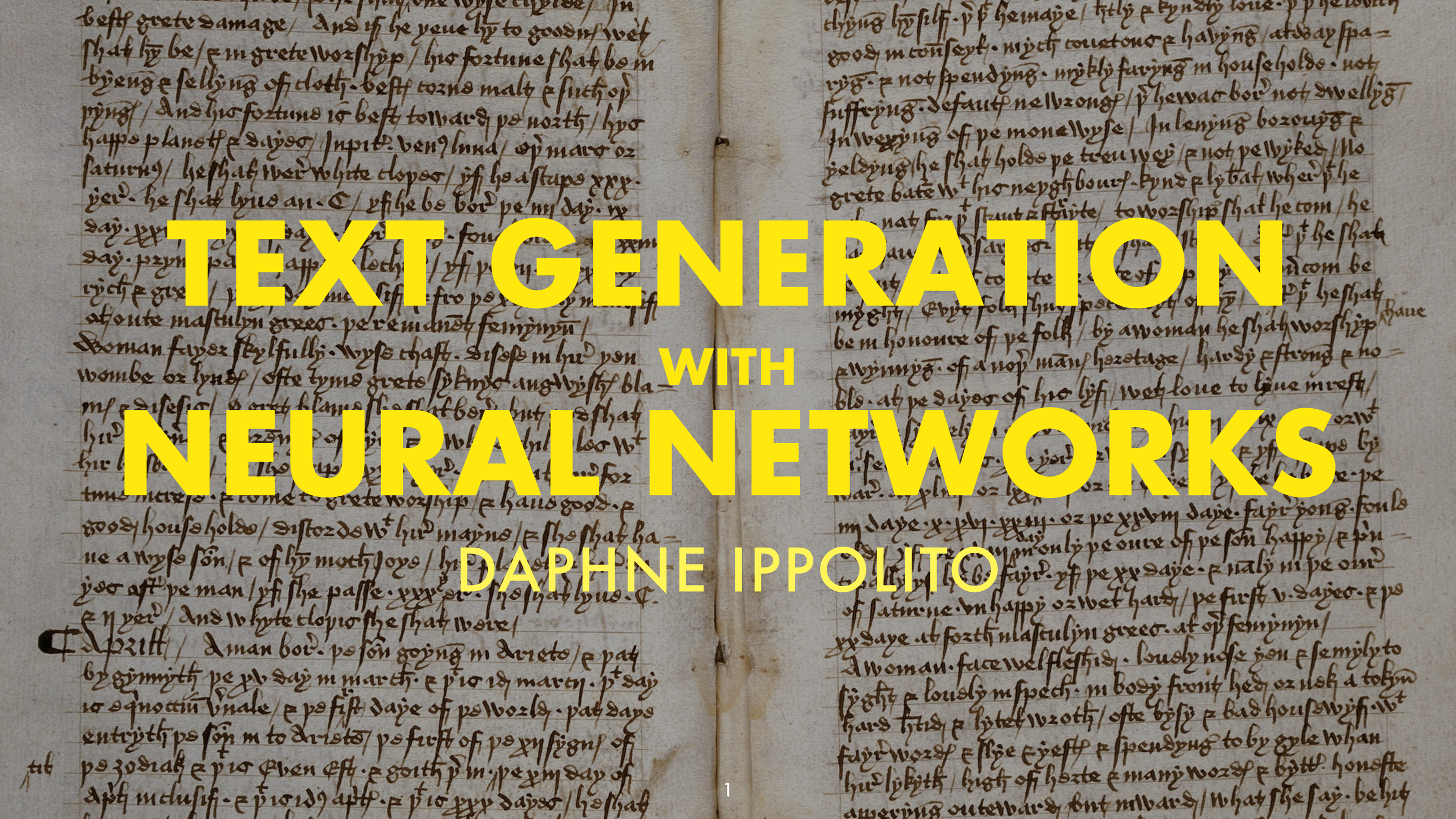


TEXT GENERATION WITH NEURAL NETWORKS

DAPHNE IPPOLITO

OUTLINE

- Neural language model framework
- LM Architectures
  - Recurrent neural networks
  - Transformers
- Decoding Strategies
- Transformers for natural language understanding
  - BERT
  - T5

OUTLINE

- **Neural language model framework**
- LM Architectures
  - Recurrent neural networks
  - Transformers
- Decoding Strategies
- Transformers for natural language understanding
  - BERT
  - T5

WHAT IS A LANGUAGE MODEL?

A language model outputs the probability distribution over the next word given the previous words in a string.

Historically, language models were statistical. If the word “apple” follows the word “the” 2% of the times that “the” occurs in the text corpus, then `P(“apple” | “the”) = 0.02`.

More recently, we use neural language models, which can condition on much longer sequences, ie. `P(“apple" | “I was about to eat the”).` They are also able to generalize to sequences which are not in the training set.

WHAT IS A LANGUAGE MODEL?

A REMINDER ABOUT THE CHAIN RULE

Using the chain rule, we can refer to the probability of a sequence of words as the product of the conditional probability of each word given the words that precede it.

```
P(["I", "eat", "the", "apple"]) =
P("apple" | ["I", "eat", "the"]) * P("the" | ["I", "eat"]) * P("eat" | ["I"]) * P("I")
```

This is helpful since language models output $P(y_t | y_{1:t-1})$.

NEURAL LANGUAGE MODELS

UNCONDITIONED VS CONDITIONED

Neural language models can either be designed to just predict the next word given the previous ones, or they can be designed to predict the next word given the previous ones and some additional conditioning sequence.

Unconditioned: $P(Y)$

At each step the LM predicts: $P(y_t \mid y_{1:t-1})$

Tasks that are usually unconditioned:

- Story generation
- News article generation

Conditioned: $P(Y \mid X)$

At each step the LM predicts: $P(y_t \mid y_{1:t-1}, x_{1:T})$

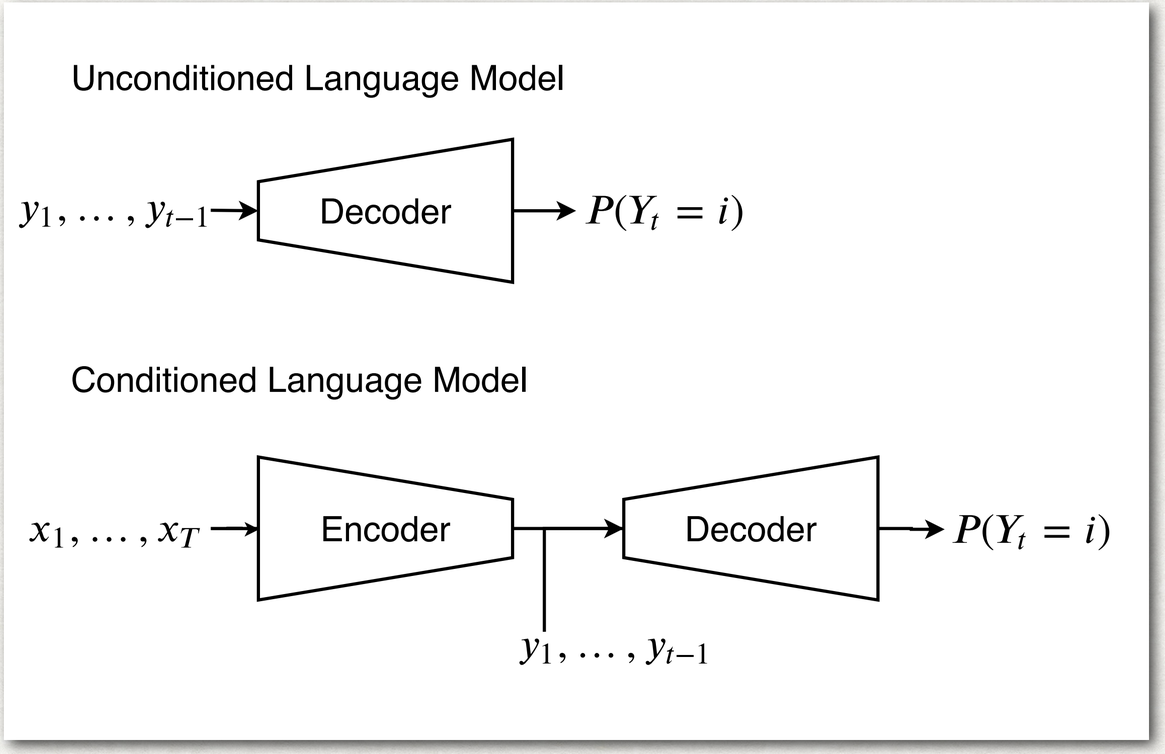
Tasks that are usually conditioned:

- Machine translation
- Abstractive text summarization
- Simplification

NEURAL LANGUAGE MODELS

UNCONDITIONED VS CONDITIONED

Unconditioned neural language models only have a decoder. Conditioned ones have an encoder and a decoder.



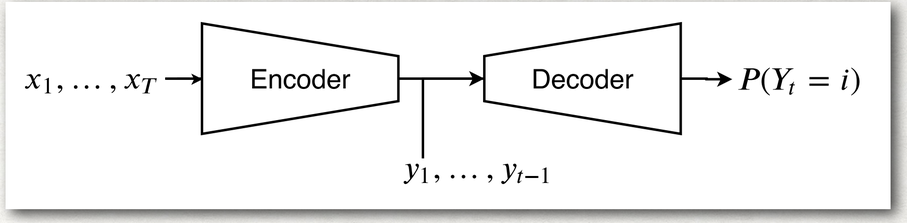
NEURAL LANGUAGE MODELS

UNCONDITIONED VS CONDITIONED

Theoretically, any task designed for a decoder-only architecture can be turned into one for an encoder-decoder architecture, and vice-versa.

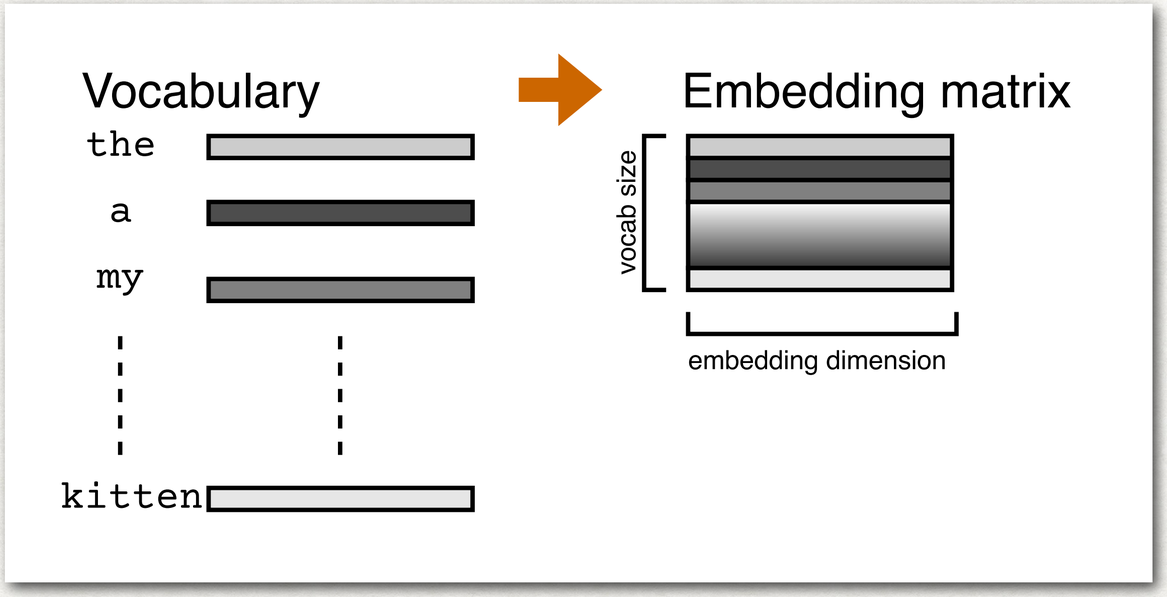
Unconditioned (decoder-only) examples	Conditioned (encoder-decoder) examples
• `Once upon a time there lived a beautiful ogre who ...`	• `Once upon a time there lived a beautiful ogre who ➡️ fell in love with...`
• `[tag_Title] Truck Overturns on Highway Spilling Maple Syrup [tag_Body] The truck was ...`	• `Truck Overturns on Highway Spilling Maple Syrup ➡️ The truck was...`
• `[source] The hippopotamus ate my homework. [target] ...`	• `The hippopotamus ate my homework. ➡️ ...`
• `[complex] The incensed hippopotamus consumed my assignment. [simple] ...`	• `The incensed hippopotamus consumed my assignment. ➡️ The angry hippo ate my ...`

NEURAL LANGUAGE MODELS



The first step of building a neural language model is constructing a vocabulary of valid tokens.

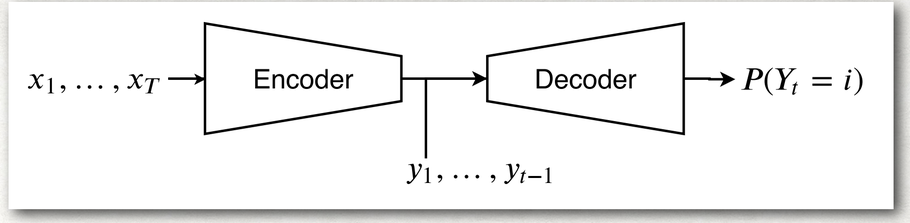
Each token in the vocabulary is associated with a vector embedding, and these are concatenated into an embedding matrix.

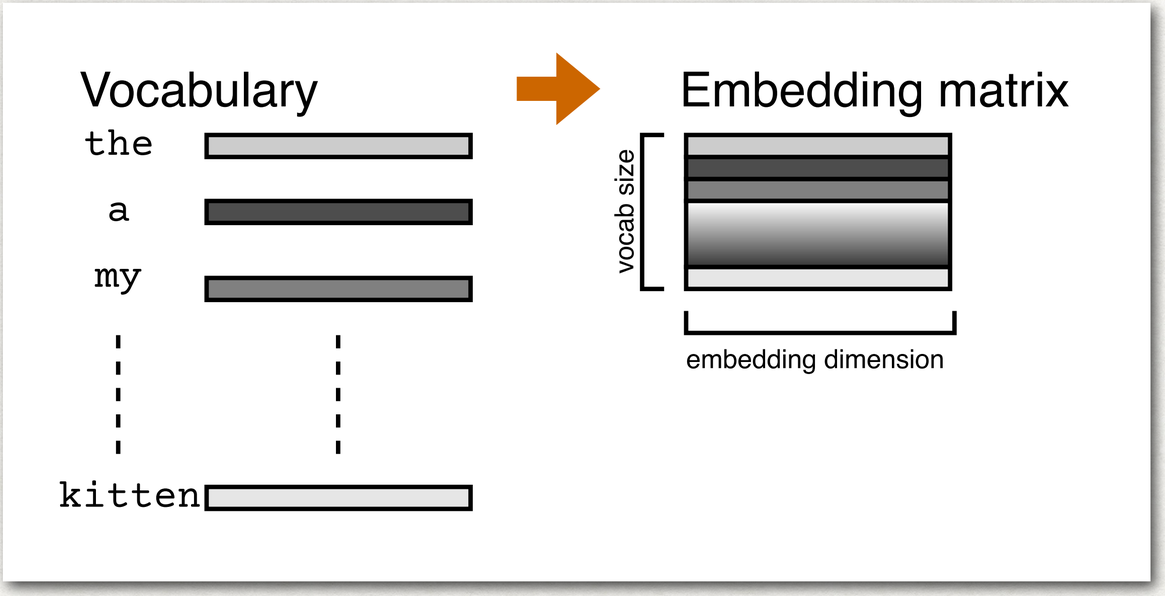


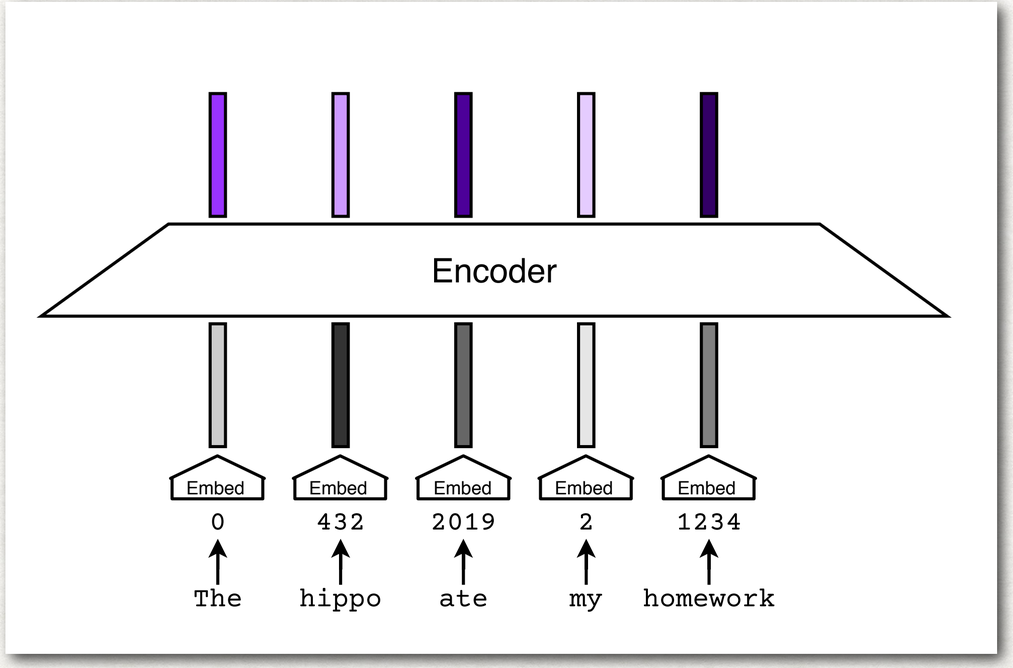
NEURAL LANGUAGE MODELS

The first step of building a neural language model is constructing a vocabulary of valid tokens.

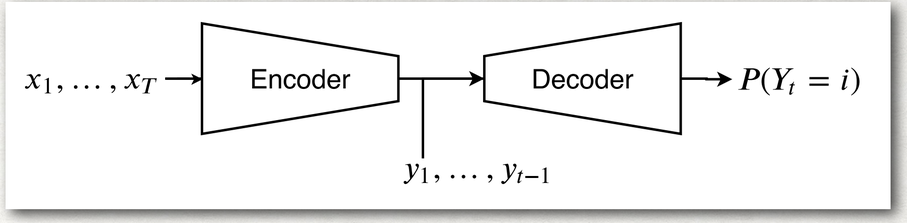
Each token in the vocabulary is associated with a vector embedding, and these are concatenated into an embedding matrix.





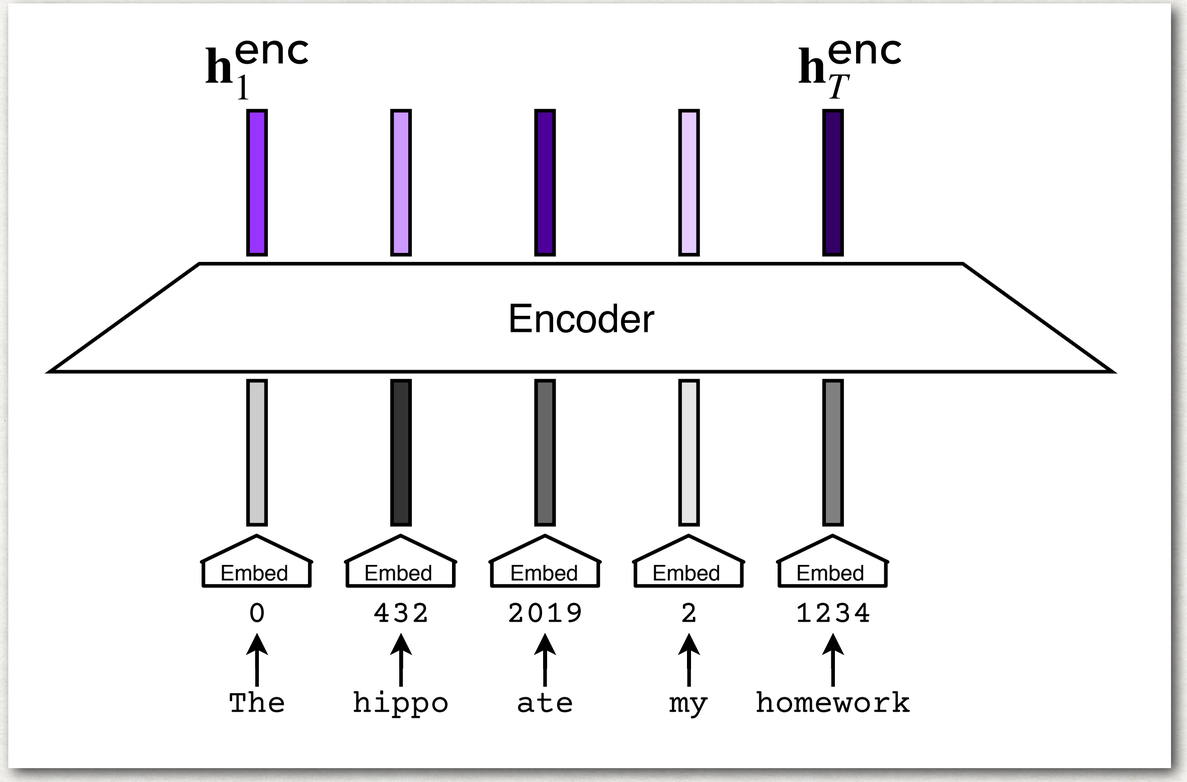


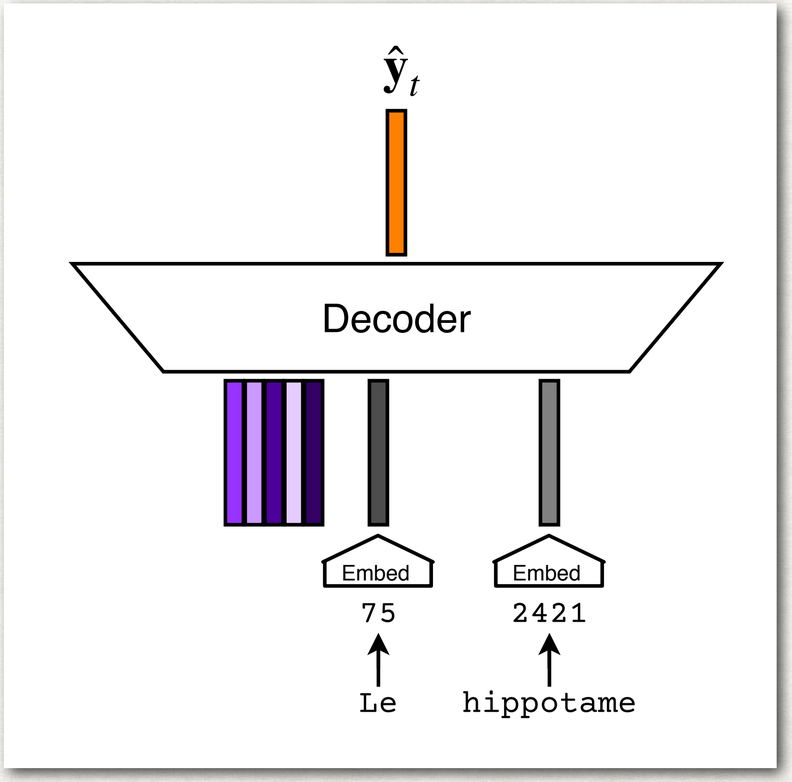
NEURAL LANGUAGE MODELS



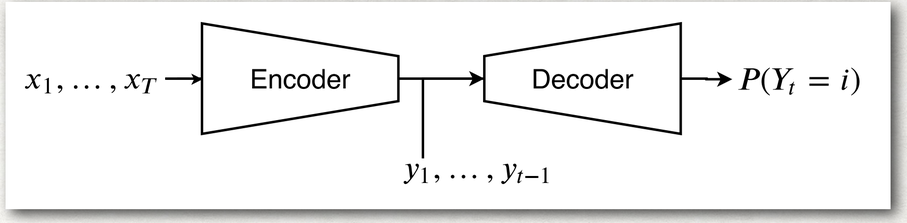
The encoder outputs a sequence of hidden states for each token in the source sequence.

The decoder takes as input the hidden states from the encoder as well as the embeddings for the tokens seen so far in the target sequence.

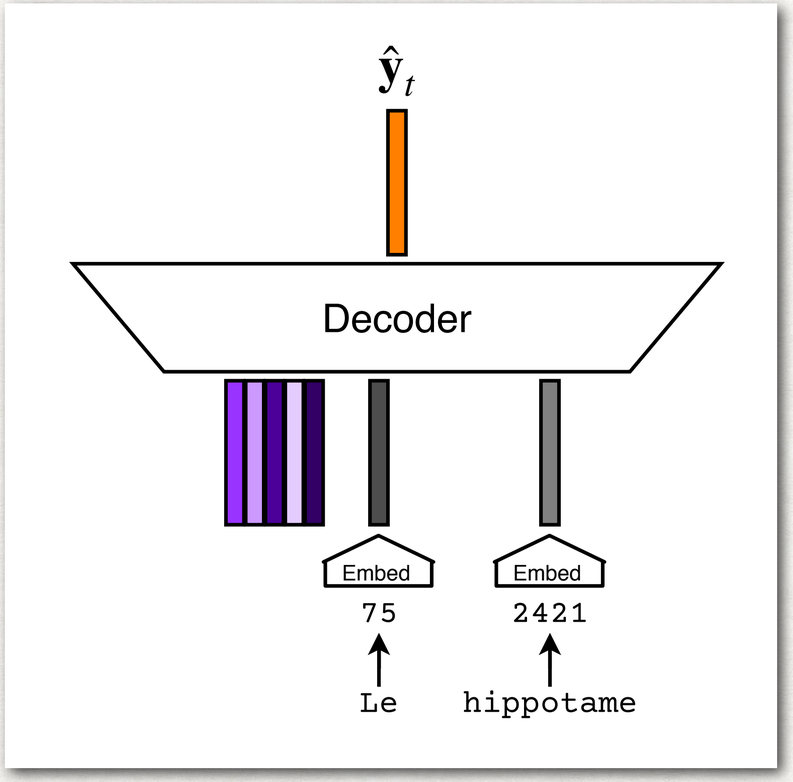




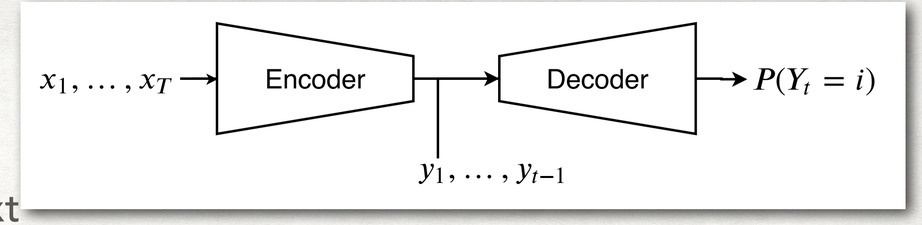
NEURAL LANGUAGE MODELS



Ideally the predicted embedding $\hat{\mathbf{y}}_t$ is close to the embedding of the true next word.



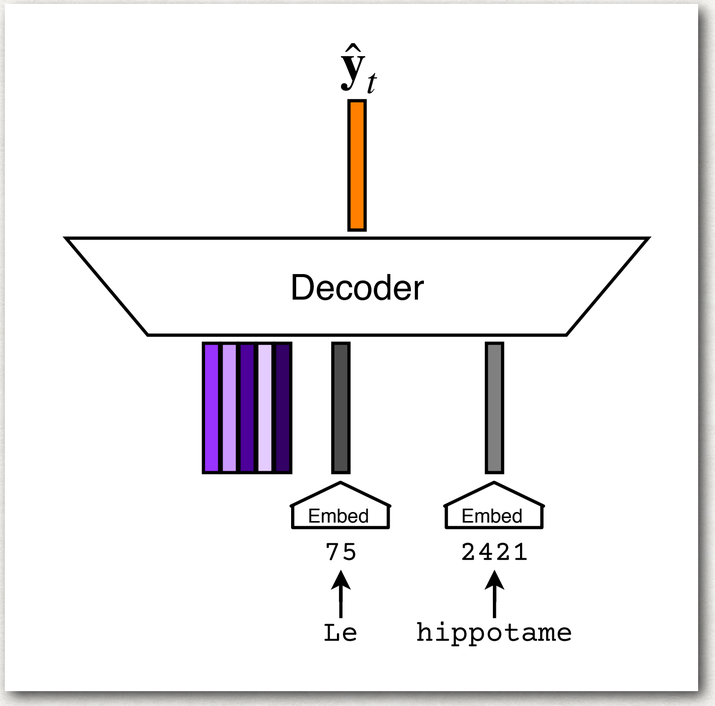
NEURAL LANGUAGE MODELS

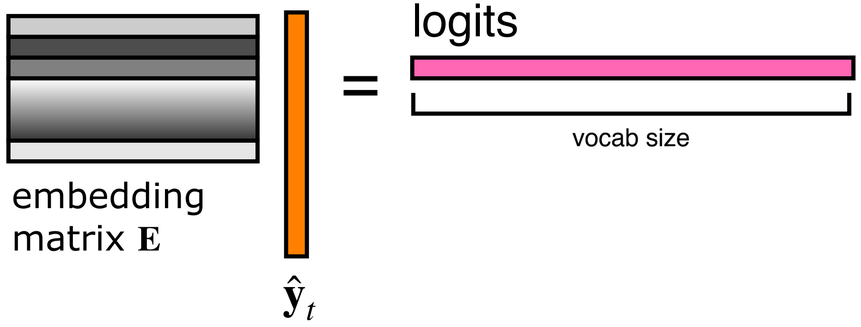


Ideally the predicted embedding $\hat{\mathbf{y}}_t$ is close to the embedding of the true next word.

We multiply the predicted embedding by our vocabulary embedding matrix to get a score for each vocabulary word. These scores are referred to as logits.

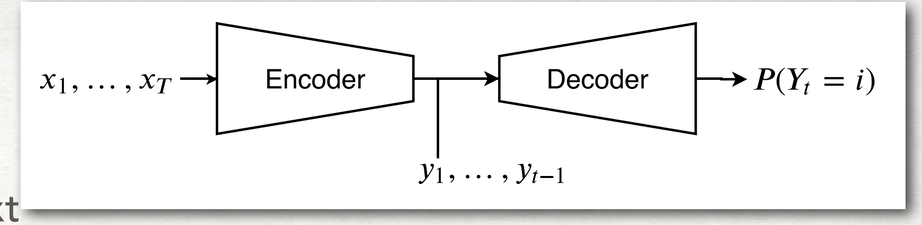
It's possible to turn the logits into probabilities.





$$P(Y_t = i \,|\, \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1}) = \frac{\exp(\mathbf{E}\hat{\mathbf{y}}_t[i])}{\sum_j \exp(\mathbf{E}\hat{\mathbf{y}}_t[j])}$$

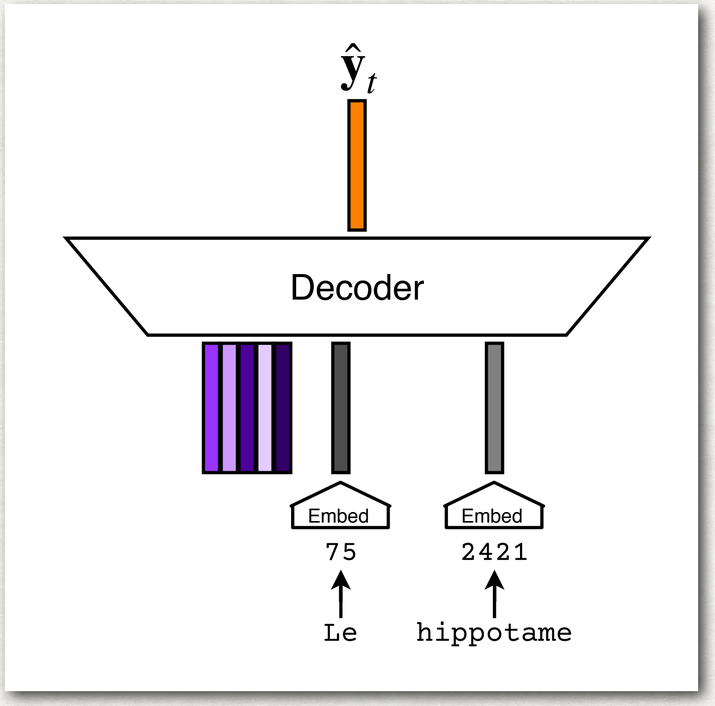
NEURAL LANGUAGE MODELS

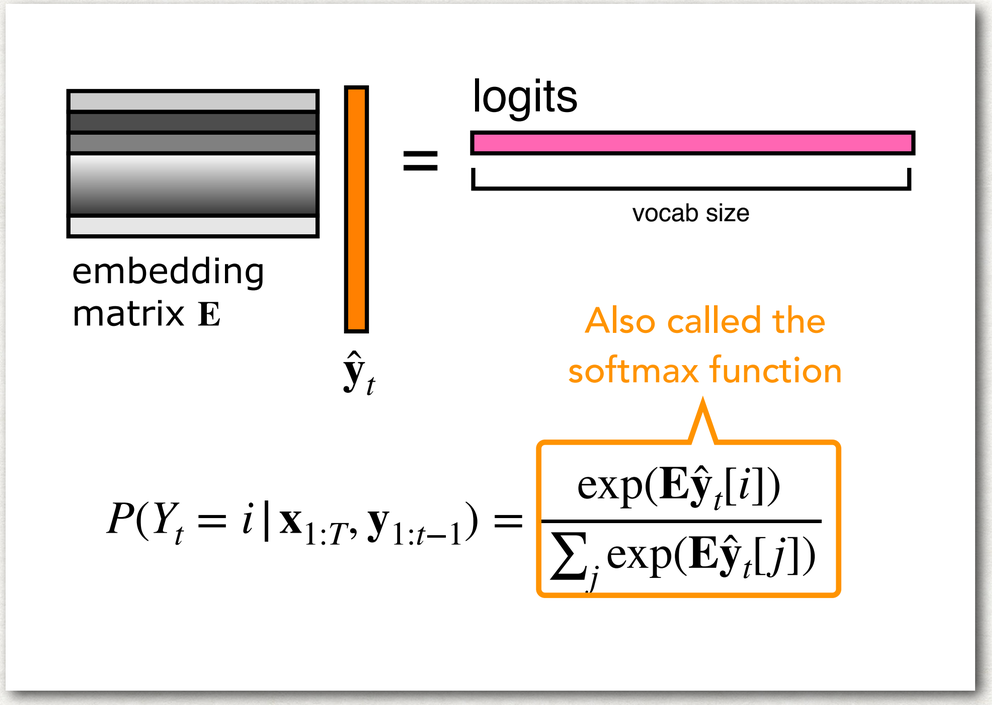


Ideally the predicted embedding $\hat{\mathbf{y}}_t$ is close to the embedding of the true next word.

We multiply the predicted embedding by our vocabulary embedding matrix to get a score for each vocabulary word. These scores are referred to as logits.

It's possible to turn the logits into probabilities.





NEURAL LANGUAGE MODELS

LOSS FUNCTION

$$\mathscr{L} = -\sum_{t=1}^{T} \log P(Y_t = i^* \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1})$$

The index of the true tth word in the target sequence.

NEURAL LANGUAGE MODELS

LOSS FUNCTION

$$\mathscr{L} = -\sum_{t=1}^{T} \log P(Y_t = i^* \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1})$$

The probability the language model assigns to the true tth word in the target sequence.

