

Transformers and Enformer

Michael Schatz
October 23, 2024
Applied Comparative Genomics



Class Schedule

M	Oct 14	Epigenome	Project Proposal Assigned
W	Oct 16	Single cell	
M	Oct 21	Transformers	Assignment 5 Assigned
W	Oct 23	Enformer	
M	Oct 28	DL in Genomics	Preliminary Report Assigned
W	Oct 30	Midterm Review	
M	Nov 4	Midterm!	
W	Nov 6	Disease Genomics	
M	Nov 11	Metagenomics	Final Report Assigned
W	Nov 13	No Class (BIODATA24)	
M	Nov 18	Cancer Genomics	
W	Nov 20	Project Presentation 1	
M	Nov 25	Thanksgiving Break	
W	Nov 27	Thanksgiving Break	
M	Dec 2	Project Presentation 2	
W	Dec 4	Project Presentation 3	
M	Dec 16	Project Report Due	



Project Proposal

Due: Monday Oct 21, 2024 by 11:59pm

Project Proposal

Assignment Date: Monday October 14, 2024
Due Date: Monday, October 21 2024 @ 11:59pm

Review the [Project Ideas](#) page

Work solo or form a team for your class project of no more than 3 people.

The proposal should have the following components:

- Name of your team
- List of team members and email addresses
- Short title for your proposal
- 1 paragraph description of what you hope to do and how you will do it
- References to 2 to 3 relevant papers
- References/URLs to datasets that you will be studying (Note you can also use simulated data)
- Please add a note if you need me to sponsor you for a MARCC account (high RAM, GPUs, many cores, etc)
- Please add a note if you need me to sponsor you for an AnVIL cloud billing account (dynamic resources)

Submit the proposal as a 1 to 2 page PDF on GradeScope (each team should submit one proposal and tag all people in the team). After submitting your proposal, I will provide feedback. If necessary, we can schedule a time to discuss your proposal, especially to ensure you have access to the data that you need. The sooner that you submit your proposal, the sooner we can schedule the meeting.

Later, you will present your project in class during the last week of class. You will also submit a written report (5–7 pages) of your project, formatting as a Bioinformatics article (Intro, Methods, Results, Discussion, References). Word and LaTeX templates are available at https://academic.oup.com/bioinformatics/pages/submission_online

Please use Piazza to coordinate proposal plans!

<https://github.com/schatzlab/appliedgenomics2024/blob/main/project/proposal.md>

Check Piazza for questions!

Assignment 5

Due: Monday Oct 28, 2024 by 11:59pm

Assignment 5

First, run the below cells to ensure you install the required dependencies

```
In [ ]: !pip install kipoiseq==0.5.2
        !pip install logomaker

In [ ]: !pip install enformer-pytorch

In [ ]: from enformer_pytorch import from_pretrained
        from enformer_pytorch.finetune import HeadAdapterWrapper
        import kipoiseq
        from kipoiseq import Interval
        import matplotlib.pyplot as plt
        import numpy as np
        import pandas as pd
        import pyfaidx
        import seaborn as sns
        import torch
        import torch.nn as nn
        import tqdm
        from transformers import GPT2Model, GPT2Tokenizer
        import seaborn as sns
        import logomaker
        import torch.nn.functional as F
        from sklearn.preprocessing import StandardScaler, LabelEncoder
        from sklearn.decomposition import PCA
        import torch.optim as optim
        from torch.utils.data import DataLoader, TensorDataset
        from sklearn.metrics import classification_report

In [ ]: device = "cuda" if torch.cuda.is_available() else "cpu"
```

Question 1: Self-attention

Self-attention allows a model to weigh the importance of different tokens in a sequence relative to each other. This allows it to capture dependencies across the entire input, so the model can learn both local and long-range relationships. In this question, you will take a look at how self-attention works for natural language by using GPT-2 and visualizing the attention weights as a heatmap.

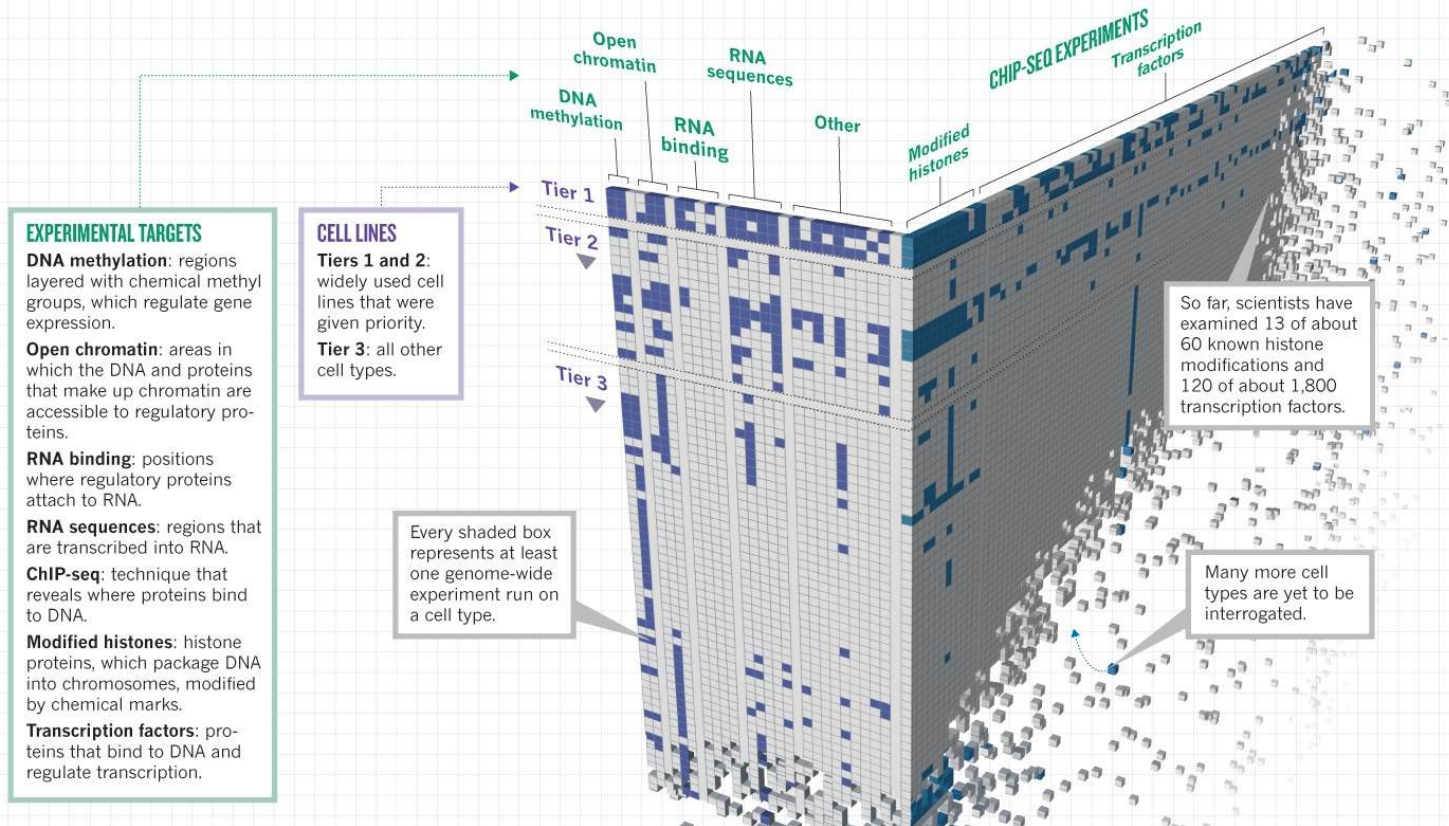
<https://schatz-lab.org/appliedgenomics2024/assignments/assignment5/>

Check Piazza for questions!

ENCODE Data Sets

MAKING A GENOME MANUAL

Scientists in the Encyclopedia of DNA Elements Consortium have applied 24 experiment types (across) to more than 150 cell lines (down) to assign functions to as many DNA regions as possible — but the project is still far from complete.



1,640 data sets total over 147 different cell types

Signal Integration

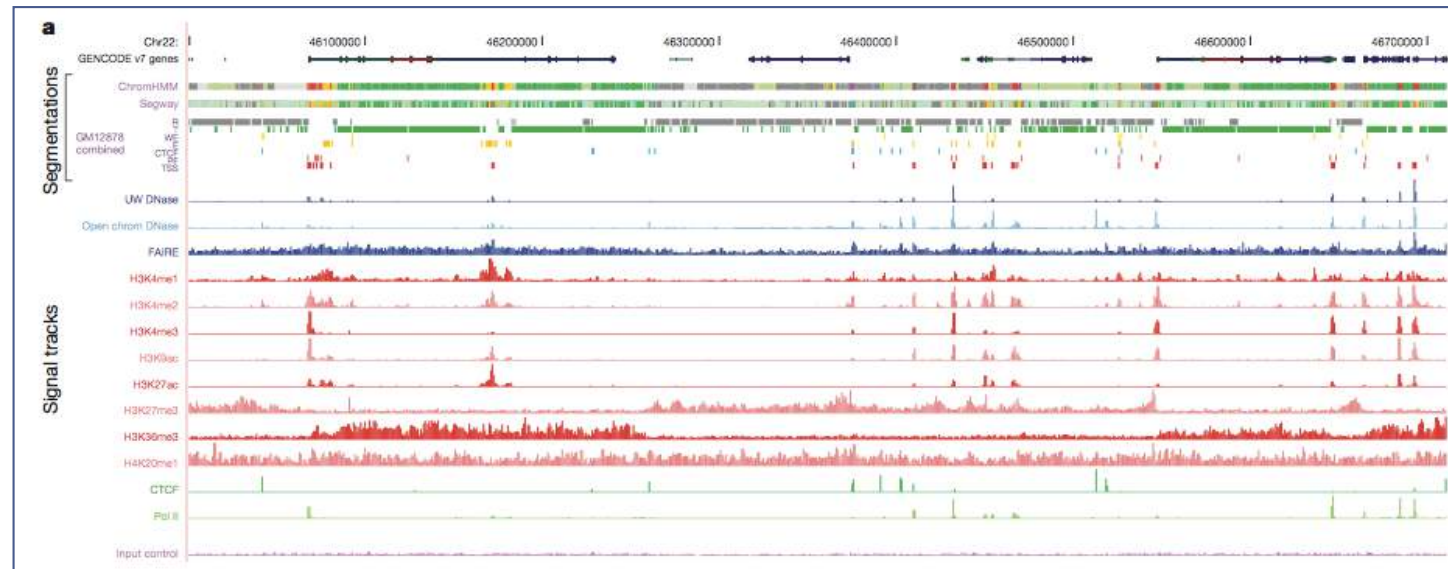


Table 3 | Summary of the combined state types

Label	Description	Details*	Colour
CTCF	CTCF-enriched element	Sites of CTCF signal lacking histone modifications, often associated with open chromatin. Many probably have a function in insulator assays, but because of the multifunctional nature of CTCF, we are conservative in our description. Also enriched for the cohesin components RAD21 and SMC3. CTCF is known to recruit the cohesin complex.	Turquoise
E	Predicted enhancer	Regions of open chromatin associated with H3K4me1 signal. Enriched for other enhancer-associated marks, including transcription factors known to act at enhancers. In enhancer assays, many of these (>50%) function as enhancers. A more conservative alternative would be cis-regulatory regions. Enriched for sites for the proteins encoded by <i>EP300</i> , <i>FOS</i> , <i>FOSL1</i> , <i>GATA2</i> , <i>HDAC8</i> , <i>JUNB</i> , <i>JUND</i> , <i>NFE2</i> , <i>SMARCA4</i> , <i>SMARCB1</i> , <i>SIRT6</i> and <i>TAL1</i> genes in K562 cells. Have nuclear and whole-cell RNA signal, particularly poly(A) - fraction.	Orange
PF	Predicted promoter flanking region	Regions that generally surround TSS segments (see below).	Light red
R	Predicted repressed or low-activity region	This is a merged state that includes H3K27me3 polycomb-enriched regions, along with regions that are silent in terms of observed signal for the input assays to the segmentations (low or no signal). They may have other signals (for example, RNA, not in the segmentation input data). Enriched for sites for the proteins encoded by <i>REST</i> and some other factors (for example, proteins encoded by <i>BRF2</i> , <i>CERPB</i> , <i>MAPK</i> , <i>TRIM28</i> , <i>ZNF274</i> and <i>SETDB1</i> genes in K562 cells).	Grey
TSS	Predicted promoter region including TSS	Found close to or overlapping GENCODE TSS sites. High precision/recall for TSSs. Enriched for H3K4me3. Sites of open chromatin. Enriched for transcription factors known to act close to promoters and polymerases Pol II and Pol III. Short RNAs are most enriched in these segments.	Bright red
T	Predicted transcribed region	Overlap gene bodies with H3K36me3 transcriptional elongation signal. Enriched for phosphorylated form of Pol II signal (elongating polymerase) and poly(A) ⁺ RNA, especially cytoplasmic.	Dark green
WE	Predicted weak enhancer or open chromatin cis-regulatory element	Similar to the E state, but weaker signals and weaker enrichments.	Yellow

- Use ChromHMM and Segway to Summarize the individual assays into 7 functional/regulatory states

Attention Is All You Need ("Transformers")

Michael Schatz
October 21, 2024
Applied Comparative Genomics

A Sequence Modeling Problem: Predict the Next Word

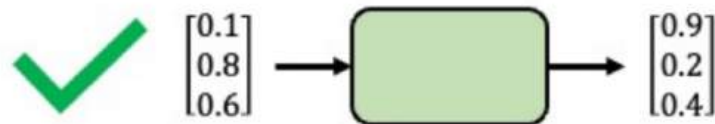
$f(\text{"This morning I took my cat for a"}) = ???$

given these words predict the next word

Representing Language to a Neural Network

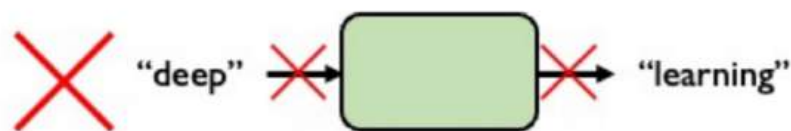


Neural networks cannot interpret words

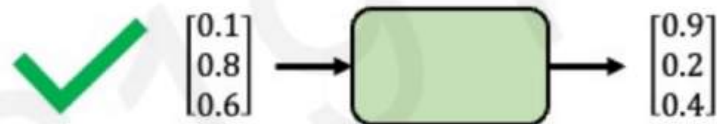


Neural networks require numerical inputs

Encoding Language for a Neural Network



Neural networks cannot interpret words



Neural networks require numerical inputs

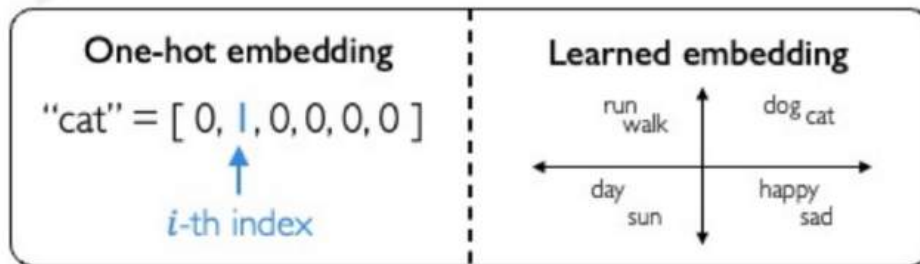
Embedding: transform indexes into a vector of fixed size.

