How does LSTM solve vanishing gradients?

The LSTM architecture makes it easier for the RNN to preserve information over many timesteps

- e.g., if the forget gate is set to remember everything on every timestep, then the info in the cell is preserved indefinitely
- By contrast, it is harder for vanilla RNN to learn a recurrent weight matrix U that preserves info in hidden state

Common RNN NLP Architectures

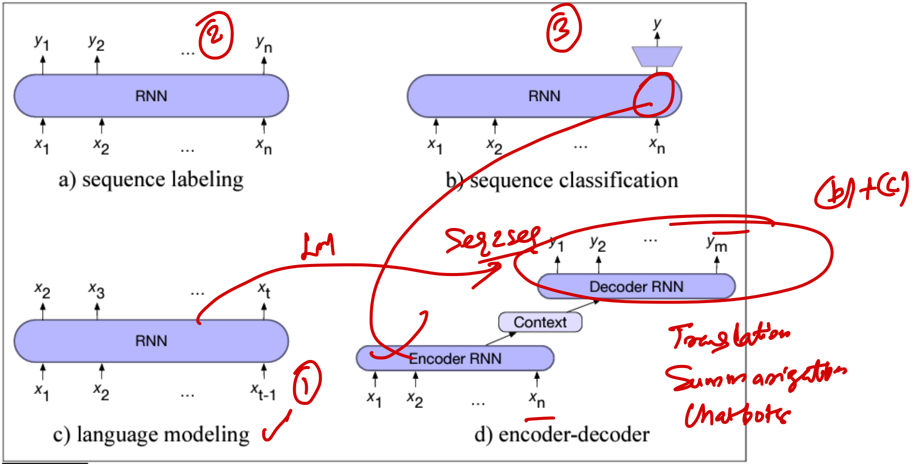


Figure 9.15 Four architectures for NLP tasks. In sequence labeling (POS or named entity tagging) we map each input token x_i to an output token y_i. In sequence classification we map the entire input sequence to a single class. In language modeling we output the next token conditioned on previous tokens. In the encoder model we have two separate RNN models, one of which maps from an input sequence **x** to an intermediate representation we call the **context**, and a second of which maps from the context to an output sequence **y**.

Encoder-decoder networks

Also known as sequence-to-sequence networks, and are capable of generating contextually appropriate, arbitrary length output sequences given the input sequence.

Three conceptual components

- An **encoder** that accepts an input sequence $x_{1:n}$ and generates a corresponding sequence of contextualized representations $h_{1:n}$
- A **context** vector, c, which is a function of $h_{1:n}$ and conveys the essence of the input to the decoder
- A **decoder** which accepts c as input and generates an arbitrary length sequence of hidden states $h_{1:m}$ from which the corresponding output states $y_{1:m}$ can be obtained.