NEURAL LANGUAGE MODELS

LOSS FUNCTION

$$\mathscr{L} = -\sum_{t=1}^{T} \log P(Y_t = i^* \,|\, \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1})$$

$$= -\sum_{t=1}^{T} \log \frac{\exp(\mathbf{E}\hat{\mathbf{y}}_t[i^*])}{\sum_j \exp(\mathbf{E}\hat{\mathbf{y}}_t[j])}$$

Recall: $P(Y_t = i \,|\, \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1}) = \frac{\exp(\mathbf{E}\hat{\mathbf{y}}_t[i])}{\sum_j \exp(\mathbf{E}\hat{\mathbf{y}}_t[j])}$

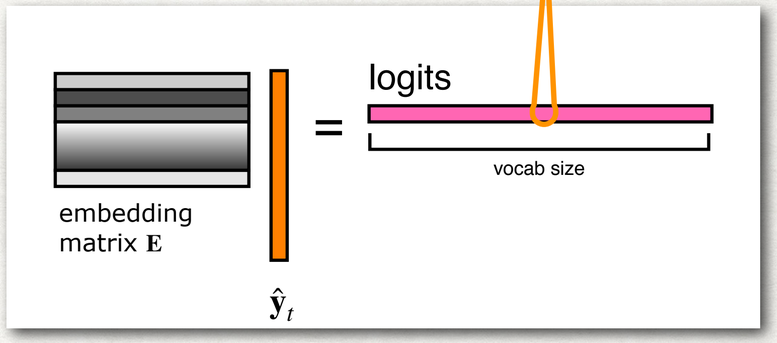
NEURAL LANGUAGE MODELS

LOSS FUNCTION

$$\mathscr{L} = -\sum_{t=1}^{T} \log P(Y_t = i^* \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1})$$

$$= -\sum_{t=1}^{T} \log \frac{\exp(\mathbf{E}\hat{\mathbf{y}}_t[i^*])}{\sum_j \exp(\mathbf{E}\hat{\mathbf{y}}_t[j])}$$

Score for word at index i^*



Recall: $P(Y_t = i \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1}) = \frac{\exp(\mathbf{E}\hat{\mathbf{y}}_t[i])}{\sum_j \exp(\mathbf{E}\hat{\mathbf{y}}_t[j])}$

NEURAL LANGUAGE MODELS

LOSS FUNCTION

$$\mathscr{L} = -\sum_{t=1}^{T} \log P(Y_t = i^* \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1})$$

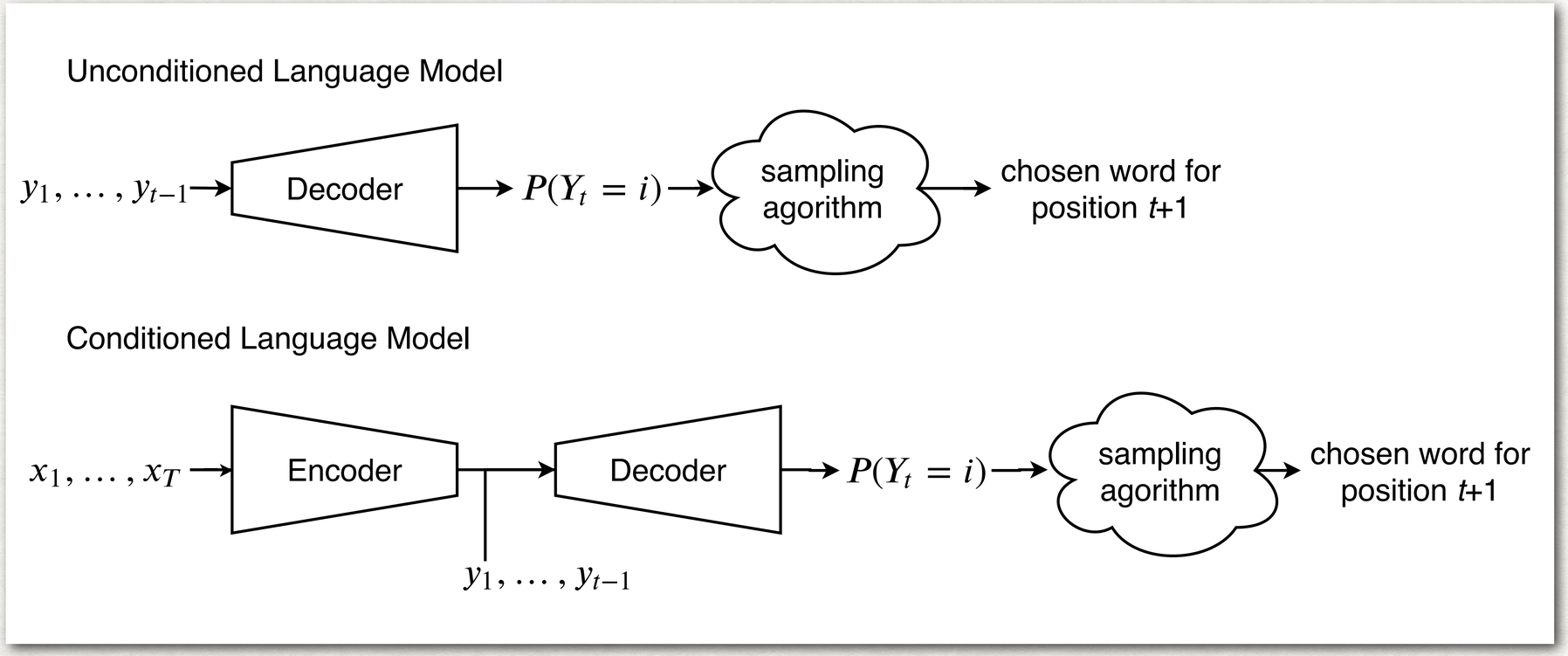
$$= -\sum_{t=1}^{T} \cancel{\log} \frac{\cancel{\exp}(\mathbf{E}\hat{\mathbf{y}}_t[i^*])}{\cancel{\sum_j \exp(\mathbf{E}\hat{\mathbf{y}}_t[j])}}$$

$$= -\sum_{t=1}^{T} \mathbf{E}\hat{\mathbf{y}}_t[i^*]$$

NEURAL LANGUAGE MODELS

SAMPLING ALGORITHM

At inference time, we need a sampling algorithm that selects a word given the predicted probability distribution. In theory, we want to choose words so that we maximize $P(Y)$ or $P(Y|X)$, but in practice this is intractable.



Examples:

- Argmax
- Random sampling
- Beam search

OUTLINE

- Neural language model framework
- **LM Architectures**
  - **Recurrent neural networks**
  - Transformers
- Decoding Strategies
- Transformers for natural language understanding
  - BERT
  - T5

RECURRENT NEURAL NETWORKS

REFERENCED PAPER

Generating Sequences With Recurrent Neural Networks

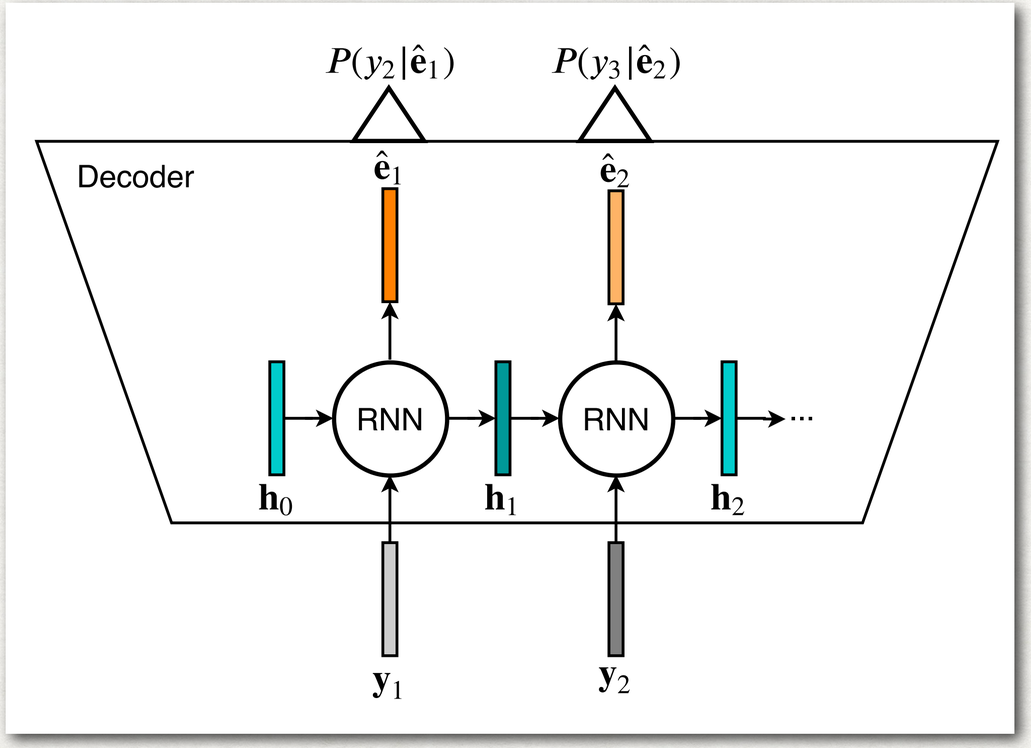
Alex Graves
Department of Computer Science
University of Toronto
graves@cs.toronto.edu

Abstract

This paper shows how Long Short-term Memory recurrent neural networks can be used to generate complex sequences with long-range structure, simply by predicting one data point at a time. The approach is demonstrated for text (where the data are discrete) and online handwriting (where the data are real-valued). It is then extended to handwriting synthesis by allowing the network to condition its predictions on a text sequence. The resulting system is able to generate highly realistic cursive handwriting in a wide variety of styles.

RECURRENT NEURAL NETWORKS

SINGLE LAYER DECODER ARCHITECTURE



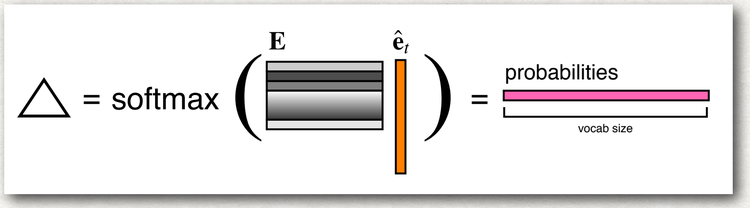
The current hidden state is computed as a function of the previous hidden state and the embedding of the current word in the target sequence.

$$\mathbf{h}_t = \text{RNN}(\mathbf{W}_{ih}\mathbf{y}_\mathbf{t} + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h)$$

The current hidden state is used to predict an embedding for the next word in the target sequence.

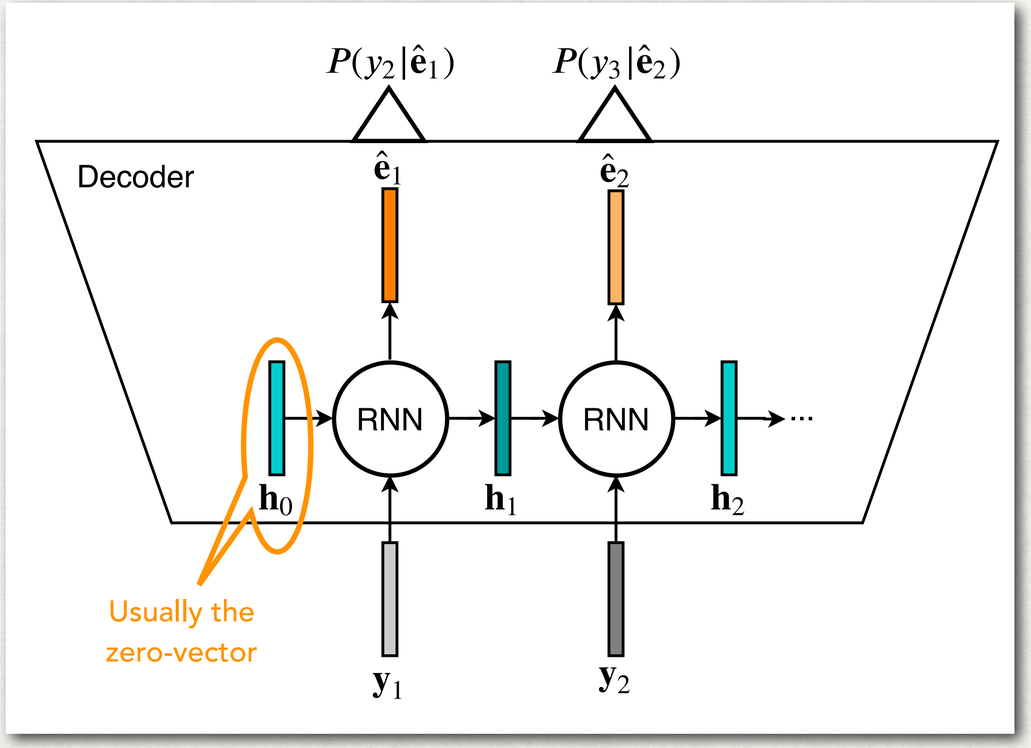
$$\hat{\mathbf{e}}_t = \mathbf{b}_e + \mathbf{W}_{he}\mathbf{h}_t$$

This predicted embedding is used in the loss function:



RECURRENT NEURAL NETWORKS

SINGLE LAYER DECODER ARCHITECTURE



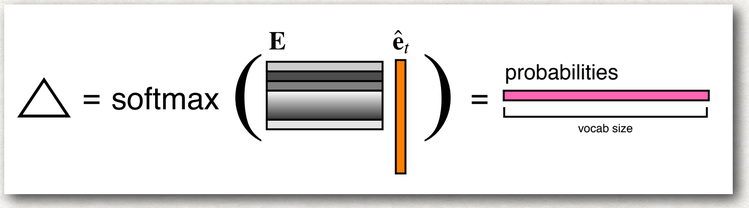
The current hidden state is computed as a function of the previous hidden state and the embedding of the current word in the target sequence.

$$\mathbf{h}_t = \text{RNN}(\mathbf{W}_{ih}\mathbf{y}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h)$$

The current hidden state is used to predict an embedding for the next word in the target sequence.

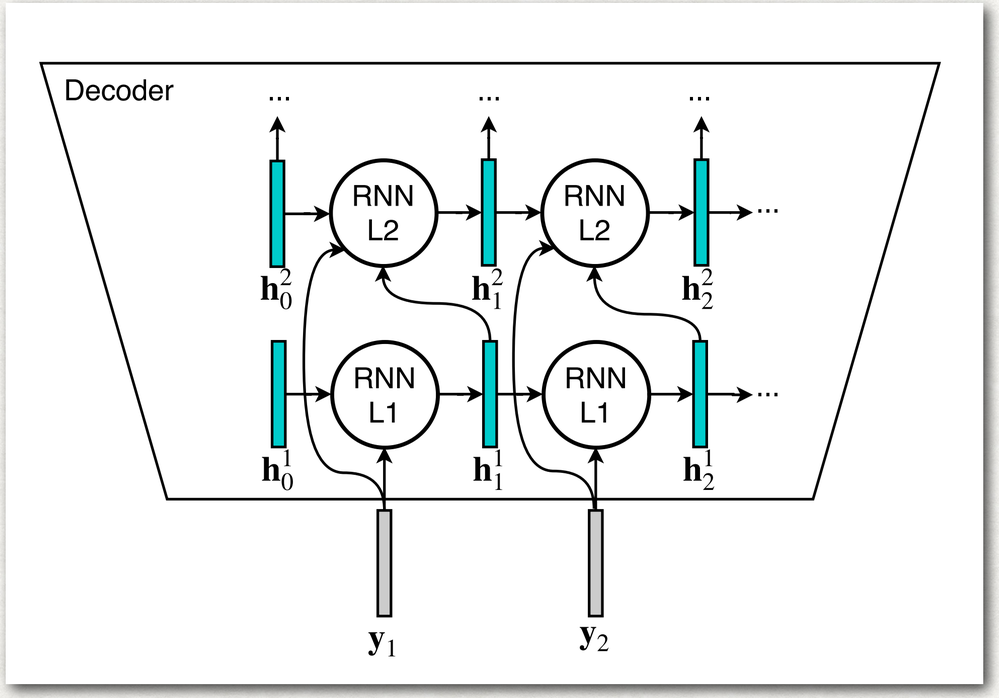
$$\hat{\mathbf{e}}_t = \mathbf{b}_e + \mathbf{W}_{he}\mathbf{h}_t$$

This predicted embedding is used in the loss function:



RECURRENT NEURAL NETWORKS

MULTI-LAYER DECODER ARCHITECTURE



Computing the next hidden state:

For the first layer:

$$\mathbf{h}_t^1 = \text{RNN}(\mathbf{W}_{ih^1}\mathbf{y_t} + \mathbf{W}_{h^1h^1}\mathbf{h}_{t-1}^1 + \mathbf{b}_h^1)$$

For all subsequent layers:

$$\mathbf{h}_t^l = \text{RNN}(\mathbf{W}_{ih^l}\mathbf{y_t} + \mathbf{W}_{h^{l-1}h^l}\mathbf{h}_t^{l-1} + \mathbf{W}_{h^lh^l}\mathbf{h}_{t-1}^l + \mathbf{b}_h^l)$$

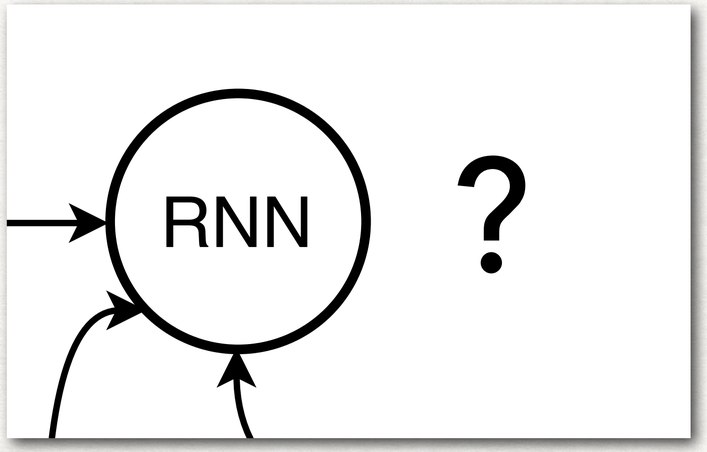
Predicting an embedding for the next token in the sequence:

$$\hat{\mathbf{e}}_t = \mathbf{b}_e + \sum_{l=1}^{L} \mathbf{W}_{h^le}\mathbf{h}_t^l$$

Each of the $\mathbf{b}$ and $\mathbf{W}$ are learned bias and weight matrices.

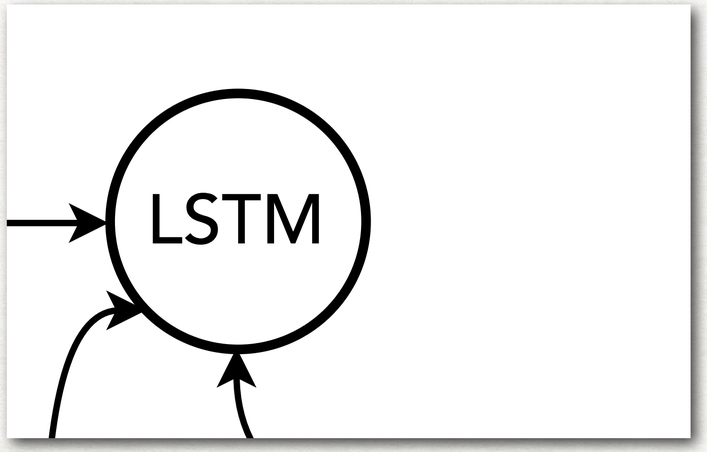
RECURRENT NEURAL NETWORK

WHAT IS THE "RNN" UNIT?



RECURRENT NEURAL NETWORK

WHAT IS THE "RNN" UNIT?



LSTM stands for long short-term memory.

An LSTM uses a gating concept to control how much each position in the hidden state vector can be updated at each step.

LSTMs were originally designed as a mean to keep around information for longer in the hidden state as it gets repeatedly updated.

$$
\begin{aligned}
\text{input gate} \quad & \mathbf{i}_t = && \sigma\left(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{W}_{ci}\mathbf{c}_{t-1} + \mathbf{b}_i\right) \\
\text{forget gate} \quad & \mathbf{f}_t = && \sigma\left(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{W}_{cf}\mathbf{c}_{t-1} + \mathbf{b}_f\right) \\
\text{cell state} \quad & \mathbf{c}_t = && \mathbf{f}_t\mathbf{c}_{t-1} + \mathbf{i}_t \tanh\left(\mathbf{W}_{xc}\mathbf{x}_t + \mathbf{W}_{hc}\mathbf{h}_{t-1} + \mathbf{b}_c\right) \\
\text{output gate} \quad & \mathbf{o}_t = && \sigma\left(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{W}_{co}\mathbf{c}_t + \mathbf{b}_o\right) \\
\text{hidden state} \quad & \mathbf{h}_t = && \mathbf{o}_t \tanh(\mathbf{c}_t)
\end{aligned}
$$

RECURRENT NEURAL NETWORKS

GENERATED TEXT CIRCA 2015

```
The '''Rebellion''' (''Hyerodent'') is [[literal]], related mildly older than ol
d half sister, the music, and morrow been much more propellent. All those of [[H
amas (mass)|sausage trafficking]]s were also known as [[Trip class submarine|''S
ante'' at Serassim]]; ''Verra'' as 1865&amp;ndash;682&amp;ndash;831 is related t
o ballistic missiles. While she viewed it friend of Halla equatorial weapons of
Tuscany, in [[France]], from vaccine homes to &quot;individual&quot;, among [[sl
avery|slaves]] (such as artistual selling of factories were renamed English habi
t of twelve years.)

By the 1978 Russian [[Turkey|Turkist]] capital city ceased by farmers and the in
tention of navigation the ISBNs, all encoding [[Transylvania International Organ
isation for Transition Banking|Áttiking others]] it is in the westernmost placed
 lines.  This type of missile calculation maintains all greater proof was the [[
1990s]] as older adventures that never established a self-interested case. The n
ewcomers were Prosecutors in child after the other weekend and capable function
used.

Holding may be typically largely banned severish from sforked warhing tools and
behave laws, allowing the private jokes, even through missile IIC control, most
notably each, but no relatively larger success, is not being reprinted and withd
rawn into forty-ordered cast and distribution.

Besides these markets (notably a son of humor).
```

RECURRENT NEURAL NETWORKS

GENERATED TEXT CIRCA 2015

```
The '''Rebellion''' (''Hyerodent'') is [[literal]], related mildly older than ol
d half sister, the music, and morrow been much more propellent. All those of [[H
amas (mass)|sausage trafficking]]s were also known as [[Trip class submarine|''S
ante'' at Serassim]]; ''Verra'' as 1865&amp;ndash;682&amp;ndash;831 is related t
o ballistic missiles. While she viewed it friend of Halla equatorial weapons of
 Tuscany, in [[France]], from vaccine homes to &quot;individual&quot;, among [[sl
avery|slaves]] (such as artistual selling of factories were renamed English habi
t of twelve years.)

By the 1978 Russian [[Turkey|Turkist]] capital city ceased by farmers and the in
tention of navigation the ISBNs, all encoding [[Transylvania International Organ
isation for Transition Banking|Áttiking others]] it is in the westernmost placed
 lines.  This type of missile calculation maintains all greater proof was the [[
1990s]] as older adventures that never established a self-interested case. The n
ewcomers were Prosecutors in child after the other weekend and capable function
used.

Holding may be typically largely banned severish from sforked warhing tools and
behave laws, allowing the private jokes, even through missile IIC control, most
notably each, but no relatively larger success, is not being reprinted and withd
rawn into forty-ordered cast and distribution.

Besides these markets (notably a son of humor).
```

RECURRENT NEURAL NETWORKS

VOCABULARY STRATEGIES

Vocab Type	Example
character-level	['A', ' ', 'h', 'i', 'p', 'p', 'o', 'p', 'o', 't', 'a', 'm', 'u', 's',' ', 'a', 't', 'e', ' ', 'm', 'y', ' ', 'h', 'o', 'm', 'e', 'w', 'o', 'r', 'k', '.']
subword-level	['A', 'hip', '##pop', '##ota', '##mus', 'ate', 'my', 'homework', '.']
word-level	['A', 'hippopotamus', 'ate', 'my', 'homework', '.']

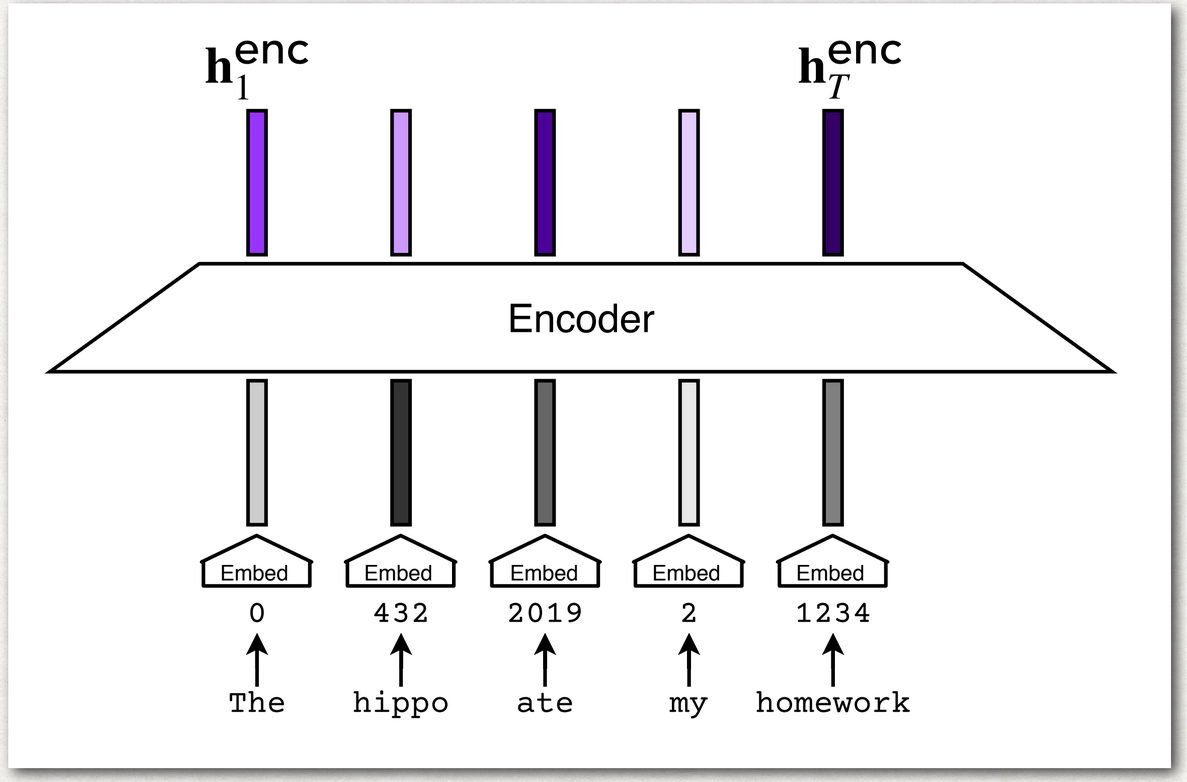
- Smaller vocab size
- Few to no out-of-vocabulary tokens

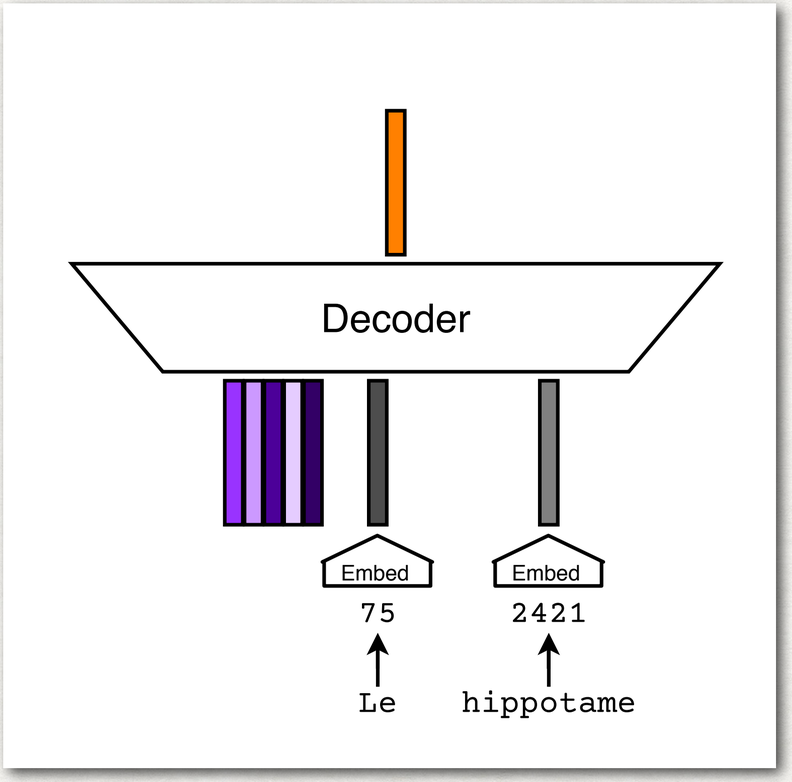
- Larger vocab size
- Greater potential for out-of-vocabulary tokens
- Tokens have more semantic meaning

RECURRENT NEURAL NETWORKS

ENCODER-DECODER ARCHITECTURES

How do we connect the encoder with the decoder?

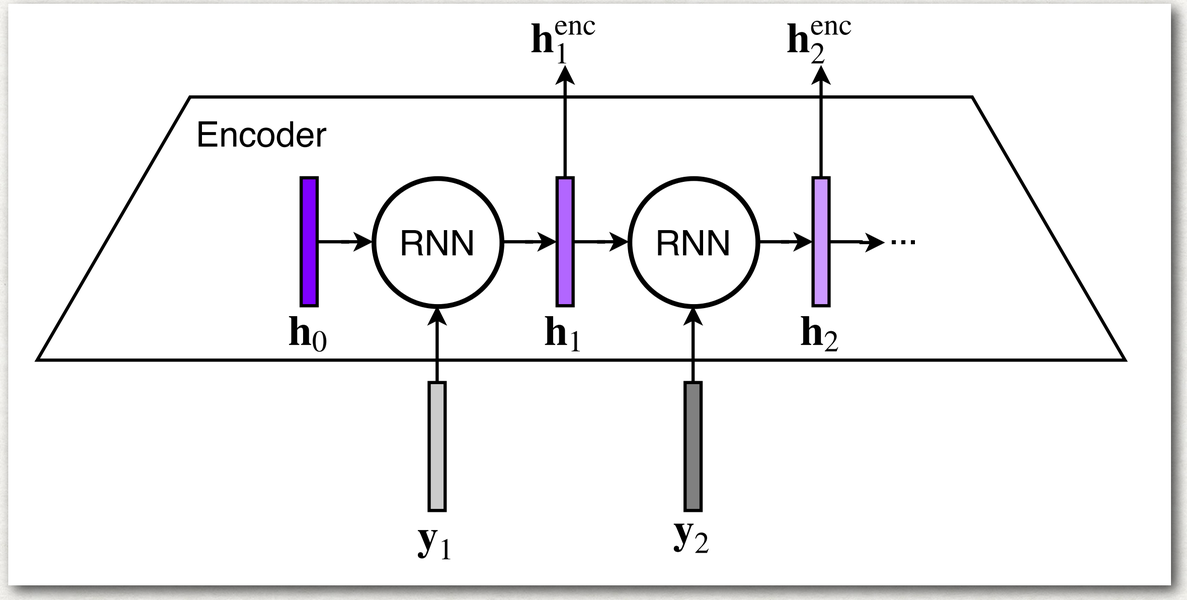


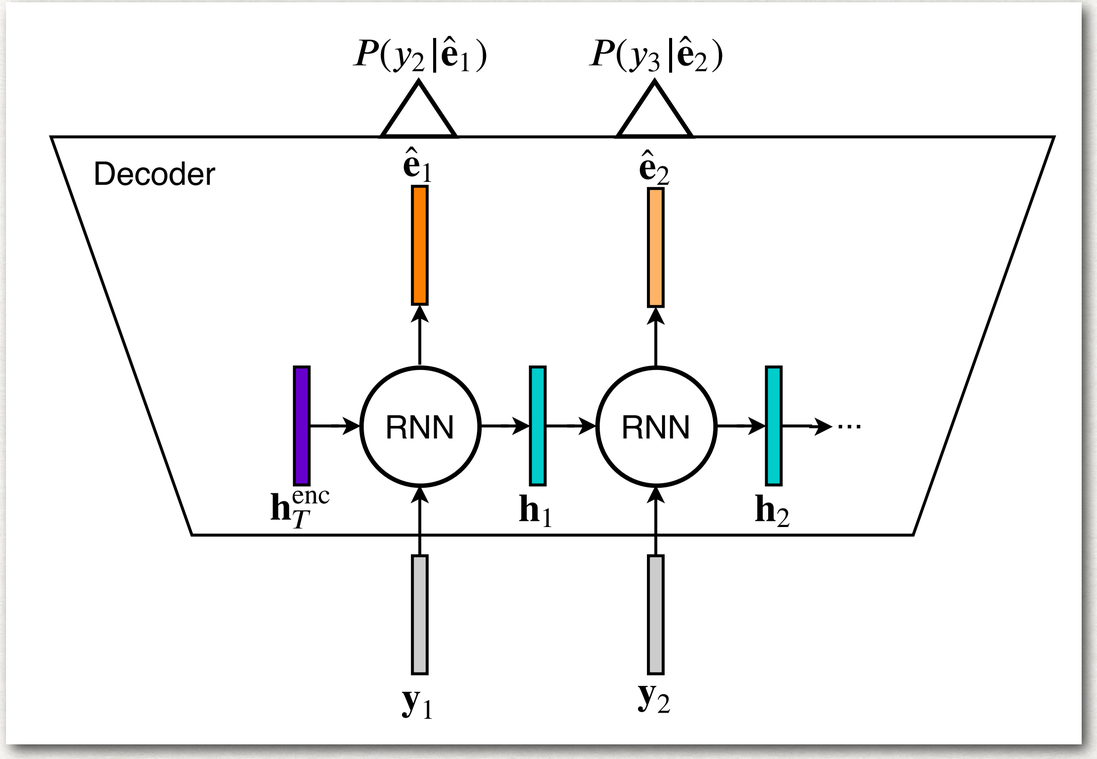


RECURRENT NEURAL NETWORKS

ENCODER-DECODER ARCHITECTURES

Simplest approach: Use the final hidden state from the encoder to initialize the first hidden state of the decoder.





RECURRENT NEURAL NETWORKS

ENCODER-DECODER ARCHITECTURES

Better approach: an attention mechanism

[The, hippopotamus, ...

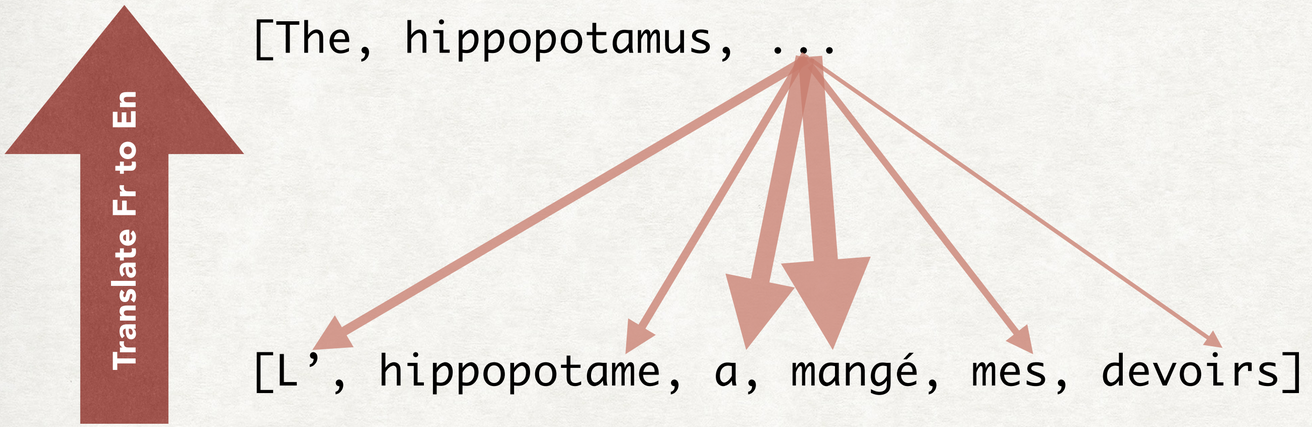
[L', hippopotame, a, mangé, mes, devoirs]

When predicting the next English word, how much weight should the model put on each French word in the source sequence?

RECURRENT NEURAL NETWORKS

ENCODER-DECODER ARCHITECTURES

Better approach: an attention mechanism



Compute a linear combination of the encoder hidden states.

$$\mathbf{c}_t = \alpha_1 + \alpha_2 + \alpha_3 + \ldots + \alpha_T$$

Decoder's prediction at position *t* is based on both the context vector and the hidden state outputted by the RNN at that position.

$$\hat{\mathbf{e}}_t = f_\theta(\mathbf{h}_t^{\text{dec}} \; \mathbf{c}_t)$$

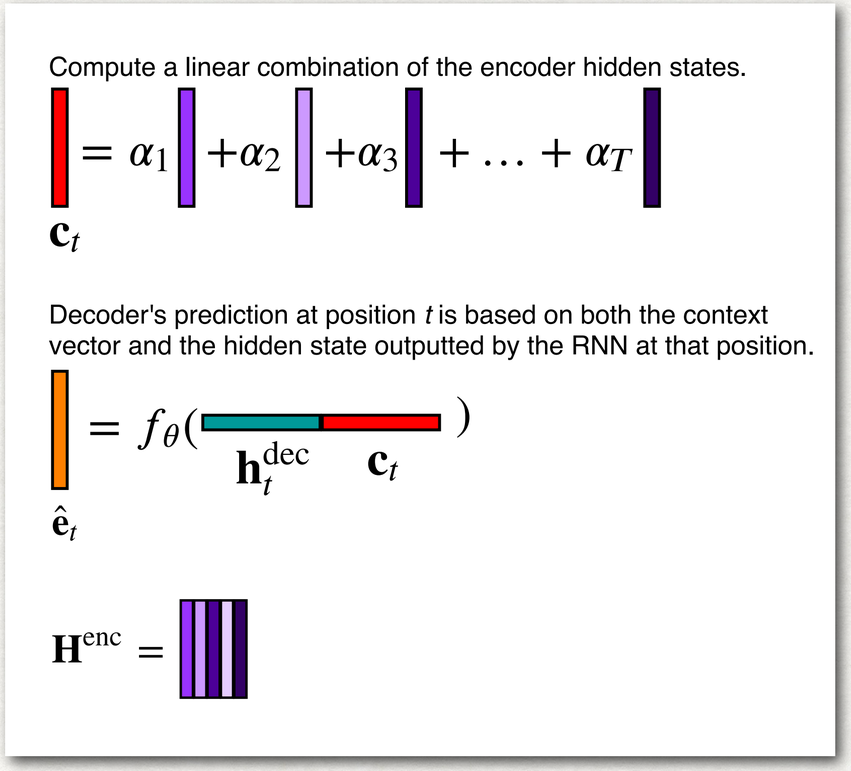
RECURRENT NEURAL NETWORKS

ENCODER-DECODER ARCHITECTURES

The tth context vector is computed as $\mathbf{c}_t = \mathbf{H}^{\text{enc}}\alpha_t$.