this cat for
my took
a | walk
morning

1. Vocabulary:
Corpus of words

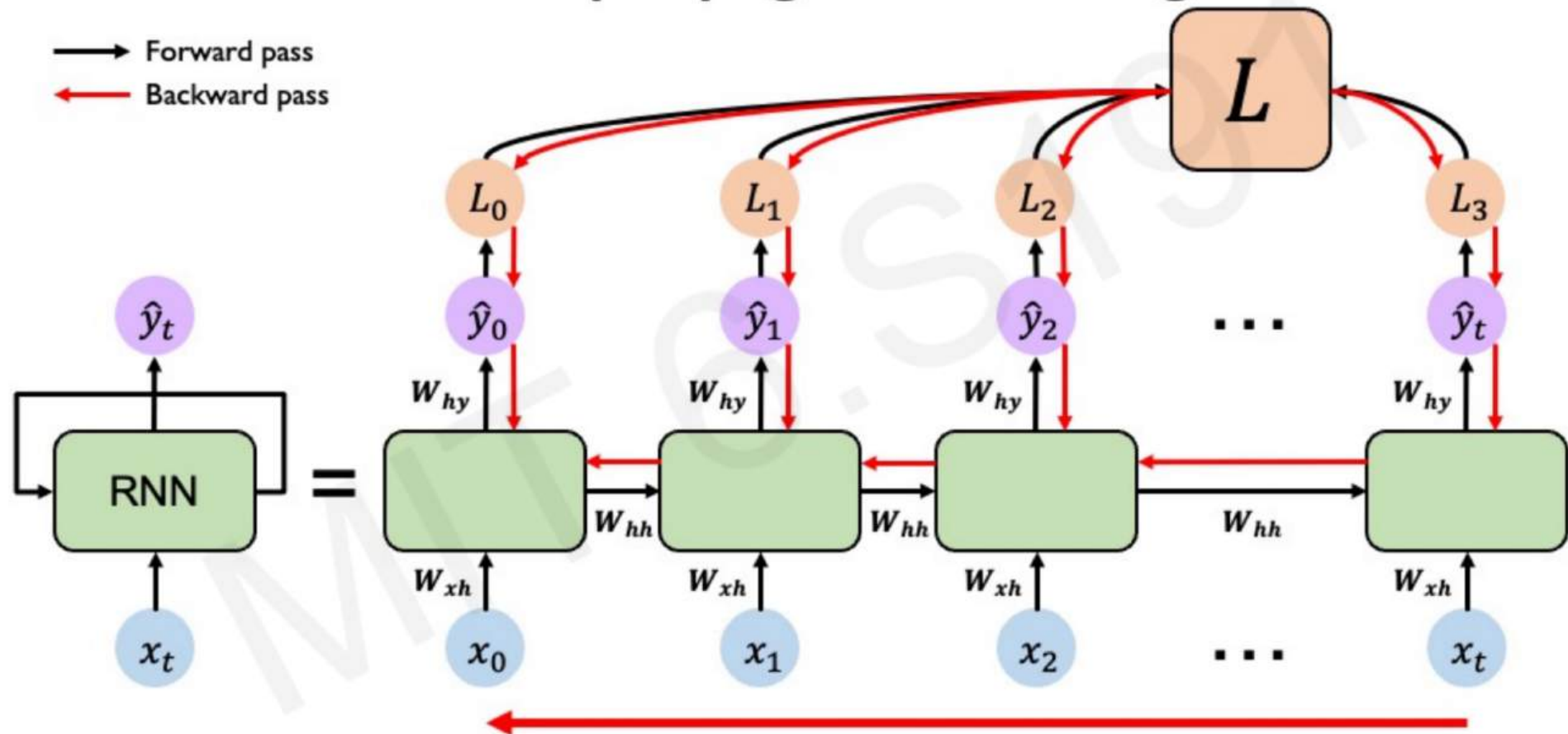
a → 1
cat → 2
... → ...
walk → N

2. Indexing:
Word to index



3. Embedding:
Index to fixed-sized vector

RNNs: Backpropagation Through Time



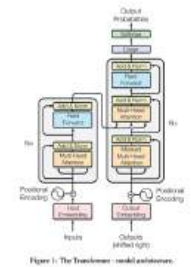
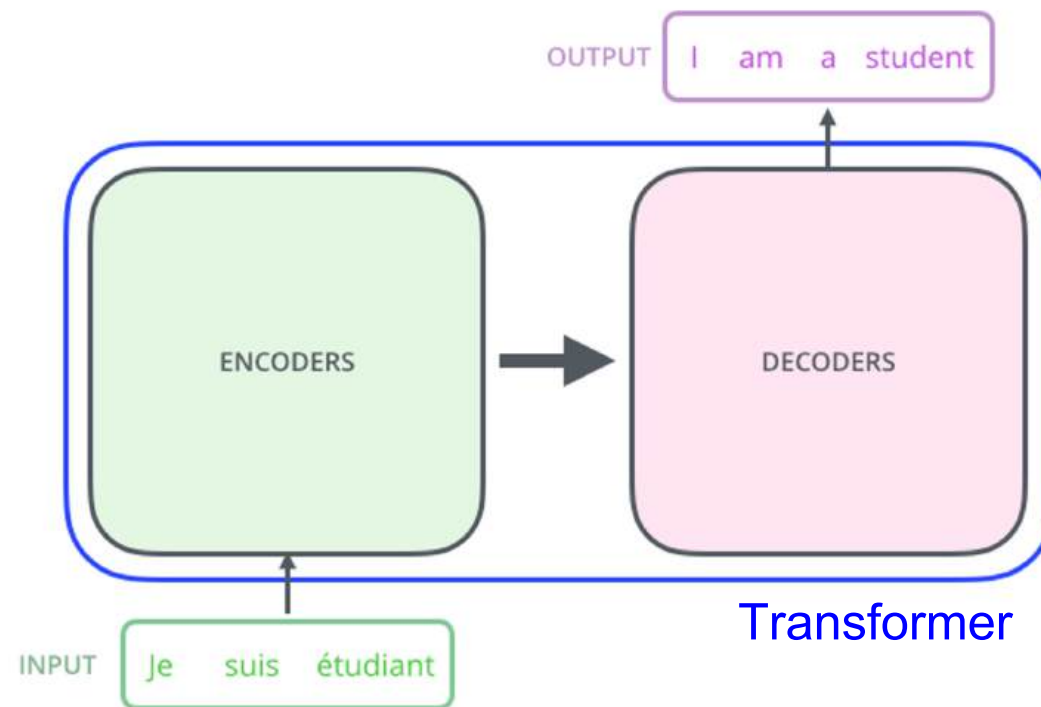
The application



Trained with a huge collection of pairs of sentences in french & english
Measure accuracy using “BLEU” scores to gold standard (higher is better)
Use a “reasonable” amount of GPU time for training (<1 week on 8 GPUs)

<https://jalammar.github.io/illustrated-transformer/>

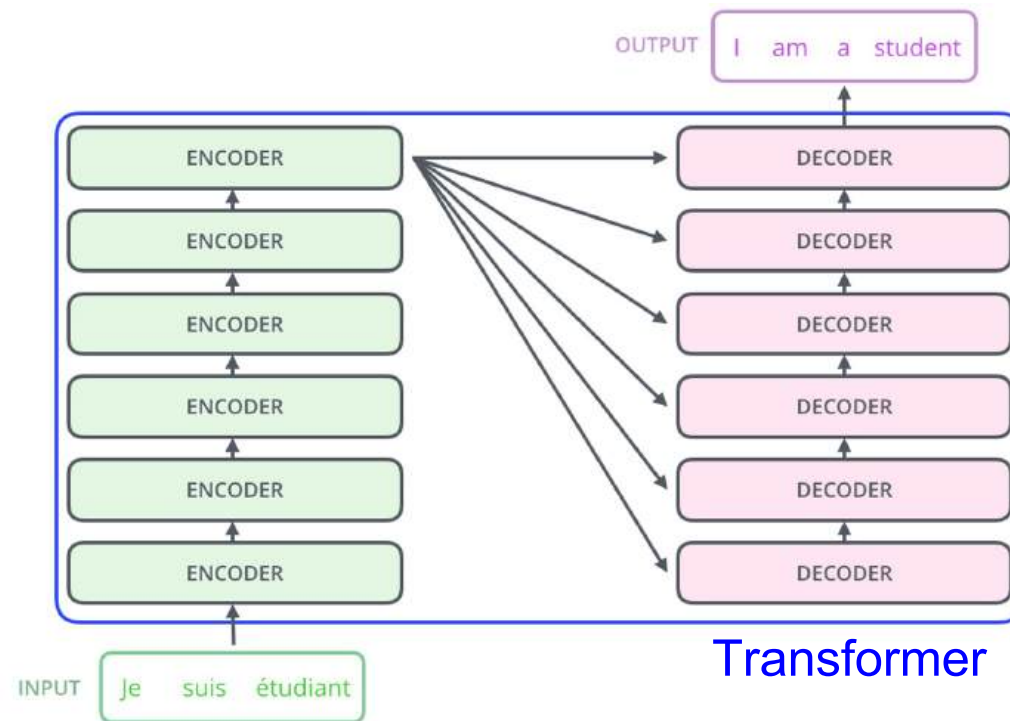
The architecture



Transformer is composed of an input encoder followed by a output decoder

<https://jalammar.github.io/illustrated-transformer/>

The architecture



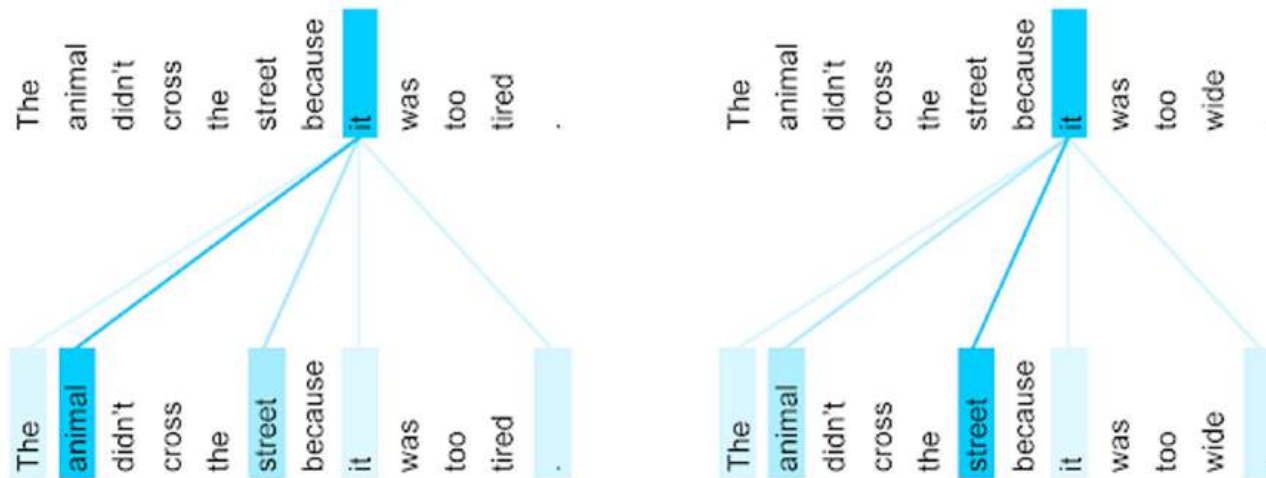
Encoder transforms sentences in language A into a highly abstract embedded space
Decoders progressively decodes highly abstract embedded space into language B

<https://jalammar.github.io/illustrated-transformer/>

Coreference resolution: self attention

The animal didn't cross the street because it was too tired.
L'animal n'a pas traversé la rue parce qu'il était trop fatigué.

The animal didn't cross the street because it was too wide.
L'animal n'a pas traversé la rue parce qu'elle était trop large.



<https://research.google/blog/transformer-a-novel-neural-network-architecture-for-language-understanding/>

Outline

1. Recap on ANNs and CNNs
2. The problem
3. Self-Attention
4. Transformers
5. Impact



What's important about this image? Where should my attention go?

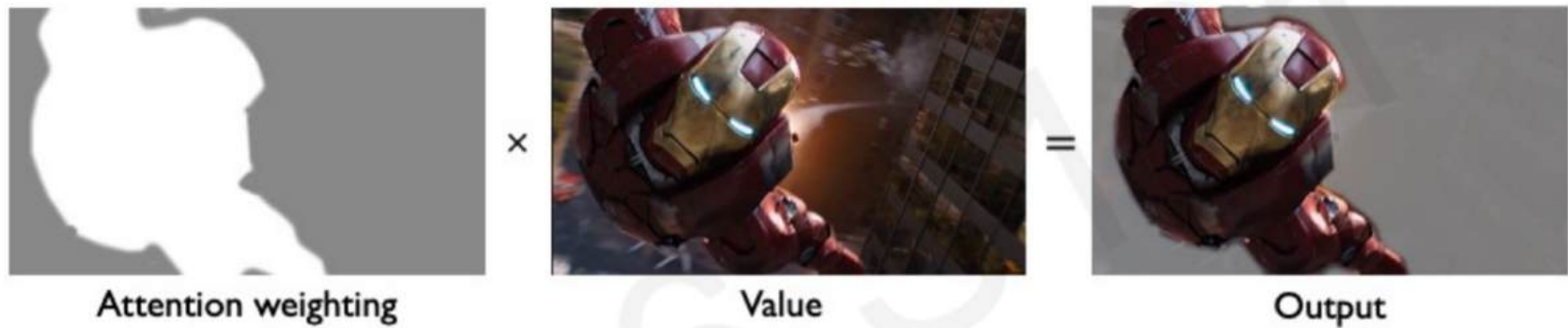
Intuition Behind Self-Attention

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!



Attention weighting identifies and focuses on the important parts of an image (input data)

- Note this is a self-attention since the inputs are the same, just the attention weighting mask (weights) are different

Applying Multiple Self-Attention Heads



Attention weighting

×



Value

=



Output



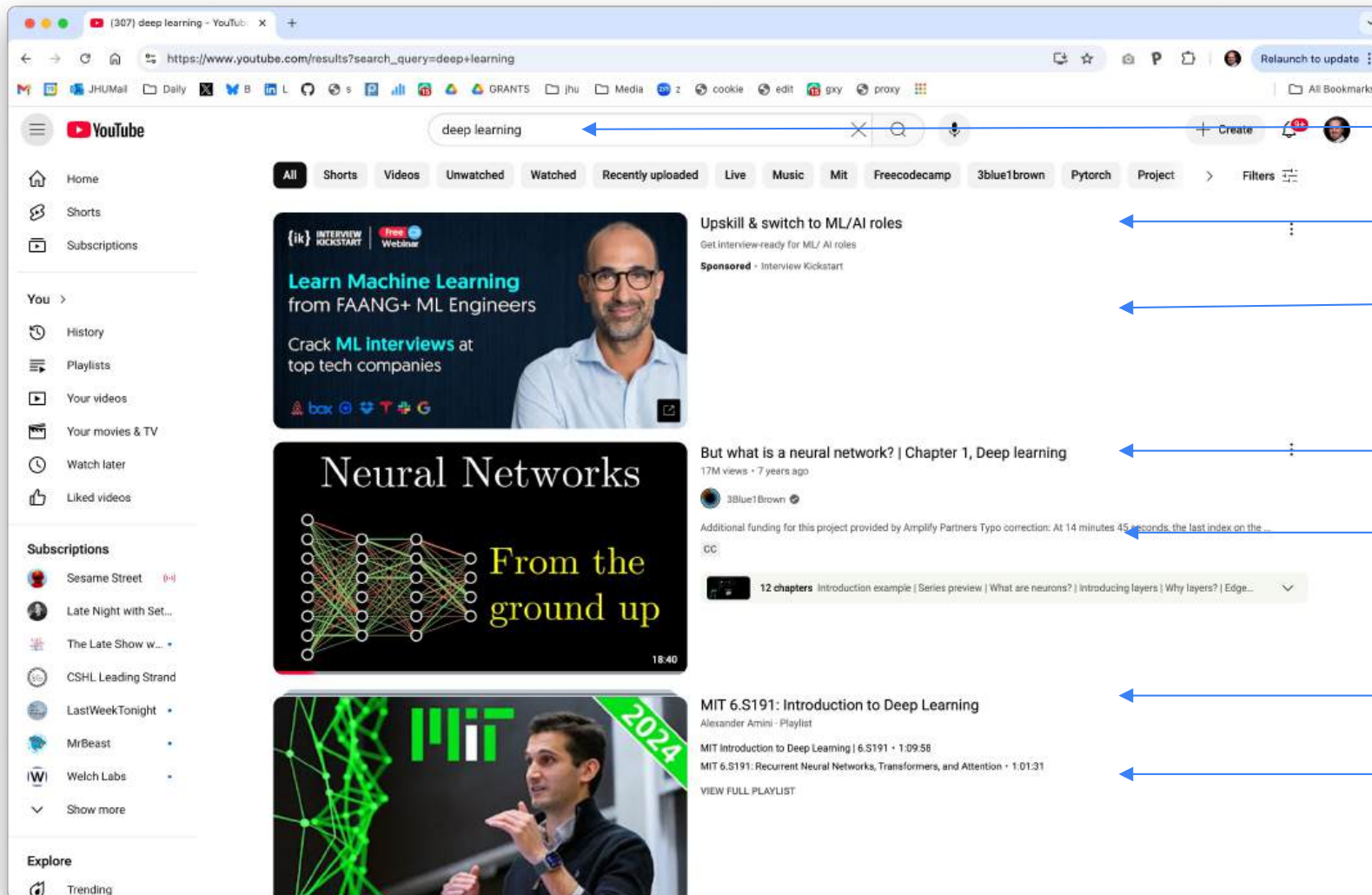
Output of attention head 1



Output of attention head 2



Output of attention head 3



Query

Key1

Value1

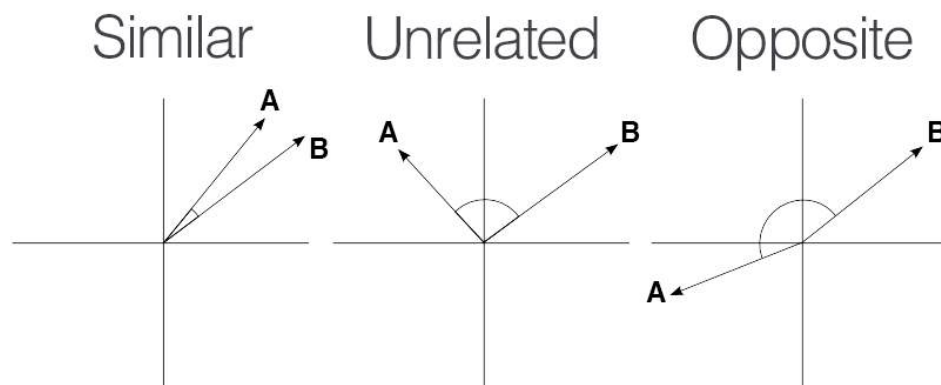
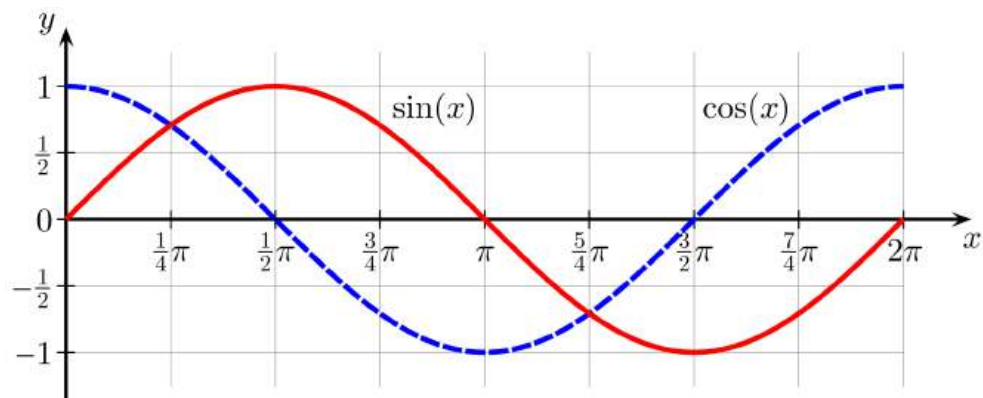
Key2

Value2

Key3

Value3

Cosine Similarity

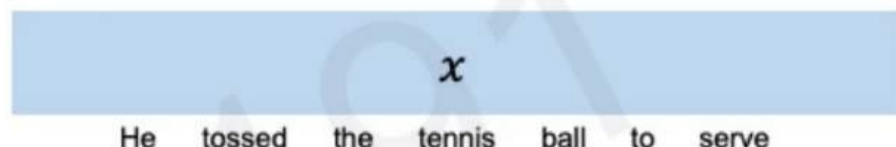


$$\text{cosine similarity} = S_C(A, B) := \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \cdot \sqrt{\sum_{i=1}^n B_i^2}},$$

where A_i and B_i are the i th **components** of vectors **A** and **B**, respectively.

