Interactive Fiction and Text Generation

Lara J. Martin (she/they)

https://laramartin.net/interactive-fiction-class

Slides modified from Dr. Daphne Ippolito

Learning Objectives

Become acquainted with language models

Compare sequence-to-sequence RNNs to transformers

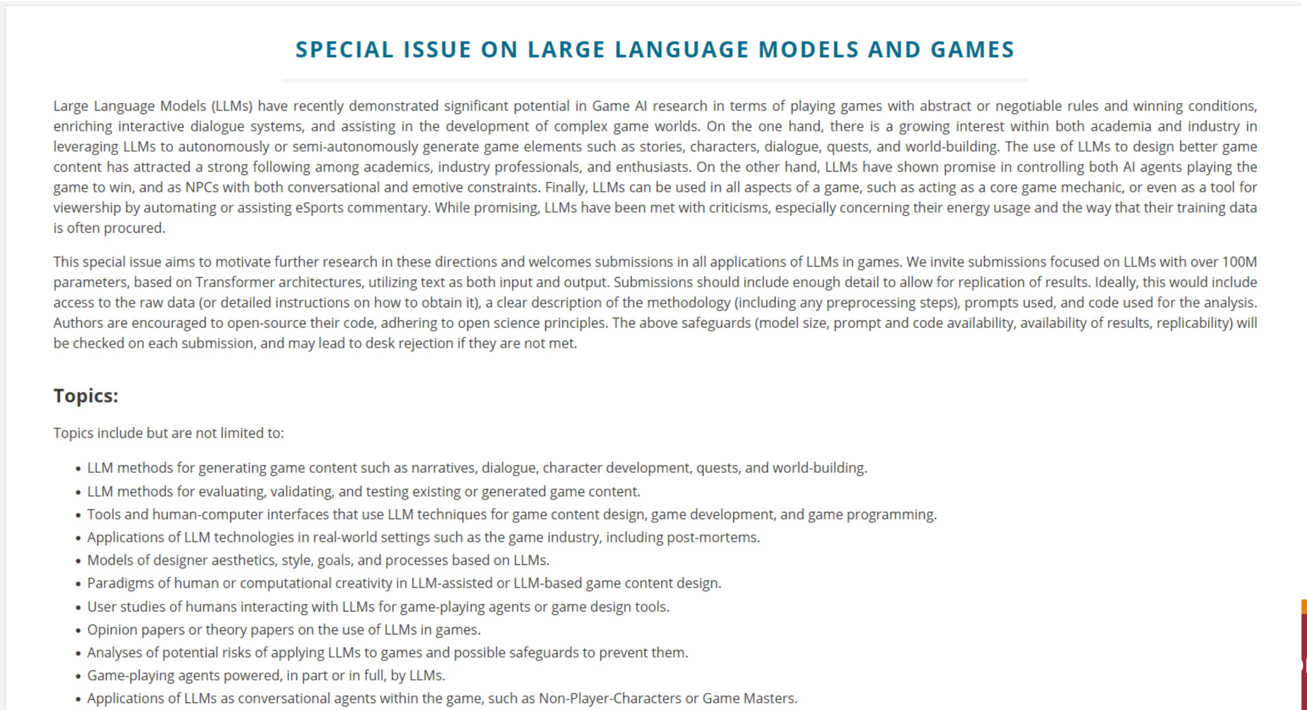
Consider the strengths and weaknesses of LLMs/transformers

Class Announcements

I will post a Google Form for Reading Presentations (grad only) sign up

I plan to create an accelerated deadline for the class project

- IEEE ToG due December 1



SPECIAL ISSUE ON LARGE LANGUAGE MODELS AND GAMES

Large Language Models (LLMs) have recently demonstrated significant potential in Game AI research in terms of playing games with abstract or negotiable rules and winning conditions, enriching interactive dialogue systems, and assisting in the development of complex game worlds. On the one hand, there is a growing interest within both academia and industry in leveraging LLMs to autonomously or semi-autonomously generate game elements such as stories, characters, dialogue, quests, and world-building. The use of LLMs to design better game content has attracted a strong following among academics, industry professionals, and enthusiasts. On the other hand, LLMs have shown promise in controlling both AI agents playing the game to win, and as NPCs with both conversational and emotive constraints. Finally, LLMs can be used in all aspects of a game, such as acting as a core game mechanic, or even as a tool for viewership by automating or assisting eSports commentary. While promising, LLMs have been met with criticisms, especially concerning their energy usage and the way that their training data is often procured.

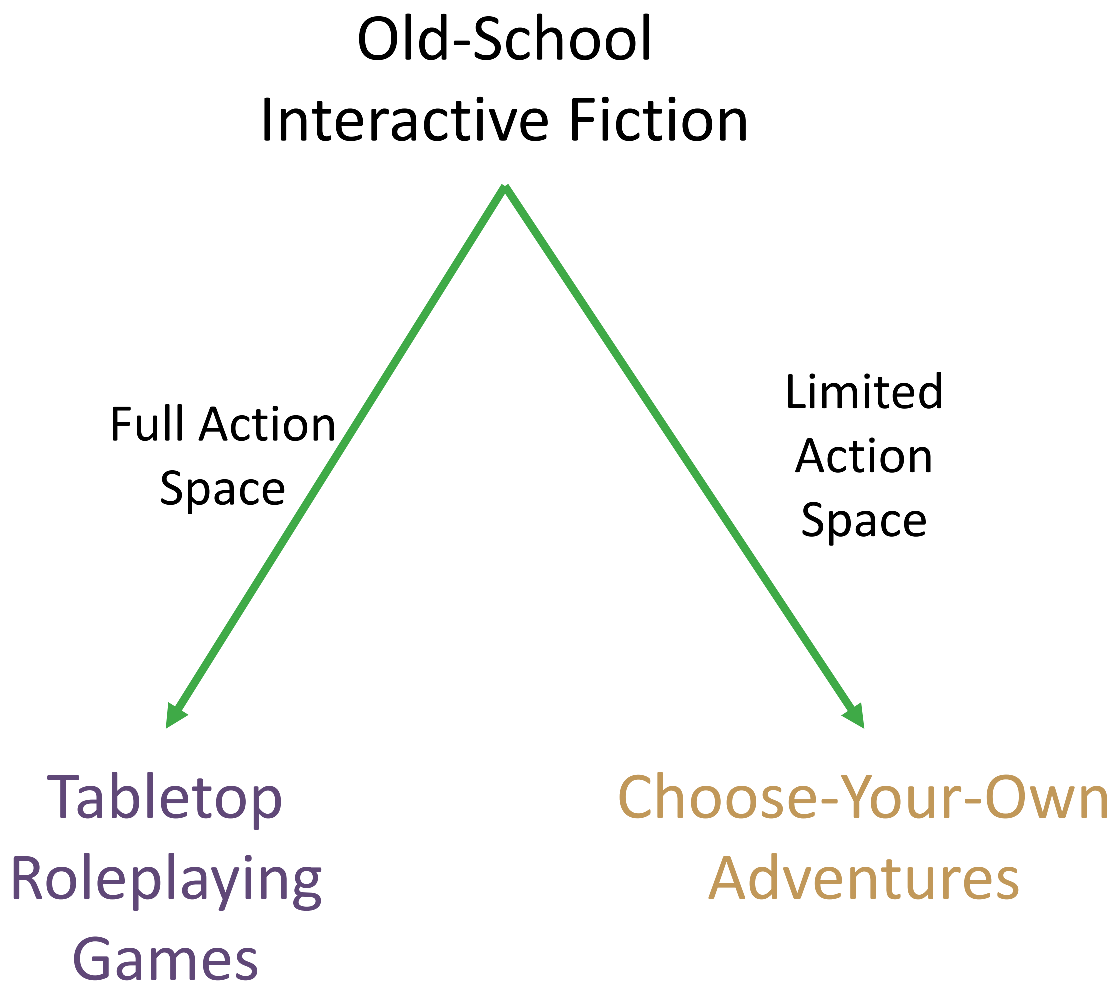
This special issue aims to motivate further research in these directions and welcomes submissions in all applications of LLMs in games. We invite submissions focused on LLMs with over 100M parameters, based on Transformer architectures, utilizing text as both input and output. Submissions should include enough detail to allow for replication of results. Ideally, this would include access to the raw data (or detailed instructions on how to obtain it), a clear description of the methodology (including any preprocessing steps), prompts used, and code used for the analysis. Authors are encouraged to open-source their code, adhering to open science principles. The above safeguards (model size, prompt and code availability, availability of results, replicability) will be checked on each submission, and may lead to desk rejection if they are not met.

Topics:

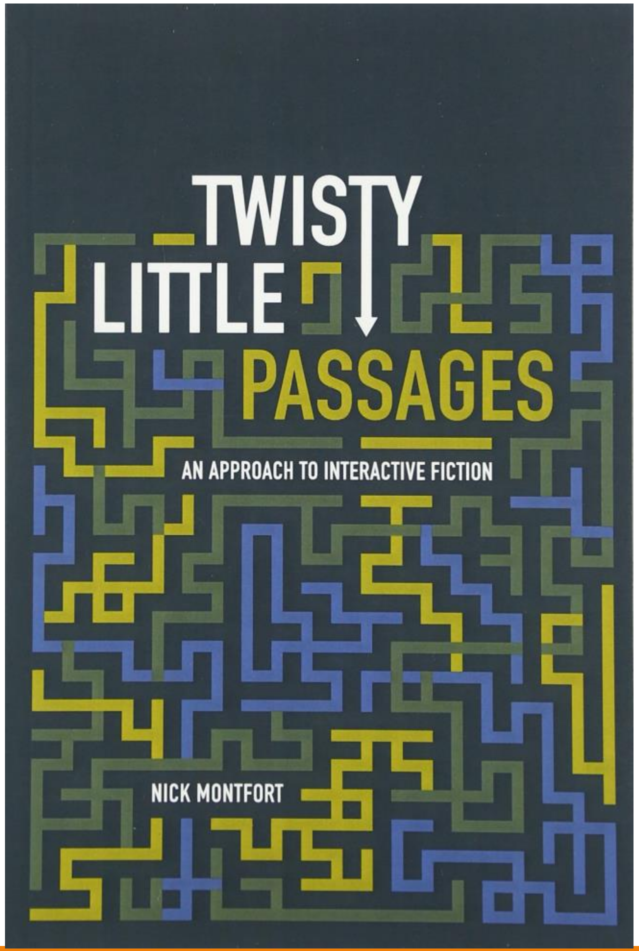
Topics include but are not limited to:

- LLM methods for generating game content such as narratives, dialogue, character development, quests, and world-building.
- LLM methods for evaluating, validating, and testing existing or generated game content.
- Tools and human-computer interfaces that use LLM techniques for game content design, game development, and game programming.
- Applications of LLM technologies in real-world settings such as the game industry, including post-mortems.
- Models of designer aesthetics, style, goals, and processes based on LLMs.
- Paradigms of human or computational creativity in LLM-assisted or LLM-based game content design.
- User studies of humans interacting with LLMs for game-playing agents or game design tools.
- Opinion papers or theory papers on the use of LLMs in games.
- Analyses of potential risks of applying LLMs to games and possible safeguards to prevent them.
- Game-playing agents powered, in part or in full, by LLMs.
- Applications of LLMs as conversational agents within the game, such as Non-Player-Characters or Game Masters.

https://transactions.games/special-issue/special-issue-on-large-language-models-and-games


Old-School Interactive Fiction
Full Action Space
Limited Action Space
Tabletop Roleplaying Games
Choose-Your-Own Adventures

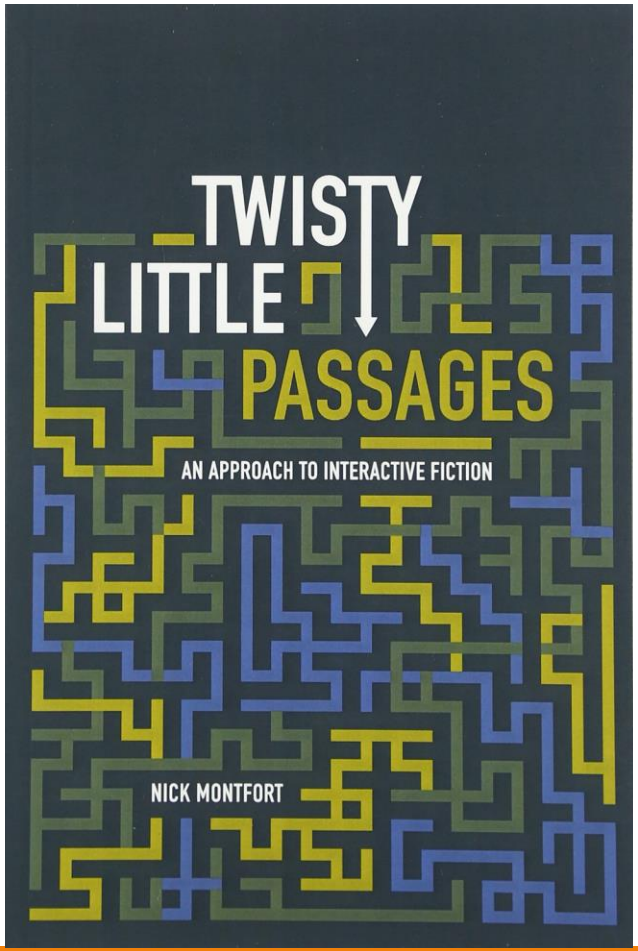
Review: Components of Interactive Fiction Games



The **parser**, which is the component that analyzes natural language input in an interactive fiction work.

The **world model**, which is setting of an interactive fiction work.

Review: Components of Interactive Fiction Games



The **parser,** which is the component that analyzes natural language input in an interactive fiction work.

The **world model,** which is setting of an interactive fiction work.

You just started up a game and now you're staring at *text* and a *blinking cursor* and you *don't know what to do*!

(> |)

Don't panic kids— Crazy Uncle Zarf is here to help you get started...

These commands are very common:

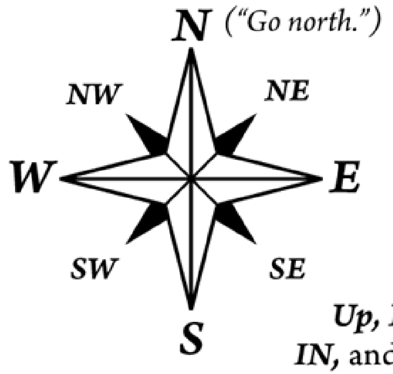
EXAMINE *it*	**PUSH** *it*
TAKE *it*	**PULL** *it*
DROP *it*	**TURN** *it*
OPEN *it*	**FEEL** *it*
PUT *it* **IN** *something*	
PUT *it* **ON** *something*	

When in doubt, examine more.

Does the game intro suggest **ABOUT, INFO, HELP**? *Try them first!*

You are standing in an open field west of a white house, with a boarded front door. There is a small mailbox* here.

*Try opening!



Also: *Up, Down, IN,* and *OUT*

You can try all sorts of commands on the things you see.

Try the commands that make sense!

Doors are for opening; buttons are for pushing; pie is for eating. *(Mmm, pie.)*

◇◇◇◇

If you meet a person, these should work:

TALK TO *name*
ASK *name* **ABOUT** *something*
TELL *name* **ABOUT** *something*
GIVE *something* **TO** *name*
SHOW *something* **TO** *name*

Each game has slightly different commands, but they all look ***pretty much like these.***

You could also try:

EAT *it*	**CLIMB** *it*
DRINK *it*	**WAVE** *it*
FILL *it*	**WEAR** *it*
SMELL *it*	**TAKE** *it* **OFF**
LISTEN TO *it*	**TURN** *it* **ON**
BREAK *it*	**DIG IN** *it*
BURN *it*	**ENTER** *it*
LOOK UNDER *it*	**SEARCH** *it*
UNLOCK *it* **WITH** *something*	

Or even:

LISTEN	**JUMP**
SLEEP	**PRAY**
WAKE UP	**CURSE**
UNDO†	**SING**

†*Take back one move — handy!*

"What if I only want to type one or two letters?"

◇◇◇◇

N/E/S/W/NE/SE/NW/SW: GO *in the indicated compass direction.*
L: LOOK *around to see what is nearby.*
X: EXAMINE *a thing in more detail.*
I: *take* **INVENTORY** *of what you possess.*
Z: WAIT *a turn without doing anything.*
G: *do the same thing* **AGAIN**

◇◇◇◇

A service of the People's Republic of Interactive Fiction: ***http://pr-if.org***

Review: Why were parsers so bad?

Limited computational resources. Computers had ≤128 KB of memory

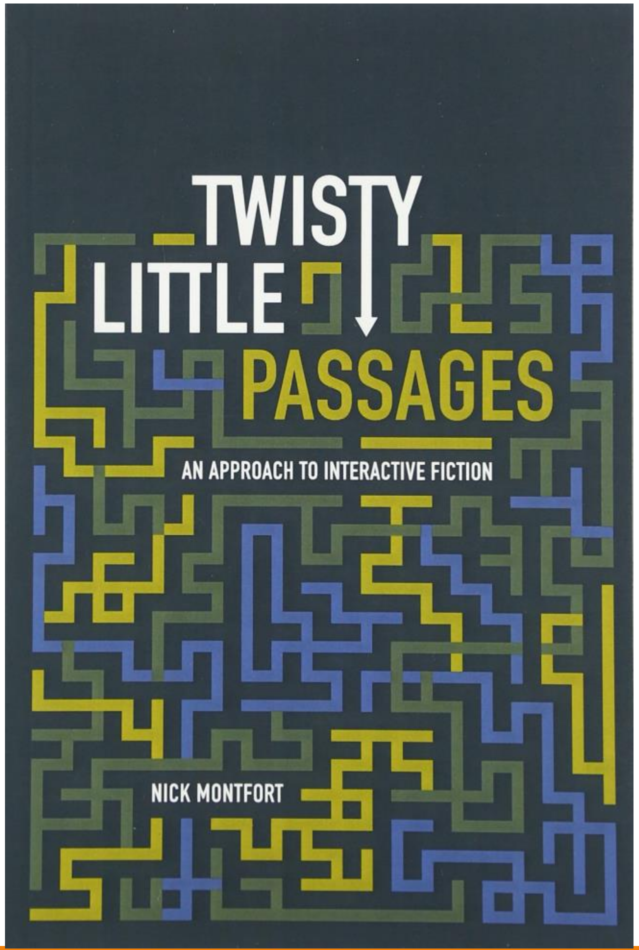
Language is difficult. There are many things that make human languages genuinely challenging for a computer to process.

Keyword-based commands. Only exact matches worked properly. No synonyms, no paraphrases.

Everything was manual. Game developers had to anticipate all possible commands, and manually code the responses.

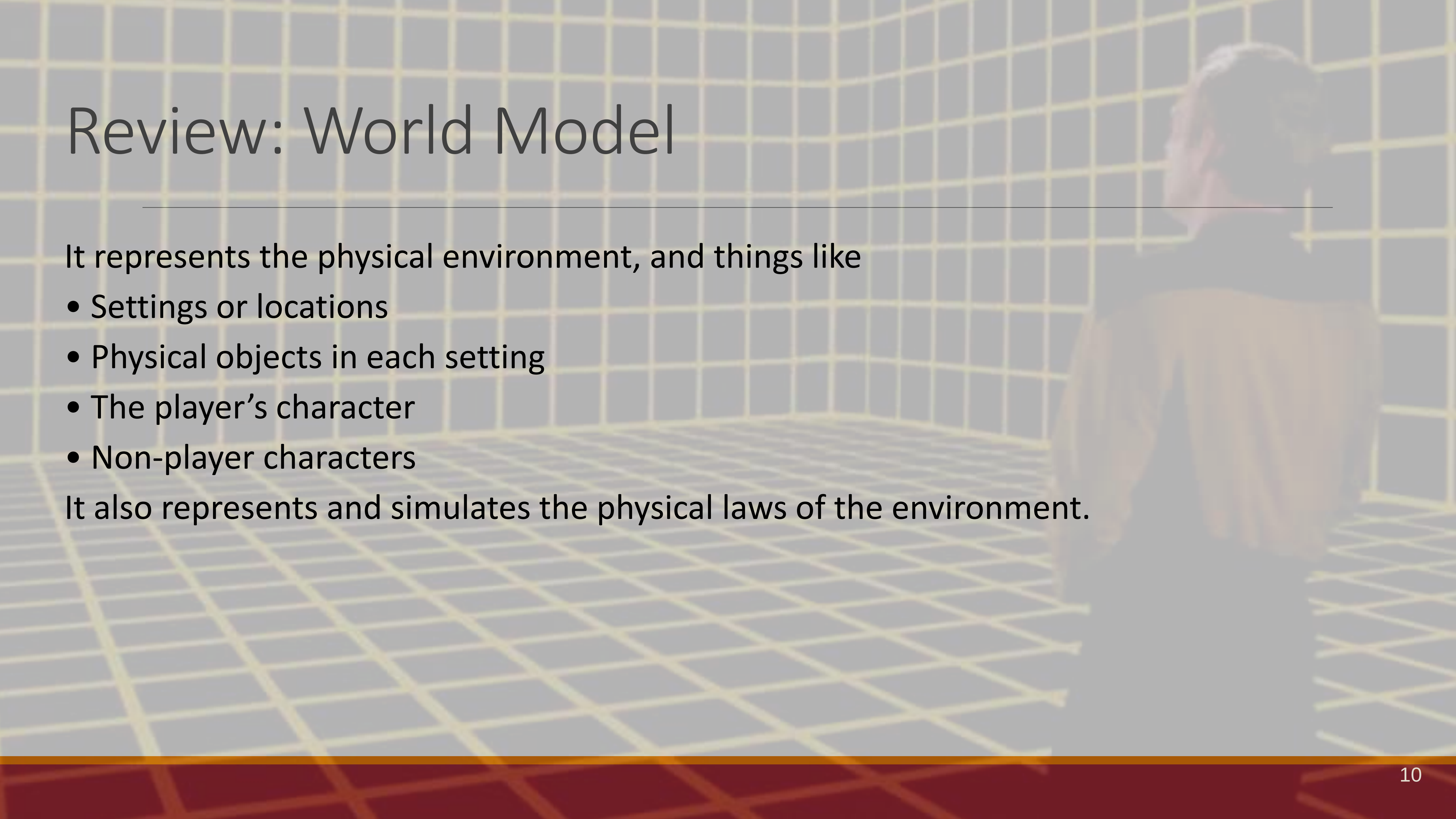
No machine learning. This was prior to the advent of machine learning based natural language processing

Review: Components of Interactive Fiction Games



The **parser,** which is the component that analyzes natural language input in an interactive fiction work.