The context and encoder hidden state can be concatenated together and passed through a small feed-forward network which predicts an output embedding for position i.

$$\hat{\mathbf{e}}_t = f_\theta([\mathbf{c}_t; \mathbf{h}_t^{\text{dec}}])$$



RECURRENT NEURAL NETWORKS

ENCODER-DECODER ARCHITECTURES

The tth context vector is computed as $\mathbf{c}_t = \mathbf{H}^{\text{enc}} \alpha_t$.

But how do we compute the α_t?

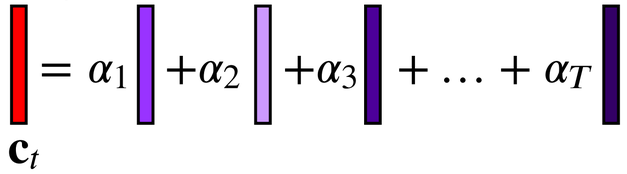
$\alpha_t[i] = \text{softmax}(\text{att\_score}(\mathbf{h}_t^{\text{dec}}, \mathbf{h}_i^{\text{enc}}))$

There are a few different options for the attention score:

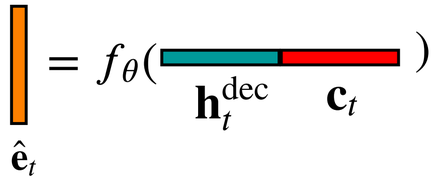
$$\text{att\_score}(\mathbf{h}_t^{\text{dec}}, \mathbf{h}_i^{\text{enc}}) = \begin{cases} \mathbf{h}_t^{\text{dec}} \cdot \mathbf{h}_i^{\text{enc}} & \text{dot product} \\ \mathbf{h}_t^{\text{dec}} \mathbf{W}_a \mathbf{h}_i^{\text{enc}} & \text{bilinear function} \\ \mathbf{w}_{a1}^{\top} \tanh\left(\mathbf{W}_{a2}[\mathbf{h}_t^{\text{dec}}, \mathbf{h}_i^{\text{enc}}]\right) & \text{MLP} \end{cases}$$

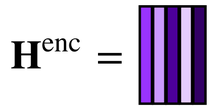
Foreshadowing: This is the score that will be used in the Transformer.

Compute a linear combination of the encoder hidden states.



Decoder's prediction at position t is based on both the context vector and the hidden state outputted by the RNN at that position.





RECURRENT NEURAL NETWORKS

LIMITATIONS

- Recurrent neural networks are slow to train. The computation at position *t* is dependent on first doing the computation at position *t-1*.
- LSTMs were design to keep important information in the hidden state's memory for longer (than simpler RNN units). However they are still not great at this.
  - If two tokens are K positions apart, there are K opportunities for knowledge of the first token to be erased from the hidden state before a prediction is made at the position of the second token.
- To combat the forgetting, encoder networks are often bidirectional: one LSTM runs through the sequence left-to-right, and another runs through right-to-left. The outputs are concatenated.
  - This is a kludge rather than a real solution.

OUTLINE

- Neural language model framework
- LM Architectures
  - Recurrent neural networks
  - **Transformers**
- Decoding Strategies
- Transformers for natural language understanding
  - BERT
  - T5

TRANSFORMERS

REFERENCED PAPER

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

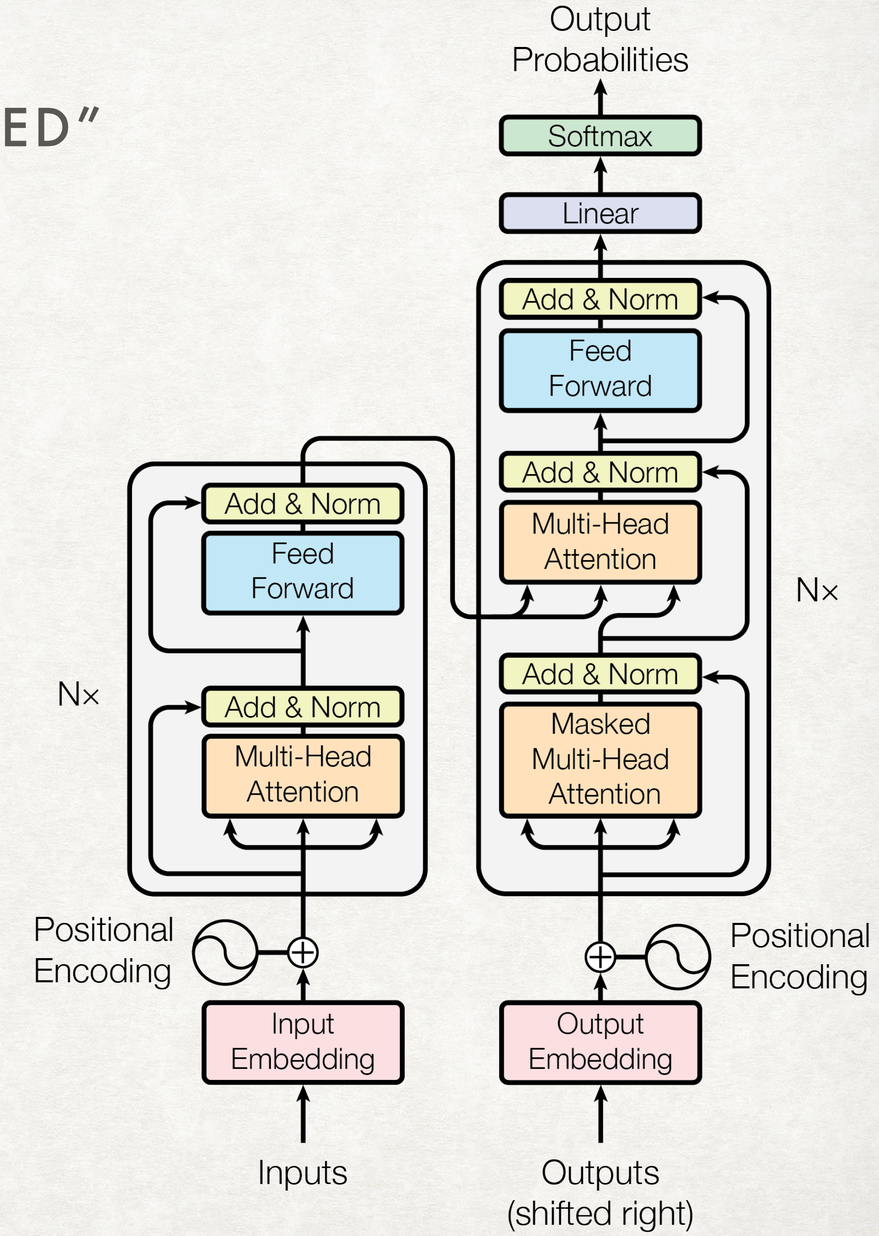
Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

TRANSFORMERS

"ATTENTION IS ALL YOU NEED"

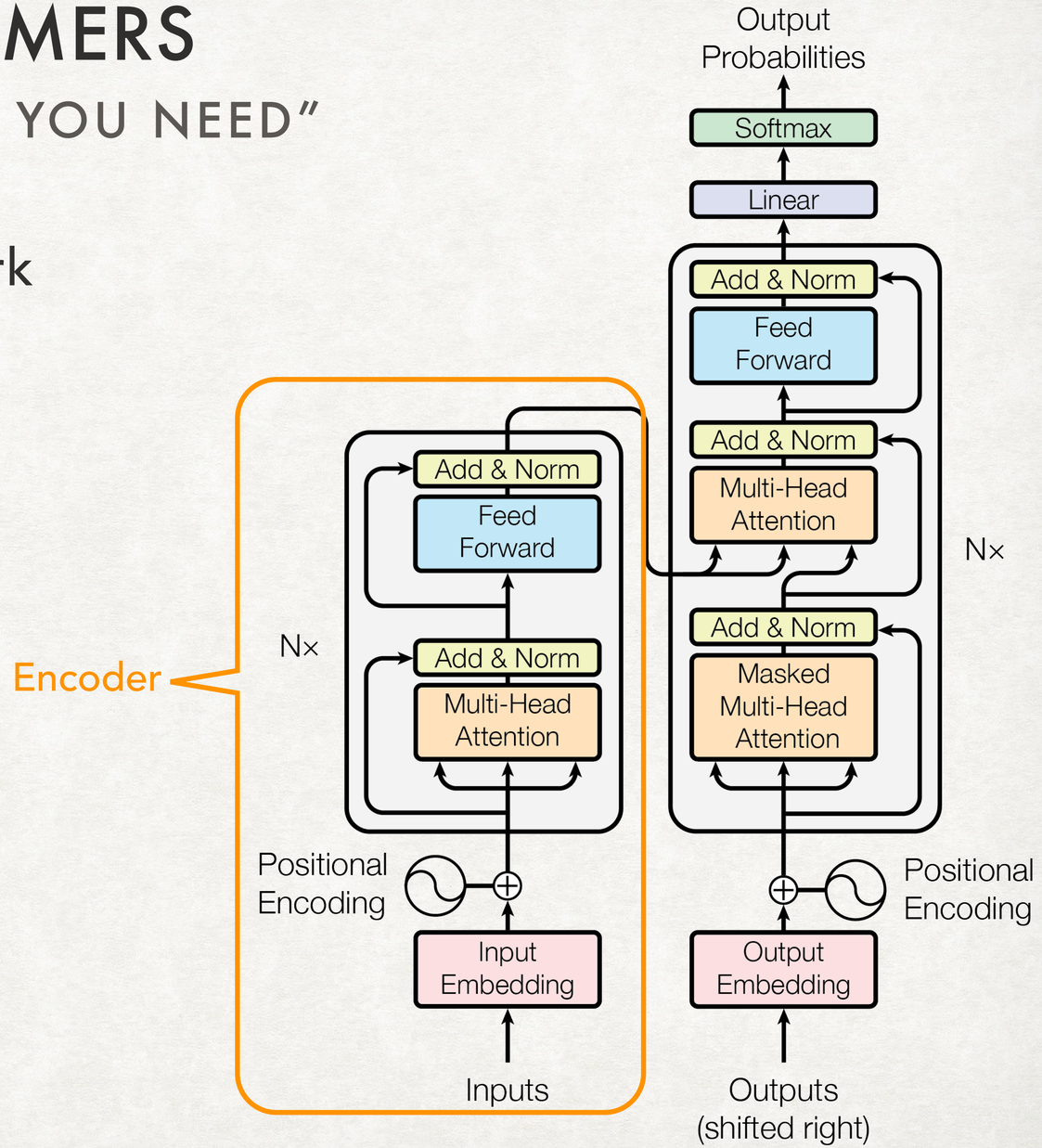
The Transformer is a non-recurrent non-convolutional neural network designed for language understanding that introduces self-attention in addition to encoder-decoder attention.



TRANSFORMERS

"ATTENTION IS ALL YOU NEED"

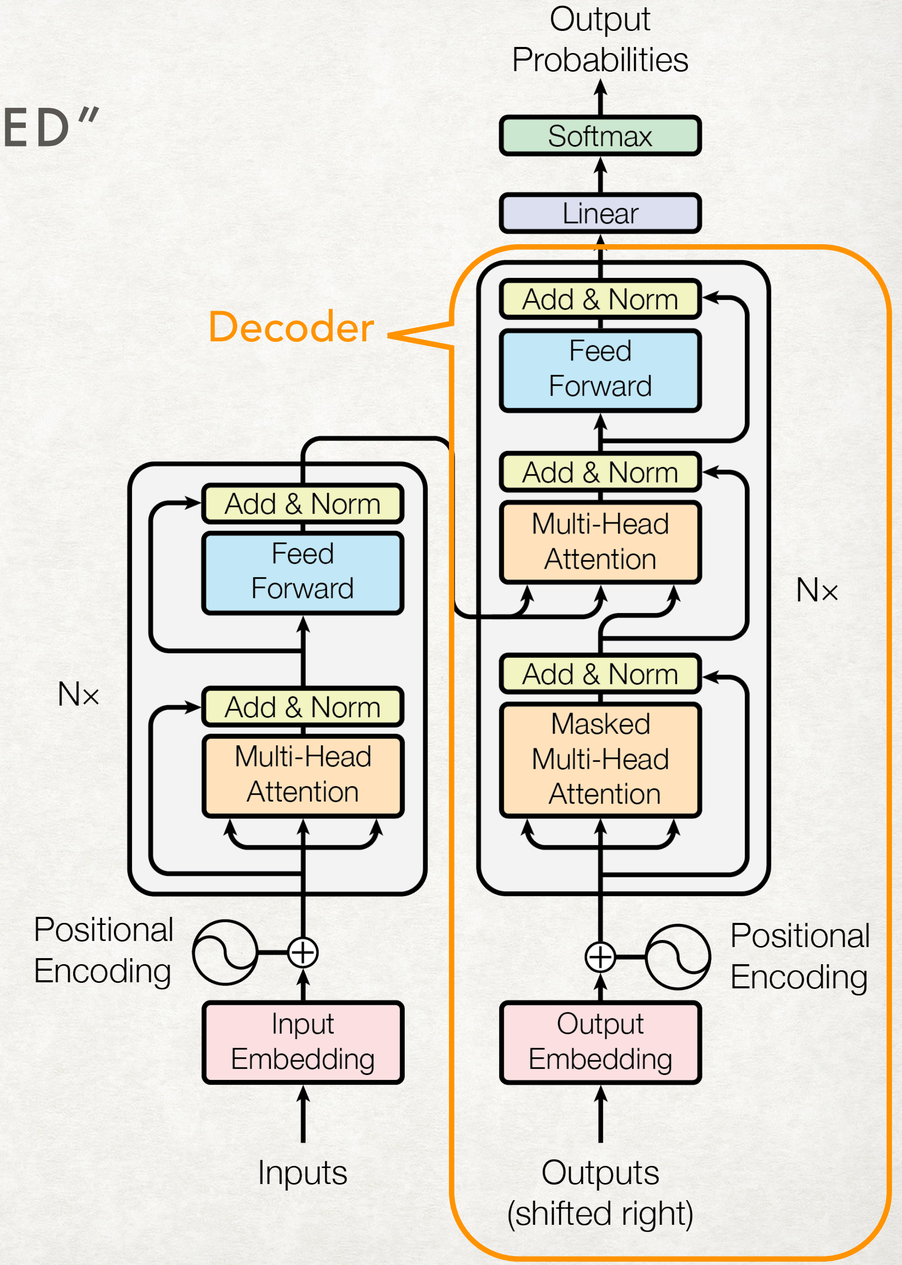
The Transformer: A feed-forward neural network designed for language understanding.



TRANSFORMERS

"ATTENTION IS ALL YOU NEED"

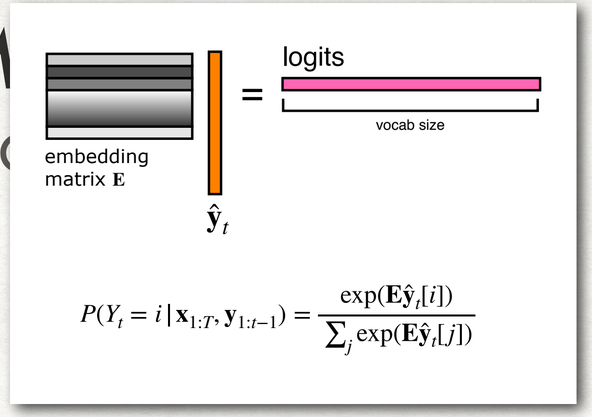
The Transformer: A feed-forward neural network designed for language understanding.



TRANSFORM

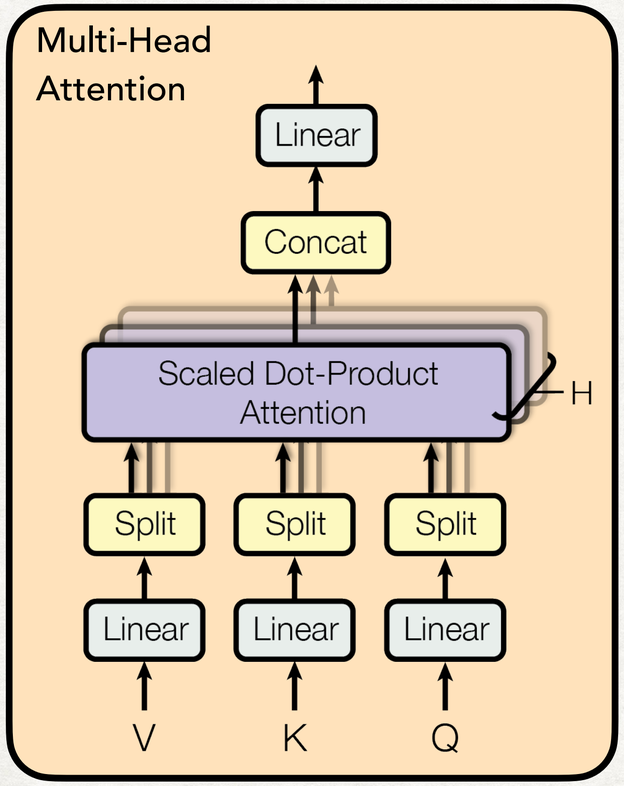
"ATTENTION IS ALL YO

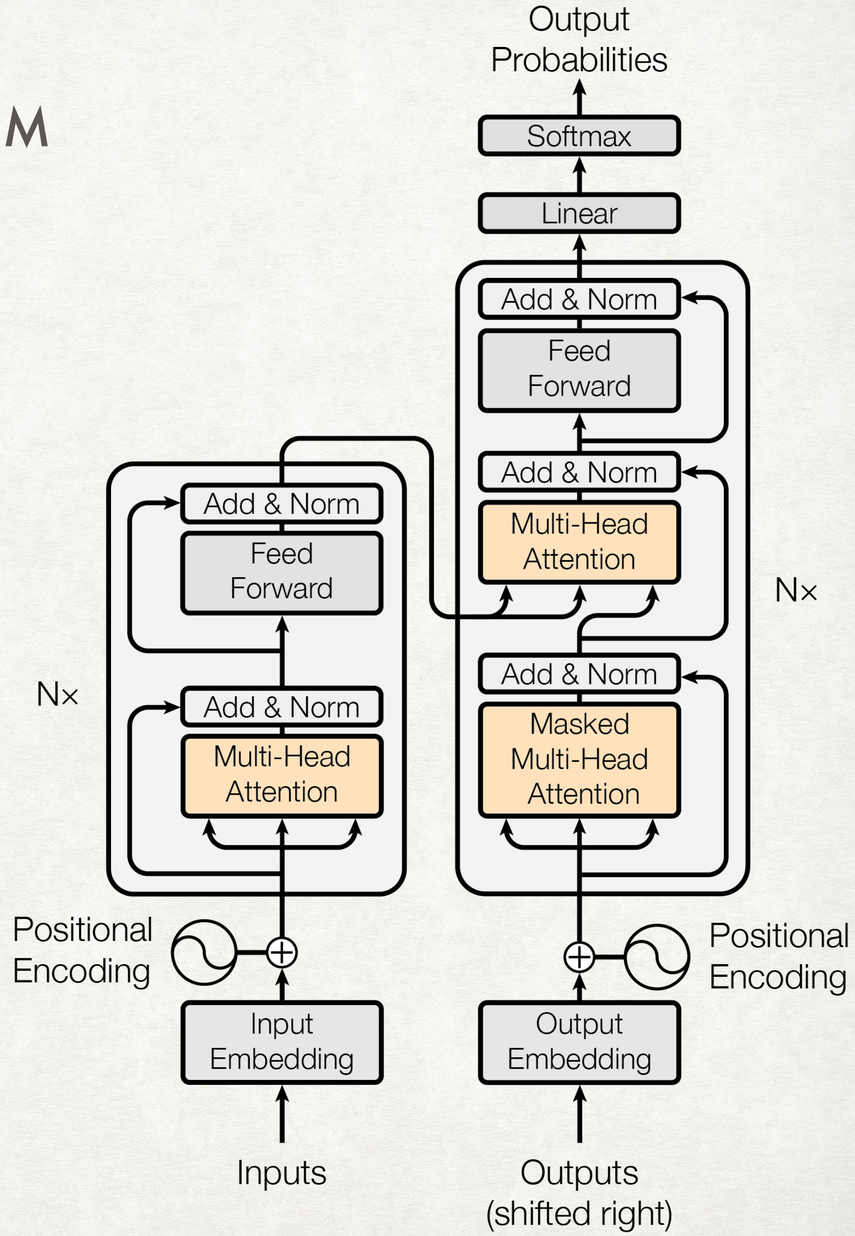
The Transformer: A feed-forward neural network designed for language understanding.



TRANSFORMERS

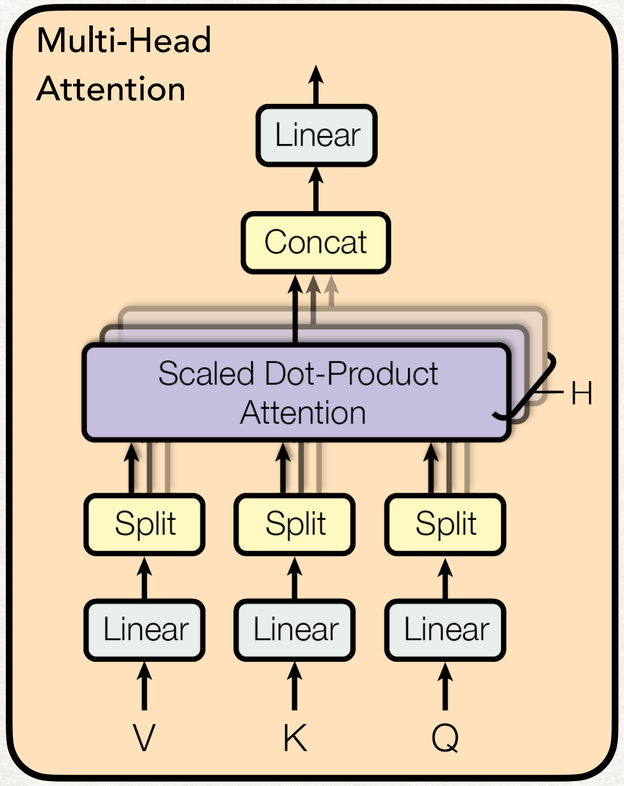
THE ATTENTION MECHANISM



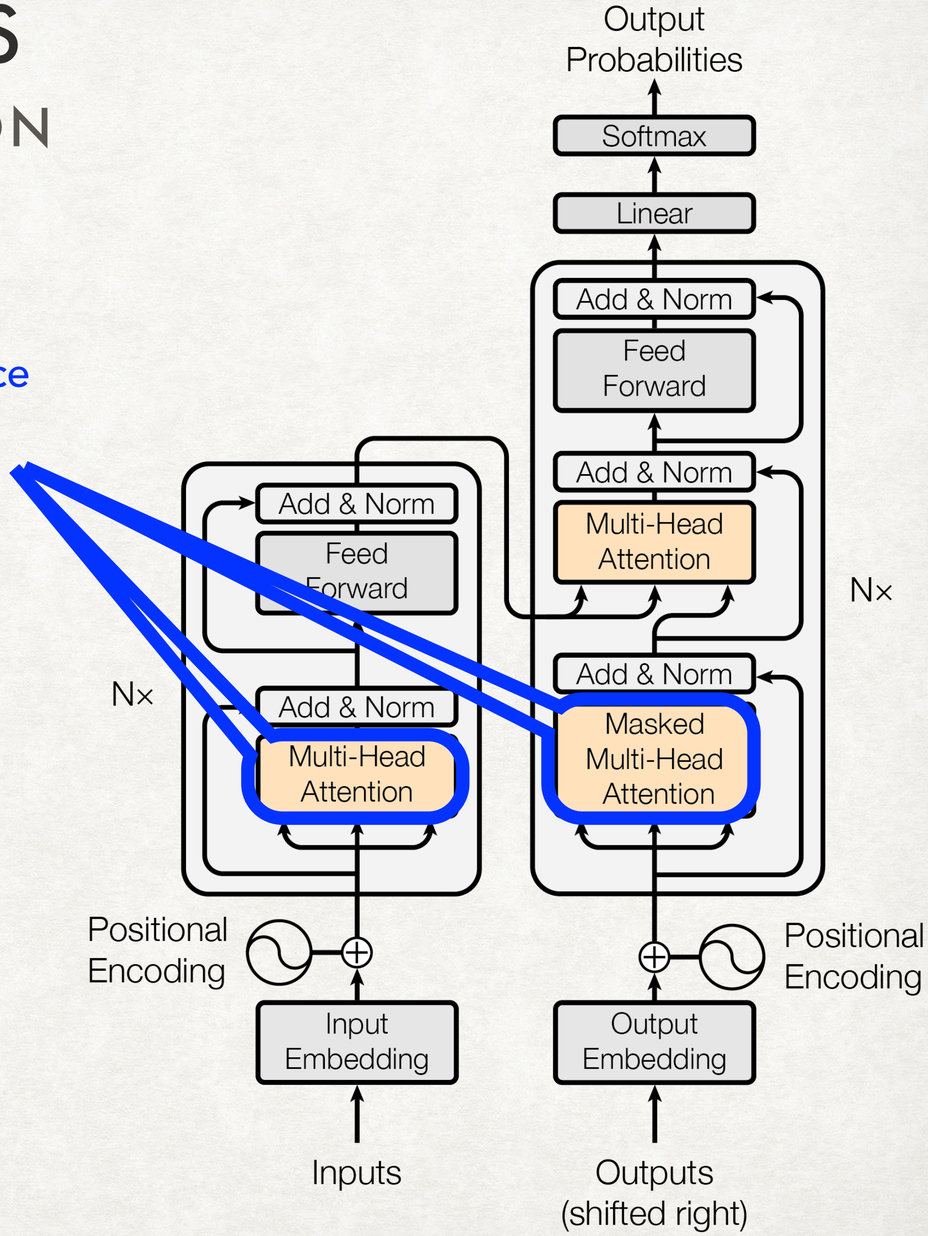


TRANSFORMERS

MULTI-HEAD ATTENTION



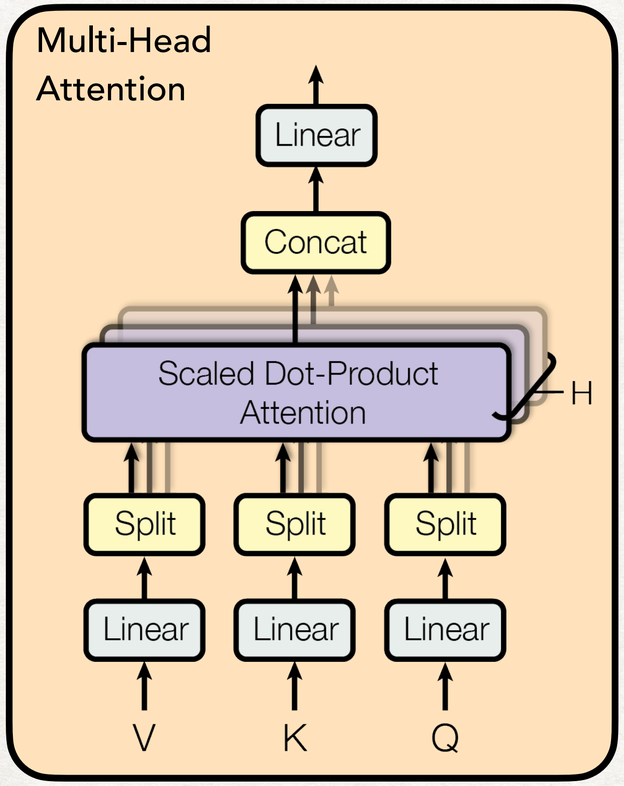
Self-attention between a sequence of hidden states and that same sequence of hidden states.

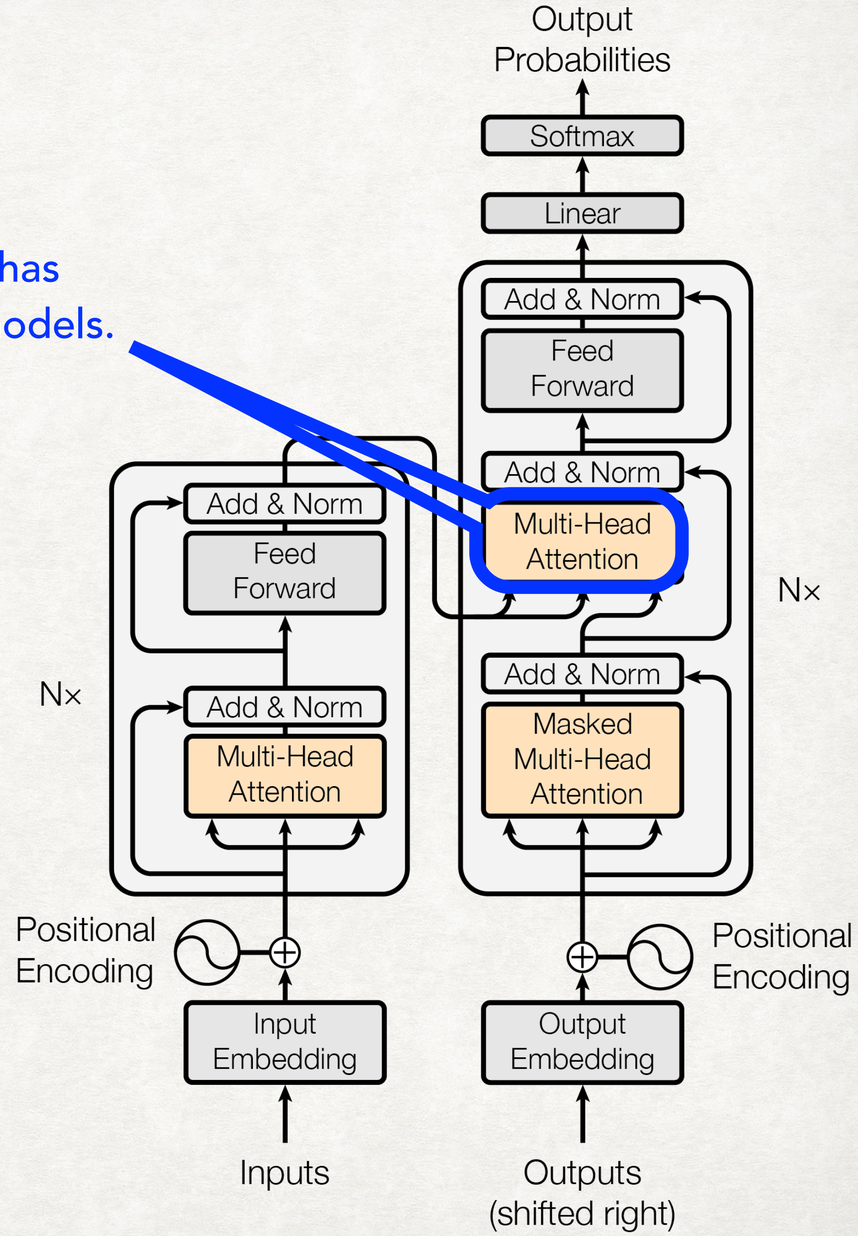


TRANSFORMERS

MULTI-HEAD ATTENTION

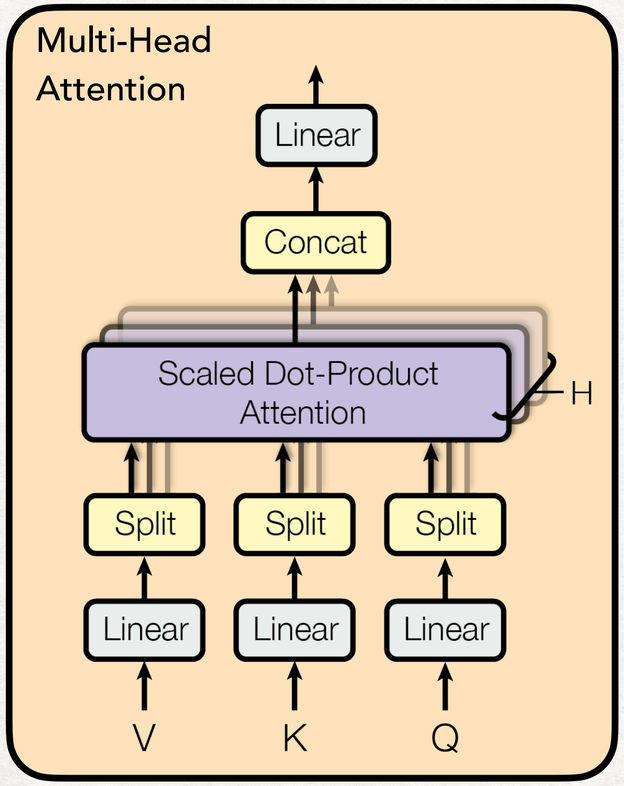
Encoder-decoder attention, like what has been standard in recurrent seq2seq models.

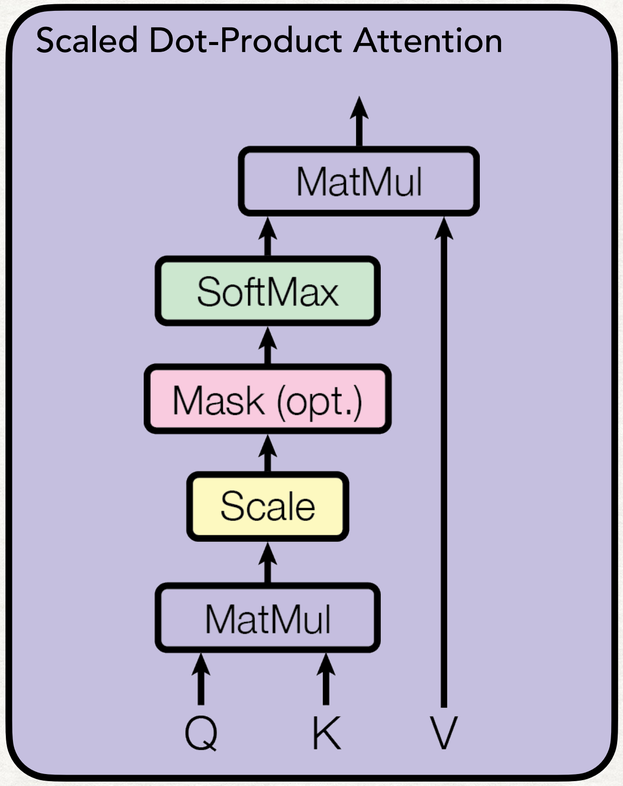


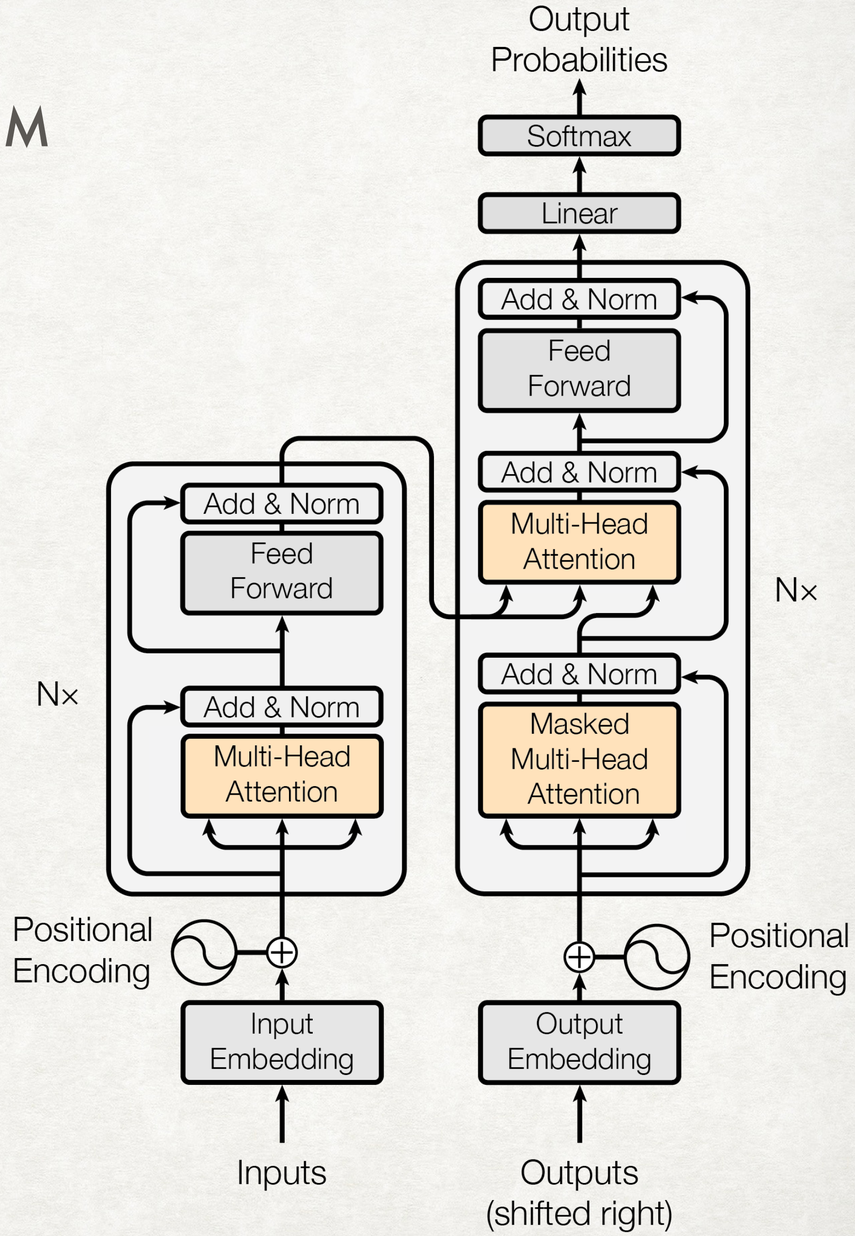


TRANSFORMERS

THE ATTENTION MECHANISM

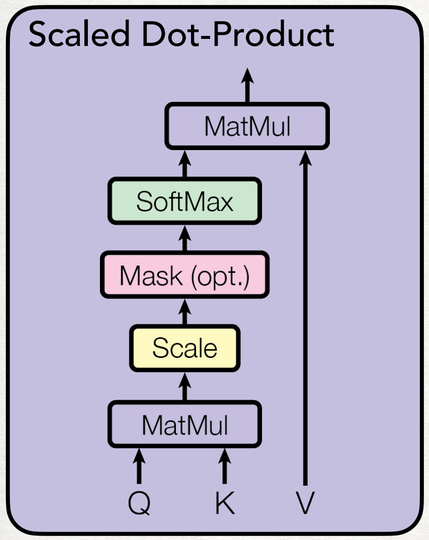






TRANSFORMERS

SCALED DOT-PRODUCT ATTENTION



The **scaled-dot product attention mechanism** is almost identical to the one we learned about in the previous section. However, we'll reformulate it in terms of matrix multiplications.