Learning Self-Attention with Neural Networks

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



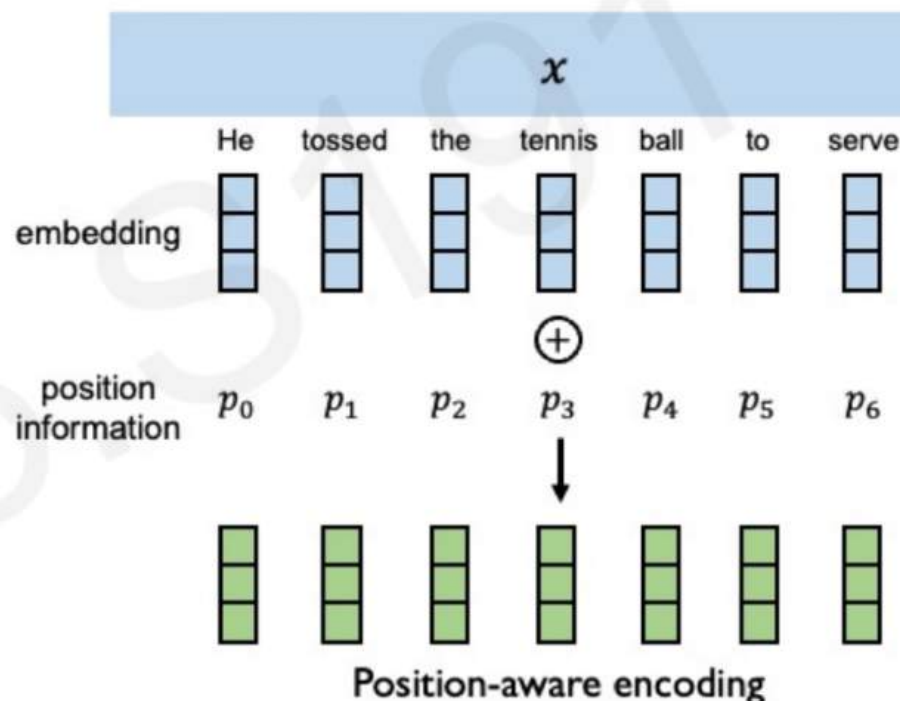
1. Encode **position** information
2. Extract query, key, value for search
3. Compute attention weighting
4. Extract features with high attention

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

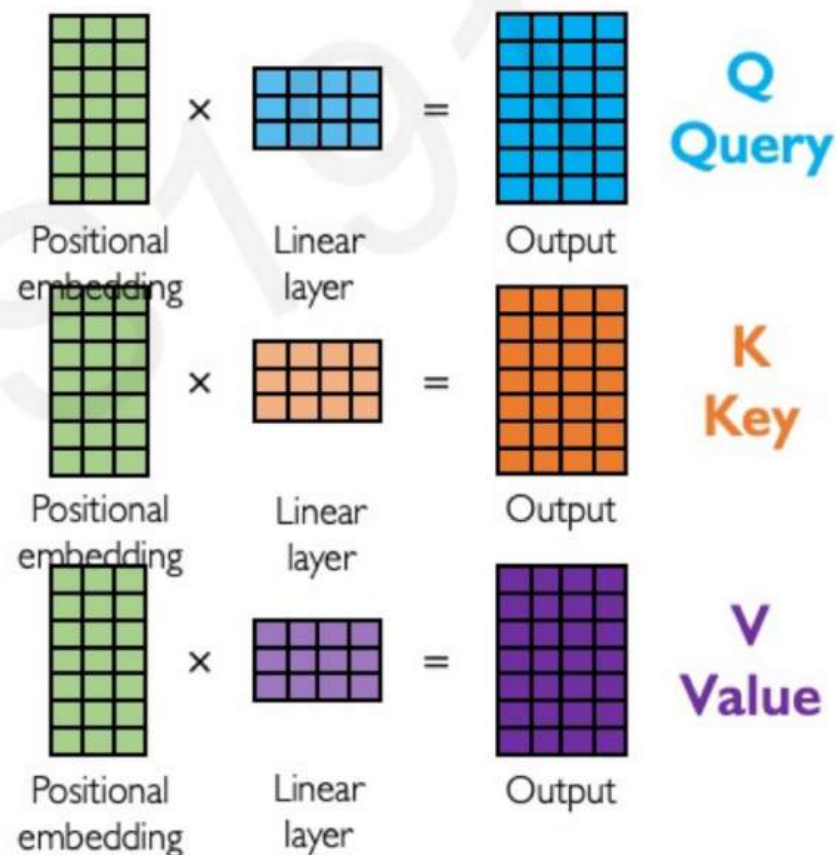


Data is fed in all at once! Need to encode position information to understand order.

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



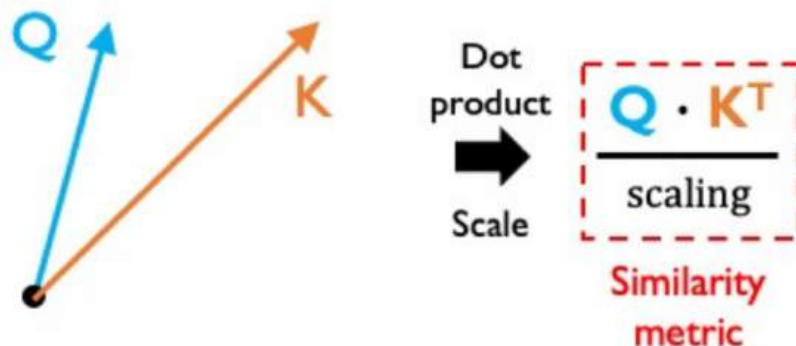
Learning Self-Attention with Neural Networks

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"

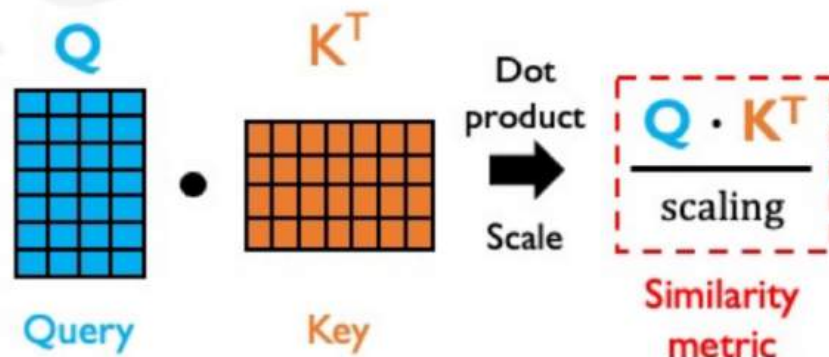
Learning Self-Attention with Neural Networks

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"

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?

	He	tossed	the	tennis	ball	to	serve
He	1	0	0	0	0	0	0
tossed	0	1	0	0	0	0	0
the	0	0	1	0	0	0	0
tennis	0	0	0	1	0	0	0
ball	0	0	0	0	1	0	0
to	0	0	0	0	0	1	0
serve	0	0	0	0	0	0	1

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

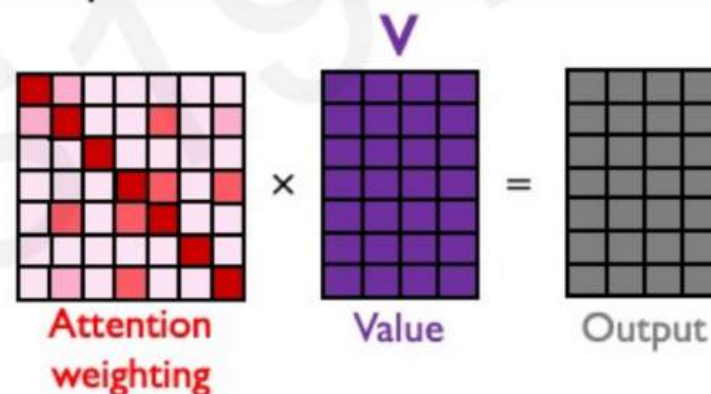
Attention weighting

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

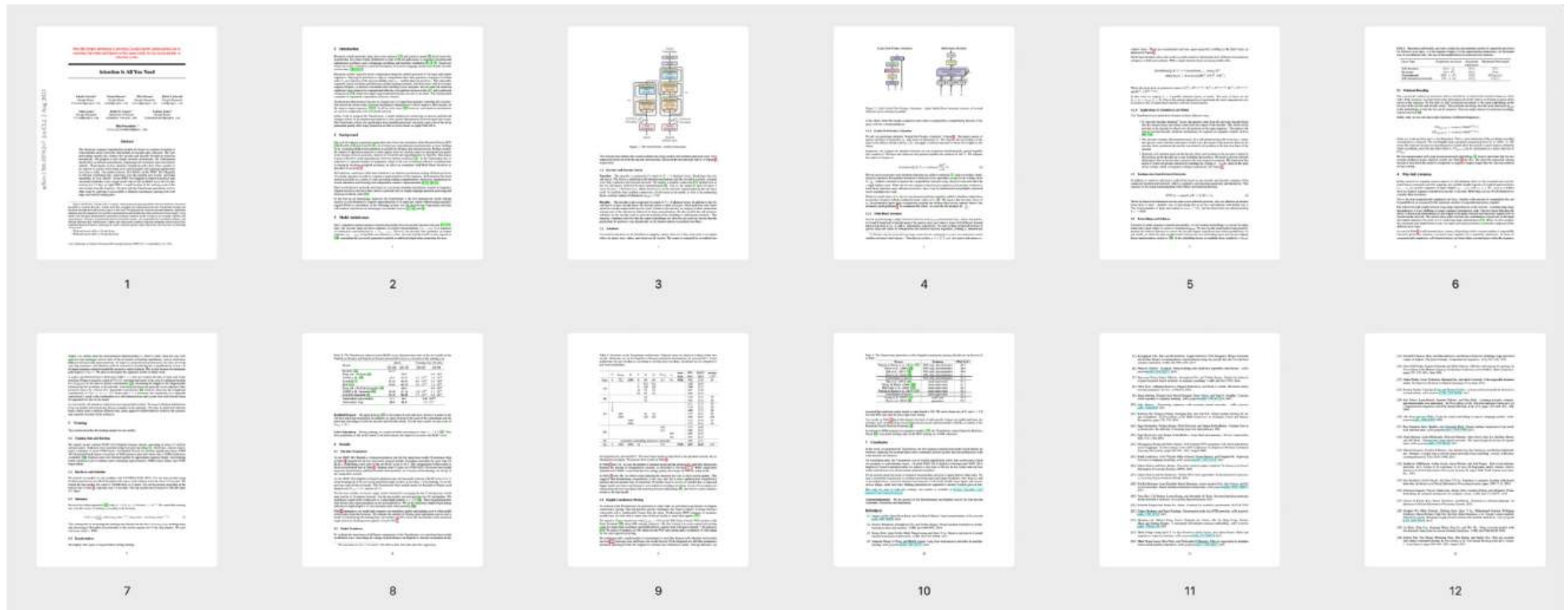


$$\text{softmax} \left(\frac{Q \cdot K^T}{\text{scaling}} \right) \cdot V = A(Q, K, V)$$

Outline

1. Recap on ANNs and CNNs
2. The problem
3. Self-Attention
4. Transformers
5. Impact

The paper



Vaswani et al (2017) arXiv:1706.03762

Transformer: Multi-headed attention to encode and decode

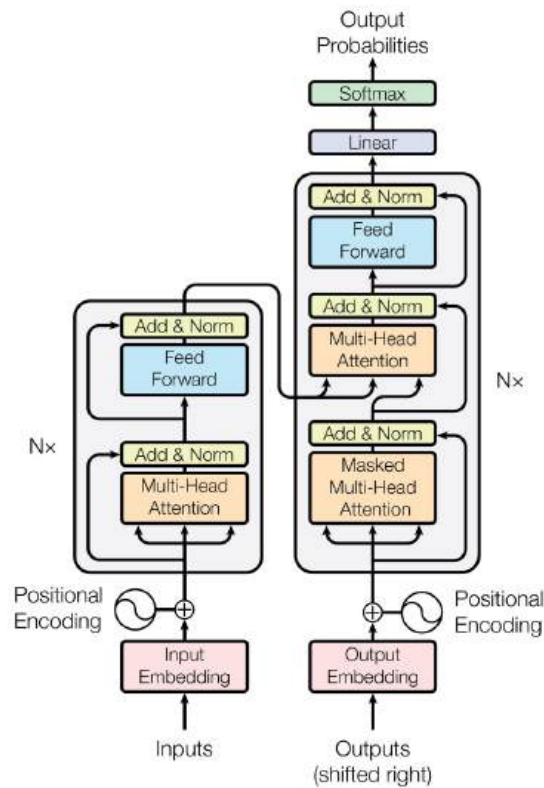
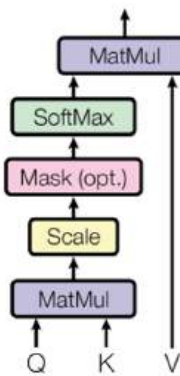


Figure 1: The Transformer - model architecture.

Scaled Dot-Product Attention



Multi-Head Attention

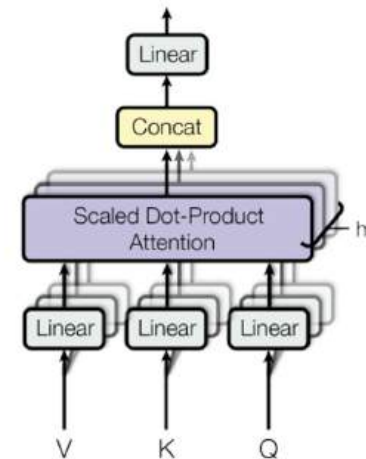


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

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

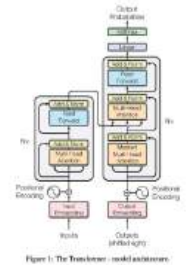
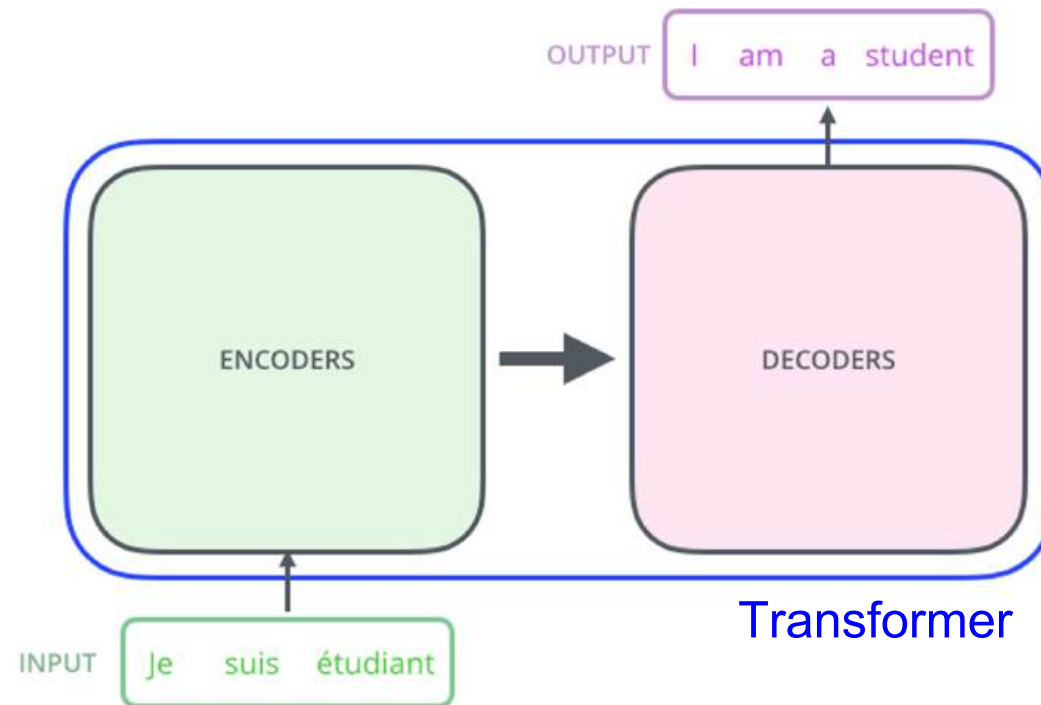
The application



Trained with a huge collection of pairs of sentences in french & english
Measure accuracy using “BLEU” scores to gold standard (higher is better)
Use a “reasonable” amount of GPU time for training (<1 week on 8 GPUs)

