



BST 261: Data Science II

Lecture 14

Attention Models, Transformers

Santiago Romero Brufau
Harvard T.H. Chan School of Public Health
Spring 2



Administrivia

Today: Transformers

Wednesday 3rd: AI Safety, guest lecture by Max Nadeau, Deputy Director,
Harvard AI Safety Team

Friday 5th: Lab 5, RNNs

Next Monday 8th: Lecture over Zoom! (guest lecture by Sandeep Konam,
CTO/Founder at Abridge)

Next Wednesday 10th: Lecture over Zoom! (student presentations,
game/exam)

The background of the slide is a complex network diagram. It consists of numerous nodes, represented by small circles, some of which are solid grey and others are hollow with a grey outline. These nodes are interconnected by a web of thin, light-grey lines, creating a dense, interconnected pattern that fills the entire slide area.

Attention

<https://youtu.be/kCc8FmEb1nY>

Attention Is All You Need

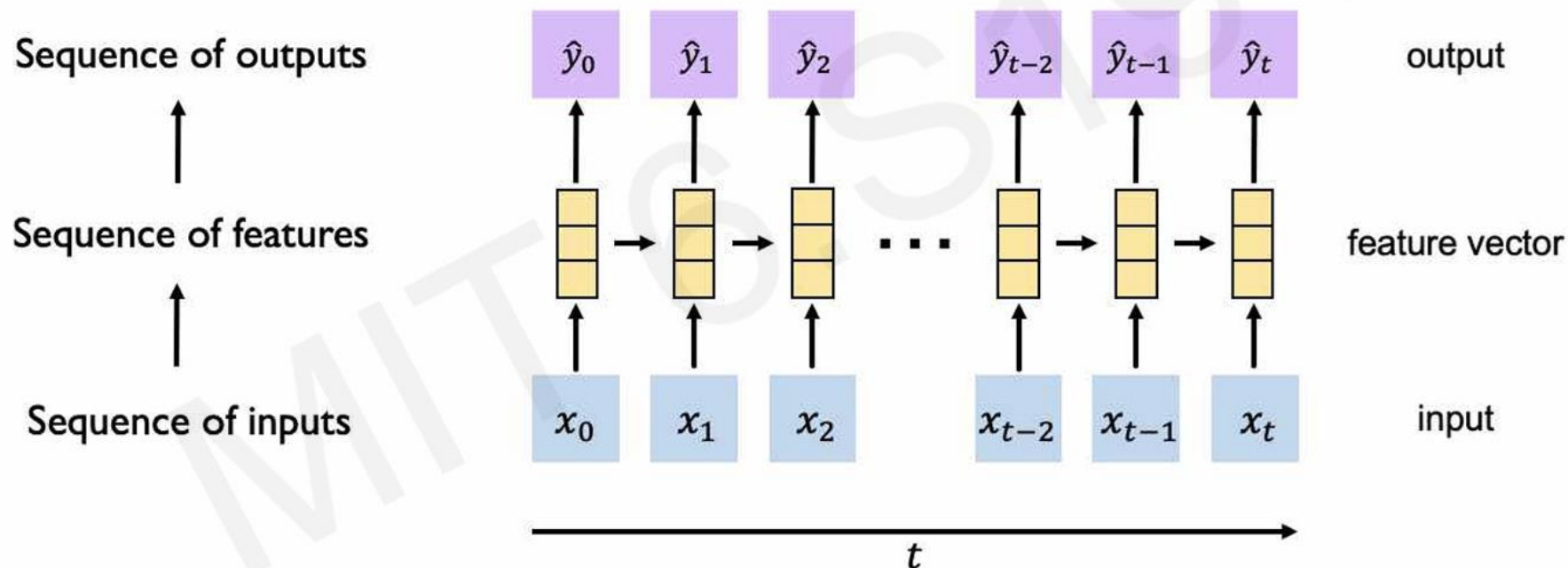
Ashish Vaswani* Google Brain avaswani@google.com	Noam Shazeer* Google Brain noam@google.com	Niki Parmar* Google Research nikip@google.com	Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com	Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu	Lukasz Kaiser* 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.

Goal of Sequence Modeling

RNNs: recurrence to model sequence dependencies



Goal of Sequence Modeling

RNNs: recurrence to model sequence dependencies

Limitations of RNNs



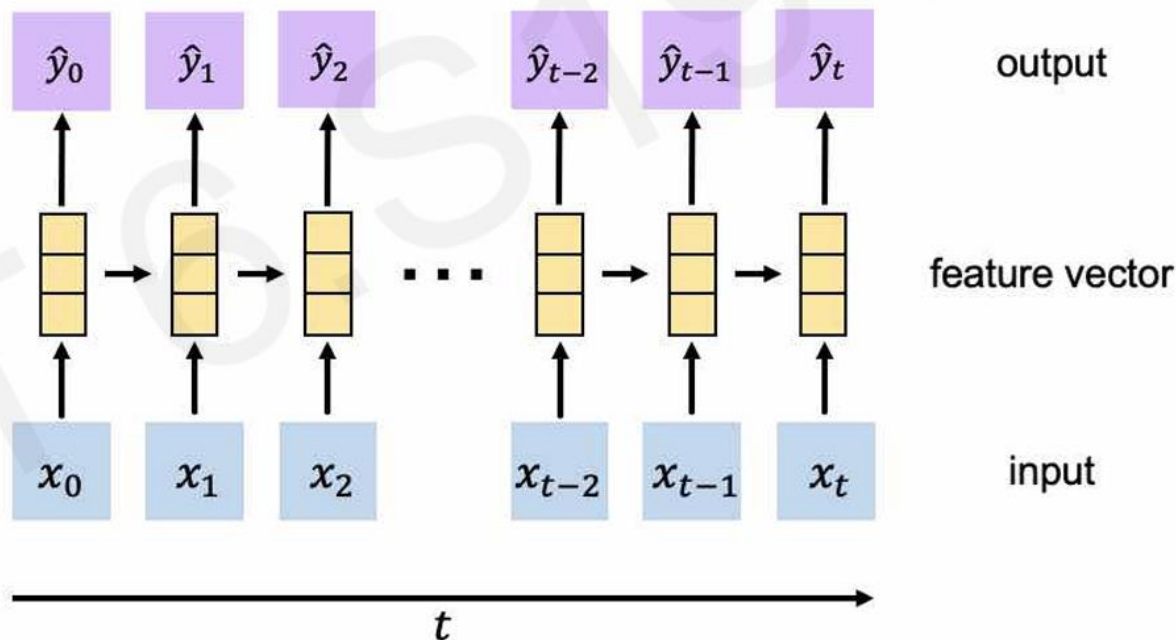
Encoding bottleneck



Slow, no parallelization



Not long memory




Goal of Sequence Modeling

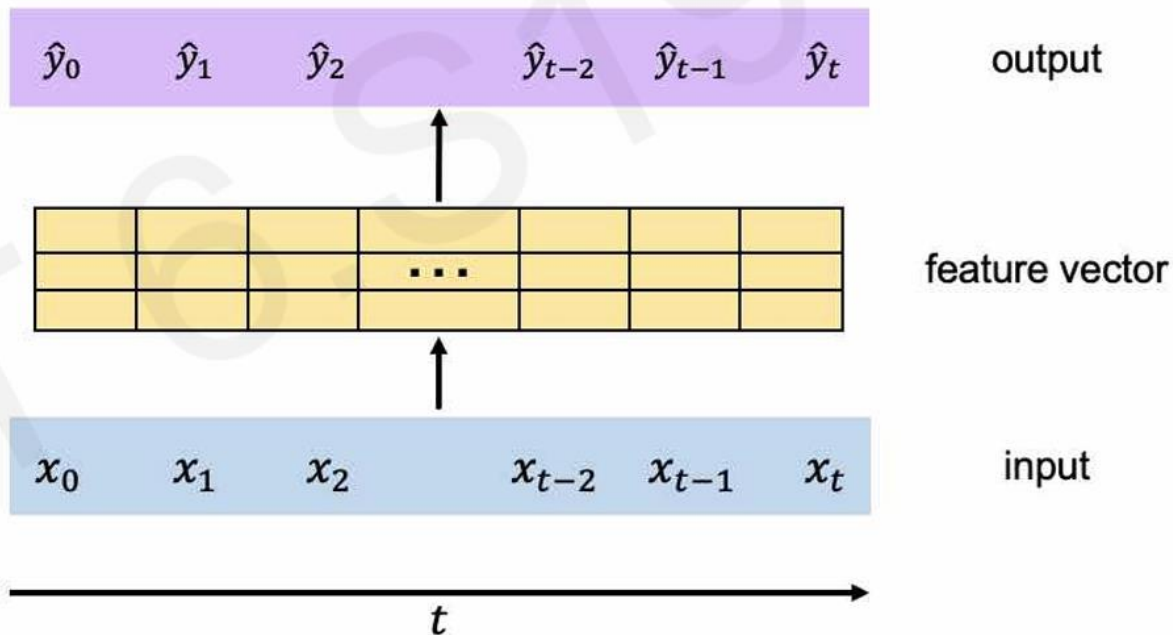
Can we eliminate the need for recurrence entirely?

