CMSC 473/673 Natural Language Processing

Instructor: Lara J. Martin (she/they)

TA: Duong Ta (he)

Slides modified from Dr. Frank Ferraro & Dr. Daphne Ippolito

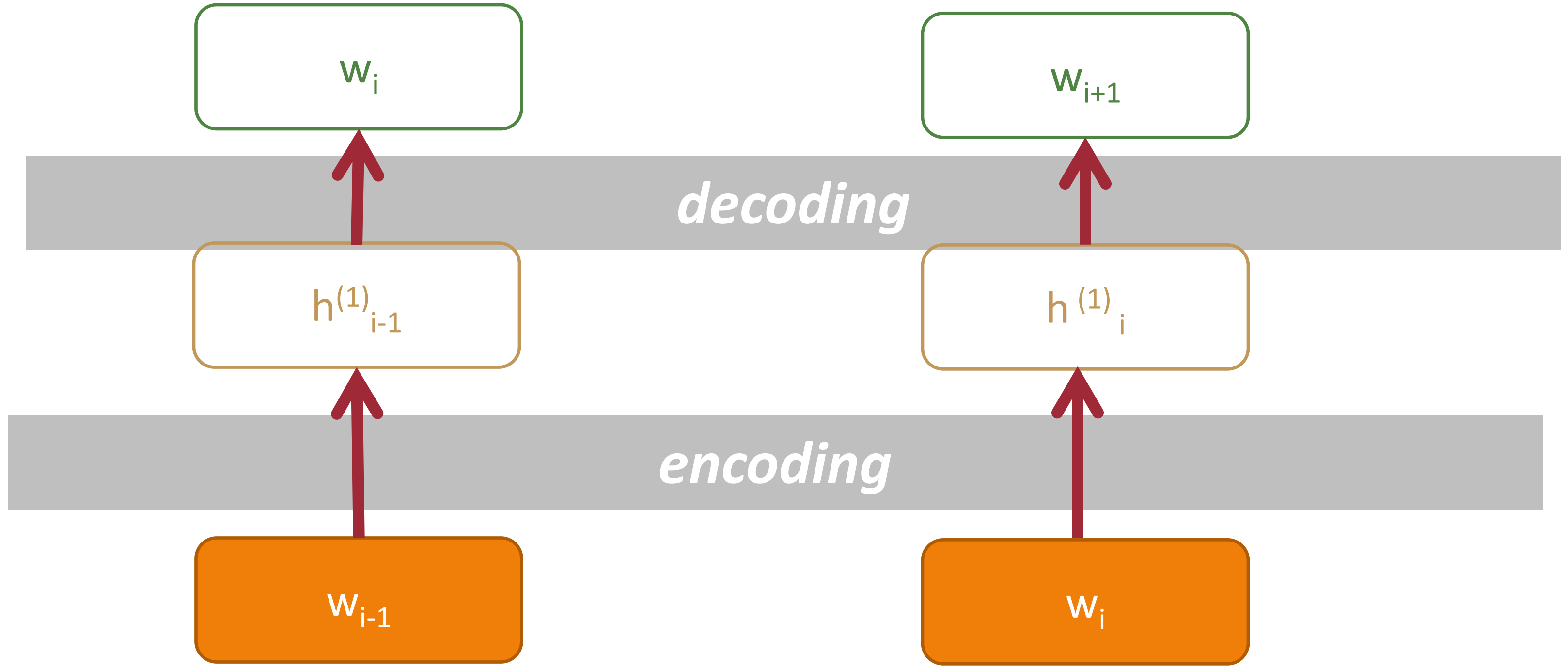
Learning Objectives

Differentiate between encoding/decoding and encoder-decoder networks

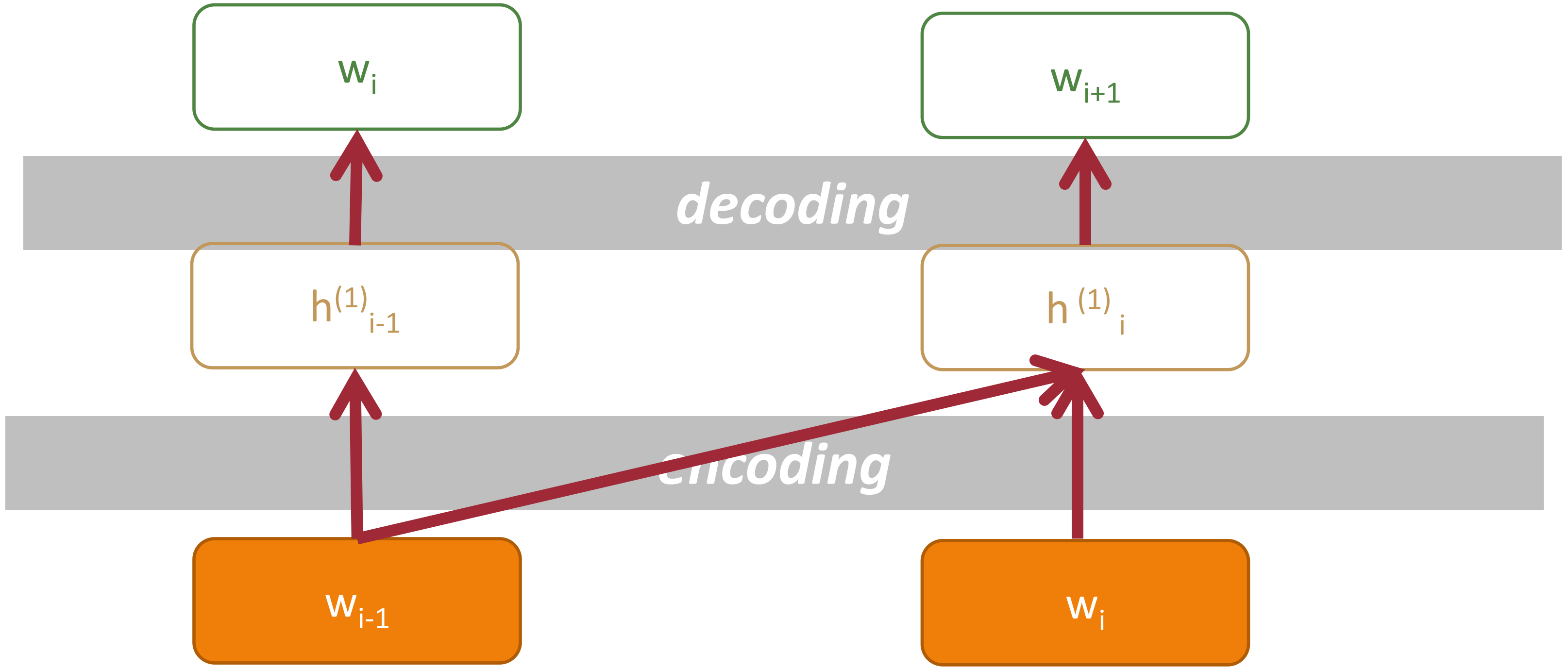
Consider when to use various sampling algorithms