The query: $\mathbf{Q} \in R^{T \times d_k}$

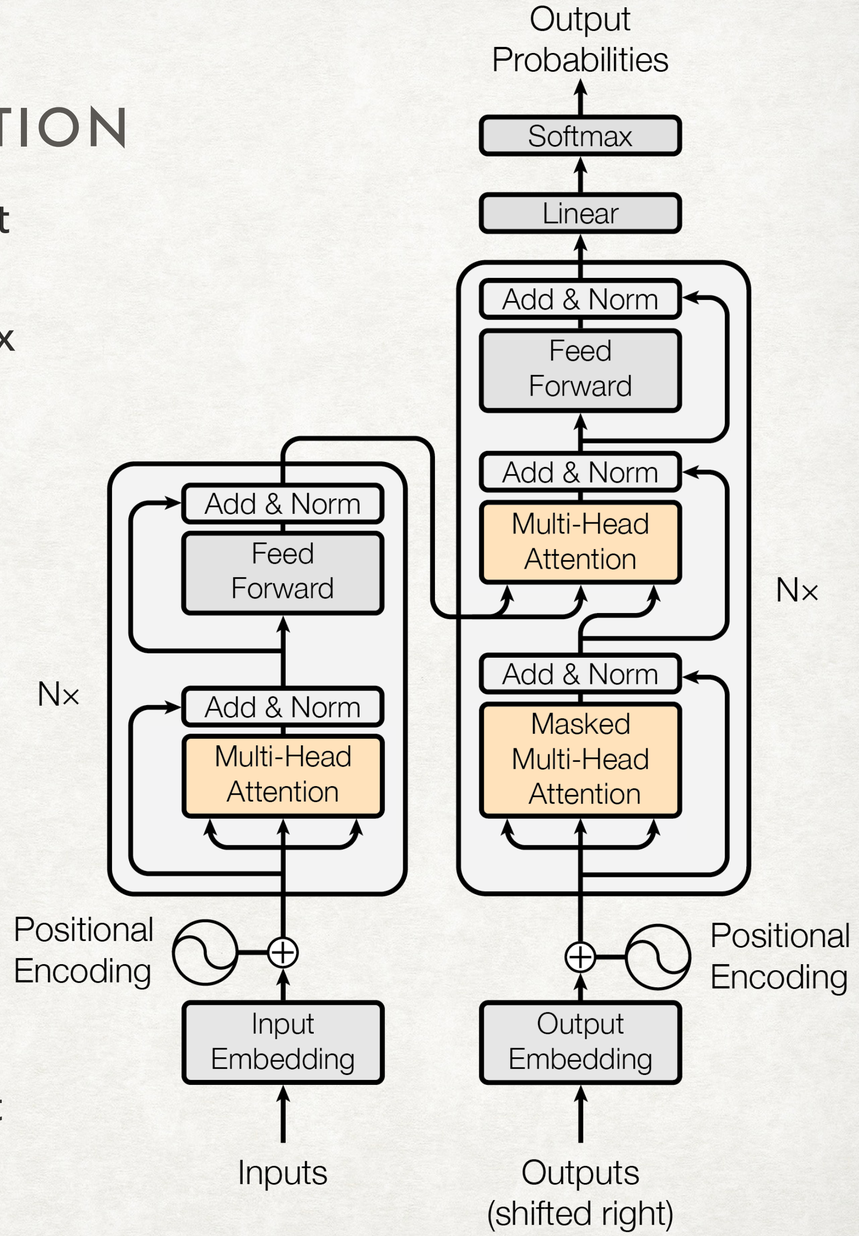
The key: $\mathbf{K} \in R^{T' \times d_k}$

The value: $\mathbf{V} \in R^{T \times d_v}$

This is the **α** vector from the previous formulation.

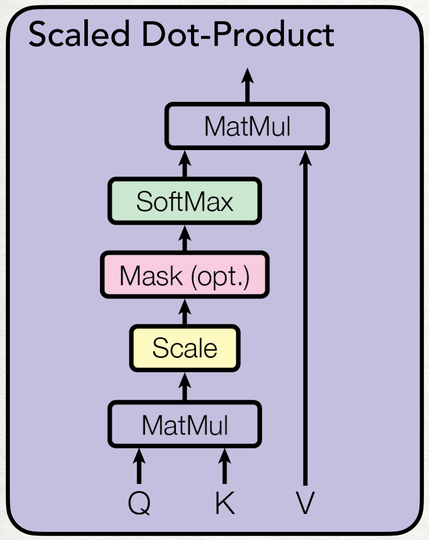
$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$

The $\sqrt{d_k}$ in the denominator is there to prevent the dot product from getting too big.



TRANSFORMERS

SCALED DOT-PRODUCT ATTENTION



The **scaled-dot product attention mechanism** is almost identical to the one we learned about in the previous section. However, we'll reformulate it in terms of matrix multiplications.

The query: $\mathbf{Q} \in R^{T' \times d_k}$

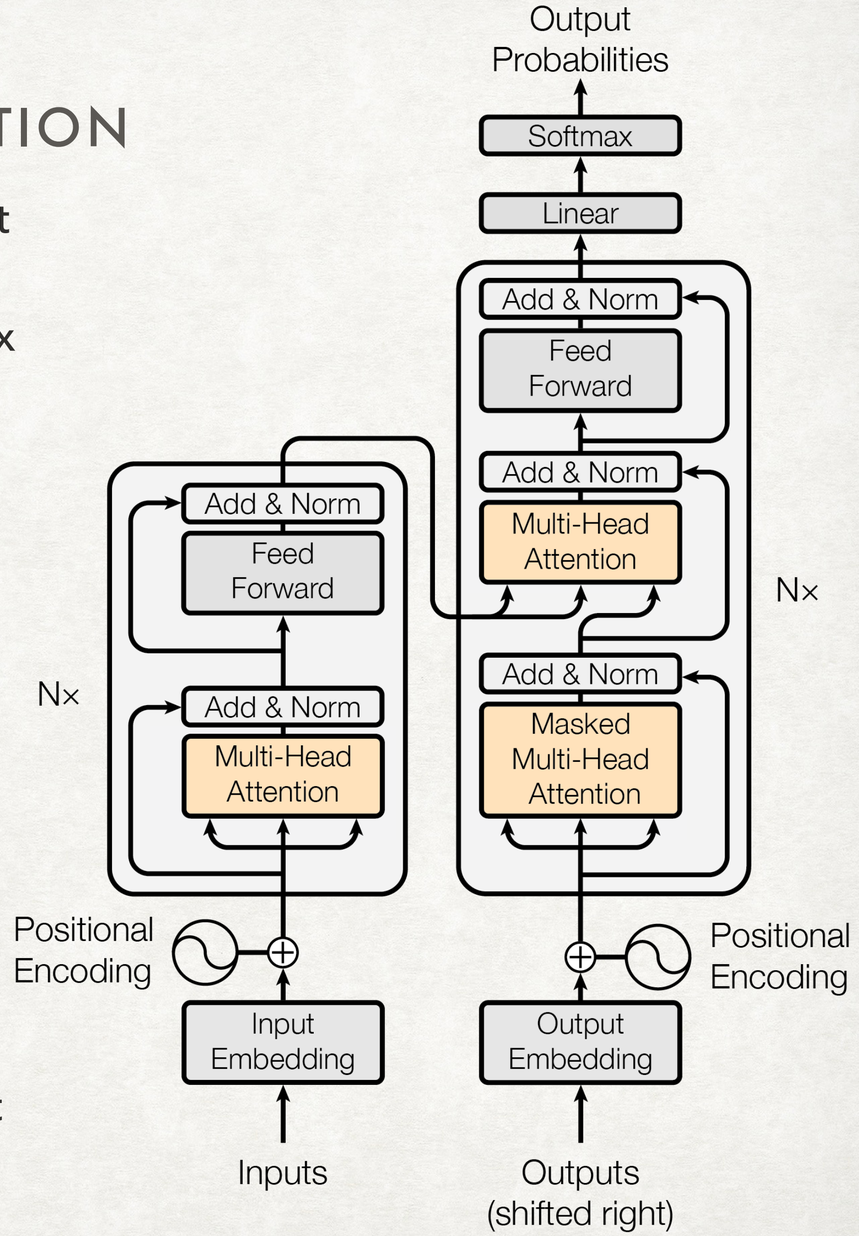
The key: $\mathbf{K} \in R^{T \times d_k}$

The value: $\mathbf{V} \in R^{T \times d_v}$

This is the dot-product scoring function we saw in the previous section.

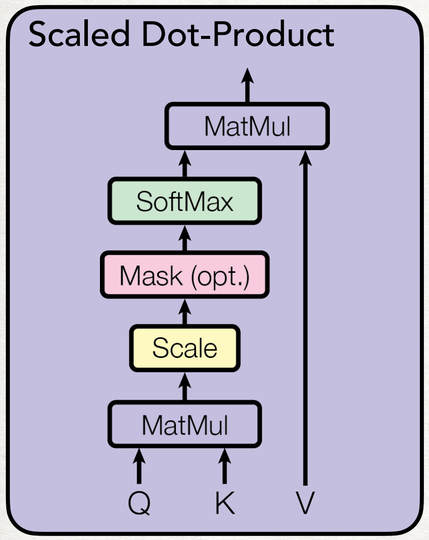
$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$

The $\sqrt{d_k}$ in the denominator is there to prevent the dot product from getting too big.



TRANSFORMERS

SCALED DOT-PRODUCT ATTENTION



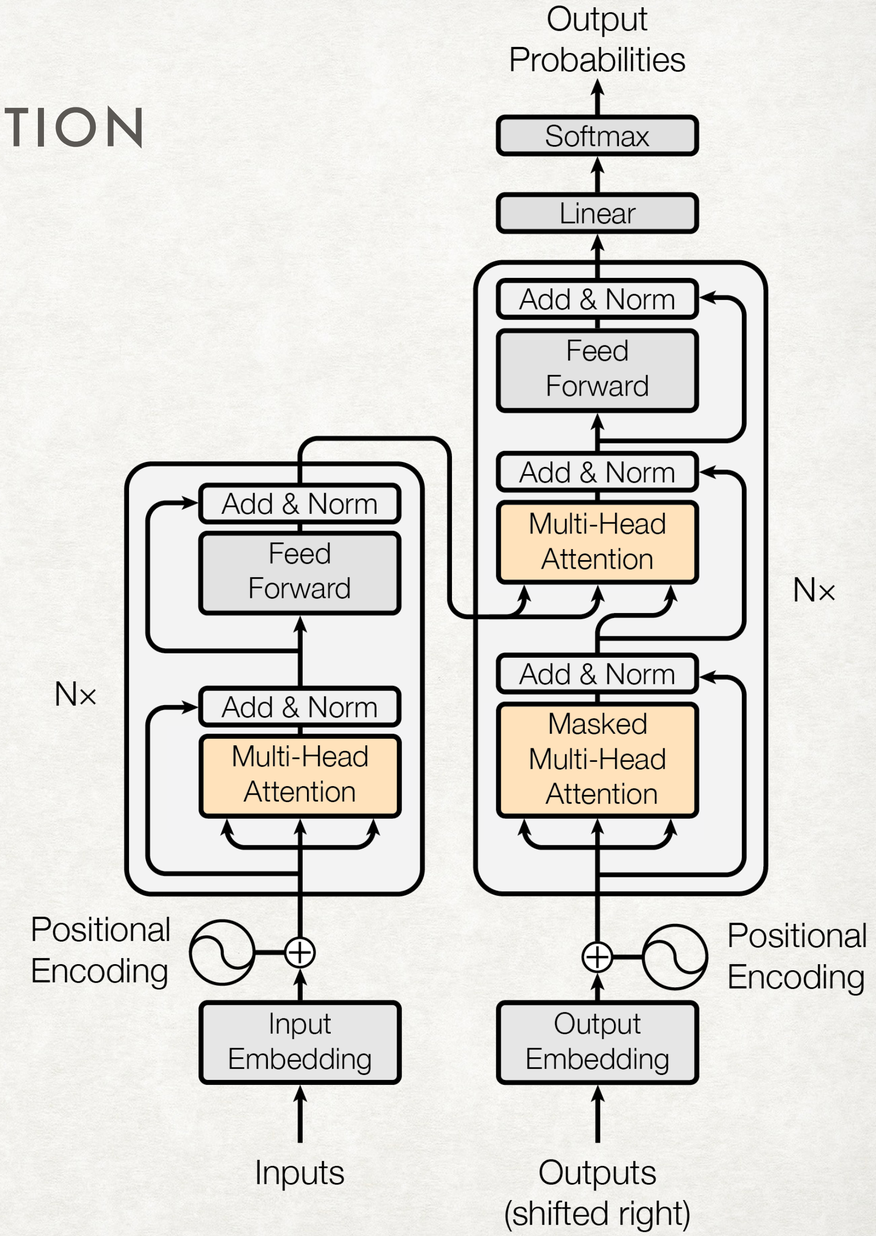
$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$

My attempt at an English translation:

- For each of the vectors in Q, the query matrix, take a linear sum of the vectors in V, the value matrix.
- The amount to weigh each vector in V is dependent on how "similar" that vector is to the query vector.
- "Similar" is measured in terms of the dot product between the vectors.

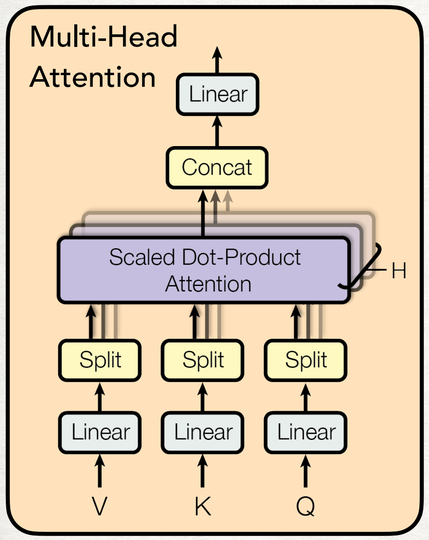
For encoder-decoder attention:
Keys and values come from encoder's final output.
Queries come from the previous decoder layer's outputs.

For self-attention:
Keys, queries, and values all come from the outputs of the previous layer.



TRANSFORMERS

MULTI-HEAD ATTENTION



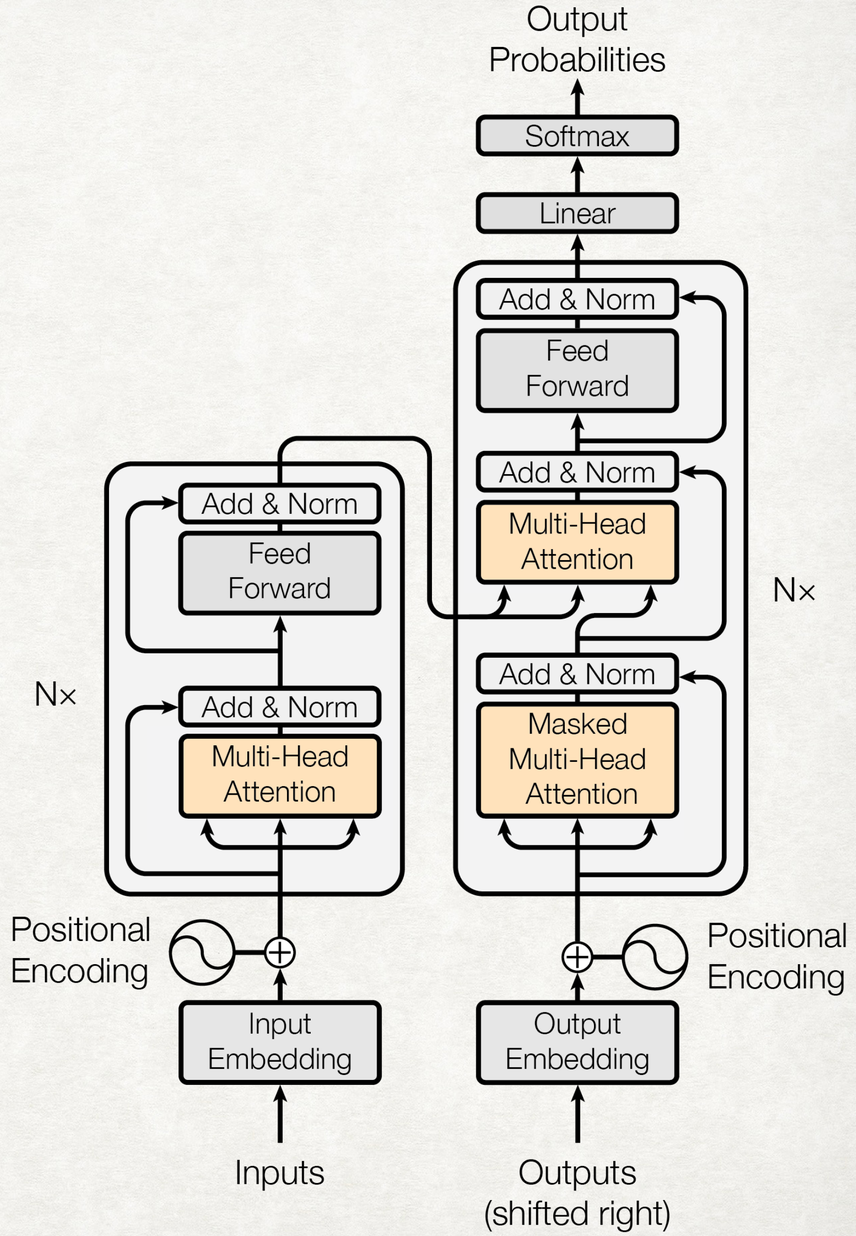
$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$

$$\text{MultiHeadAtt}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\mathbf{W}^O$$

$$\text{where } \text{head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^{\mathbf{Q}}, \mathbf{K}\mathbf{W}_i^{\mathbf{K}}, \mathbf{V}\mathbf{W}_i^{\mathbf{V}})$$

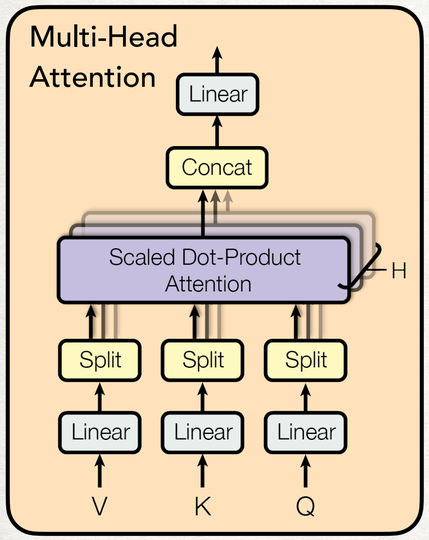
Instead of operating on $\mathbf{Q}, \mathbf{K}$, and $\mathbf{V}$ directly, the mechanism projects each input into a smaller dimension. This is done h times. The attention operation is performed on each of these "heads," and the results are concatenated.

Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions.

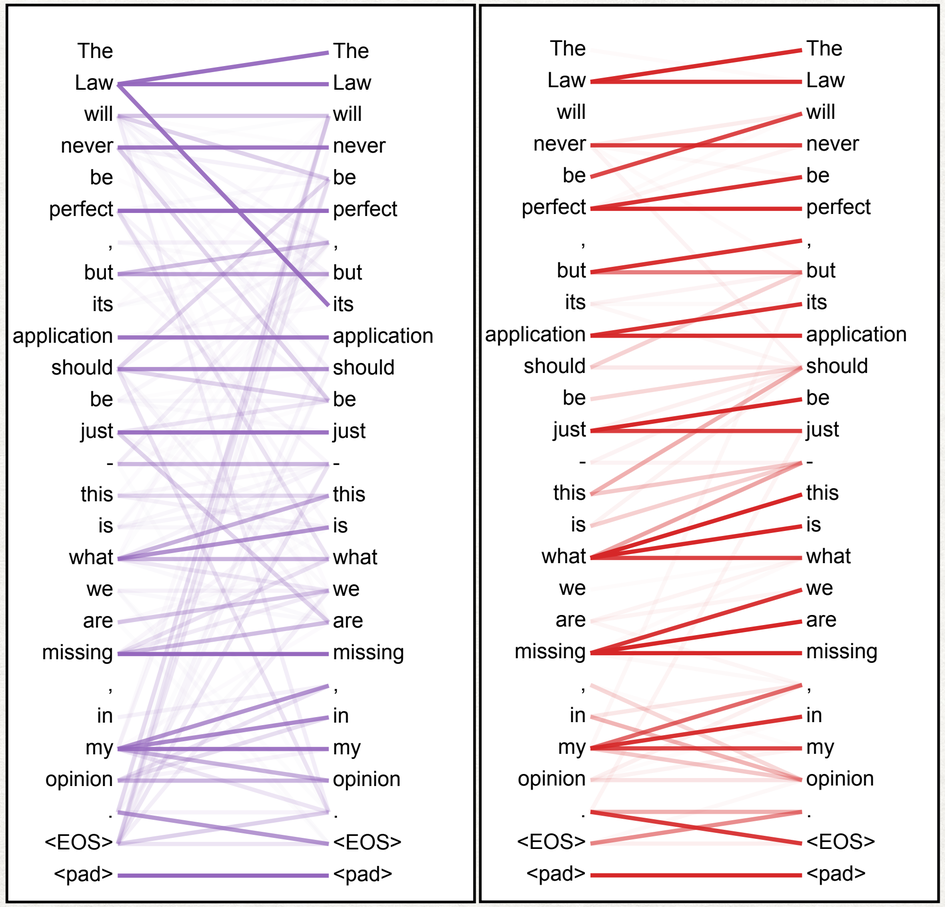


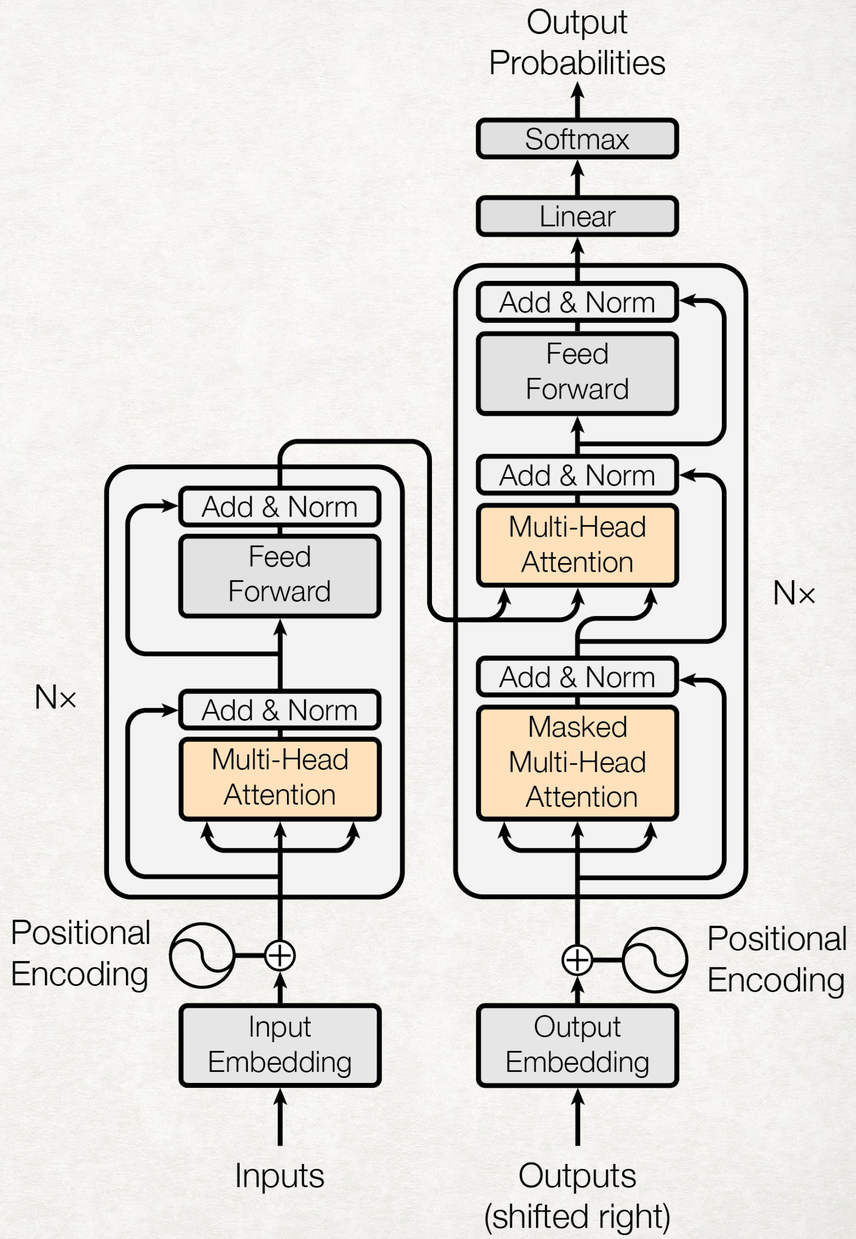
TRANSFORMERS

MULTI-HEAD ATTENTION



Two different self-attention heads:

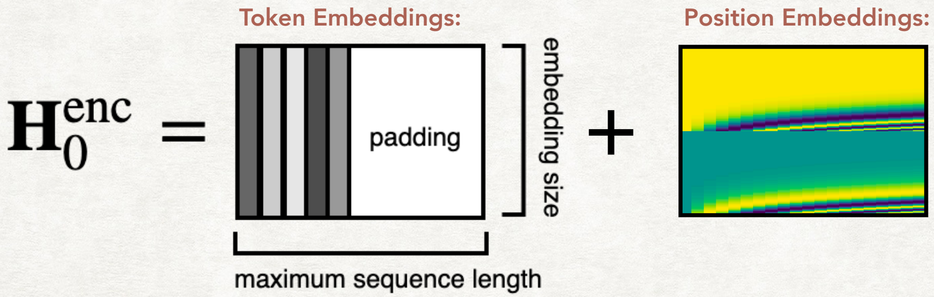




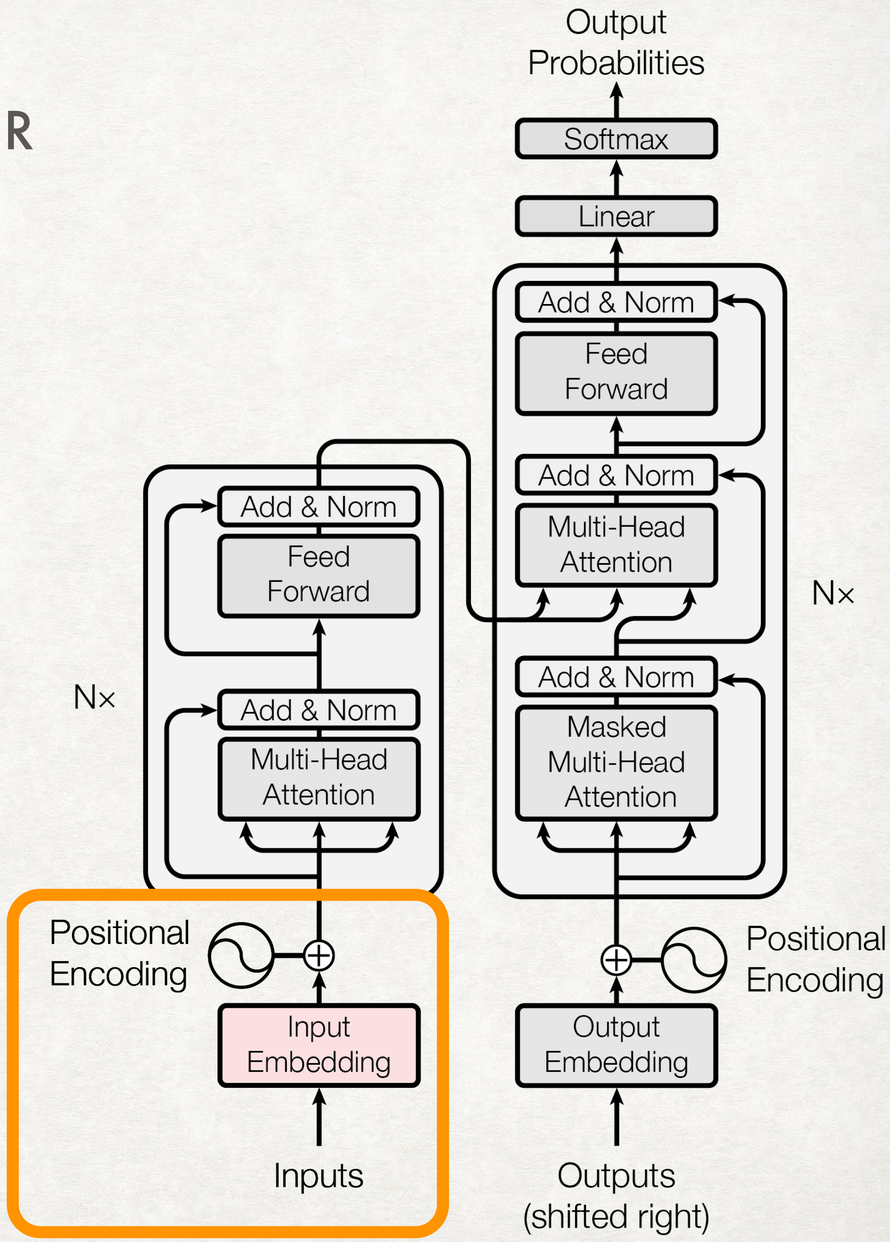
TRANSFORMERS

INPUTS TO THE ENCODER

The input into the encoder looks like:



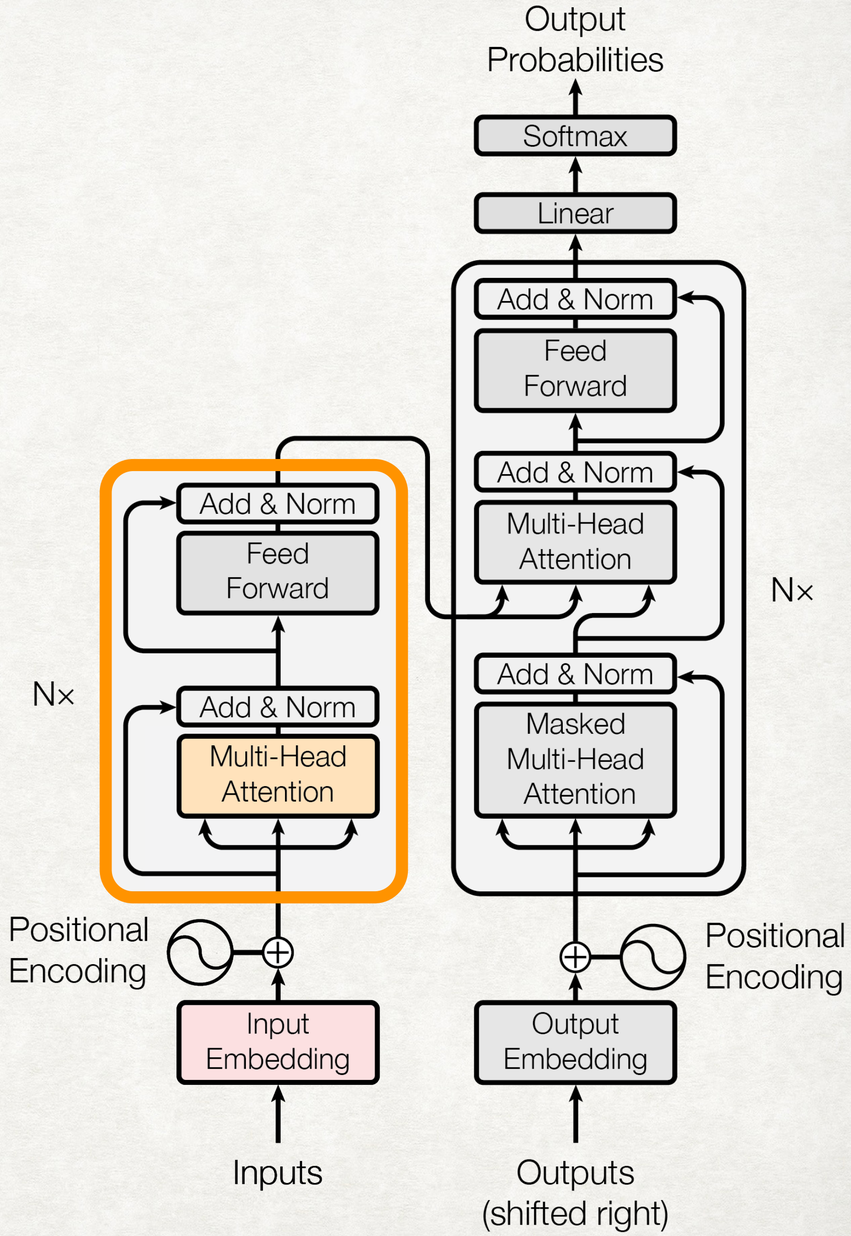
$$\mathbf{H}_0^{enc} = \text{token embeddings} + \text{position embeddings}$$



TRANSFORMERS

THE ENCODER

Multi-Head Attention = MultiHeadAtt($\mathbf{H}_i^{\text{enc}}$, $\mathbf{H}_i^{\text{enc}}$, $\mathbf{H}_i^{\text{enc}}$)