Desired Capabilities

 Continuous stream

 Parallelization

 Long memory



Goal of Sequence Modeling

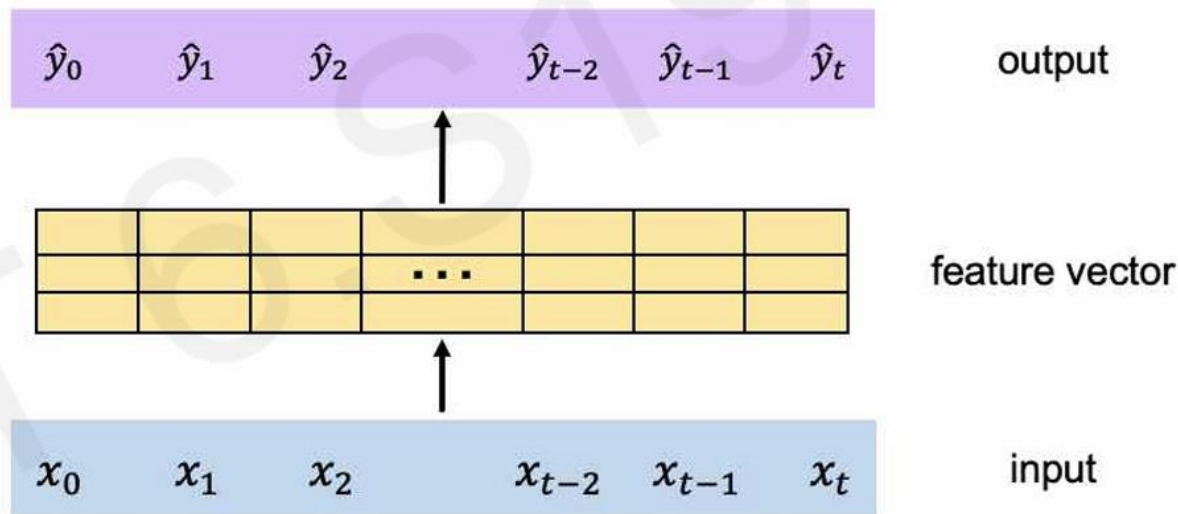
Idea 1: Feed everything
into dense network

- ✓ No recurrence
- ✗ Not scalable
- ✗ No order
- ✗ No long memory



Idea: Identify and attend
to what's important

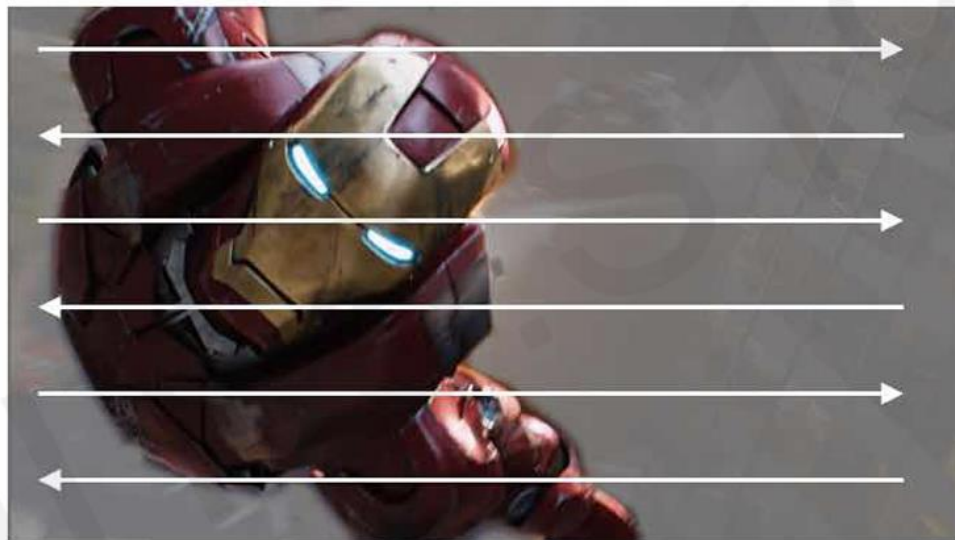
Can we eliminate the need for
recurrence entirely?



Attention Is All You Need

Intuition Behind Self-Attention

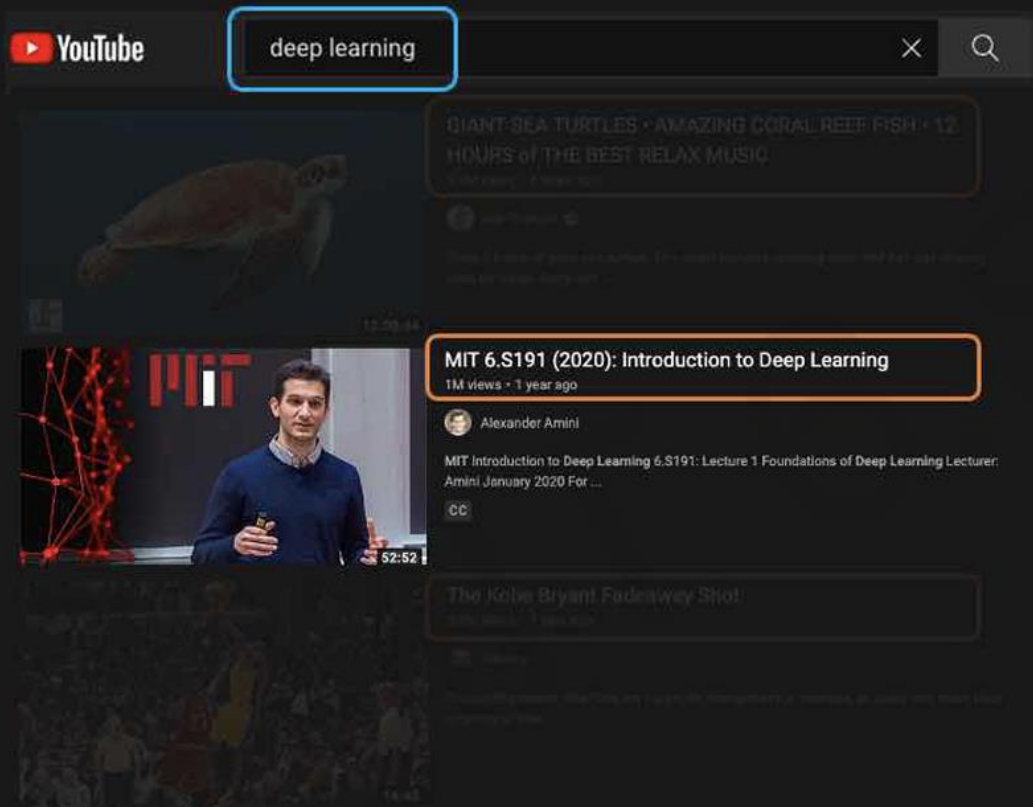
Attending to the most important parts of an input.



1. Identify which parts to attend to
2. Extract the features with high attention

Similar to a search problem!

Understanding Attention with Search



Query (Q)

Key (K_1)

Key (K_2)

Key (K_3)

How similar is the
key to the query?

I. **Compute attention mask:** how
similar is each key to the desired query?

Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract query, key, value for search
3. Compute attention weighting
4. Extract features with high attention

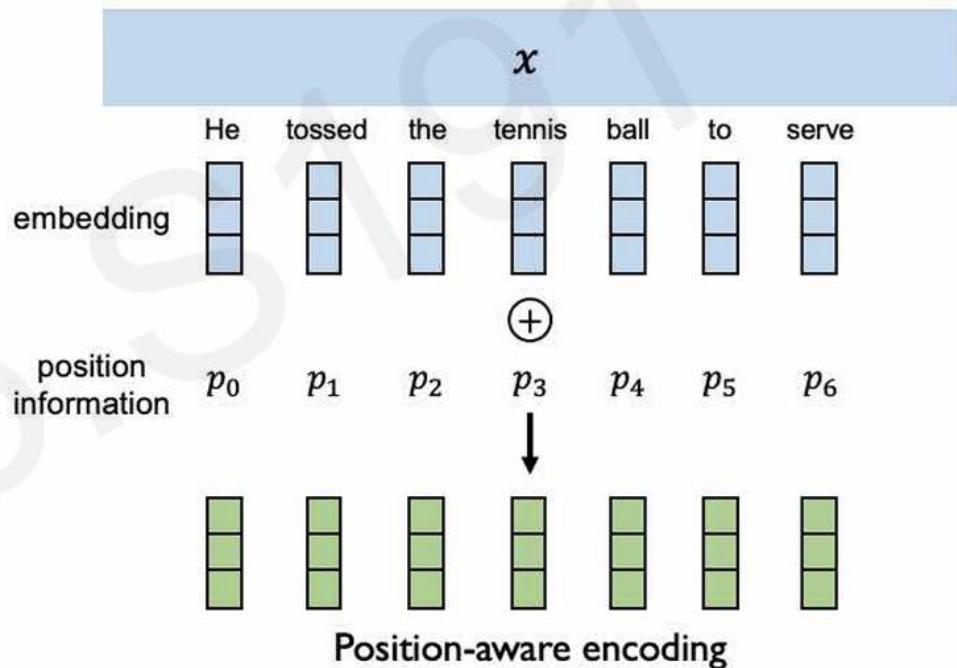


Data is fed in all at once! Need to encode position information to understand order.

Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract query, key, value for search
3. Compute attention weighting
4. Extract features with high attention

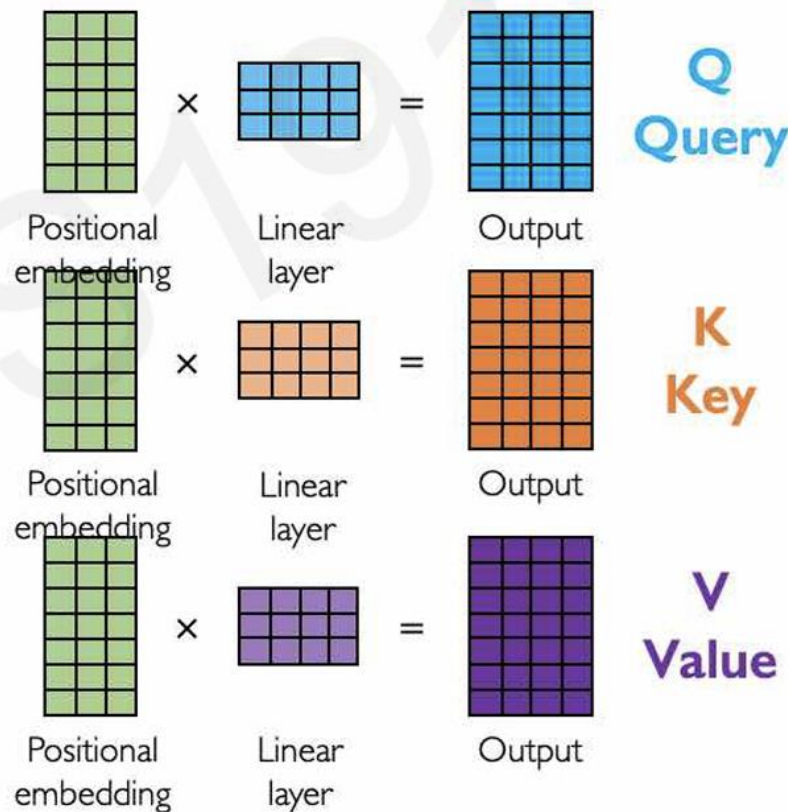


Data is fed in all at once! Need to encode position information to understand order.

Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute attention weighting
4. Extract features with high attention



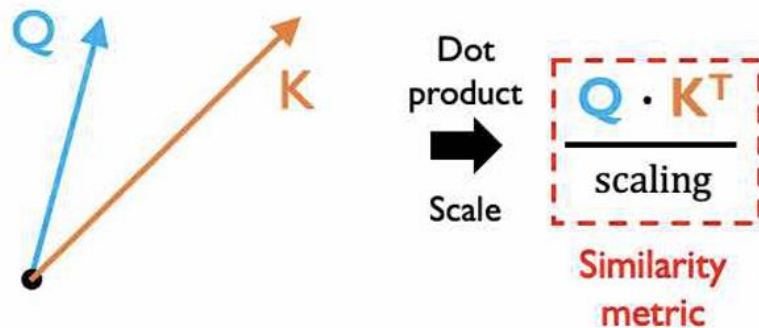
Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract features with high attention

Attention score: compute pairwise similarity between each **query** and **key**

How to compute similarity between two sets of features?



Also known as the "cosine similarity"

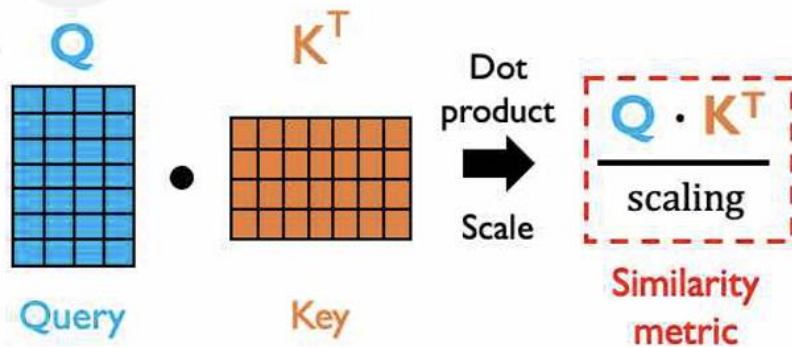
Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

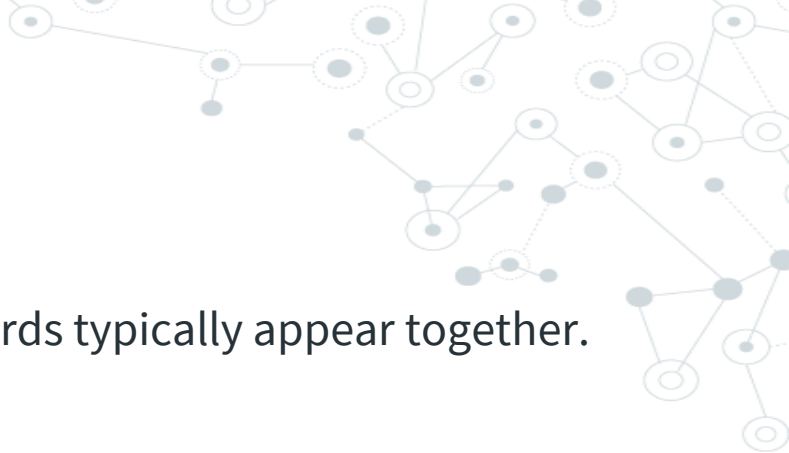
1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract features with high attention

Attention score: compute pairwise similarity between each **query** and **key**

How to compute similarity between two sets of features?



Also known as the “cosine similarity”



Remember embeddings are based on which words typically appear together.

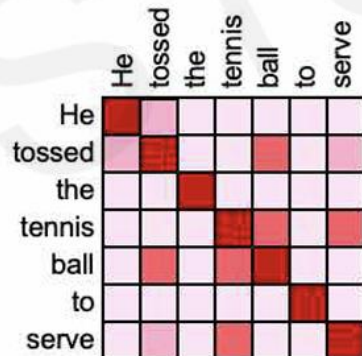
So, computing the “similarity” between two words is actually estimating how “related” they are, in the sense of how often they tend to appear with similar words.

Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

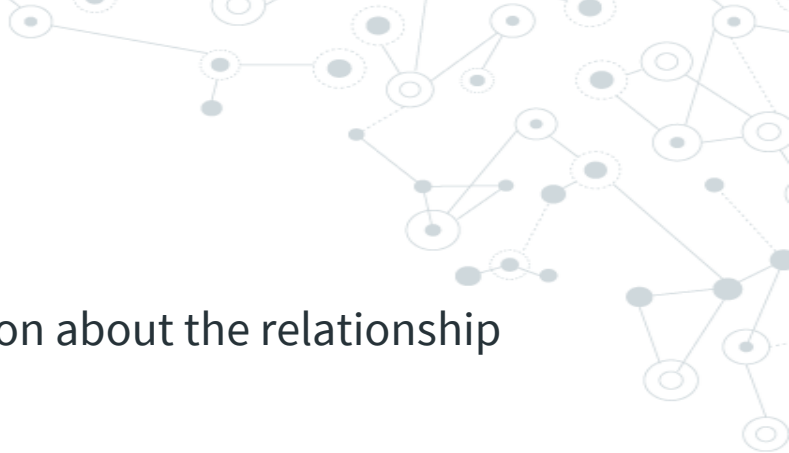
1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract features with high attention

Attention weighting: where to attend to?
How similar is the key to the query?



$$\text{softmax} \left(\frac{Q \cdot K^T}{\text{scaling}} \right)$$

Attention weighting



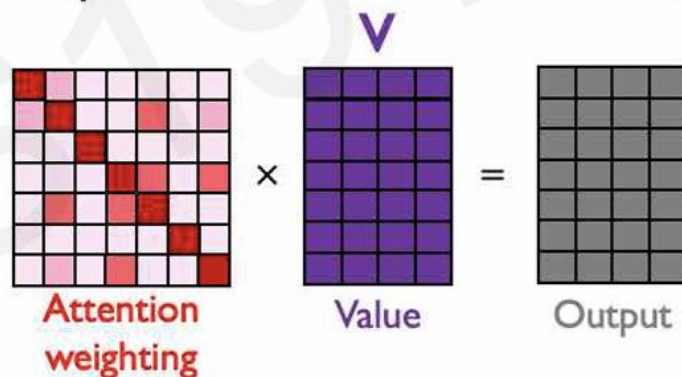
That “similarity matrix” now includes information about the relationship between different words.

Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**

Last step: self-attend to extract features



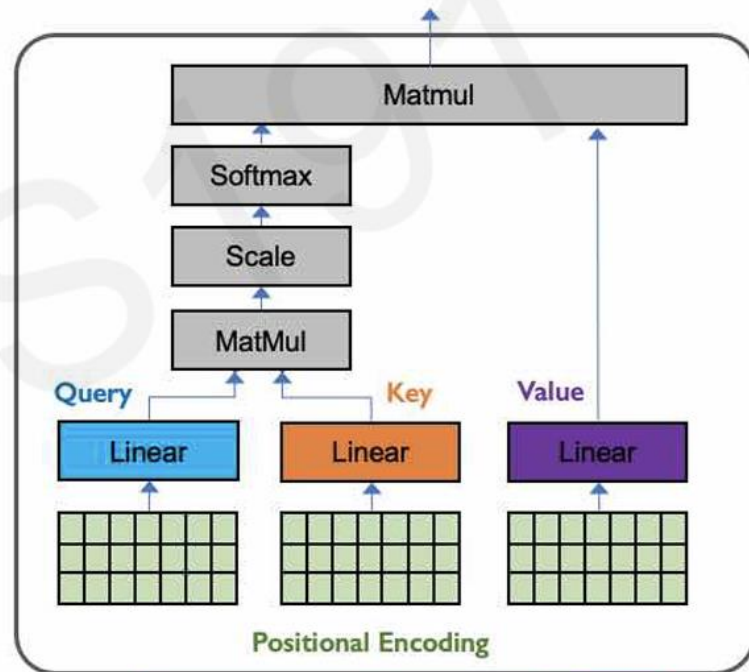
$$\text{softmax} \left(\frac{Q \cdot K^T}{\text{scaling}} \right) \cdot V = A(Q, K, V)$$

Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**

These operations form a self-attention head that can plug into a larger network.
Each head attends to a different part of input.



$$\text{softmax} \left(\frac{Q \cdot K^T}{\text{scaling}} \right) \cdot V$$

The background of the slide is a light gray network pattern. It consists of numerous small circles, some of which are solid gray and others are hollow with a gray outline. These circles are interconnected by a web of thin, light gray lines, creating a complex, organic structure that resembles a neural network or a data graph.

Transformers

Recall

Problems with RNNs (even LSTM and GRU layers)

- ◎ Loss of information (very long sequences)
- ◎ Vanishing gradient problem
- ◎ No parallel computing
 - Values calculated at the end depend on all previous calculations in all previous time steps
 - Very computationally expensive

Recall

Problems with RNNs (even LSTM and GRU layers)

- ◎ Loss of information (very long sequences)
- ◎ Vanishing gradient problem
- ◎ No parallel computing
 - Values calculated at the end depend on all previous calculations in all previous time steps
 - Very computationally expensive



Attention helps with these

Recall

Problems with RNNs (even LSTM and GRU layers)

- ◎ Loss of information (very long sequences)
- ◎ Vanishing gradient problem
- ◎ No parallel computing
 - Values calculated at the end depend on all previous calculations in all previous time steps
 - Very computationally expensive



Transformers to the rescue!



Transformers

- ◎ Solve all of the problems with classic RNNs
 - Allow for parallel computing
 - Use **attention**
 - ◎ Helps with loss of information problem
- ◎ [Attention is all you need paper](#)
 - December 2017
 - Huge breakthrough in NLP

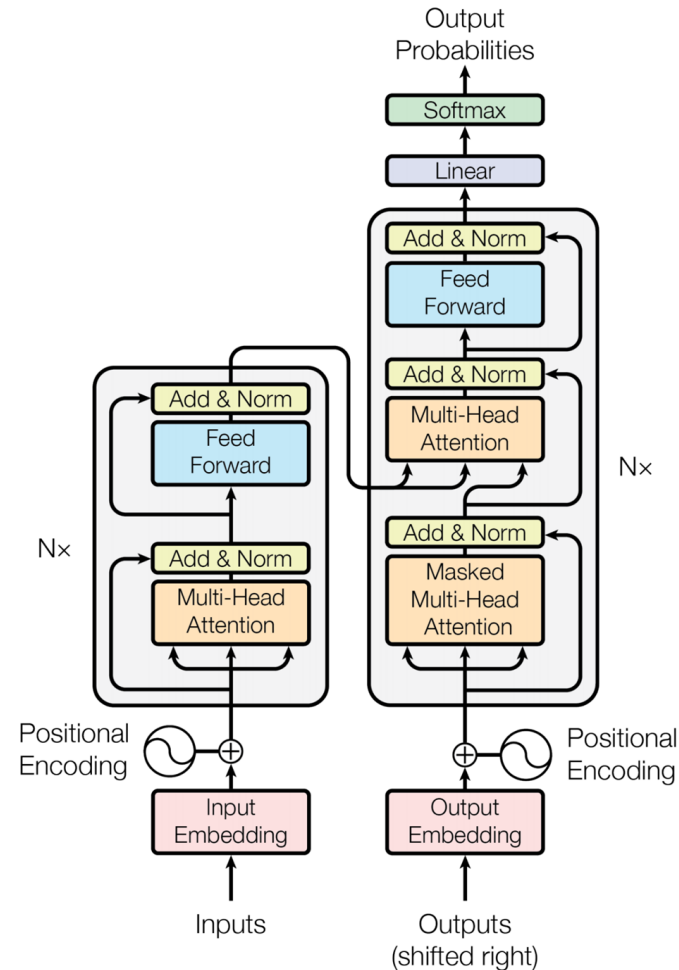


Figure 1: The Transformer - model architecture.

Transformers

- ◎ Solve all of the problems with classic RNNs
 - Allow for parallel computing
 - Use **attention**
 - ◎ Helps with loss of information problem
- ◎ [Attention is all you need paper](#)
 - December 2017
 - Huge breakthrough in NLP

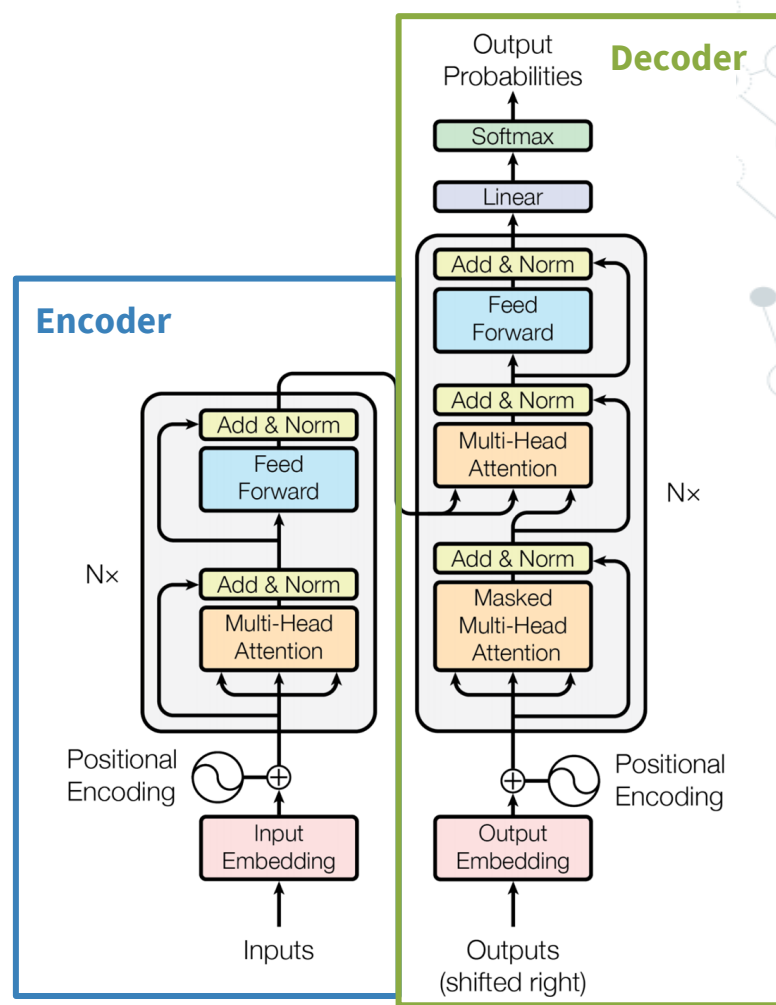


Figure 1: The Transformer - model architecture.

