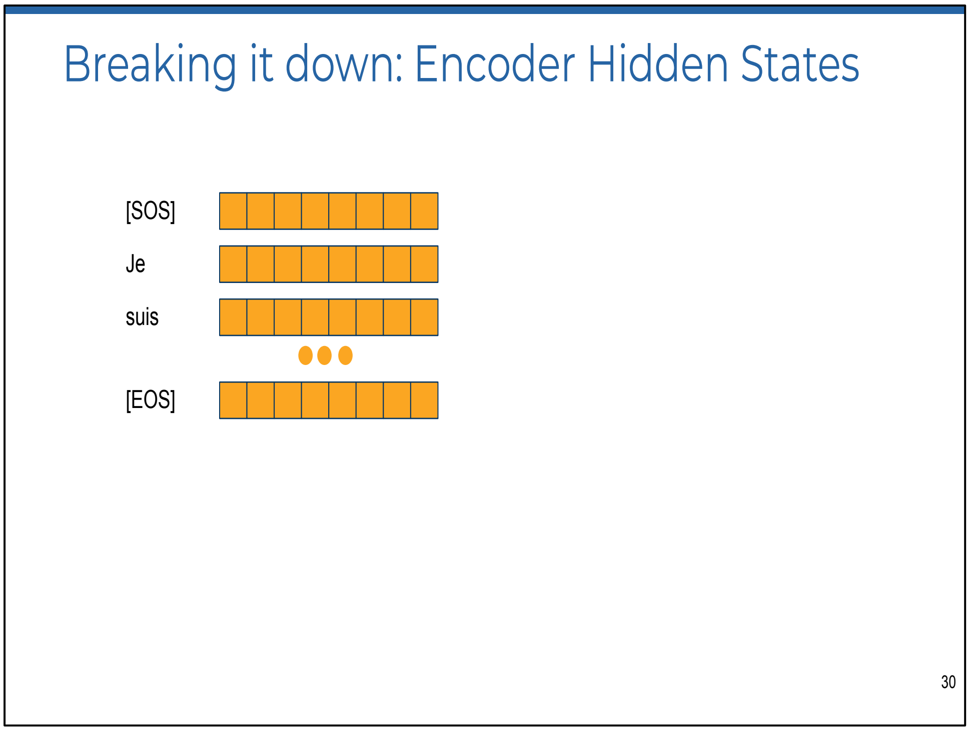
$$c_t = \sum_{i=1}^{I} \alpha_i h_i$$

28

- One of the biggest innovations in NLP and ML.
- Think of it as a dynamic weighted average
- In the immediate context, it allows us to dynamically look at individual tokens in the input and decide how much weighting a token should have with respect to the current timestep of decoding. As we'll see in future lectures, the weighting does not necessarily need to be in respect to decoding though.
- We'll focus on Bahdanau's variant.
- Ok. Let's break down this equation:
  - c_t is the context vector for the t'th decoding timestep
  - We then loop over all the hidden states i, and weight it by a scalar value alpha
  - So if alpha is 0, then the i'th hidden state is 0
  - If alpha is 1, then we retain the full information of that hidden state
  - We then sum together our weighted hidden states to obtain a contextualised representation for the t'th decoding step
  - Now the question is… how do we obtain alpha?
-



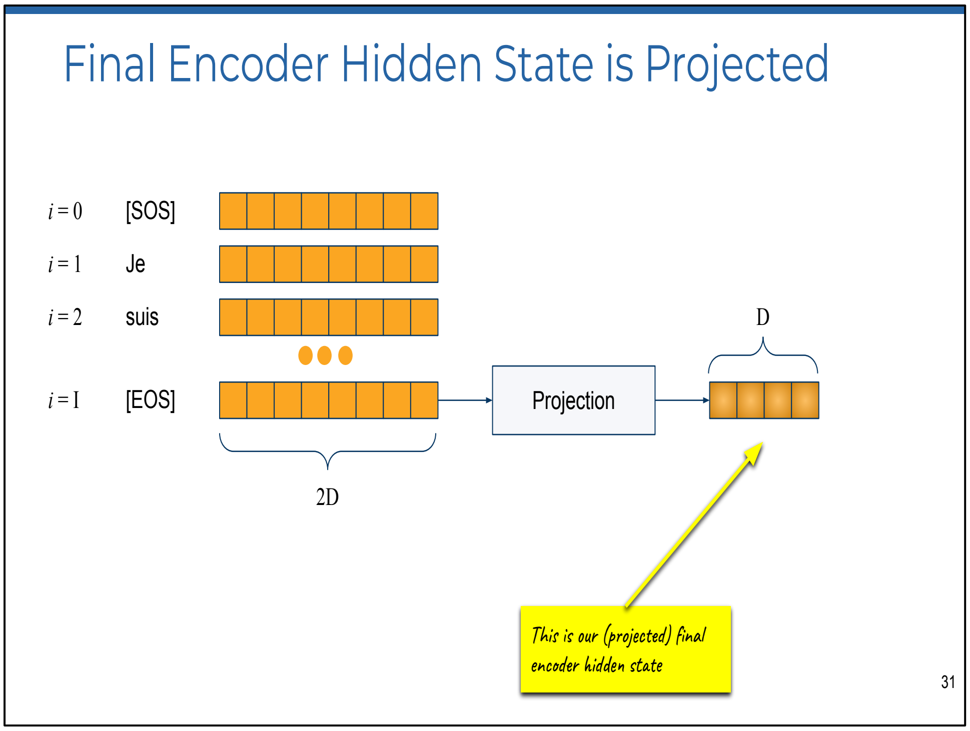
1. What we're trying to do is decode the y_t word
2. And we have access to bidirectional encodings of each source word
3. Think of the attention module as a black box for a second. We'll look at how it works in the next slide
4. So, before we decode y_t, we're going to feed all our encoder hidden states 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 decoding 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 also takes in c_t and the word embedding for y_t-1 (alongside s_t) to predict the t'th word.



Breaking it down: Encoder Hidden States

SPEAK SLOWLY

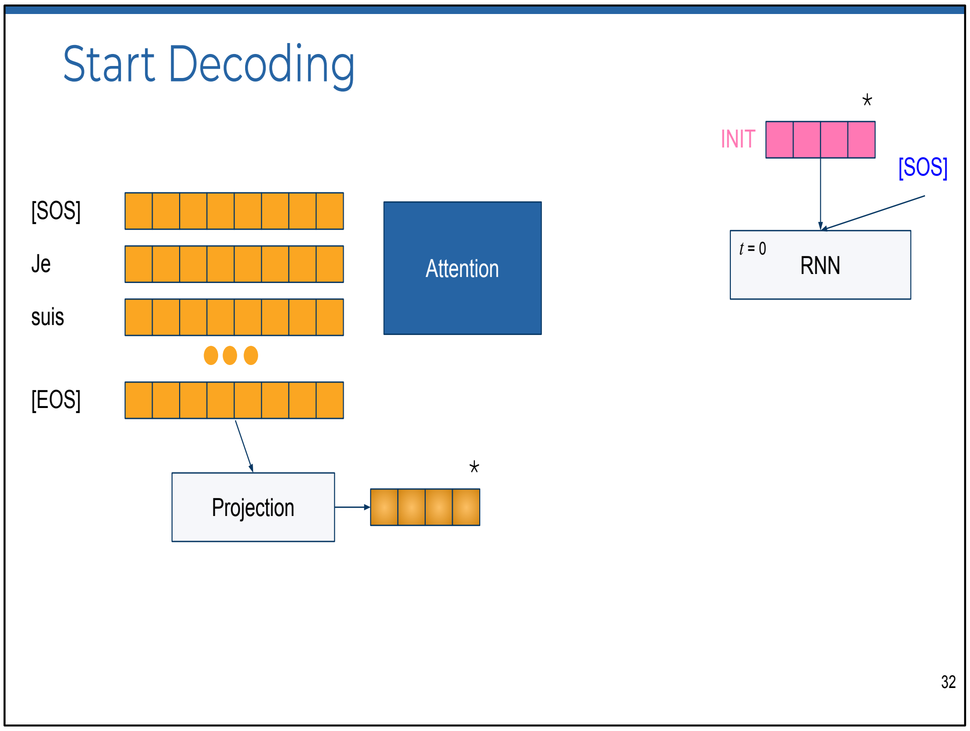
- These are our bidirectionally encoded encoder hidden states



Final Encoder Hidden State is Projected

SPEAK SLOWLY

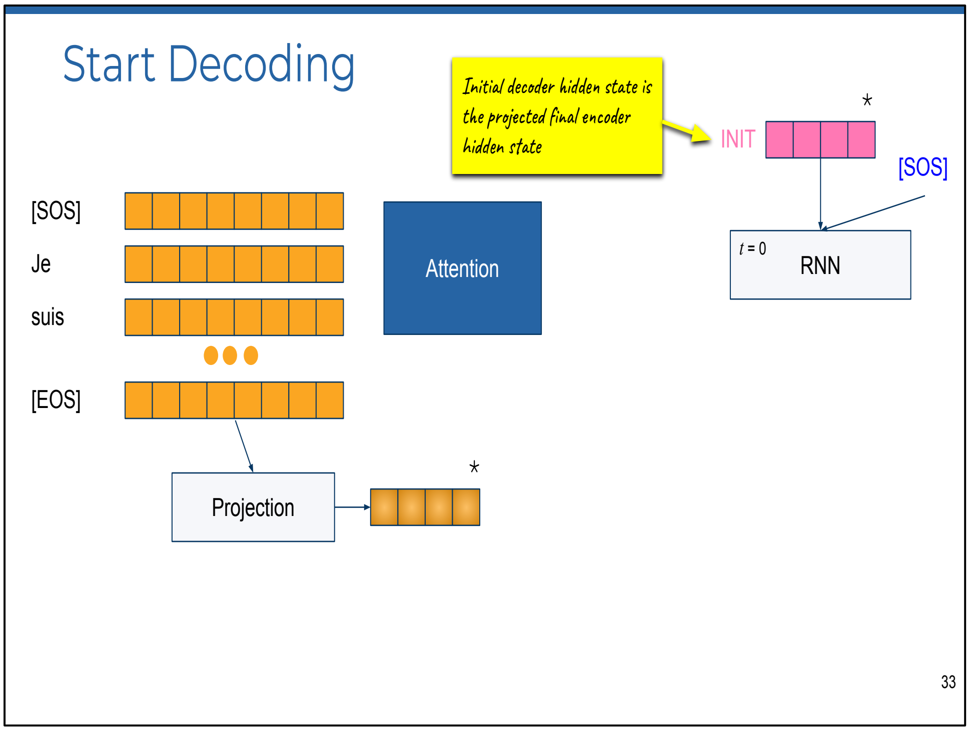
1. We have *I* words, and each word is encoded in *2D* dimensions
2. We project the last hidden state to obtain a *D* dimensional vector



Start Decoding

SPEAK SLOWLY

1. Now we’re looking at the decoding process. The decoder RNN has inputs: INIT vector, and the SOS token.

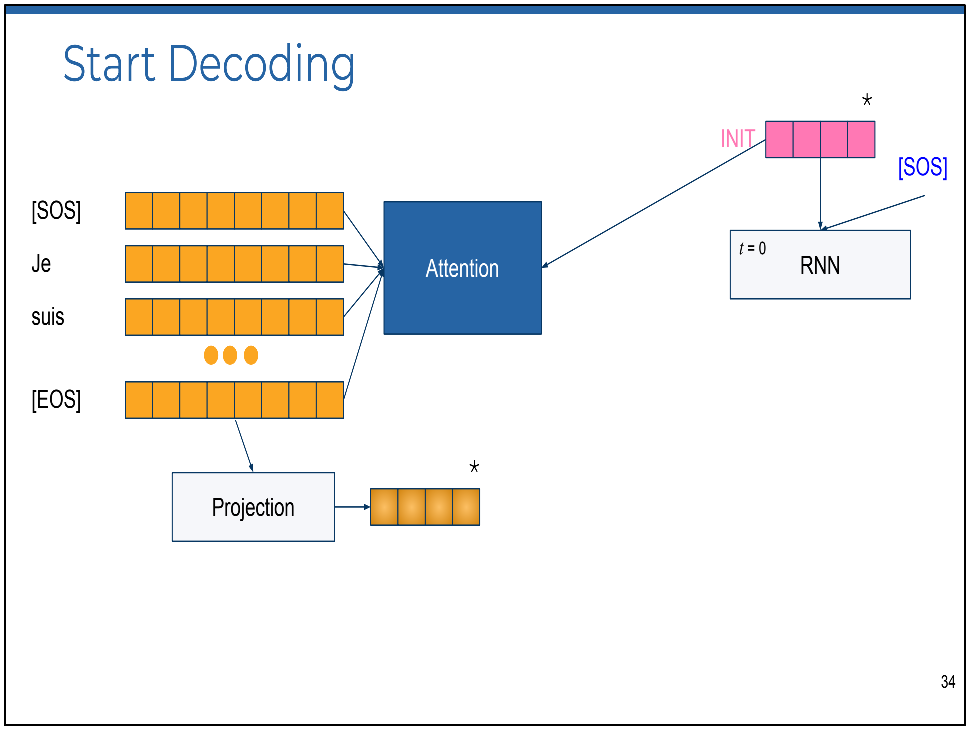


Start Decoding

Initial decoder hidden state is the projected final encoder hidden state
INIT
*
[SOS]
t = 0
RNN
[SOS]
Je
suis
[EOS]
Attention
Projection
*
33

SPEAK SLOWLY

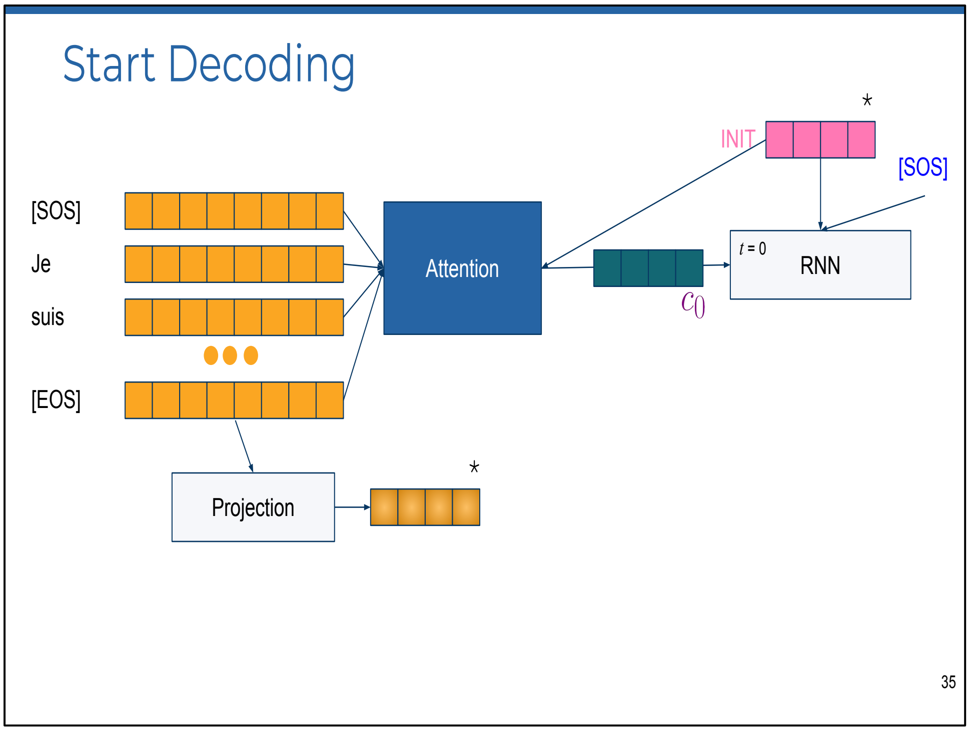
1. The INIT vector is the projected final hidden state



Start Decoding

SPEAK SLOWLY

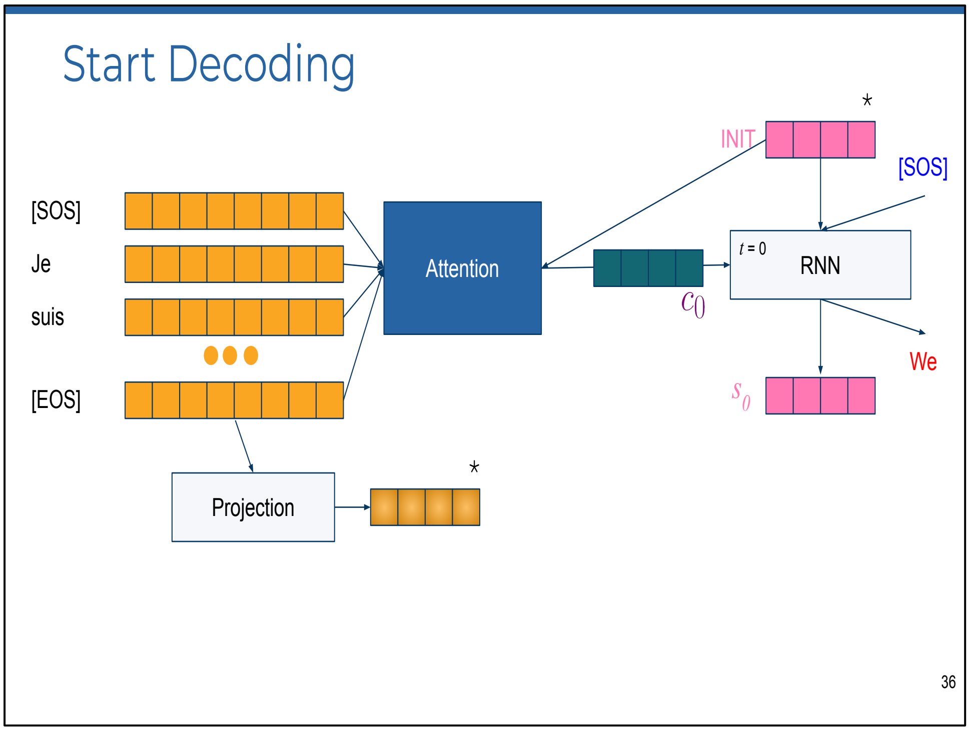
1. Before we generate the next predicted word, we run the attention mechanism
2. The attention mechanism takes ALL the encoder hidden states as input, and the decoder hidden state being inputted to the RNN to decode the current word



Start Decoding

SPEAK SLOWLY

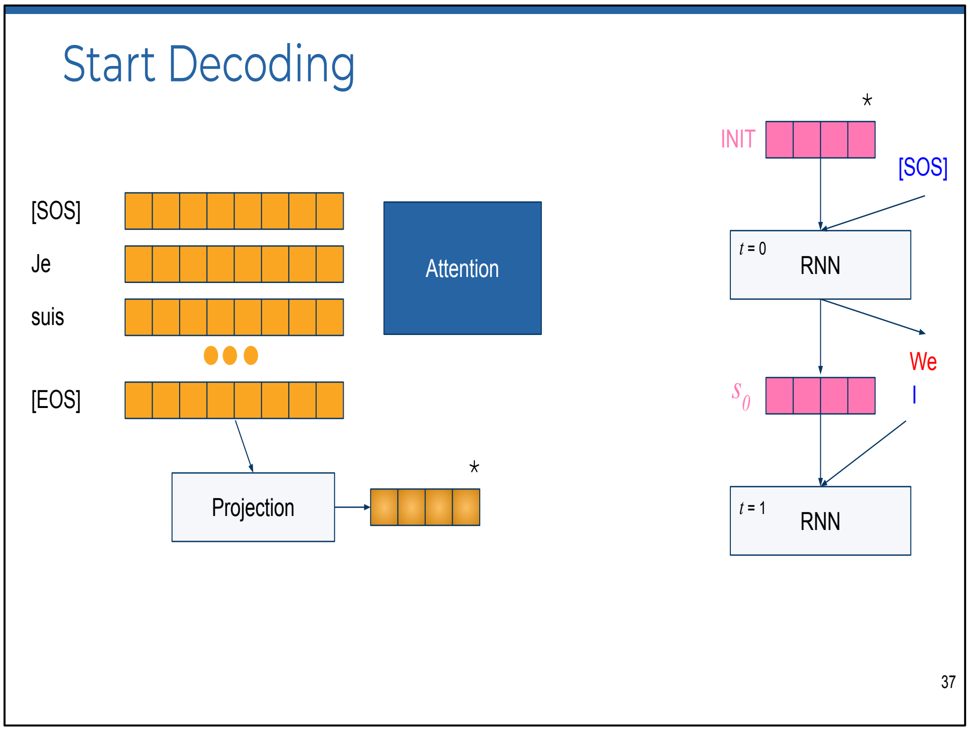
1. Attention is then performed over these inputs, and the module outputs a context vector specific to this decoding time step
2. We'll look through how the attention module itself works in a minute