The **world model,** which is setting of an interactive fiction work.



Review: World Model

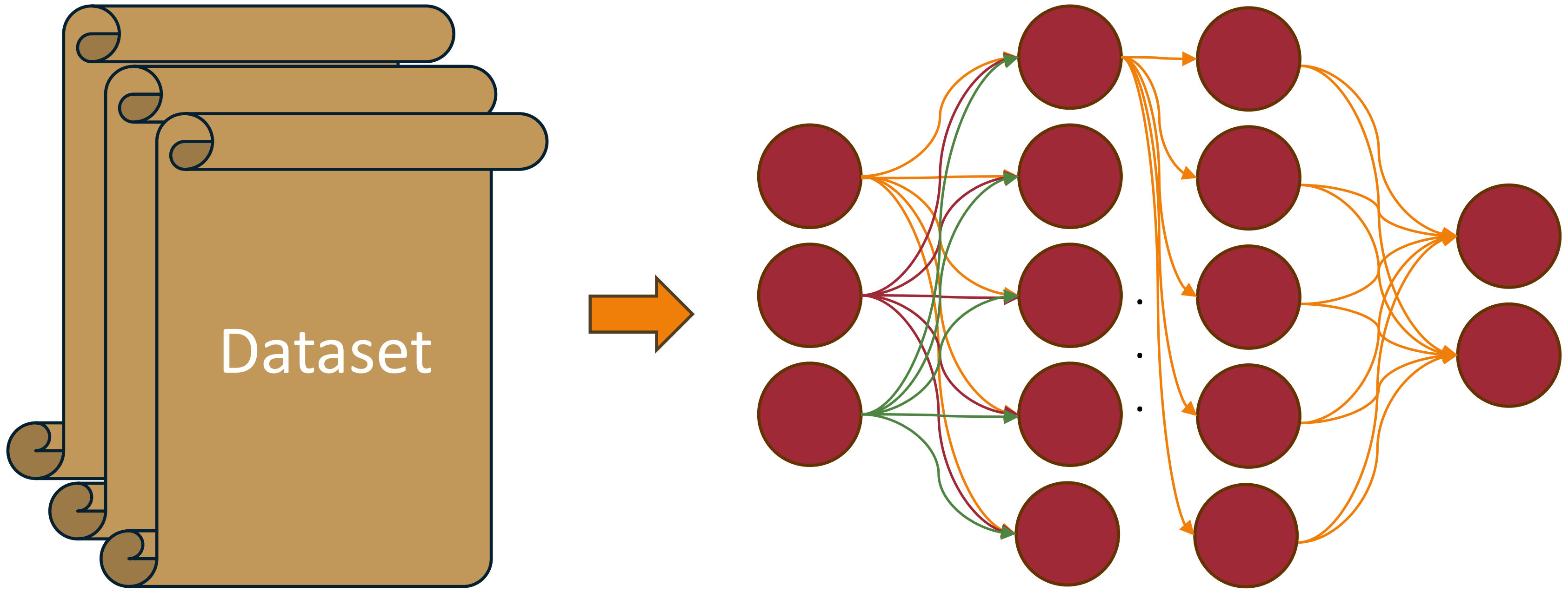
It represents the physical environment, and things like

- Settings or locations
- Physical objects in each setting
- The player’s character
- Non-player characters

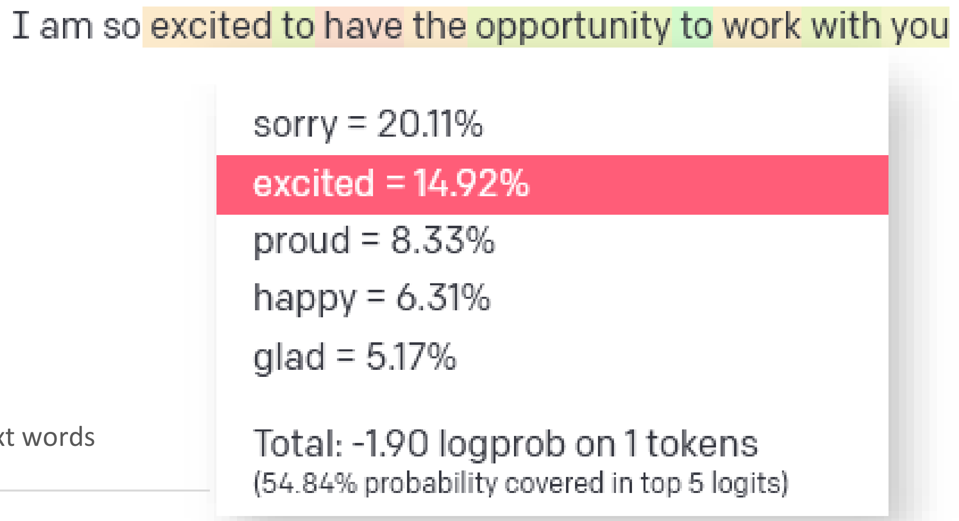
It also represents and simulates the physical laws of the environment.

Language Models

Making a neural language model



Using a neural language model



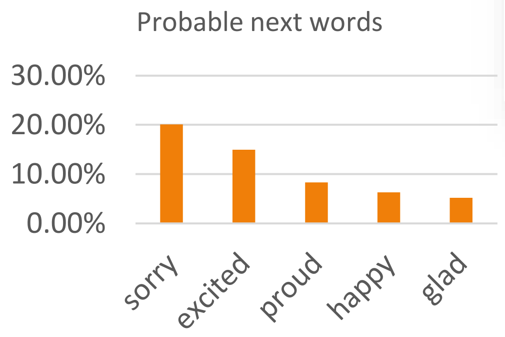
I am so excited to have the opportunity to work with you

sorry = 20.11%
excited = 14.92%
proud = 8.33%
happy = 6.31%
glad = 5.17%

Total: -1.90 logprob on 1 tokens
(54.84% probability covered in top 5 logits)

I am so

excited



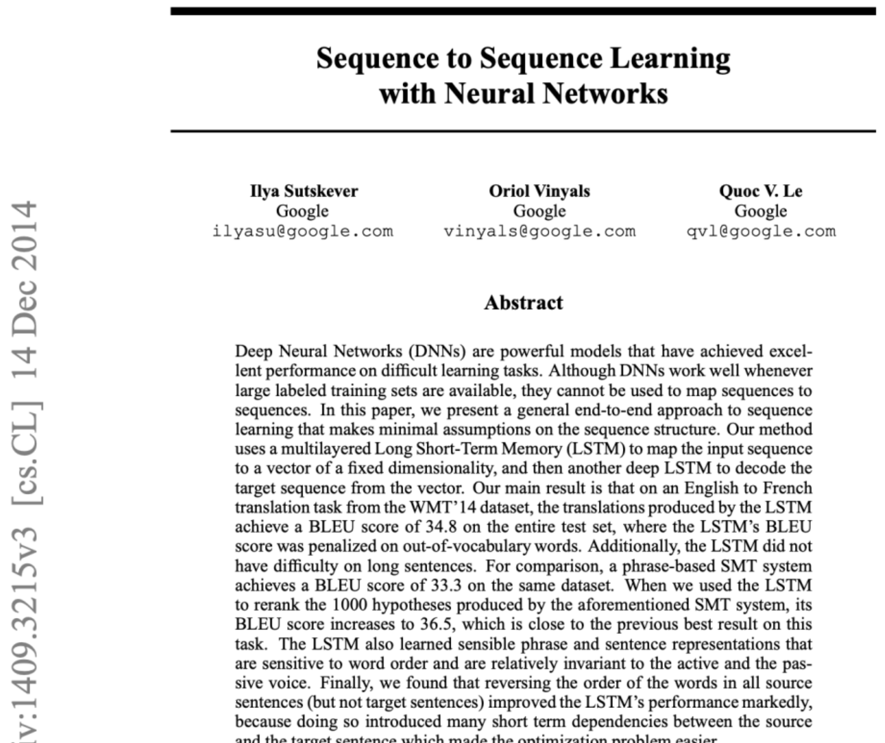
Example created with text-davinci-003 on openai.com in Feb. 2023

Quick Poll

1. Select which classes you have taken (can be either at UMBC or another institution).
   a) Introduction to AI/Principles of AI
   b) Natural Language Processing
   c) Machine Learning
   d) Any statistics course
2. Have you used an LLM (e.g., GPT, Llama, Gemini, Mistral) before?
   a) Yes, I do research/work with them
   b) Yes, I use them frequently (for fun/to help me)
   c) Yes, I’ve played around with them a few times
   d) No, I haven’t had the chance
   e) No, I don’t know what that is

Sequence-to-Sequence RNNs

Up until 2017 or so, neural language models were mostly built using recurrent neural networks.