Discuss the uses of finetuning

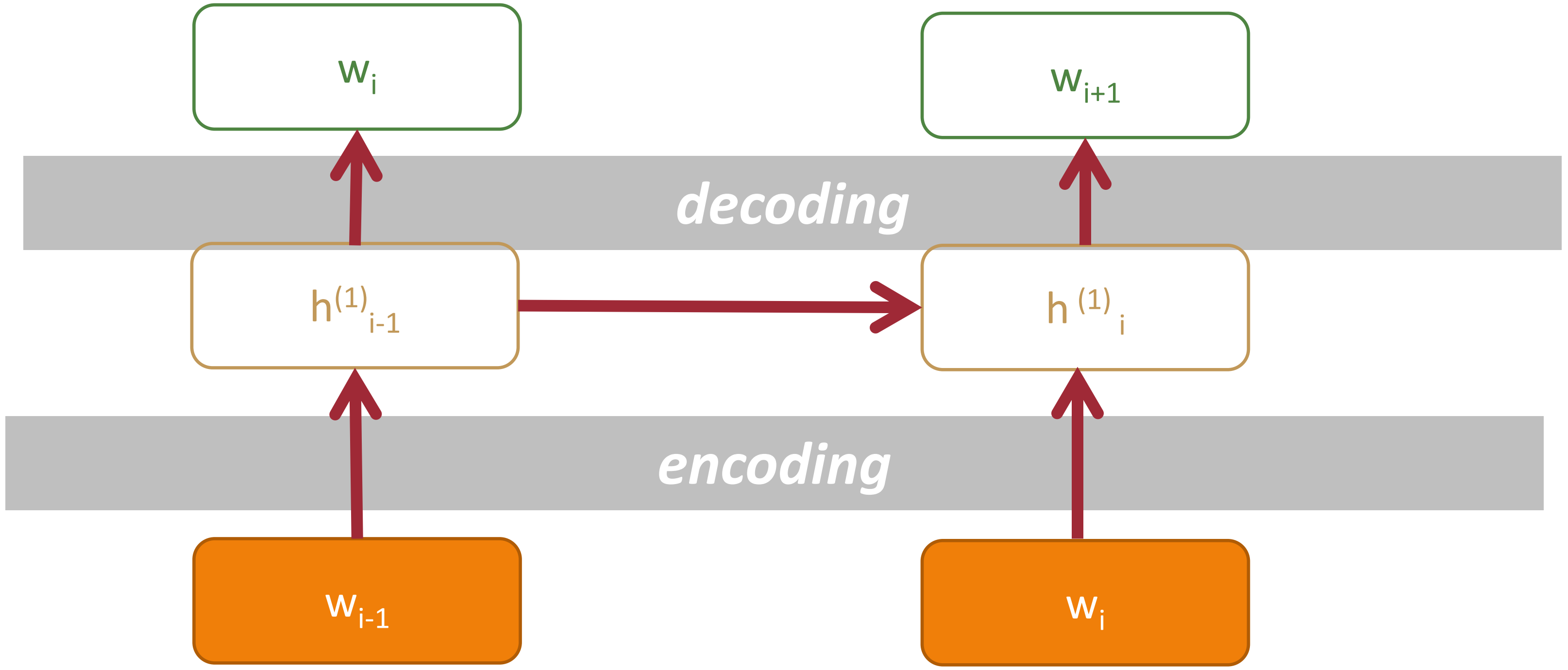
Feedforward Network



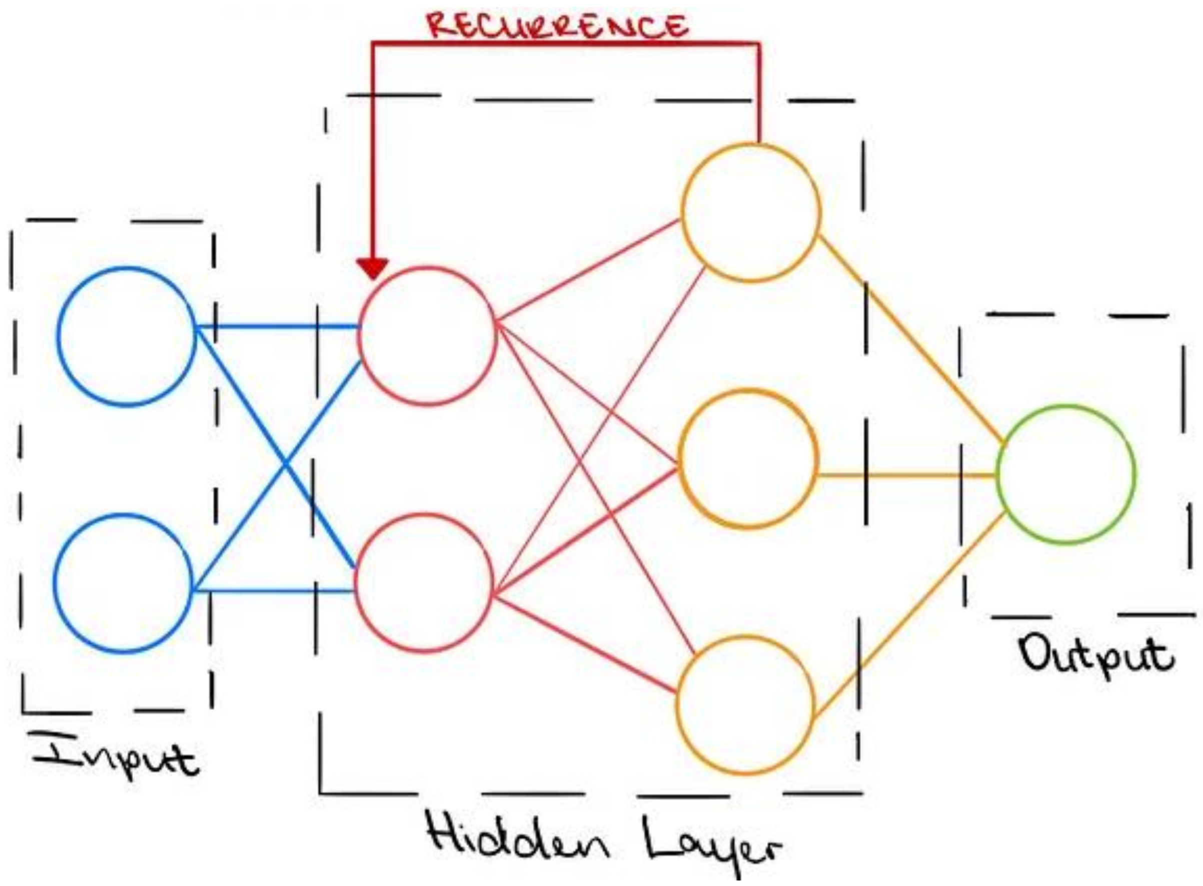
Tri-gram Feedforward Neural Network



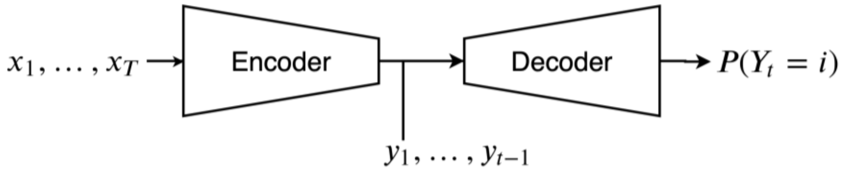
Recurrent Neural Network



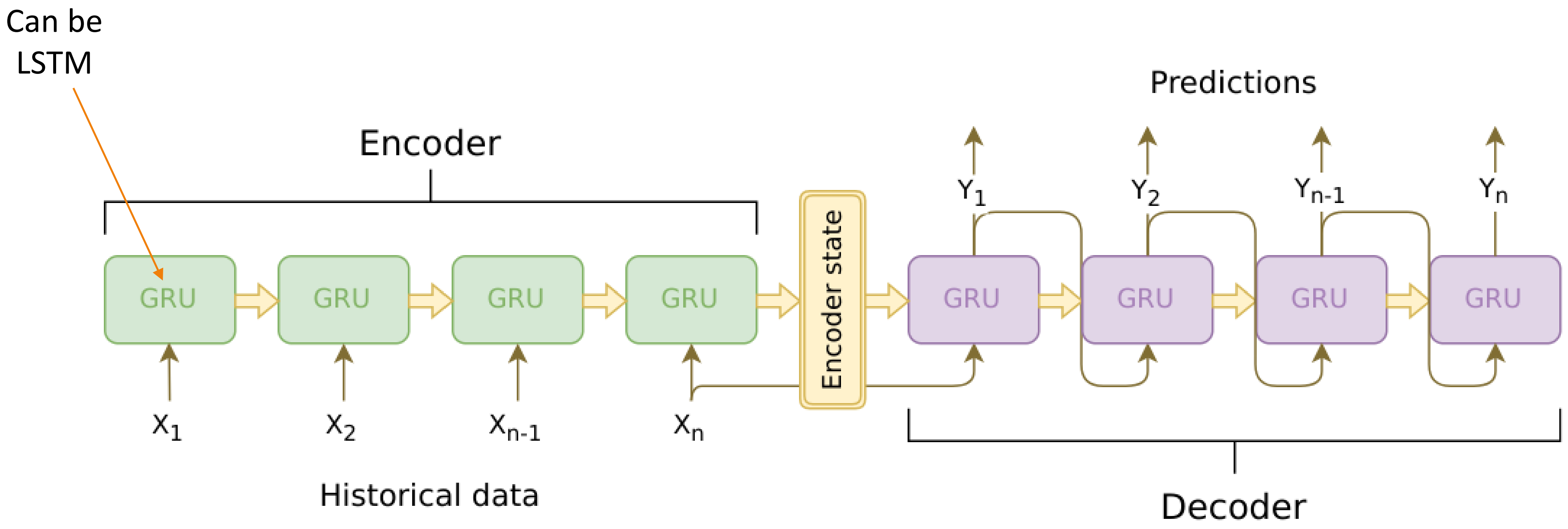
Another way of illustrating it



https://towardsdatascience.com/introducing-recurrent-neural-networks-f359653d7020



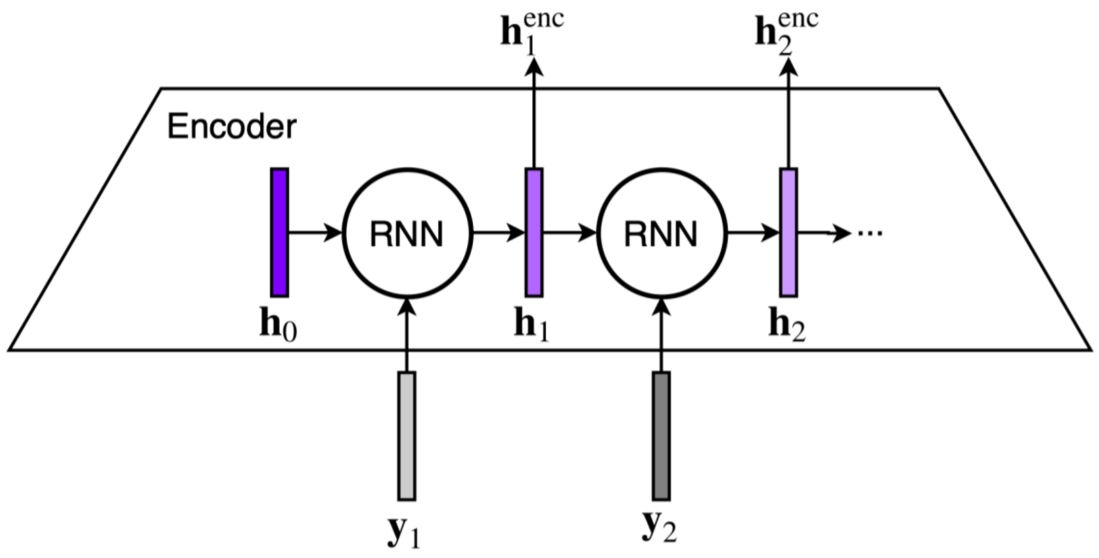
Sequence-to-Sequence / Encoder-Decoder Models

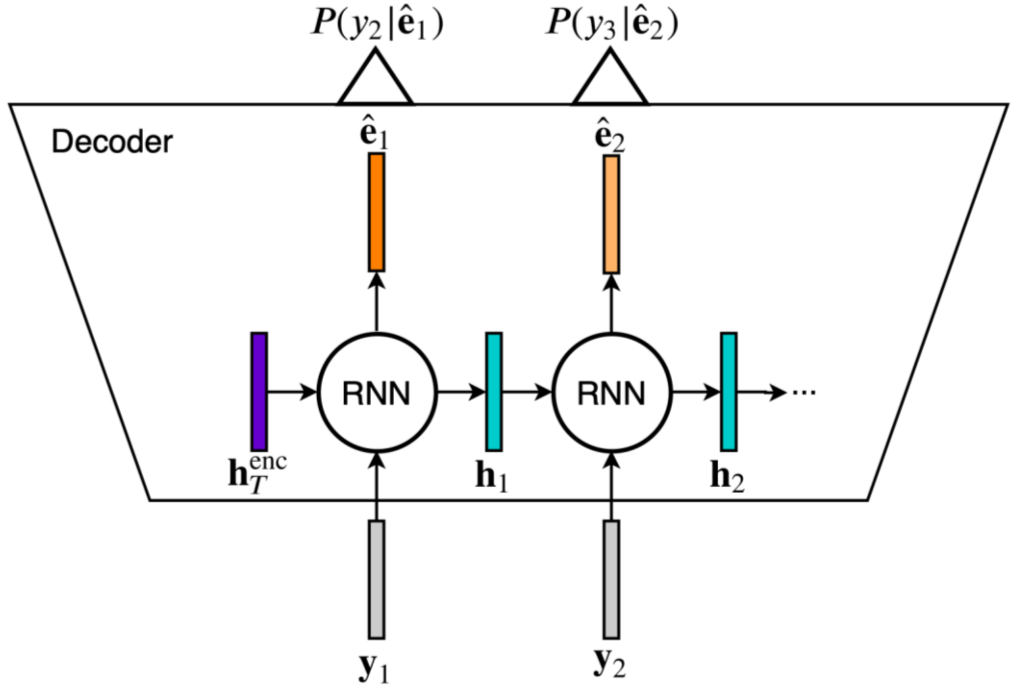


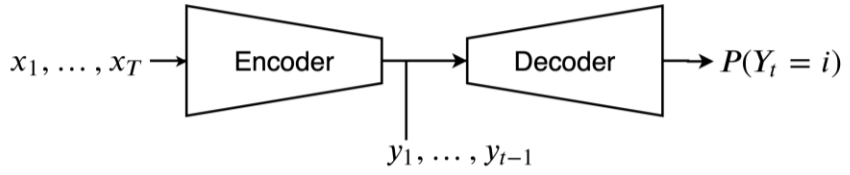
https://jeddy92.github.io/JEddy92.github.io/ts_seq2seq_intro/