Start Decoding

SPEAK SLOWLY

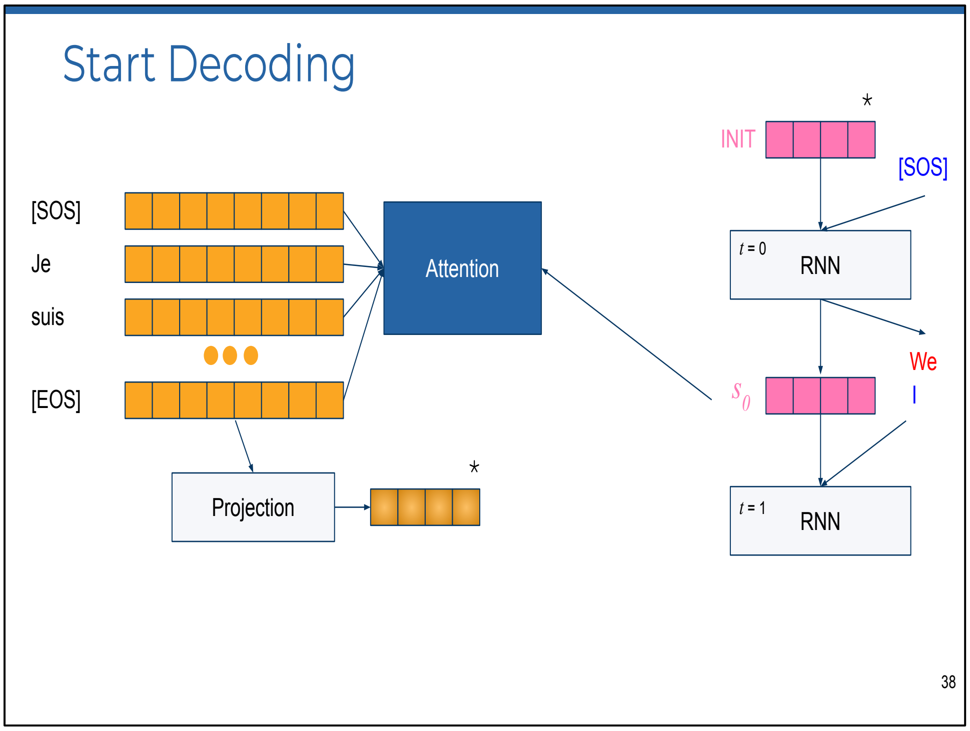
1. The RNN now has 3 inputs: The hidden state (INIT in this case), a context vector specific to the current decoding step (c_0), and the input token (SOS)
2. Using these 3 inputs, it gives us 2 outputs: 1) Our decoder hidden state vector (s_0), and also a prediction of the first word (We)



Start Decoding

SPEAK SLOWLY

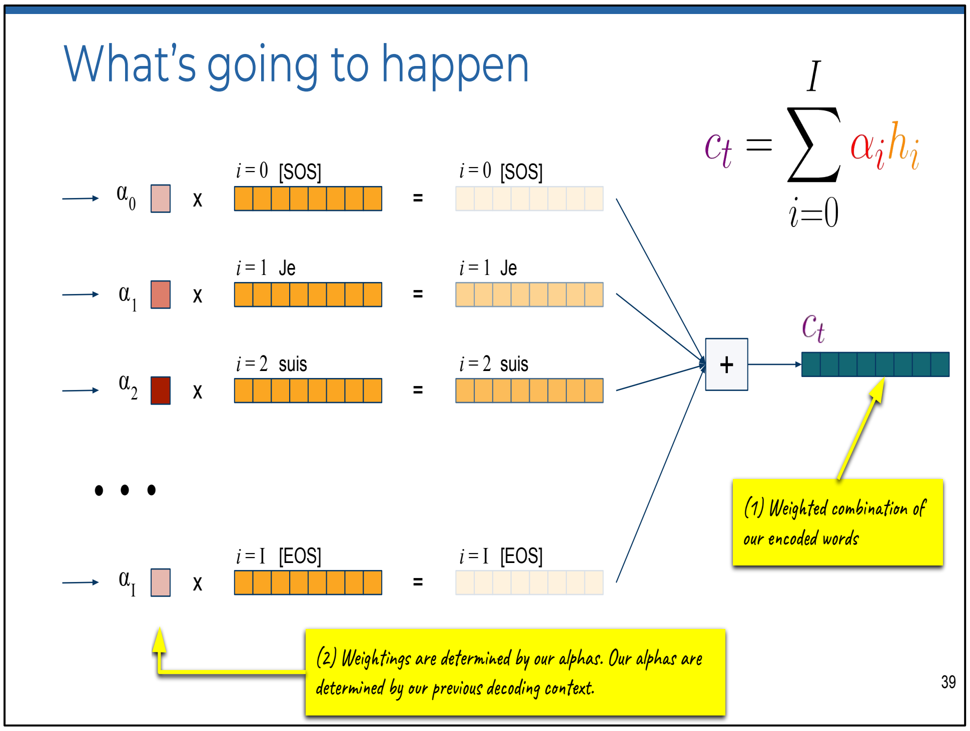
1. We then start decoding the next word. If we're using teacher forcing, we feed in the correct ground truth word as input (instead of the prediction)



Start Decoding

SPEAK SLOWLY

1. Again, we run the attention process. We take in ALL encoder hidden states as input, and the previously outputted decoder hidden state
2. Let’s now see what happens inside the attention process



What’s going to happen

$$c_t = \sum_{i=0}^{I} a_i h_i$$

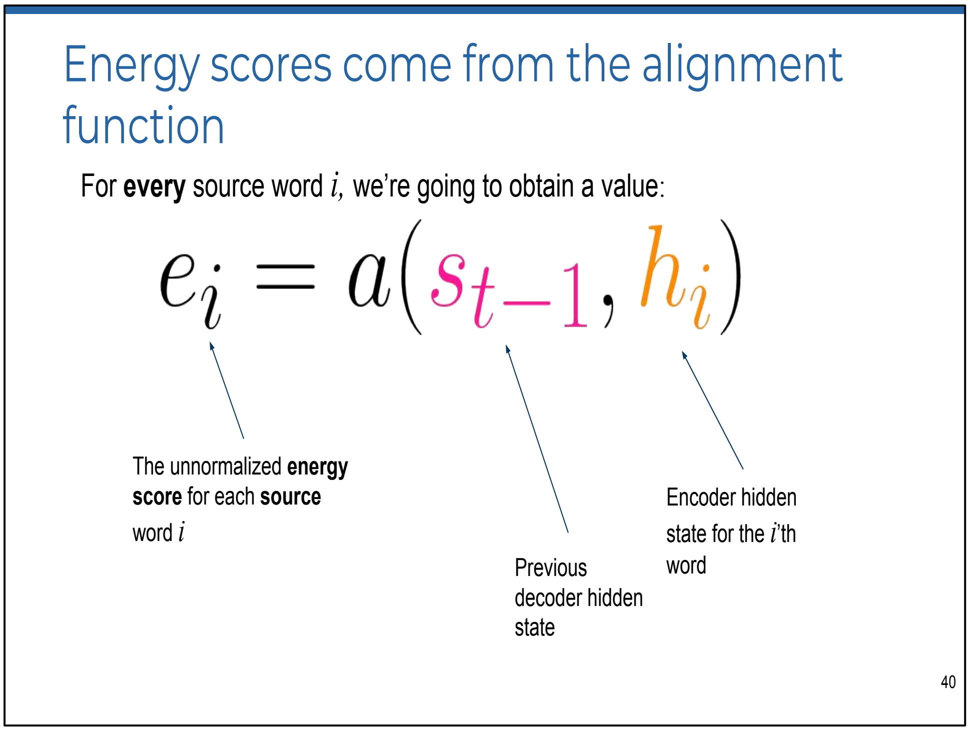
α_0 x $i=0$ [SOS] = $i=0$ [SOS]
α_1 x $i=1$ Je = $i=1$ Je
α_2 x $i=2$ suis = $i=2$ suis
α_I x $i=I$ [EOS] = $i=I$ [EOS]
+
c_t

(1) Weighted combination of our encoded words

(2) Weightings are determined by our alphas. Our alphas are determined by our previous decoding context.

SPEAK SLOWLY

1. Before we break down exactly what’s going on - let’s look at what it’s doing
2. Attention is going to give us alpha values for EACH encoded word. Alphas will be between 0 and 1.
3. We’re going to scale each encoded word by alpha
4. And sum them all together to obtain a context vector
5. We already have the encoded hidden states. So the question is - how do we get our alphas?



Energy scores come from the alignment function

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

$$e_i = a(s_{t-1}, h_i)$$

The unnormalized **energy score** for each **source** word i

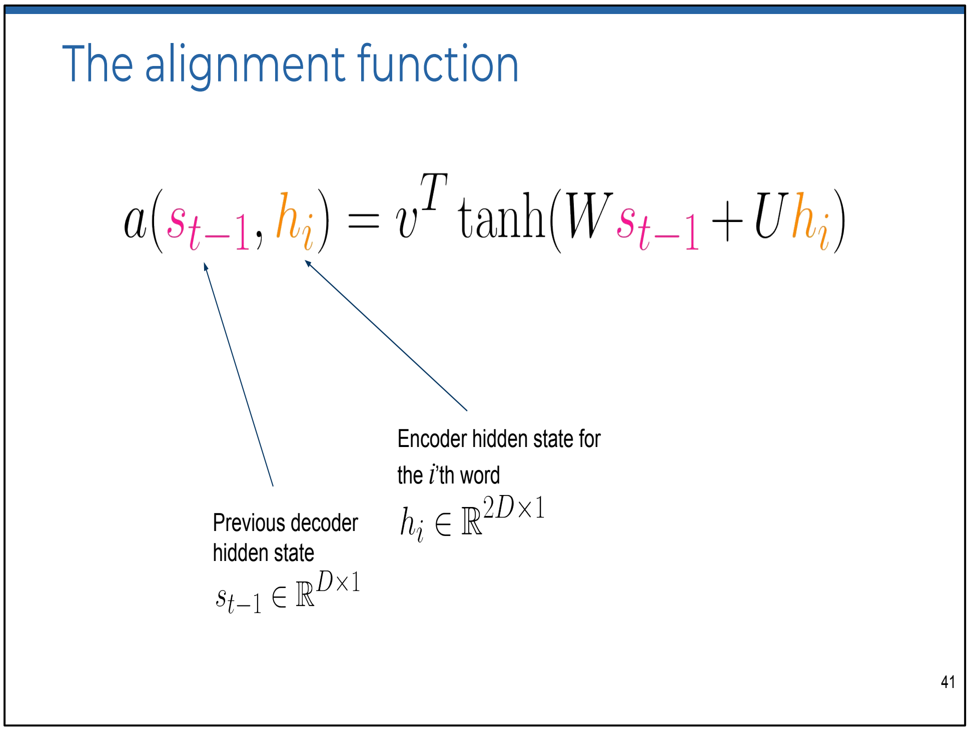
Previous decoder hidden state

Encoder hidden state for the i'th word

40

SPEAK SLOWLY

1. Our alphas will be normalized energy scores
2. We're going to obtain an energy score for EACH source word
3. Energy scores are obtained using an alignment function, a
4. a() takes 2 inputs: The previous decoder hidden state (s_t-1) and the encoded hidden state for ONE word (h_i)
5. Just to be clear - this function is run for every source word we have



The alignment function

$$a(s_{t-1}, h_i) = v^T \tanh(W s_{t-1} + U h_i)$$

Encoder hidden state for the i'th word

$h_i \in \mathbb{R}^{2D\times 1}$

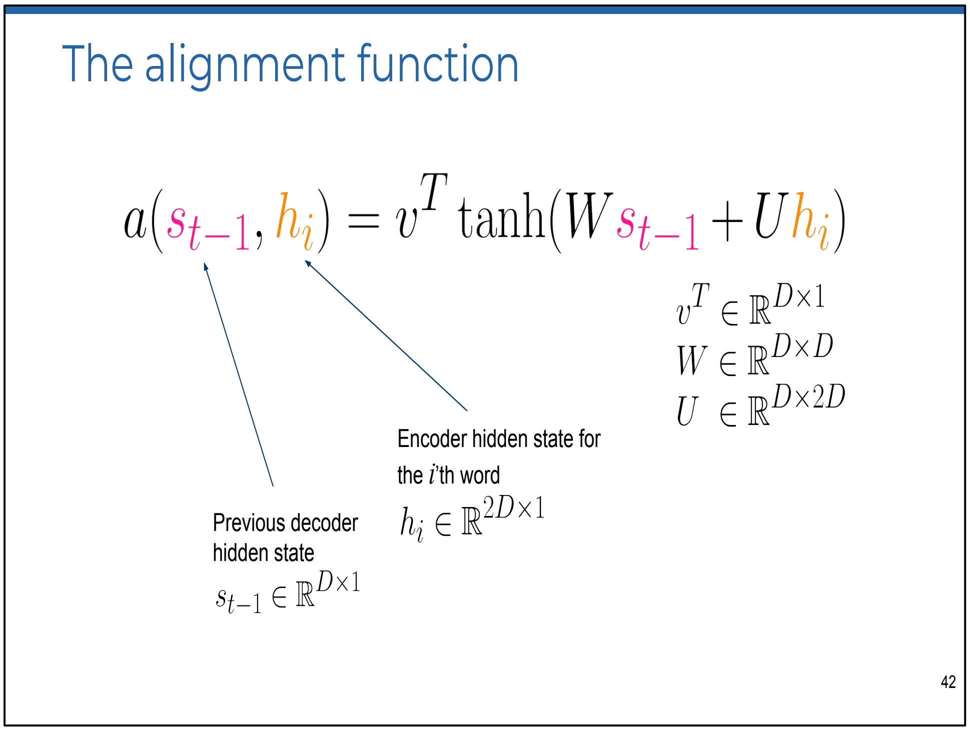
Previous decoder hidden state

$s_{t-1} \in \mathbb{R}^{D\times 1}$

41

SPEAK SLOWLY

1. Before we try to interpret what a() is doing, let's take a look at the shapes it takes as input:
   a. The previous decoder hidden state s_t-1 is a D dimensional vector.
   b. The encoder hidden state for ONE word (the i'th word), is a 2D dimensional vector



The alignment function

$$a(s_{t-1}, h_i) = v^T \tanh(W s_{t-1} + U h_i)$$

$v^T \in \mathbb{R}^{D \times 1}$
$W \in \mathbb{R}^{D \times D}$
$U \in \mathbb{R}^{D \times 2D}$

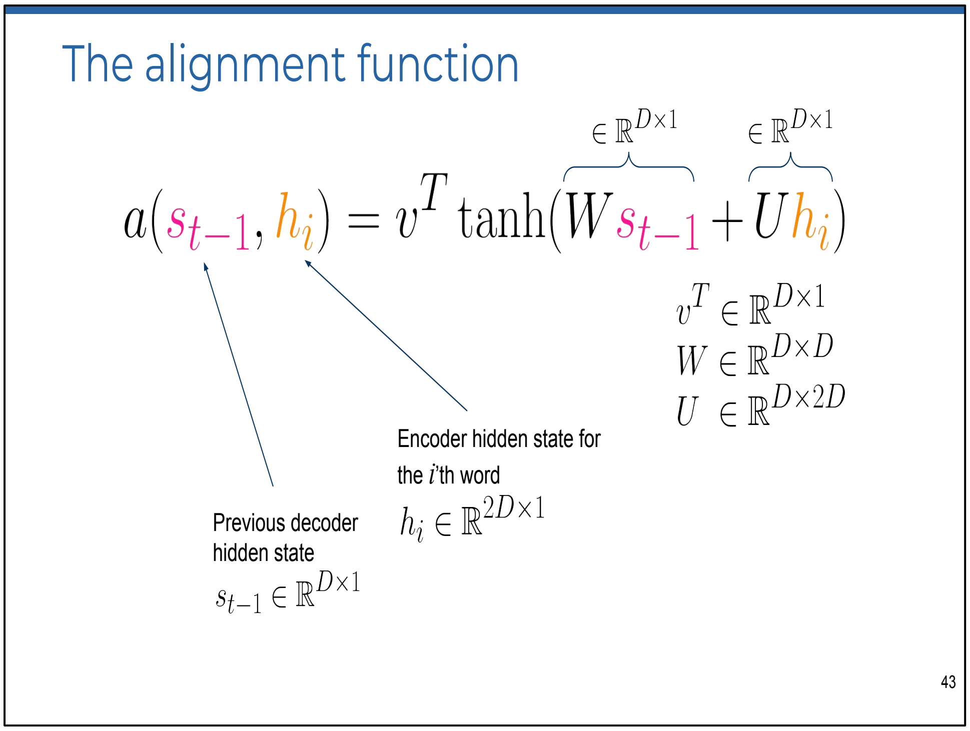
Encoder hidden state for the i'th word
$h_i \in \mathbb{R}^{2D \times 1}$

Previous decoder hidden state
$s_{t-1} \in \mathbb{R}^{D \times 1}$

42

SPEAK SLOWLY

1. The function itself can be broken down into 5 steps



The alignment function

$$a(s_{t-1}, h_i) = v^T \tanh(\overbrace{W s_{t-1}}^{\in \mathbb{R}^{D\times 1}} + \overbrace{U h_i}^{\in \mathbb{R}^{D\times 1}})$$

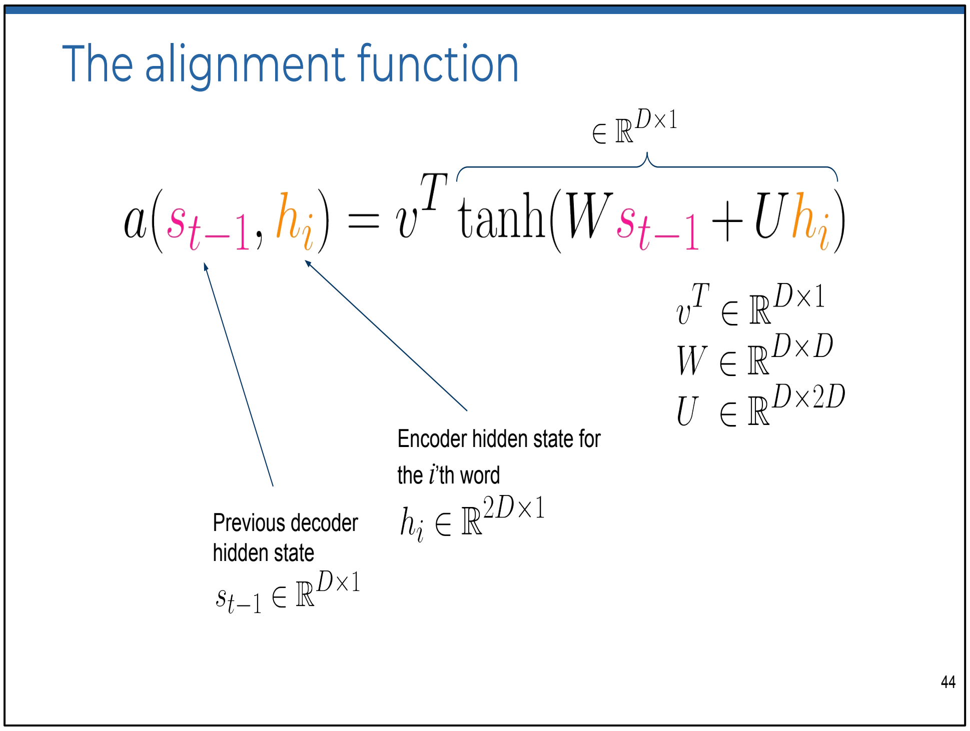
$$v^T \in \mathbb{R}^{D\times 1}$$
$$W \in \mathbb{R}^{D\times D}$$
$$U \in \mathbb{R}^{D\times 2D}$$