Encoder-decoder networks

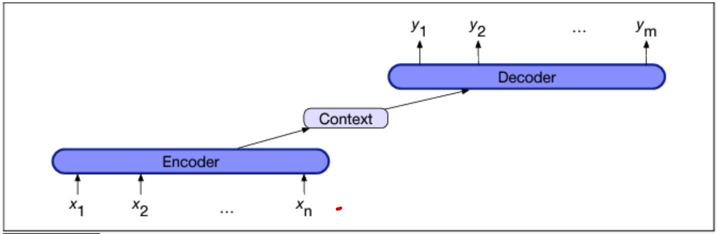
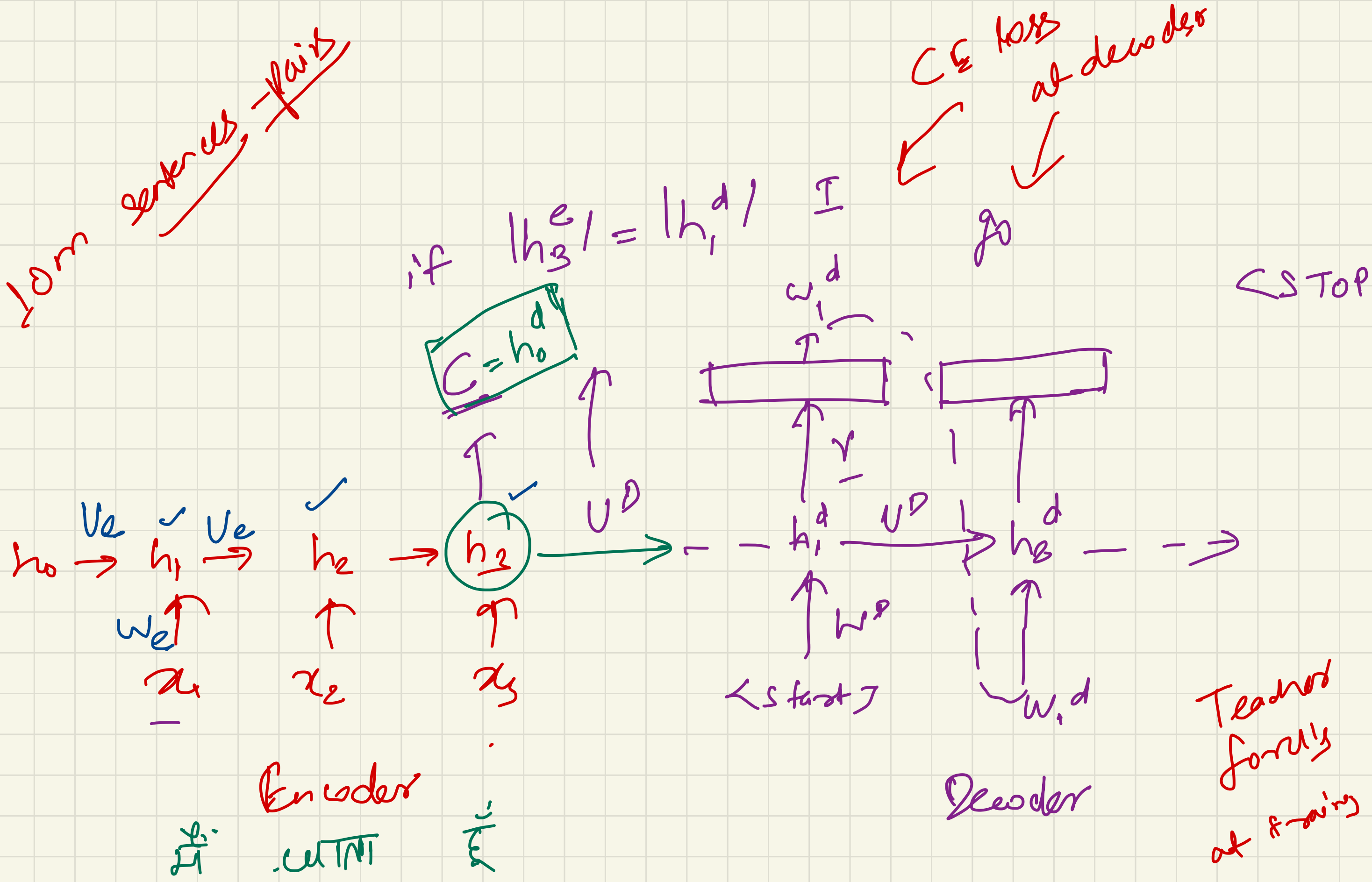


Figure 9.16 The encoder-decoder architecture. The context is a function of the hidden representations of the input, and may be used by the decoder in a variety of ways.


10m sentences-pairs
CE loss
at decoder
if $|h_3^e| = |h_1^d|$
I
go
<STOP>
w_1^d
$C = h_0^d$
U^D
V
h_0
V_e
h_1
h_2
h_3
h_1^d
h_2^d
W_e
W^D
x_1
x_2
x_3
<start>
w_1^d
Encoder
Decoder
Teacher forcing
at training