Multi-Head Attention

Main idea: self-attention, multiple times

- ◎ Improves performance of the self-attention layer
 - Expands the model's ability to focus on different positions in the input sequence
 - Gives the attention layer multiple “representation subspaces”
 - ◎ Multiple sets of query, key, and value matrices

Multi-Head Attention

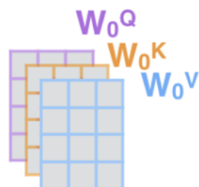
1) This is our input sentence*

Thinking
Machines

2) We embed each word*



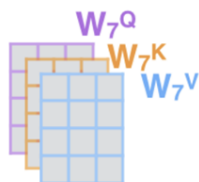
3) Split into 8 heads. We multiply X or R with weight matrices



...

...

...



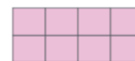
4) Calculate attention using the resulting $Q/K/V$ matrices

5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

W^O

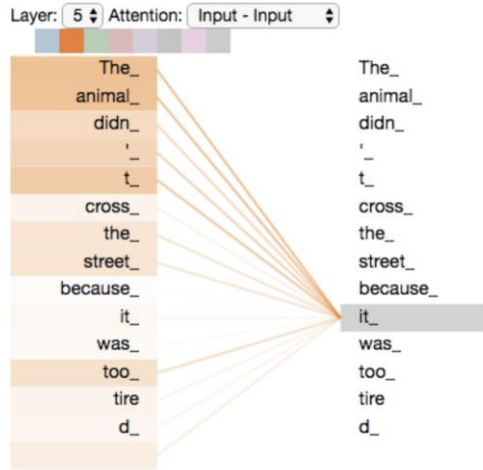


Z

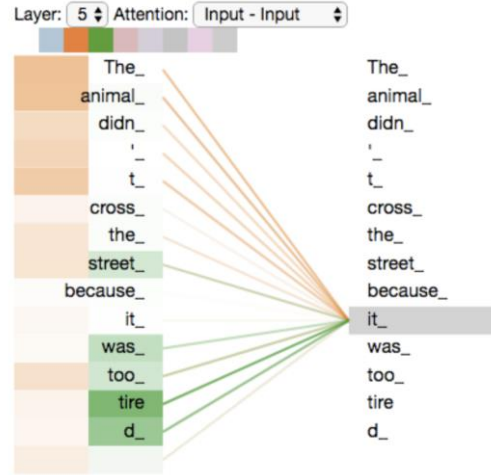


* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

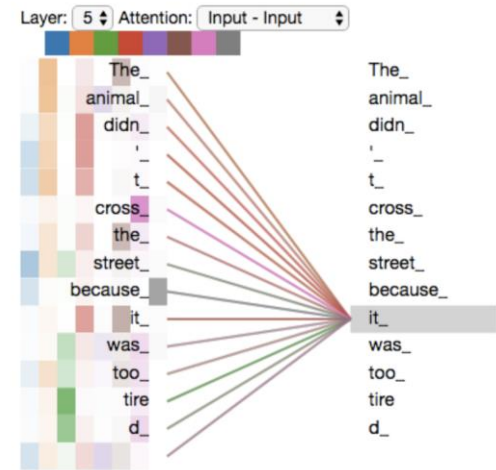
Multi-Head Attention



Self-attention



2-head Self-attention



8-head Self-attention

Transformers

State of the art models

- [GPT-2](#), [GPT-3](#)
- [BERT](#)
- [T5](#)

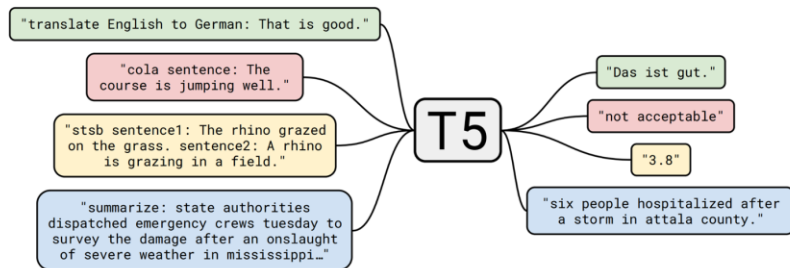
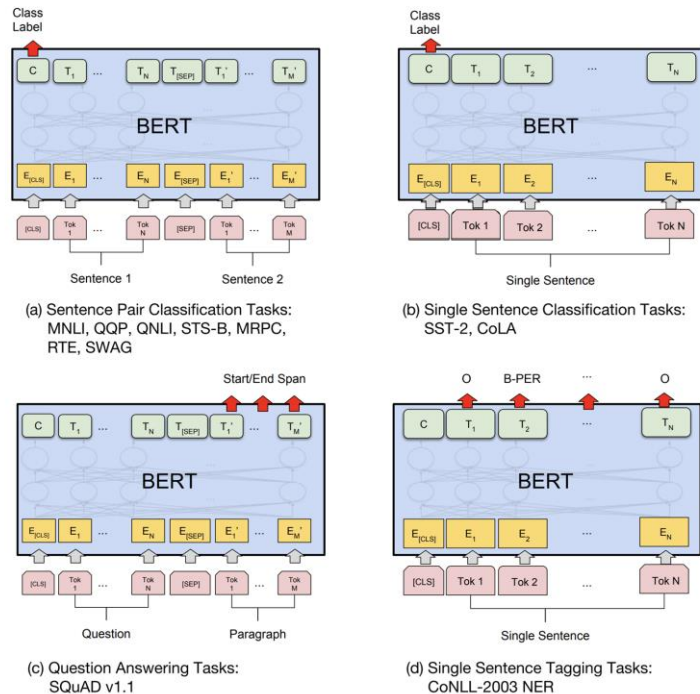
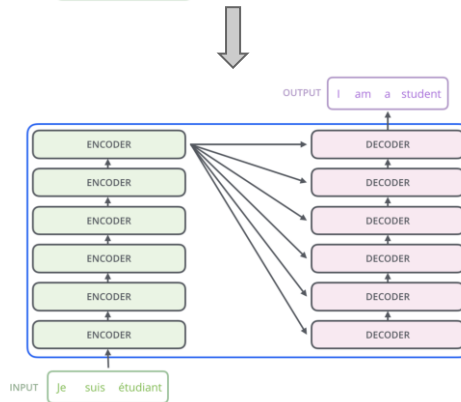
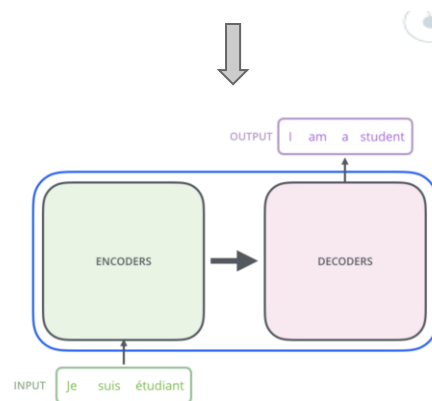
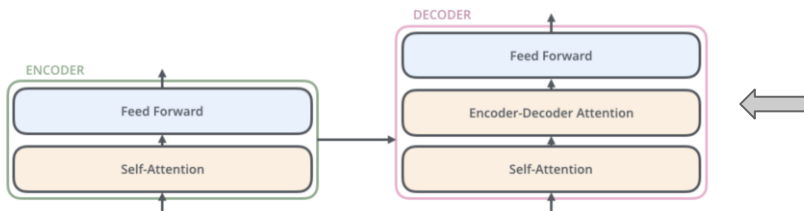


Figure 1: A diagram of our text-to-text framework. Every task we consider—including translation, question answering, and classification—is cast as feeding our model text as input and training it to generate some target text. This allows us to use the same model, loss function, hyperparameters, etc. across our diverse set of tasks. It also provides a standard testbed for the methods included in our empirical survey. “T5” refers to our model, which we dub the “Text-to-Text Transfer Transformer”.