arXiv:1409.3215v3 [cs.CL] 14 Dec 2014

Sequence to Sequence Learning with Neural Networks

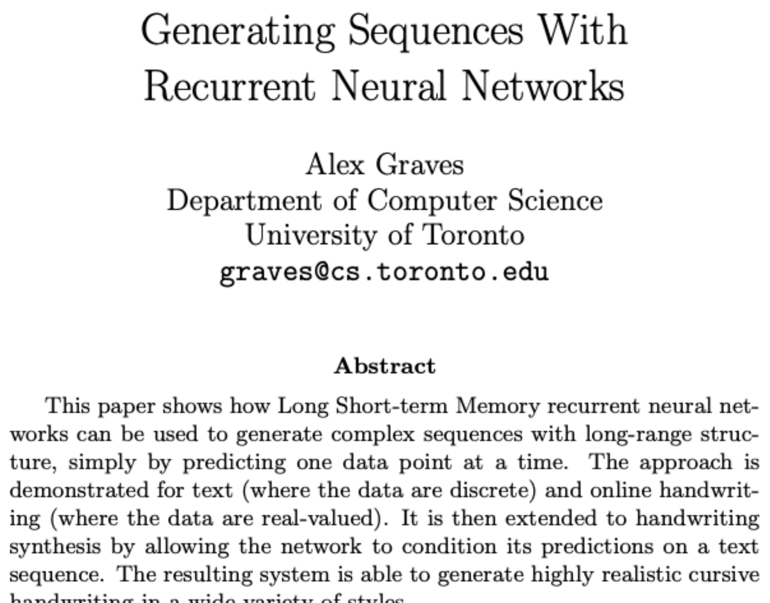
Ilya Sutskever
Google
ilyasu@google.com

Oriol Vinyals
Google
vinyals@google.com

Quoc V. Le
Google
qvl@google.com

Abstract

Deep Neural Networks (DNNs) are powerful models that have achieved excellent performance on difficult learning tasks. Although DNNs work well whenever large labeled training sets are available, they cannot be used to map sequences to sequences. In this paper, we present a general end-to-end approach to sequence learning that makes minimal assumptions on the sequence structure. Our method uses a multilayered Long Short-Term Memory (LSTM) to map the input sequence to a vector of a fixed dimensionality, and then another deep LSTM to decode the target sequence from the vector. Our main result is that on an English to French translation task from the WMT'14 dataset, the translations produced by the LSTM achieve a BLEU score of 34.8 on the entire test set, where the LSTM's BLEU score was penalized on out-of-vocabulary words. Additionally, the LSTM did not have difficulty on long sentences. For comparison, a phrase-based SMT system achieves a BLEU score of 33.3 on the same dataset. When we used the LSTM to rerank the 1000 hypotheses produced by the aforementioned SMT system, its BLEU score increases to 36.5, which is close to the previous best result on this task. The LSTM also learned sensible phrase and sentence representations that are sensitive to word order and are relatively invariant to the active and the passive voice. Finally, we found that reversing the order of the words in all source sentences (but not target sentences) improved the LSTM's performance markedly, because doing so introduced many short term dependencies between the source and the target sentence which made the optimization problem easier.

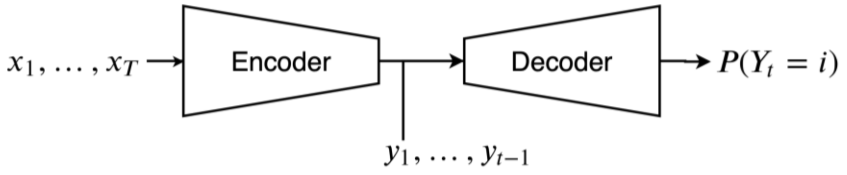


Generating Sequences With Recurrent Neural Networks

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

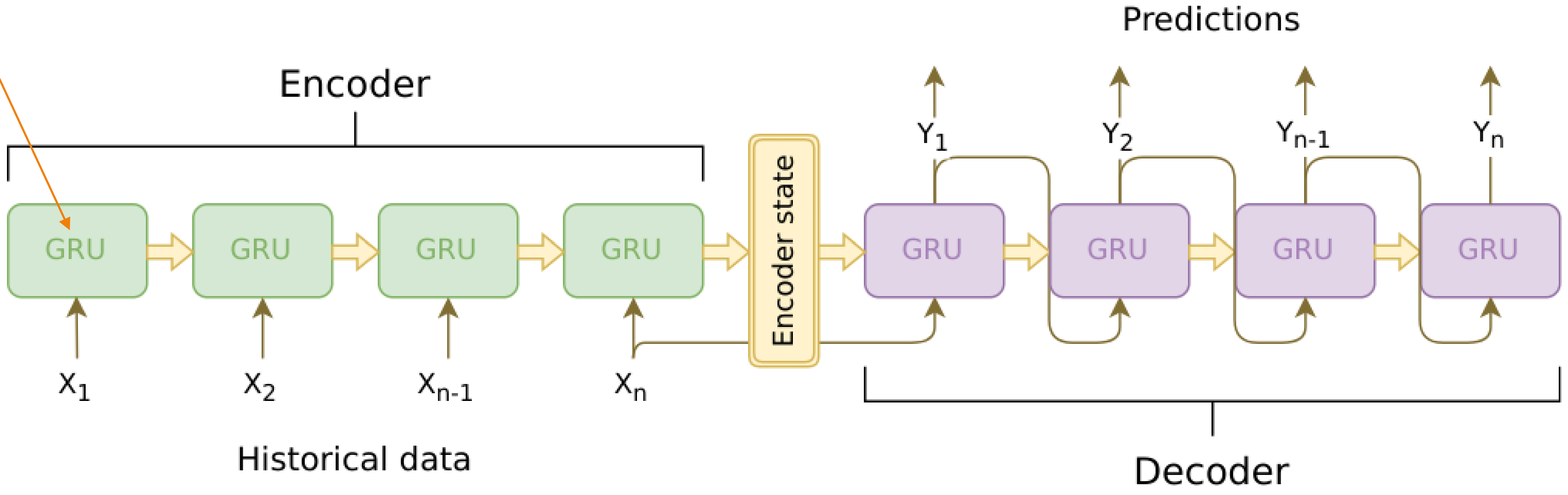
Abstract

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.



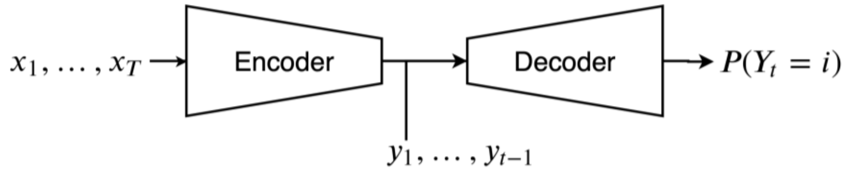
Sequence-to-Sequence / Encoder-Decoder Models

Can be LSTM



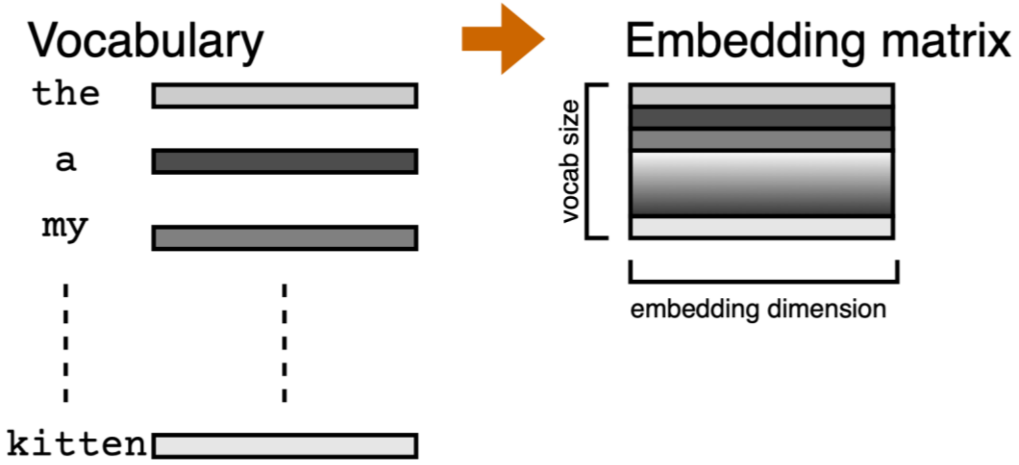
https://jeddy92.github.io/JEddy92.github.io/ts_seq2seq_intro/

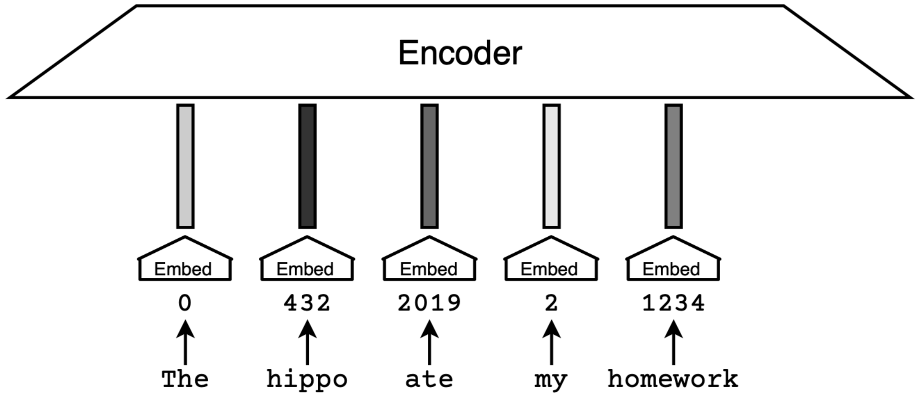
I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," in *Conference on Advances in Neural Information Processing Systems (NeurIPS)*, Montréal, Canada, 2014, pp. 3104–3112. https://proceedings.neurips.cc/paper_files/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2-Abstract.html