Encoder-decoder networks for translation

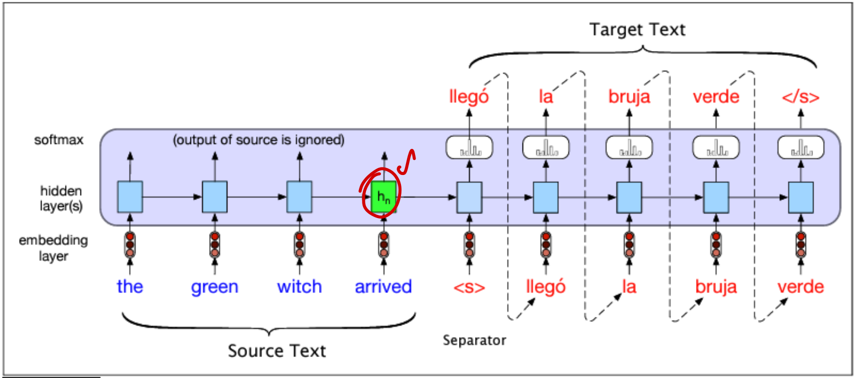


Figure 9.17 Translating a single sentence (inference time) in the basic RNN version of encoder-decoder approach to machine translation. Source and target sentences are concatenated with a separator token in between, and the decoder uses context information from the encoder's last hidden state.

Training the Encoder-decoder model

End-to-end training

- For MT, the training data typically consists of set of sentences and their translations
- The network is given a source sentence and then a separator token, it is trained auto-regressively to predict the next word
- Teacher forcing is used during training, i.e., the system is forced to use the gold target token from training as the next input x_{t+1}, rather than relying on the last decoder output $\hat{y}_t$

Training the Encoder-decoder model

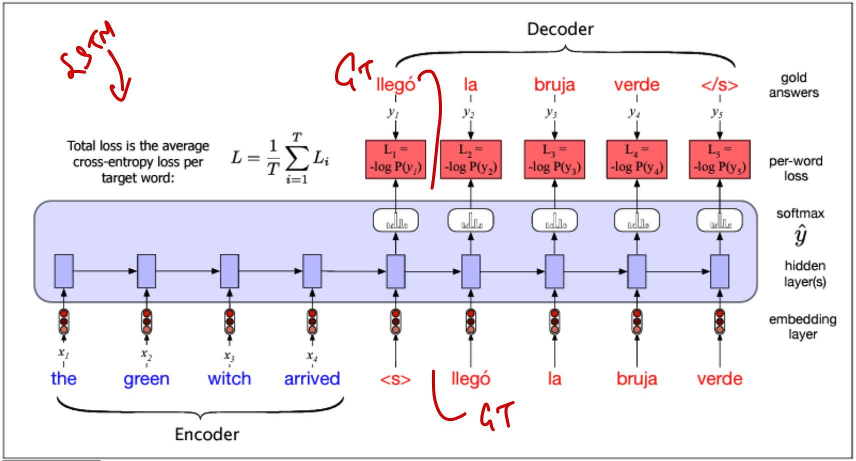


Figure 9.19 Training the basic RNN encoder-decoder approach to machine translation. Note that in the decoder we usually don't propagate the model's softmax outputs $\hat{y}_t$, but use **teacher forcing** to force each input to the correct gold value for training. We compute the softmax output distribution over $\hat{y}$ in the decoder in order to compute the loss at each token, which can then be averaged to compute a loss for the sentence.

Encoder-decoder: Bottleneck

- The context vector, h_n is the hidden state of the last time step of the source text
- It acts as a bottleneck, as it has to represent absolutely everything about the meaning of the source text, as this is the only thing decoder knows about the source text

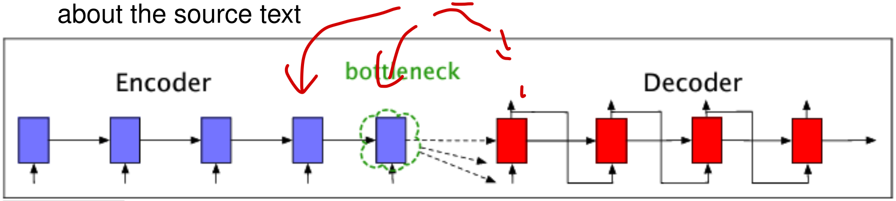


Figure 9.20 Requiring the context c to be only the encoder's final hidden state forces all the information from the entire source sentence to pass through this representational bottleneck.