I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to Sequence Learning with Neural Networks,” in *Conference on Advances in Neural Information Processing Systems (NeurIPS)*, Montréal, Canada, 2014, pp. 3104–3112. https://proceedings.neurips.cc/paper_files/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2-Abstract.html

Review: RNN Encoder-Decoder Architectures

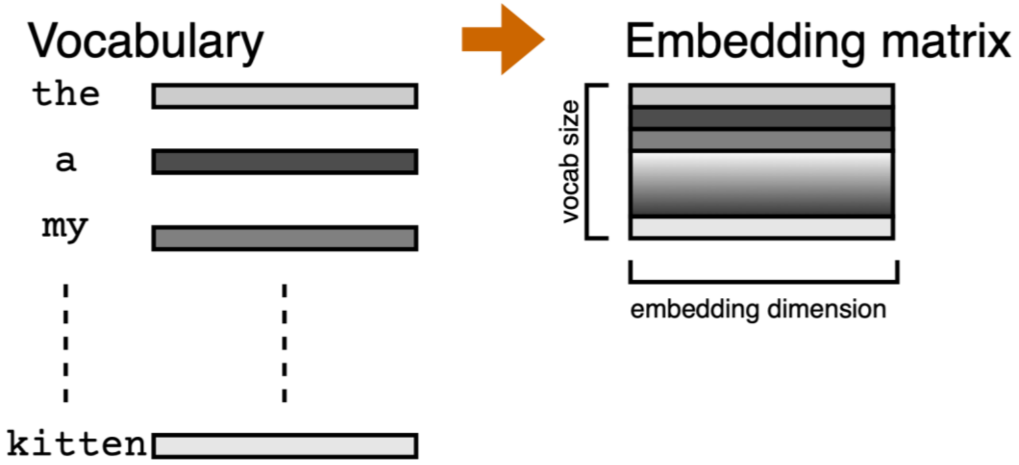


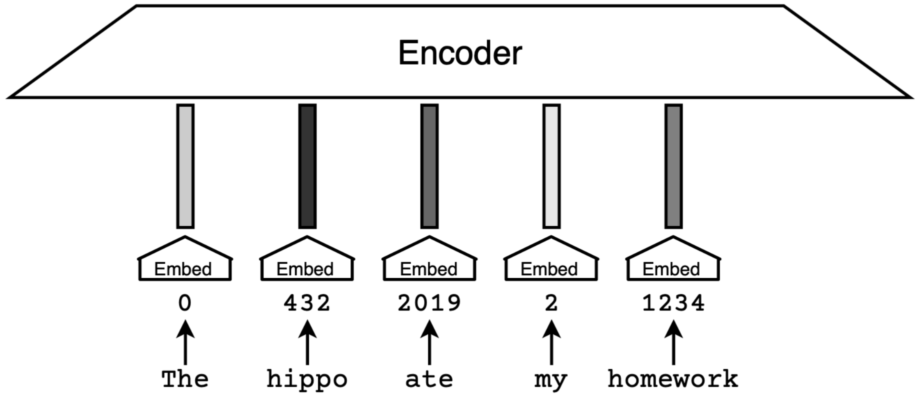


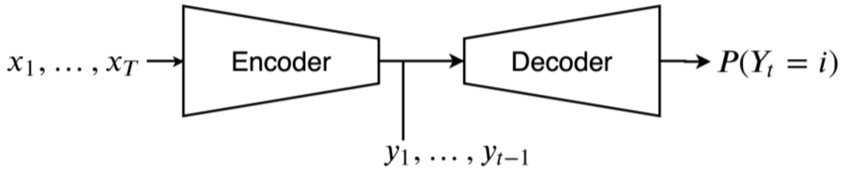


Review: Inputs to the Encoder

The encoder takes as input the embeddings corresponding to each token in the sequence.

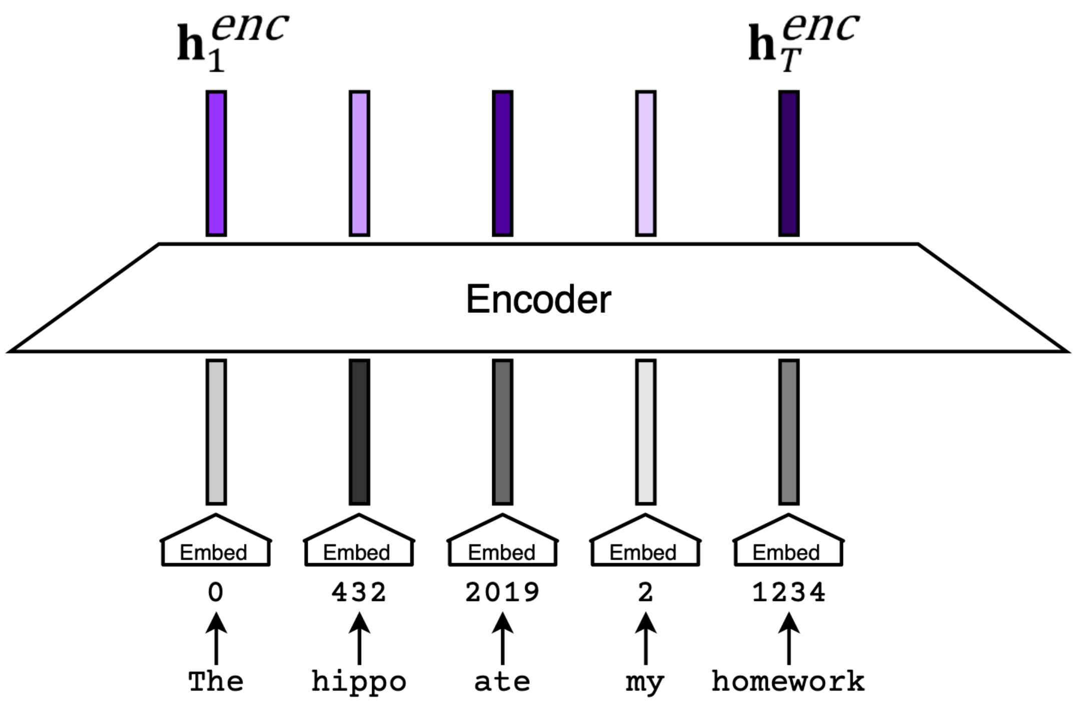


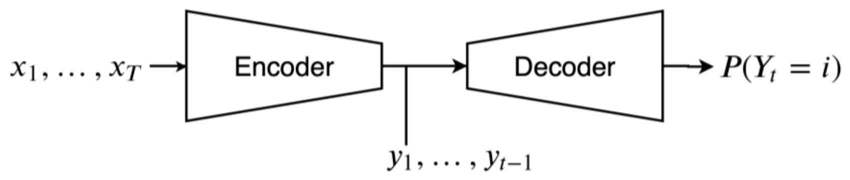




Review: Outputs from the Encoder

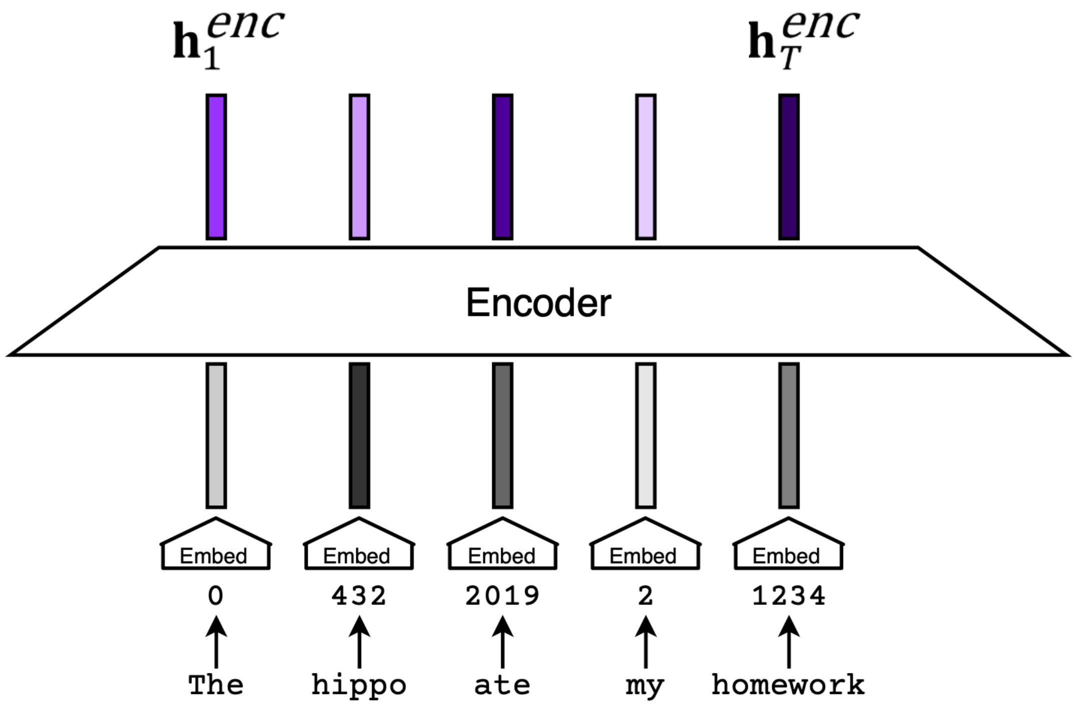
The encoder outputs a sequence of vectors. These are called the hidden state of the encoder.