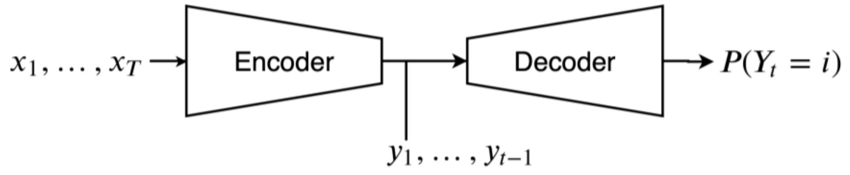


Inputs to the Encoder

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

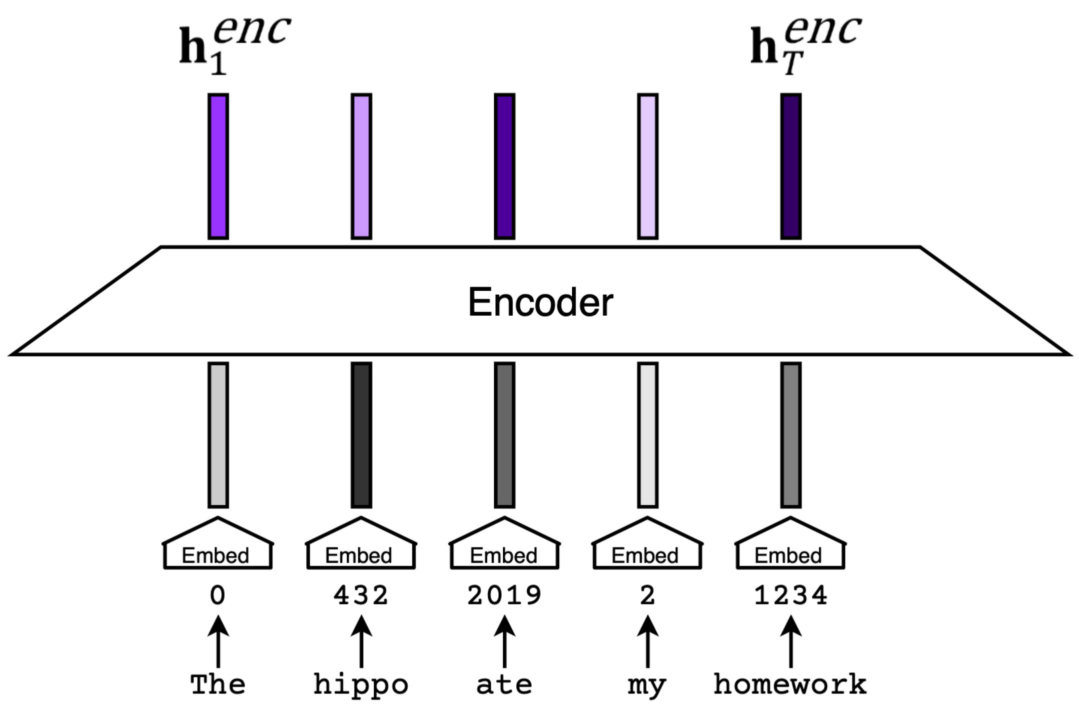


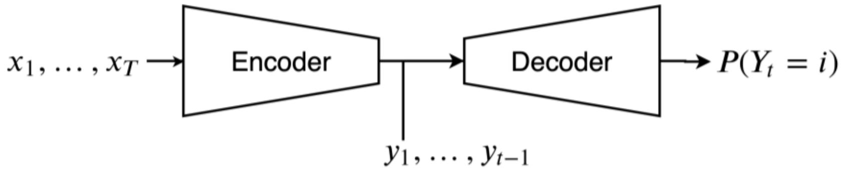




Outputs from the Encoder

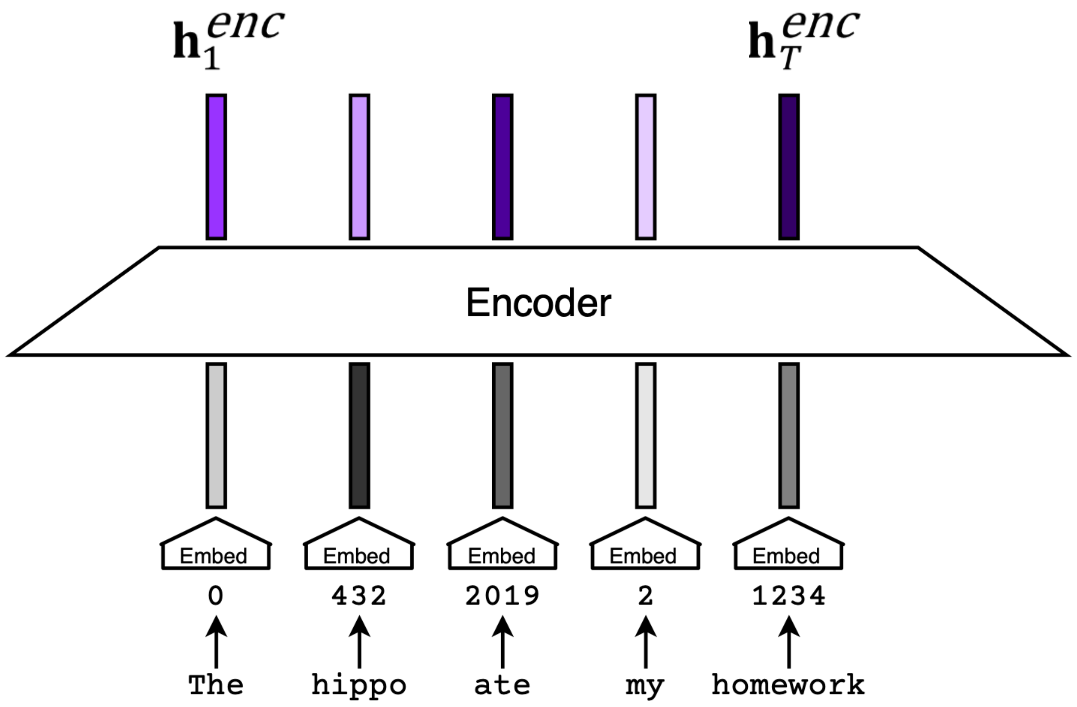
The encoder outputs a sequence of vectors. These are called the hidden state of the encoder.

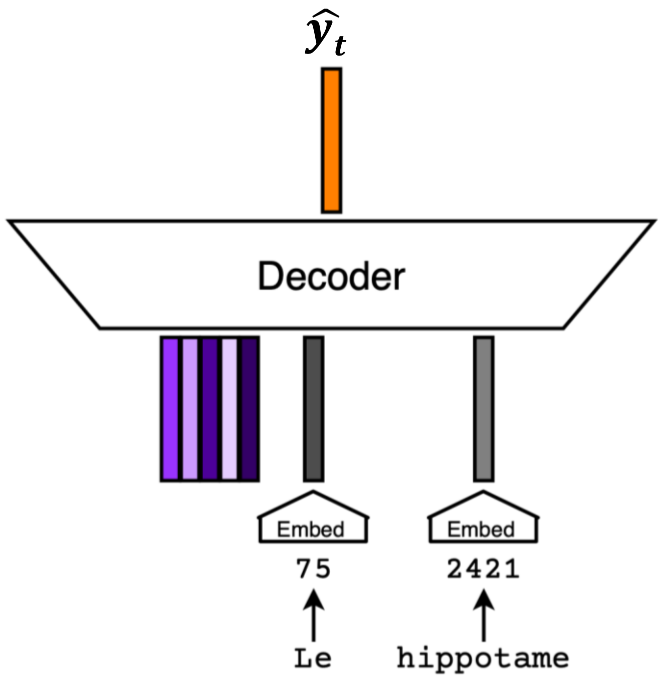


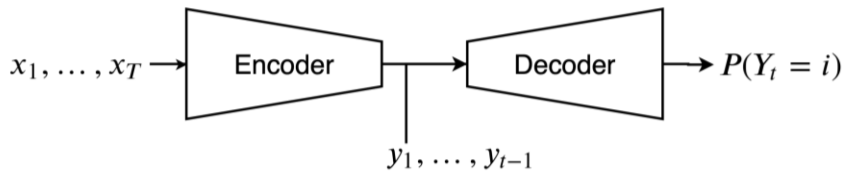


Inputs to the Decoder

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.

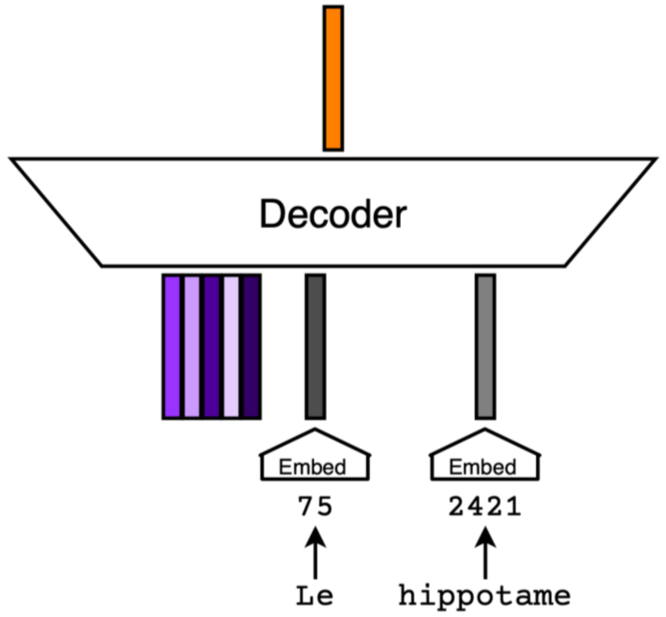






Outputs from the Decoder

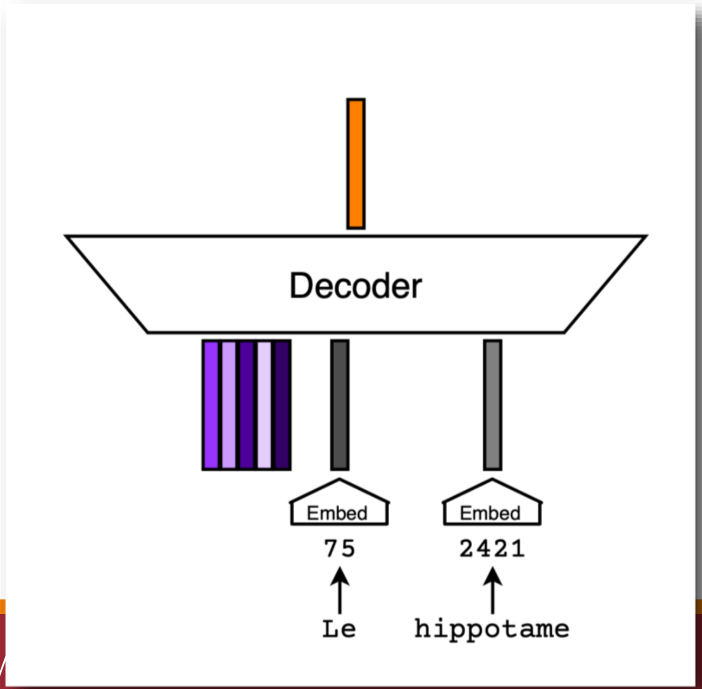
The decoder outputs an embedding $\widehat{\boldsymbol{yt}}$. The goal is for this embedding to be as close as possible to the embedding of the true next token.

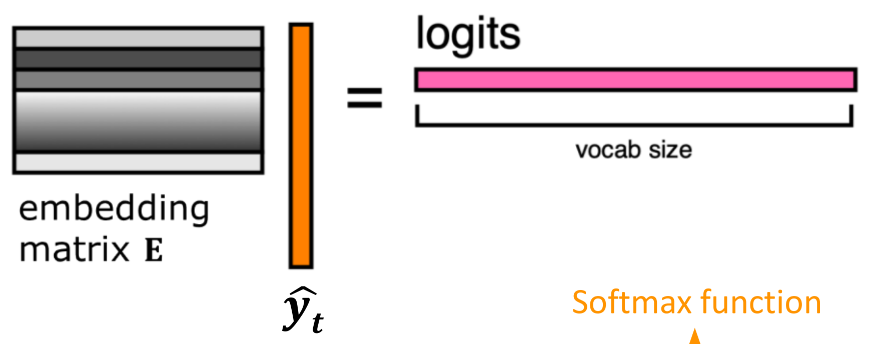


Turning $\widehat{yt}$ into a Probability Distribution

We can multiply the predicted embedding $\widehat{yt}$ by our vocabulary embedding matric to get a score for each vocabulary word. These scores are referred to as logits.

The softmax function then lets us turn the logits into probabilities.