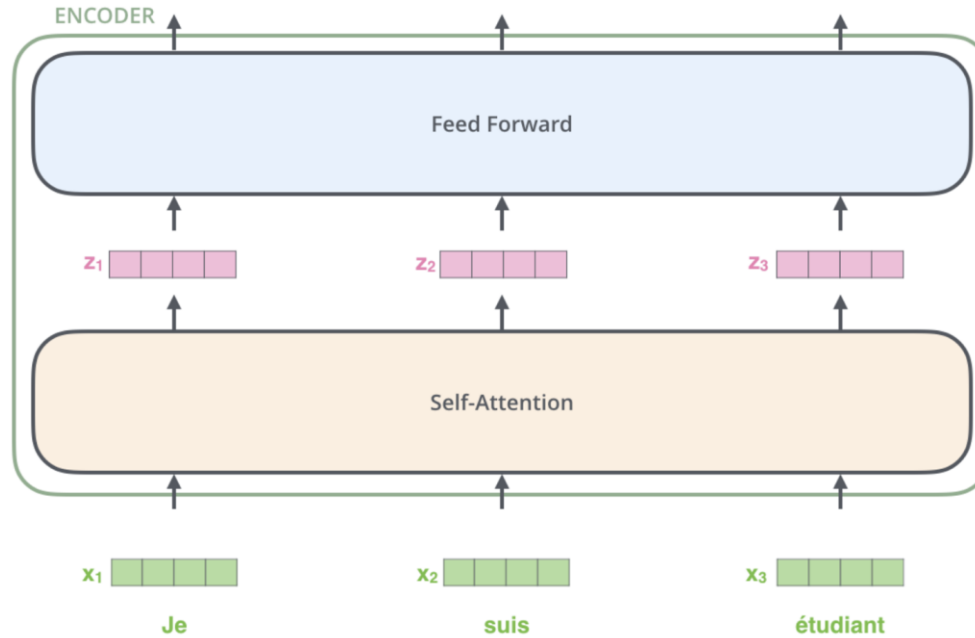


Transformers

- Consist of “transformer blocks”
 - There are encoder and decoder transformer blocks
 - 6 layers each in the original paper
 - Each encoder block contains a self-attention layer and a dense (feedforward) network
 - Each decoder block contains a self-attention layer, an encoder-decoder attention layer, and a dense (feedforward) network

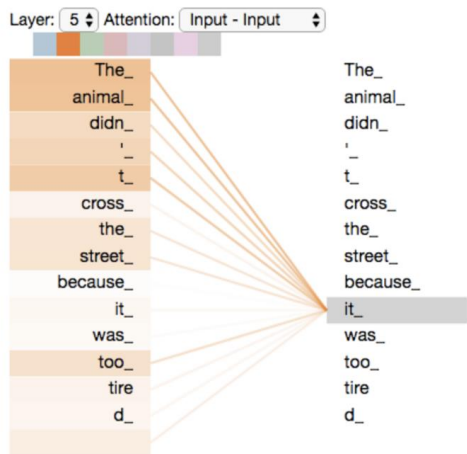


Encoder Block



Self-Attention

- Self-attention is attention that takes place in one block - rather than between an encoder and decoder
- Uses the input to learn context and identify important words

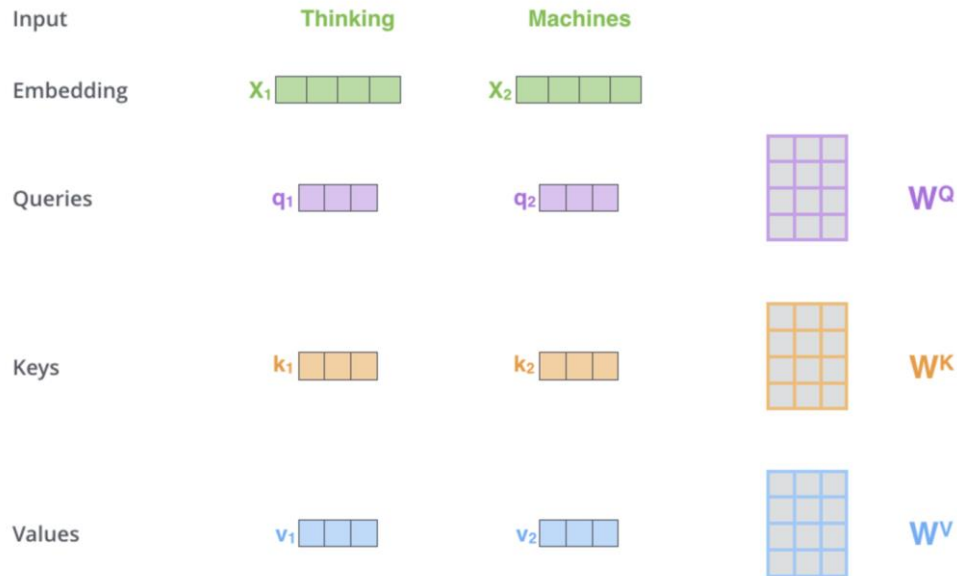


As we are encoding the word "it" in encoder #5 (the top encoder in the stack), part of the attention mechanism was focusing on "The Animal", and baked a part of its representation into the encoding of "it".

Self-Attention

Steps

1. Create 3 vectors from the embedding vector of each word in the input sequence - query, keys, and values vectors.
 - These are created by multiplying the embedding vector by the query, keys, and values matrices that are trained during the training process



Multiplying x_1 by the W^Q weight matrix produces q_1 , the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.

Self-Attention

2. Calculate a score for each word

- ⦿ Dot product of the query and key vectors
- ⦿ Determines which other words in the input sequence to focus on

3. Divide scores by $\sqrt{d_k}$ (d_k is the dimension of the key vector)

- ⦿ Leads to more stable gradients

4. Pass the result through a softmax operation

Input

Embedding

Queries

Keys

Values

Score

Divide by 8 ($\sqrt{d_k}$)

Softmax

Thinking

x_1

q_1

k_1

v_1

$q_1 \cdot k_1 = 112$

14

0.88

Machines

x_2

q_2

k_2

v_2

$q_1 \cdot k_2 = 96$

12

0.12

Self-Attention

5. Multiple each value vector by the softmax score

- ⦿ Keeps the values of important words intact and minimizes the values of non-informative words (words you can ignore)

6. Sum the weighted value vectors

- ⦿ This is the self-attention output

Input

Embedding

Queries

Keys

Values

Score

Divide by $8 (\sqrt{d_k})$

Softmax

Softmax
X
Value

Sum

Thinking

x_1 [] [] [] []

q_1 [] [] []

k_1 [] [] []

v_1 [] [] []

$q_1 \cdot k_1 = 112$

14

0.88

v_1 [] [] []

z_1 [] [] []

Machines

x_2 [] [] [] []

q_2 [] [] []

k_2 [] [] []

v_2 [] [] []

$q_2 \cdot k_2 = 96$

12

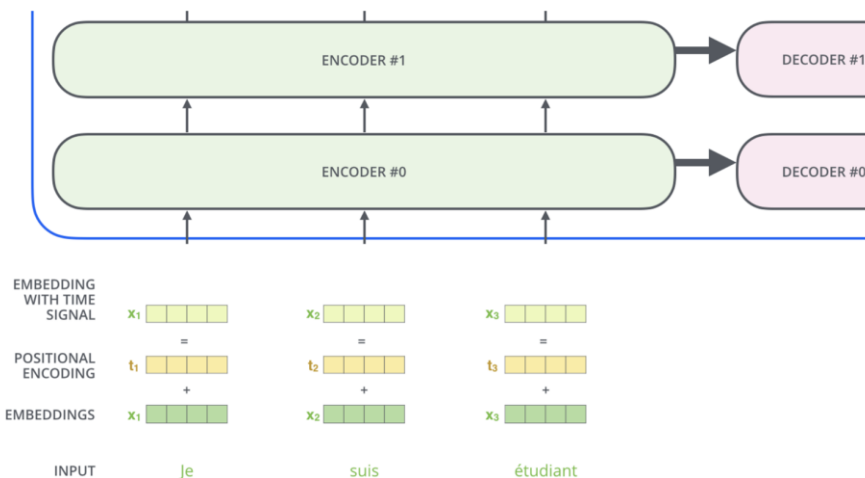
0.12

v_2 [] [] []

z_2 [] [] []

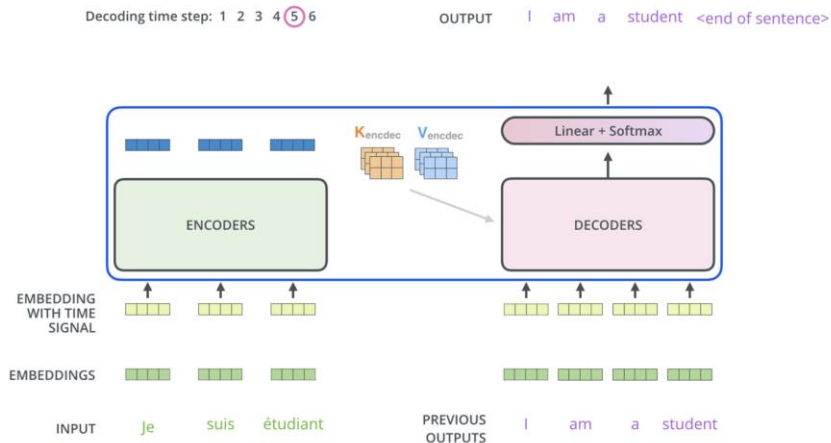
Positional Encoding

- ⦿ Accounts for the ordering of the words in the input sequence
- ⦿ A vector is added to the embedding vector
 - Model learns the position of the word in the sequence, as well as the distance between different words in the sequence