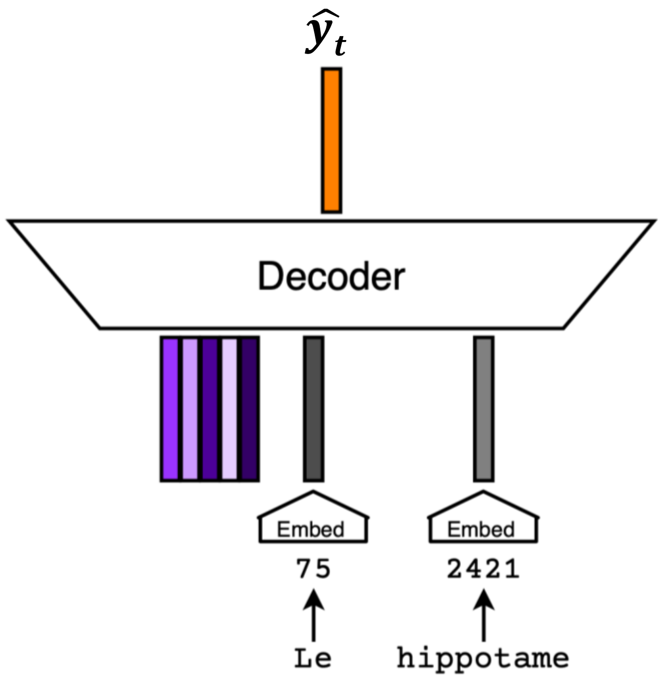


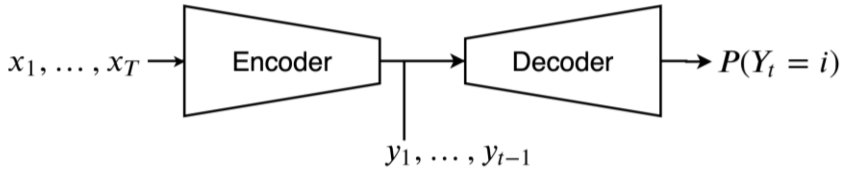


Review: Inputs to the Decoder

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.

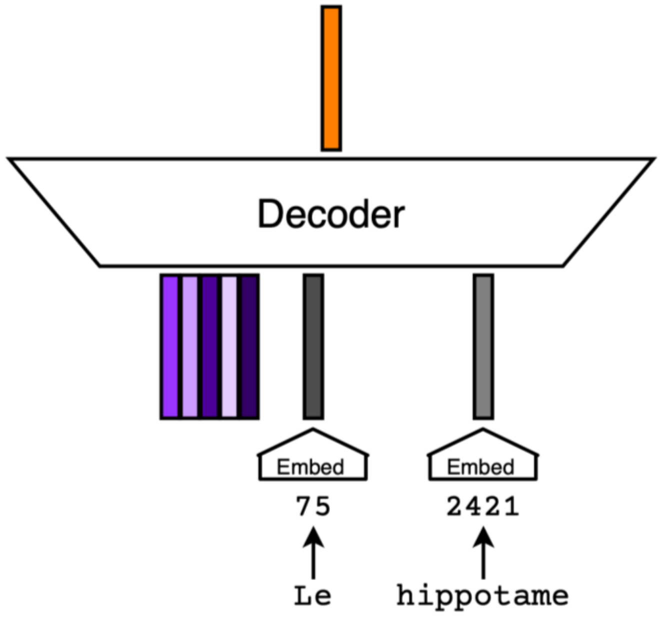






Review: Outputs from the Decoder

The decoder outputs an embedding $\widehat{yt}$. The goal is for this embedding to be as close as possible to the embedding of the true next token.



Review: Generating Text

Also sometimes called decoding 🙄

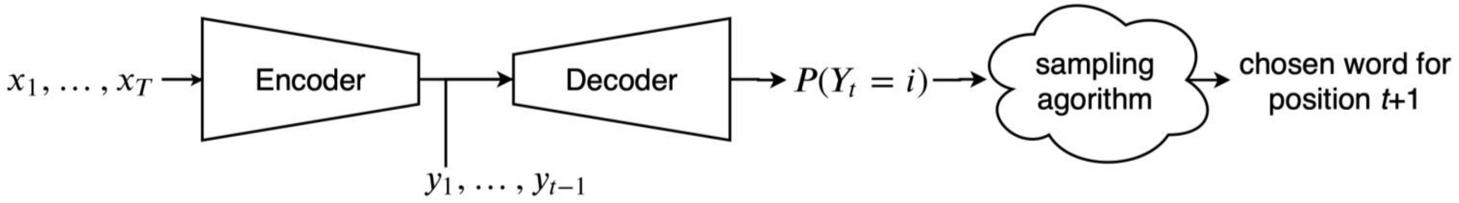
To generate text, we need an algorithm that selects tokens given the predicted probability distributions.

Examples:

Argmax

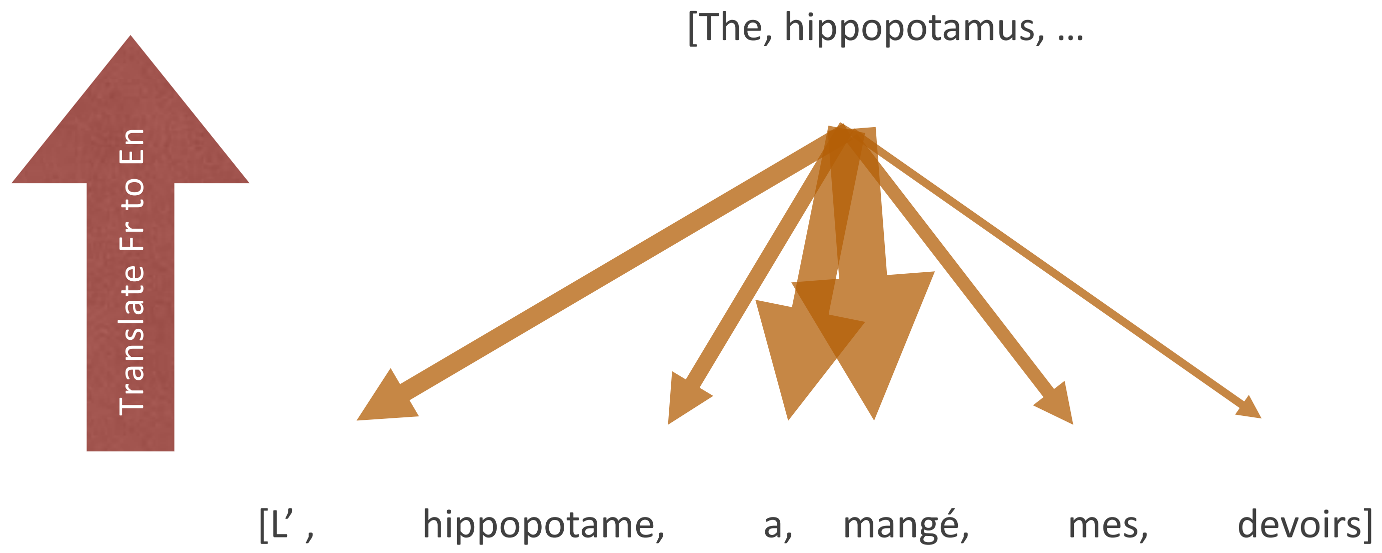
Beam search

Random sampling

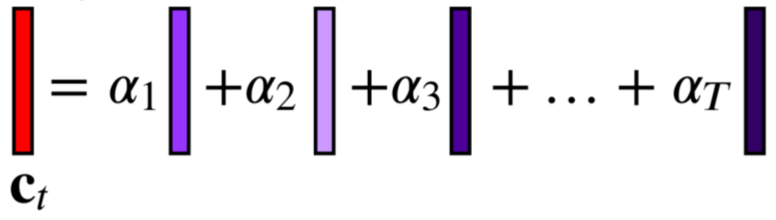


Review: Attention

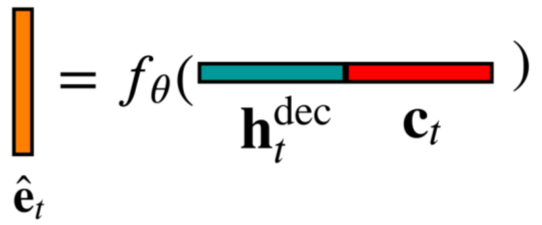
Better approach: an attention mechanism



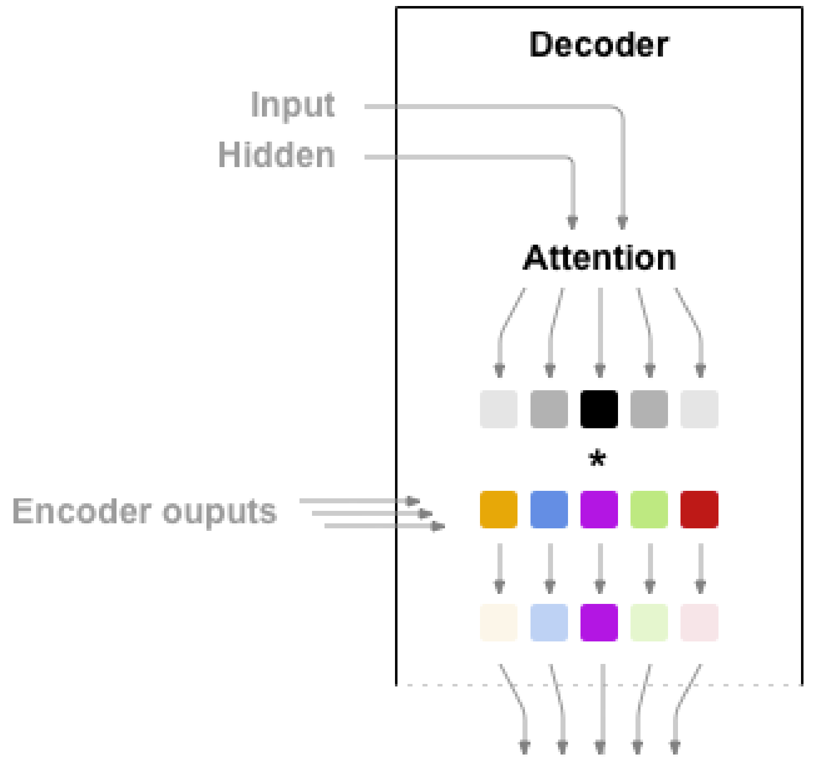
Compute a linear combination of the encoder hidden states.