<https://jalammar.github.io/illustrated-transformer/>

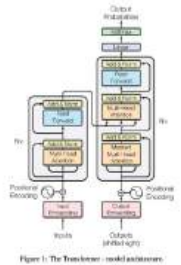
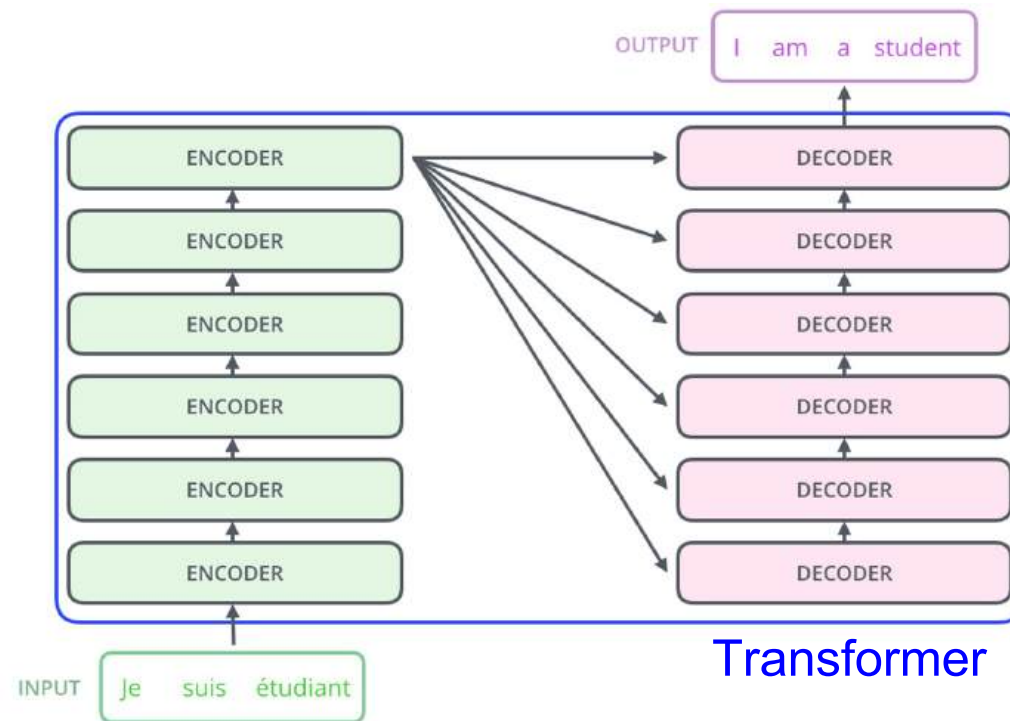
The architecture



Transformer is composed of an input encoder followed by a output decoder

<https://jalammar.github.io/illustrated-transformer/>

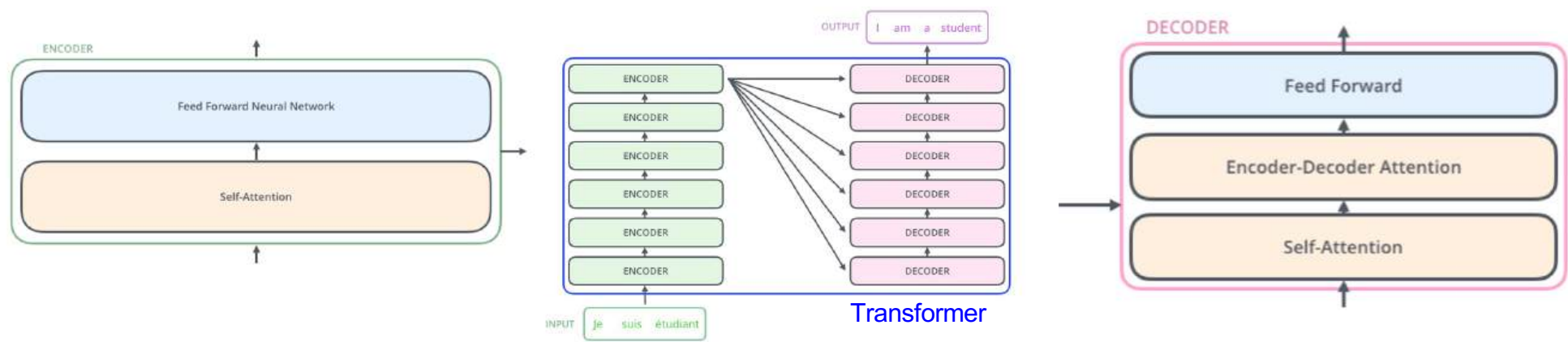
The architecture



Encoder transforms sentences in language A into a highly abstract embedded space
Decoders progressively decodes highly abstract embedded space into language B

<https://jalammar.github.io/illustrated-transformer/>

The architecture



Encoders & Decoder layers use self-attention and simple feed forward networks

- Attention is all you need!

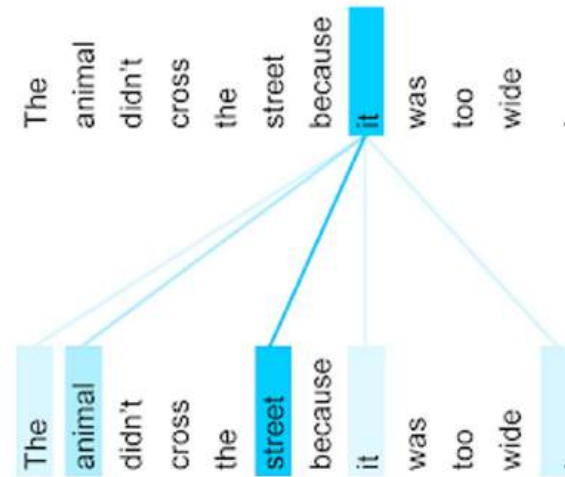
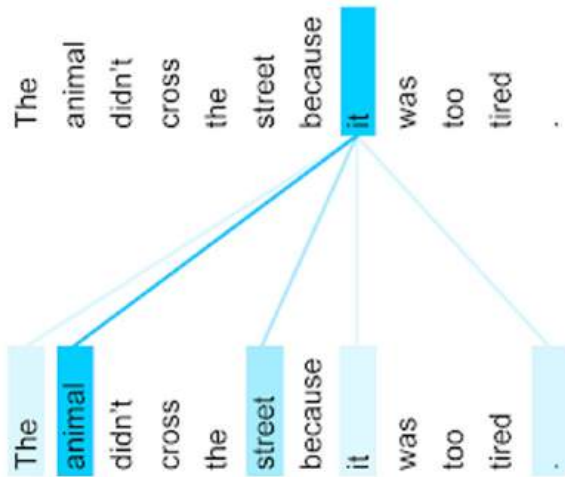
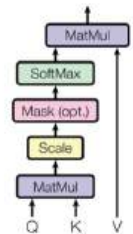
<https://jalammar.github.io/illustrated-transformer/>

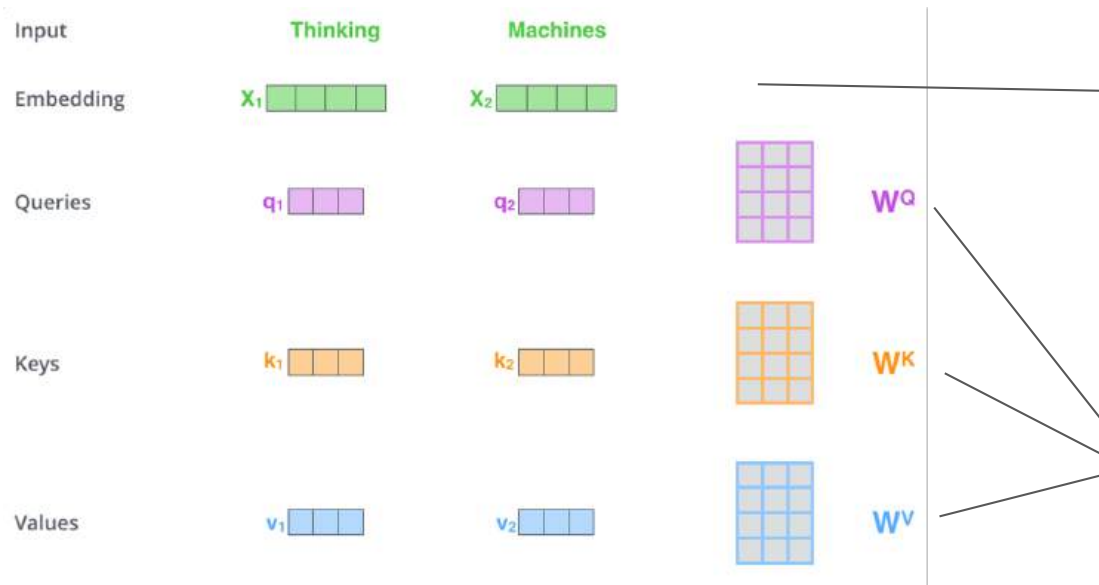
Self-attention

The animal didn't cross the street because **it** was too tired.
L'animal n'a pas traversé la rue parce qu'**il** était trop fatigué.

The animal didn't cross the street because **it** was too wide.
L'animal n'a pas traversé la rue parce qu'**elle** était trop large.

Scaled Dot-Product Attention





Embeddings are learned to project similar words into similar directions (cat & dog; king & queen)

- Similar to AlexNet!

Weight matrices ("masks") learned during training

Multiply embedded input value times W^Q , W^K , and W^V to compute q , k , v

Query (Q): Represents the current token's intention to "seek information" from other tokens in the sequence.

- It asks the question, "Which parts of the input sequence are relevant to me?"

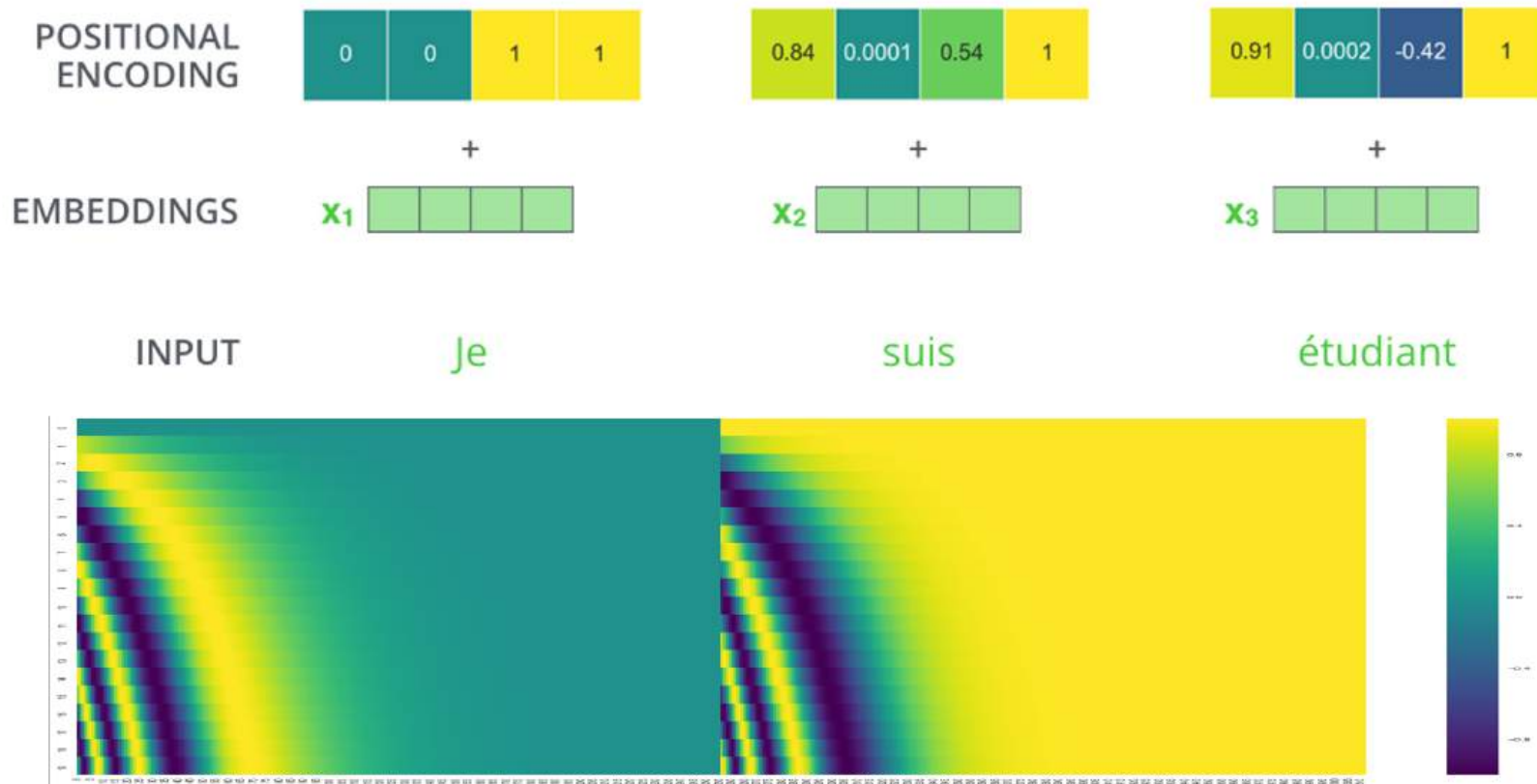
Key (K): Represents the content of the tokens against which the queries are compared.

- It acts like an "address" or "identifier" that the query uses to find relevant information.

Value (V): Contains the actual information that will be attended to. Once the attention mechanism determines which keys are most relevant to the query, the corresponding values are aggregated to form the final output.

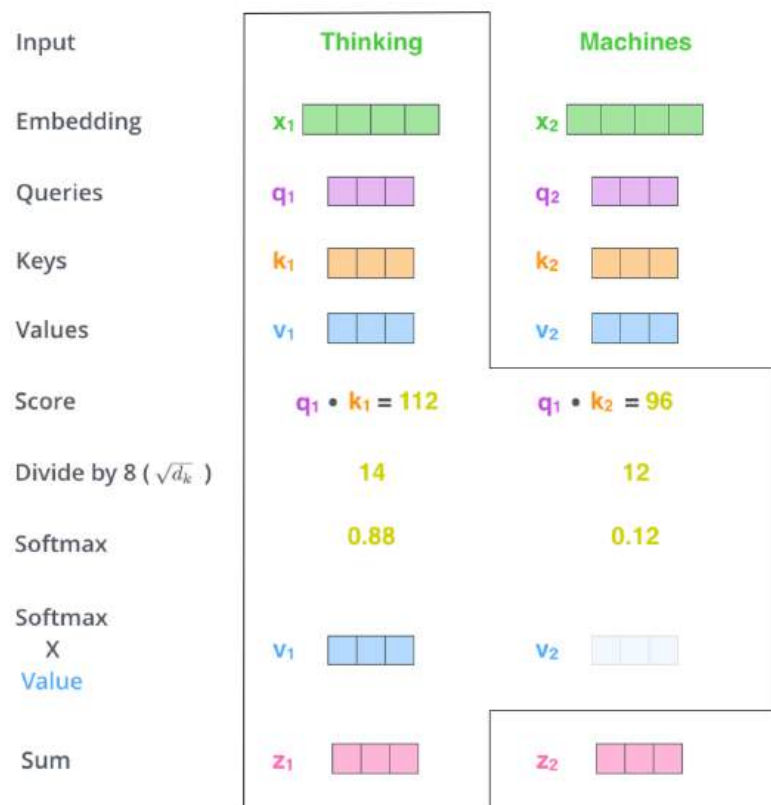
<https://jalammar.github.io/illustrated-transformer/>

Positional Encoding



<https://jalammar.github.io/illustrated-transformer/>

Attention computation

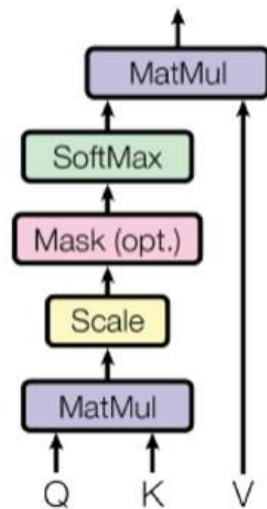


1. The score is calculated by taking the dot product (cosine similarity) of the **query vector** with the **key vector** of the respective word we're scoring.
2. The next steps are to divide the **scores** by 8 (the square root of the dimension of the key vectors used in the paper – 64). This leads to having more stable gradients.
3. Softmax normalizes the **scores** so they're all positive and add up to 1. This softmax **score** determines how much each word will be expressed at this position. ("Thinking" is mostly "Thinking" but a little "Machines")
4. Multiply each value vector by the softmax score to keep values of the word(s) we want to **focus** on, and drown-out irrelevant words
5. Sum up the weighted value vectors to produces the **output** of the self-attention layer at this position (for the first word). "Thinking" becomes "Thinking-machines"

<https://jalammar.github.io/illustrated-transformer/>

what is the softmax of the vector [14,12]

Scaled Dot-Product Attention



To compute the softmax of the vector [14, 12], we can follow these steps:

1. **Compute the exponential** of each element in the vector.
2. **Sum** all the exponentials computed in the first step.
3. **Divide** each exponential by the sum from the second step to get the softmax values.

Let's calculate this:

1. Exponentials:

$$e^{14} \approx 1202604.28$$

$$e^{12} \approx 162754.79$$

2. Sum of exponentials:

$$1202604.28 + 162754.79 = 1365359.07$$

3. Softmax values:

$$\text{Softmax of 14} = \frac{1202604.28}{1365359.07} \approx 0.881$$

$$\text{Softmax of 12} = \frac{162754.79}{1365359.07} \approx 0.119$$

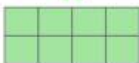
So, the softmax of the vector [14, 12] is approximately [0.881, 0.119].