Encoder hidden state for the i'th word
$h_i \in \mathbb{R}^{2D\times 1}$

Previous decoder hidden state
$s_{t-1} \in \mathbb{R}^{D\times 1}$

SPEAK SLOWLY

1. The first 2 parts perform transformations on our inputs.
2. We project/transform h_i to a D-dimensional vector via a learned weights matrix U
3. Similarly, we project s_t-1 via W. This also results in a D-dimensional vector



The alignment function

$$a(s_{t-1}, h_i) = v^T \overbrace{\tanh(W s_{t-1} + U h_i)}^{\in \mathbb{R}^{D\times 1}}$$

$$v^T \in \mathbb{R}^{D\times 1}$$
$$W \in \mathbb{R}^{D\times D}$$
$$U \in \mathbb{R}^{D\times 2D}$$

Encoder hidden state for the i'th word

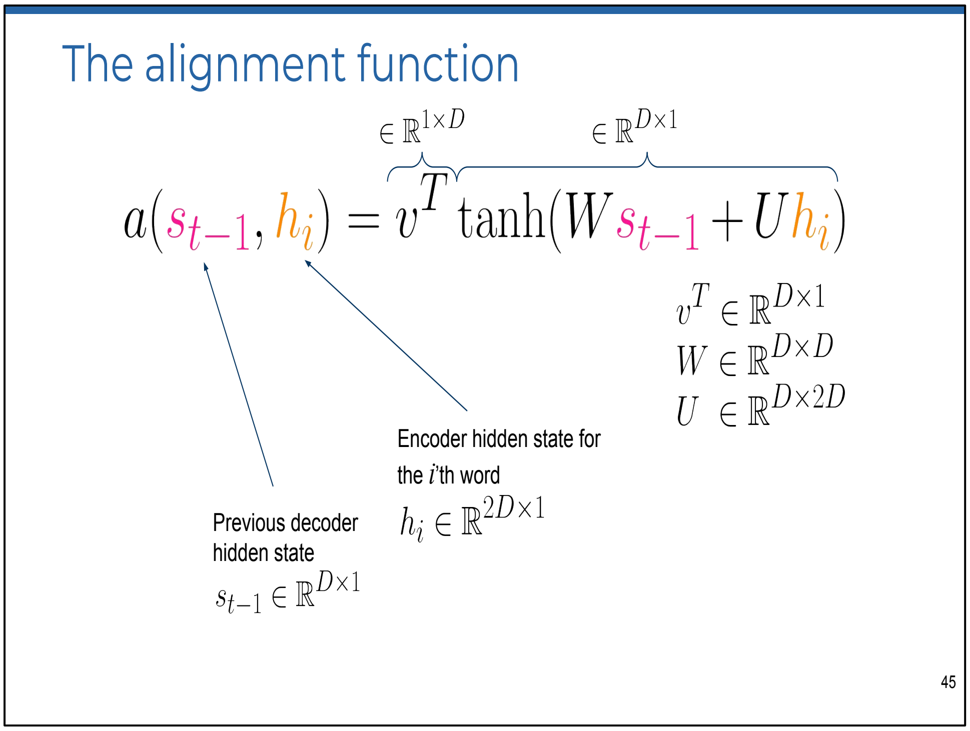
$h_i \in \mathbb{R}^{2D\times 1}$

Previous decoder hidden state

$s_{t-1} \in \mathbb{R}^{D\times 1}$

SPEAK SLOWLY

1. Now, we add the 2 vectors together. This is key - the addition means that the resulting vector now fuses information from both the the encoder hidden state and the decoder hidden state
2. Tanh is just a non-linearity we're applying to it. We could have used another activation function if we wanted to



The alignment function

$$a(s_{t-1}, h_i) = \overbrace{v^T}^{\in \mathbb{R}^{1\times D}} \overbrace{\tanh(W s_{t-1} + U h_i)}^{\in \mathbb{R}^{D\times 1}}$$

$v^T \in \mathbb{R}^{D\times 1}$
$W \in \mathbb{R}^{D\times D}$
$U \in \mathbb{R}^{D\times 2D}$

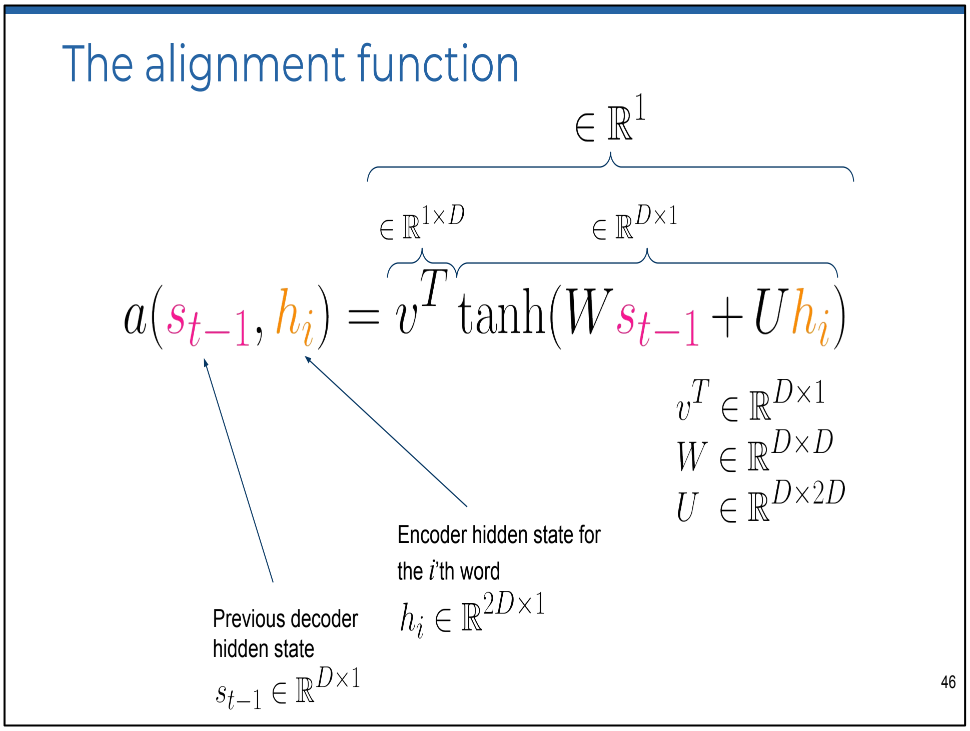
Encoder hidden state for the i'th word
$h_i \in \mathbb{R}^{2D\times 1}$

Previous decoder hidden state
$s_{t-1} \in \mathbb{R}^{D\times 1}$

45

SPEAK SLOWLY

1. v is responsible for creating the energy score



The alignment function

$$a(s_{t-1}, h_i) = \overbrace{\overbrace{v^T}^{\in \mathbb{R}^{1\times D}} \overbrace{\tanh(W s_{t-1} + U h_i)}^{\in \mathbb{R}^{D\times 1}}}^{\in \mathbb{R}^1}$$

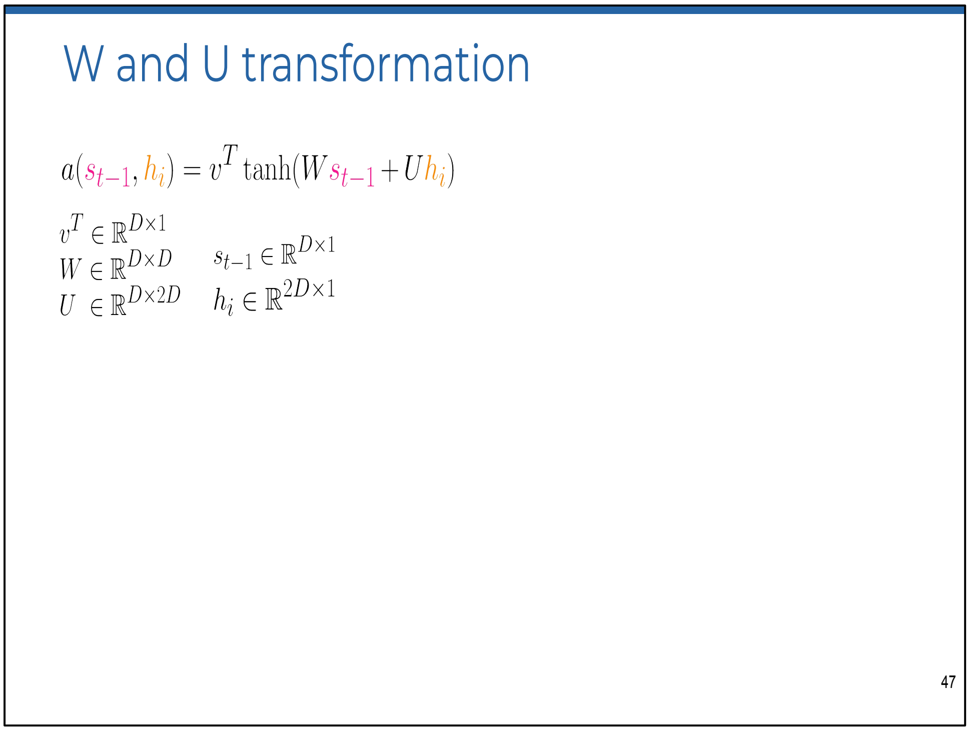
$v^T \in \mathbb{R}^{D\times 1}$
$W \in \mathbb{R}^{D\times D}$
$U \in \mathbb{R}^{D\times 2D}$

Encoder hidden state for the i'th word
$h_i \in \mathbb{R}^{2D\times 1}$

Previous decoder hidden state
$s_{t-1} \in \mathbb{R}^{D\times 1}$

SPEAK SLOWLY

1. It projects the D-dimensional vector down to a scalar
2. This scalar is e_i, the unnormalized energy score



W and U transformation

$$a(s_{t-1}, h_i) = v^T \tanh(W s_{t-1} + U h_i)$$

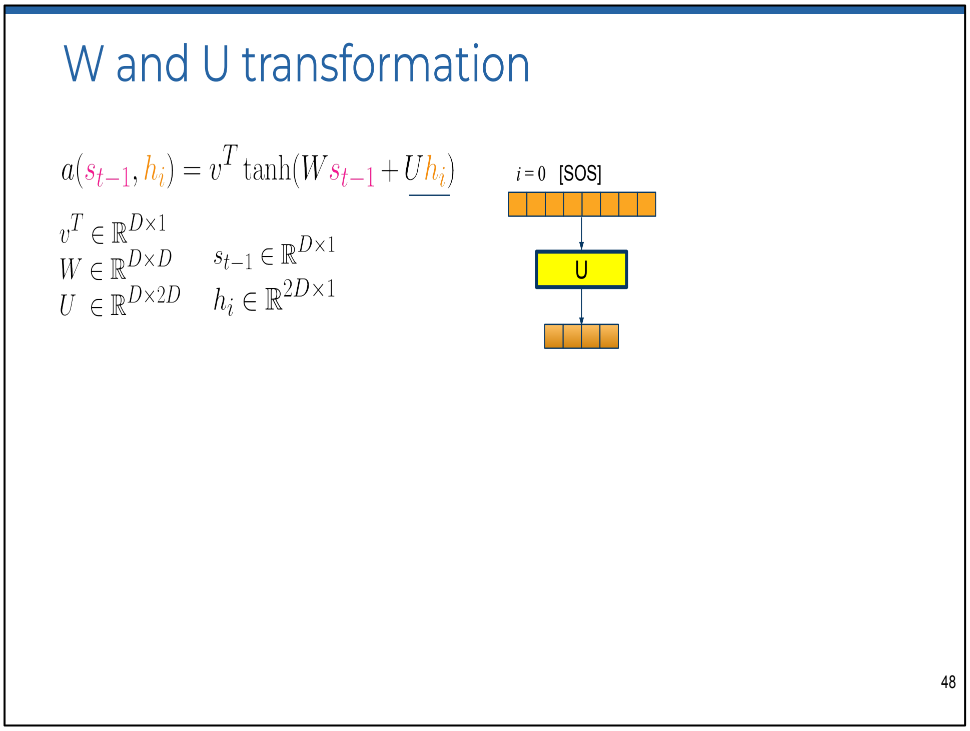
$v^T \in \mathbb{R}^{D\times 1}$

$W \in \mathbb{R}^{D\times D}$ $\quad s_{t-1} \in \mathbb{R}^{D\times 1}$

$U \in \mathbb{R}^{D\times 2D}$ $\quad h_i \in \mathbb{R}^{2D\times 1}$

SPEAK SLOWLY

1. Now that I've covered the maths, let look at it visually



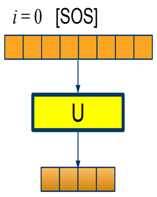
W and U transformation

$$a(s_{t-1}, h_i) = v^T \tanh(W s_{t-1} + \underline{U h_i})$$

$v^T \in \mathbb{R}^{D\times 1}$

$W \in \mathbb{R}^{D\times D}$ $s_{t-1} \in \mathbb{R}^{D\times 1}$

$U \in \mathbb{R}^{D\times 2D}$ $h_i \in \mathbb{R}^{2D\times 1}$



SPEAK SLOWLY

1. First, we project h_i

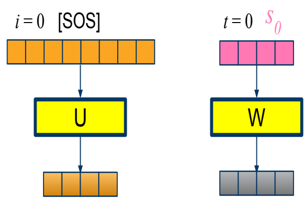
W and U transformation

$$a(s_{t-1}, h_i) = v^T \tanh(\underline{W s_{t-1}} + U h_i)$$

$v^T \in \mathbb{R}^{D \times 1}$

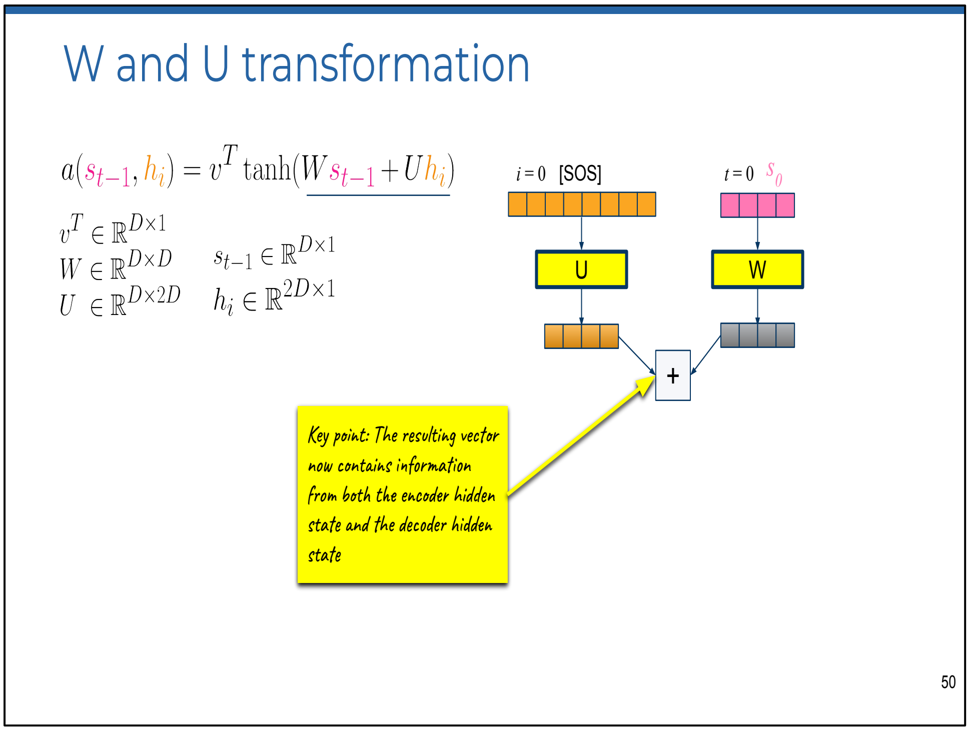
$W \in \mathbb{R}^{D \times D}$ $\quad s_{t-1} \in \mathbb{R}^{D \times 1}$

$U \in \mathbb{R}^{D \times 2D}$ $\quad h_i \in \mathbb{R}^{2D \times 1}$



SPEAK SLOWLY

1. And then we project s_t-1



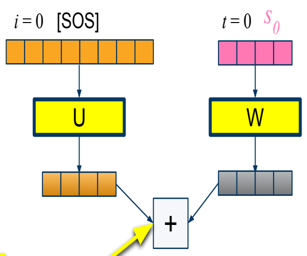
W and U transformation

$$a(s_{t-1}, h_i) = v^T \tanh(\underline{W s_{t-1} + U h_i})$$

$v^T \in \mathbb{R}^{D\times 1}$

$W \in \mathbb{R}^{D\times D}$ $\quad s_{t-1} \in \mathbb{R}^{D\times 1}$

$U \in \mathbb{R}^{D\times 2D}$ $\quad h_i \in \mathbb{R}^{2D\times 1}$

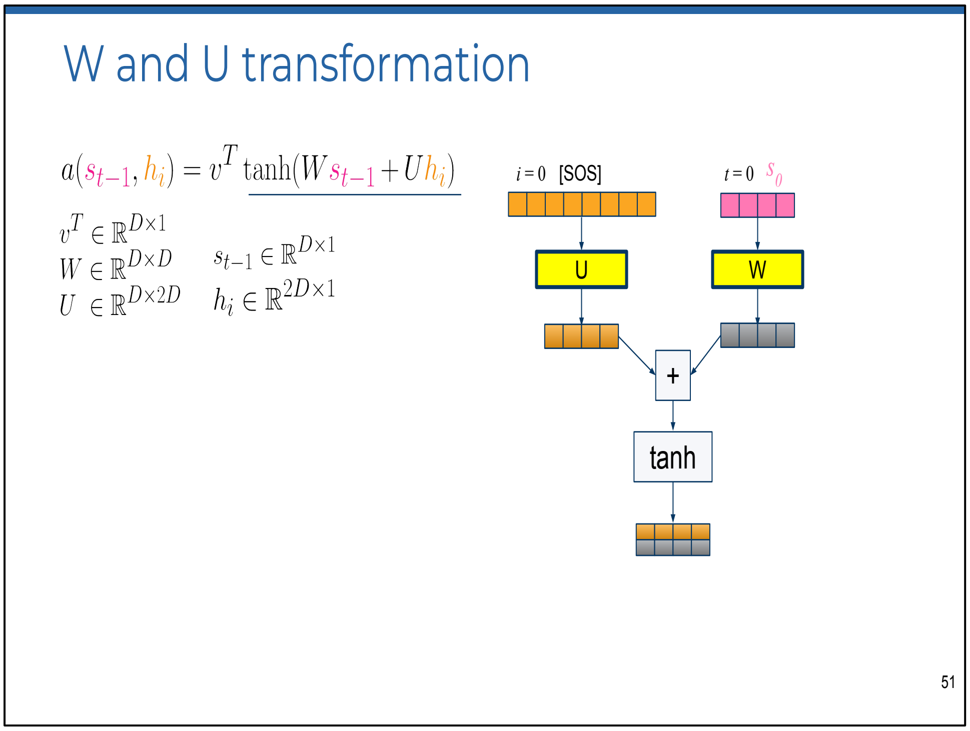


Key point: The resulting vector now contains information from both the encoder hidden state and the decoder hidden state