The Decoder

- ◎ The inner workings of the decoder are very similar to the encoder
 - Extra layer: “**encoder-decoder attention**” layer where the encoder sends information to the decoder to help with attention and prediction
 - Creates its query matrix from the layer below
 - Takes the keys and values matrices from the output of the encoder
 - Self-attention within the decoder blocks is a little different
 - Only allowed to attend to earlier positions in the output sequence



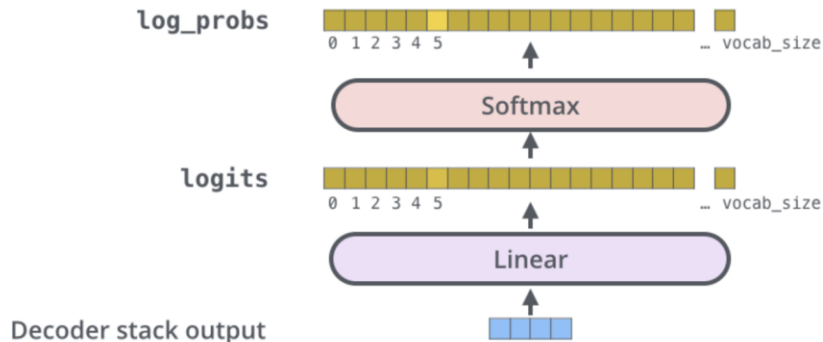
The Final Layer

- ◎ The decoder outputs a vector that is transformed into a larger vector by a dense (linear) network
 - Output is called a logits vector
 - Each cell is the score of a particular word
- ◎ Pass the logits vector through a softmax function to get probabilities for each word

The cell with the highest probability is chosen

Which word in our vocabulary is associated with this index?

Get the index of the cell with the highest value (argmax)



Text Generation API

You can play with text generation with this [API](#)

Hugging Face Search models, datasets, users...

Models Datasets Pricing Resources **We're hiring!** Log In Sign Up

distilgpt2

Text Generation PyTorch TensorFlow TF Lite Rust openwebtext en apache-2.0 gpt2 lm-head causal-lm exbert

Model card Files and versions Use Accelerated Inference Use in transformers

DistilGPT2

DistilGPT2 English language model pretrained with the supervision of **GPT2** (the smallest version of GPT2) on **OpenWebTextCorpus**, a reproduction of OpenAI's WebText dataset. The model has 6 layers, 768 dimension and 12 heads, totalizing 82M parameters (compared to 124M parameters for GPT2). On average, DistilGPT2 is two times faster than GPT2.

On the **WikiText-103** benchmark, GPT2 reaches a perplexity on the test set of 16.3 compared to 21.1 for DistilGPT2 (after fine-tuning on the train set).

We encourage to check **GPT2** to know more about usage, limitations and potential biases.

Visualize in **exBERT**

Downloads last month
72,267

Hosted inference API


Text Generation

The semester is almost over and Compute

Computation time on cpu: 0.650 s

The semester is almost over and I expect to get more new teachers here, they're just learning the basics and a little bit more programming on my hands.>

</> JSON Output Maximize

The background of the slide is a light gray network pattern. It consists of numerous small circles, some of which are solid gray and others are hollow with a gray outline. These circles are interconnected by a web of thin, light gray lines, creating a complex, organic-looking structure that resembles a neural network or a data graph.

Reinforcement Learning

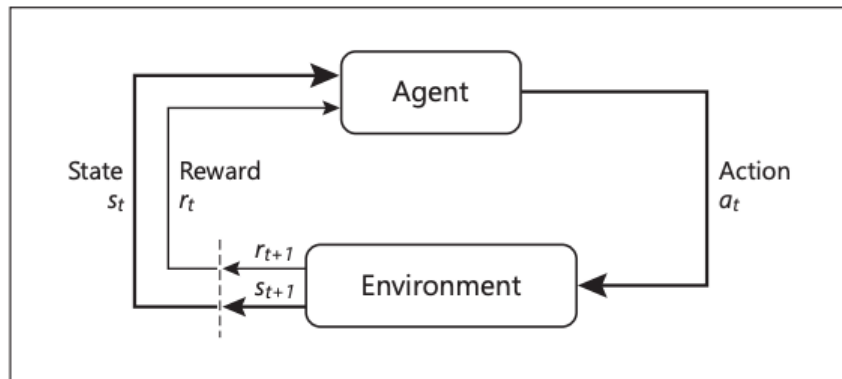
Reinforcement Learning (RL)

- ◎ A subfield of AI that provides tools to optimize **sequences of decisions** for **long-term outcomes**
- ◎ Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal
- ◎ The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them
 - Lots of interacting with environment
 - Lots of trial and error
 - A decision will affect not only the next action, but actions after that as well

RL

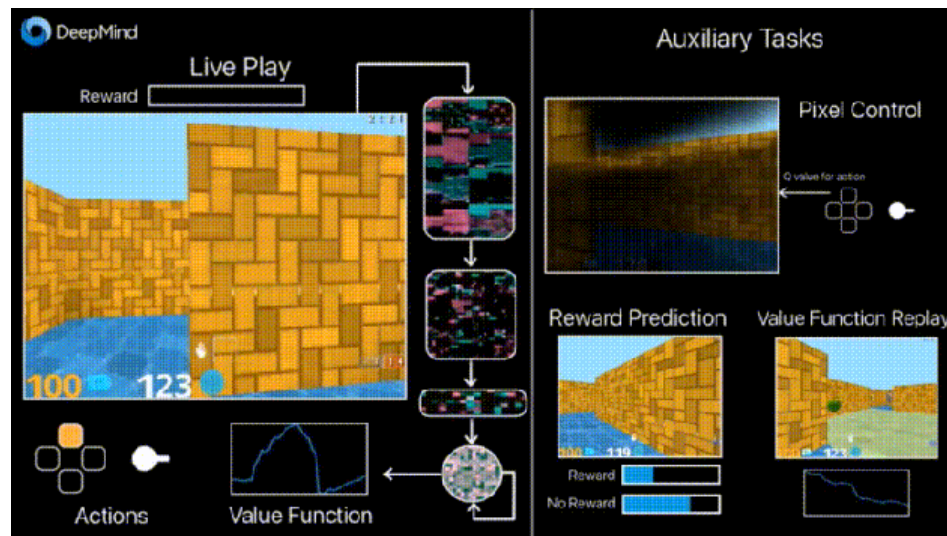
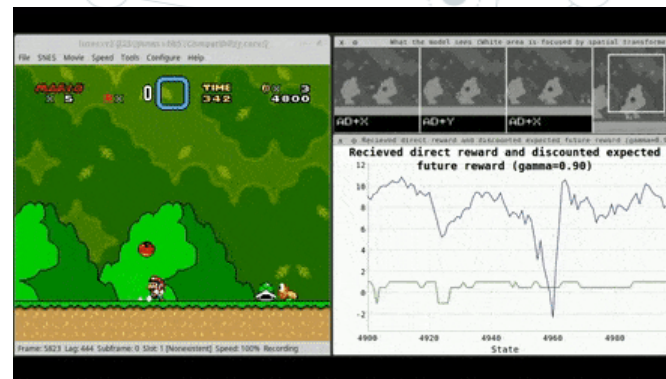
◎ Framework

- Trial and error search, and delayed reward
- Input: sequences of interactions (called **histories**) between the decision maker and their environment
- At every decision point, the RL algorithm chooses an **action** according to its policy and receives new observations and immediate outcomes (often called **rewards**)



RL

- Has been really popular with games
 - AlphaGo is better than the best Go players in the world



RL in Healthcare

- ◎ Still a recent method being applied in healthcare contexts
- ◎ Examples
 - Optimizing antiretroviral therapy in HIV
 - Tailoring antiepilepsy drugs for seizure control
 - Determining the best approach to managing sepsis
- ◎ Rather than a one-time prediction, RL affects a patient's future health and future treatment options
 - Long-term effects are more difficult to estimate

<https://www.nature.com/articles/s41591-018-0310-5>

<https://towardsdatascience.com/a-review-of-recent-reinforcement-learning-applications-to-healthcare-1f8357600407>

Sepsis Example

- ◎ There is wide variability in the way clinicians make decisions about sepsis management
 - Can RL help with this?
- ◎ **History:** may include a patient's vital signs and laboratory tests
- ◎ **Actions:** all the treatments available to the clinician, including medications and interventions
- ◎ **Rewards:** require clinician input - they should represent the achievement of desirable tasks, such as stabilization of vital signs or survival at the end of the stay
 - Short-term: liberation from mechanical ventilation
 - Long-term: prevention of permanent organ damage
 - <https://arxiv.org/pdf/1711.09602.pdf>