Multi-headed Attention

- 1) This is our input sentence*
- 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

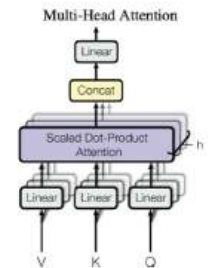
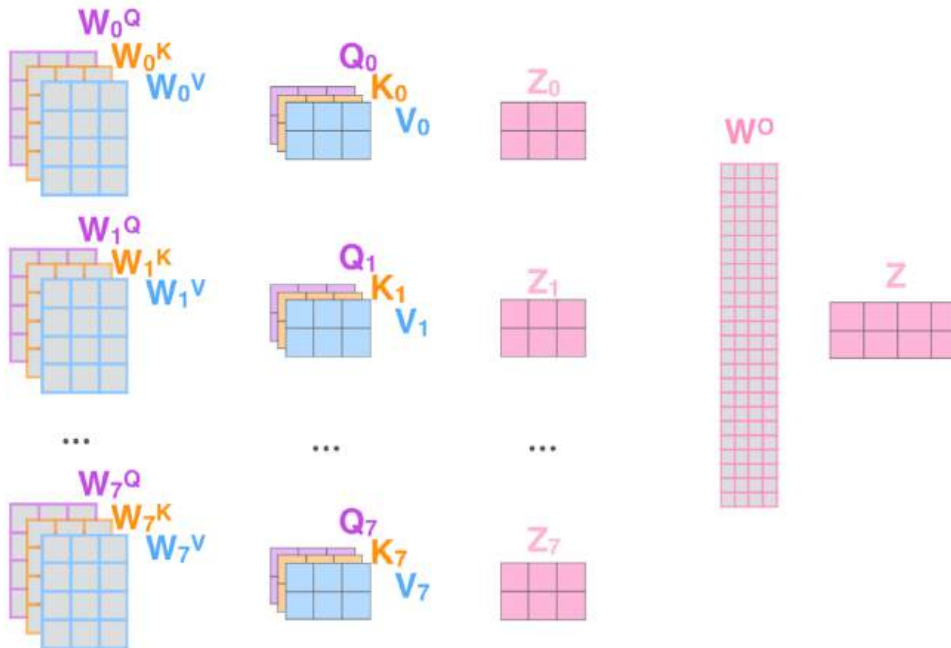
Thinking
Machines

X



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

R

Similar to having multiple convolutional filters

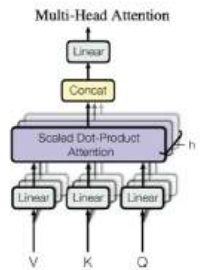
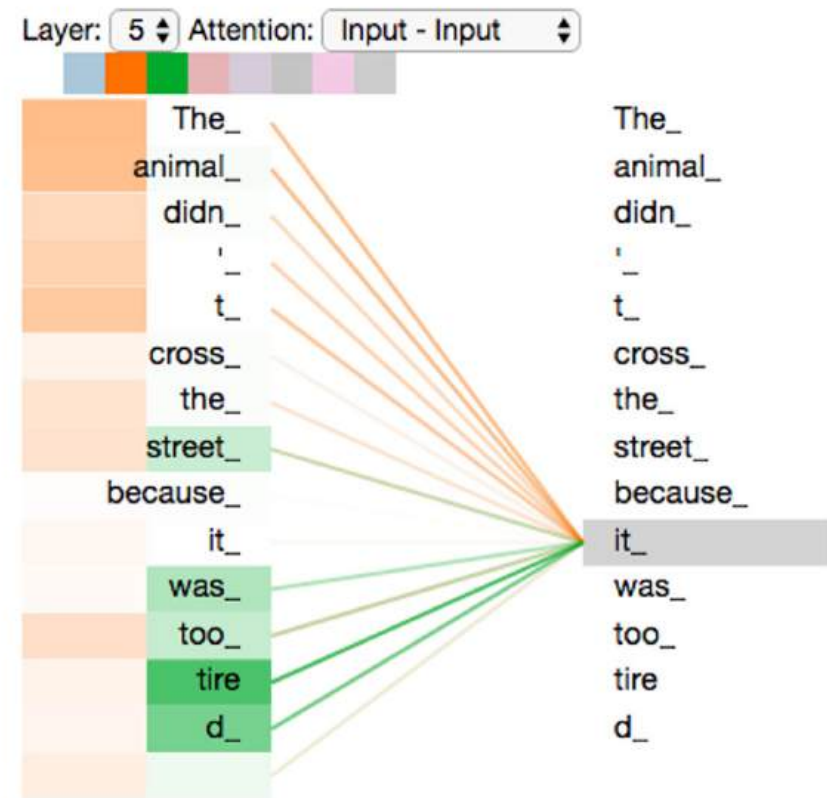
<https://jalammar.github.io/illustrated-transformer/>

Multi-headed Attention

The orange attention head “knows” “it” refers to animal

The green attention head “knows” “it” refers to tired

Added together we form a new vector representing the concept of “tired animal”



<https://jalammar.github.io/illustrated-transformer/>

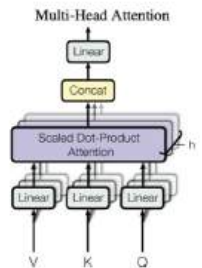
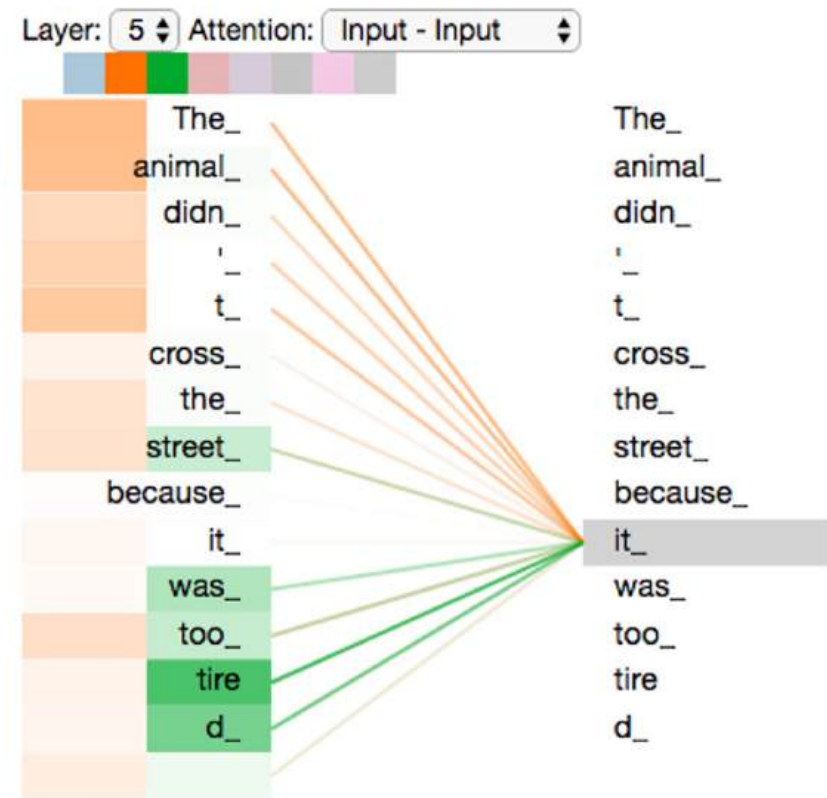
Multi-headed Attention

The orange attention head
“knows” “it” refers to
animal

The green attention head
“knows” “it” refers to tired

Added together we form a
new vector representing
the concept of “tired
animal”

* One head will be trained
to look at previous word



<https://jalammar.github.io/illustrated-transformer/>

Decoding

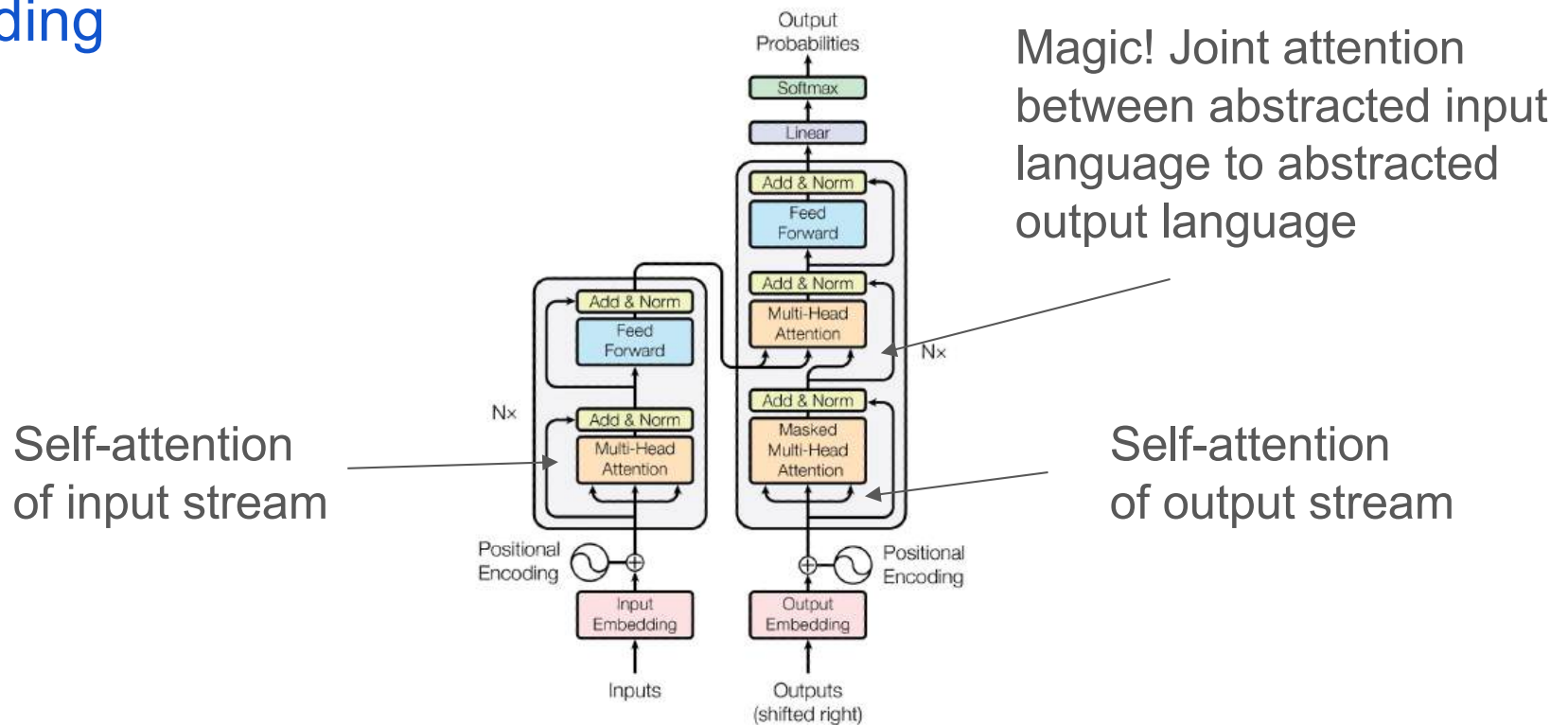
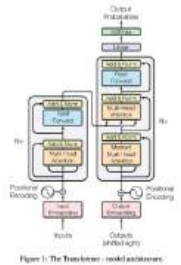
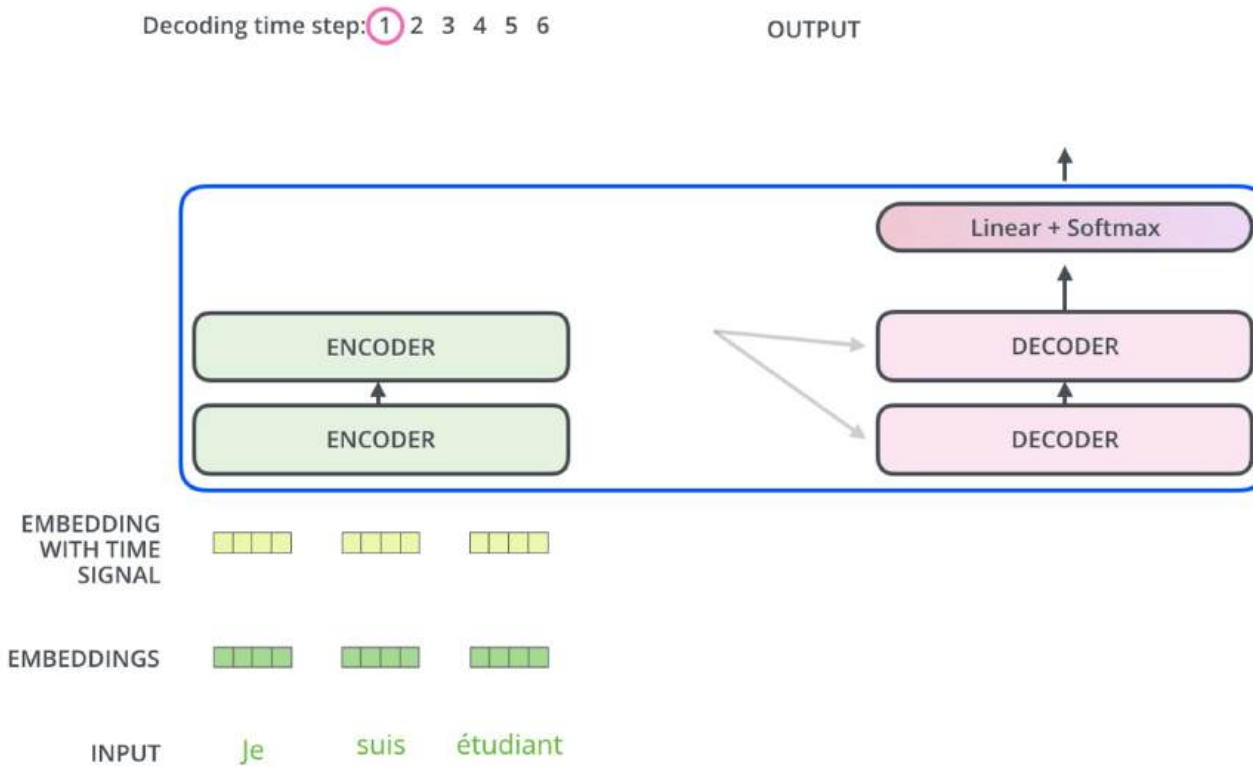


Figure 1: The Transformer - model architecture.

Encoder transforms sentences in language A into a highly abstract embedded space
Decoders progressively decodes highly abstract embedded space into language B

<https://jalammar.github.io/illustrated-transformer/>

Decoding

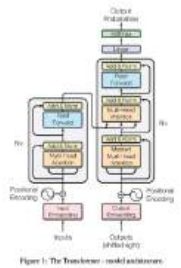
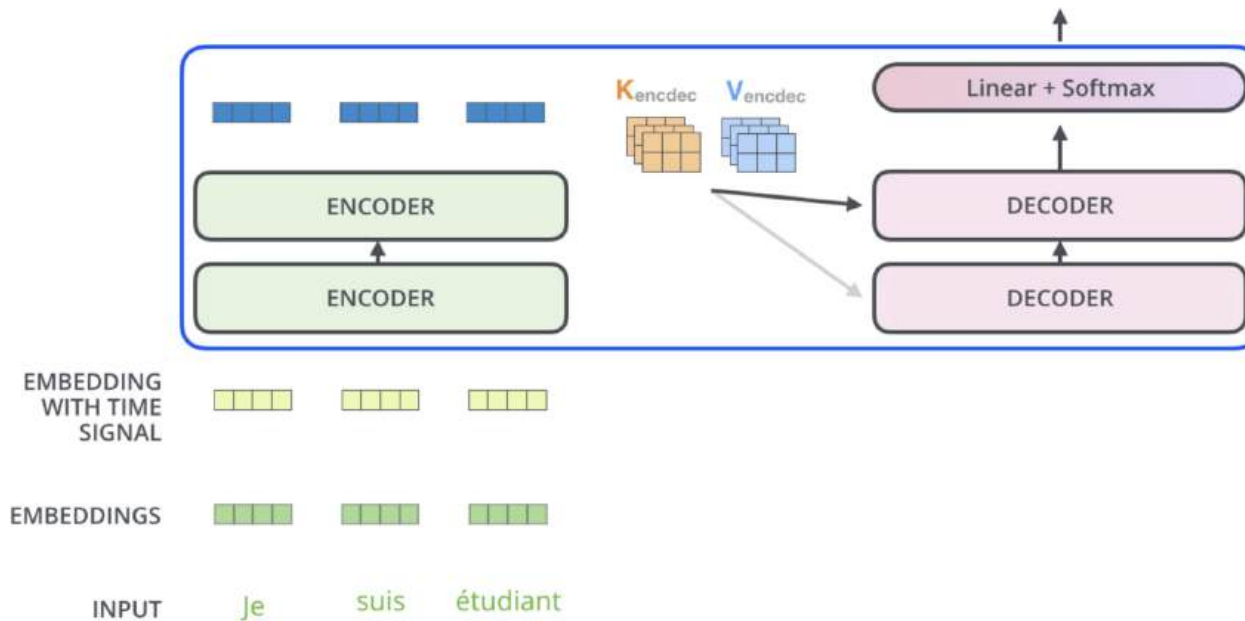