Softmax function

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

Loss Function

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

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

Loss Function

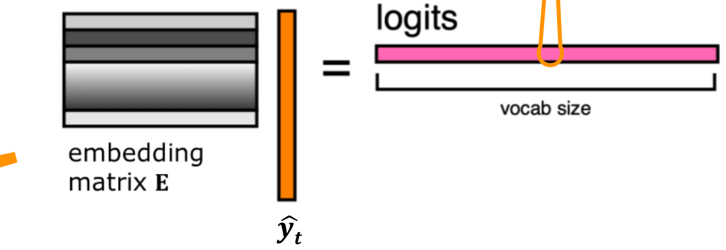
$$\mathcal{L} = -\sum_{t=1}^{T} \log \boxed{P(Y_t = i^* | \mathbf{x}_{1:T}, \mathbf{y}_{1:t-1})}$$

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

Loss Function

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

Score for word at index i^*



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

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

Loss Function

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

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

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

Generating Text

Also sometimes called decoding 🙄

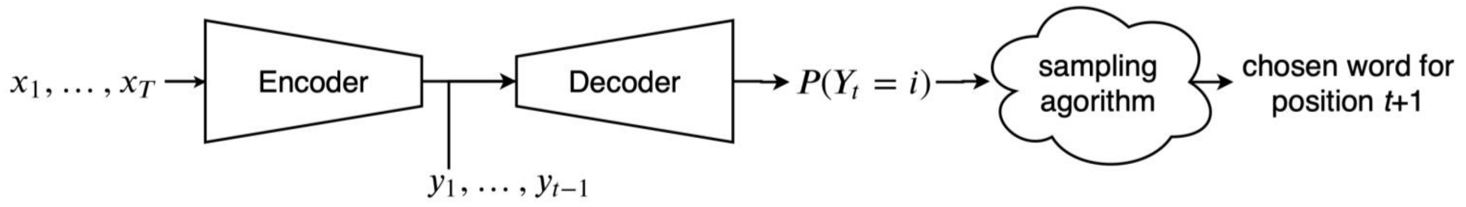
To generate text, we need an algorithm that selects tokens given the predicted probability distributions.

Examples:

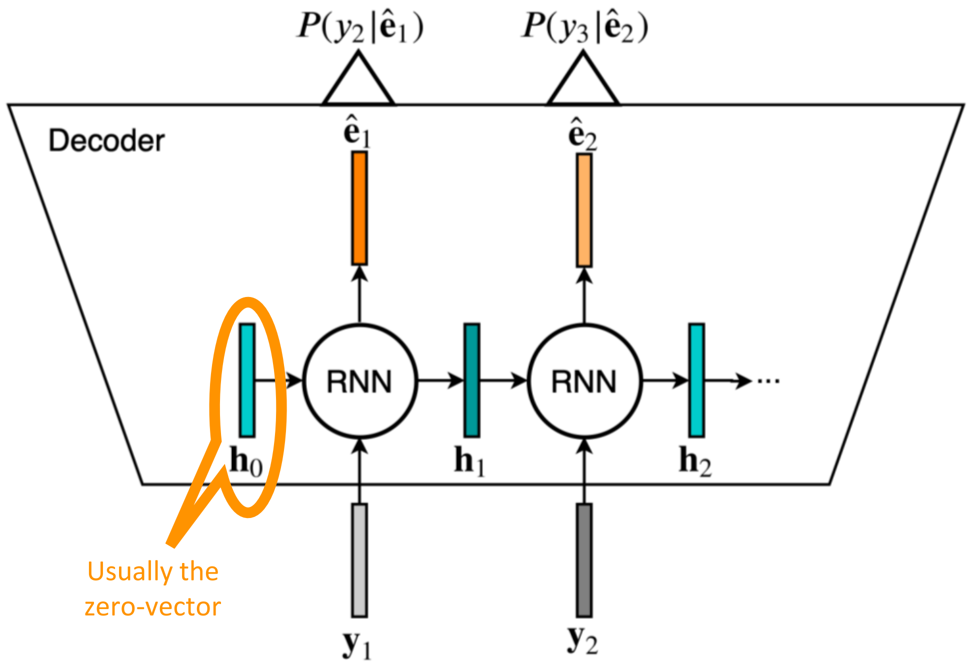
Argmax

Random sampling

Beam search



RNNs - Single Layer Decoder



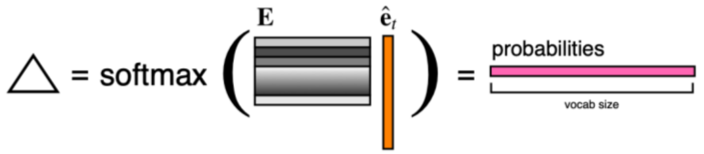
The current hidden state is computed as a function of the previous hidden state and the embedding of the current word in the target sequence.

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

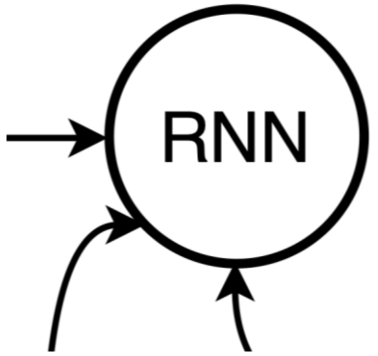
The current hidden state is used to predict an embedding for the next word in the target sequence.

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

This predicted embedding is used in the loss function:



What is the “RNN” unit?



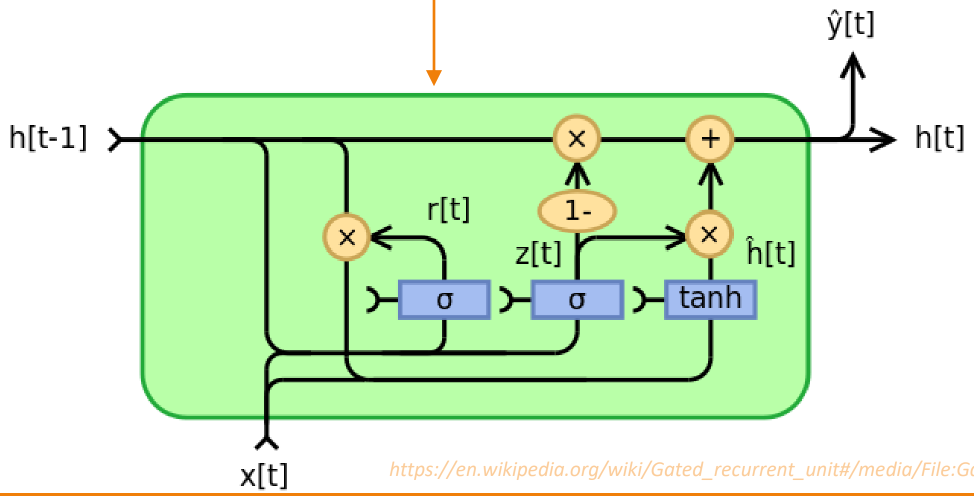
?

LSTMs/GRUs

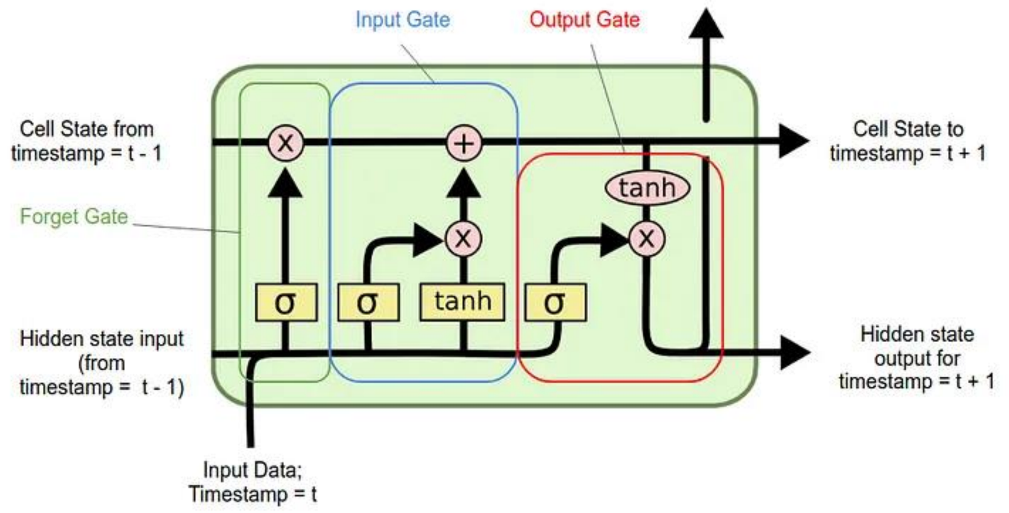
LSTM: Long Short-Term Memory (Hochreiter & Schmidhuber, 1997)

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

GRU: Gated Recurrent Unit (Cho et al., 2014)

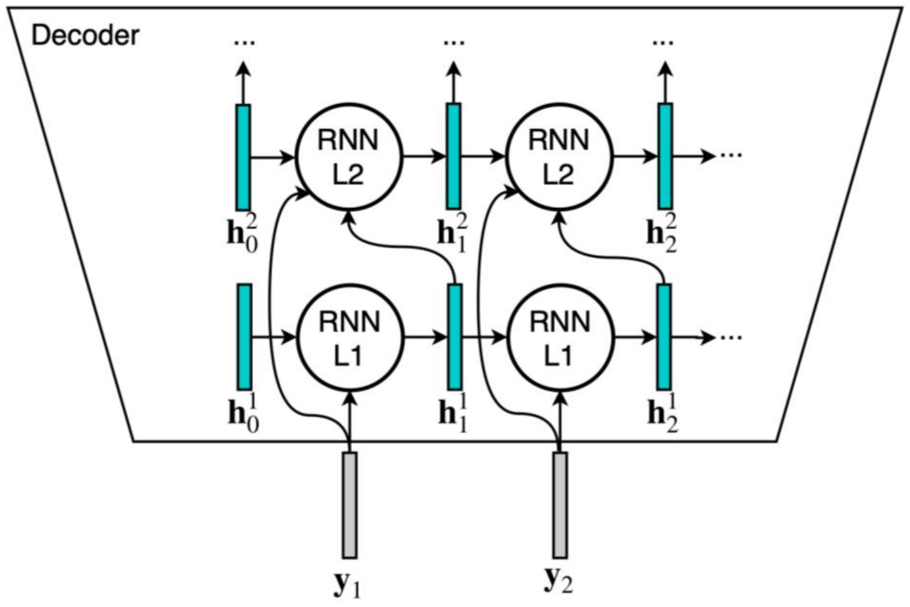


https://en.wikipedia.org/wiki/Gated_recurrent_unit#/media/File:Gated_Recurrent_Unit,_base_type.svg


Input Gate
Output Gate
Cell State from timestamp = t - 1
Cell State to timestamp = t + 1
Forget Gate
tanh
σ
σ
tanh
σ
Hidden state input (from timestamp = t - 1)
Hidden state output for timestamp = t + 1
Input Data; Timestamp = t

https://medium.com/analytics-vidhya/lstms-explained-a-complete-technically-accurate-conceptual-guide-with-keras-2a650327e8f2