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



Review: Attention Decoder



https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

Limitations of Recurrent architecture

Slow to train.

- Can’t be easily parallelized.
- The computation at position t is dependent on first doing the computation at position t-1.

Difficult to access information from many steps back.

- 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.

Review: Generating Text

Also sometimes called decoding 🙄

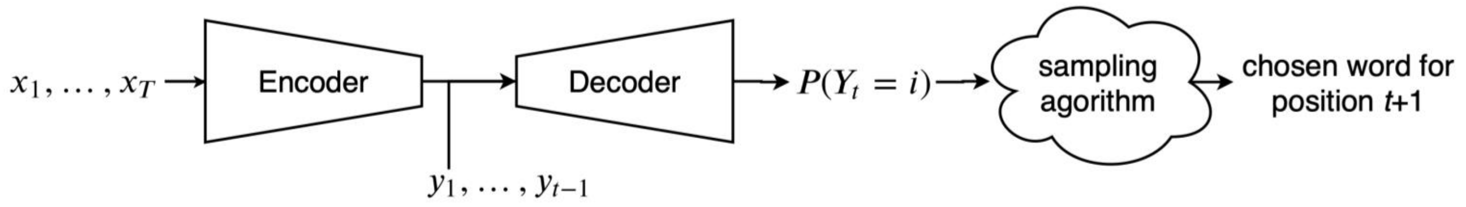
To generate text, we need an algorithm that selects tokens given the predicted probability distributions.

Examples:

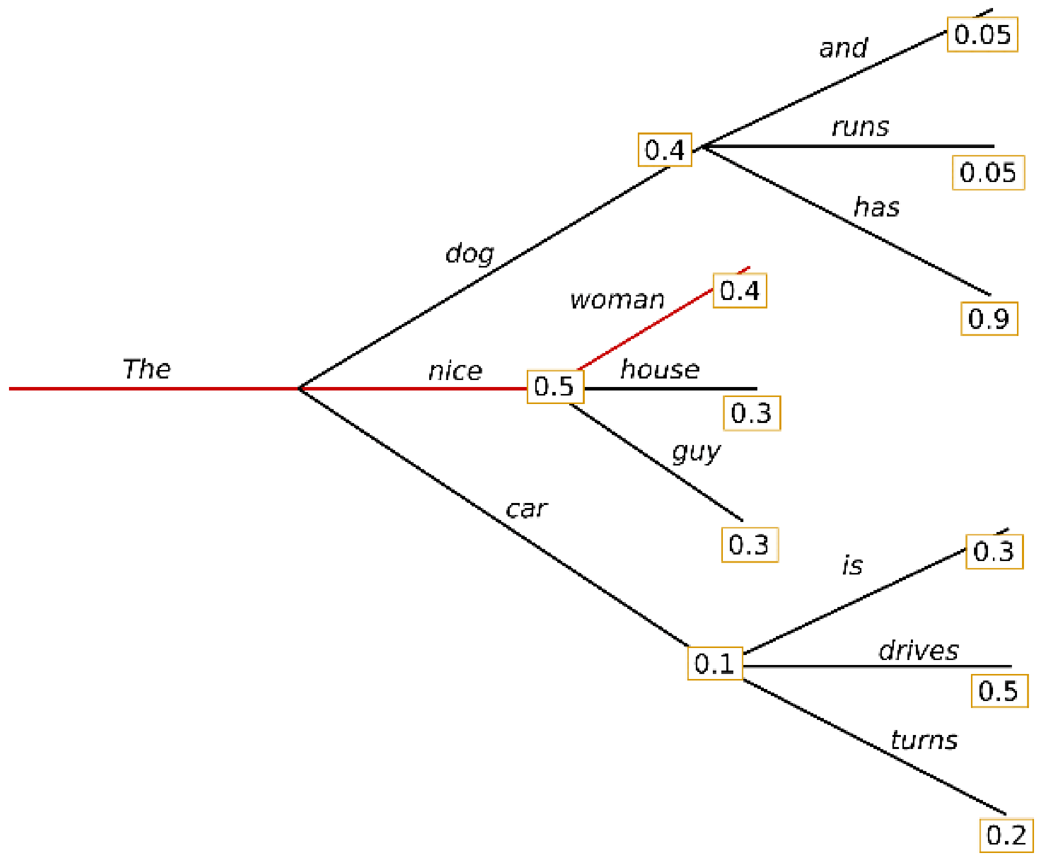
Argmax

Beam search

Random sampling



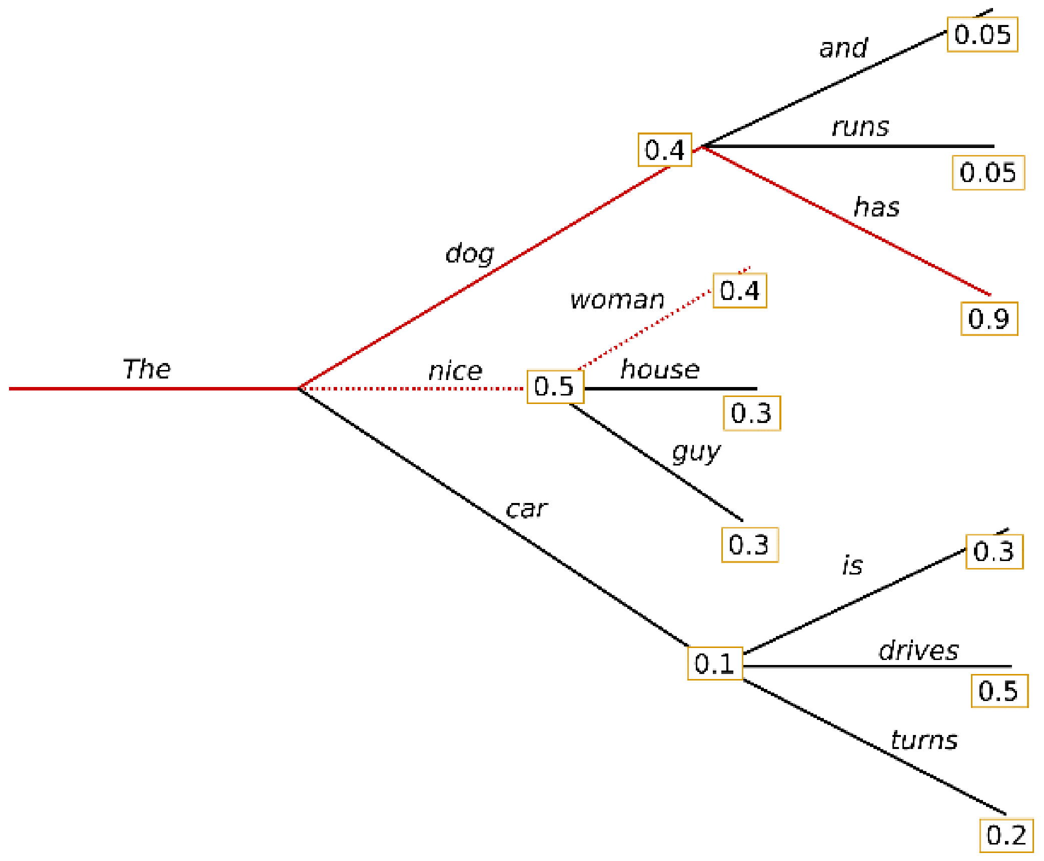
Greedy Search (Argmax)