Encoder-decoder with attention

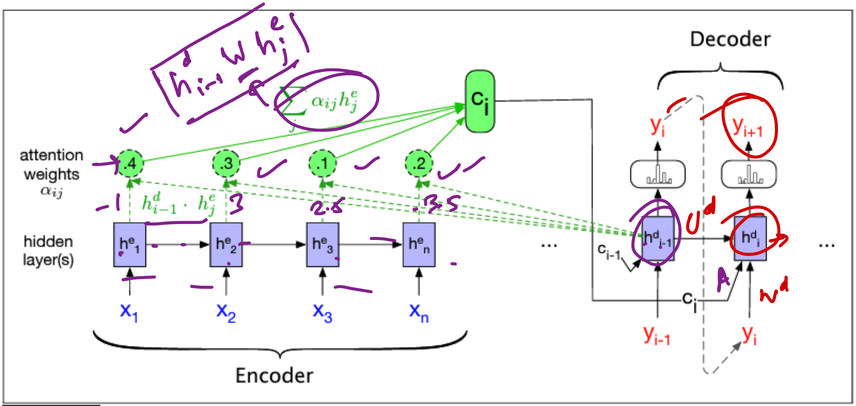


Figure 9.22 A sketch of the encoder-decoder network with attention, focusing on the computation of $\mathbf{c}_i$. The context value $\mathbf{c}_i$ is one of the inputs to the computation of $\mathbf{h}^d_i$. It is computed by taking the weighted sum of all the encoder hidden states, each weighted by their dot product with the prior decoder hidden state $\mathbf{h}^d_{i-1}$.

Attention: In Equations

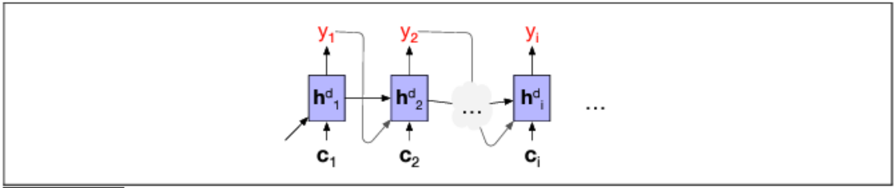


Figure 9.21 The attention mechanism allows each hidden state of the decoder to see a different, dynamic, context, which is a function of all the encoder hidden states.

The context vector c_i is generated anew with each decoding step i

$$\boxed{h_i^d = g(\hat{y}_{i-1}, h_{i-1}^d, c_i)}$$

$$= \tanh\left(W\hat{y}_{i-1} + U h_{i-1}^d + A c_i\right)$$

Attention: In Equations

Computing c_i

- Compute how much to focus on each encoder state, by seeing how relevant it is to the decoder state captured in h^d_{i-1} – give it a score
- Simplest scoring mechanism is dot-product attention

$$\underline{score(h^d_{i-1}, h^e_j)} = \underline{h^d_{i-1}} \cdot \underline{h^e_j}$$

- Normalize these scores using softmax to create a vector of weights $\underline{\alpha_{ij}} = \underline{softmax}(score(h^d_{i-1}, h^e_j))$
- A fixed-length context vector is created for the current decoder state