50

SPEAK SLOWLY

1. We then add the 2 together. The resulting vector creates a contextual representation



W and U transformation

$$a(s_{t-1}, h_i) = v^T \underline{\tanh(W s_{t-1} + U h_i)}$$

$v^T \in \mathbb{R}^{D\times1}$

$W \in \mathbb{R}^{D\times D}$ $\quad s_{t-1} \in \mathbb{R}^{D\times1}$

$U \in \mathbb{R}^{D\times 2D}$ $\quad h_i \in \mathbb{R}^{2D\times1}$

$i = 0$ [SOS]
$t = 0$ s_0
U
W
+
tanh

SPEAK SLOWLY

1. We apply the tanh non-linearity to the added vector

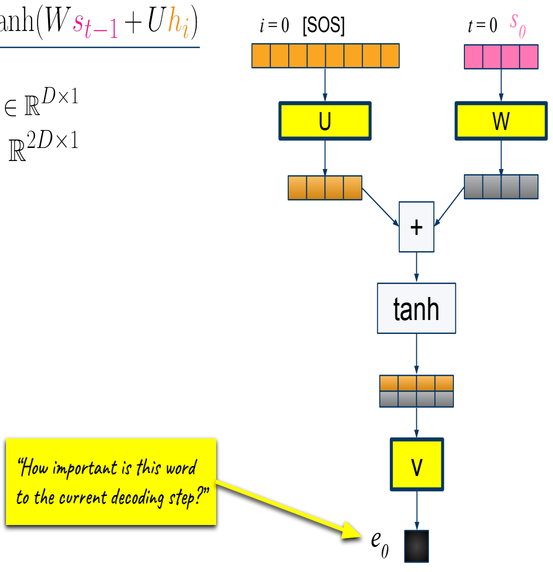
The energy score

$$a(s_{t-1}, h_i) = \underline{v^T \tanh(W s_{t-1} + U h_i)}$$

$v^T \in \mathbb{R}^{D\times 1}$

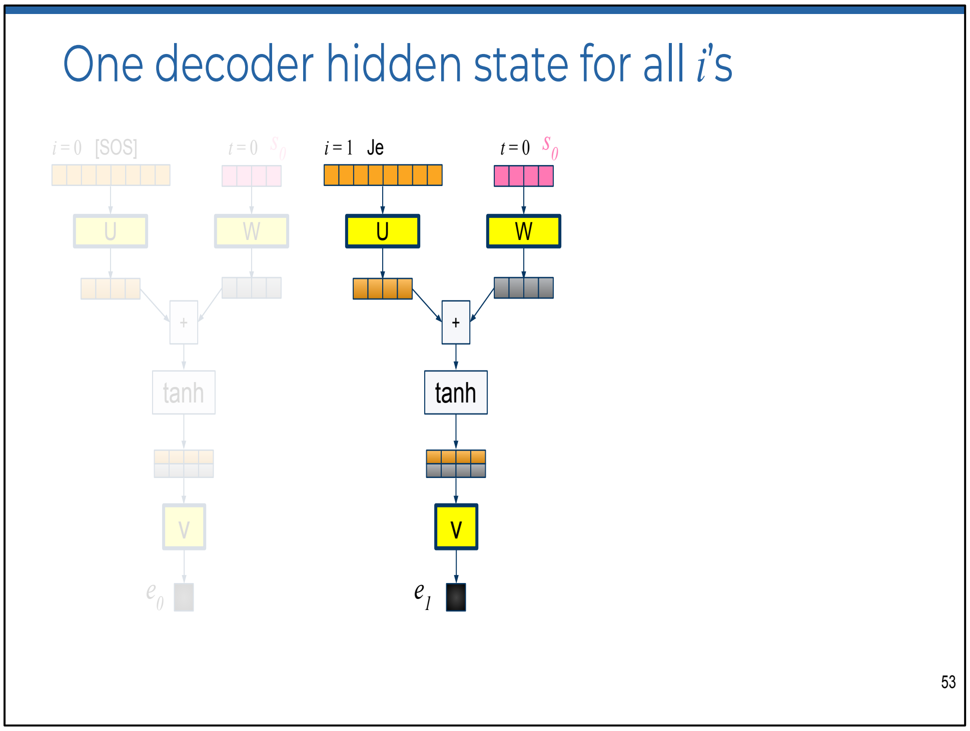
$W \in \mathbb{R}^{D\times D}$ $\quad s_{t-1} \in \mathbb{R}^{D\times 1}$

$U \in \mathbb{R}^{D\times 2D}$ $\quad h_i \in \mathbb{R}^{2D\times 1}$

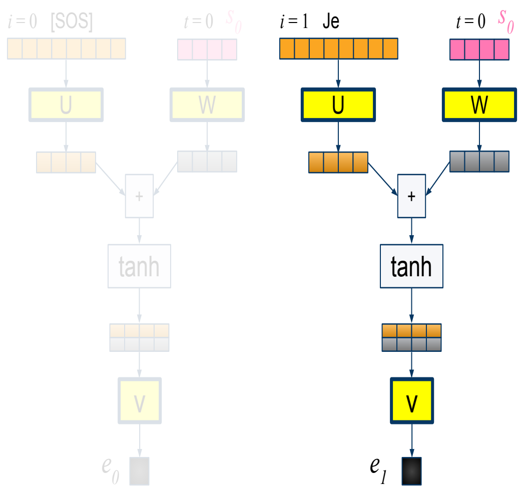


SPEAK SLOWLY

1. The v function now projects the added vector to a scalar value. This is our energy score
2. An intuitive interpretation of the energy score is "How important is this word to the current decoding step?"



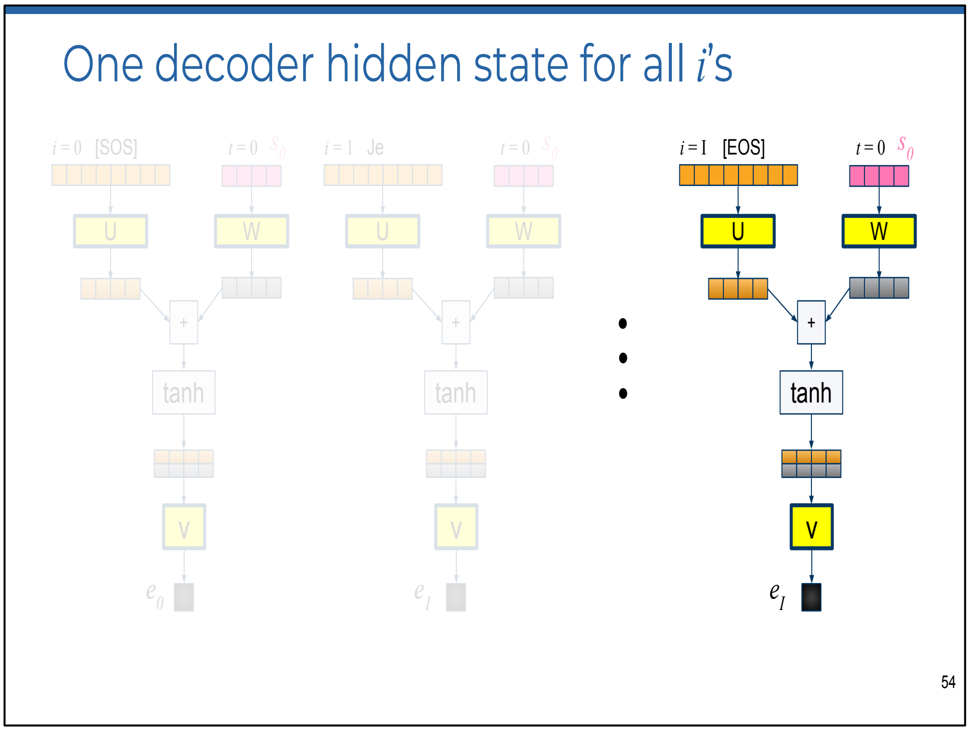
One decoder hidden state for all *i*'s



53

SPEAK SLOWLY

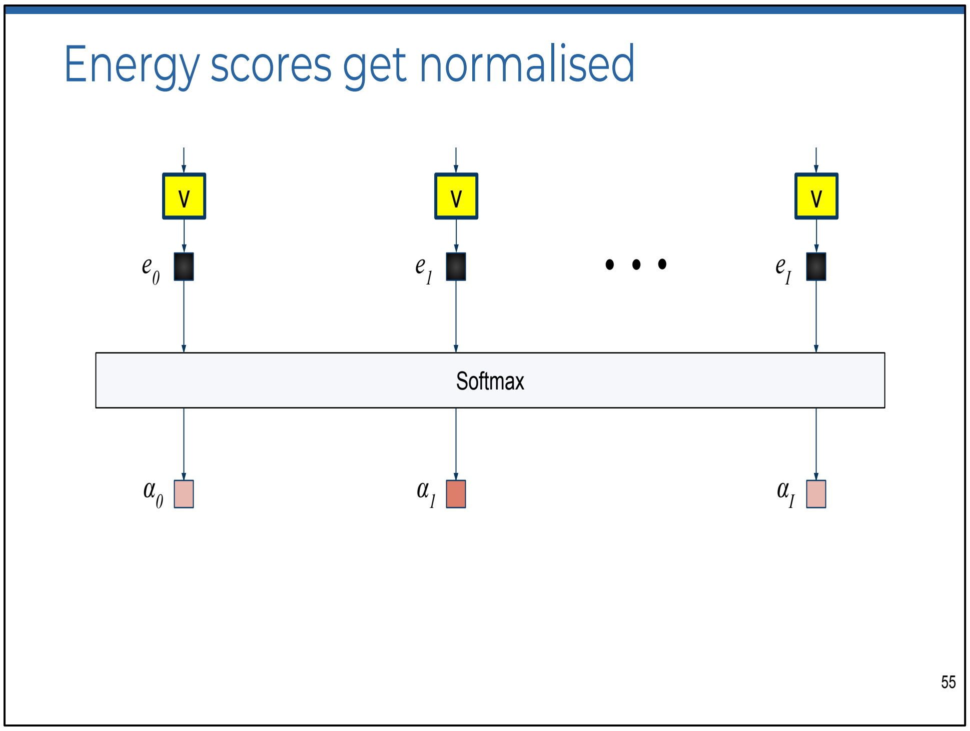
1. The same process for the next word



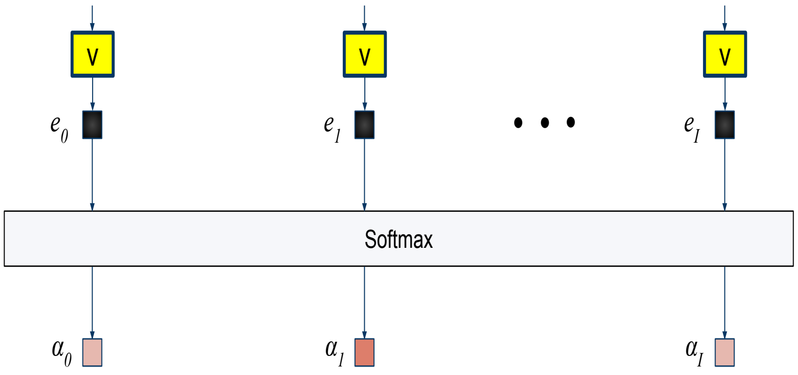
One decoder hidden state for all *i*'s

SPEAK SLOWLY

1. Repeat for all h_i's

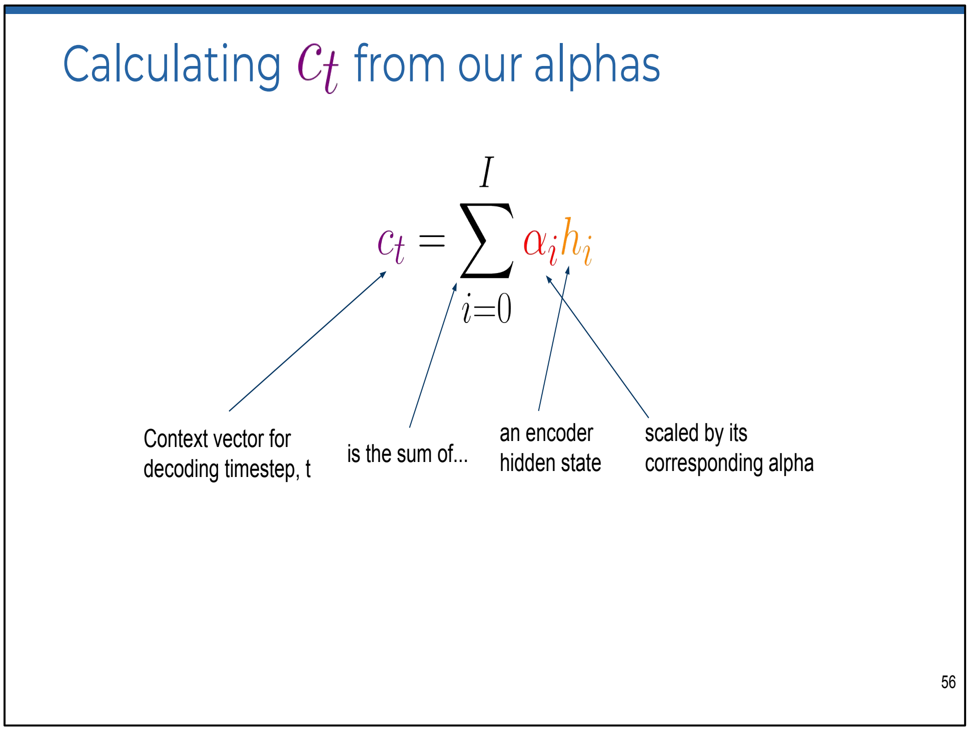


Energy scores get normalised



SPEAK SLOWLY

1. We now have all our energy scores, but they are currently unnormalized
2. We normalize them by sending all the energy scores to a softmax function.
3. This gives us alpha values between 0 and 1



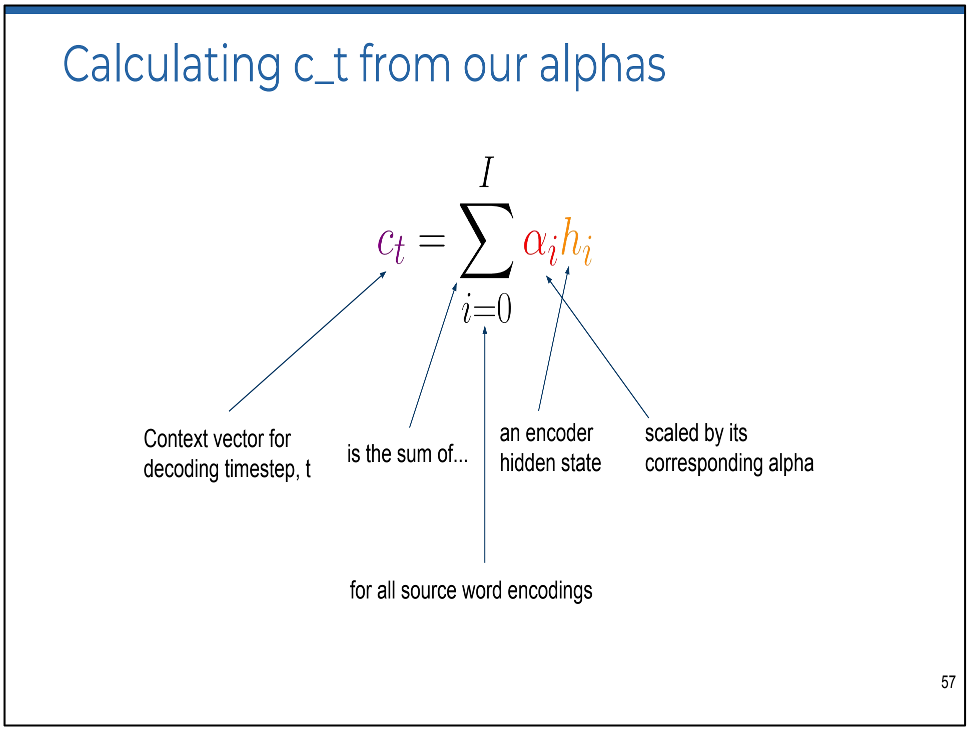
Calculating c_t from our alphas

$$c_t = \sum_{i=0}^{I} \alpha_i h_i$$

Context vector for decoding timestep, t
is the sum of...
an encoder hidden state
scaled by its corresponding alpha

SPEAK SLOWLY

1. Revisiting the original attention formula:



Calculating c_t from our alphas

$$c_t = \sum_{i=0}^{I} \alpha_i h_i$$

Context vector for decoding timestep, t

is the sum of...

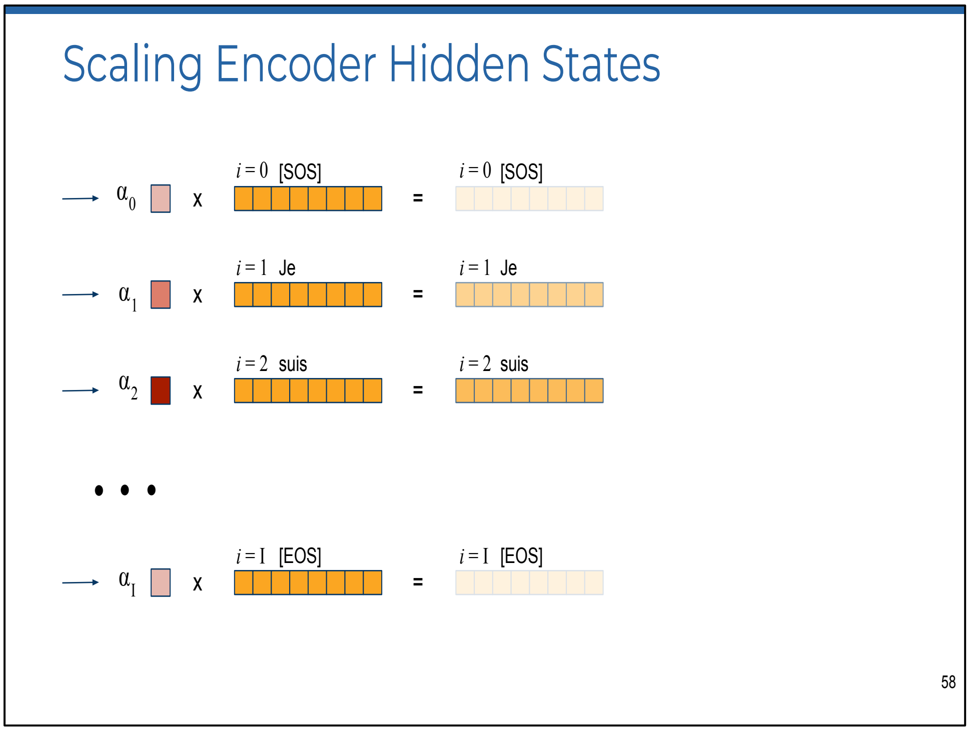
an encoder hidden state

scaled by its corresponding alpha

for all source word encodings

57

SPEAK SLOWLY



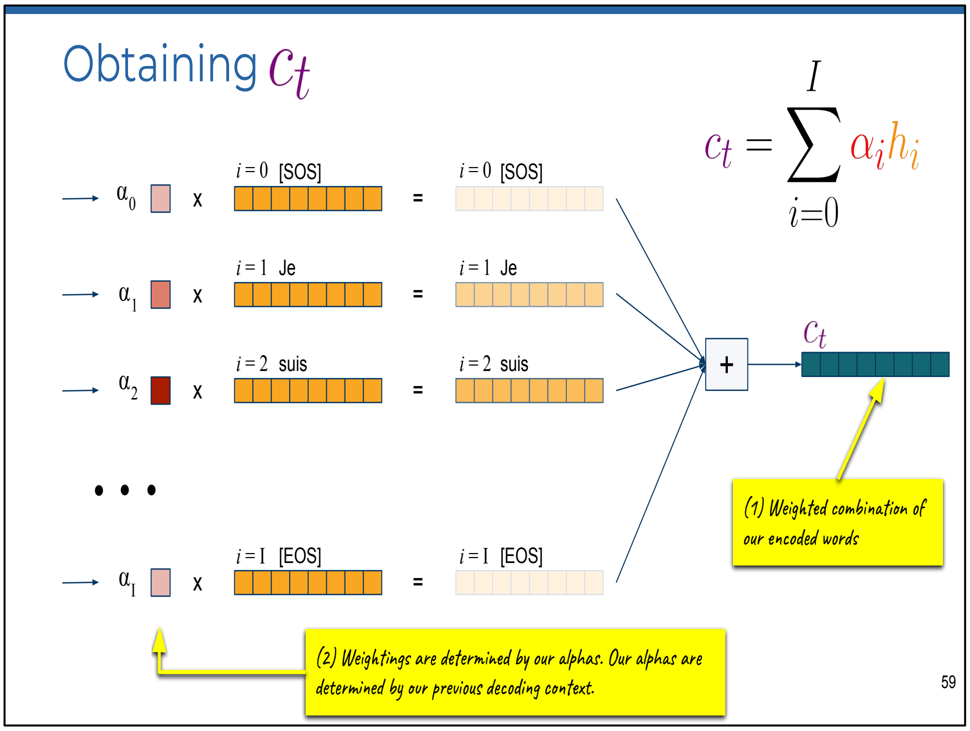
Scaling Encoder Hidden States

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]