https://huggingface.co/blog/how-to-generate

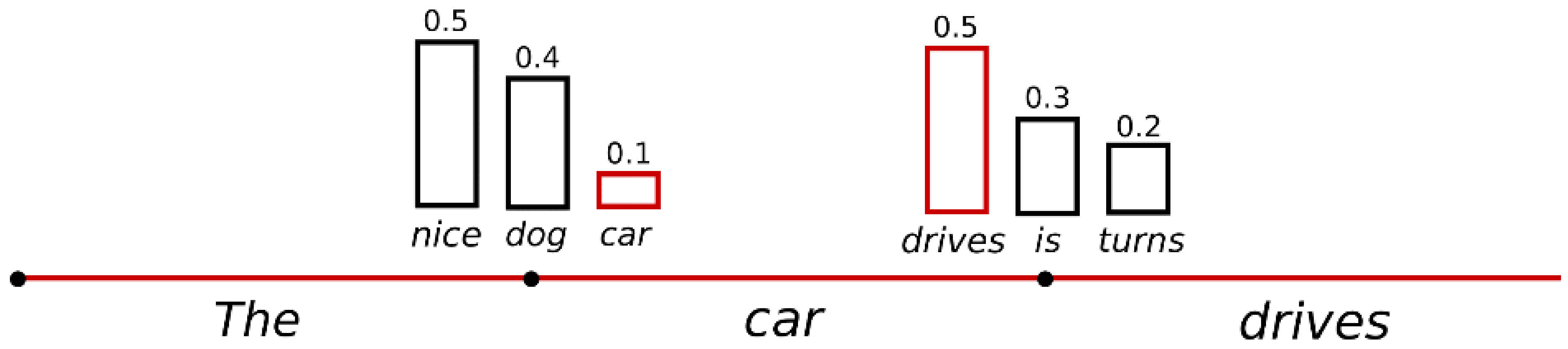
Beam Search

Number of beams = 2



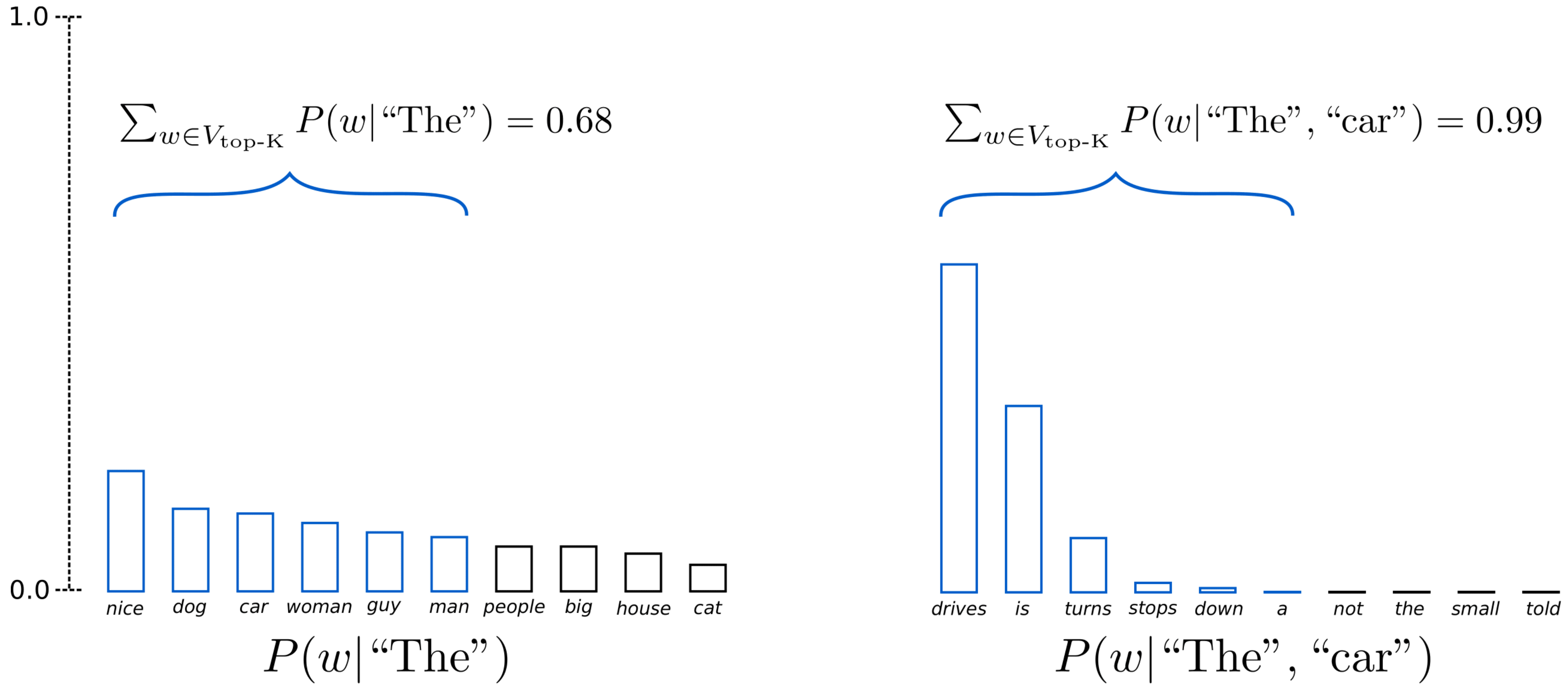
https://huggingface.co/blog/how-to-generate

Random Sampling



https://huggingface.co/blog/how-to-generate

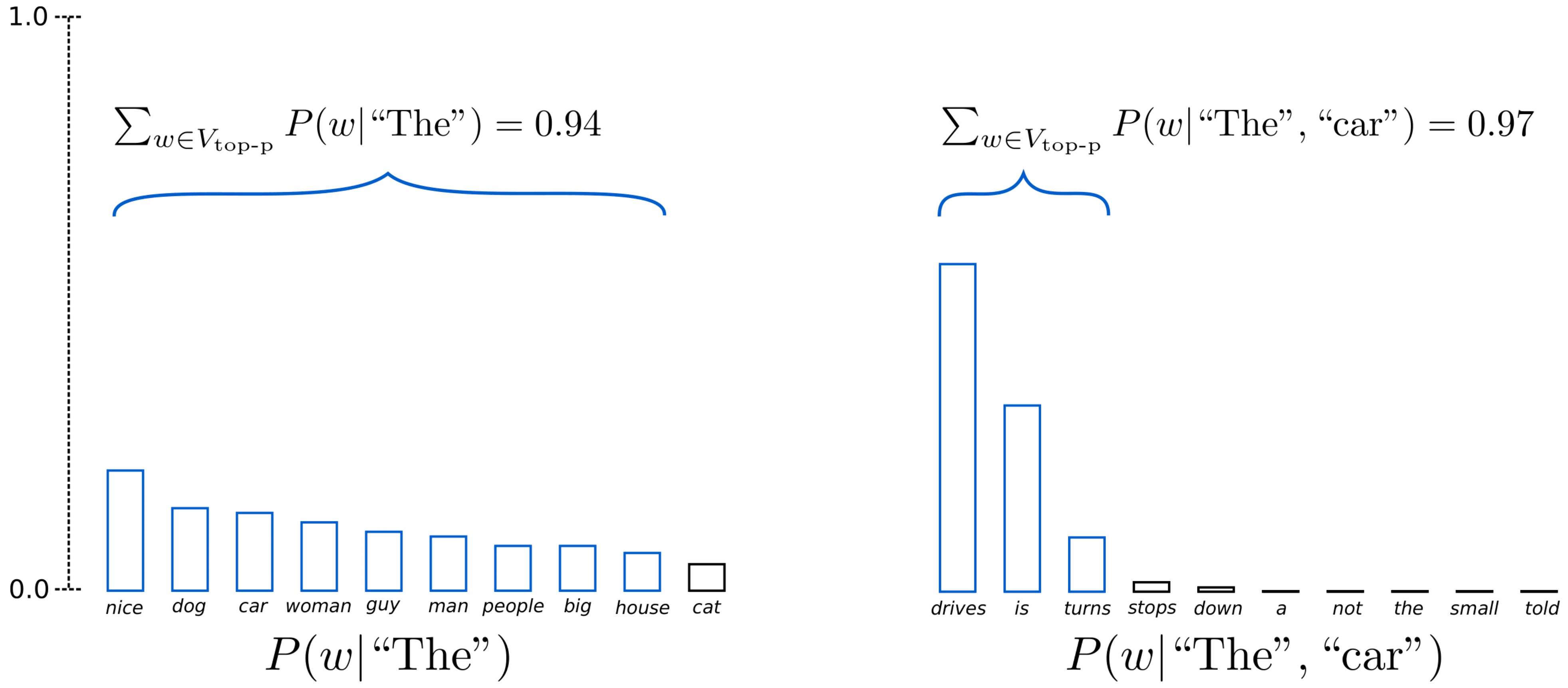
Top-K Sampling



A. Holtzman, J. Buys, M. Forbes, and Y. Choi, "The Curious Case of Neural Text Degeneration," in *International Conference on Learning Representations (ICLR)*, 2020, p. 16.
https://openreview.net/forum?id=rygGQyrFvH

https://huggingface.co/blog/how-to-generate

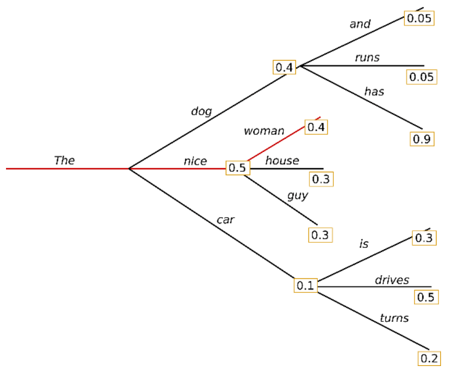
Top-P Sampling



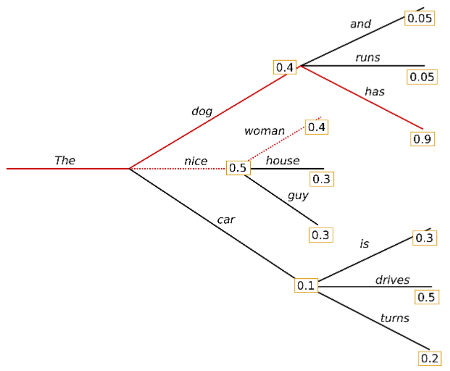
https://huggingface.co/blog/how-to-generate

Think-Pair-Share

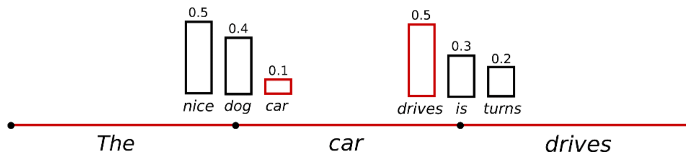
When might you want to use one sampling algorithm over the other?