$$c_i = \sum_j \alpha_{ij} h^e_j$$

Attention is quite helpful

Attention improves NMT performance

It is useful to allow decoder to focus on certain parts of the source

Attention helps with the long-term dependency problem

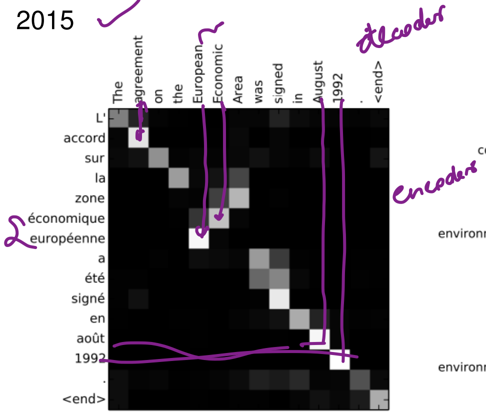
Provides shortcut to faraway states ✓

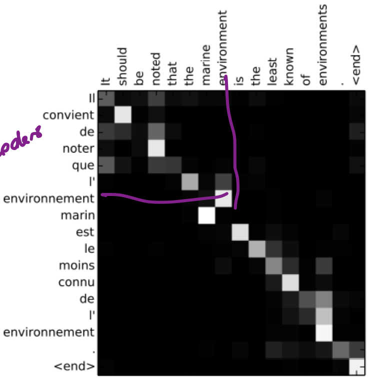
Attention provides some interpretability

- By inspecting attention distribution, we can see what the decoder was focusing on
- We get alignment for free even if we never explicitly trained an alignment system

Example: Machine Translation

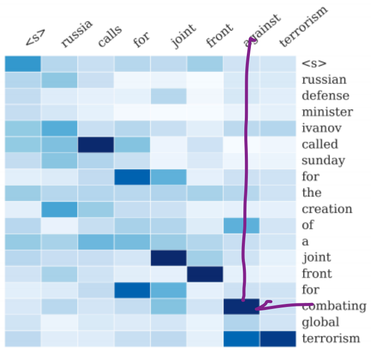
Neural Machine Translation by jointly learning to align and Translate, ICLR 2015 ✓