58

SPEAK SLOWLY

1. Now we weight our encoded words with our obtained alphas



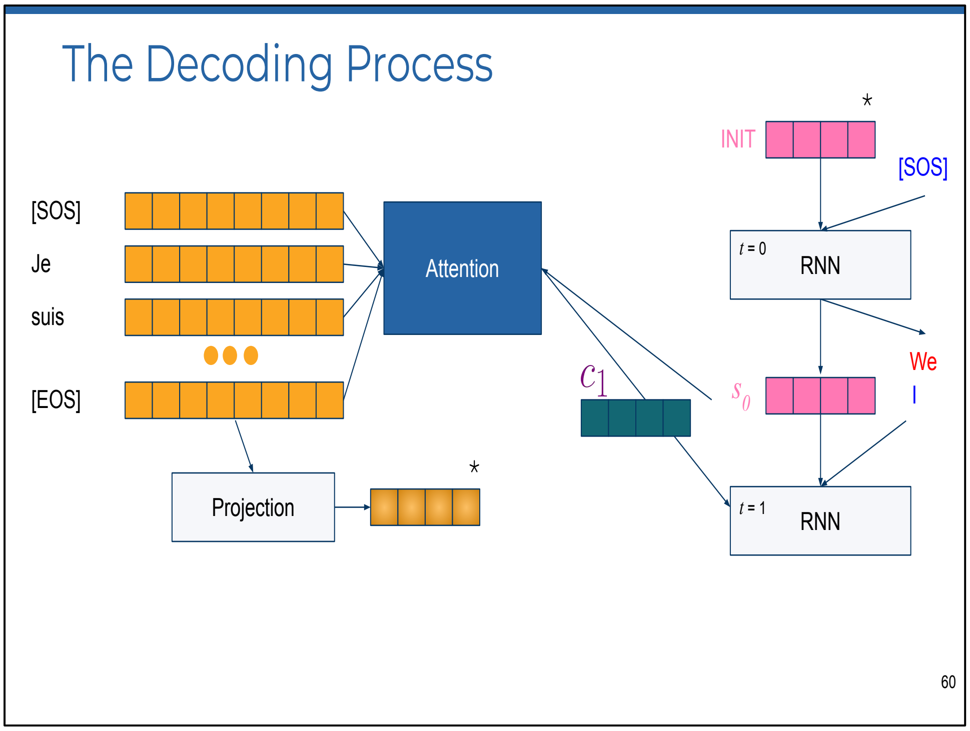
Obtaining c_t

$$c_t = \sum_{i=0}^{I} a_i h_i$$

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

SPEAK SLOWLY

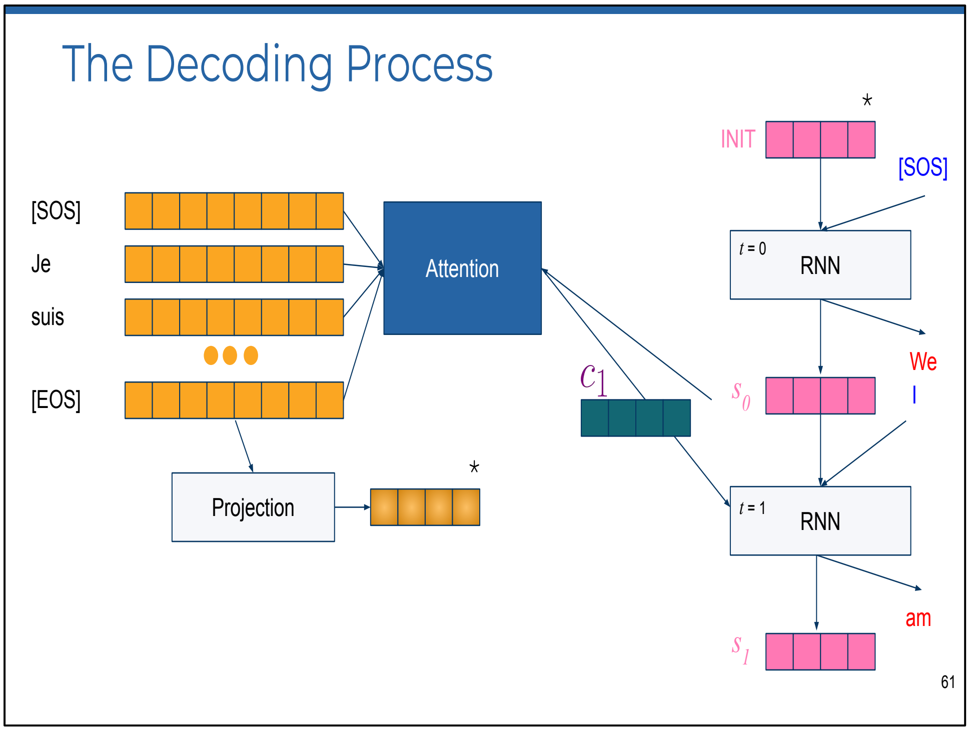
1. And add them all together to obtain c_t - our context vector



The Decoding Process

SPEAK SLOWLY

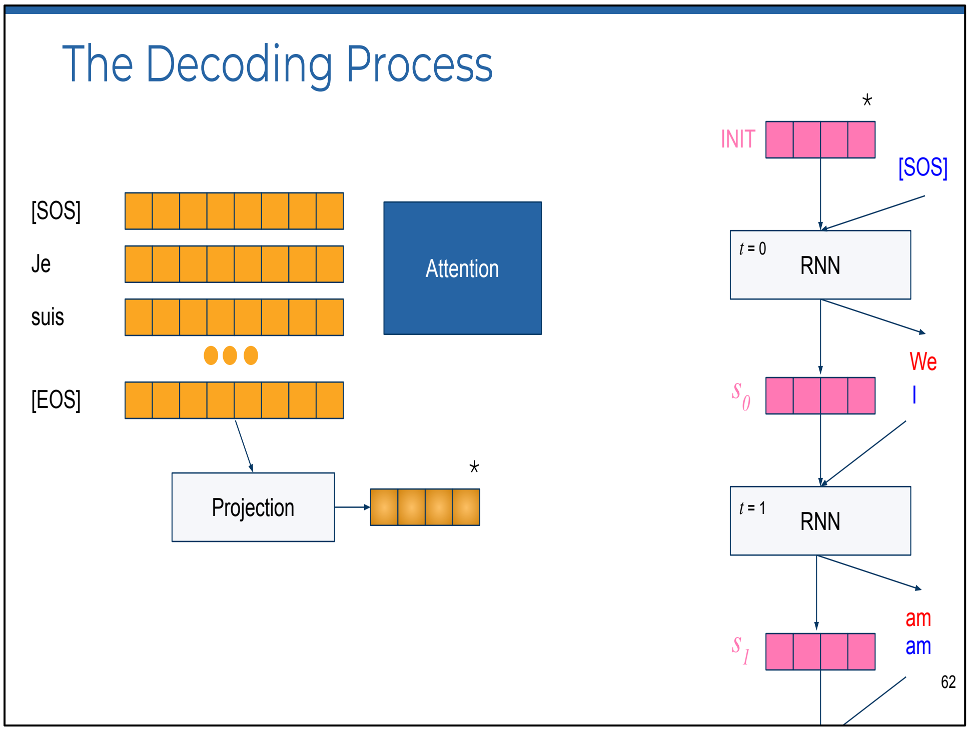
1. Continuing from where we were in our diagram, we then output the context vector for the t=1 word (c1)
2. This forms our 3rd input to the current RNN step (alongside s_0 and an input word)



The Decoding Process

SPEAK SLOWLY

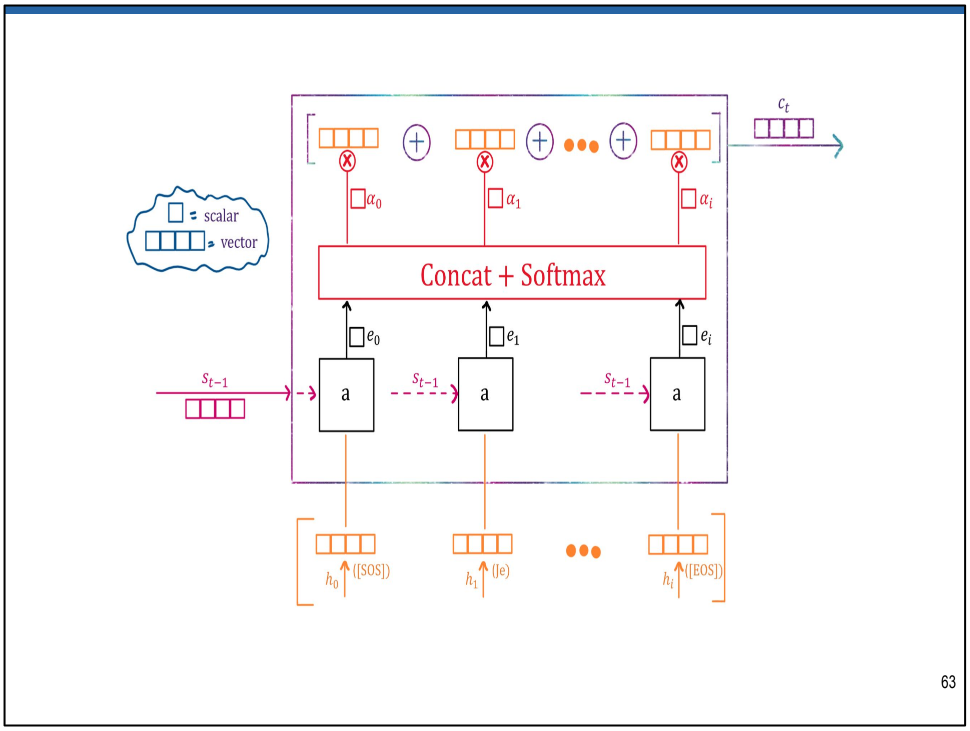
1. The RNN uses these 3 inputs to output decoder hidden state s_1, and a prediction (am)



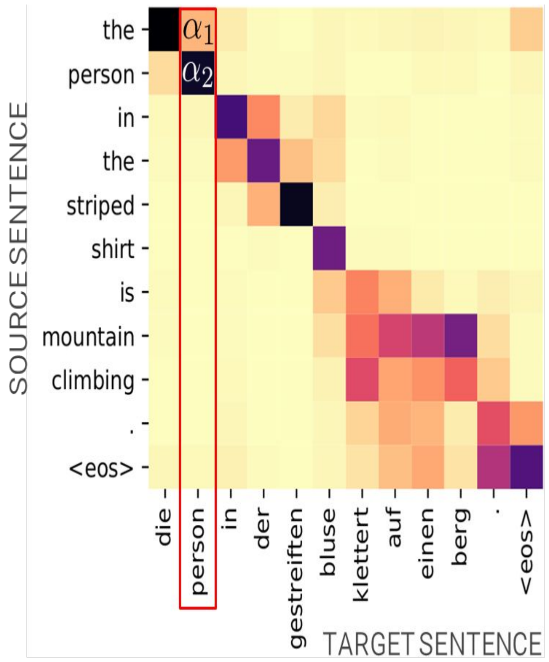
The Decoding Process

SPEAK SLOWLY

1. And the process repeats until we hit our max len



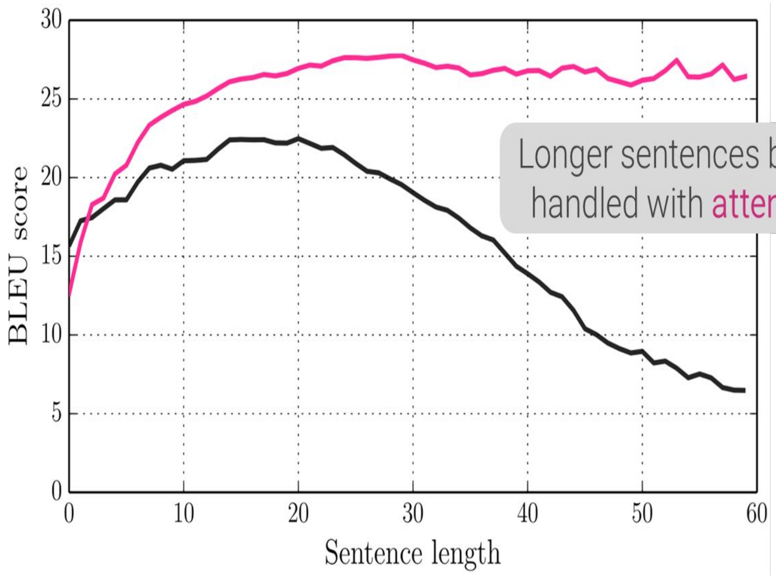
1. Tell them: alpha is a scalar value. h_i is 2d
2. Alpha represents how important the i'th source word is to the current decoding step.
3. To obtain this, we need to calculate the energy scores for each word (e_i).
4. Energy scores are calculated by using an alignment function: a.
5. Once we have energy scores, we concatenate them together. Then we apply a softmax. The softmax is the normalized relevance of each source word with respect to the current decoding step.
6. Then we perform the alpha*h_i multiplication: This is a keypoint of attention. It applies a "mask" to each of our hidden states. Low energy values tend to 0 which means that we do not need that word's information to decode the next word
7. Think of the alignment function as a 2 layered MLP. The first layer combines our decoder hidden state with all our encoder hidden states. The second layer (v) uses this contextualised representation to predict energy scores: i.e. unnormalized "importance" of each of the source words.
8. Ask them to clarify shapes for: st-1 and h. And the shape output of the tanh function.
    a. Mention that we have *i* encoder hidden states, but only 1 decoder hidden state. How can we perform addition in this case?



SPEAK SLOWLY

- 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".

It works 😱😱

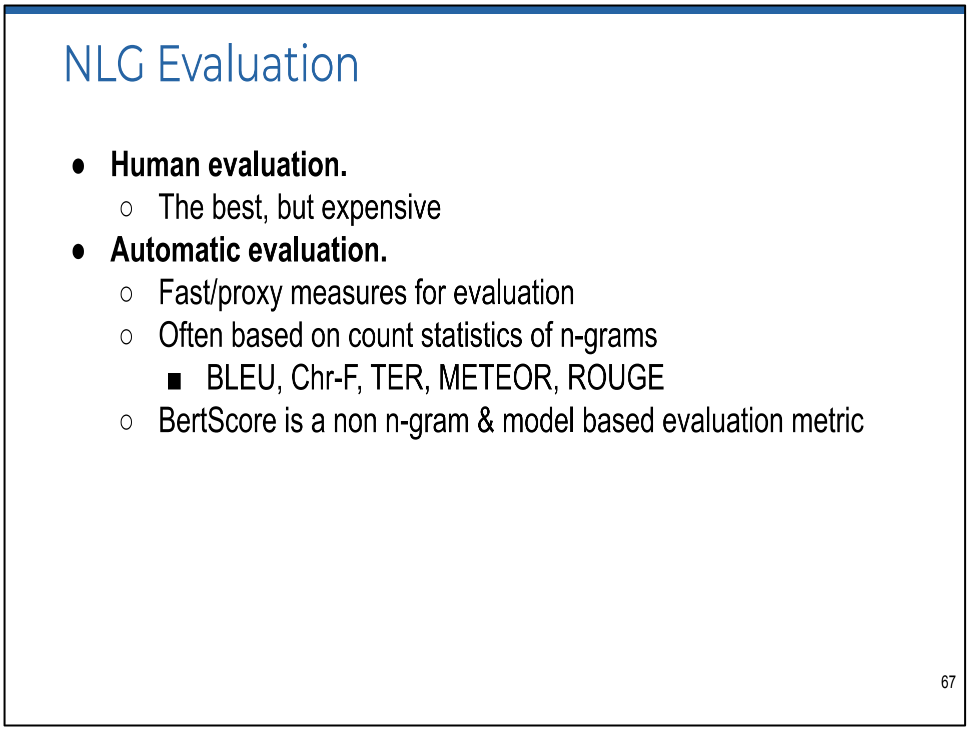


Longer sentences better handled with attention

SPEAK SLOWLY

Evaluation metrics of MT/NLG systems

Running ETA: 62



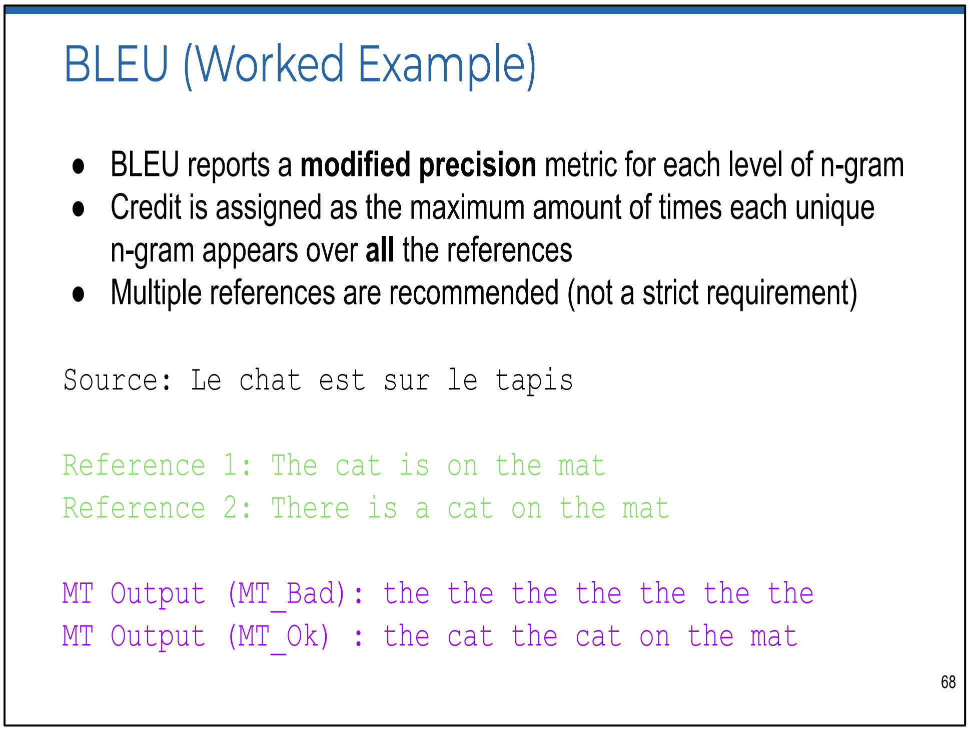
NLG Evaluation

- **Human evaluation.**
  - The best, but expensive
- **Automatic evaluation.**
  - Fast/proxy measures for evaluation
  - Often based on count statistics of n-grams
    - BLEU, Chr-F, TER, METEOR, ROUGE
  - BertScore is a non n-gram & model based evaluation metric

67

SPEAK SLOWLY

- Mention that BertScore is not count based
- Mention that it’s still an active area of research. Heard about BLONDE at a conference last year



BLEU (Worked Example)