RNN 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 subsequent layers:

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

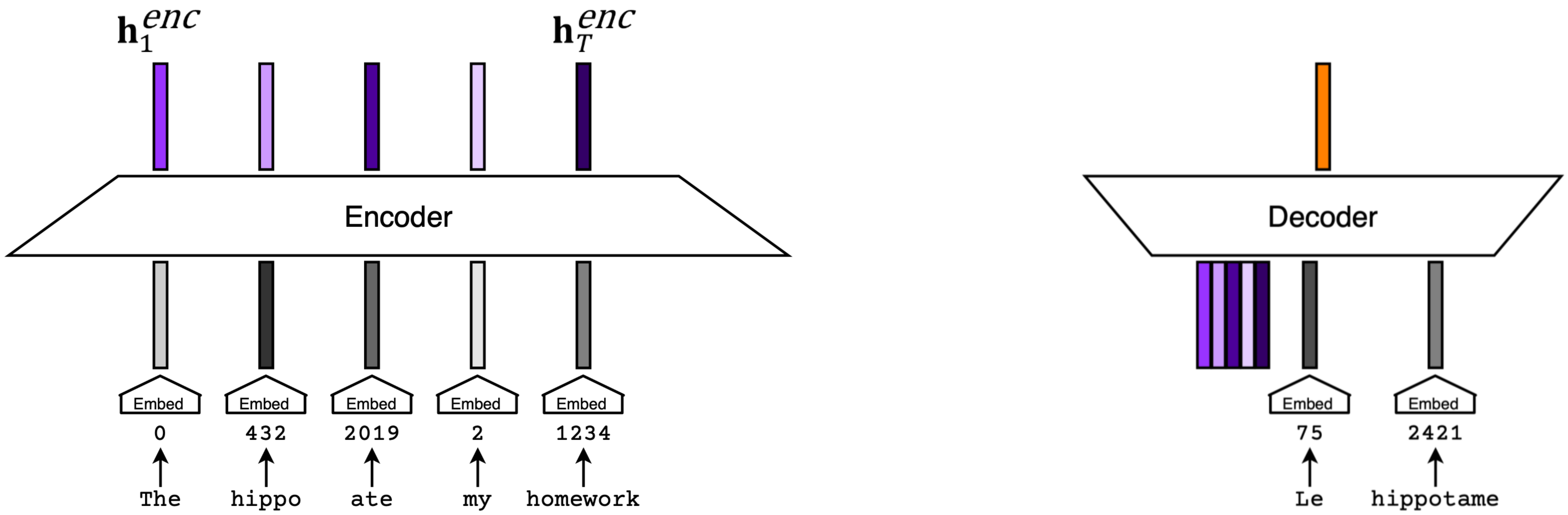
Predicting an embedding for the next token in the sequence:

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

Each of the b and W are learned bias and weight matrices.

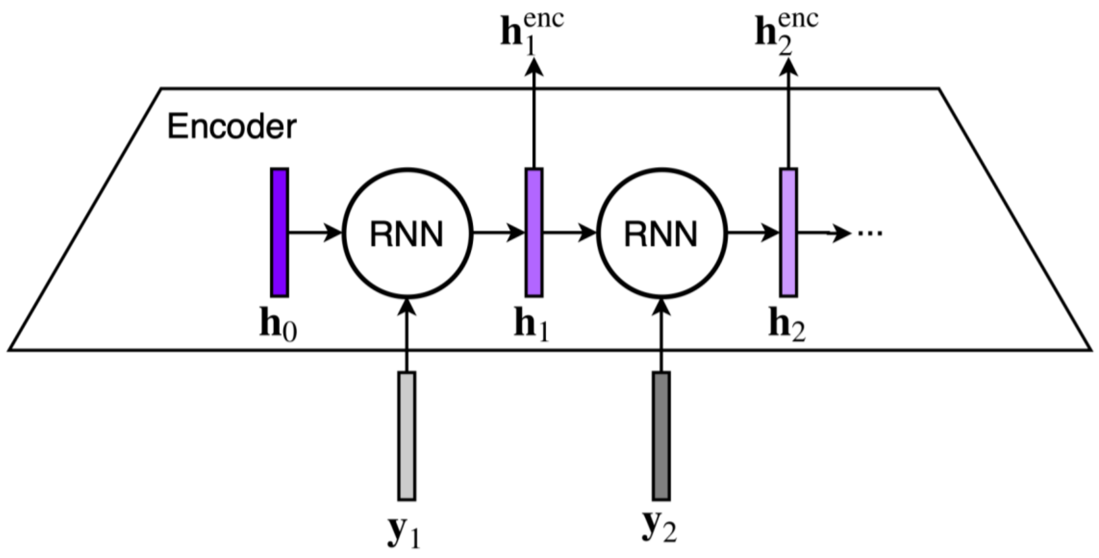
RNN Encoder-Decoder Architectures

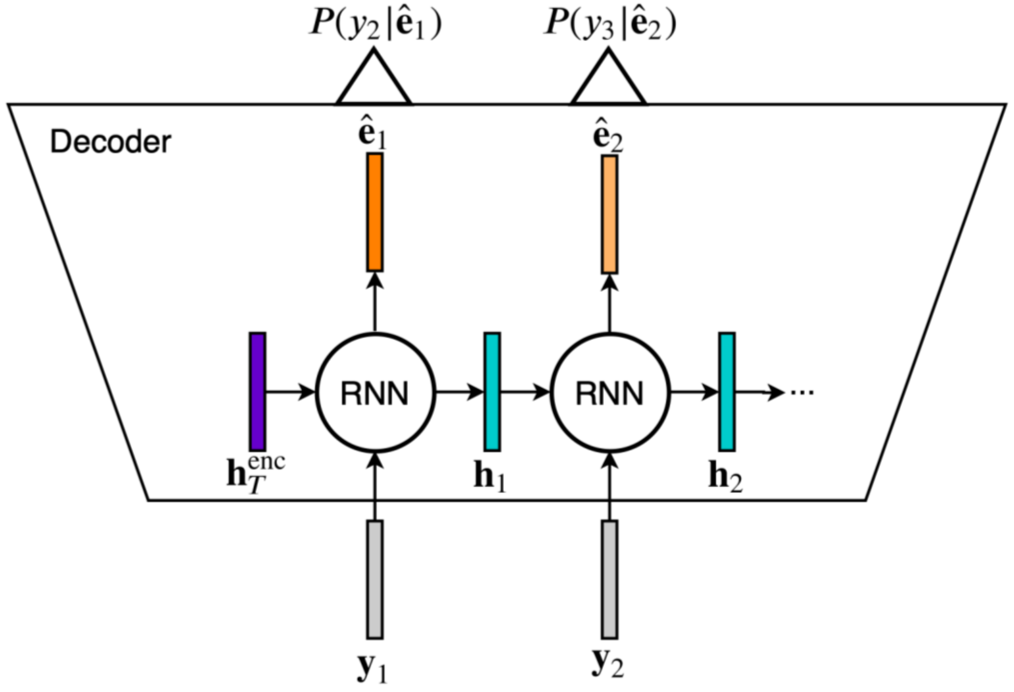
How do we implement an encoder-decoder model?



RNN Encoder-Decoder Architectures

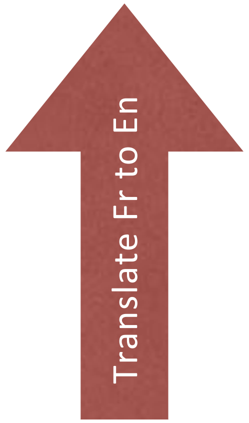
Simplest approach: Use the final hidden state from the encoder to initialize the first hidden state of the decoder.





RNN Encoder-Decoder Architectures

[The, hippopotamus, …

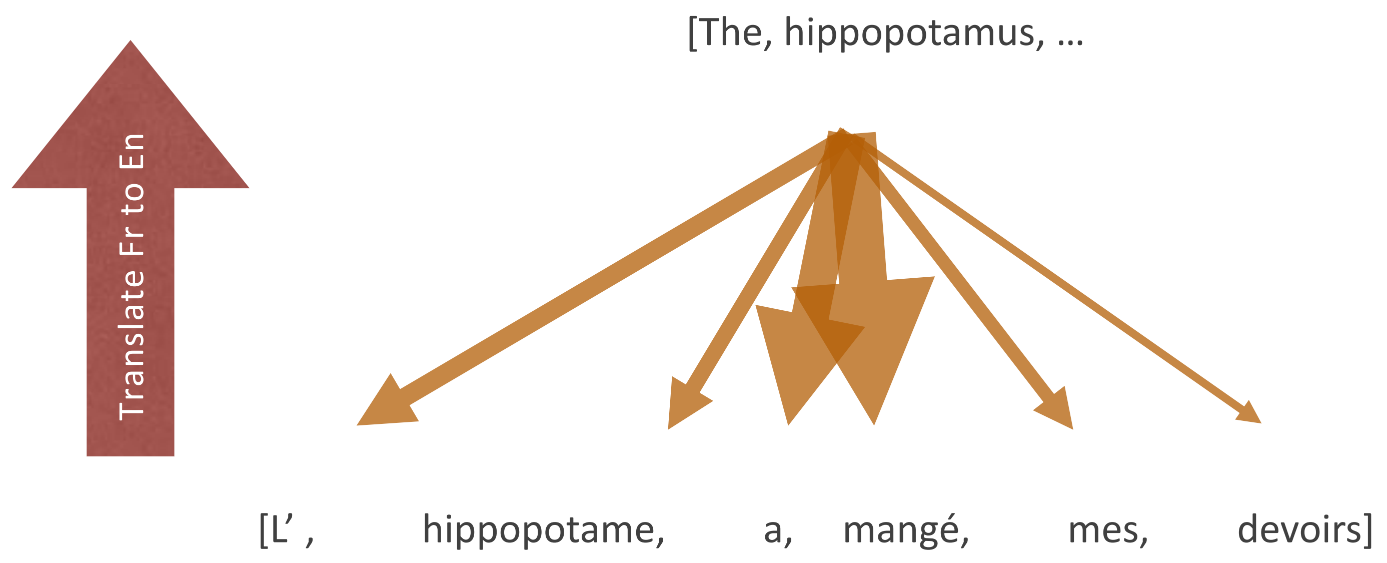


When predicting the next English word, how much weight should the model put on each French word in the source sequence?