Example: Text Summarization

A Neural Attention Model for Sentence Summarization, EMNLP 2015

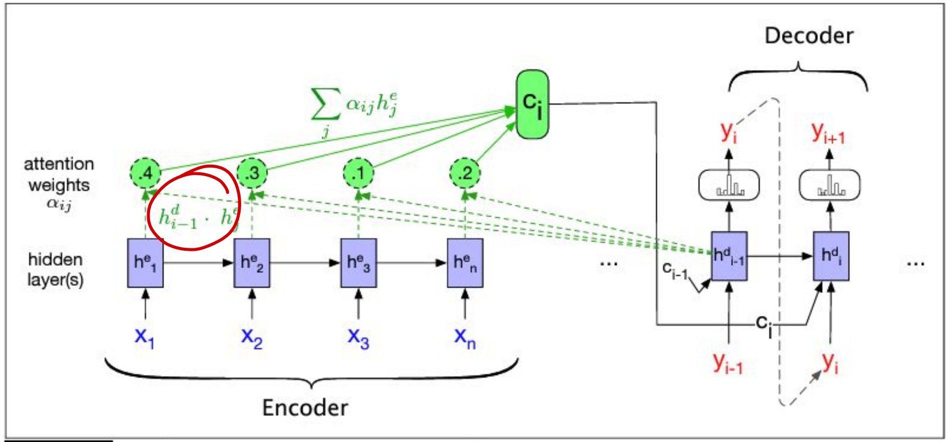


Summary

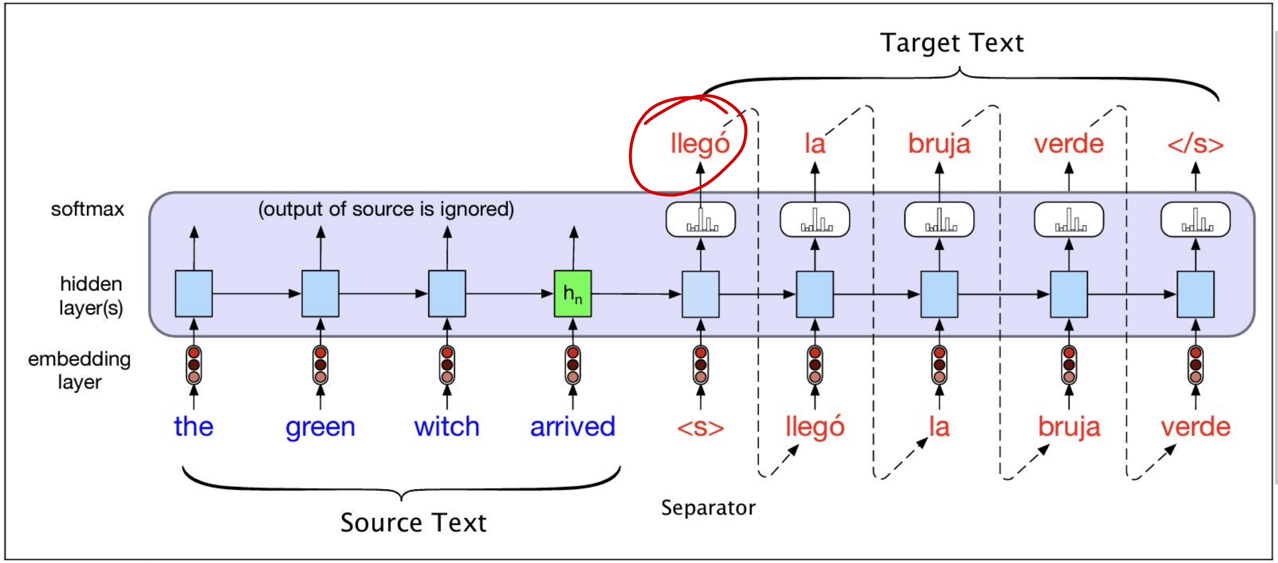
- Attention has proved to be a very impactful idea in NLP
- Lot of new models are based on self-attention, e.g., Transformer, BERT

Try this Problem

Suppose you are using sequence to sequence RNN model with attention for Machine Translation from English to Hindi. Assume that the vocabulary for English is 40k and that for Hindi is 50k, and both use 300-dimensional embeddings. Also, assume that the encoder hidden state is 100-dimensional, while decoder hidden state is 200-dimensional. What will be the number of parameters to be trained? You may ignore the bias terms.



Recap: Encoder-decoder model at inference



Translating a single sentence (inference time) in the basic RNN version of encoder-decoder approach to machine translation. Source and target sentences are concatenated with a separator token in between, and the decoder uses context information from the encoder's last hidden state.

How do we generate the tokens in an autoregressive manner?

Source: https://web.stanford.edu/~jurafsky/slp3.

Which words do we generate at each step?

Greedy Decoding

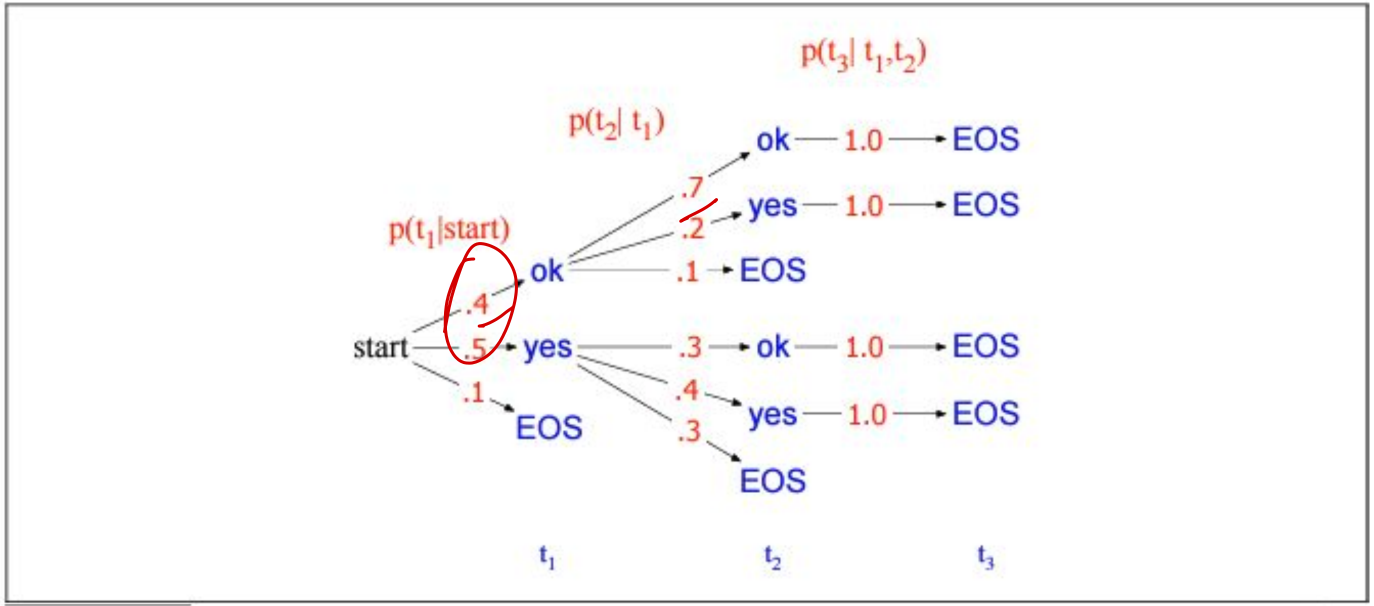
One simple way is to always generate the most likely word given the context