<https://jalammar.github.io/illustrated-transformer/>

Decoding

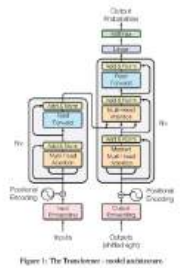
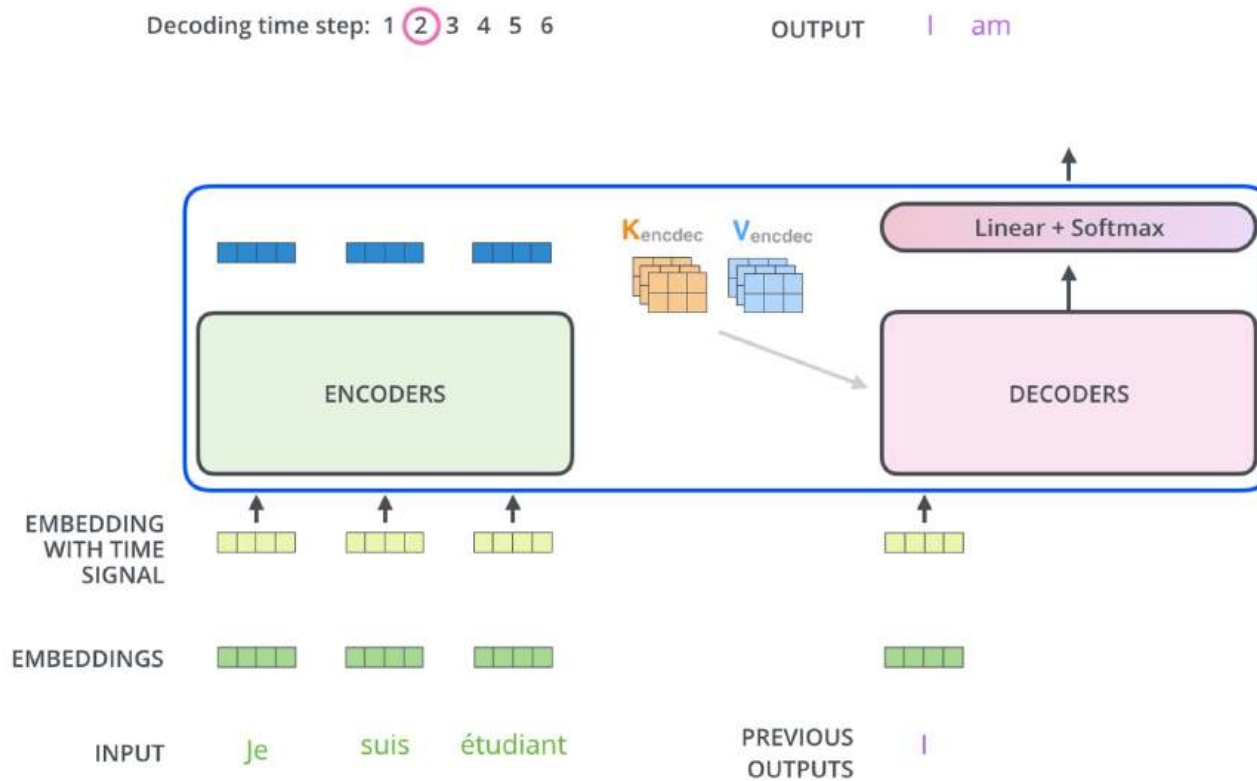
Decoding time step: 1 2 3 4 5 6

OUTPUT |



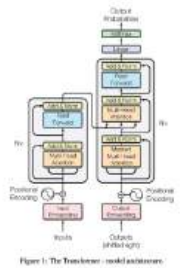
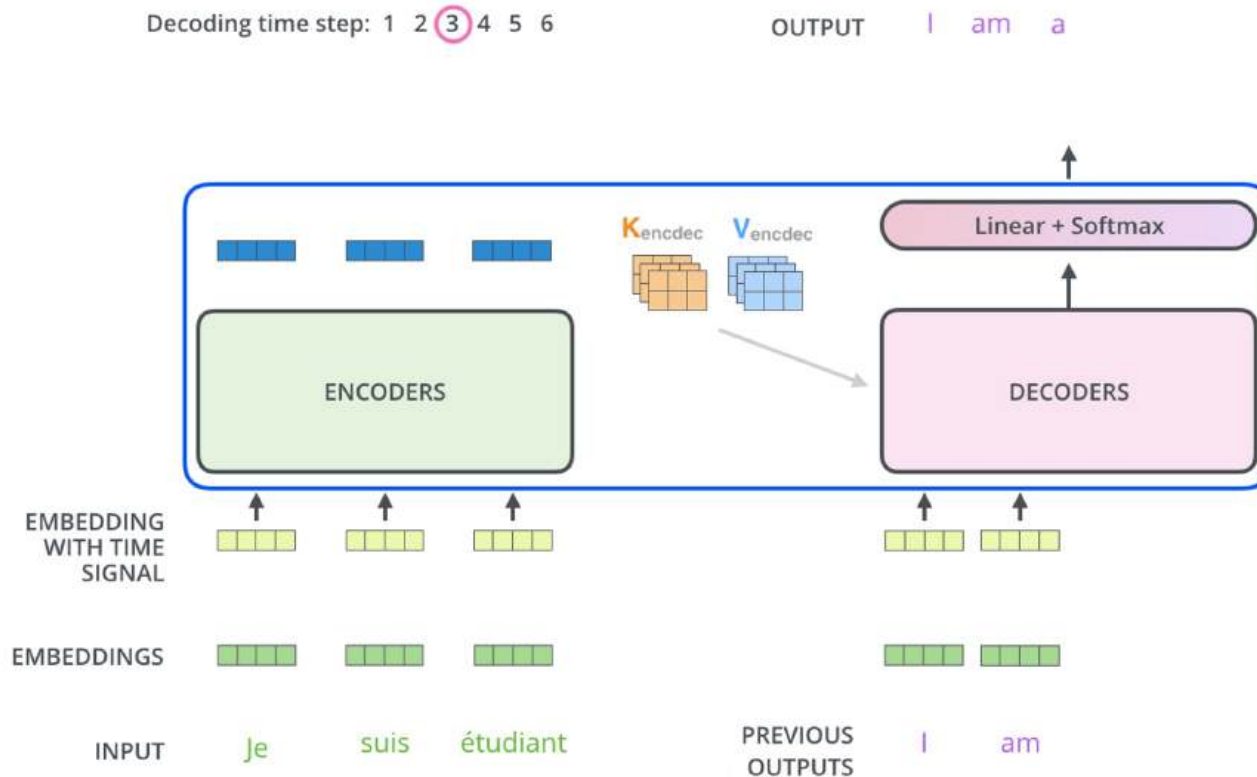
<https://jalammar.github.io/illustrated-transformer/>

Decoding



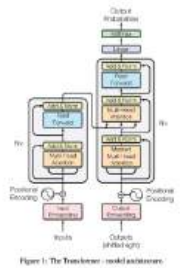
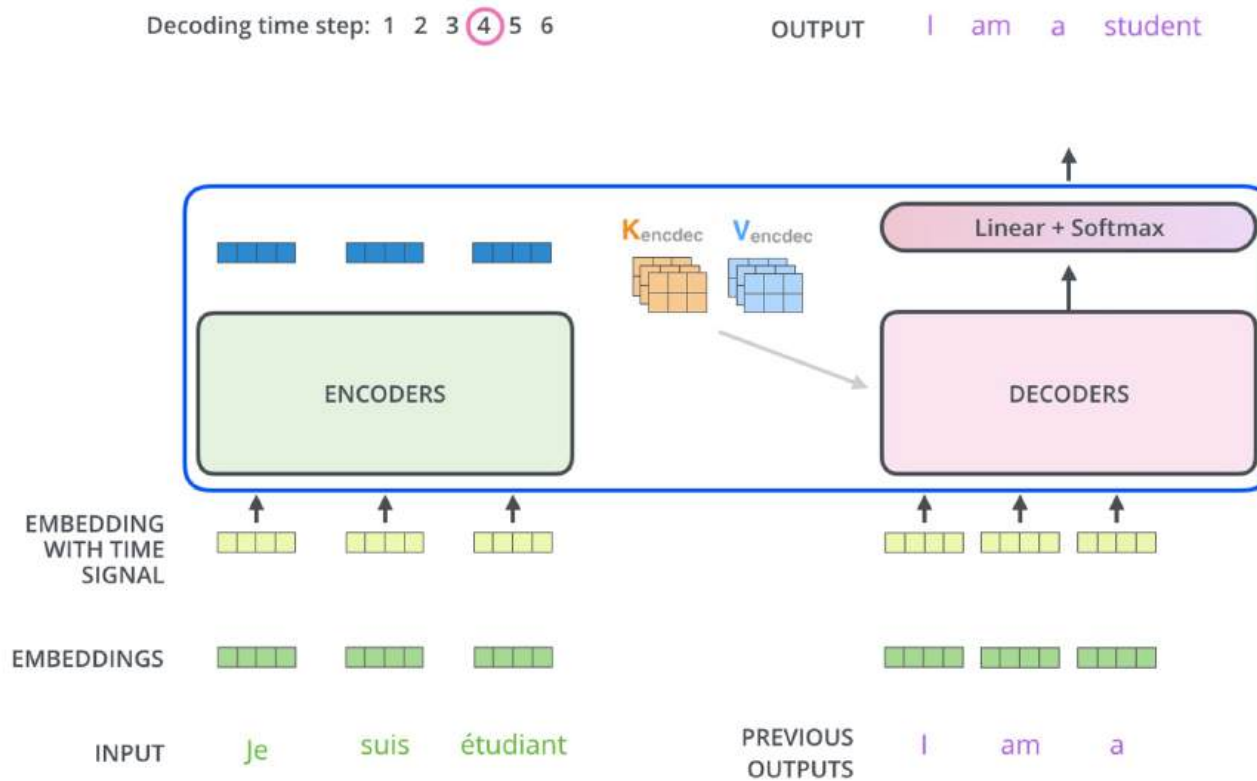
<https://jalammar.github.io/illustrated-transformer/>

Decoding



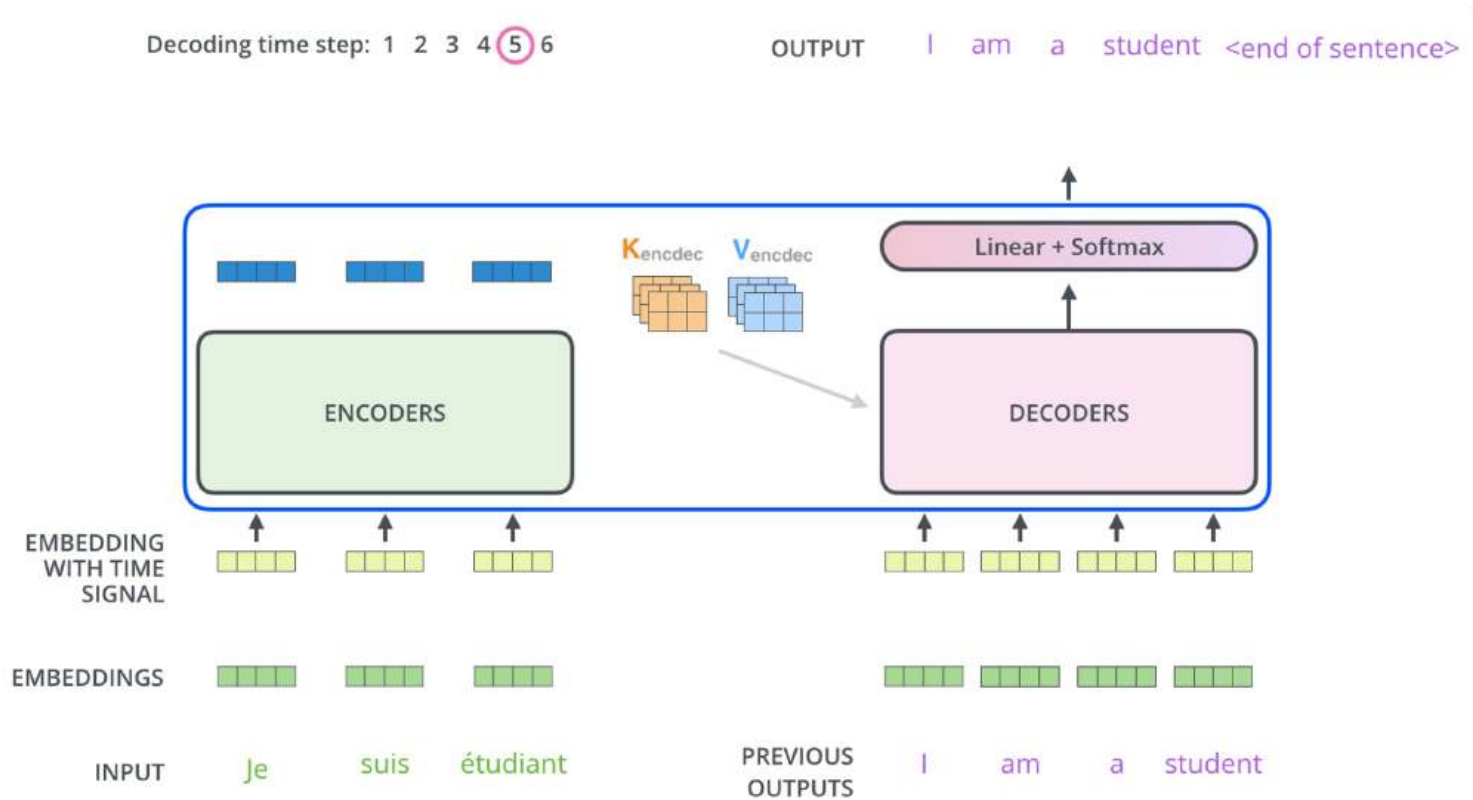
<https://jalammar.github.io/illustrated-transformer/>

Decoding



<https://jalammar.github.io/illustrated-transformer/>

Decoding



<https://jalammar.github.io/illustrated-transformer/>

Decoding back into words

Very similar to last few layers of AlexNet!

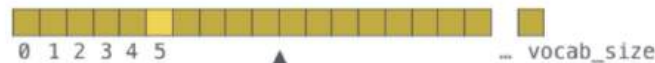
Which word in our vocabulary is associated with this index?

am

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

5

log_probs



Softmax

logits



Linear

Decoder stack output



Which word is this?

Pick the most probable

Prob. of being each term

Score for being a term

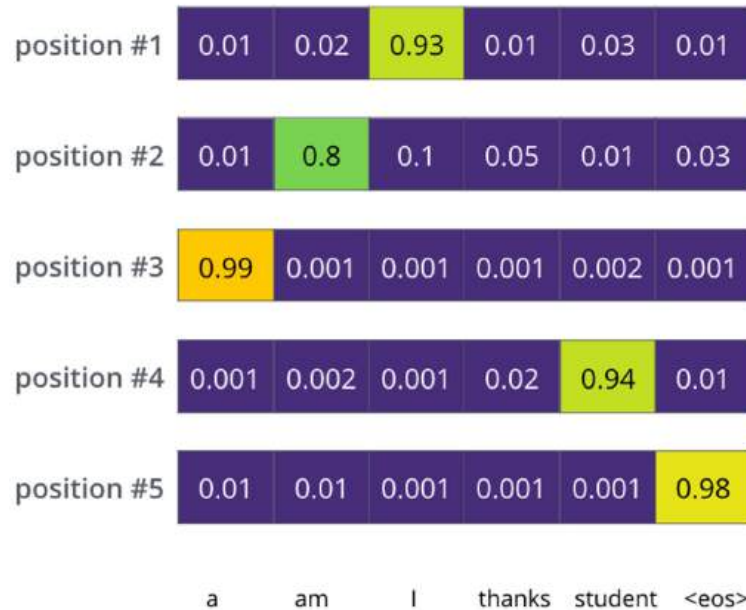
Abstract embedding

<https://jalammar.github.io/illustrated-transformer/>

Complete output

Trained Model Outputs

Output Vocabulary: a am I thanks student <eos>



Greedy decoding:

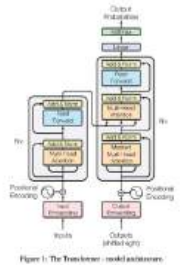
- Always pick the word with the highest probability

Beam search:

- run the model twice: once assuming the first output position was 'I', and another as 'a', maintain both decodings

Perplexity:

- Introduce randomness into the output



<https://jalammar.github.io/illustrated-transformer/>

Transformer: Multi-headed attention to encode and decode

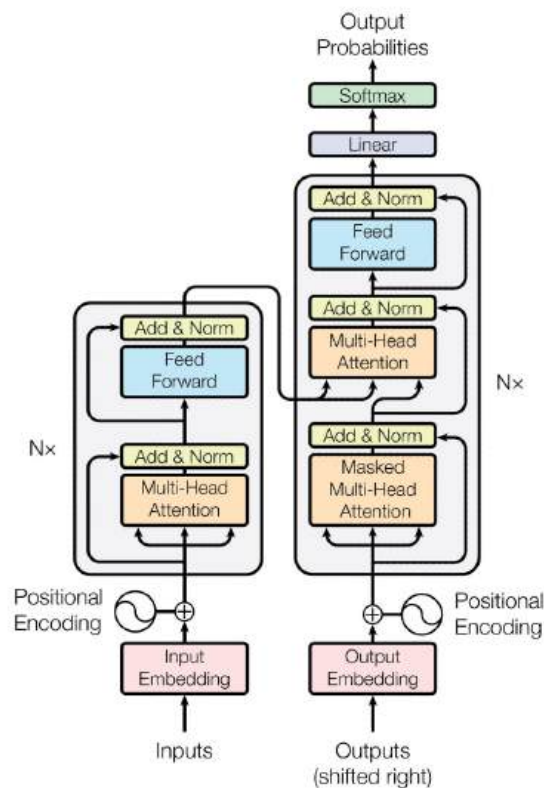


Figure 1: The Transformer - model architecture.

The Transformer uses multi-head attention in three different ways:

“Encoder self-attention”: In a self-attention layer all of the keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder.

“Encoder-decoder attention”: The queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sequence.

“Self-attention in the decoder”: Allows each position in the decoder to attend to all positions in the decoder up to and including that position. We need to prevent leftward information flow in the decoder to preserve the auto-regressive property. We implement this inside of scaled dot-product attention by masking out (setting to $-\infty$) all values in the input of the softmax which correspond to illegal connections.

Performance

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	41.29	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$3.3 \cdot 10^{18}$	
Transformer (big)	28.4	41.8	$2.3 \cdot 10^{19}$	

More accurate and 100 to 1000 times faster!