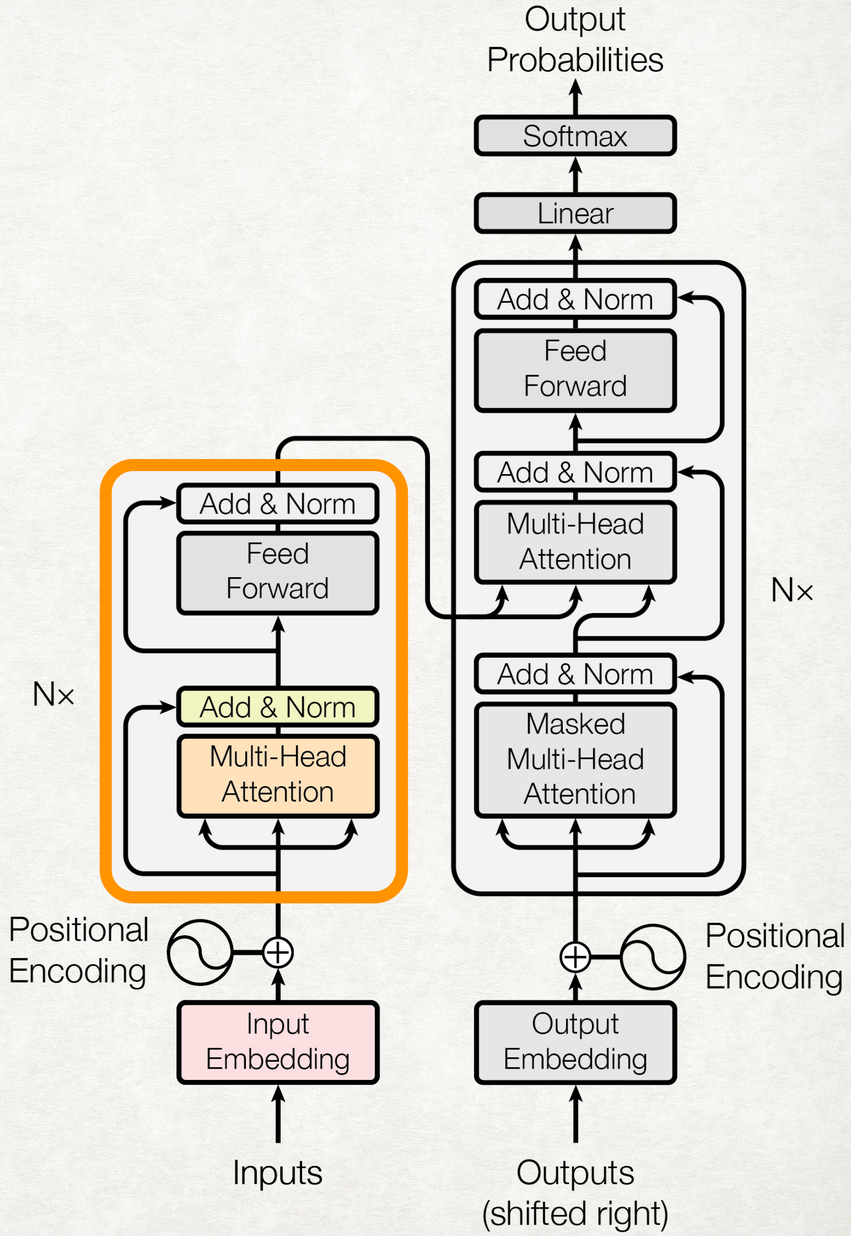


TRANSFORMERS

THE ENCODER

Multi-Head Attention $= \text{MultiHeadAtt}(\mathbf{H}_i^{\text{enc}}, \mathbf{H}_i^{\text{enc}}, \mathbf{H}_i^{\text{enc}})$

Add & Norm $= \text{LayerNorm}($ Multi-Head Attention $+ \mathbf{H}_i^{\text{enc}})$



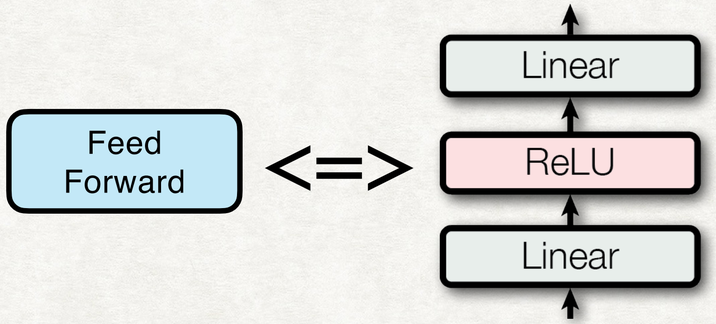
TRANSFORMERS

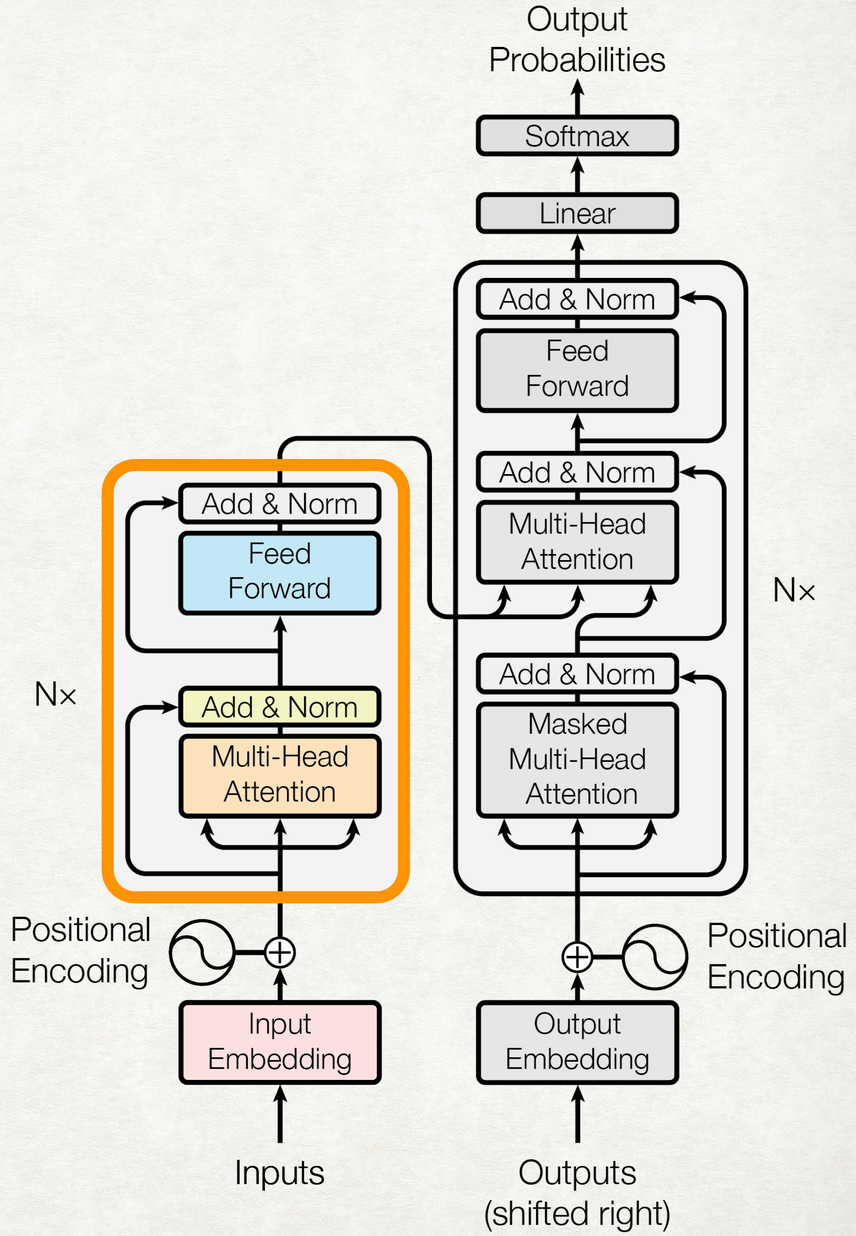
THE ENCODER

Multi-Head Attention $= \text{MultiHeadAtt}(\mathbf{H}_i^{\text{enc}}, \mathbf{H}_i^{\text{enc}}, \mathbf{H}_i^{\text{enc}})$

Add & Norm $= \text{LayerNorm}($ Multi-Head Attention $+ \mathbf{H}_i^{\text{enc}})$

Feed Forward $= \max(0,$ Add & Norm $\mathbf{W}_1 + b_1)\mathbf{W}_2 + b_2$





TRANSFORMERS

THE ENCODER

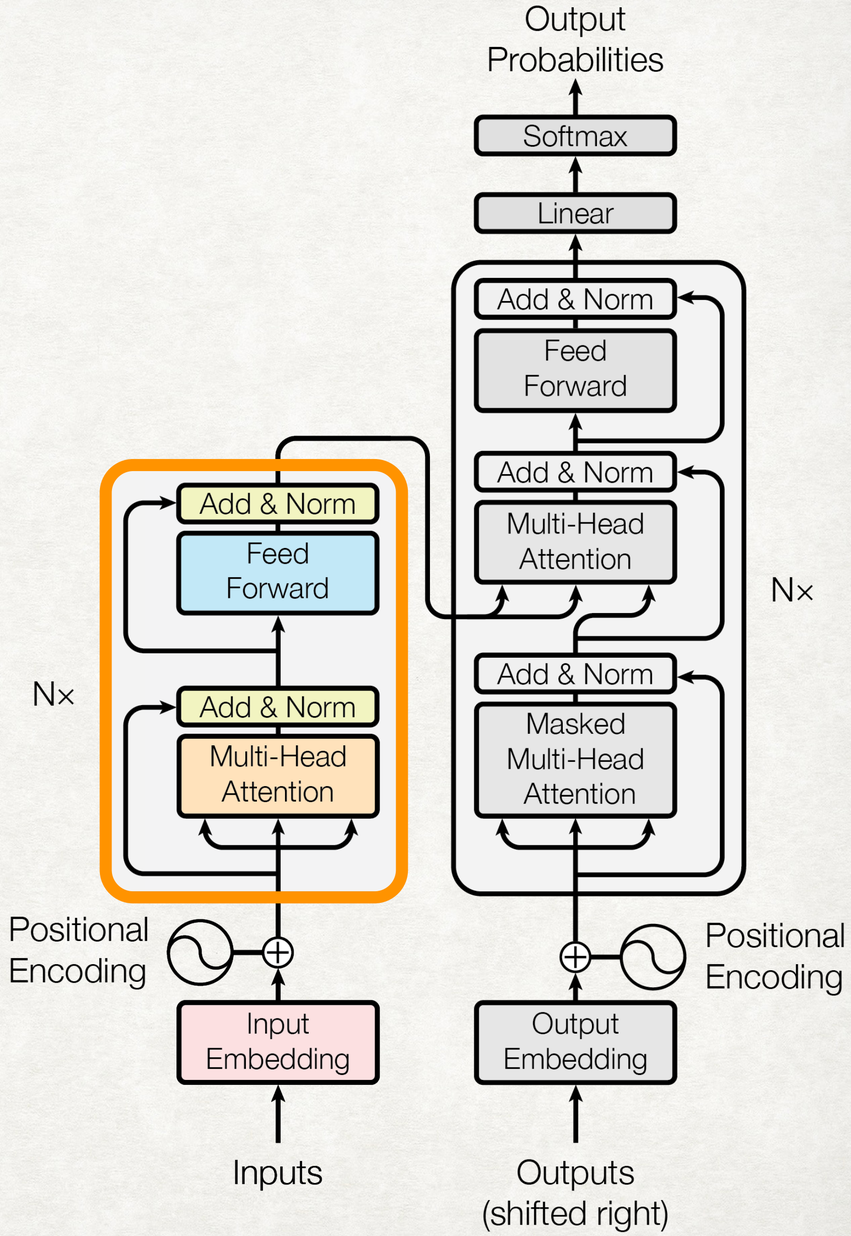
Multi-Head Attention $= \text{MultiHeadAtt}(\mathbf{H}_i^{\text{enc}}, \mathbf{H}_i^{\text{enc}}, \mathbf{H}_i^{\text{enc}})$

Add & Norm $= \text{LayerNorm}($ Multi-Head Attention $+ \mathbf{H}_i^{\text{enc}})$

Feed Forward $= \max(0,$ Add & Norm $\mathbf{W}_1 + b_1)\mathbf{W}_2 + b_2$

Add & Norm (2) $= \text{LayerNorm}($ Feed Forward $+ \mathbf{H}_i^{\text{enc}})$

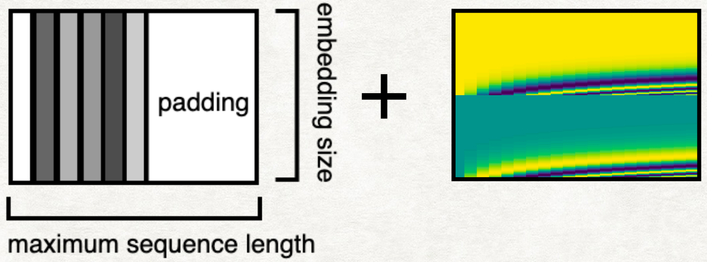
$\mathbf{H}_{i+1}^{\text{enc}} =$ Add & Norm (2)



TRANSFORMERS

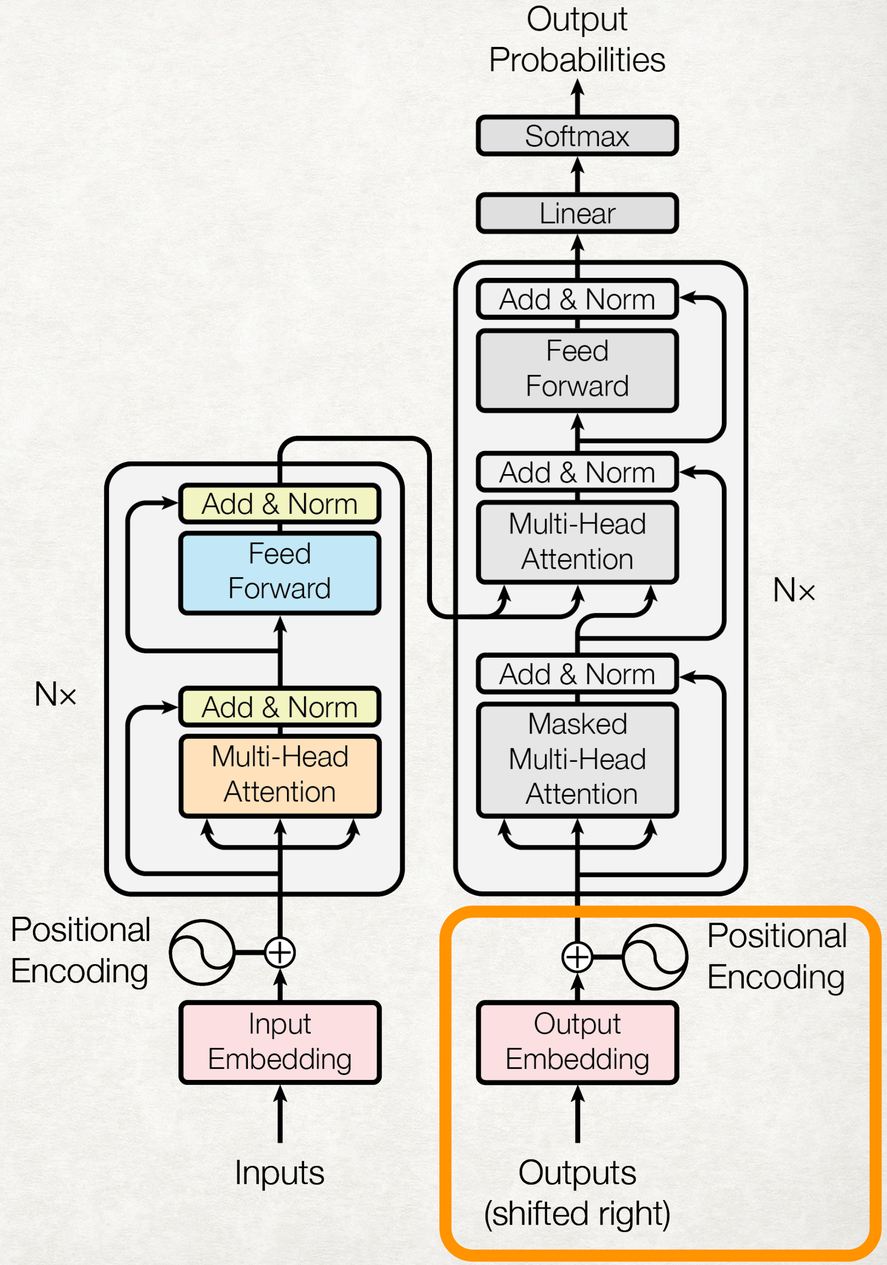
THE DECODER

$$\mathbf{H}_0^{dec} =$$



$+$

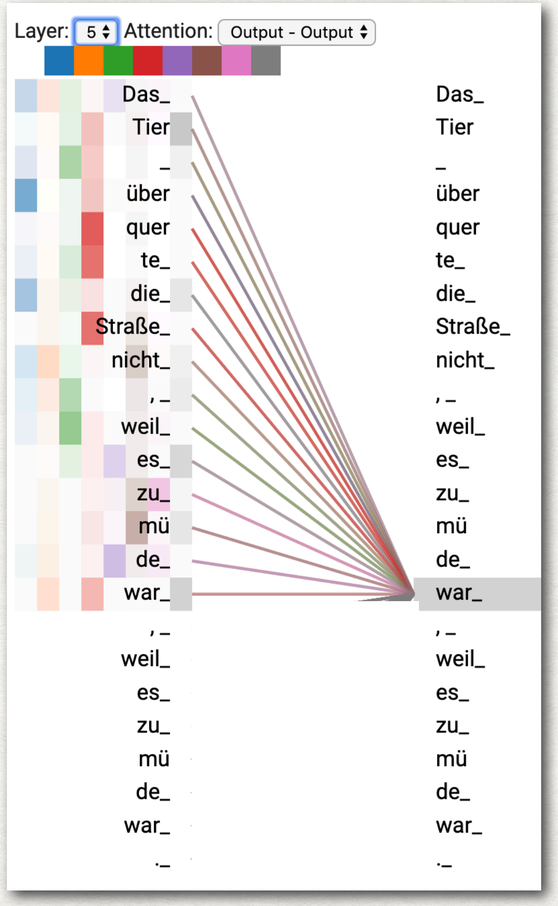
= token embeddings + position embeddings

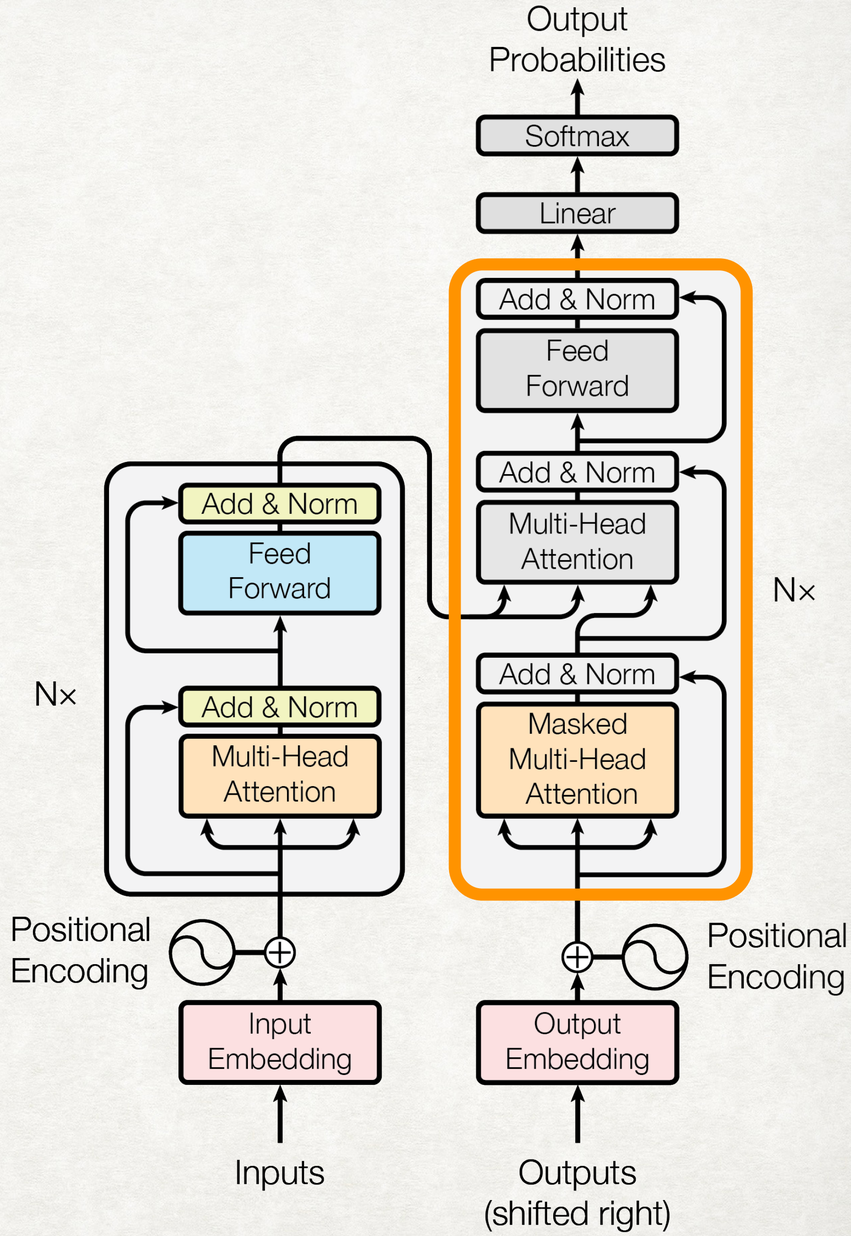


TRANSFORMERS

THE DECODER

Masked Multi-Head Attention $= \text{MaskedMultiHeadAtt}(\mathbf{H}_i^{\text{dec}}, \mathbf{H}_i^{\text{dec}}, \mathbf{H}_i^{\text{dec}})$



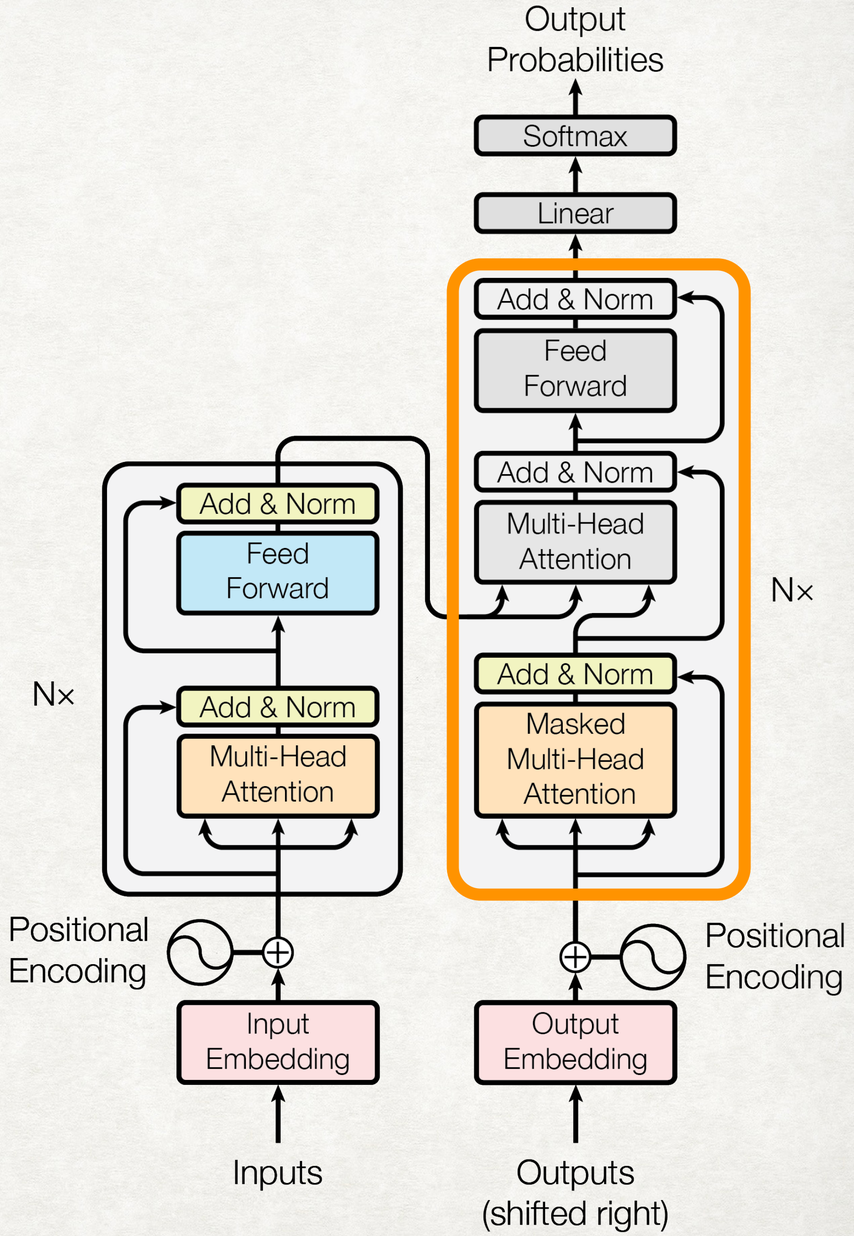


TRANSFORMERS

THE DECODER

Masked Multi-Head Attention $= \text{MaskedMultiHeadAtt}(\mathbf{H}_i^{\text{dec}}, \mathbf{H}_i^{\text{dec}}, \mathbf{H}_i^{\text{dec}})$

Add & Norm $= \text{LayerNorm}($ Multi-Head Attention $+ \mathbf{H}_i^{\text{dec}})$



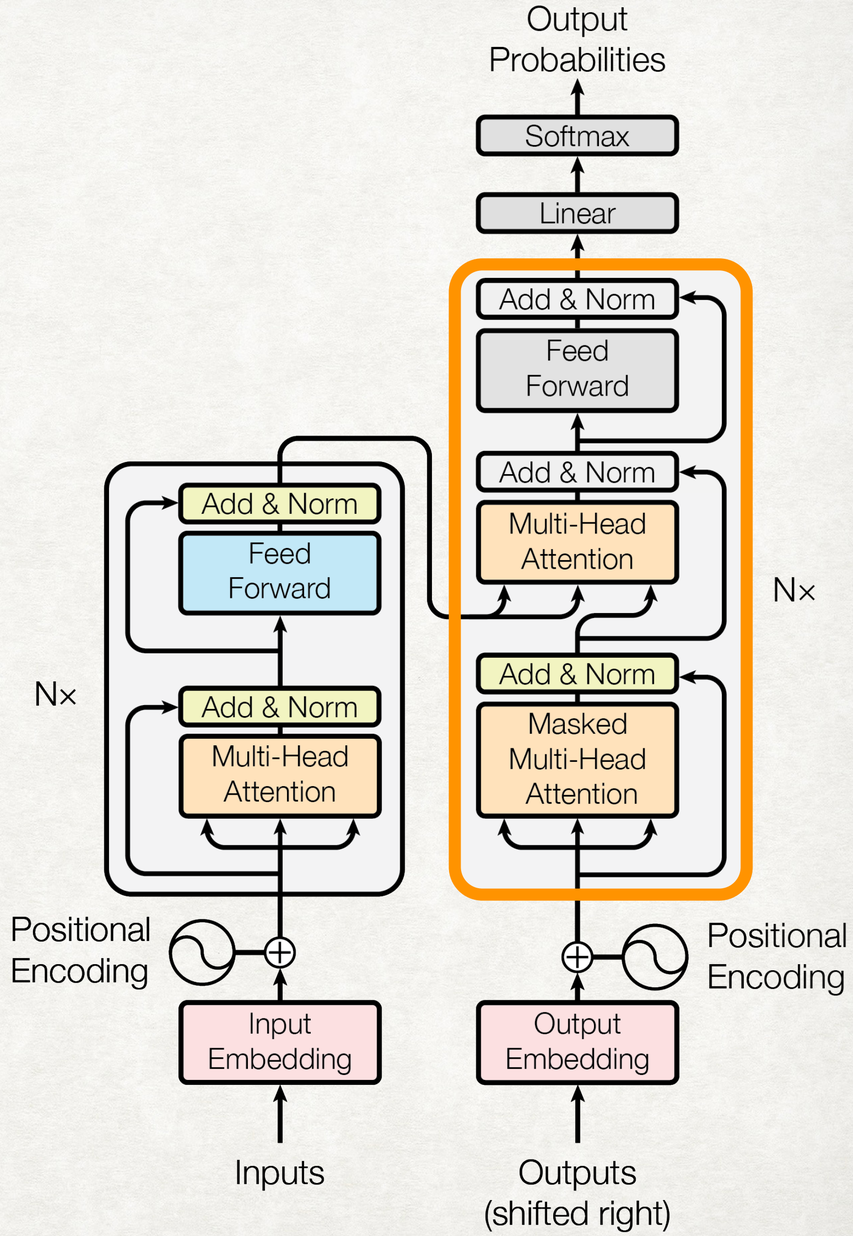
TRANSFORMERS

THE DECODER

Masked Multi-Head Attention $= \text{MaskedMultiHeadAtt}(\mathbf{H}_i^{\text{dec}}, \mathbf{H}_i^{\text{dec}}, \mathbf{H}_i^{\text{dec}})$

Add & Norm $= \text{LayerNorm}($ Multi-Head Attention $+ \mathbf{H}_i^{\text{dec}})$

Enc-Dec Multi-Head Attention $= \text{MultiHeadAtt}(\mathbf{H}_i^{\text{enc}}, \mathbf{H}_i^{\text{dec}}, \mathbf{H}_i^{\text{dec}})$



TRANSFORMERS

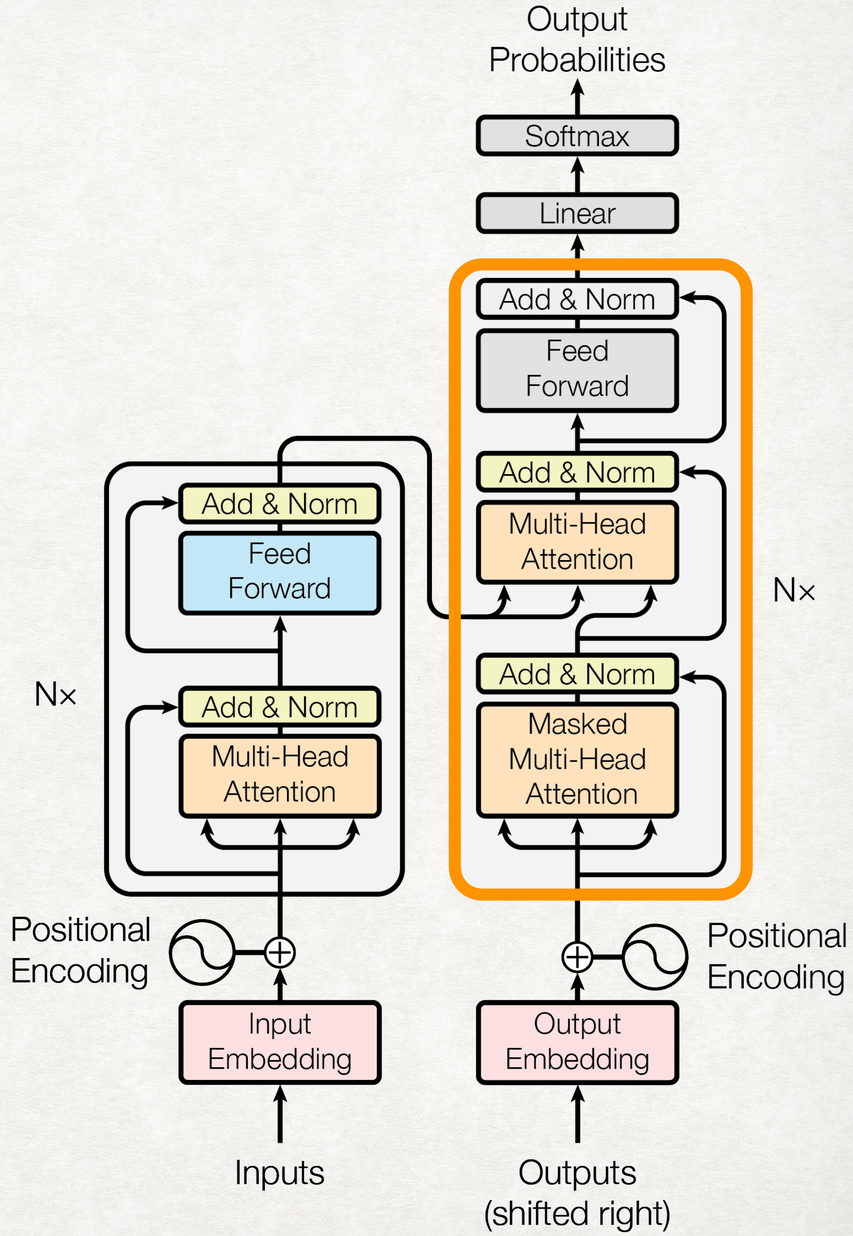
THE DECODER

Masked Multi-Head Attention = MaskedMultiHeadAtt($\mathbf{H}_i^{\text{dec}}$, $\mathbf{H}_i^{\text{dec}}$, $\mathbf{H}_i^{\text{dec}}$)

Add & Norm = LayerNorm(Multi-Head Attention + $\mathbf{H}_i^{\text{dec}}$)

Enc-Dec Multi-Head Attention = MultiHeadAtt($\mathbf{H}_i^{\text{dec}}$, $\mathbf{H}_i^{\text{enc}}$, $\mathbf{H}_i^{\text{enc}}$)

Add & Norm (2) = LayerNorm(Multi-Head Attention + Add & Norm)



TRANSFORMERS

THE DECODER

Masked Multi-Head Attention $= \text{MaskedMultiHeadAtt}(\mathbf{H}_i^{\text{dec}}, \mathbf{H}_i^{\text{dec}}, \mathbf{H}_i^{\text{dec}})$

Add & Norm $= \text{LayerNorm}($ Multi-Head Attention $+ \mathbf{H}_i^{\text{dec}})$

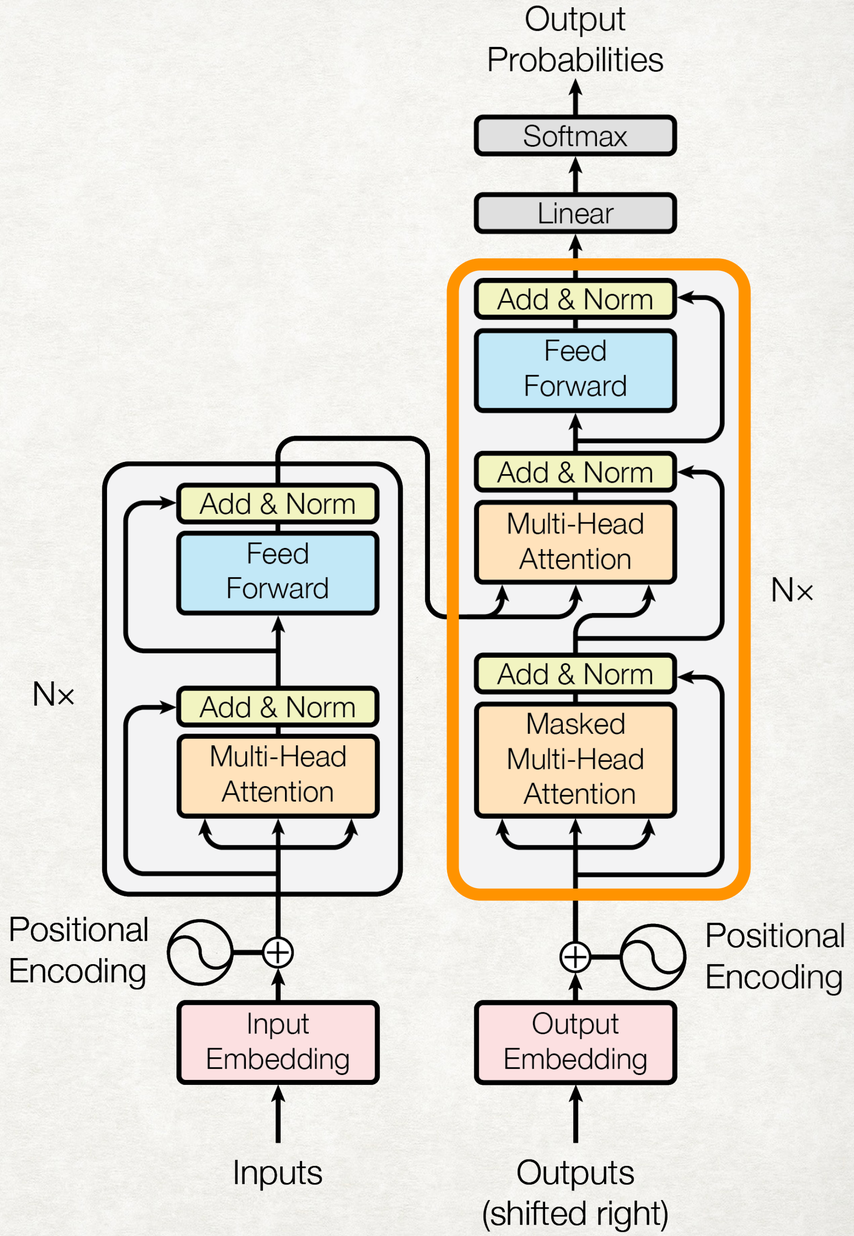
Enc-Dec Multi-Head Attention $= \text{MultiHeadAtt}(\mathbf{H}_i^{\text{dec}}, \mathbf{H}_i^{\text{enc}}, \mathbf{H}_i^{\text{enc}})$

Add & Norm (2) $= \text{LayerNorm}($ Multi-Head Attention $+$ Add & Norm $)$

Feed Forward $= \max(0,$ Add & Norm (2) $\mathbf{W}_1 + b_1)\mathbf{W}_2 + b_2$

Add & Norm (3) $= \text{LayerNorm}($ Feed Forward $+$ Add & Norm (2) $)$

$\mathbf{H}_{i+1}^{\text{dec}} =$ Add & Norm (3)



TRANSFORMERS
GENERATED TEXT CIRCA 2018

'''The Transformer''' are a Japanese [[hardcore punk]] band.