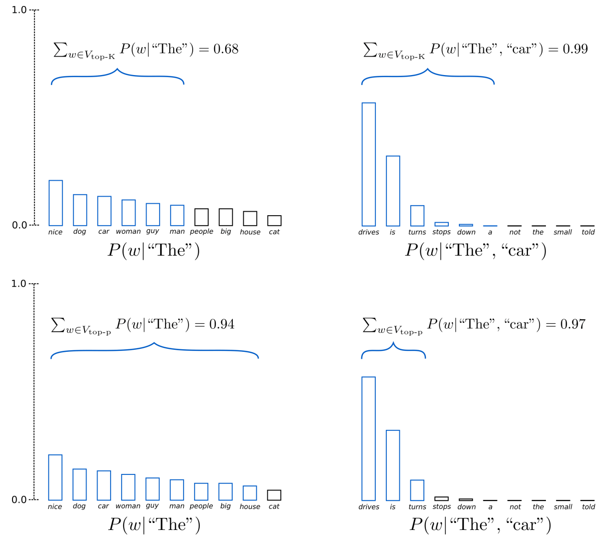
Greedy



Beam Search



Random Sampling

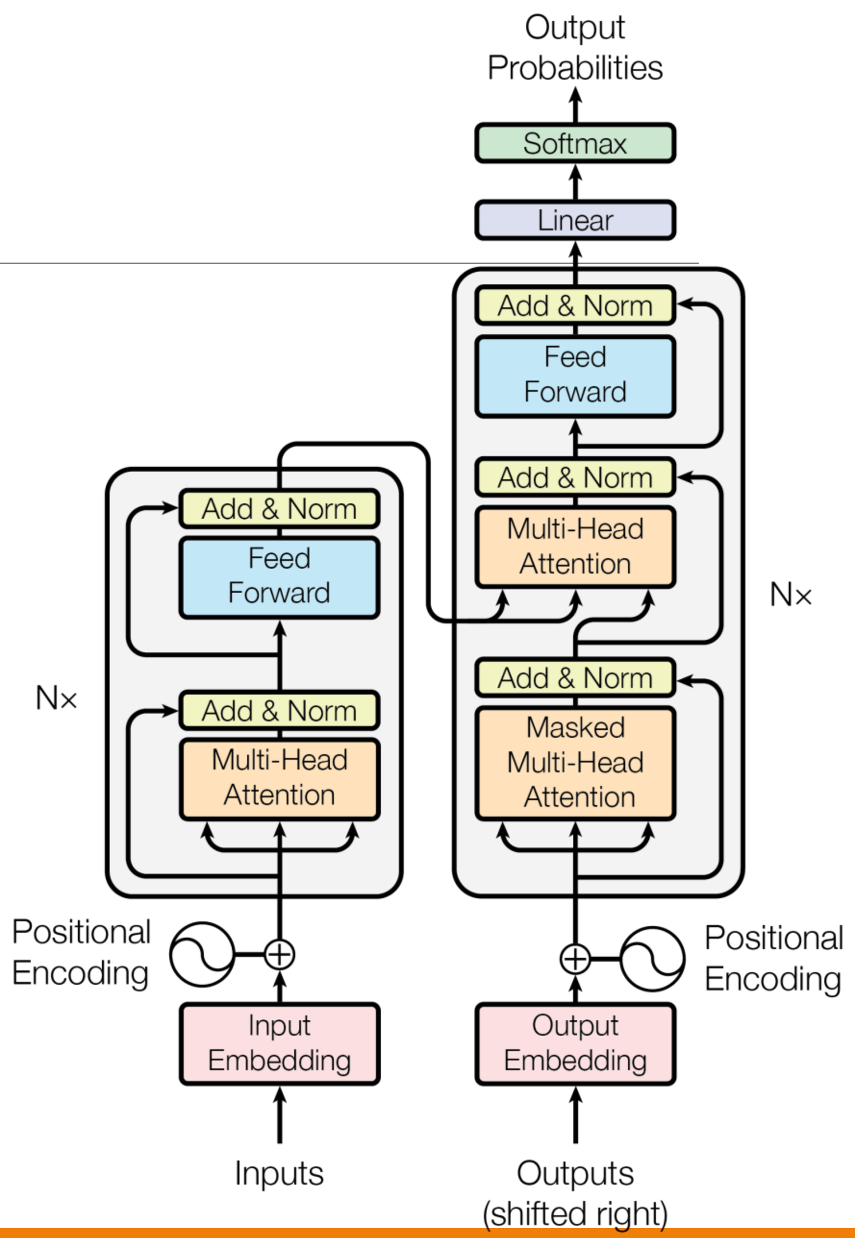


Top-K/P

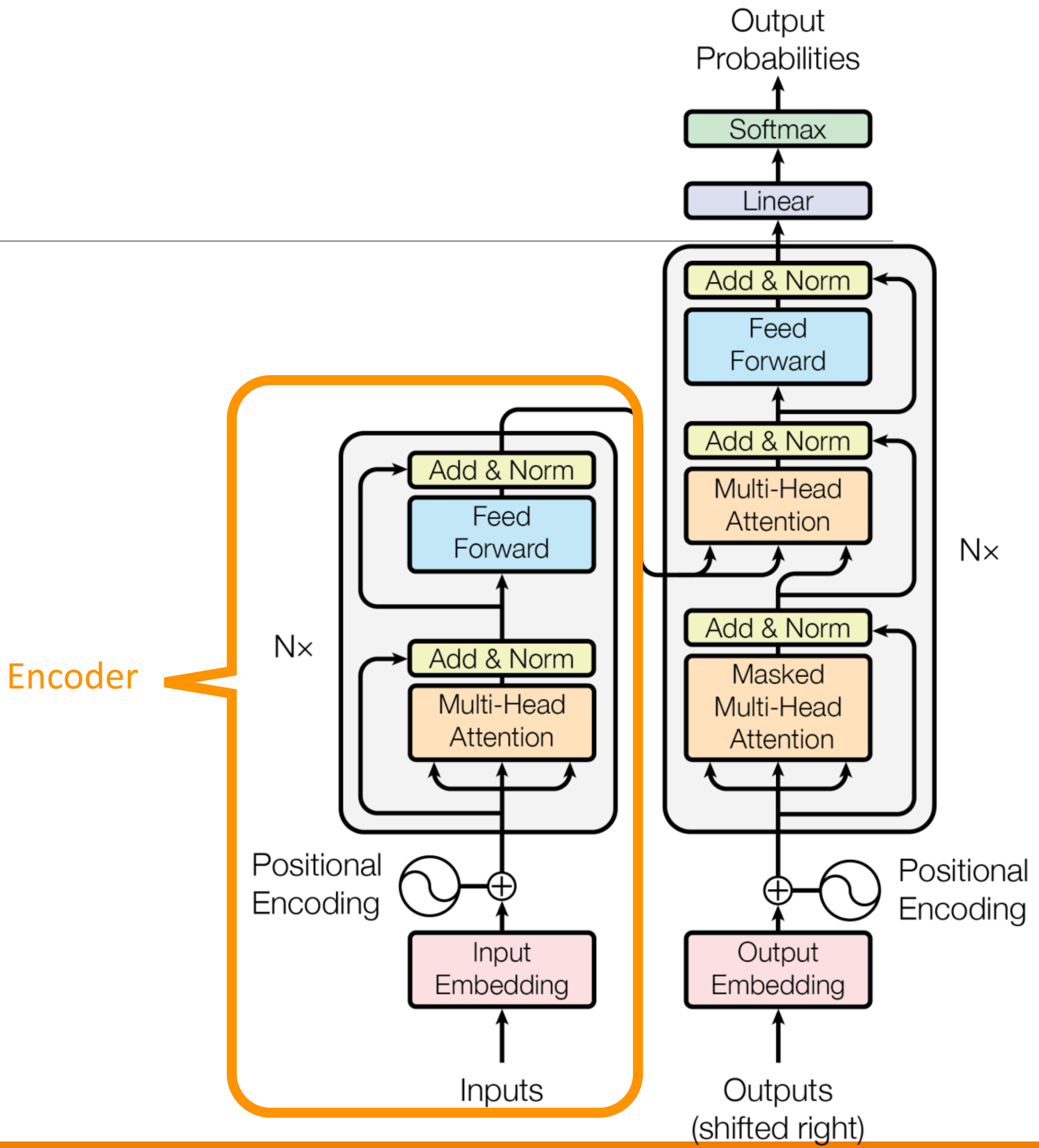
Transformers

The Transformer is a **non-recurrent** non-convolutional (feed-forward) neural network designed for language understanding

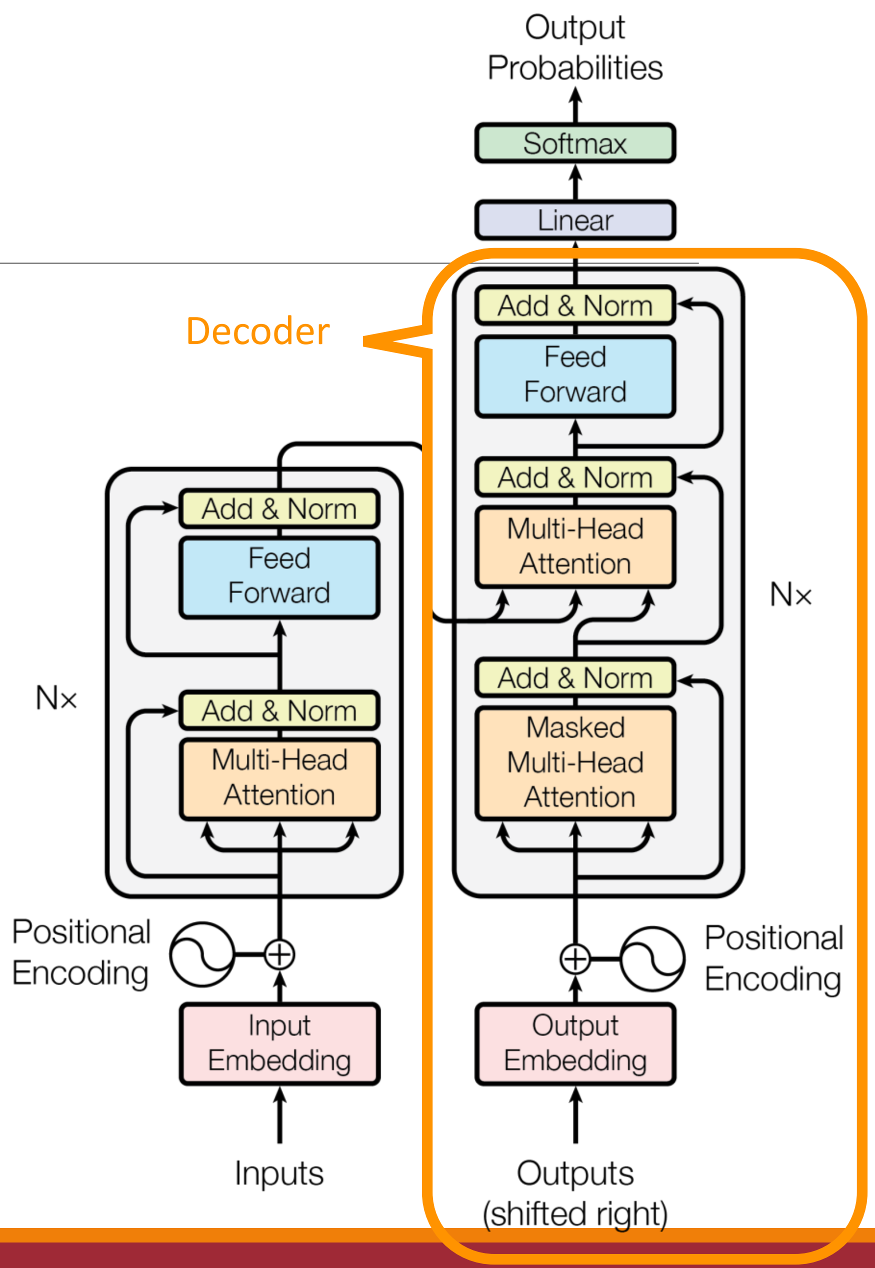
- introduces self-attention in addition to encoder-decoder attention