Outline

1. Recap on ANNs and CNNs
2. The problem
3. Self-Attention and Transformers
4. Impact

GPT-1

Improving Language Understanding by Generative Pre-Training

Alec Radford
OpenAI
alec@openai.com

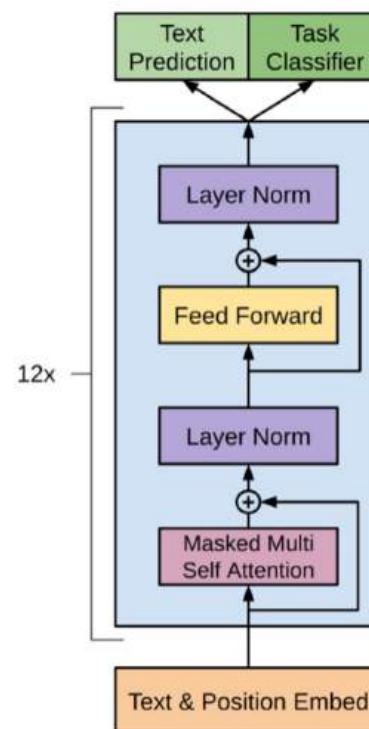
Karthik Narasimhan
OpenAI
karthikn@openai.com

Tim Salimans
OpenAI
tim@openai.com

Ilya Sutskever
OpenAI
ilyasu@openai.com

Abstract

Natural language understanding comprises a wide range of diverse tasks such as textual entailment, question answering, semantic similarity assessment, and document classification. Although large unlabeled text corpora are abundant, labeled data for learning these specific tasks is scarce, making it challenging for discriminatively trained models to perform adequately. We demonstrate that large gains on these tasks can be realized by *generative pre-training* of a language model on a diverse corpus of unlabeled text, followed by *discriminative fine-tuning* on each specific task. In contrast to previous approaches, we make use of task-aware input transformations during fine-tuning to achieve effective transfer while requiring minimal changes to the model architecture. We demonstrate the effectiveness of our approach on a wide range of benchmarks for natural language understanding. Our general task-agnostic model outperforms discriminatively trained models that use architectures specifically crafted for each task, significantly improving upon the state of the art in 9 out of the 12 tasks studied. For instance, we achieve absolute improvements of 8.9% on commonsense reasoning (Stories Cloze Test), 5.7% on question answering (RACE), and 1.5% on textual entailment (MultiNLI).



GPT-2

Language Models are Unsupervised Multitask Learners

Alec Radford^{*1} Jeffrey Wu^{*1} Rewon Child¹ David Luan¹ 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 but still underfits WebText. Samples from the model reflect these improvements and contain coherent paragraphs of text. These findings suggest a promising path towards building language processing systems which learn to perform tasks from their naturally occurring demonstrations.

1. Introduction

Machine learning systems now excel (in expectation) at tasks they are trained for by using a combination of large datasets, high-capacity models, and supervised learning (Krizhevsky et al., 2012) (Sutskever et al., 2014) (Amodei et al., 2016). Yet these systems are brittle and sensitive to slight changes in the data distribution (Recht et al., 2018) and task specification (Kirkpatrick et al., 2017). Current systems are better characterized as narrow experts rather than

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.

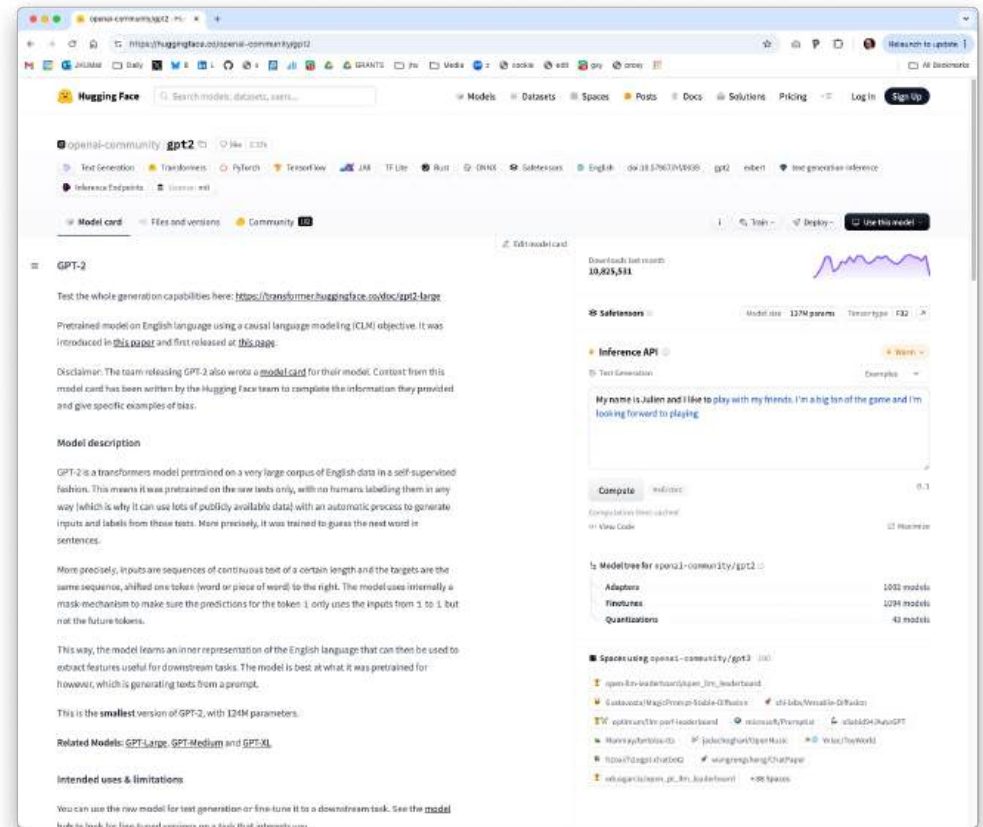
The dominant approach to creating ML systems is to collect a dataset of training examples demonstrating correct behavior for a desired task, train a system to imitate these behaviors, and then test its performance on independent and identically distributed (IID) held-out examples. This has served well 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 have been proposed such as GLUE (Wang et al., 2018) and decaNLP (McCann et al., 2018) to begin studying this.

Multitask learning (Caruana, 1997) is a promising framework for improving general performance. However, multitask training in NLP is still nascent. Recent work reports modest performance improvements (Vogelstein et al., 2019) and the two most ambitious efforts to date have trained on a total of 10 and 17 (dataset, objective) pairs respectively (McCann et al., 2018) (Bowman et al., 2018). From a meta-learning perspective, each (dataset, objective) pair is a single training example sampled from the distribution of datasets and objectives. Current ML systems need hundreds to thousands of examples to induce functions which generalize well. This suggests that multitask training may need just as many effective training pairs to realize its promise with current approaches. It will be very difficult to continue to scale the creation of datasets and the design of objectives to the degree that may be required to brute force our way there with current techniques. This motivates exploring additional setups for performing multitask learning.

The current best performing systems on language tasks

^{*}^{*}Equal contribution. ¹OpenAI, San Francisco, California, United States. Correspondence to: Alec Radford <alec@openai.com>.



	 LeNet-5	 AlexNet	 GPT-3	 GPT-4
YEAR	1998	2012	2020	2023
TRAINING DATA	<ul style="list-style-type: none"> · MNIST Dataset · 60k training examples · 10 classes 	<ul style="list-style-type: none"> · ImageNet Dataset (ILSVRC) · 1.2M training examples · 1000 classes 	<ul style="list-style-type: none"> · Common Crawl, WebText, Wikipedia, others · ~500B training tokens · ~100k unique tokens 	<ul style="list-style-type: none"> · ~13T training tokens*
TRAINING COMPUTE	<ul style="list-style-type: none"> · Pentium II CPU · ~0.27 GFLOPs 	<ul style="list-style-type: none"> · Dual Nvidia GTX 580 · 3162 GFLOPs 	<ul style="list-style-type: none"> · 10,000 Nvidia V100 GPUs · 1+ ExaFLOPs 	<ul style="list-style-type: none"> · 25,000 Nvidia A100s GPUs* · ~4+ ExaFLOPs
ALGORITHM	<ul style="list-style-type: none"> · ~60k Parameters · 5 Layers · Sigmoid Activation Function 	<ul style="list-style-type: none"> · ~60M Parameters · 8 Layers · ReLU Activation Function · Dropout 	<ul style="list-style-type: none"> · 175B Parameters · 96 Layers · Transformers 	<ul style="list-style-type: none"> · 1T+ Parameters* · ~120 Layers · Transformers

<https://www.youtube.com/watch?v=UZDiGooFs54>

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin Ming-Wei Chang Kenton Lee Kristina Toutanova
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

Abstract

We introduce a new language representation model called **BERT**, which stands for **Bidirectional Encoder Representations from Transformers**. Unlike recent language representation models (Peters et al., 2018a; Radford et al., 2018), BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications.

BERT is conceptually simple and empirically powerful. It obtains new state-of-the-art results on eleven natural language processing tasks, including pushing the GLUE score to 80.5% (7.7% point absolute improvement), MultiNLI accuracy to 86.7% (4.6% absolute improvement), SQuAD v1.1 question answering Test F1 to 93.2 (1.5 point absolute improvement) and SQuAD v2.0 Test F1 to 83.1 (5.1 point absolute improvement).

1 Introduction

Language model pre-training has been shown to be effective for improving many natural language processing tasks (Dai and Le, 2015; Peters et al., 2018a; Radford et al., 2018; Howard and Ruder, 2018). These include sentence-level tasks such as natural language inference (Bowman et al., 2015; Williams et al., 2018) and paraphrasing (Dolan and Brockett, 2005), which aim to predict the relationships between sentences by analyzing them holistically, as well as token-level tasks such as

There are two existing strategies for applying pre-trained language representations to downstream tasks: *feature-based* and *fine-tuning*. The feature-based approach, such as ELMo (Peters et al., 2018a), uses task-specific architectures that include the pre-trained representations as additional features. The fine-tuning approach, such as the Generative Pre-trained Transformer (OpenAI GPT) (Radford et al., 2018), introduces minimal task-specific parameters, and is trained on the downstream tasks by simply fine-tuning *all* pre-trained parameters. The two approaches share the same objective function during pre-training, where they use unidirectional language models to learn general language representations.

We argue that current techniques restrict the power of the pre-trained representations, especially for the fine-tuning approaches. The major limitation is that standard language models are unidirectional, and this limits the choice of architectures that can be used during pre-training. For example, in OpenAI GPT, the authors use a left-to-right architecture, where every token can only attend to previous tokens in the self-attention layers of the Transformer (Vaswani et al., 2017). Such restrictions are sub-optimal for sentence-level tasks, and could be very harmful when applying fine-tuning based approaches to token-level tasks such as question answering, where it is crucial to incorporate context from both directions.

In this paper, we improve the fine-tuning based approaches by proposing BERT: **Bidirectional Encoder Representations from Transformers**. BERT alleviates the previously mentioned unidirectionality constraint by using a “masked language model” (MLM) pre-training objective, in-

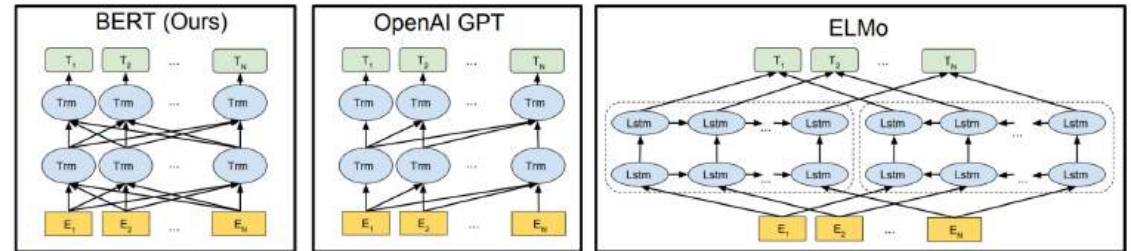


Figure 3: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTMs to generate features for downstream tasks. Among the three, only BERT representations are jointly conditioned on both left and right context in all layers. In addition to the architecture differences, BERT and OpenAI GPT are fine-tuning approaches, while ELMo is a feature-based approach.

3. Using BERT for Different NLP Tasks

- **Text Classification:** BERT can be used for tasks like sentiment analysis, spam detection, or topic classification. After fine-tuning, the “[CLS]” token’s output (the special token at the start of the sequence) is passed through a classifier to predict the label of the text.
- **Named Entity Recognition (NER):** BERT can be fine-tuned to recognize and classify named entities (e.g., names, organizations, dates) in text. Each token’s output from BERT is passed through a classification layer to predict the entity label.
- **Question Answering:** For tasks like SQuAD (Stanford Question Answering Dataset), BERT can be used to find the answer to a question within a given passage. The model is fine-tuned to predict the start and end positions of the answer span in the passage.
- **Sentence Pair Classification:** Tasks like entailment (e.g., recognizing textual entailment) or semantic textual similarity can be handled by BERT. Two sentences are input, and BERT is fine-tuned to classify their relationship (e.g., entailment, contradiction, neutral).
- **Text Generation:** While BERT itself is not primarily a text generation model (since it’s not designed to generate text sequentially), it can be adapted for certain text completion tasks or used in conjunction with other models for generation.

BERT: Pre-training of Deep Bidirectional Transformers for ...

by J Devlin · 2018 · Cited by 108443 — We introduce a new language representation model

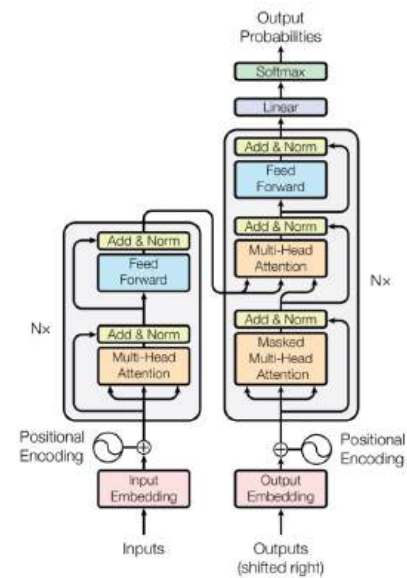
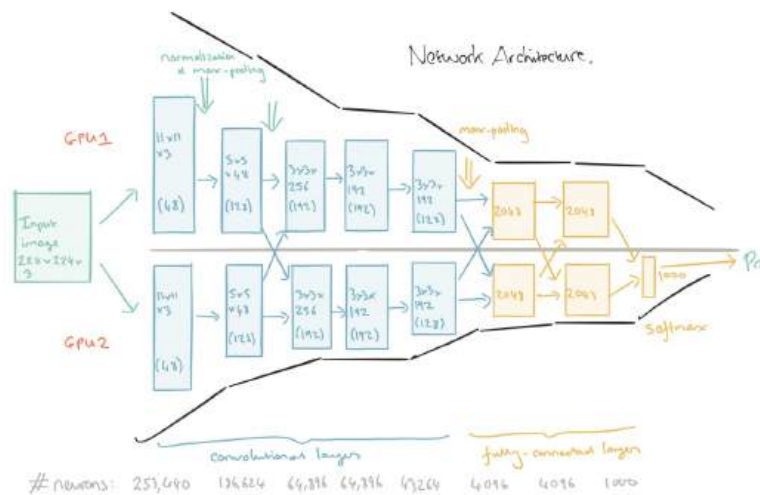
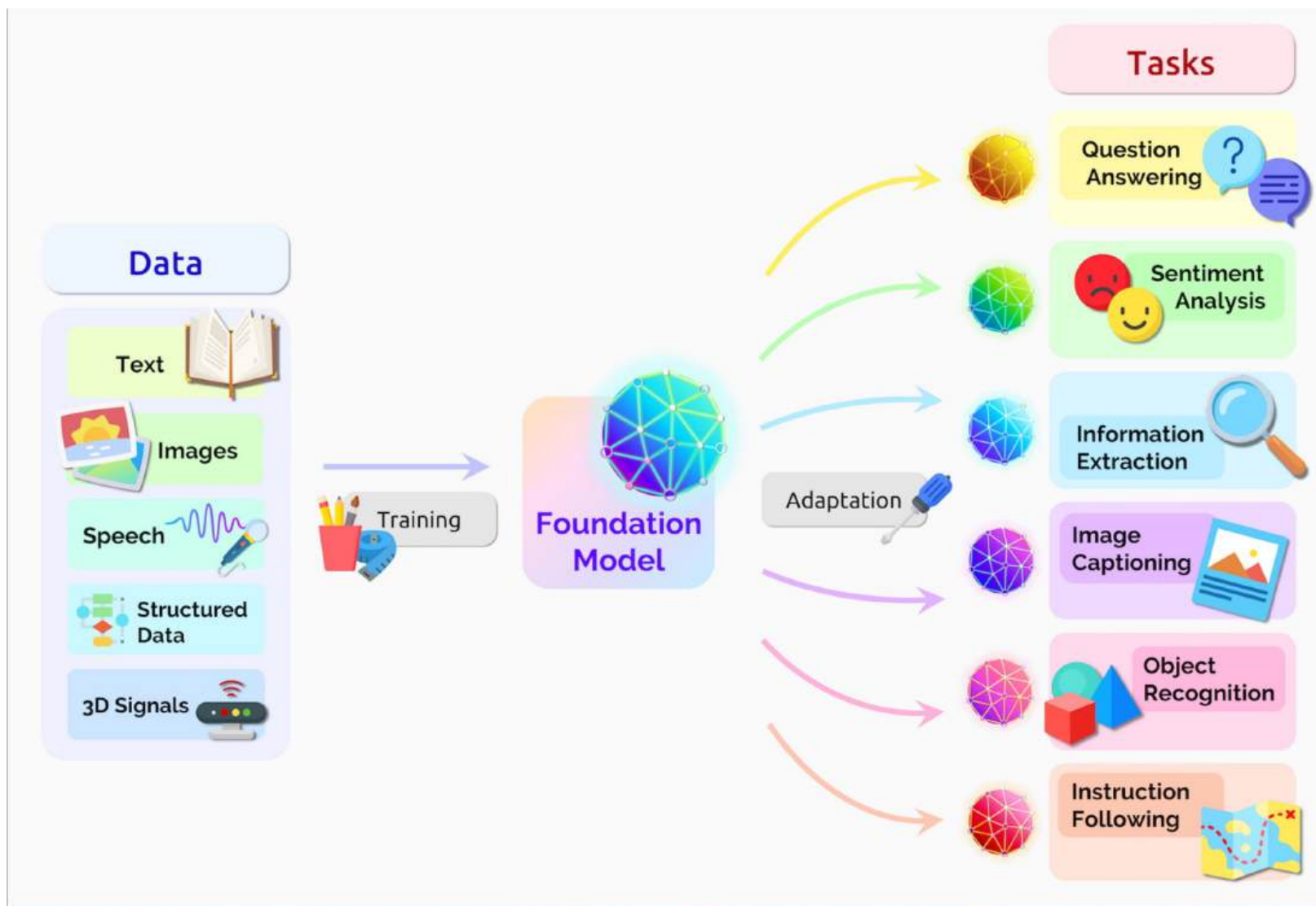