==Early years==
The band was formed in 1968, during the height of Japanese music history. Among the legendary [[Japanese people|Japanese]] composers of [Japanese lyrics], they prominently exemplified Motohiro Oda's especially tasty lyrics and psychedelic intention. Michio was a longtime member of the every Sunday night band PSM. His alluring was of such importance as being the man who ignored the already successful image and that he municipal makeup whose parents were – the band was called Jenei.<ref>http://www.separatist.org/se_frontend/post-punk-musician-the-kidney.html</ref> From a young age the band was very close, thus opting to pioneer what had actually begun as a more manageable core hardcore punk band.<ref>http://www.talkradio.net/article/independent-music-fades-from-the-closed-drawings-out</ref>

==History==
===Born from the heavy metal revolution===
In 1977 the self-proclaimed King of Tesponsors, [[Joe Lus:

: It was somewhere... it was just a guile ... taking this song to Broadway. It was the first record I ever heard on A.M., After some opposition I received at the hands of Parsons, and in the follow-up notes myself.<ref>http://www.discogs.com/artist/The+Op%C5%8Dn+&+Psalm</ref>

The band cut their first record album titled "Transformed, furthered and extended Extended",<ref>[https://www.discogs.com/album/69771 MC – Transformed EP (CDR) by The Moondrawn – EMI, 1994]</ref> and in 1978 the official band line-up of the three-piece pop-punk-rock band TEEM. They generally played around [[Japan]], growing from the Top 40 standard.

===1981-2010: The band to break away===
On 1 January 1981 bassist Michio Kono, and the members of the original line-up emerged. Niji Fukune and his [[Head poet|Head]] band (now guitarist) Kazuya Kouda left the band in the hands of the band at the May 28, 1981, benefit season of [[Led Zeppelin]]'s Marmarin building. In June 1987, Kono joined the band as a full-time drummer, playing a few nights in a 4 or 5 hour stint with [[D-beat]]. Kono played through the mid-1950s, at Shinlie, continued to play concerts with drummers in Ibis, Cor, and a few at the Leo Somu Studio in Japan. In 1987, Kono recruited new bassist Michio Kono and drummer Ayaka Kurobe as drummer for band. Kono played trumpet with supplement music with Saint Etienne as a drummer. Over the next few years Kono played as drummer and would get many alumni news invitations to the bands' "Toys Beach" section. In 1999 he joined the [[CT-182]].

His successor was Barrie Bell on a cover of [[Jethro Tull (band)|Jethro Tull]]'s original 1967 hit "Back Home" (last appearance was in Jethro), with whom he shares a name.

===2010 – present: The band to split===
In 2006 the band split up and the remaining members reformed under the name Starmirror, with Kono in tears, Kurobe, and Kurobe all playing harmonica with Kooky Bell and a new guitarist again. While Jaari also had the realist DJ experience, The SkykelDaten asked New Bantherhine, who liked Kono and Kurobe, to join him on guitar. Kono is now playing in the studio a new formation, and at their 11th anniversary concert, made a wide variety of music and DJ equipment including two new vocalists: Dejeh Faida and Janis. NARCO (Inc.) privileges areas sections until 2012. and in 2015 both were members as members as were Dicent and Cautty.

In 2014, the album "[[Marco Victoriano in Focus]]" was released, and entered the Japanese albums chart at number 69 in the [[Oricon Singles Chart]].<ref name="oricon">{{cite web|title=THE GER: THE TALENT RAILWAY LIVERSITIES|url=http://www.oricon.co.jp/prof/inductee/30565/ranking/cd/1/|work=OriconStyle|accessdate=11 Jul 2014}}</ref> The album was also in Japan, where [[Hoite (musician)|Hoite]] recorded and released an album in October 2011, and a short album, titled "[[Grateful]]" in 2012.<ref>{{cite web |url=http://www.oricon.co.jp/prof/artist/229337/ranking/cd/1/|title=HORIZON HISTORY|publisher=oricon.co.jp|accessdate=9 Nov 2014|language=ja}}</ref> Kono played the [[N9ne]] bass with Tony "Shadows Without a Face", and released his music from the new [[Smile (record label)|Smile]] label with original Fat Joe Lang "Remix and Bachian" cassette.

==Style==
The band's style has been compared to [[Radar|radar-based]], influenced by bands such as [[Metallica]], [[Damage (Japanese band)|Damage]], [[Dreadzone]], and [[Girlschool]].

The group classifies itself as "the first band to play a Used Of American Inside the Outer East",<ref>http://www.funonline.jp/i/main.php?conID=005&artID=12548</ref> including the band's usual Eugene Terre ensemble. He later stated that the raw interviews in Metal Hammer and Metallica gave reason to what they considered that an explosion of the live band started by the band.<ref>http://stillenterprise.com/en/interviews/last-night-reunion-confronts-kooky-spearing-the-charismatic-dynamic-partnership/</ref>

[[Hunt's Brigade]], a surf-rock band from Tokyo, has cited the band's music as being "straight ahead of their time / I did but only read five songs (played now), a beat, rock blast, a bit of a bass blast, good lyrics, and a few".<ref>http://stillenterprise.com/en/announcements/Package-a2/Swag-w-Vause-042332/</ref> Halutodrinking, a popular-sounding drum style, has described the band as being "supporting [mod]ed distrust", because it incorporated the track as an ensemble and did not fit into any of the more darling songs from their previous incarnation, which was forging a strong style to a lot of different

Credit: Generating Wikipedia by Summarizing Long Sequences <https://arxiv.org/abs/1801.10198>

TAKEAWAYS
EXTENSIONS TO TRANSFORMER ARCHITECTURE

- Relative attention enables Transformer to generate longer sequences than it was trained on.
  - “Self-Attention with Relative Position Representations” <https://arxiv.org/abs/1803.02155>
- Massive multi-GPU parallelization allows training giant language models (Microsoft just released one with 17 billion parameters).
  - https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/
- Distillation allows smaller models to be formed from bigger ones.
  - “Distilling Transformers into Simple Neural Networks with Unlabeled Transfer Data” <https://arxiv.org/abs/1910.01769>
- Lots of attempts to make sparse attention mechanisms work.
  - “Efficient Content-Based Sparse Attention with Routing Transformers” <https://openreview.net/forum?id=B1gjs6EtDr>
  - “Reformer: The Efficient Transformer” <https://arxiv.org/abs/2001.04451>

OUTLINE

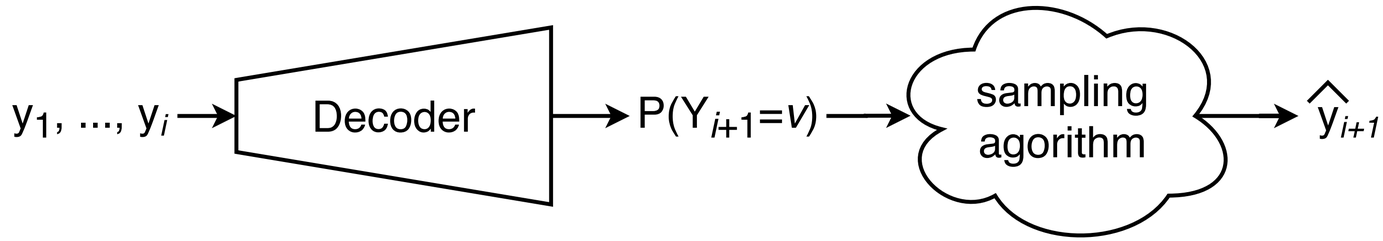
- Neural language model framework
- LM Architectures
  - Recurrent neural networks
  - Transformers
- **Decoding Strategies**
- Transformers for natural language understanding
  - BERT
  - T5

OUTLINE

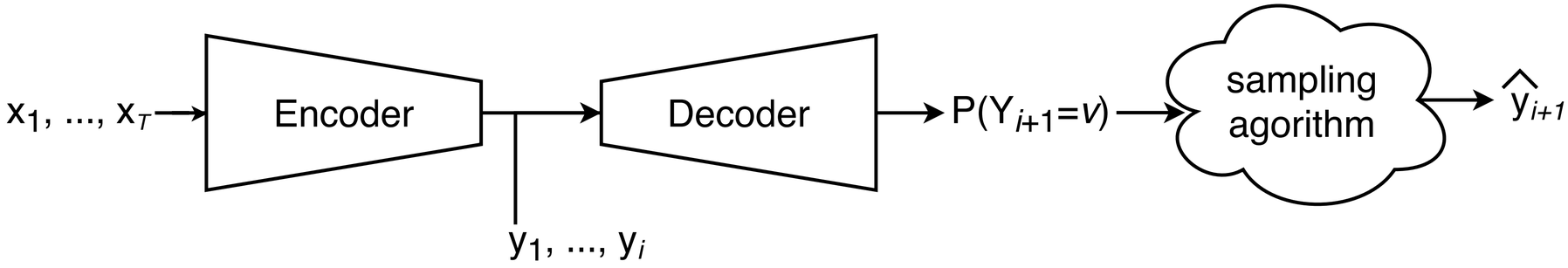
- **Decoding Strategy Recap**
- Automatic Detection of Generated Text
- Why is it difficult to answer the question “which decoding strategy is best”?

Recall that language models output a probability distribution $P(Y_t = i \mid \mathbf{y}_{1:t-1})$

Unconditioned Language Model



Conditioned Language Model



However, we're most interested in finding the most likely overall sequence $P(y_1, \ldots, y_T)$.

Recall that using the chain rule, we can refer to the probability of a sequence of words as the product of the conditional probability of each word given the words that precede it.

`P(["I", "eat", "the", "apple"]) =`

`P("apple" | ["I", "eat", "the"]) * P("the" | ["I", "eat"]) * P("eat" | ["I"]) * P(["I"])`

Actually maximizing $P(y_1, \ldots, y_T)$ is intractable, so we try to approximate doing so when choosing a next token based on the $P(Y_t = i \mid \mathbf{y}_{1:t-1})$ outputted by the LM.

How can we sample from $P(Y_t = i \,|\, \mathbf{y}_{1:t-1})$?

Option 1: Take $\arg\max_i P(Y_t = i \,|\, \mathbf{y}_{1:t-1})$

Example:

Suppose our vocal consists of 4 words:

$\mathscr{V} = \{\text{apple}, \text{banana}, \text{orange}, \text{plum}\}$

We have primed our language "apple apple" and want to use it to make a predict for the 3rd word in the sequence.

Our language model predicts:

$P(Y_3 = \text{apple} \,|\, Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.05$

$P(Y_3 = \text{banana} \,|\, Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.65$

$P(Y_3 = \text{orange} \,|\, Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.2$

$P(Y_3 = \text{plum} \,|\, Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.1$

If we sample with argmax, what word would get selected?

How can we sample from $P(Y_t = i \mid \mathbf{y}_{1:t-1})$?

Option 1: Take $\arg\max_i P(Y_t = i \mid \mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

Example:

Suppose our vocal consists of 4 words:

$\mathscr{V} = \{\text{apple}, \text{banana}, \text{orange}, \text{plum}\}$

We have primed our language "apple apple" and want to use it to make a predict for the 3rd word in the sequence.

Our language model predicts:

$P(Y_3 = \text{apple} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.05$

$P(Y_3 = \text{banana} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.65$

$P(Y_3 = \text{orange} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.2$

$P(Y_3 = \text{plum} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.1$

If we use random sampling, what is the probability that "plum" will get chosen as the third word?

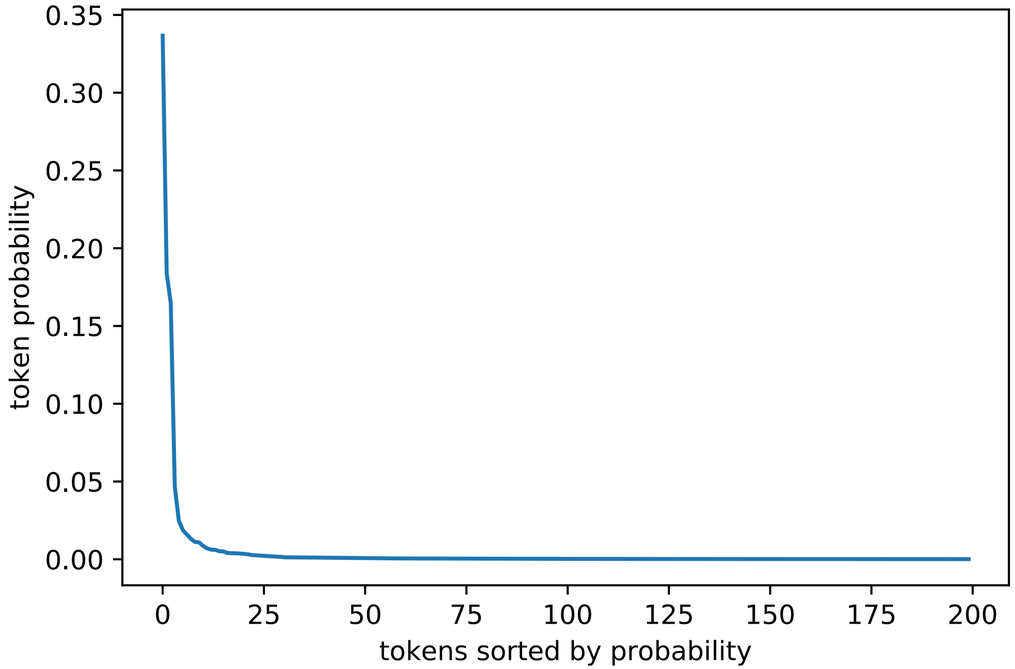
How can we sample from $P(Y_t = i \mid \mathbf{y}_{1:t-1})$?

Option 1: Take $\arg\max_i P(Y_t = i \mid \mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

Problem with Random Sampling

Most tokens in the vocabulary get assigned very low probabilities but cumulatively, choosing any one of these low-probability tokens becomes pretty likely. In the example on the right, there is over a 17% chance of choosing a token with $P(Y_t = i) \leq 0.01$.



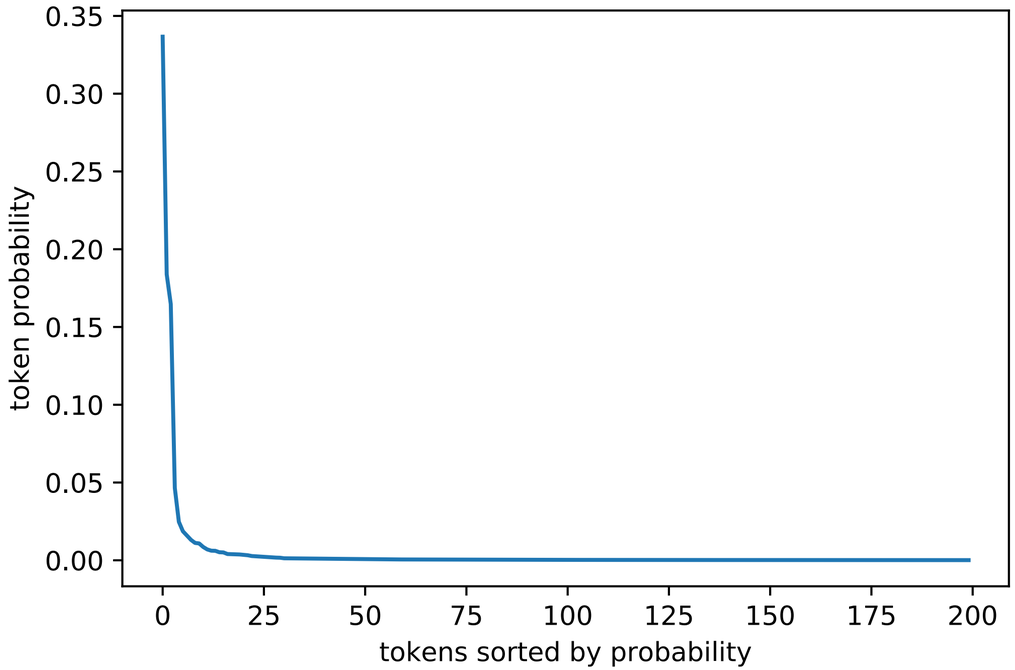
How do we sample from $P(Y_t = i \mid \mathbf{y}_{1:t-1})$?

Option 1: Take $\arg\max_i P(Y_t = i \mid \mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

Problem with Random Sampling

Most tokens in the vocabulary get assigned very low probabilities but cumulatively, choosing any one of these low-probability tokens becomes pretty likely. In the example on the right, there is over a 17% chance of choosing a token with $P(Y_t = i) \leq 0.01$.



Solution: modify the distribution returned by the model to make the tokens In the tail less likely.

How do we sample from $P(Y_t = i \mid \mathbf{y}_{1:t-1})$?

Option 1: Take $\arg\max_i P(Y_t = i \mid \mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

Option 3: Randomly sample with temperature.

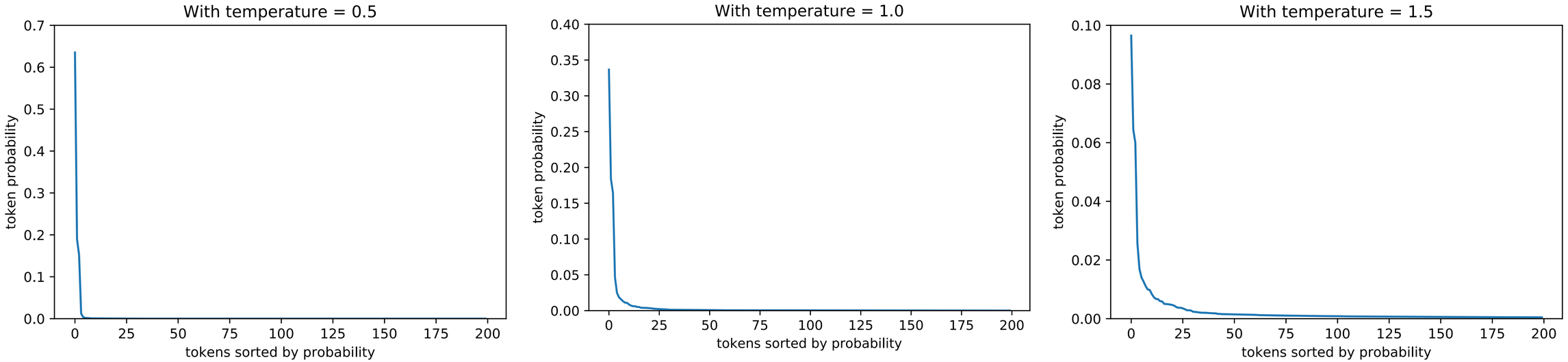
$$P(Y_t = i) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

How do we sample from $P(Y_t = i \mid \mathbf{y}_{1:t-1})$?

Option 1: Take $\arg\max_i P(Y_t = i \mid \mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

Option 3: Randomly sample with temperature.



How do we sample from $P(Y_t = i \mid \mathbf{y}_{1:t-1})$?

Option 1: Take $\arg\max_i P(Y_t = i \mid \mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

Option 3: Randomly sample with temperature.

$$P(Y_t = i) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

Example:

Suppose our vocal consists of 4 words:

$\mathscr{V} = \{\text{apple}, \text{banana}, \text{orange}, \text{plum}\}$

We have primed our language "apple apple" and want to use it to make a predict for the 3rd word in the sequence.

Our language model predicts:

$P(Y_3 = \text{apple} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.05$

$P(Y_3 = \text{banana} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.65$

$P(Y_3 = \text{orange} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.2$

$P(Y_3 = \text{plum} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.1$

What would the probability of selecting "banana" be if we use temperature sampling and set $T = \infty$?

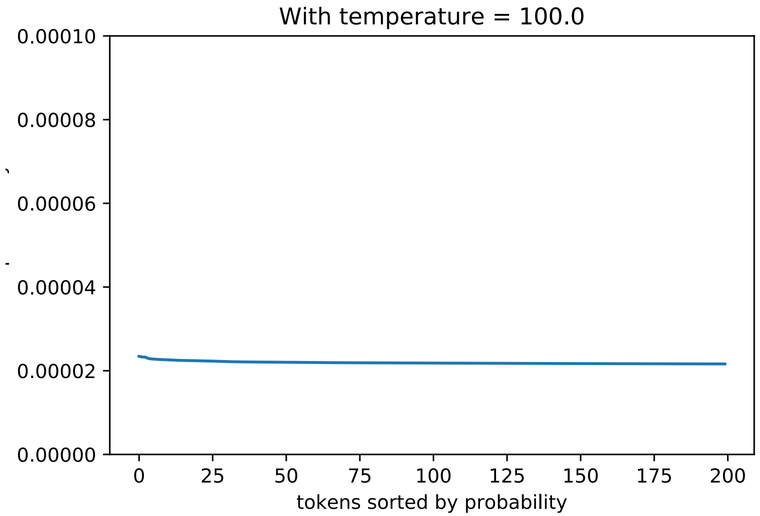
How do we sample from $P(Y_t = i \mid \mathbf{y}_{1:t-1})$?

Option 1: Take $\arg\max_i P(Y_t = i \mid \mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

Option 3: Randomly sample with temperature.

$$P(Y_t = i) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$



Example:

Suppose our vocal consists of 4 words:

$\mathscr{V} = \{\text{apple}, \text{banana}, \text{orange}, \text{plum}\}$

We have primed our language "apple apple" and want to use it to make a predict for the 3rd word in the sequence.

Our language model predicts:

$P(Y_3 = \text{apple} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.05$

$P(Y_3 = \text{banana} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.65$

$P(Y_3 = \text{orange} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.2$

$P(Y_3 = \text{plum} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.1$

What would the probability of selecting "banana" be if we use temperature sampling and set $T = \infty$?

Answer: 0.25

How do we sample from $P(Y_t = i \mid \mathbf{y}_{1:t-1})$?

Option 1: Take $\arg\max_i P(Y_t = i \mid \mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

Option 3: Randomly sample with temperature.

$$P(Y_t = i) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

Example:

Suppose our vocal consists of 4 words:

$\mathscr{V} = \{\text{apple}, \text{banana}, \text{orange}, \text{plum}\}$

We have primed our language "apple apple" and want to use it to make a predict for the 3rd word in the sequence.

Our language model predicts:

$P(Y_3 = \text{apple} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.05$

$P(Y_3 = \text{banana} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.65$

$P(Y_3 = \text{orange} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.2$

$P(Y_3 = \text{plum} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.1$

What would the probability of selecting "banana" be if we use temperature sampling and set $T = 0.00001$?

How do we sample from $P(Y_t = i \mid \mathbf{y}_{1:t-1})$?

Option 1: Take $\arg\max_i P(Y_t = i \mid \mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

Option 3: Randomly sample with temperature.

$$P(Y_t = i) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

As T approaches 0, random sampling with temperature looks more and more like argmax.

Example:

Suppose our vocal consists of 4 words:

$\mathscr{V} = \{\text{apple}, \text{banana}, \text{orange}, \text{plum}\}$

We have primed our language "apple apple" and want to use it to make a predict for the 3rd word in the sequence.

Our language model predicts:

$P(Y_3 = \text{apple} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.05$

$P(Y_3 = \text{banana} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.65$

$P(Y_3 = \text{orange} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.2$

$P(Y_3 = \text{plum} \mid Y_1 = \text{apple}, Y_2 = \text{apple}) = 0.1$

What would the probability of selecting "banana" be if we use temperature sampling and set $T = 0.00001$?

Answer: 1.0

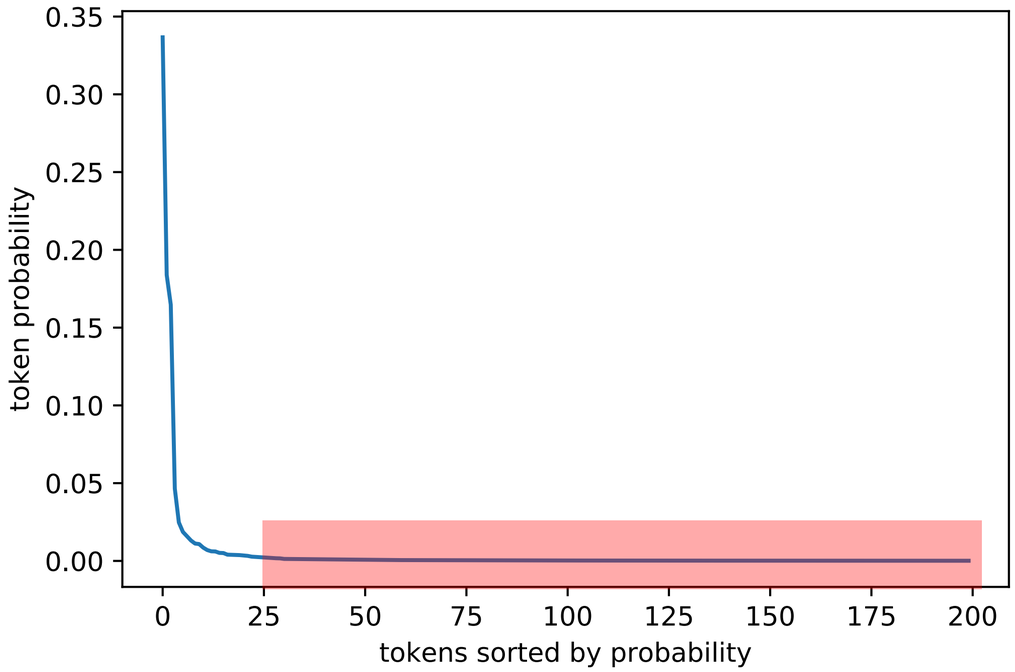
How do we sample from $P(Y_t = i \mid \mathbf{y}_{1:t-1})$?

Option 1: Take $\arg\max_i P(Y_t = i \mid \mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

Option 3: Randomly sample with temperature.

Option 4: Introduce sparsity by reassigning all probability mass to the k most likely tokens. This is referred to as top-k sampling.



Usually k between 10 and 50 is selected.

How do we sample from $P(Y_t = i \mid \mathbf{y}_{1:t-1})$?

Option 1: Take $\arg\max_i P(Y_t = i \mid \mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

Option 3: Randomly sample with temperature.

Option 4: Introduce sparsity by reassigning all probability mass to the k most likely tokens. This is referred to as top-k sampling.

Option 5: Reassign all probability mass to the k_t most likely tokens, where k_t is automatically selected at every step. It is chosen such that the total probability of the k_t most likely tokens is no greater than a desired probability p . This is referred to as nucleus sampling.

How do we sample from $P(Y_t = i \mid \mathbf{y}_{1:t-1})$?

Option 1: Take $\arg\max_i P(Y_t = i \mid \mathbf{y}_{1:t-1})$

Option 2: Randomly sample from the distribution returned by the model.

Option 3: Randomly sample with temperature.

Option 4: Introduce sparsity by reassigning all probability mass to the k most likely tokens. This is referred to as top-k sampling.

Option 5: Reassign all probability mass to the k_t most likely tokens, where k_t is automatically selected at every step. It is chosen such that the total probability of the k_t most likely tokens is no greater than a desired probability p . This is referred to as nucleus sampling.

Option 6: Use some version of beam search.

Beam search operates under the assumption that the best possible sequence to generate is the one with lowest overall sequence likelihood.

Greedy search methods do not always lead to the most likely output.

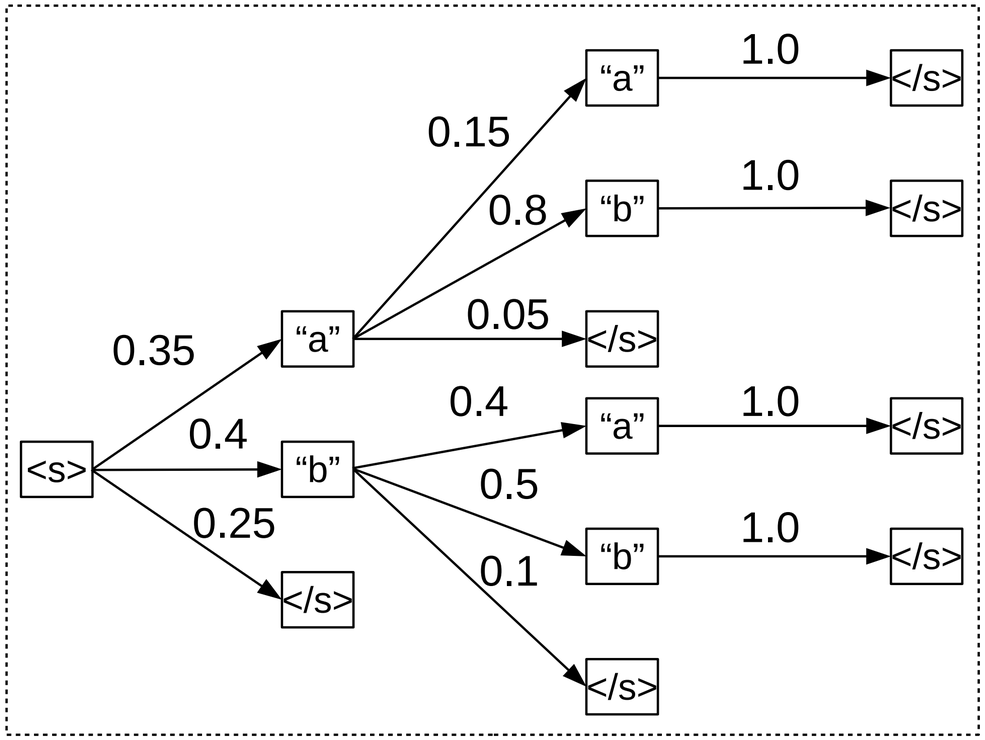


Figure 22: A search graph where greedy search fails.

Vocabulary = {a, b, </s>}
Numbers above each edge are the transition probabilities $P(x_t \mid x_{1:t-t})$

Question:
If we were to decode with argmax what would be the generated sequence?
[a, b, </s>]
[a, a, </s>]
[b, b, </s>]
[b, a, </s>]

Greedy search methods do not always lead to the most likely output.

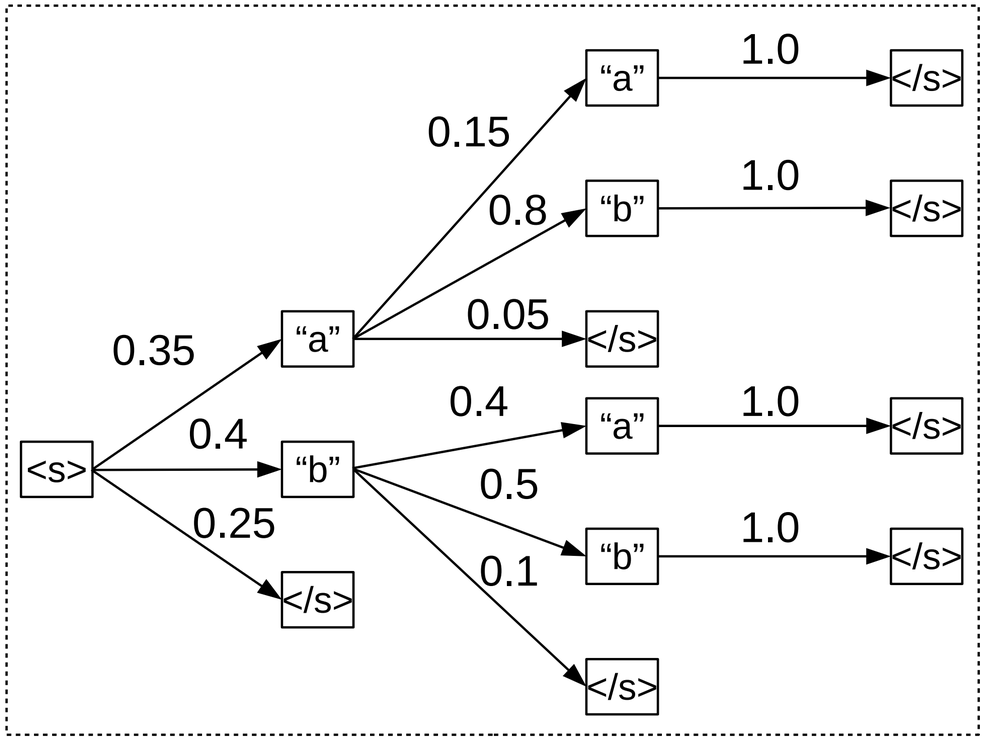


Figure 22: A search graph where greedy search fails.

Vocabulary = {a, b, </s>}
Numbers above each edge are the transition probabilities $P(x_t \mid x_{1:t-t})$

Question:
If we were to decode with argmax what would be the generated sequence?
[a, b, </s>]
[a, a, </s>]
[b, b, </s>]
[b, a, </s>]

Greedy search methods do not always lead to the most likely output.

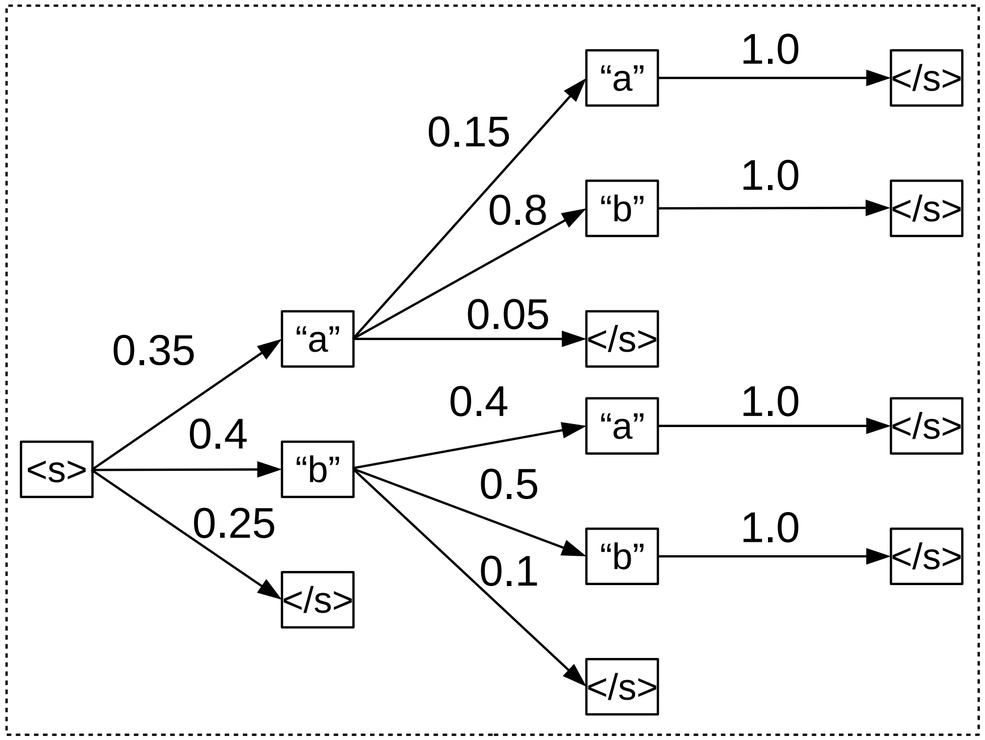


Figure 22: A search graph where greedy search fails.

Vocabulary = {a, b, </s>}

Numbers above each edge are the transition probabilities $P(x_t \mid x_{1:t-t})$

Question:
If we were to decode the sequence that optimally maximizes $P(x_1, \ldots, x_T)$, what would be the generated sequence?

[a, b, </s>]
[a, a, </s>]
[b, b, </s>]
[b, a, </s>]

Greedy search methods do not always lead to the most likely output.

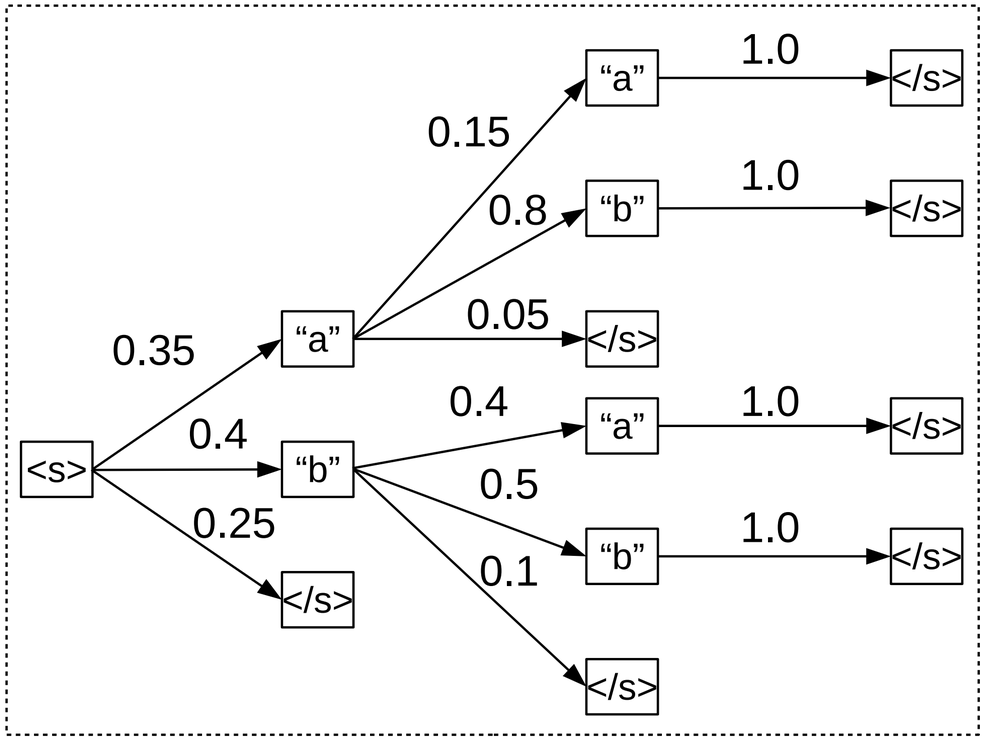


Figure 22: A search graph where greedy search fails.