Generating the most likely word given the context is called *greedy decoding*.

$$\hat{w}_t = \arg\max_{w \in V} P(w|w_{<t})$$

What is the issue with greedy decoding?

beam-size =K

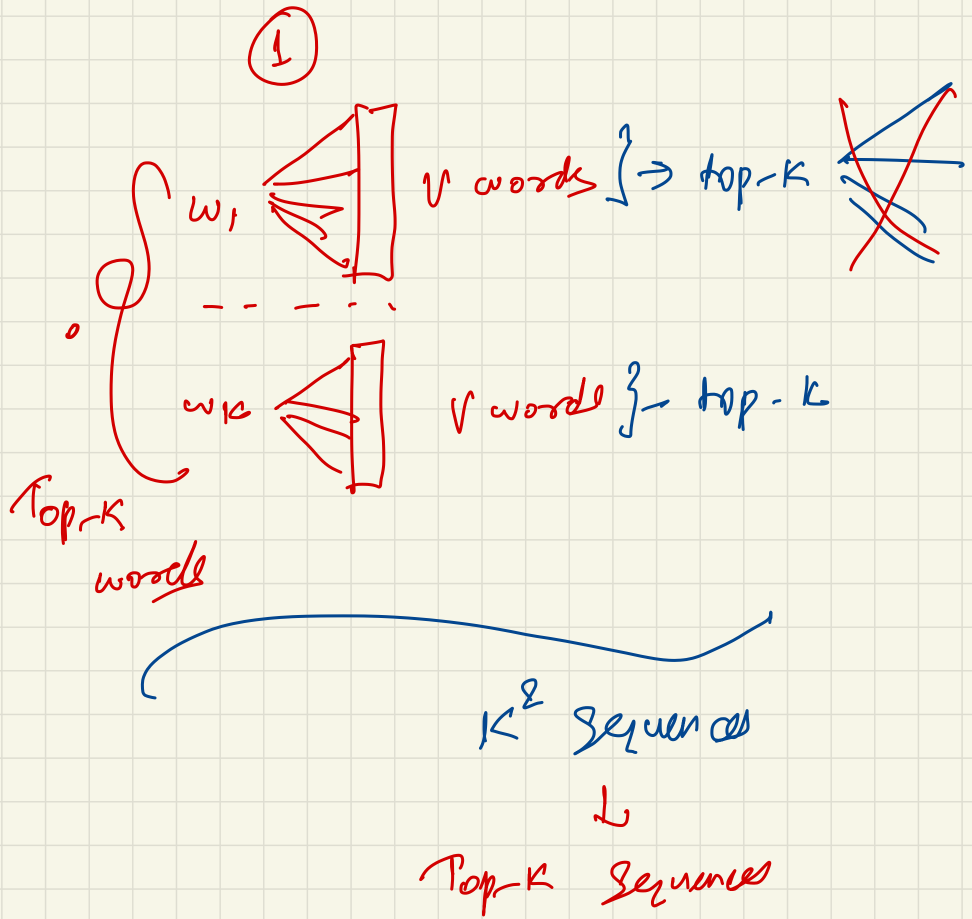


T-length seq

(V^T possible sequences)

- *What will greedy decoding choose?* Yes Yes EOS (0.2)
- *What is globally optimal?* OK OK EOS (0.28)

Source: https://web.stanford.edu/~jurafsky/slp3.