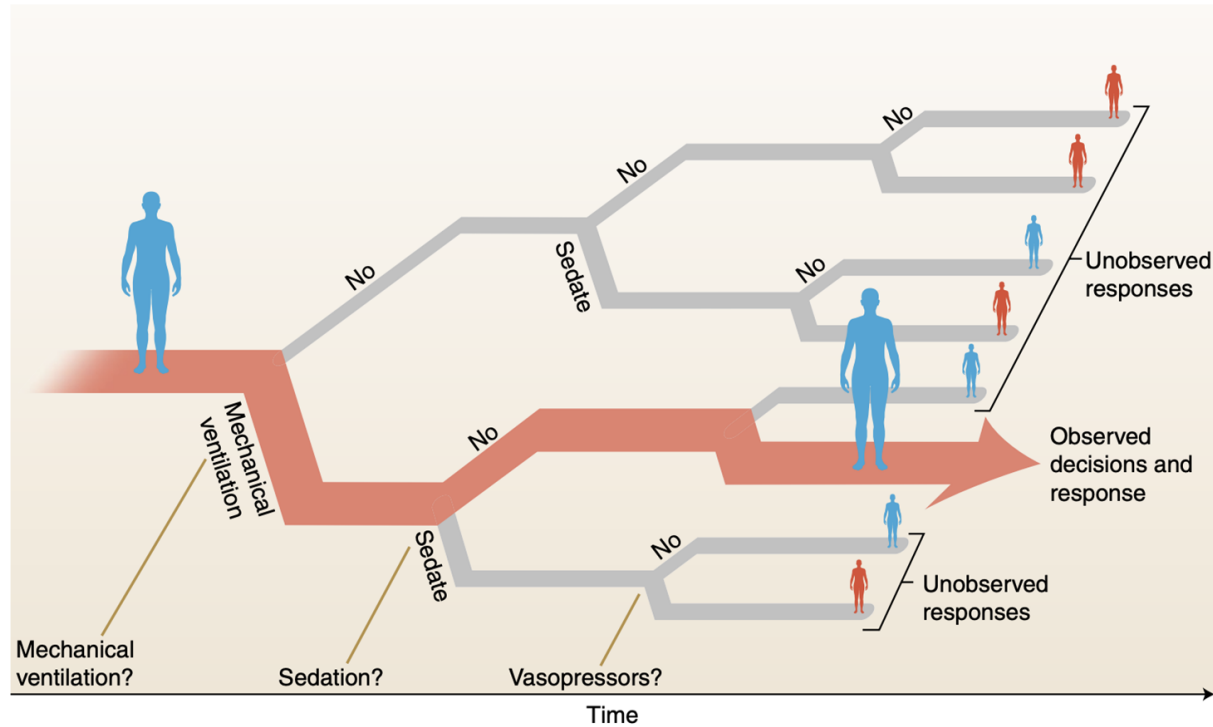


Fig. 1 | Sequential decision-making tasks. To perform sequential decision making, such as for sepsis management, treatment-effect estimation must be solved at a grand scale—every possible combination of interventions could be considered to find an optimal treatment policy. The diagram shows the scale of such a problem with only three distinct decisions. Blue and red people denote positive and negative outcomes, respectively. Credit: Debbie Maizels/Springer Nature

Challenges

- ◎ We only observe one set of actions and rewards for each patient
 - We can't keep trying different combinations of actions to optimize a reward - forced to use previous observational data, called “off-policy” learning
- ◎ We don't observe everything going on in the body
 - We also don't observe the values we do record (blood pressure, etc.) at every time step (dynamic data)
- ◎ It's difficult to find a reward function
 - How do we balance short and long-term rewards?

Need a ton of data, which is difficult to come by

Questions to Consider

- ◎ Is the AI given access to all variables that influence decision making?
 - Typically no because of confounding variables
 - Can lead to confounding in the short term and long term
- ◎ How big is your effective sample size?
 - Most approaches for evaluating RL policies from observational data weigh each patient's history on the basis of whether the clinician decisions match the decisions of the policy proposed by the RL algorithm
 - The reliability (variance) of the treatment-quality estimate depends on the number of patient histories for which the proposed and observed treatment policies agree—a quantity known as the effective sample size
 - The possibilities for mismatch between the actual decision and the proposed decision grow with the number of decisions in the patient's history, and thus RL evaluation is especially prone to having small effective sample sizes

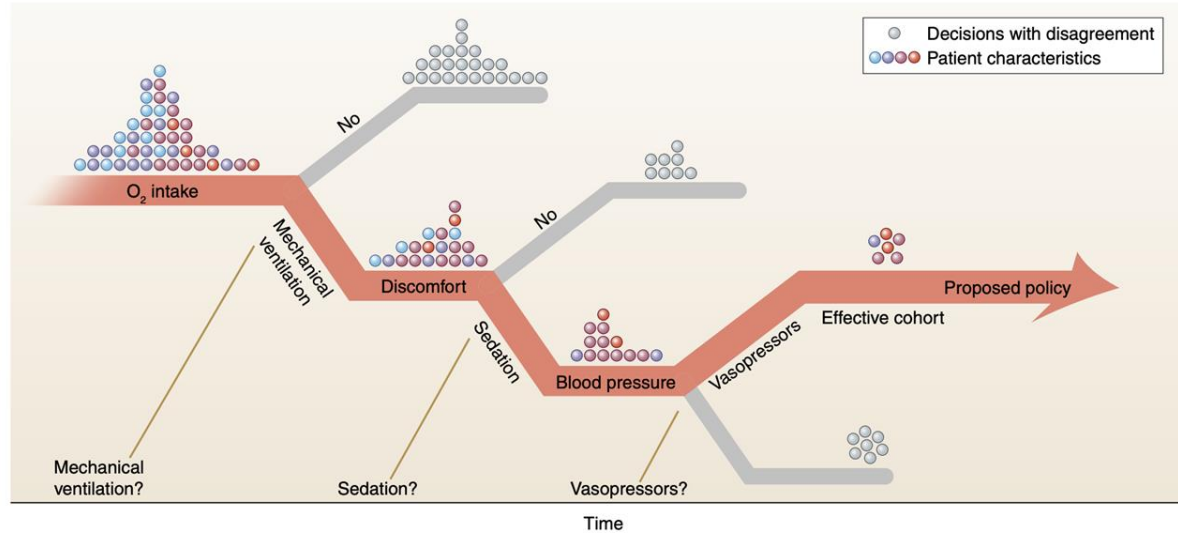


Fig. 2 | Effective sample size in off-policy evaluation. Each dot represents a single patient at each stage of treatment, and its color indicates the patient's characteristics. The more decisions that are performed in sequence, the likelier it is that a new policy disagrees with the one that was learned from. Gray decision points indicate disagreement. Use of only samples for which the old policy agrees with the new results in a small effective sample size and a biased cohort, as illustrated by the difference in color distribution in the original and final cohort. Credit: Debbie Maizels/Springer Nature

Questions to Consider

- ◎ Will the AI behave prospectively as intended?
 - Errors in problem formulation or data processing can lead to poor decisions
 - Simplistic reward functions may neglect long-term effects for meaningless gains: for example, rewarding only blood pressure targets may result in an AI that causes long-term harm by excessive dosing of vasopressors
 - Errors in data recording or preprocessing may introduce errors in the reward signal, misleading the RL algorithm
 - The learned policy may not work well at a different hospital or even in the same hospital a year later if treatment standards shift

RL in Medicine

- ◎ RL in medicine seems promising, but very difficult
- ◎ May help guide clinicians in treatment decisions based on all of a patient's history and not their immediate symptoms/responses to treatment
- ◎ Really nice [review paper of RL in medicine](#)

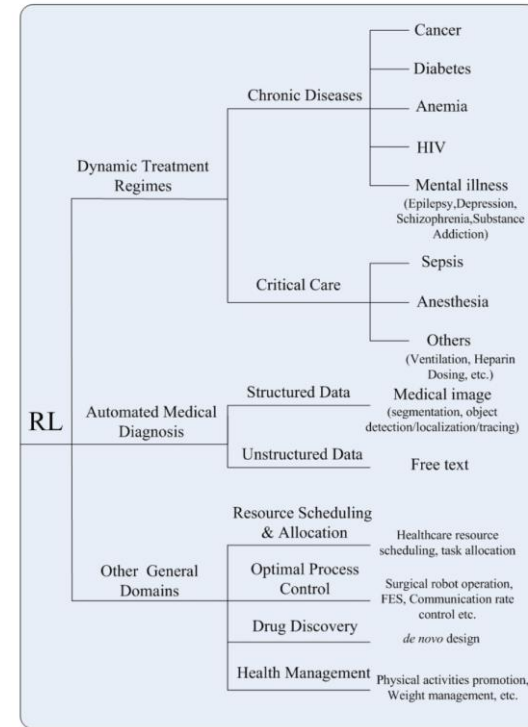


Fig. 2. The outline of application domains of RL in healthcare.