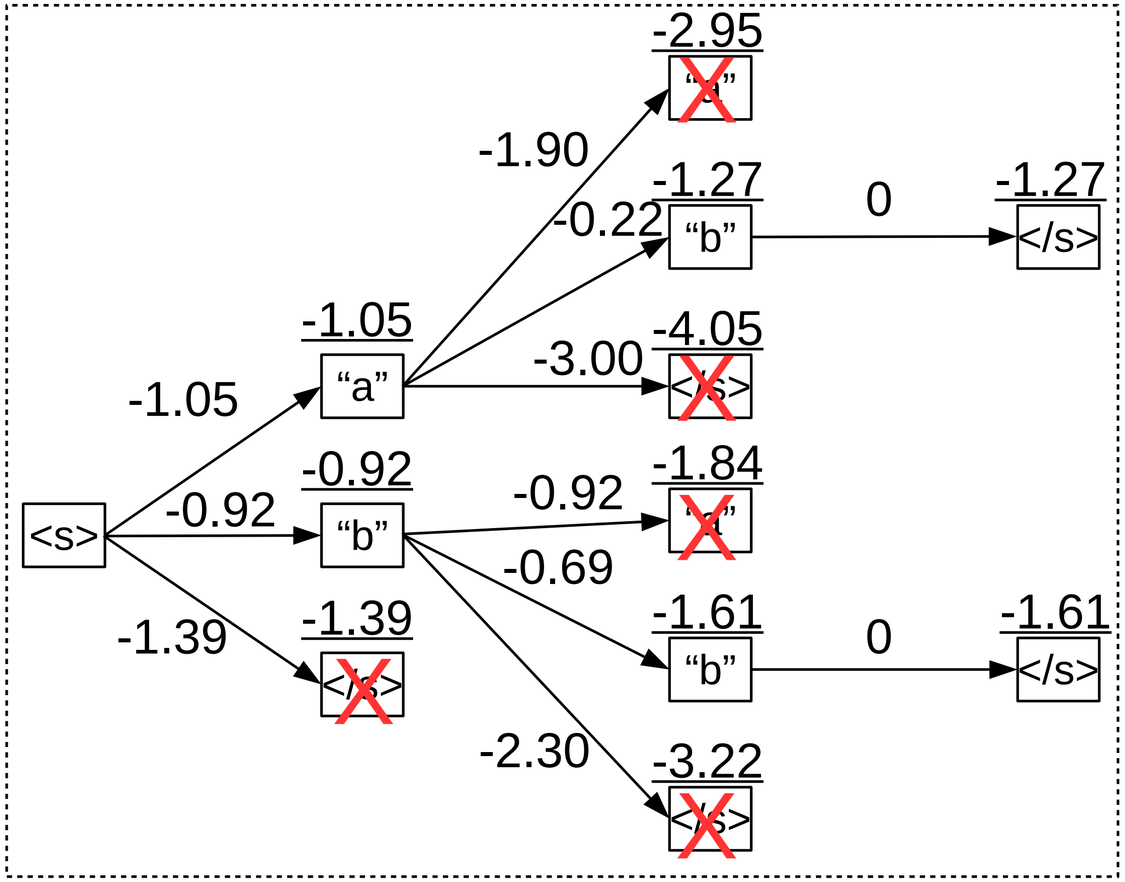
Vocabulary = {a, b, </s>}

Numbers above each edge are the transition probabilities $P(x_t \mid x_{1:t-t})$

Question:
If we were to decode the sequence that optimally maximizes $P(x_1, \ldots, x_T)$, what would be the generated sequence?

[a, b, </s>]
[a, a, </s>]
[b, b, </s>]
[b, a, </s>]

Beam search is an algorithm that explores multiple possible output sequences to find the overall most likely one.



Vocabulary = {a, b, </s>}

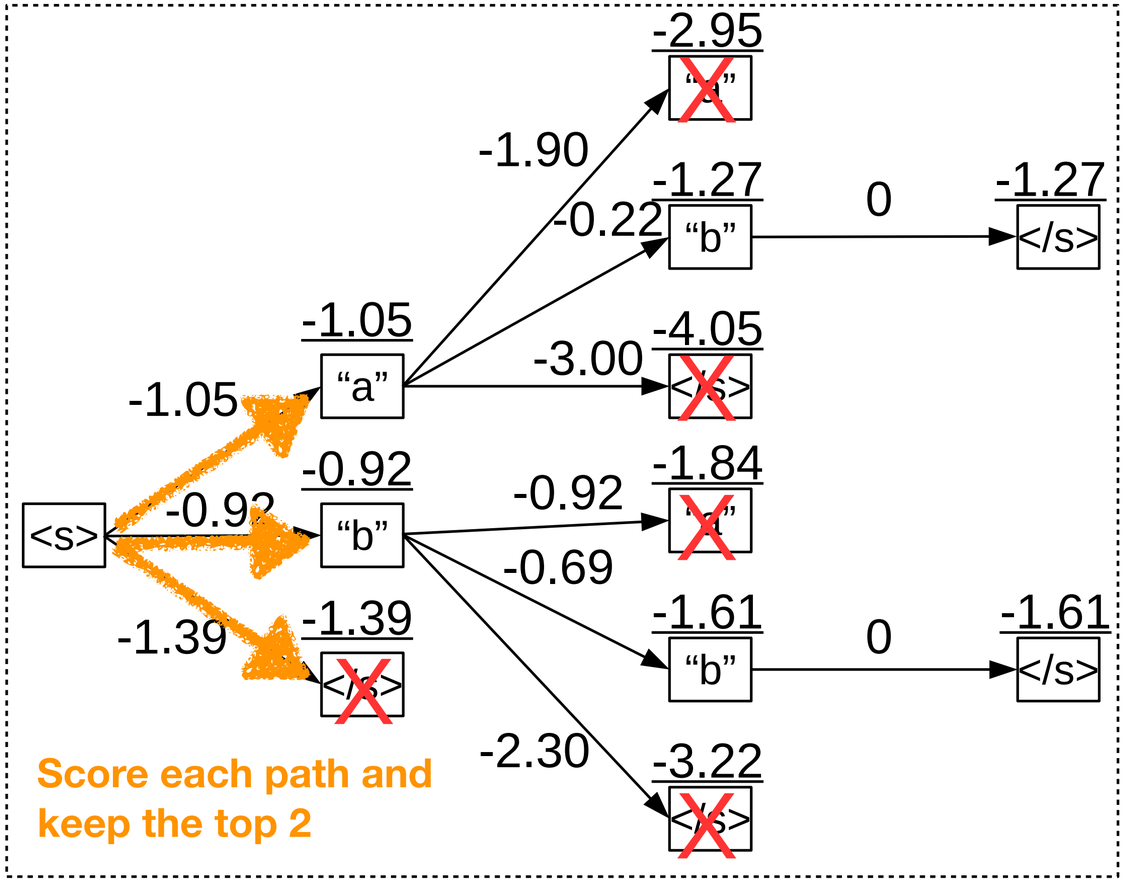
Numbers above the boxes are $\log P(x_t \mid x_{1:t-1})$

Numbers shown on edges are $\log P(x_1, \ldots, x_t)$

Recall that minimizing log probability is equivalent to maximizing probability.

Suppose we use beam search with a beam size of 2.

Beam search is an algorithm that explores multiple possible output sequences to find the overall most likely one.



Vocabulary = {a, b, </s>}

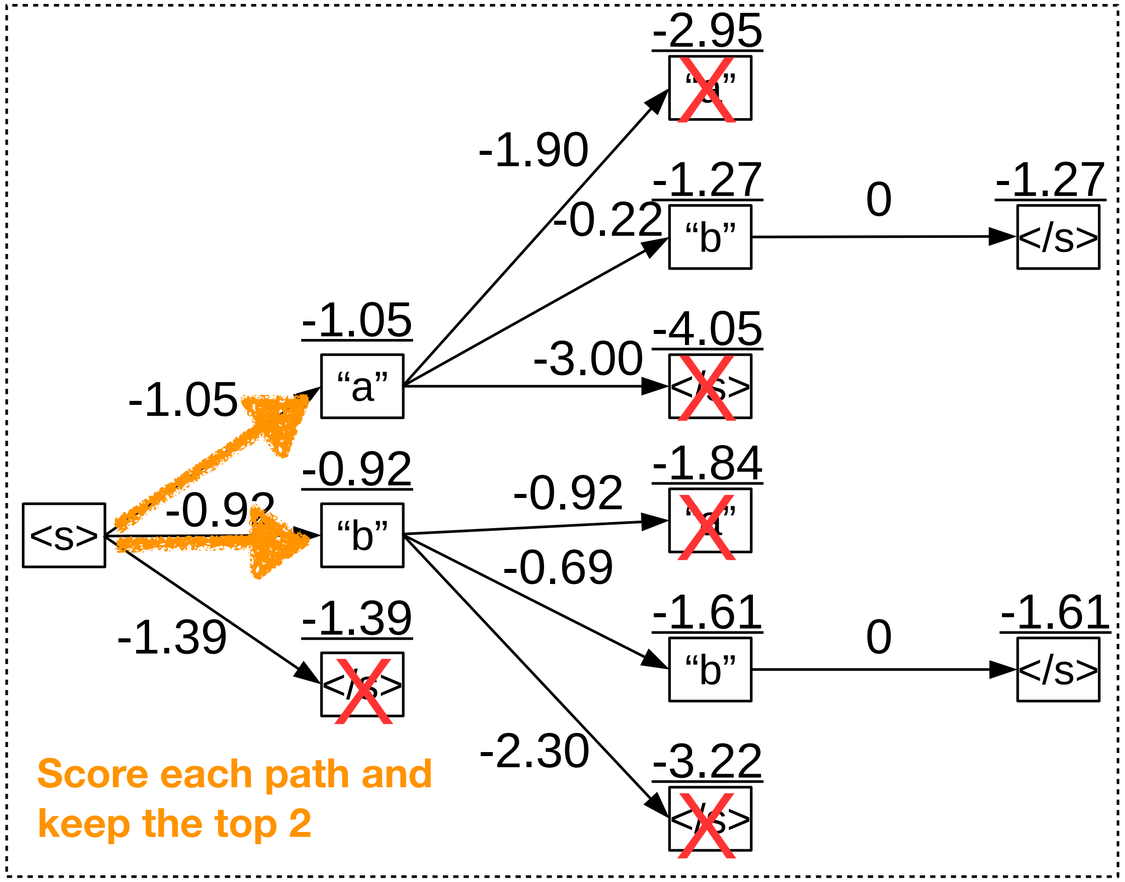
Numbers above the boxes are $\log P(x_t \mid x_{1:t-1})$

Numbers shown on edges are $\log P(x_1, \ldots, x_t)$

Recall that minimizing log probability is equivalent to maximizing probability.

Suppose we use beam search with a beam size of 2.

Beam search is an algorithm that explores multiple possible output sequences to find the overall most likely one.



Vocabulary = {a, b, </s>}

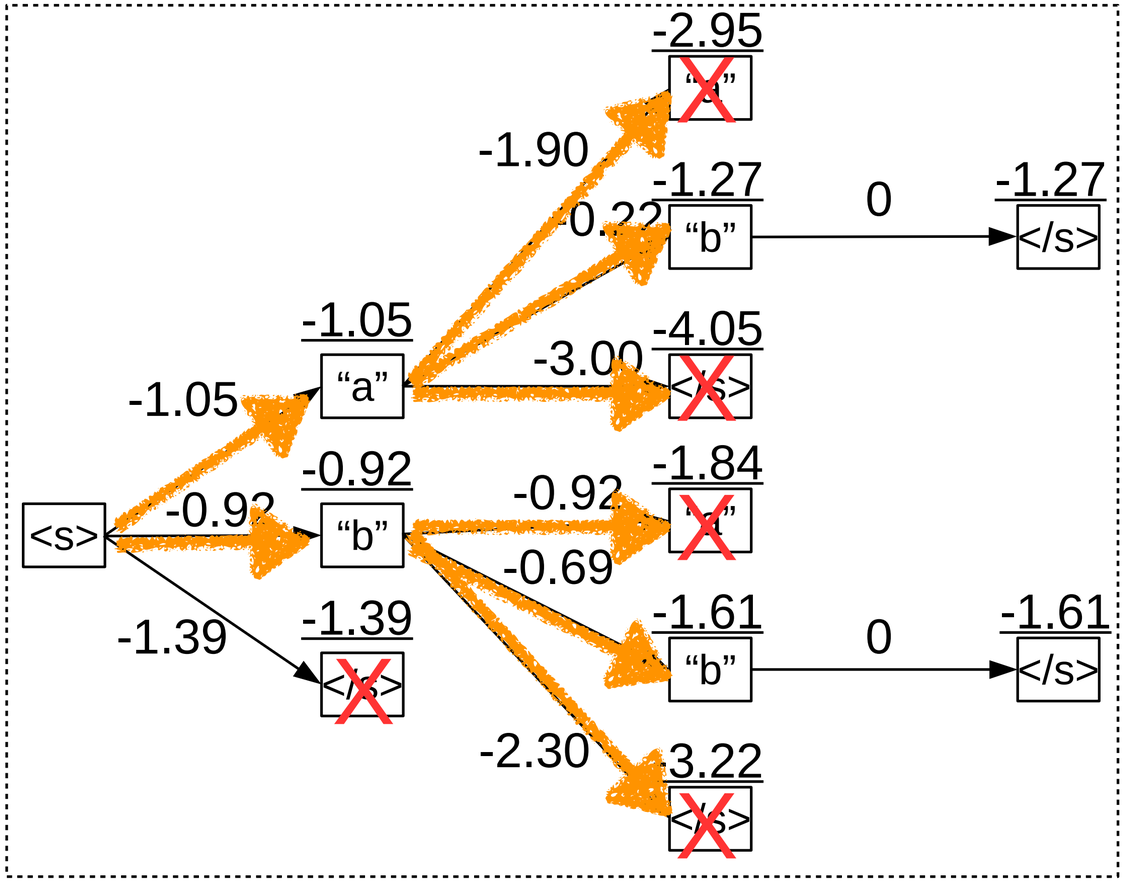
Numbers above the boxes are $\log P(x_t \mid x_{1:t-1})$

Numbers shown on edges are $\log P(x_1, \ldots, x_t)$

Recall that minimizing log probability is equivalent to maximizing probability.

Suppose we use beam search with a beam size of 2.

Beam search is an algorithm that explores multiple possible output sequences to find the overall most likely one.



Score each path and keep the top 2

Vocabulary = {a, b, </s>}

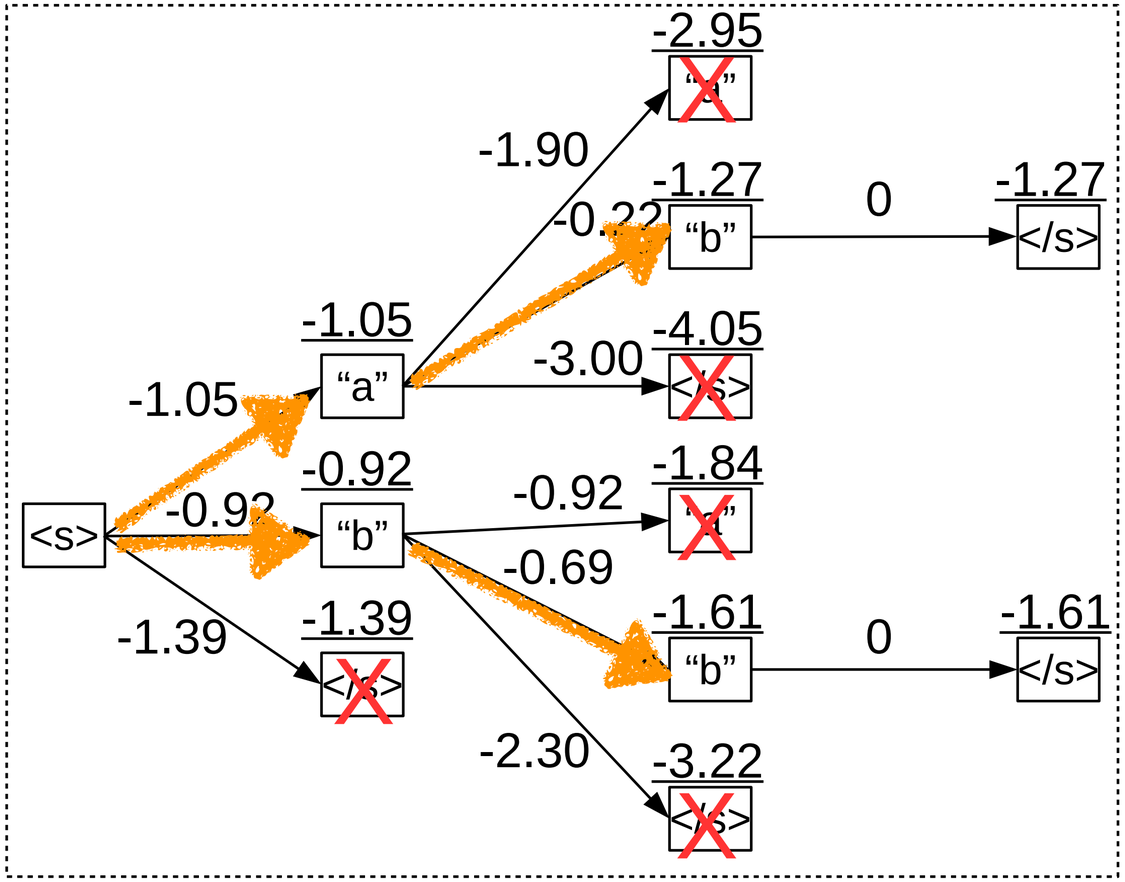
Numbers above the boxes are $\log P(x_t \mid x_{1:t-1})$

Numbers shown on edges are $\log P(x_1, \ldots, x_t)$

Recall that minimizing log probability is equivalent to maximizing probability.

Suppose we use beam search with a beam size of 2.

Beam search is an algorithm that explores multiple possible output sequences to find the overall most likely one.



Score each path and keep the top 2

Vocabulary = {a, b, </s>}

Numbers above the boxes are $\log P(x_t \mid x_{1:t-1})$

Numbers shown on edges are $\log P(x_1, \ldots, x_t)$

Recall that minimizing log probability is equivalent to maximizing probability.

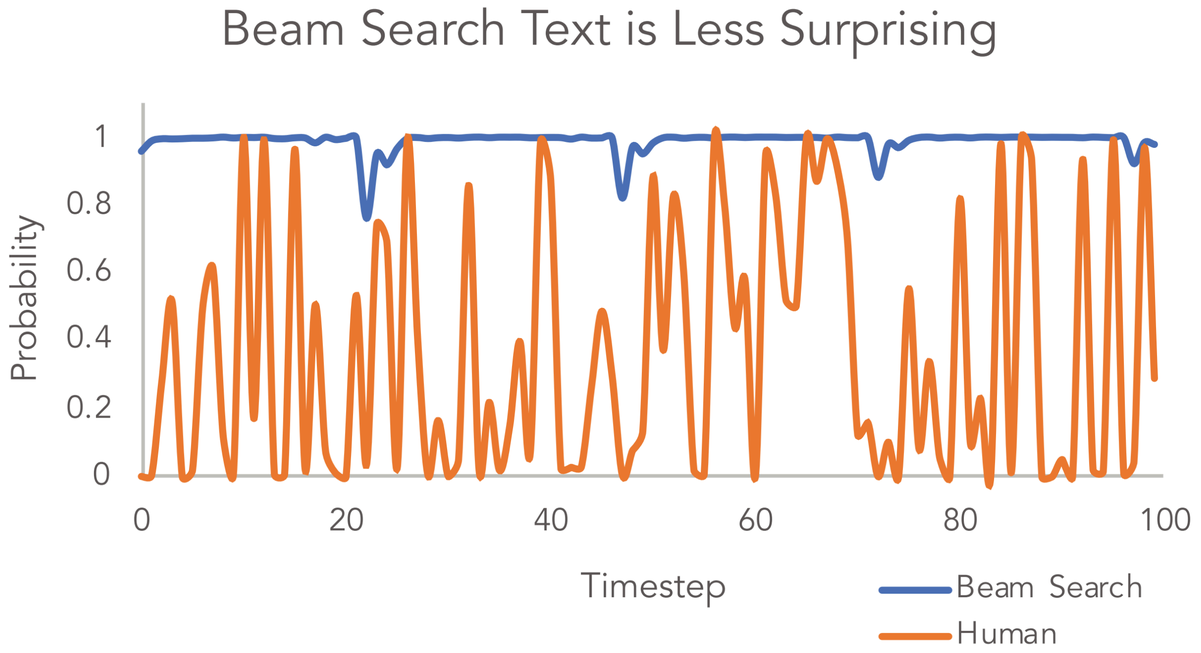
Suppose we use beam search with a beam size of 2.

Problems with Beam Search

- It turns out for open-ended tasks like dialog or story generation, optimizing for the sequence with the highest possible $P(x_1, \ldots, x_T)$ isn't actually a great idea.

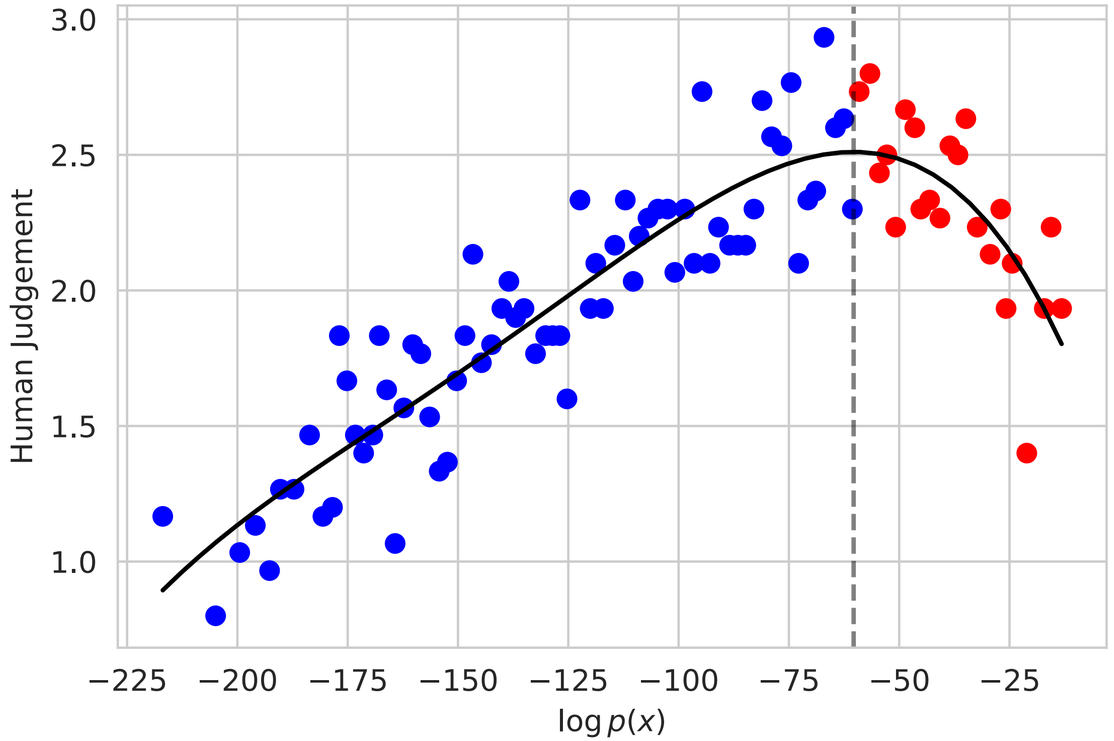
Problems with Beam Search

- It turns out for open-ended tasks like dialog or story generation, optimizing for the sequence with the highest possible $P(x_1, \ldots, x_T)$ isn't actually a great idea.
  - Beam search generates text with a very different distribution of sequence likelihoods than human-written text.



Problems with Beam Search

- It turns out for open-ended tasks like dialog or story generation, optimizing for the sequence with the highest possible $P(x_1, \ldots, x_T)$ isn't actually a great idea.
  - Beam search generates text with a very different distribution of sequence likelihoods than human-written text.
  - When sequence likelihood is too high, humans rate the text as bad.



Diverse Beam Search Algorithms

- For a long time, people tried to improve beam to make it produce more diverse text. Methods included:
  - Restrict the set of hypotheses that get considered at each step.

Method	Description	Method	Description
Random Sampling	Standard decoding mechanism, greedily samples a token from the distribution at each time step.	Random Sampling with Temperature	Before sampling, modify entropy of predicted distribution.
Top-s Random Sampling (Fan et al., 2018)	Restrict sampling to the s-most likely words in the distribution. (story generation)	Beam Search	Standard decoding mechanism, keeps the top b partial hypotheses at every time step. (machine translation)
NPAD Beam Search (Cho, 2016)	Add random noise to the hidden state of the decoder at each time step. (machine translation)	Top-g Capping Beam Search (Li and Jurafsky, 2016)	Only consider the top c hypotheses from each parent hypothesis at each time step. (machine translation, dialog)
Hamming Diversity Beam Search (Vijayakumar et al., 2016)	Penalize new hypotheses that have many of the same tokens as existing partial hypotheses. (image captioning)	Iterative Beam Search (Kulikov et al., 2018)	Run beam search several times, preventing later iterations from generating intermediate states already explored. (dialog)
Clustered Beam Search (Tam et al., 2019)	Initially consider more hypotheses at each time step, and then cluster similar hypotheses together. (dialog)	Post-Decoding Clustering (Ours)	Sample a large number of candidates, and then cluster similar outputs together.

Diverse Beam Search Algorithms

- For a long time, people tried to improve beam to make it produce more diverse text. Methods included:
  - Restrict the set of hypotheses that get considered at each step.
  - Incorporate diversity into the scoring function used to rank the current hypothesis set.

Method	Description	Method	Description
Random Sampling	Standard decoding mechanism, greedily samples a token from the distribution at each time step.	Random Sampling with Temperature	Before sampling, modify entropy of predicted distribution.
Top-s Random Sampling (Fan et al., 2018)	Restrict sampling to the s-most likely words in the distribution. (story generation)	Beam Search	Standard decoding mechanism, keeps the top b partial hypotheses at every time step. (machine translation)
NPAD Beam Search (Cho, 2016)	Add random noise to the hidden state of the decoder at each time step. (machine translation)	Top-g Capping Beam Search (Li and Jurafsky, 2016)	Only consider the top c hypotheses from each parent hypothesis at each time step. (machine translation, dialog)
Hamming Diversity Beam Search (Vijayakumar et al., 2016)	Penalize new hypotheses that have many of the same tokens as existing partial hypotheses. (image captioning)	Iterative Beam Search (Kulikov et al., 2018)	Run beam search several times, preventing later iterations from generating intermediate states already explored. (dialog)
Clustered Beam Search (Tam et al., 2019)	Initially consider more hypotheses at each time step, and then cluster similar hypotheses together. (dialog)	Post-Decoding Clustering (Ours)	Sample a large number of candidates, and then cluster similar outputs together.

Diverse Beam Search Algorithms

- For a long time, people tried to improve beam to make it produce more diverse text. Methods included:
  - Restrict the set of hypotheses that get considered at each step.
  - Incorporate diversity into the scoring function used to rank the current hypothesis set.
  - Add noise to the model weights to encourage diversity

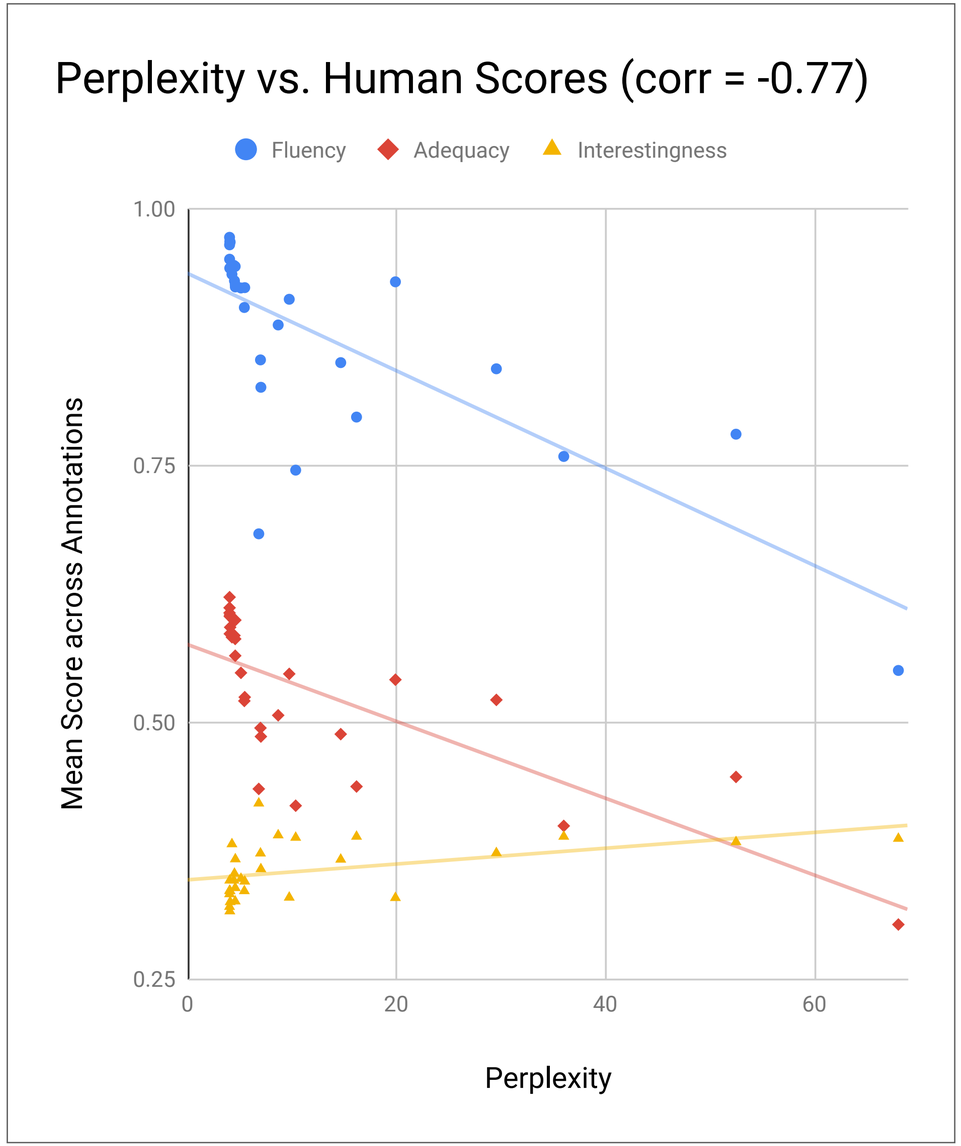
Method	Description	Method	Description
Random Sampling	Standard decoding mechanism, greedily samples a token from the distribution at each time step.	Random Sampling with Temperature	Before sampling, modify entropy of predicted distribution.
Top-s Random Sampling (Fan et al., 2018)	Restrict sampling to the s-most likely words in the distribution. (story generation)	Beam Search	Standard decoding mechanism, keeps the top b partial hypotheses at every time step. (machine translation)
NPAD Beam Search (Cho, 2016)	Add random noise to the hidden state of the decoder at each time step. (machine translation)	Top-g Capping Beam Search (Li and Jurafsky, 2016)	Only consider the top c hypotheses from each parent hypothesis at each time step. (machine translation, dialog)
Hamming Diversity Beam Search (Vijayakumar et al., 2016)	Penalize new hypotheses that have many of the same tokens as existing partial hypotheses. (image captioning)	Iterative Beam Search (Kulikov et al., 2018)	Run beam search several times, preventing later iterations from generating intermediate states already explored. (dialog)
Clustered Beam Search (Tam et al., 2019)	Initially consider more hypotheses at each time step, and then cluster similar hypotheses together. (dialog)	Post-Decoding Clustering (Ours)	Sample a large number of candidates, and then cluster similar outputs together.

When to use standard beam search:

- Your domain is relatively closed (for example, machine translation)
- Your language model is not very good (you don't trust the $P(x_t \mid x_{1:t-1})$ it returns)

When to use one of the diverse beam search methods discussed in paper:

- Almost never, especially if your language model is good.



OUTLINE

- Decoding Strategy Recap
- **Automatic Detection of Generated Text**
- Why is it difficult to answer the question “which decoding strategy is best”?

Why are we interested in systems that automatically detect generated text?

- Combat the propagation of fake text
- Improve training of text generation models (adversarial training)
- Evaluate the quality of generated text

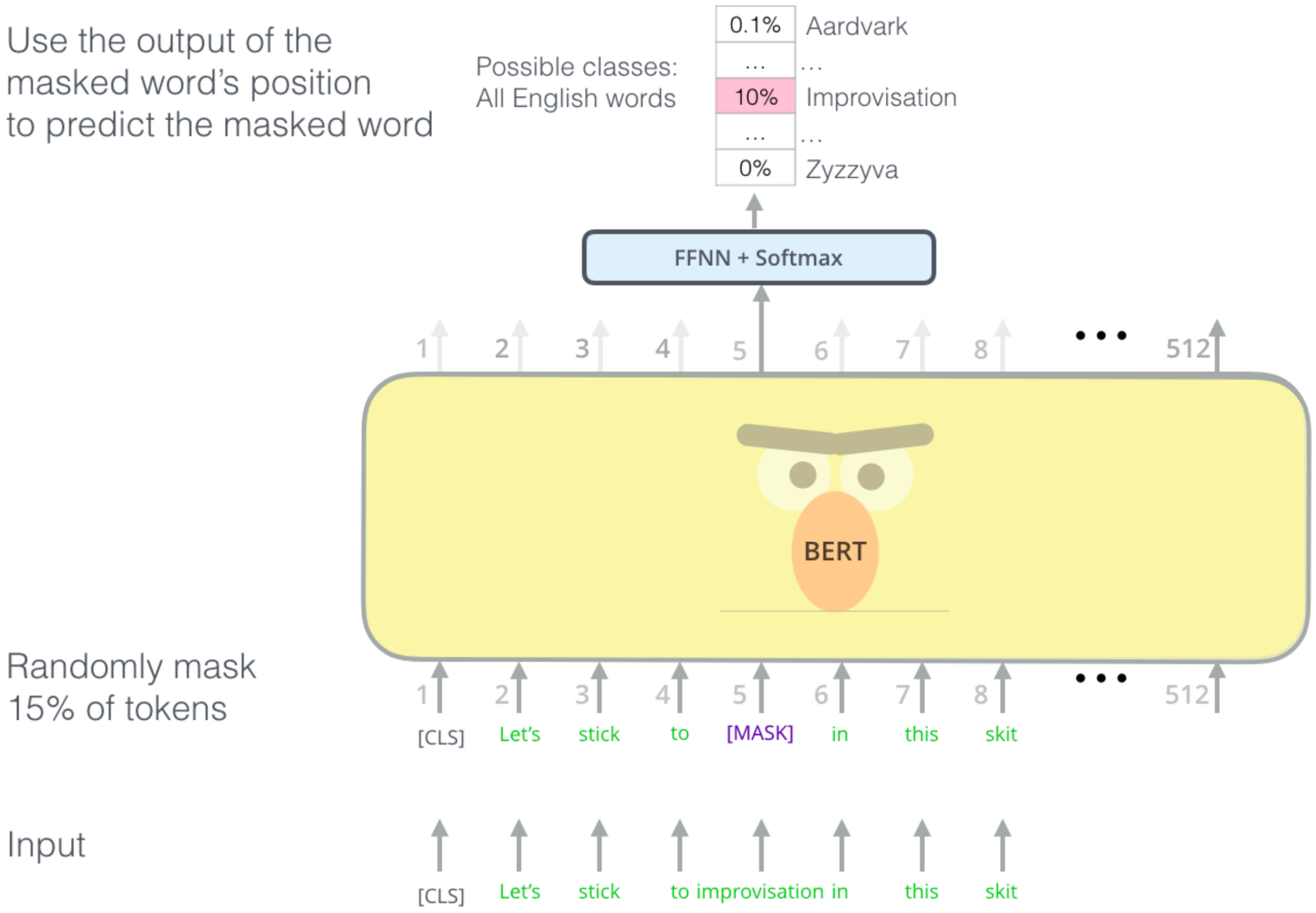
Method for Building a Detector

- Train a simple classifier on top of a bag-of-words representation of the text
- Compute a histogram of the token likelihoods $P(x_t \mid x_{1:t-1})$ over all the tokens in the text, then train a simple classier on top of the histogram. http://gltr.io/
- Train a neural network to make a prediction given a text sequence
  - Train from scratch
  - Fine-tune for classification the same language model that was used for generating the samples
  - Fine-tune some other pre-trained language model on the detection classification task.