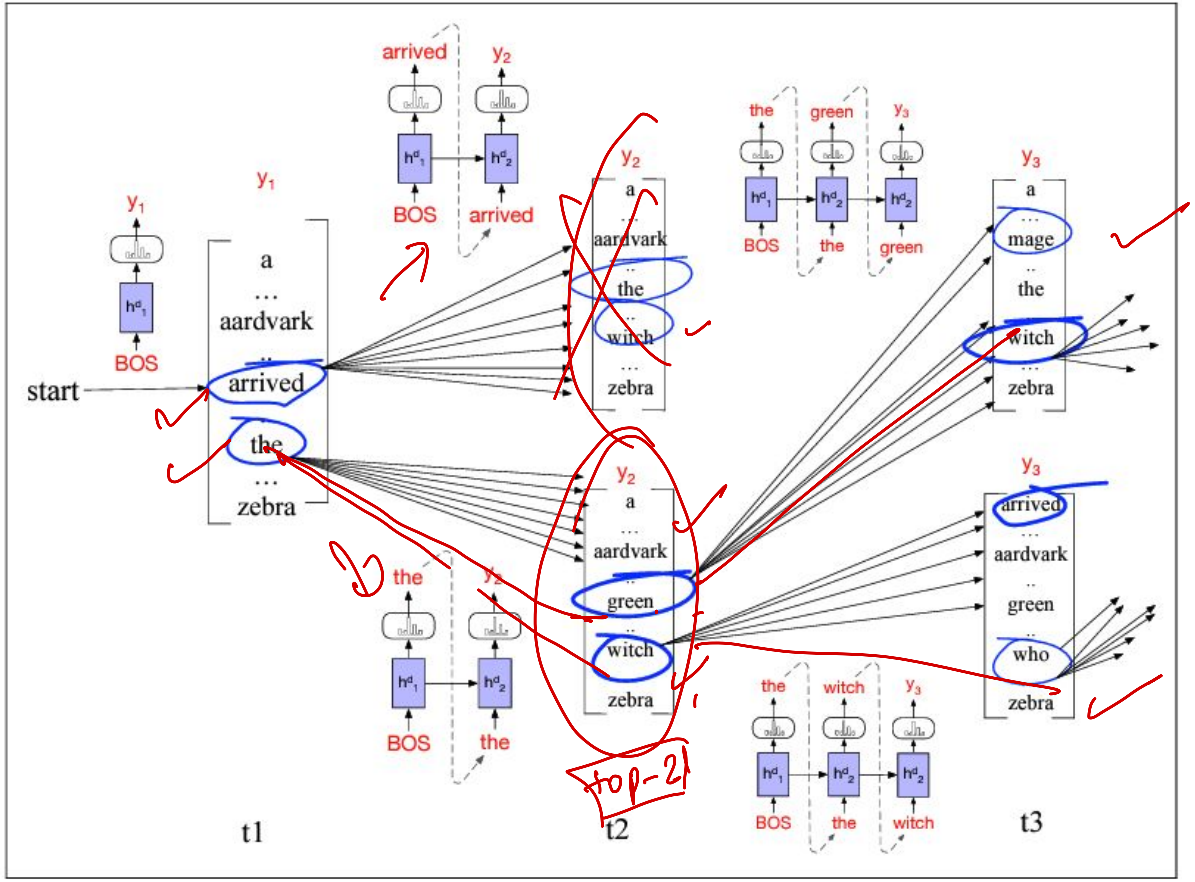

1
w_1
V words
top-K
w_K
V words
top-K
Top-K words
K^2 Sequences
Top-K Sequences

Beam Search

Core Idea

- On each step of the decoder, keep track of the k most probable partial translations (hypotheses)
- k is the beam size / beam width (in practice around 5 to 10)

Beam Search: How does it work?



https://web.stanford.edu/~jurafsky/slp3

$-\log p(\text{arrived}) +$
$-\log p(\text{the}$ ①
$-\log p(\text{the}) - \log p(\text{green}$