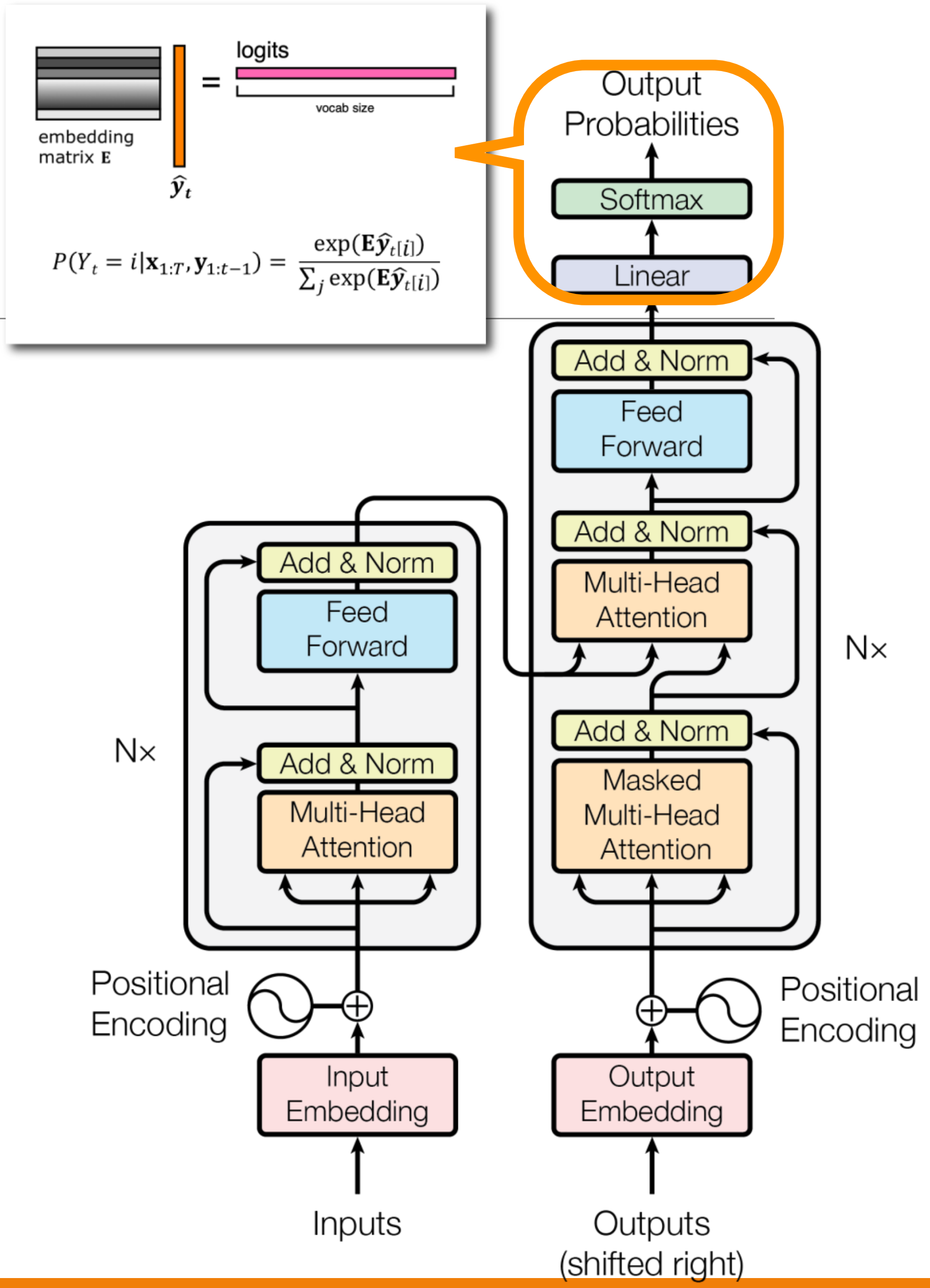
Transformers



Transformers



Transformers



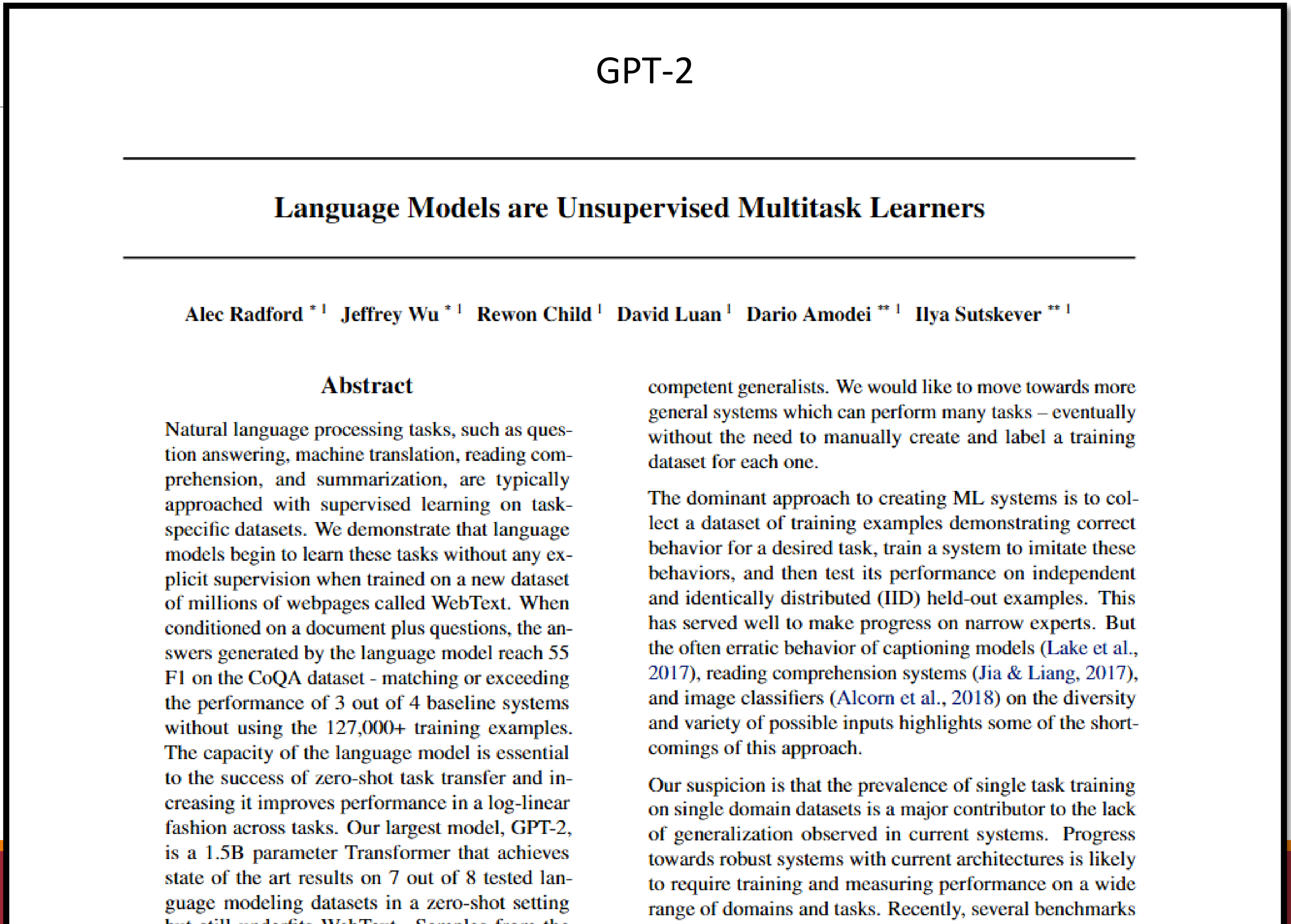
$$P(Y_t = i \mid \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1}) = \frac{\exp(\mathbf{E}\hat{\mathbf{y}}_{t[i]})}{\sum_j \exp(\mathbf{E}\hat{\mathbf{y}}_{t[i]})}$$

Fine-tuning

Start with pre-trained model

Freeze the model (don't touch it) except for the last layer

- Start with generalized "foundational" model
- Train on a new, small dataset for your specific task



GPT-2

Language Models are Unsupervised Multitask Learners

Alec Radford * [1] Jeffrey Wu * [1] Rewon Child [1] David Luan [1] Dario Amodei ** [1] Ilya Sutskever ** [1]

Abstract

Natural language processing tasks, such as question answering, machine translation, reading comprehension, and summarization, are typically approached with supervised learning on task-specific datasets. We demonstrate that language models begin to learn these tasks without any explicit supervision when trained on a new dataset of millions of webpages called WebText. When conditioned on a document plus questions, the answers generated by the language model reach 55 F1 on the CoQA dataset - matching or exceeding the performance of 3 out of 4 baseline systems without using the 127,000+ training examples. The capacity of the language model is essential to the success of zero-shot task transfer and increasing it improves performance in a log-linear fashion across tasks. Our largest model, GPT-2, is a 1.5B parameter Transformer that achieves state of the art results on 7 out of 8 tested language modeling datasets in a zero-shot setting

competent generalists. We would like to move towards more general systems which can perform many tasks – eventually without the need to manually create and label a training dataset for each one.

The dominant approach to creating ML systems is to collect a dataset of training examples demonstrating correct behavior for a desired task, train a system to imitate these behaviors, and then test its performance on independent and identically distributed (IID) held-out examples. This has served well to make progress on narrow experts. But the often erratic behavior of captioning models (Lake et al., 2017), reading comprehension systems (Jia & Liang, 2017), and image classifiers (Alcorn et al., 2018) on the diversity and variety of possible inputs highlights some of the shortcomings of this approach.

Our suspicion is that the prevalence of single task training on single domain datasets is a major contributor to the lack of generalization observed in current systems. Progress towards robust systems with current architectures is likely to require training and measuring performance on a wide range of domains and tasks. Recently, several benchmarks

Pre-trained models

Most LLMs people use today are pre-trained foundational models

- Has a grasp on human language but not trained on a specific task

Trained on “the Internet” → Impossible to know all of what it’s train on

What types of things can go wrong with finetuning?

Underfitting – finetuning data is too different from what the foundational model was train on

Overfitting – overwrites what the model learned originally