Method for Building a Detector

- Train a simple classifier on top of a bag-of-words representation of the text
- Compute a histogram of the token likelihoods $P(x_t \mid x_{1:t-1})$ over all the tokens in the text, then train a simple classier on top of the histogram. http://gltr.io/
- Train a neural network to make a prediction given a text sequence
  - Train from scratch
  - Fine-tune for classification the same language model that was used for generating the samples
  - Fine-tune some other pre-trained language model on the detection classification task.

Bidirectional Encoder Representations from Transformers (BERT)



Credit: http://jalammar.github.io/illustrated-bert/

Method

- I fine-tuned each classifier on ~200,000 excerpts of web text and ~200,000 excerpts of text that were generated by GPT-2.
- Classifiers were trained to perform binary classification: predicting whether an excerpt was human-written or machine-generated (from GPT-2 XL).
- In total, I had 6 datasets, each with ~400,000 examples in it:
  - Both with and without priming:

Examples with priming:
[start] Once upon -> a time there was a beautiful ogre.
[start] Today -> is going to be a great day.

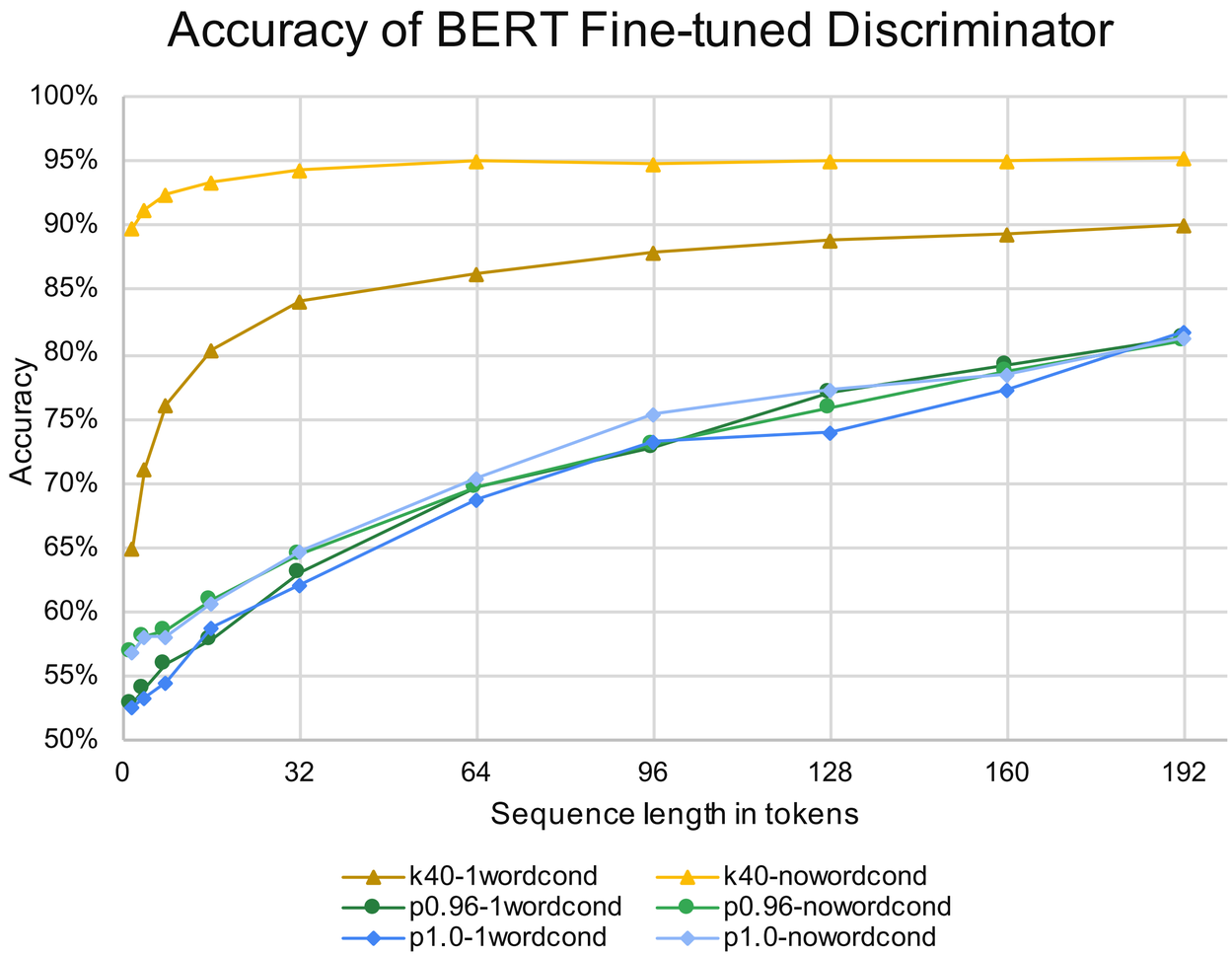
Example without priming:
[start] -> Today it is going to rain cats and dogs.

Method

- I fine-tuned each classifier on ~200,000 excerpts of web text and ~200,000 excerpts of text that were generated by GPT-2.
- Classifiers were trained to perform binary classification: predicting whether an excerpt was human-written or machine-generated.
- In total, I had 6 datasets, each with ~400,000 examples in it:
  - Both with and without priming:
    - One where the machine-generated text was sampled using top-k sampling with k=50
    - One where the machine-generated text was sampled using nucleus sampling with p=0.96
    - One where the $P(x_t \mid x_{1:t-1})$ returned by the LM was used without modification (I'll refer too this as p=1.0)

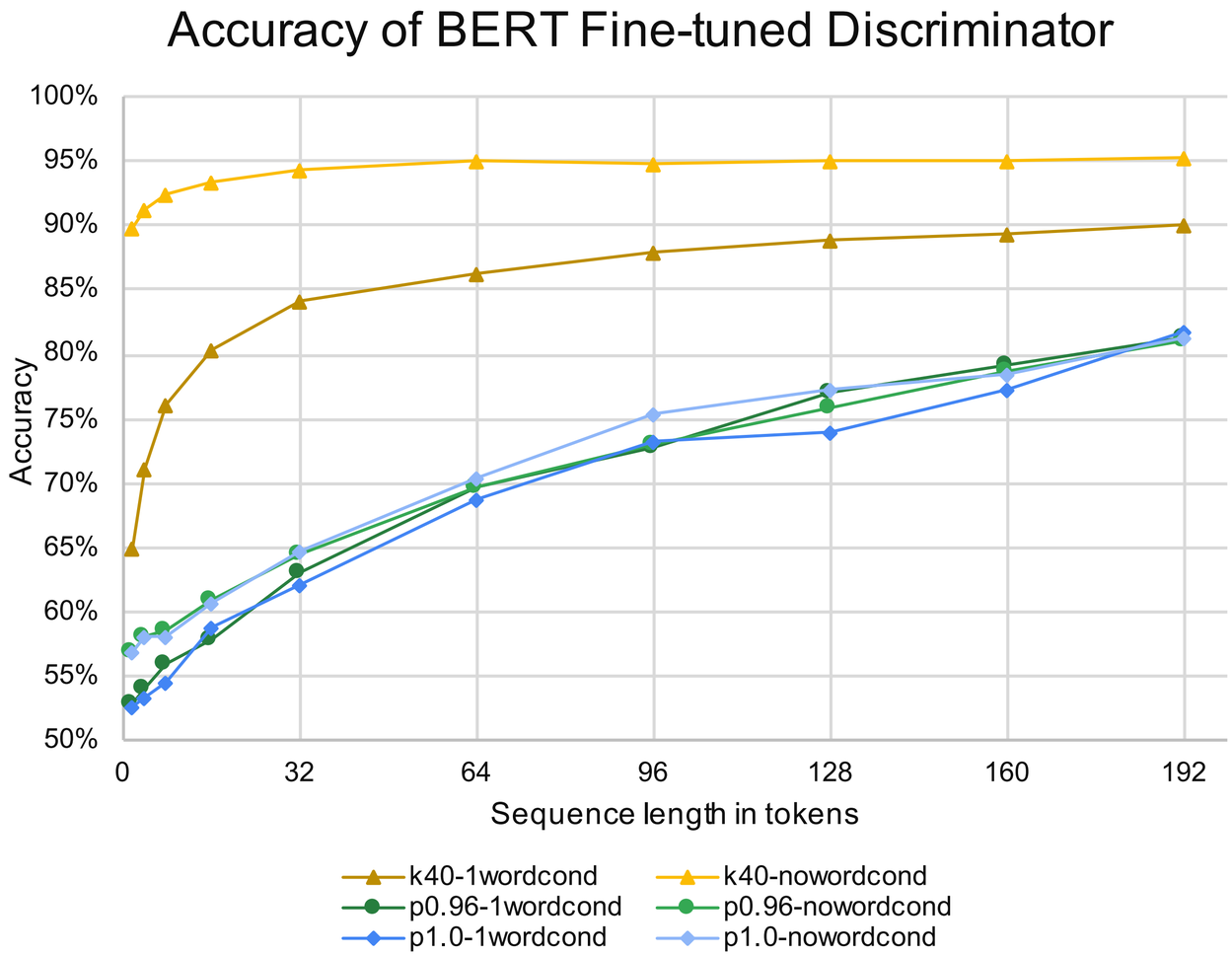
QUESTIONS OF INTEREST

- How does accuracy vary by sequence length?



QUESTIONS OF INTEREST

- How does accuracy vary by sequence length?
- Why are accuracies so much higher for top-k than the other strategies?

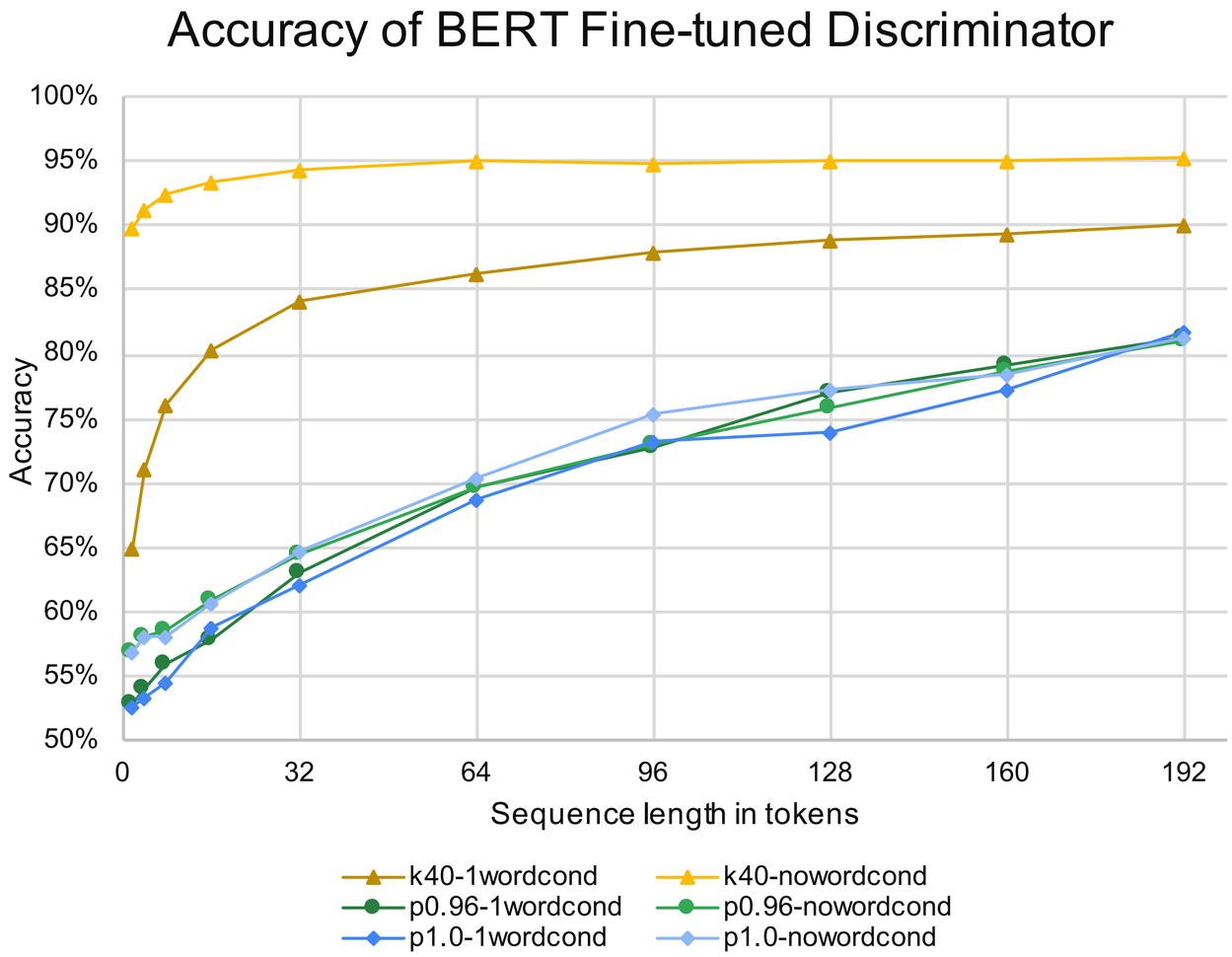


QUESTIONS OF INTEREST

- How does accuracy vary by sequence length?
- Why are accuracies so much higher for top-k than the other strategies?

QUESTIONS OF INTEREST

- How does accuracy vary by sequence length?
- Why are accuracies so much higher for top-k than the other strategies?
- Why are accuracies well above random chance even for very short sequence lengths?



For sequences of length 2, BERT gets 65% accuracy if there is some priming text, 90% accuracy if not.

CAN YOU DO IT?

Recall that top-k (with $k = 40$) means that there are only 40 possible tokens the language model can generate in the first position.

For each of the following excerpts, predict whether it's human-written or machine-generated, assuming top-k sampling was used.

1. "The cat"

CAN YOU DO IT?

Recall that top-k (with $k = 40$) means that there are only 40 possible tokens the language model can generate in the first position.

For each of the following excerpts, predict whether it's human-written or machine-generated, assuming top-k sampling was used.

1. "The cat"

2. “Felines are”

CAN YOU DO IT?

For each of the following excerpts predict whether it's human-written or machine-generated.

BERT trained on generated text that had no priming would predict….

1. "The cat"

machine-generated

2. "Felines are"

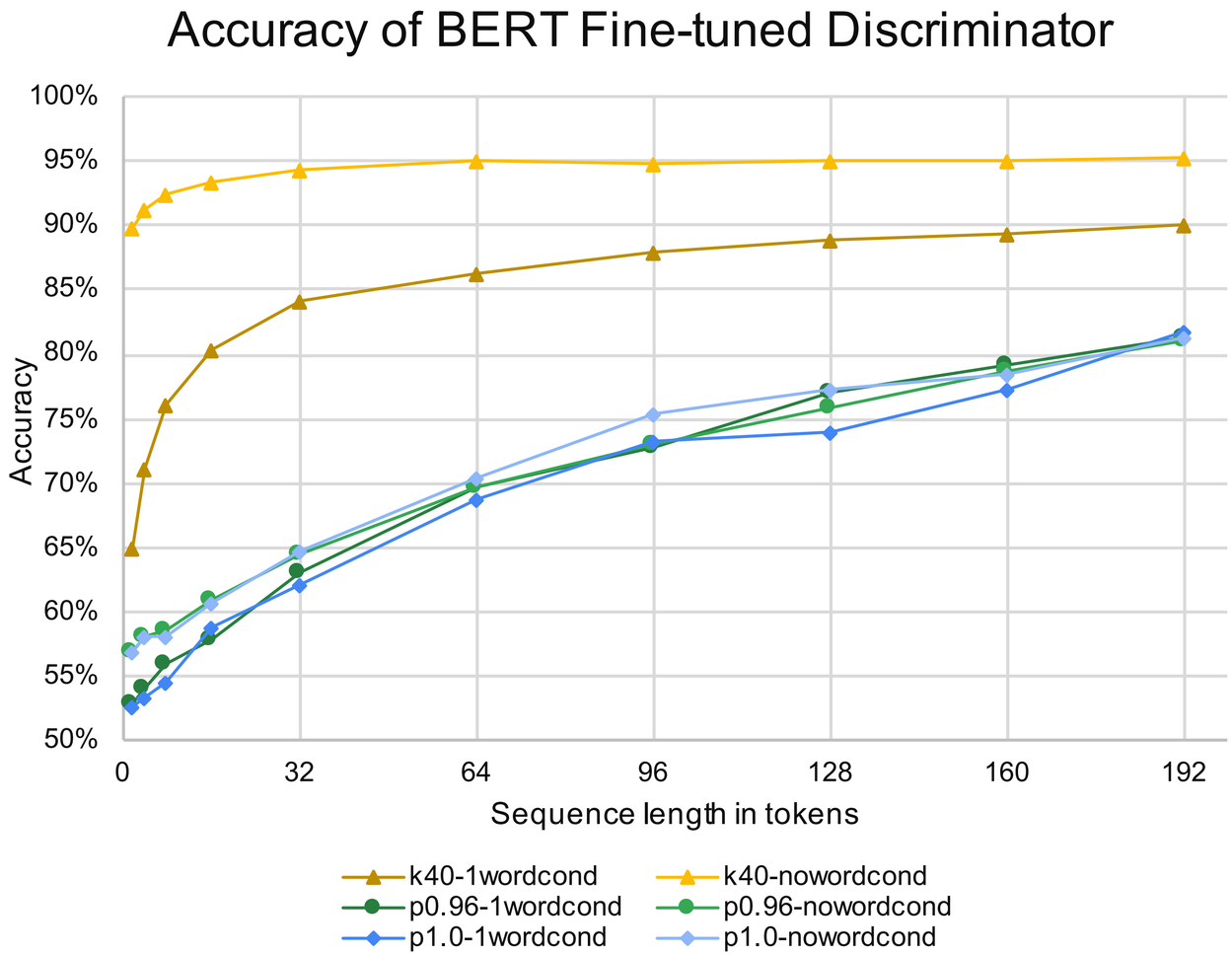
human-written

If we instead primed the language model with a bunch of text for it to continue, the detection task would be harder because there are more options for the next token.

P(next word | "I am") vs P(next word | "The monstrous") look very different.

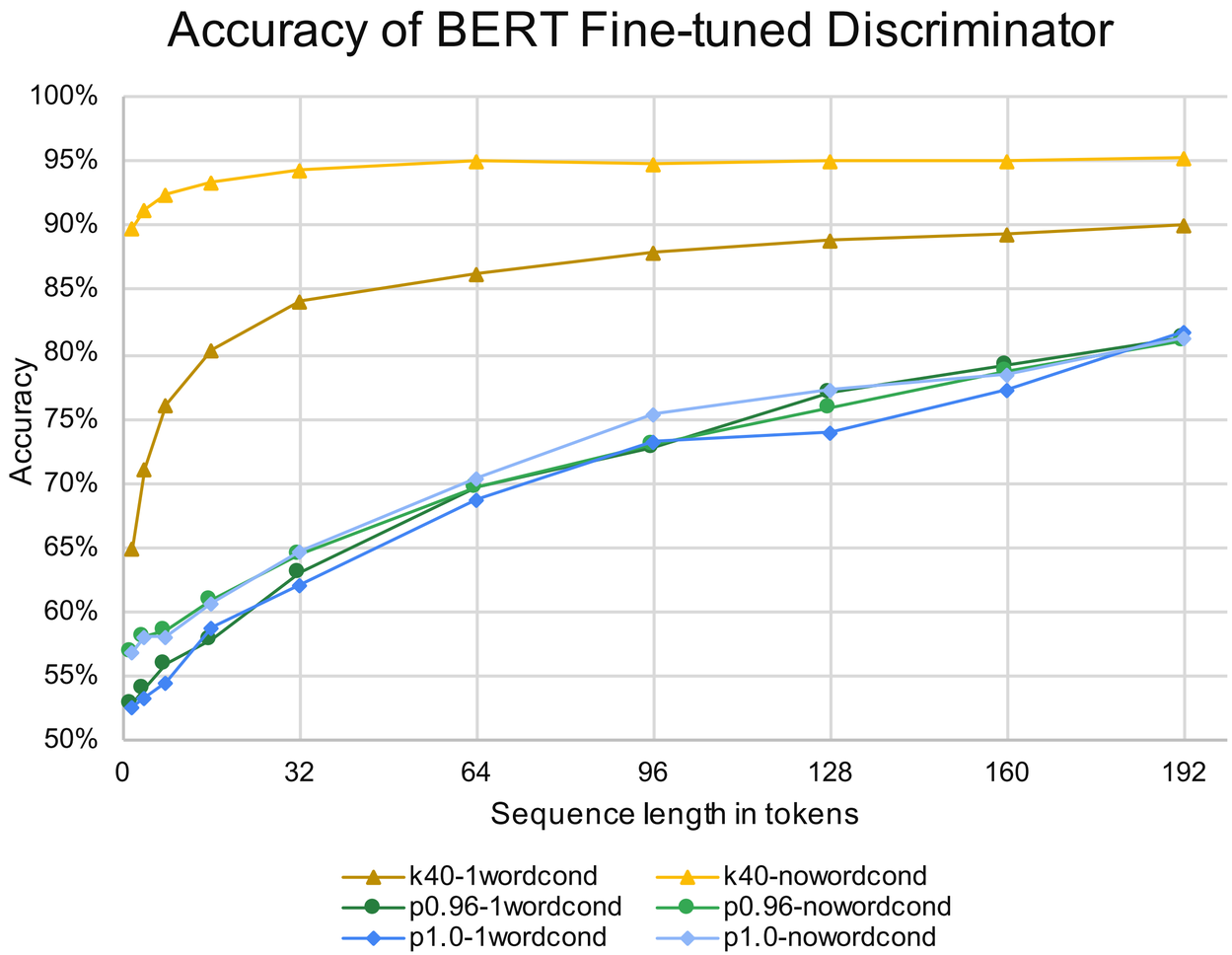
QUESTIONS OF INTEREST

- How does accuracy vary by sequence length?
- Why are accuracies so much higher for top-k than the other strategies?
- Why are accuracies well above random chance even for very short sequence lengths?
- Why does priming the language model with some text make a big difference for top-k?



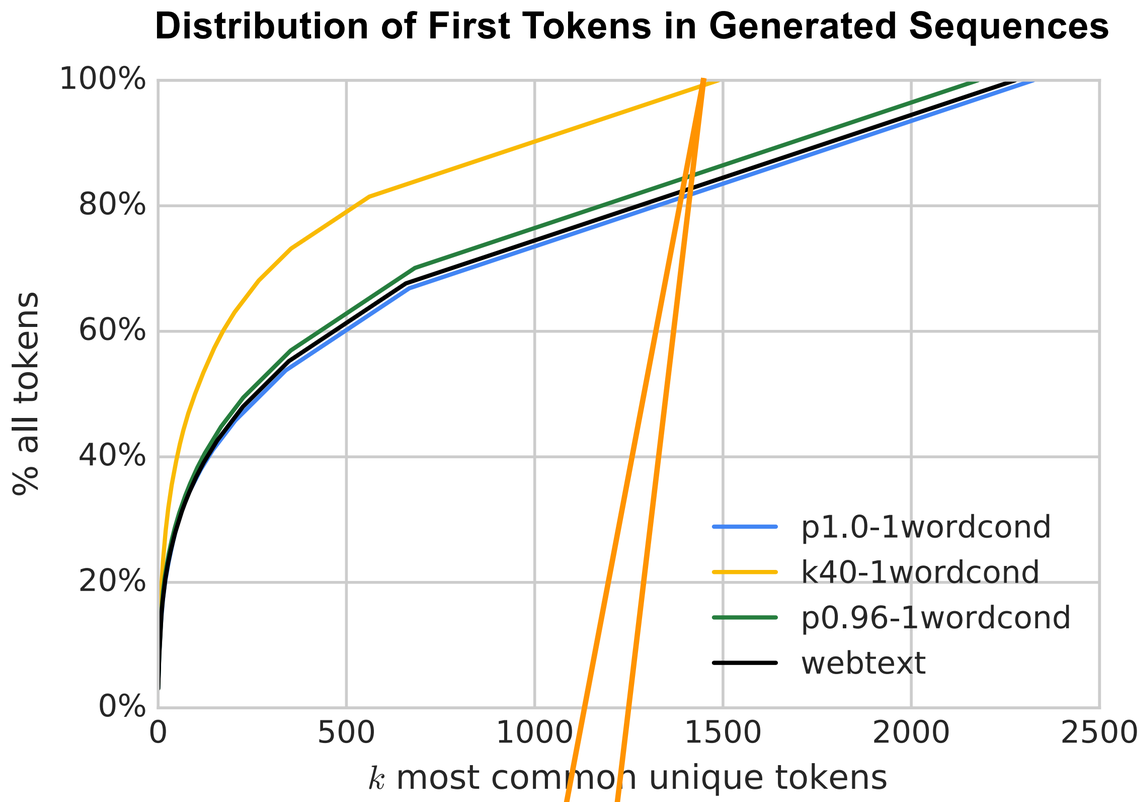
QUESTIONS OF INTEREST

- How does accuracy vary by sequence length?
- Why are accuracies so much higher for top-k than the other strategies?
- Why are accuracies well above random chance even for very short sequence lengths?
- Why does priming the language model with some text make a big difference for top-k?
- Why does top-k look so different?



QUESTIONS OF INTEREST

- How does accuracy vary by sequence length?
- Why are accuracies so much higher for top-k than the other strategies?
- Why are accuracies well above random chance even for very short sequence lengths?
- Why does priming the language model with some text make a big difference for top-k?
- Why does top-k look so different?

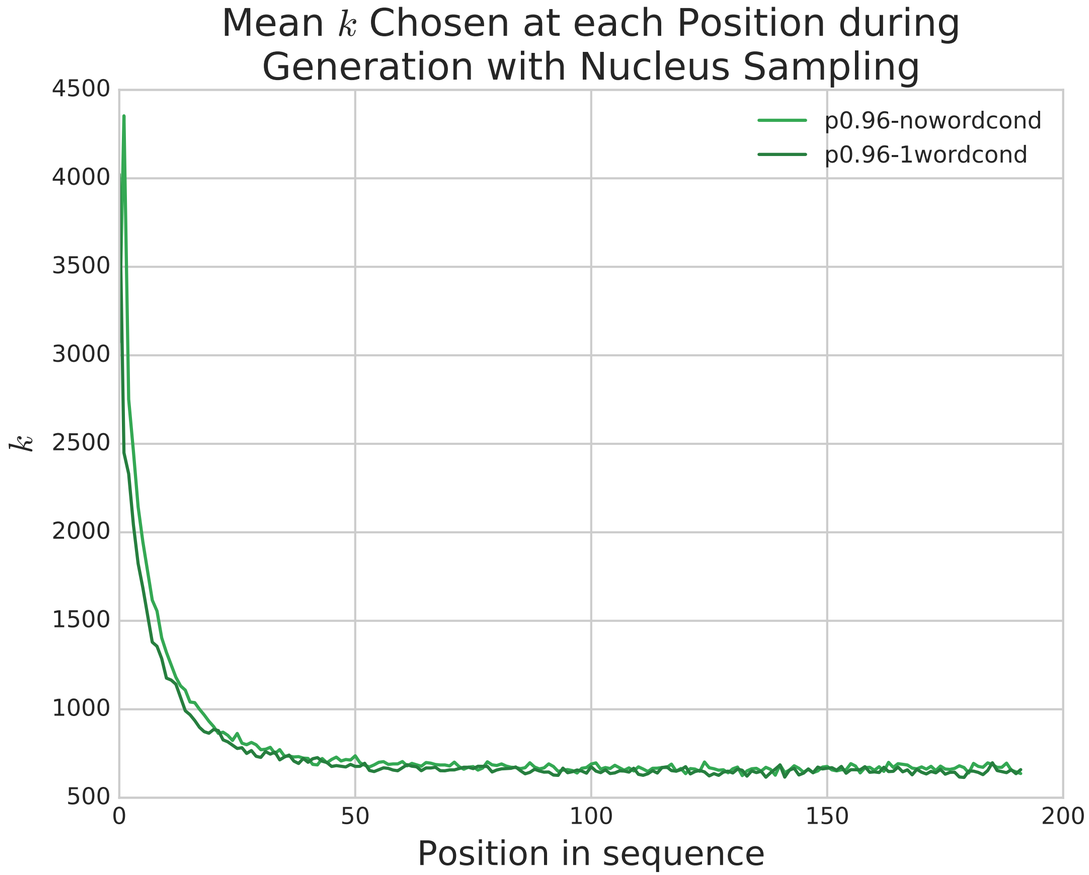


Even when using a word of priming, for top-k samples, the 1500 most common tokens form 100% of the first words in the generated sequences.

TOP-K IS NOT BAD, THE METHODS ARE JUST IMBALANCED

Recall that nucleus sampling chooses a k_t at every sampling step such that the total probability of the k_t most likely words is as close as possible to some constant p.

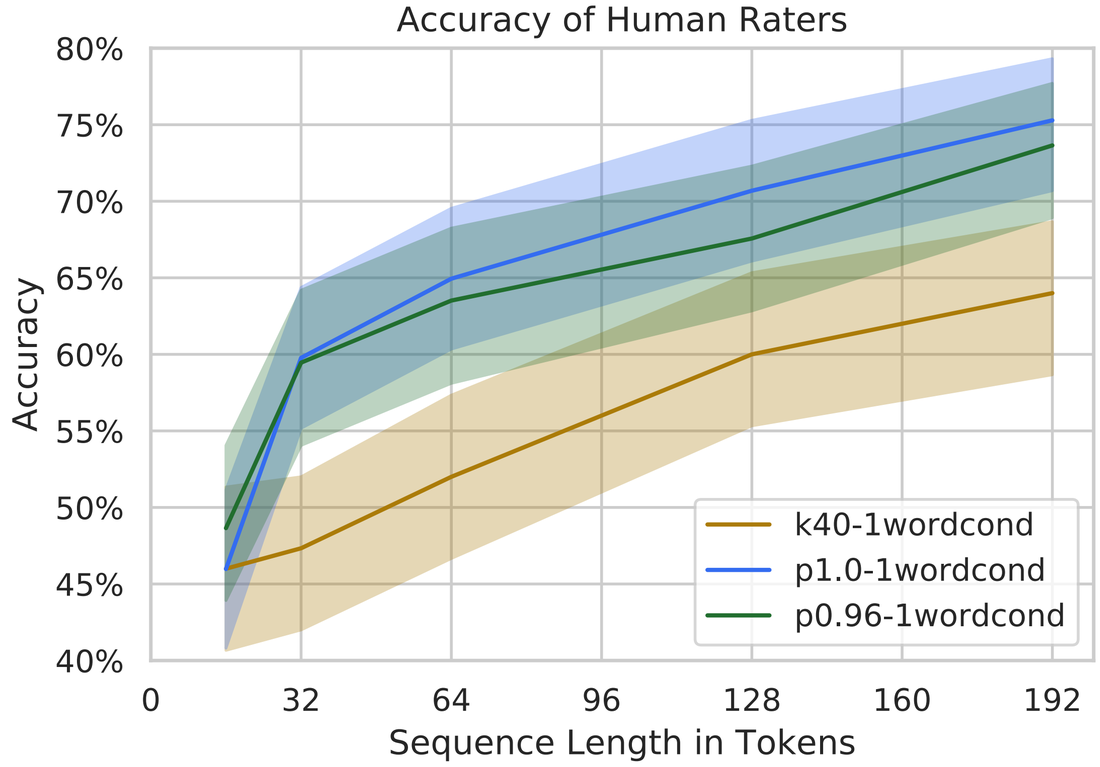
In our experiments we set $p = 0.96$. This meant that most of the time the k_t chosen by nucleus sampling was a lot bigger than the constant value of $k = 40$ we were using for our top-k experiments.



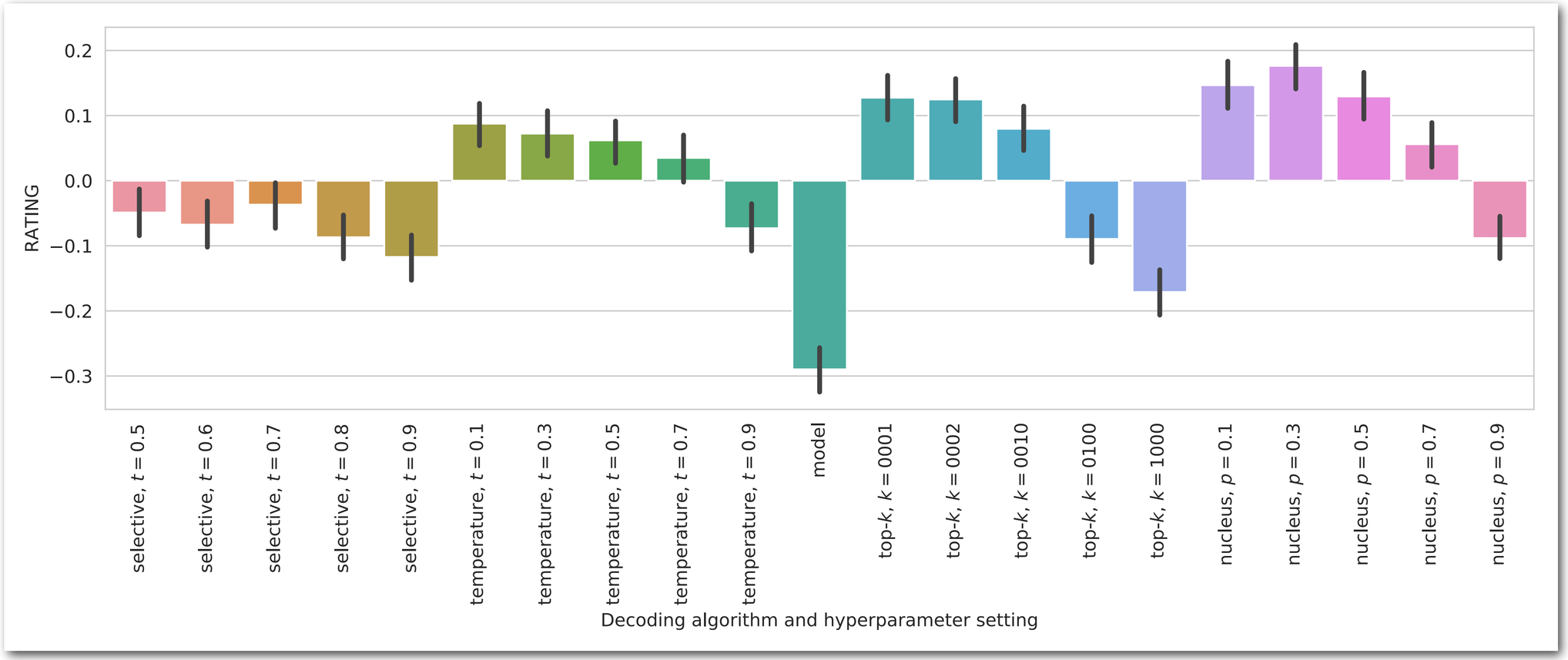
OUTLINE

- Decoding Strategy Recap
- Automatic Detection of Generated Text
- **Why is it difficult to answer the question “which decoding strategy is best”?**

Recall that lower accuracy means that humans had a harder time distinguishing these samples from human-written ones.



Human judged quality of generated tex:



OUR RELATIVE STRENGTHS

Humans are good at detecting:

- Co-reference errors
- Contradictions
- Falsehoods or statements unlikely to be true
- Incorrect uses of a word
- Lack of fluency

Automatic systems are good at detecting:

- Differences in token frequencies
- Differences in the patterns of token likelihoods

Neural networks are lazy. They will learn semantic information (like the properties listed on the left) if they need to, but if there is some easier signal to pick up on, they will take advantage of that first.

TRADEOFFS IN DECODING

Sample from full distribution ↔ Reduce likelihood of already low likelihood words

Diversity ↔ Quality

Fool Machines ↔ Fool Humans

CONCLUSIONS

- Even the best language models aren't good enough at modeling language for us to sample from the full distribution and not make bad word choices.
- Reducing the weight of words in the tail decreases the chance we'll make a bad word choice, but it also reduces the chance we'll make interesting good word choices.
- Sampling from the tail of LM distributions, but sampling from the tail is necessary to get diverse text.

ANY QUESTIONS?