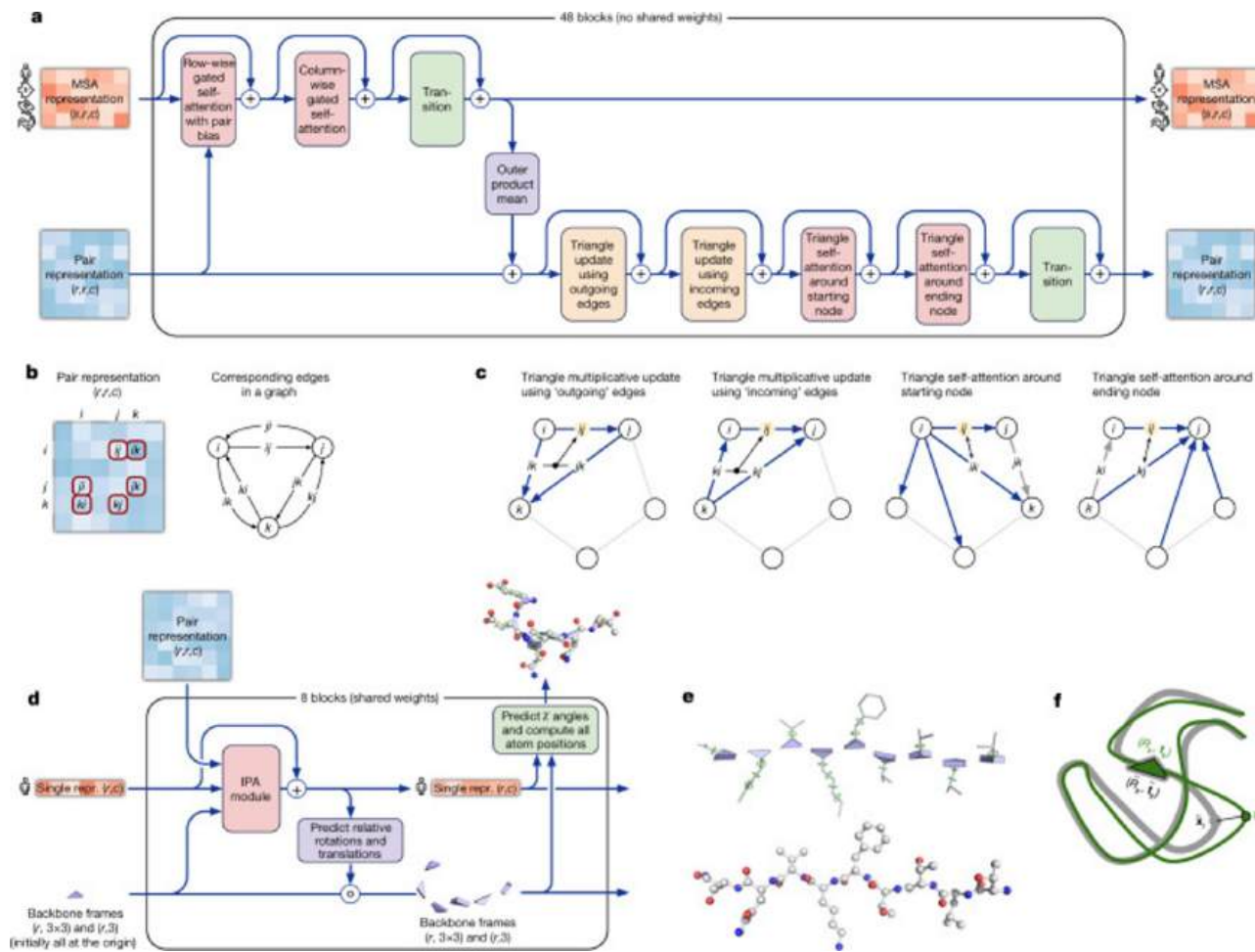
Figure 1: The Transformer - model architecture.

Image to text



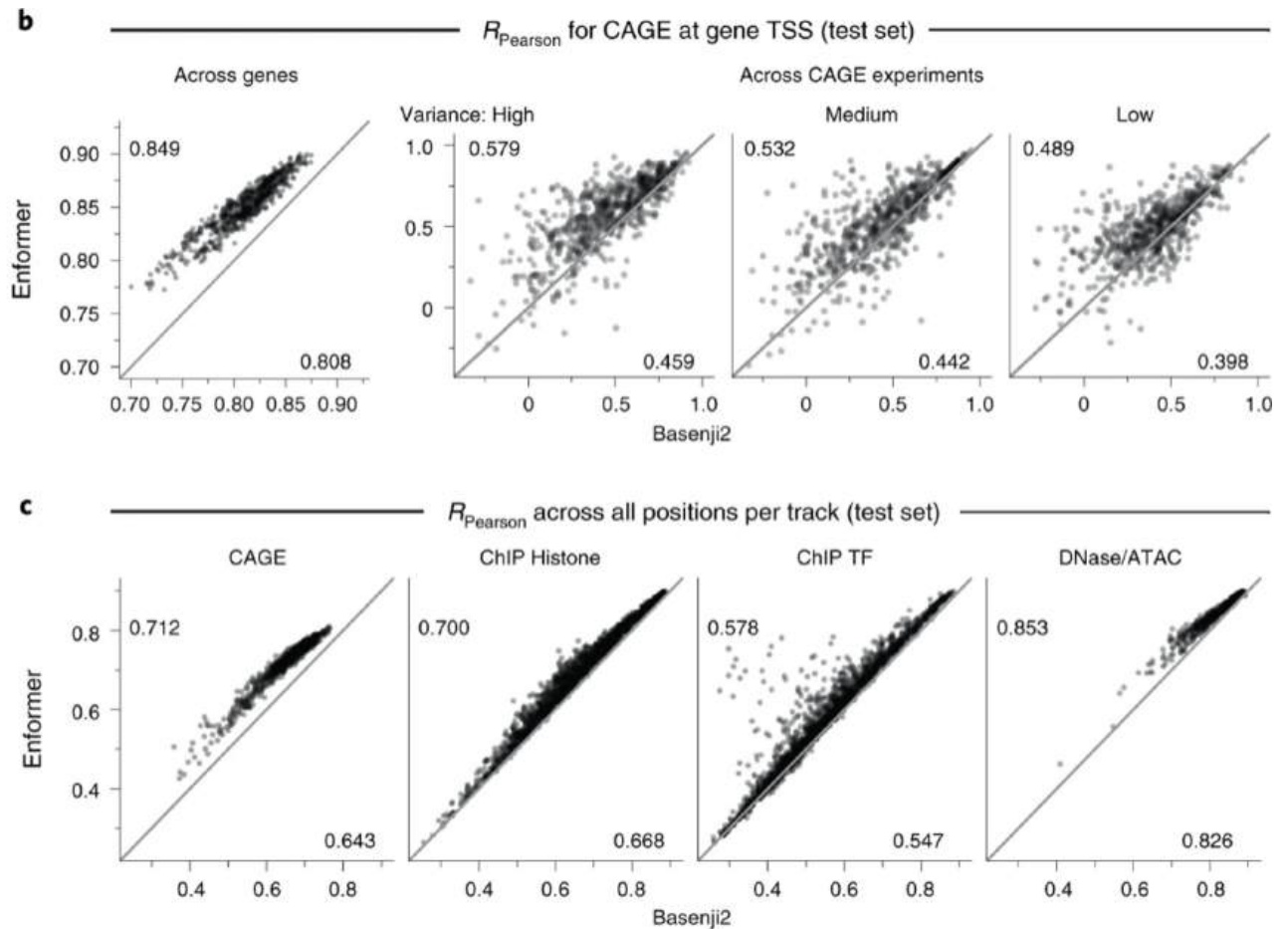
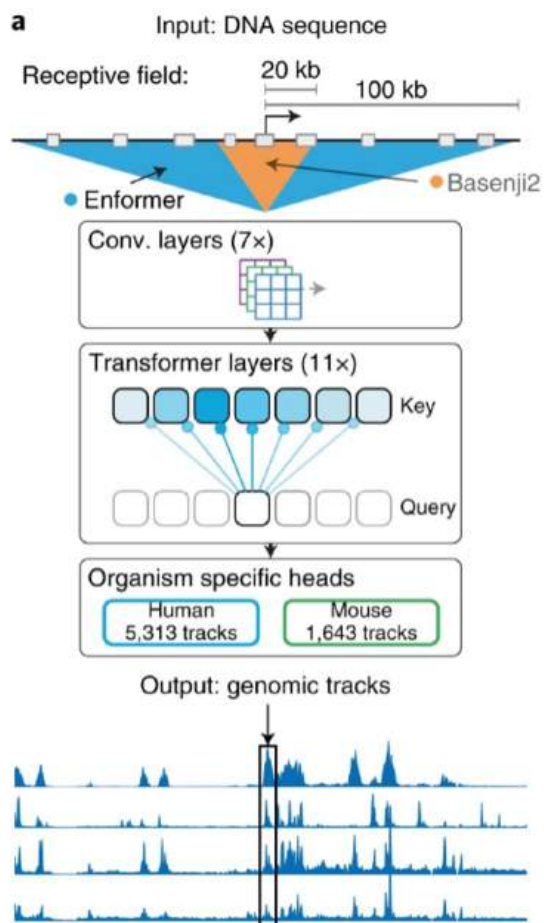
Text to image





Highly accurate protein structure prediction with AlphaFold (“Evoformer”)

Jumper et al. (2021) Nature. doi: 10.1038/s41586-021-03819-2



“Enformer” Effective gene expression prediction from sequence by integrating long-range interactions
 Avsec et al. (2021) Nature Methods. <https://doi.org/10.1038/s41592-021-01252-x>

Fig. 2: Enformer attends to cell-type-specific enhancers, enabling enhancer prioritization.

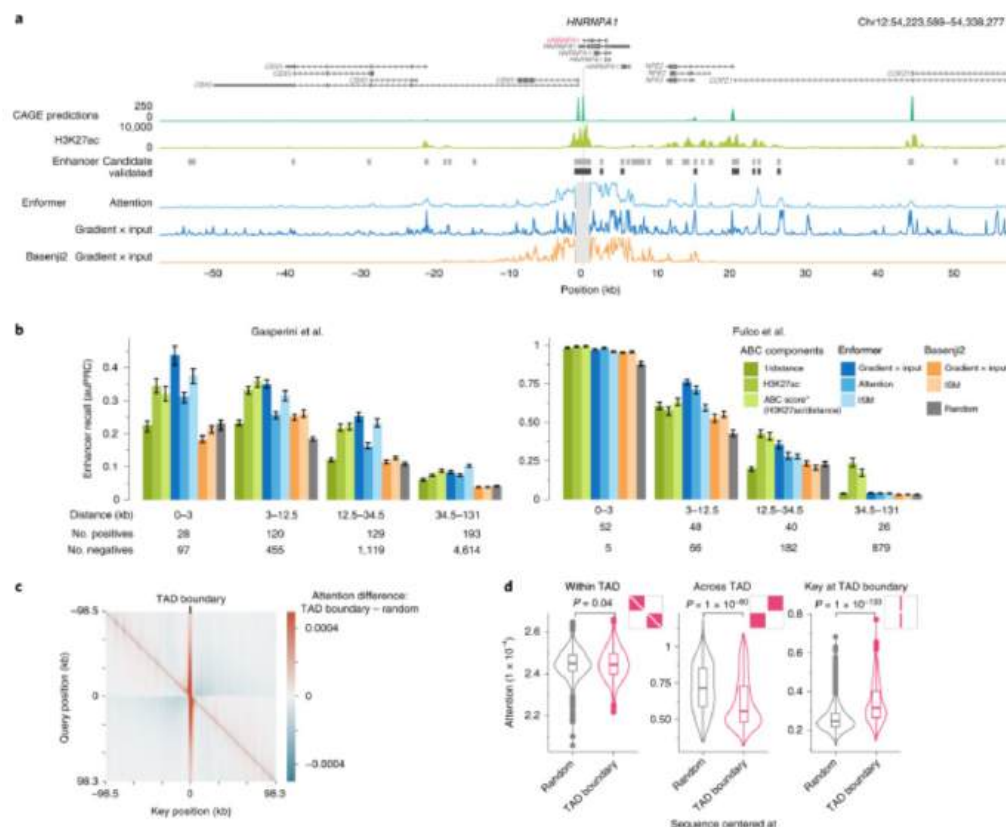
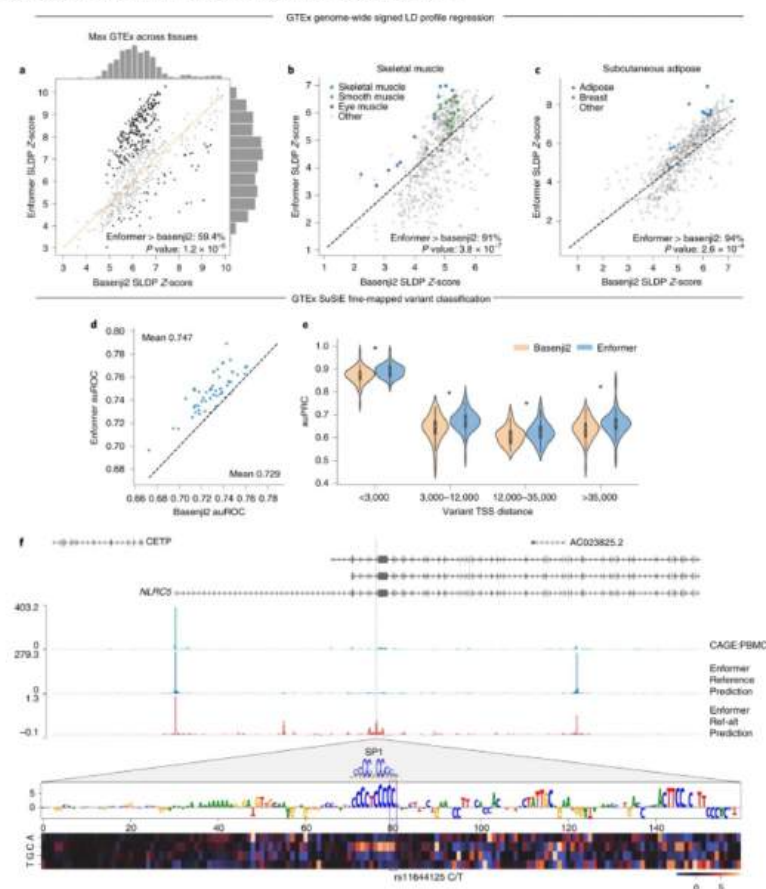


Fig. 3: Enformer improves variant effect prediction on eQTL data as measured by SLDP regression and fine-mapped variant classification.



“Enformer” Effective gene expression prediction from sequence by integrating long-range interactions
 Avsec et al. (2021) Nature Methods. <https://doi.org/10.1038/s41592-021-01252-x>

Personal transcriptome variation is poorly explained by current genomic deep learning models

[Connie Huang](#), [Richard W. Shuai](#), [Parth Baokar](#), [Ryan Chung](#), [Ruchir Rastogi](#), [Pooja Kathail](#) & [Nilah M. Ioannidis](#) 

[Nature Genetics](#) **55**, 2056–2059 (2023) | [Cite this article](#)

14k Accesses | **11** Citations | **79** Altmetric | [Metrics](#)

Abstract

Genomic deep learning models can predict genome-wide epigenetic features and gene expression levels directly from DNA sequence. While current models perform well at predicting gene expression levels across genes in different cell types from the reference genome, their ability to explain expression variation between individuals due to *cis*-regulatory genetic variants remains largely unexplored. Here, we evaluate four state-of-the-art models on paired personal genome and transcriptome data and find limited performance when explaining variation in expression across individuals. In addition, models often fail to predict the correct direction of effect of *cis*-regulatory genetic variation on expression.

Benchmarking of deep neural networks for predicting personal gene expression from DNA sequence highlights shortcomings

[Alexander Sasse](#), [Bernard Ng](#), [Anna E. Spiro](#), [Shinya Tasaki](#), [David A. Bennett](#), [Christopher Gaiteri](#), [Philip L. De Jager](#), [Maria Chikina](#)  & [Sara Mostafavi](#) 

[Nature Genetics](#) **55**, 2060–2064 (2023) | [Cite this article](#)

10k Accesses | **14** Citations | **77** Altmetric | [Metrics](#)

Abstract

Deep learning methods have recently become the state of the art in a variety of regulatory genomic tasks^{1,2,3,4,5,6}, including the prediction of gene expression from genomic DNA. As such, these methods promise to serve as important tools in interpreting the full spectrum of genetic variation observed in personal genomes. Previous evaluation strategies have assessed their predictions of gene expression across genomic regions; however, systematic benchmarking is lacking to assess their predictions across individuals, which would directly evaluate their utility as personal DNA interpreters. We used paired whole genome sequencing and gene expression from 839 individuals in the ROSMAP study⁷ to evaluate the ability of current methods to predict gene expression variation across individuals at varied loci. Our approach identifies a limitation of current methods to correctly predict the direction of variant effects. We show that this limitation stems from insufficiently learned sequence motif grammar and suggest new model training strategies to improve performance.

Discussion

Transformers and related architectures are transforming language processing and related tasks: attention is all you need

- Attention finds for a given term which other terms are important
- Transformers are great for language processing by computing progressively more abstract features and concepts in a very high dimensional space

Questions:

- DNA sequences share some properties with natural language
- Natural language has a lot of semantics, and is constrained by human brain
- If we apply to bioinformatics, do the same semantics and constraints apply?
- Some parts yes (Enformer is worker), some parts need more research!