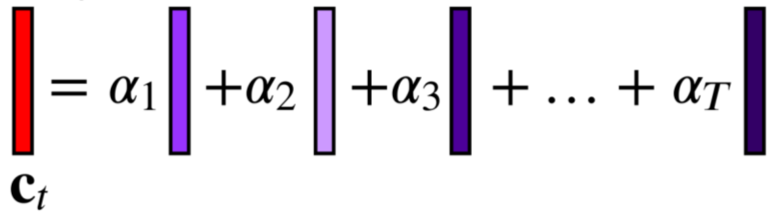
[L’ , hippopotame, a, mangé, mes, devoirs]

Attention

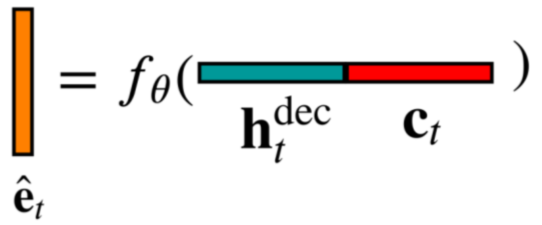
Better approach: an attention mechanism



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.



RNN Encoder-Decoder Architectures

The tth context vector is computed as $\mathbf{c}t = \mathbf{H}^{\text{enc}} at$

at[i] = softmax(att_score($\mathbf{h}_t^{\text{dec}}$, $\mathbf{h}i^{\text{enc}}$))

There are a few different options for the attention score:

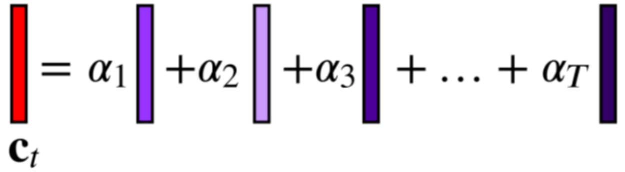
$$\text{att\_score}(\mathbf{h}_t^{\text{dec}}, \mathbf{h}_i^{\text{enc}}) = \begin{cases} \mathbf{h}_t^{\text{dec}} \cdot \mathbf{h}_i^{\text{enc}} \\ \mathbf{h}_t^{\text{dec}}\, \mathbf{W}a\, \mathbf{h}_i^{\text{enc}} \\ w_{a1}^{\top} \tanh(\mathbf{W}a\mathbf{2}[\mathbf{h}_t^{\text{dec}}, \mathbf{h}_i^{\text{enc}}]) \end{cases}$$

dot product

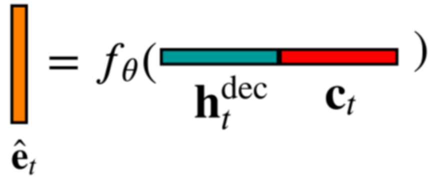
bilinear function

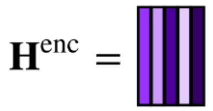
MLP

Compute a linear combination of the encoder hidden states.

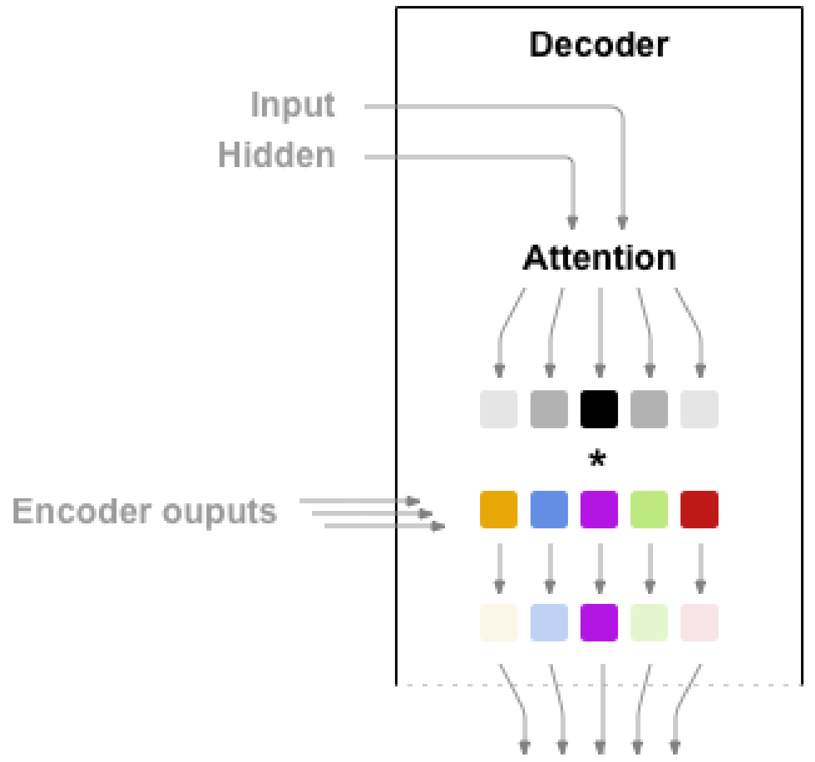


Decoder's prediction at position *t* is based on both the context vector and the hidden state outputted by the RNN at that position.





Attention Decoder



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

Limitations of Recurrent architecture

Slow to train.

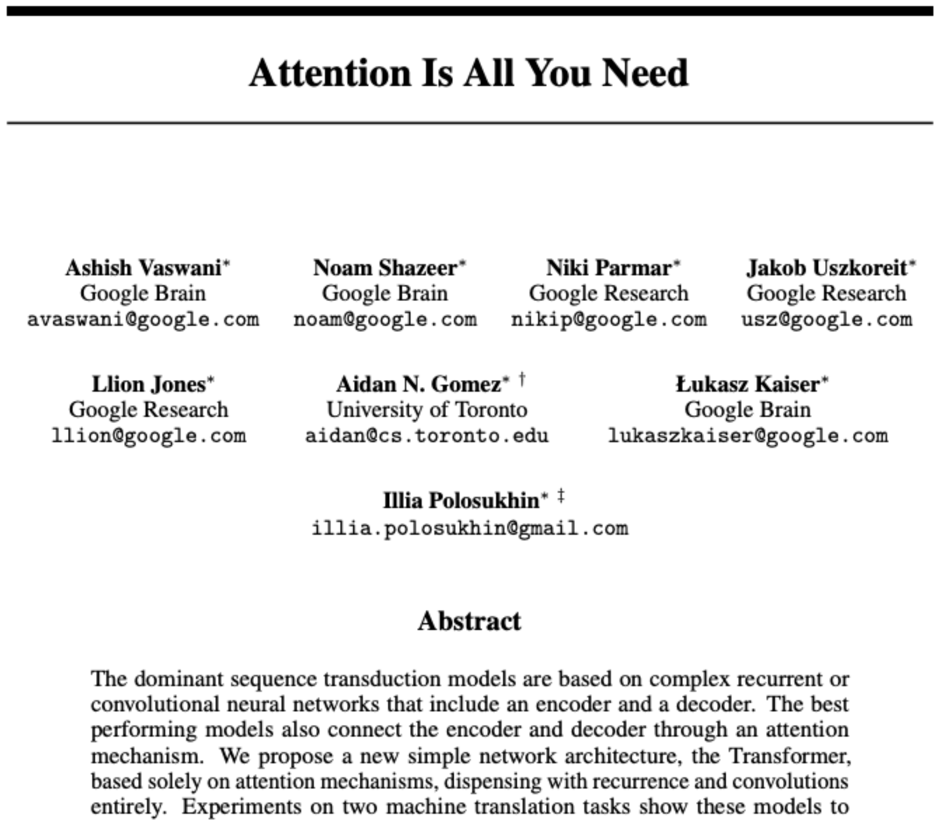
- Can’t be easily parallelized.
- The computation at position t is dependent on first doing the computation at position t-1.

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.

Transformers

Since 2018, the field has rapidly standardized on the Transformer architecture



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

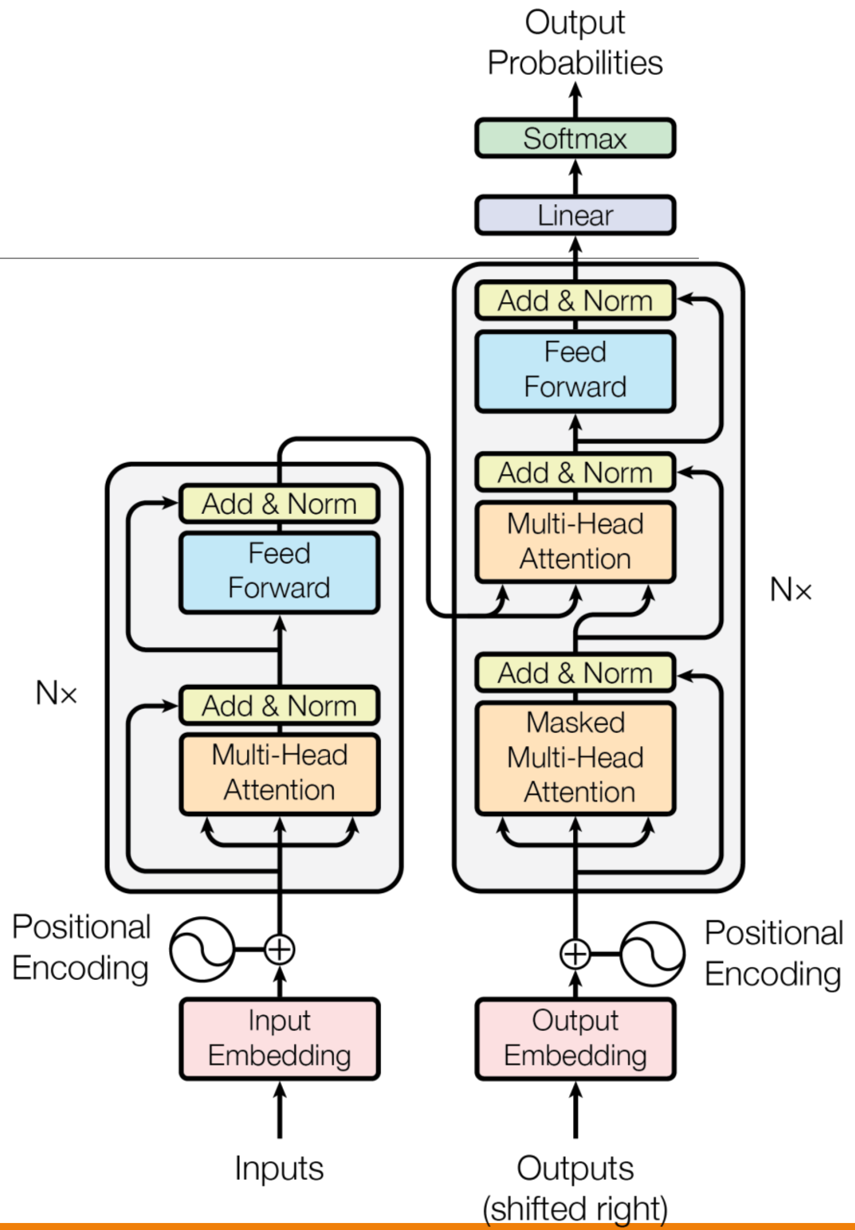
Abstract

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

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


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)