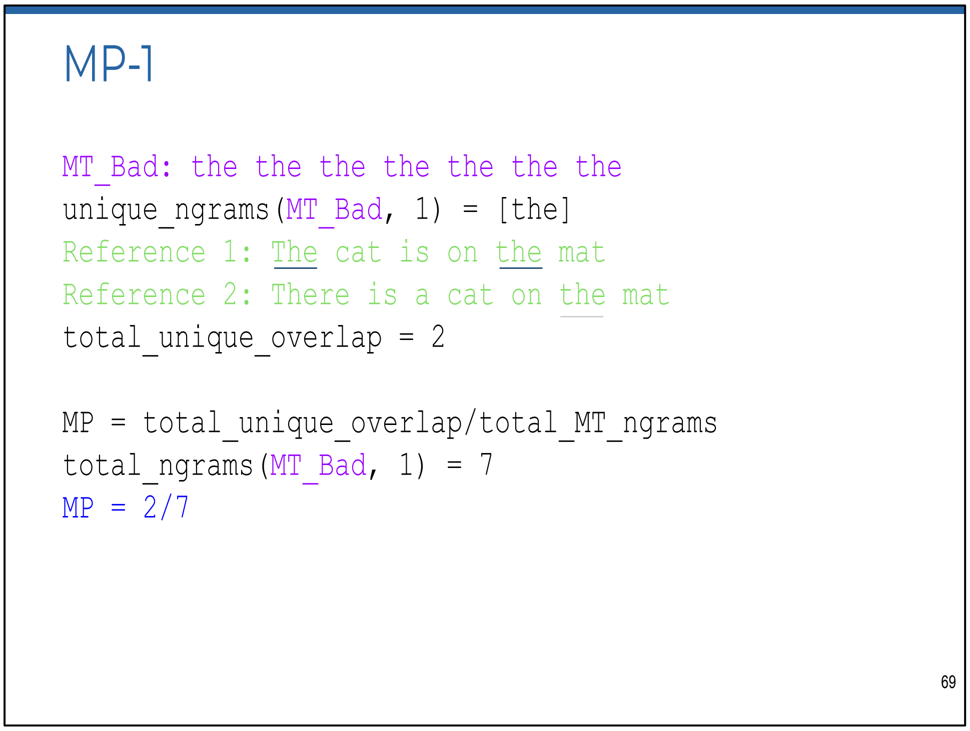
- BLEU reports a **modified precision** metric for each level of n-gram
- Credit is assigned as the maximum amount of times each unique n-gram appears over **all** the references
- Multiple references are recommended (not a strict requirement)

```
Source: Le chat est sur le tapis

Reference 1: The cat is on the mat
Reference 2: There is a cat on the mat

MT Output (MT_Bad): the the the the the the the
MT Output (MT_Ok) : the cat the cat on the mat
```

68

SPEAK SLOWLY

- We'll look through the formal definition in a minute, but with BLEU being the most popular MT evaluation metric, I thought it worthwhile to run through a worked example



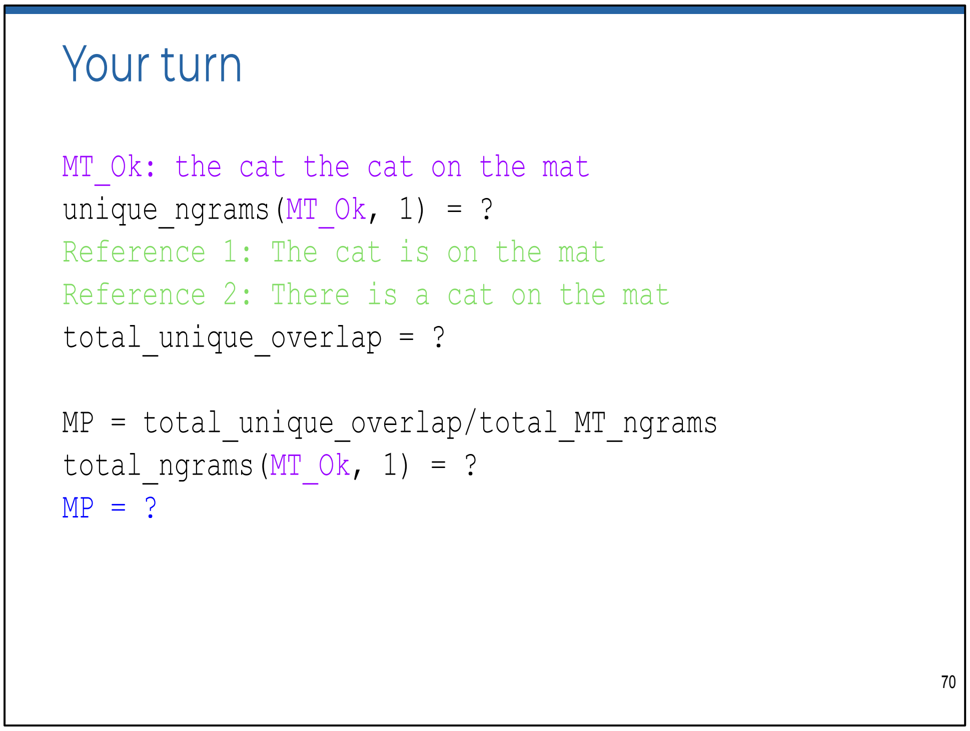
MP-1

```
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
```

SPEAK SLOWLY

- MP stands for modified precision. MP is not an acronym in the literature, but just something I came up with to explain the intermediate steps of calculating the BLEU score

1. First we work out the number of unique ngrams in our Hyp. MP-1 means we're looking at unigrams (MP-2 would be bigrams, etc)
2. Then we look at the ngram overlap between the unique ngrams in our Hyp and *all* ngrams in our Refs.
3. Work out the number of unique overlaps between the unique ngrams and all the Refs
4. Then we obtain our modified-precision scores: the total unique overlap divided by the number of ngrams in our Hyp



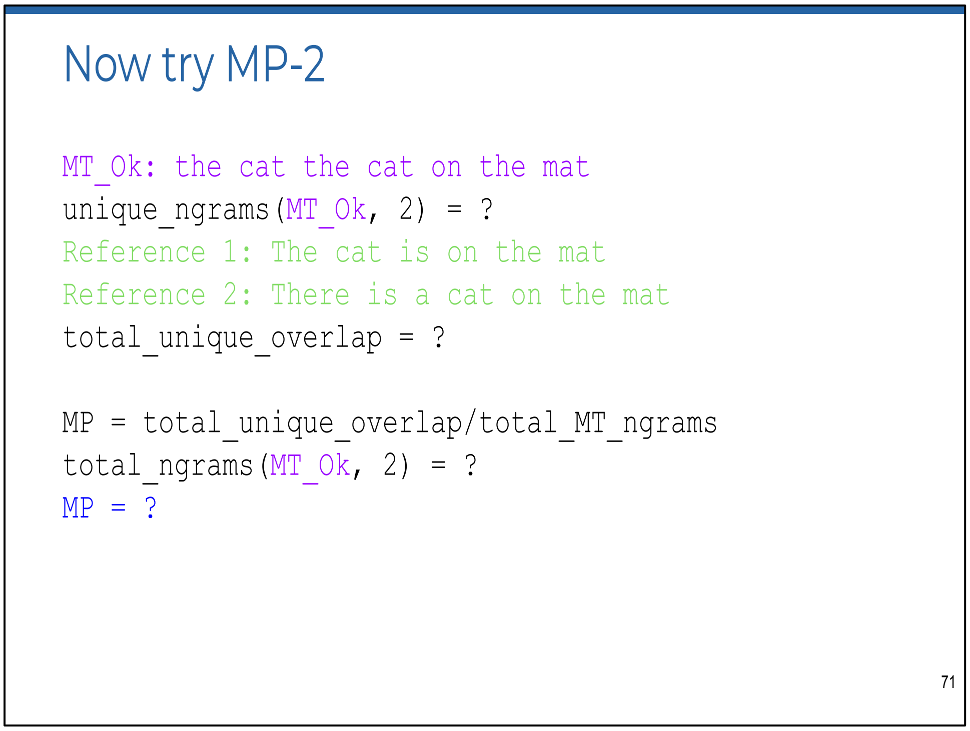
Your turn

```
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 = ?
```

70

SPEAK SLOWLY

- Work out the BLEU-1 score for MT_Ok

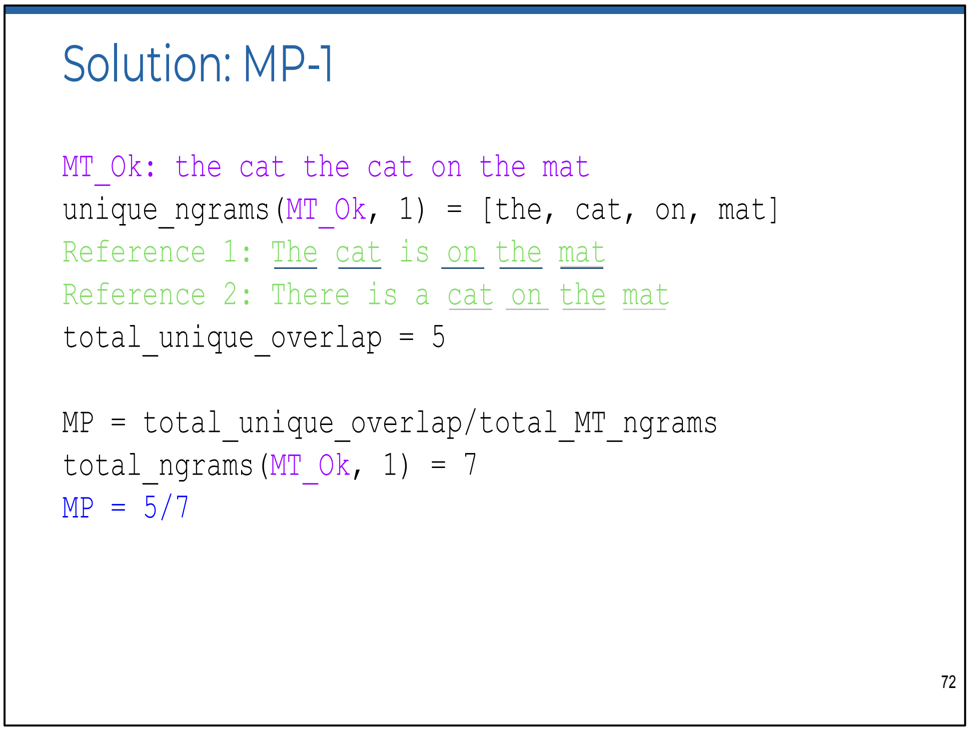


Now try MP-2

```
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 = ?
```

SPEAK SLOWLY

- Work out the BLEU-1 score for MT_Ok



Solution: MP-1

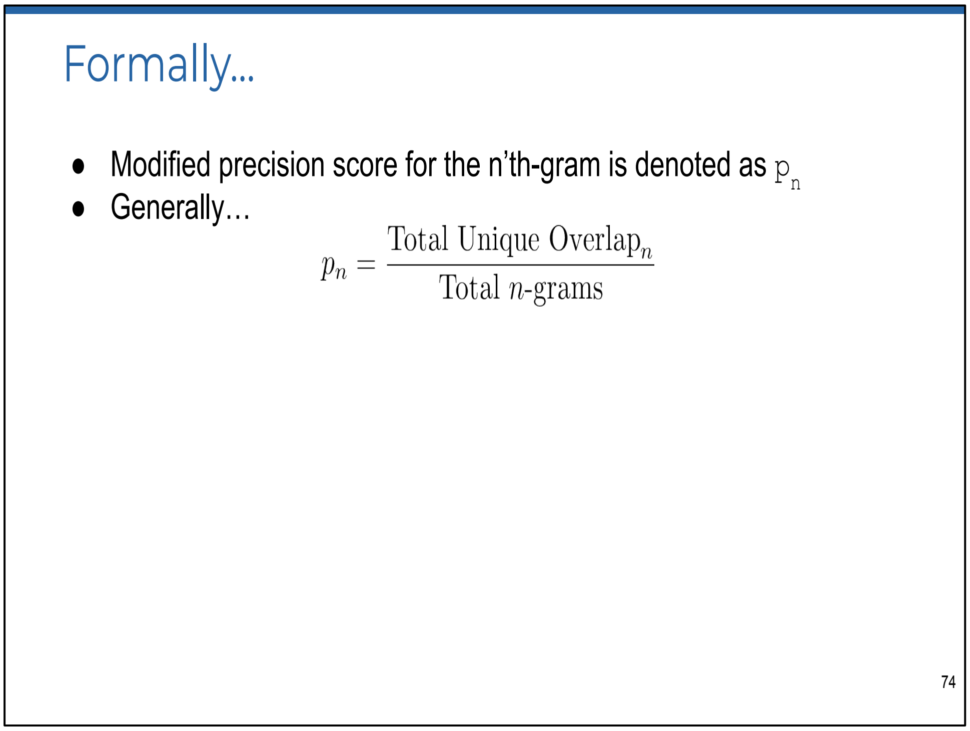
```
MT_Ok: the cat the cat on the mat
unique_ngrams(MT_Ok, 1) = [the, cat, on, mat]
Reference 1: The cat is on the mat
Reference 2: There is a cat on the mat
total_unique_overlap = 5

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

SPEAK SLOWLY

Solution: MP-2

```
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
```

SPEAK SLOWLY



Formally...

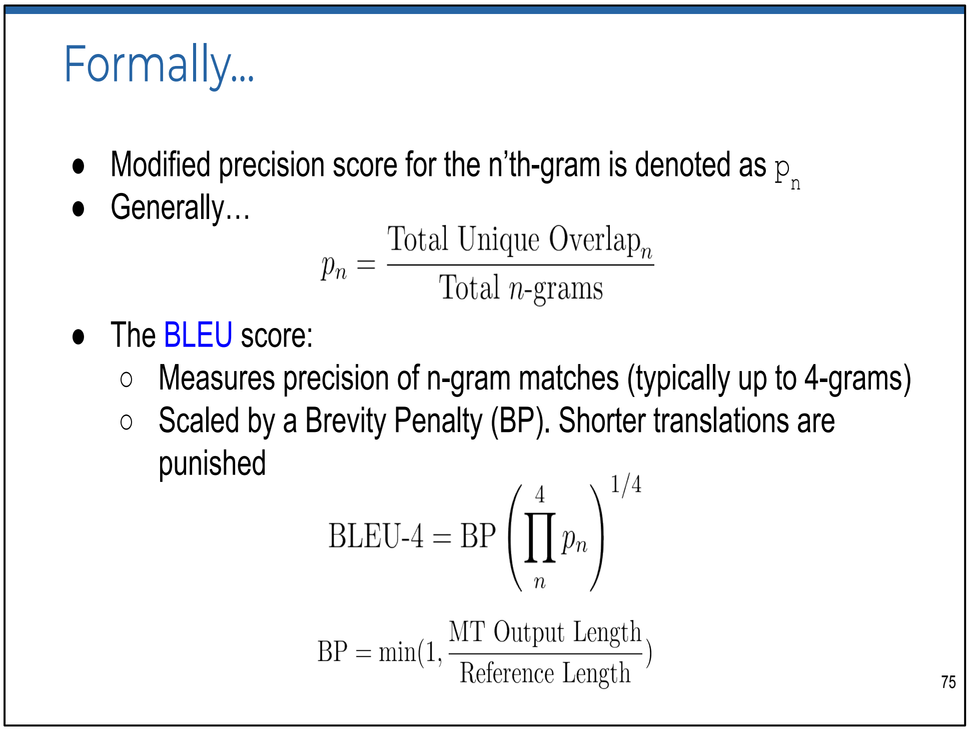
- Modified precision score for the n'th-gram is denoted as p_n
- Generally...

$$p_n = \frac{\text{Total Unique Overlap}_n}{\text{Total } n\text{-grams}}$$

74

SPEAK SLOWLY

- E.g. For unigrams, p1. For bigrams, p2



Formally…

- Modified precision score for the n'th-gram is denoted as p_n
- Generally…

$$p_n = \frac{\text{Total Unique Overlap}_n}{\text{Total } n\text{-grams}}$$

- The BLEU score:
  - Measures precision of n-gram matches (typically up to 4-grams)
  - Scaled by a Brevity Penalty (BP). Shorter translations are punished

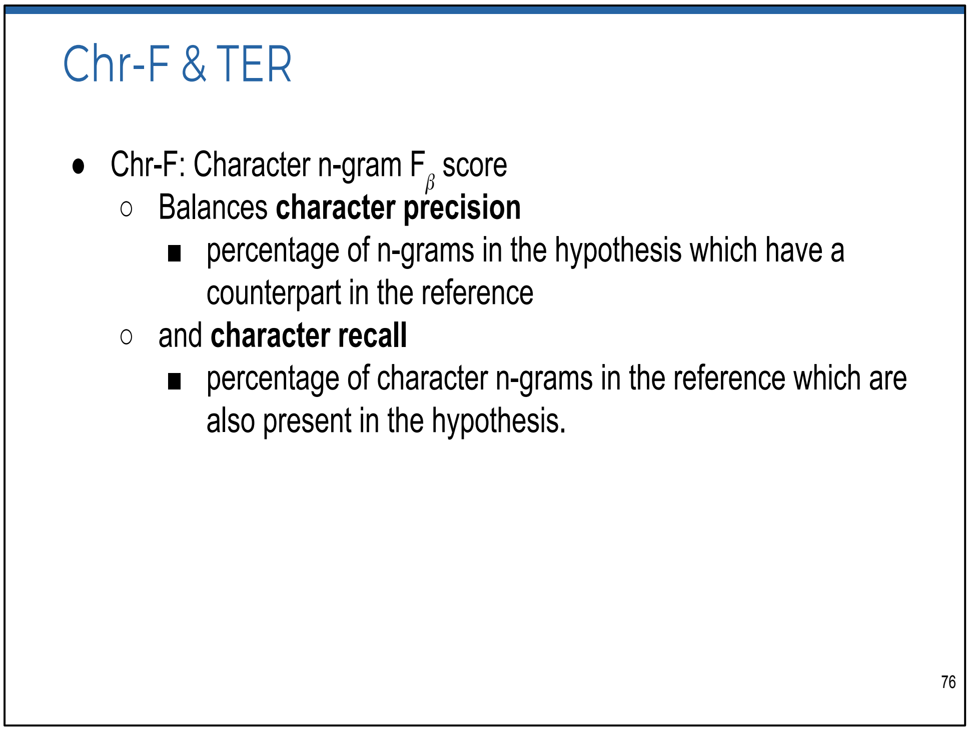
$$\text{BLEU-4} = \text{BP}\left(\prod_{n}^{4} p_n\right)^{1/4}$$

$$\text{BP} = \min(1, \frac{\text{MT Output Length}}{\text{Reference Length}})$$

75

SPEAK SLOWLY

- E.g. For unigrams, p1. For bigrams, p2
- Definition of BLEU:
  - Is a precision based metric over the product of n-gram matches
  - The matches are scaled by the brevity penalty which penalises shorter translations.
  - There are a couple of interpretations about why we use a BP. They're mostly about encouraging the Hyps to be of a similar length to a reference (see BP equation). Feel free to research more about it in your own time. An intuitive reason is for its existence is to account for the lack of recall term in the metric.
- Practically, there are some differing definitions and implementations of BLEU. When you want to report this score, it is good practise to use a standardized library (e.g. SacreBLEU)

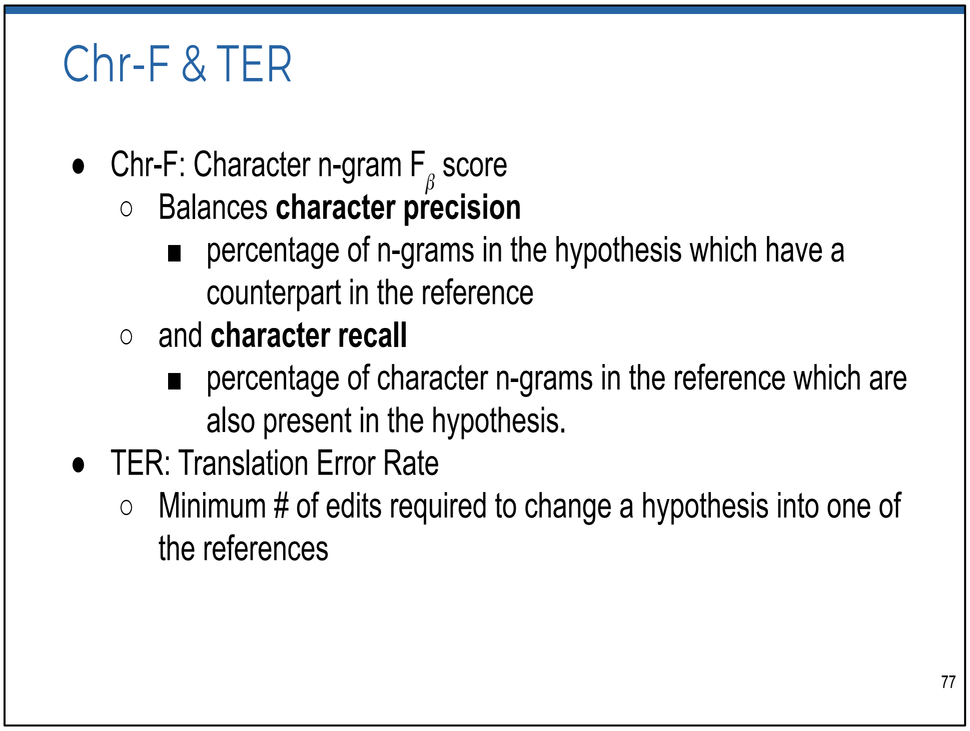


Chr-F & TER

- Chr-F: Character n-gram F_β score
  - Balances **character precision**
    - percentage of n-grams in the hypothesis which have a counterpart in the reference
  - and **character recall**
    - percentage of character n-grams in the reference which are also present in the hypothesis.

SPEAK SLOWLY

- Chr-F is an F beta -score based metric over character n-grams. This metric balances character precision and character recall

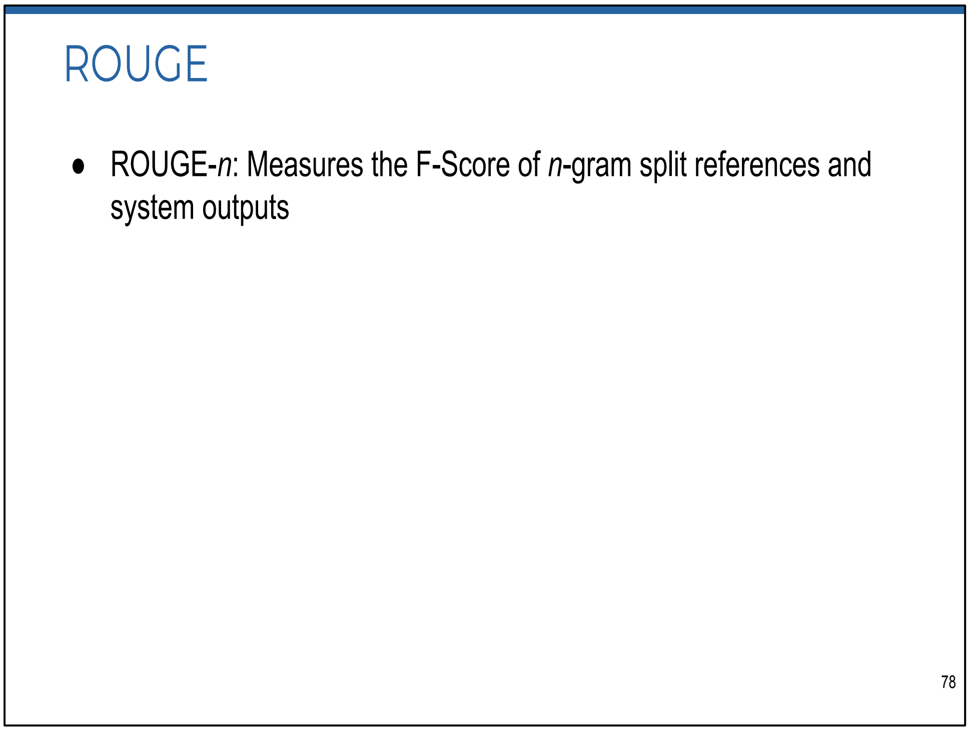


Chr-F & TER

- Chr-F: Character n-gram F_β score
  - Balances **character precision**
    - percentage of n-grams in the hypothesis which have a counterpart in the reference
  - and **character recall**
    - percentage of character n-grams in the reference which are also present in the hypothesis.
- TER: Translation Error Rate
  - Minimum # of edits required to change a hypothesis into one of the references

SPEAK SLOWLY

- 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



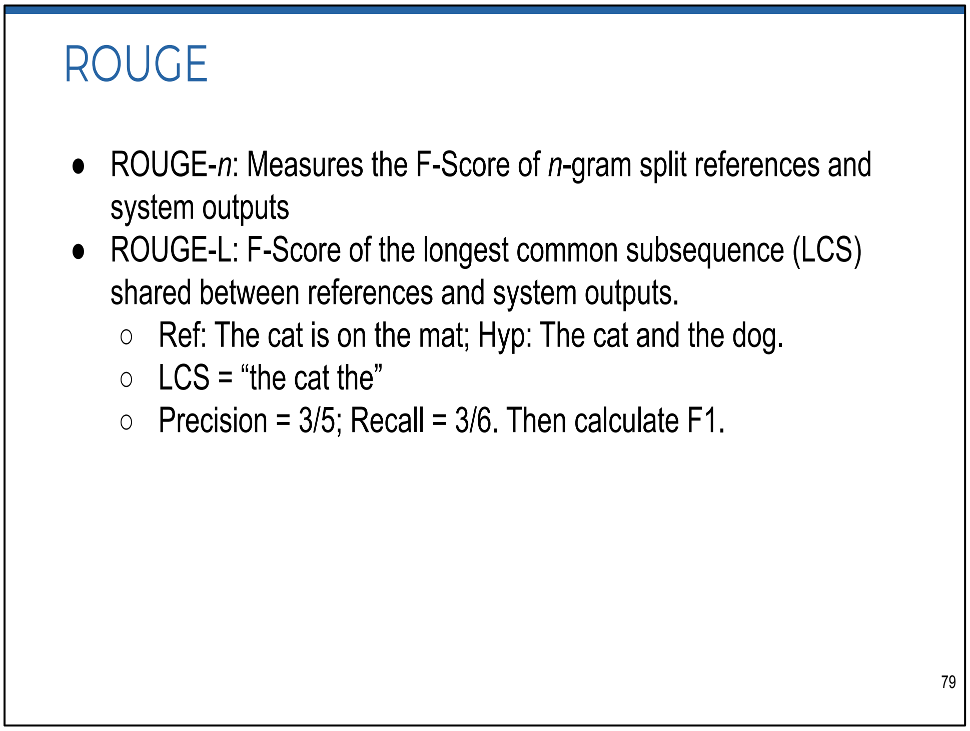
ROUGE

- ROUGE-*n*: Measures the F-Score of *n*-gram split references and system outputs

78

SPEAK SLOWLY

- As implied in a previous slide, a shortcoming of BLEU is that it focuses a lot on the precision between Hyp and Ref, but not the recall.
-
- 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

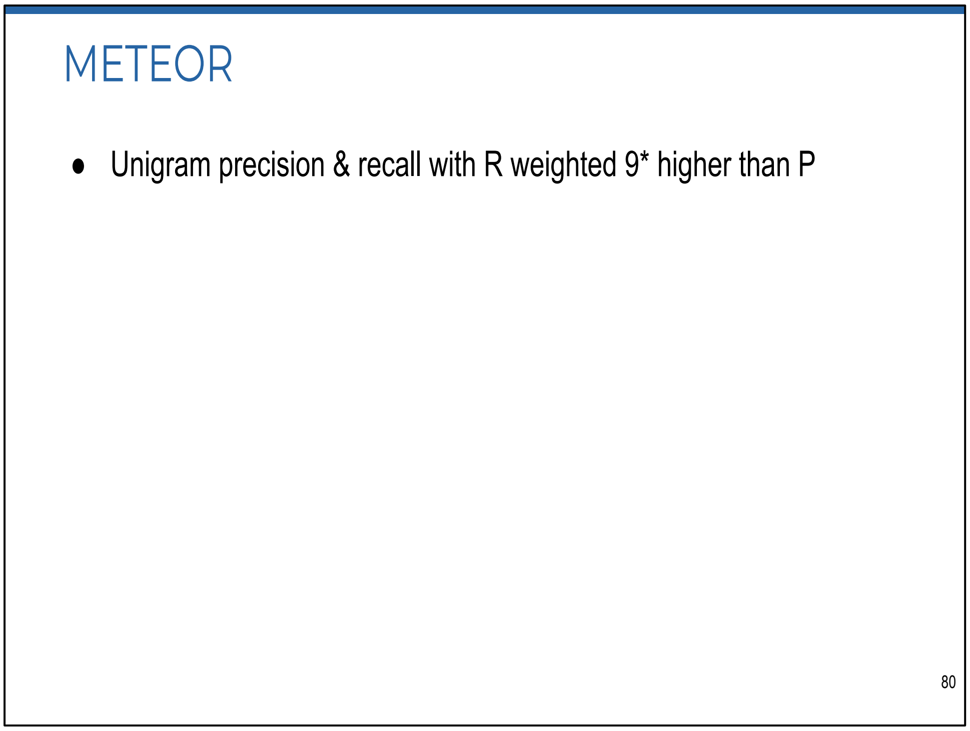


ROUGE

- 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.

SPEAK SLOWLY

- 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



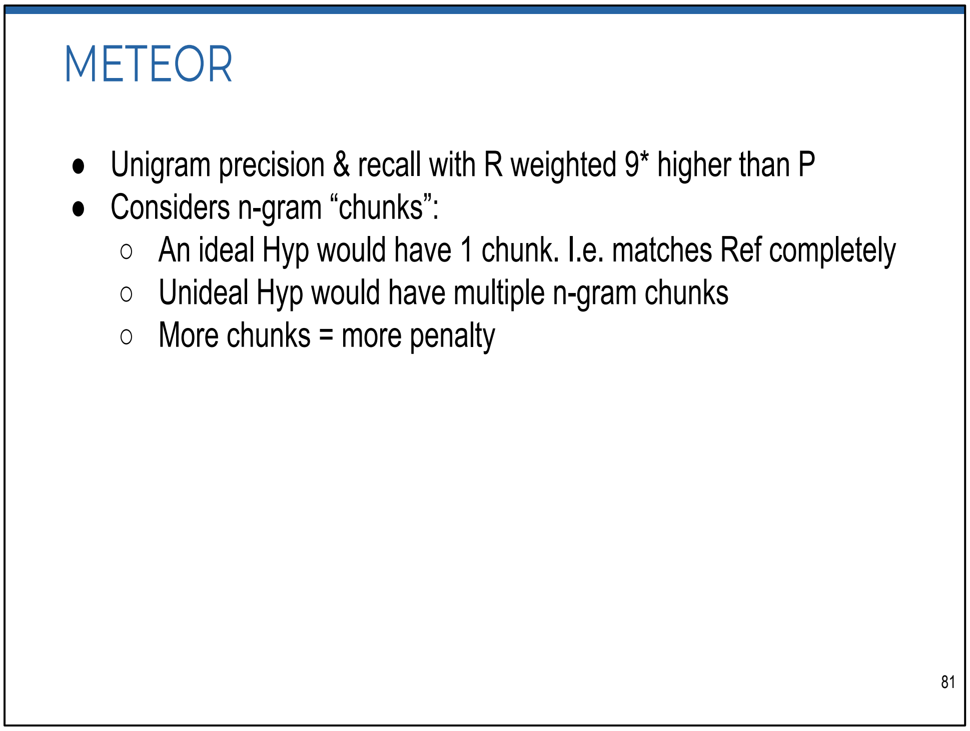
METEOR

- Unigram precision & recall with R weighted 9* higher than P

80

SPEAK SLOWLY

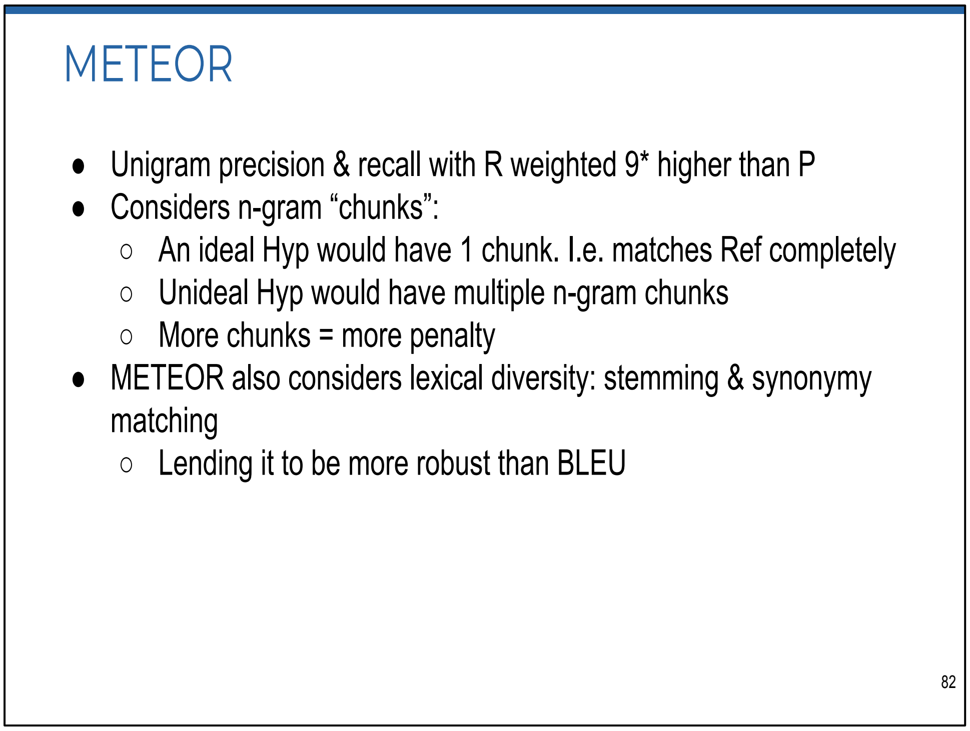
- METEOR is more modern and a better metric than BLEU, though for some reason the NMT community still adopts BLEU a lot more frequently. You will find METEOR in other generation tasks such as summarization and captioning



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

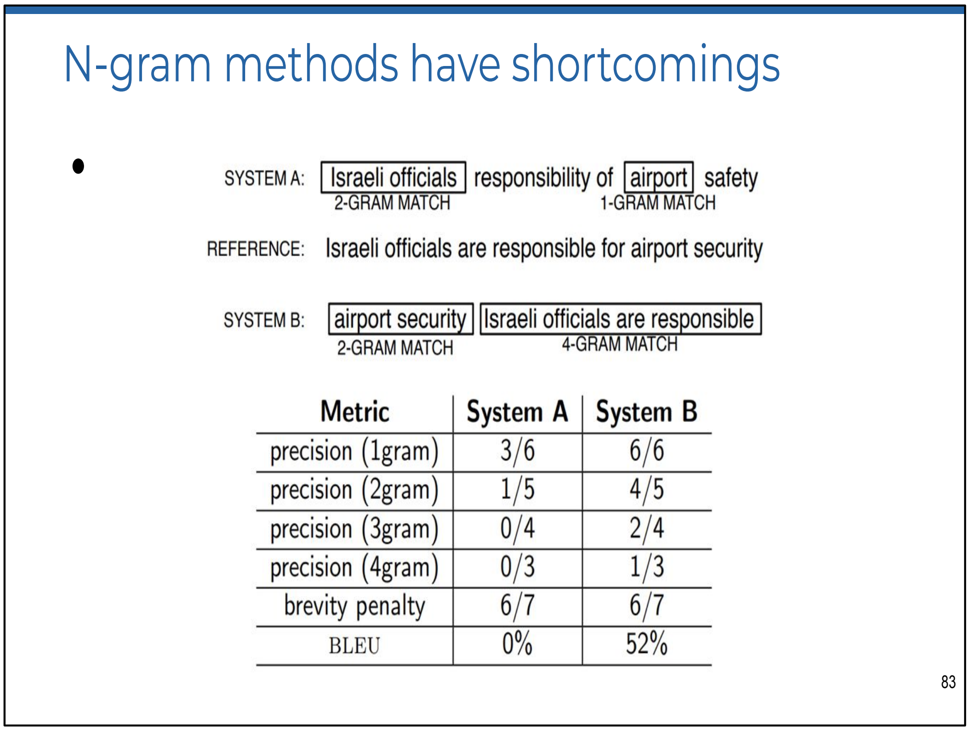
SPEAK SLOWLY



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

SPEAK SLOWLY



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

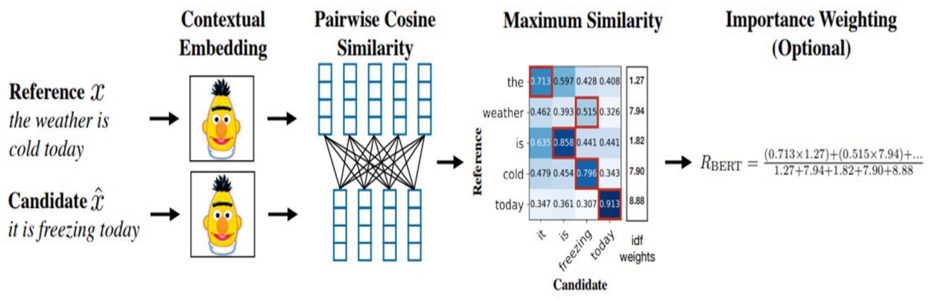
Metric	System A	System B
precision (1gram)	3/6	6/6
precision (2gram)	1/5	4/5
precision (3gram)	0/4	2/4
precision (4gram)	0/3	1/3
brevity penalty	6/7	6/7
BLEU	0%	52%

SPEAK SLOWLY

- Explain diagram. Maybe ask the audience to take 30 seconds and have someone volunteer to explain it?
- 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. (Maybe mention responsibility // responsible?)
-

BertScore

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

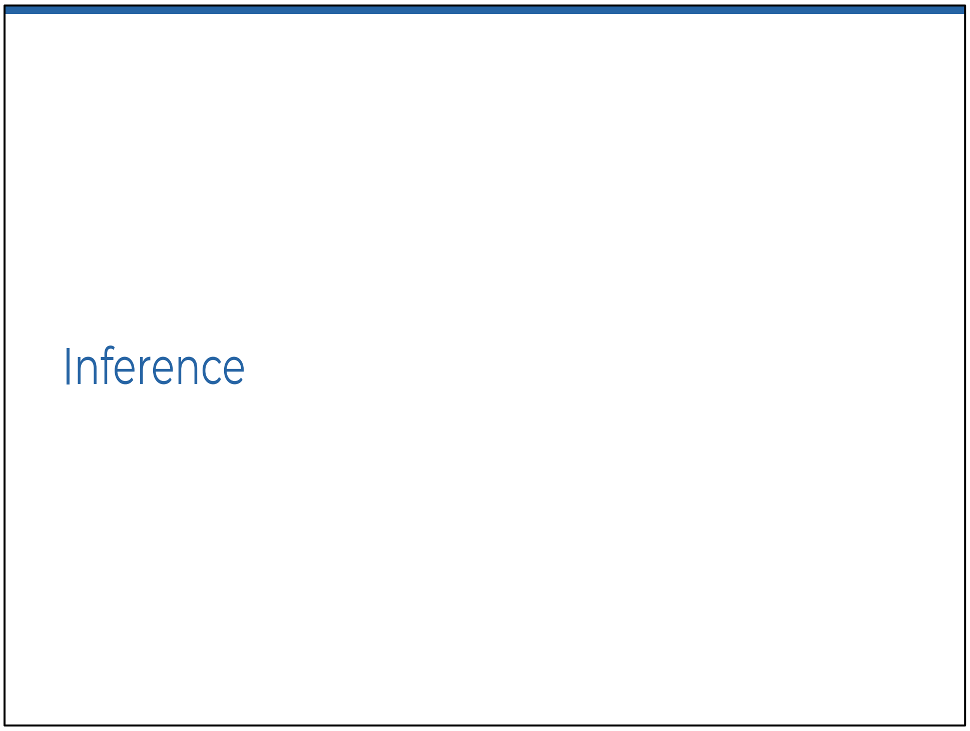


Contextual Embedding
Pairwise Cosine Similarity
Maximum Similarity
Importance Weighting (Optional)
Reference x
the weather is cold today
Candidate $\hat{x}$
it is freezing today
Reference
the 0.713 0.597 0.428 0.408 1.27
weather 0.462 0.393 0.515 0.326 7.94
is 0.635 0.858 0.441 0.441 1.82
cold 0.479 0.454 0.796 0.343 7.90
today 0.347 0.361 0.307 0.913 8.88
it is freezing today
idf weights
Candidate
$R_{\text{BERT}} = \frac{(0.713\times1.27)+(0.515\times7.94)+\ldots}{1.27+7.94+1.82+7.90+8.88}$

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.

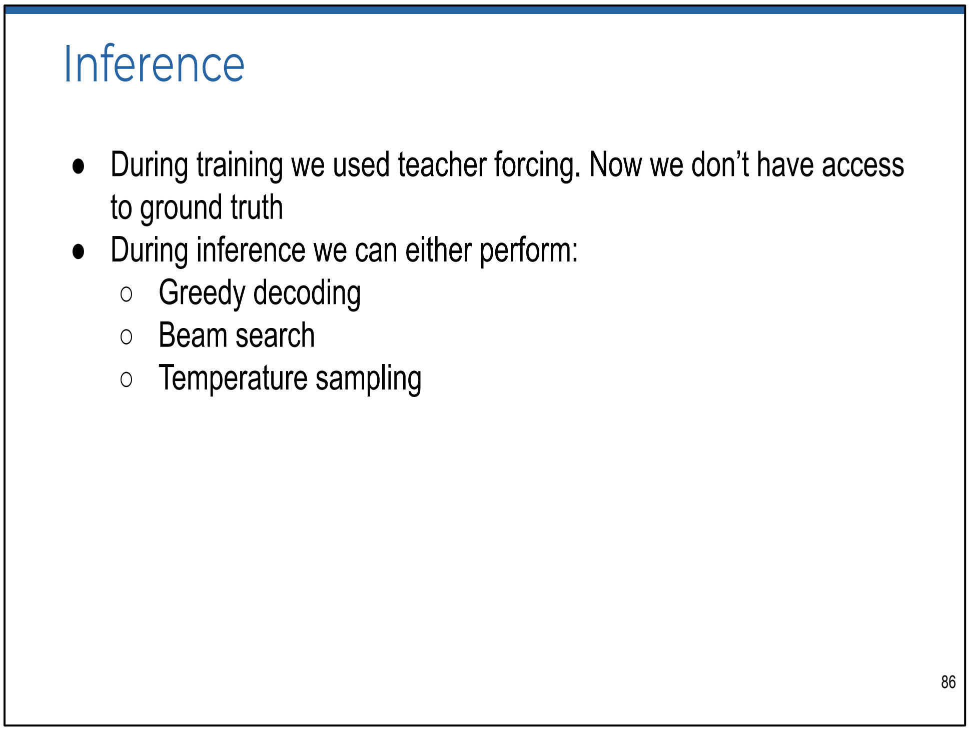
SPEAK SLOWLY

- Don't worry about what BERT is right now (we'll look at it in a future lecture). Right now think of it as a trained language model that can give you contextual representations of tokens/words (e.g. like in a BiRNN)
- Not used so much for translation, but still a good metric for most sequence-to-sequence tasks
- Biggest drawback is now not the n-gram matching. Rather, the scores can vary if evaluated against different BERT models



Inference

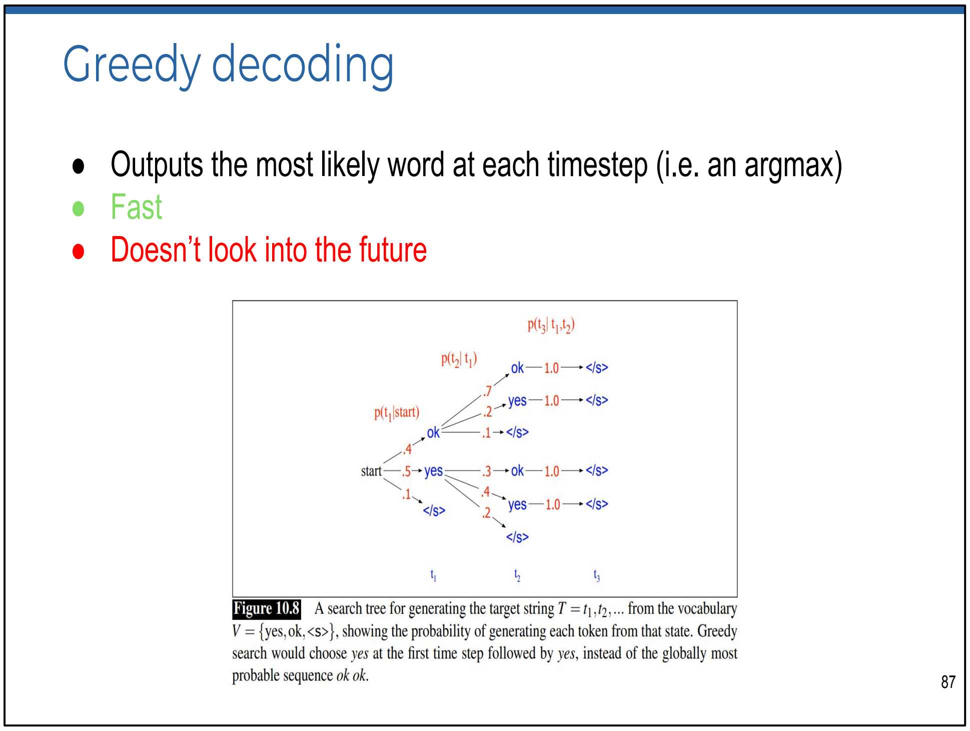
Running ETA: 87



Inference

- During training we used teacher forcing. Now we don’t have access to ground truth
- During inference we can either perform:
  - Greedy decoding
  - Beam search
  - Temperature sampling

SPEAK SLOWLY



Greedy decoding

- Outputs the most likely word at each timestep (i.e. an argmax)
- Fast
- Doesn’t look into the future

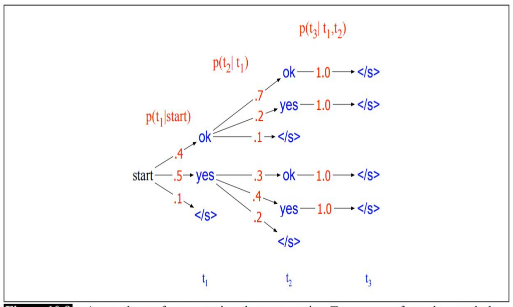


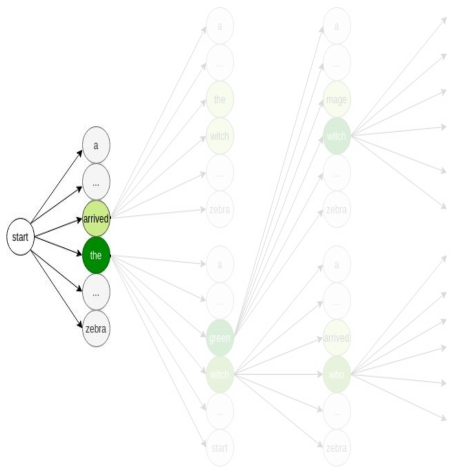
Figure 10.8 A search tree for generating the target string $T = t_1, t_2, \ldots$ from the vocabulary $V = \{\text{yes}, \text{ok}, \text{<s>}\}$, showing the probability of generating each token from that state. Greedy search would choose *yes* at the first time step followed by *yes*, instead of the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

SPEAK SLOWLY

- Chosen word might be best at current timestep. 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

Beam Search t=0

- Instead of choosing the best token to generate at each timestep, we keep *k* possible tokens at each step

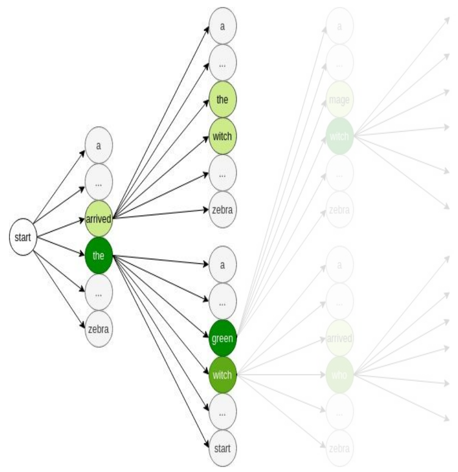


SPEAK SLOWLY

- In practise, k is normally between 5-10. For the example we're about to work through, k=2
- Note that we will end up with k hypothesis at the end of decoding. Decoding finishes when we hit an EOS token
- Example run through:
    - t=0: arrived and the are the 2 most likely words
    -

Beam Search (t=1)

- Instead of choosing the best token to generate at each timestep, we keep *k* possible tokens at each step

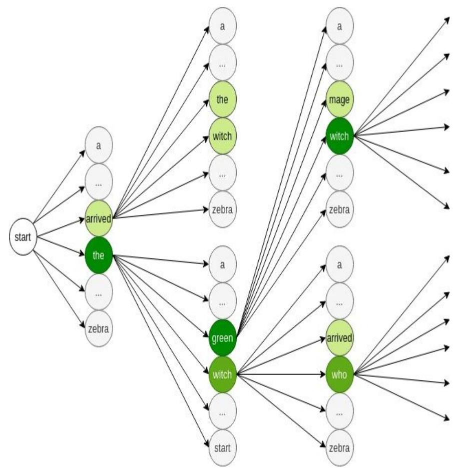


SPEAK SLOWLY

t=1: for each of these words, decode the next k. So we have [start arrived the, start arrived witch, start the green, start the witch]
Then we prune all but the top-k: [start the green, start the witch]

Beam Search t=2

- Instead of choosing the best token to generate at each timestep, we keep *k* possible tokens at each step



SPEAK SLOWLY

t=2: Repeat. Now we have [start the green mage, start the green witch, start the witch arrived, start the witch who]
After pruning: [start the green witch, start the green who]

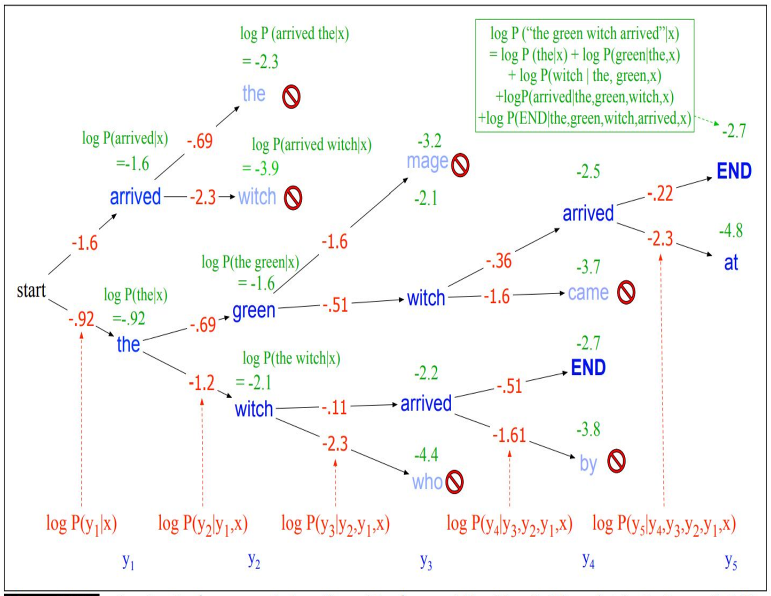
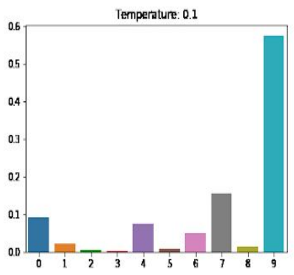


Figure 10.10 Scoring for beam search decoding with a beam width of $k = 2$. We maintain the log probability of each hypothesis in the beam by incrementally adding the logprob of generating each next token. Only the top k paths are extended to the next step.

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

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



SPEAK SLOWLY

- Temperature sampling lets us inject non-determinism into our decoder.
  - Perhaps not ideal for translation, but can be useful for language modelling
- Gif is the post softmax value of each of the 10 classes.
- Higher temperature leads to smoother softmax operation.
- Thus more diverse (but sometimes less coherent) outputs.

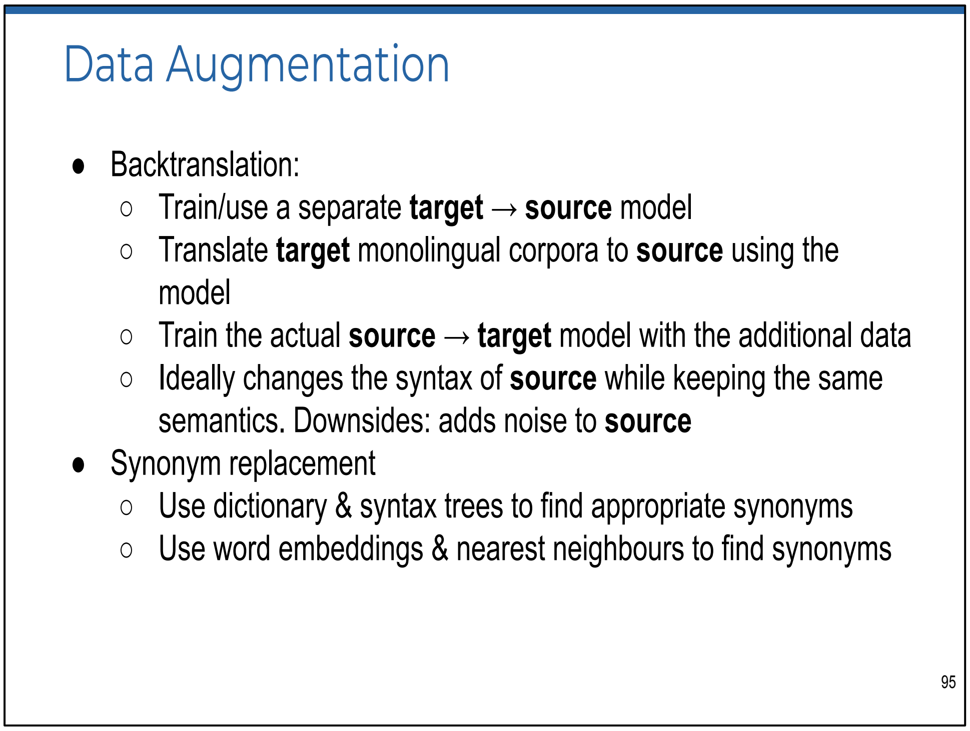
Tricks

Running ETA: 87

Data Augmentation

- Your ideas?

-

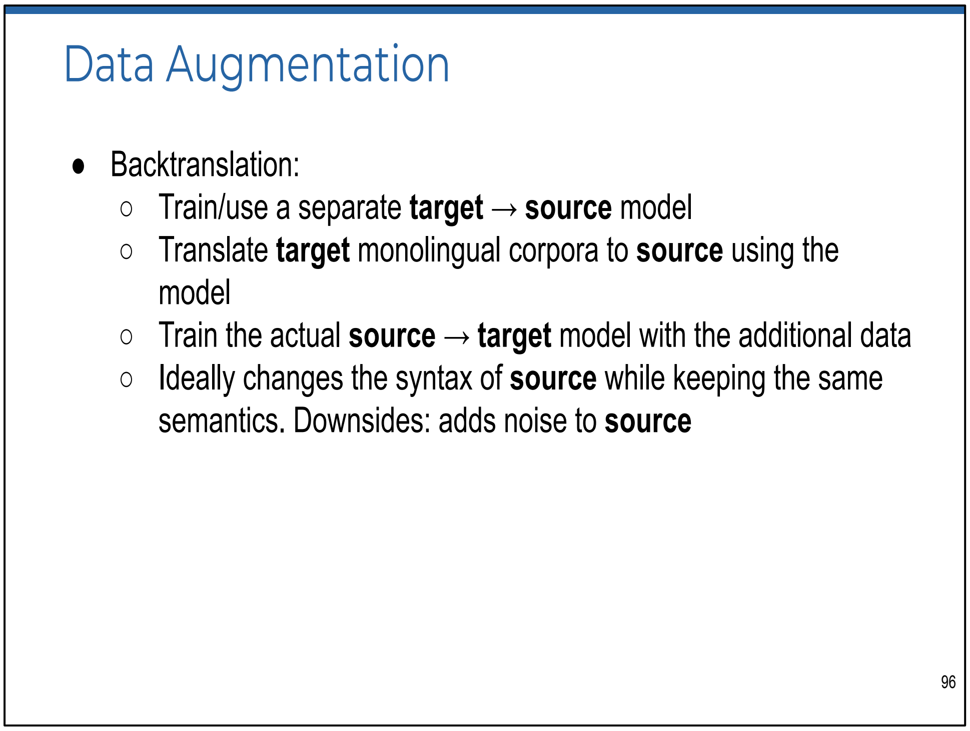


Data Augmentation

- Backtranslation:
  - Train/use a separate **target** → **source** model
  - Translate **target** monolingual corpora to **source** using the model
  - Train the actual **source** → **target** model with the additional data
  - Ideally changes the syntax of **source** while keeping the same semantics. Downsides: adds noise to **source**
- Synonym replacement
  - Use dictionary & syntax trees to find appropriate synonyms
  - Use word embeddings & nearest neighbours to find synonyms

SPEAK SLOWLY

- Noise in **source** may not always be detrimental. You'll see more about this when looking at BART in a couple of lectures
-

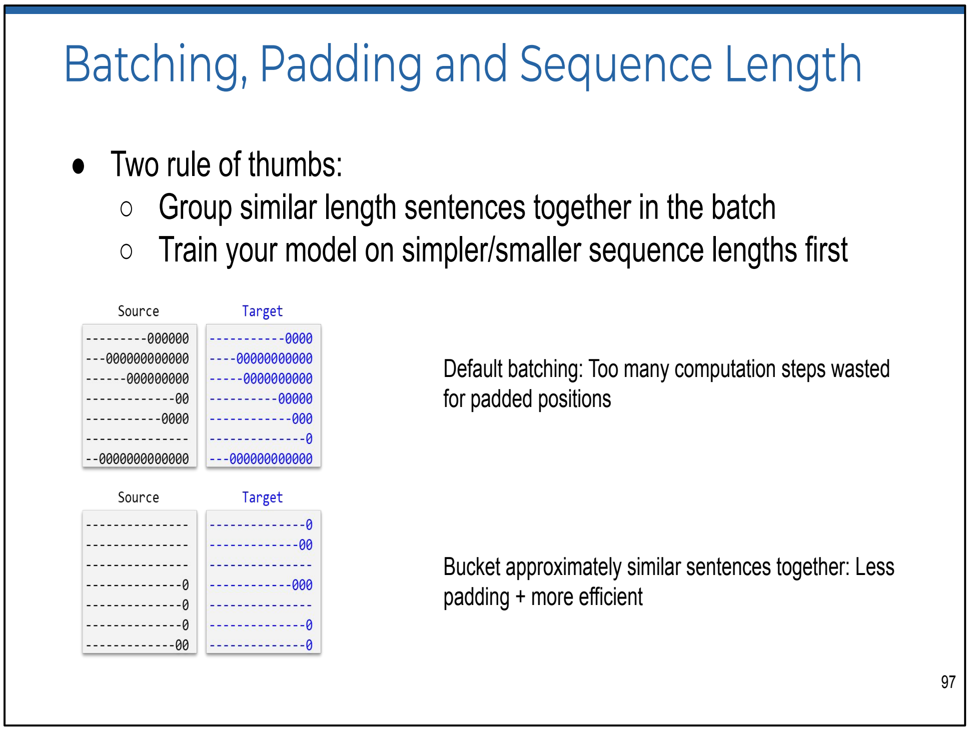


Data Augmentation

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

SPEAK SLOWLY

- Noise in **source** may not always be detrimental. You'll see more about this when looking at BART in a couple of lectures
-



Batching, Padding and Sequence Length

- Two rule of thumbs:
  - Group similar length sentences together in the batch
  - Train your model on simpler/smaller sequence lengths first

```
    Source            Target
---------000000   -----------0000
---000000000000   ----00000000000
------000000000   -----0000000000
-------------00   ----------00000
-----------0000   ------------000
---------------   --------------0
--0000000000000   ---000000000000
```

Default batching: Too many computation steps wasted for padded positions

```
    Source            Target
---------------   --------------0
---------------   -------------00
---------------   ---------------
--------------0   ------------000
--------------0   ---------------
--------------0   --------------0
-------------00   --------------0
```

Bucket approximately similar sentences together: Less padding + more efficient

SPEAK SLOWLY

- Regarding training on smaller sequence lengths first.. Models have been shown to better model more complicated sequences when they've been exposed to sequences in gradually increasing complexity

CODE

- Running ETA: 46

Running ETA: 95