

Embeddings, Attention and Transformers

Luiz Velho

Outlook

- Embeddings and Word2Vec
- From Word2Vec to Attention
- Self-Attention and Transformers
- Biases of Attention
- Visualization of Transformers

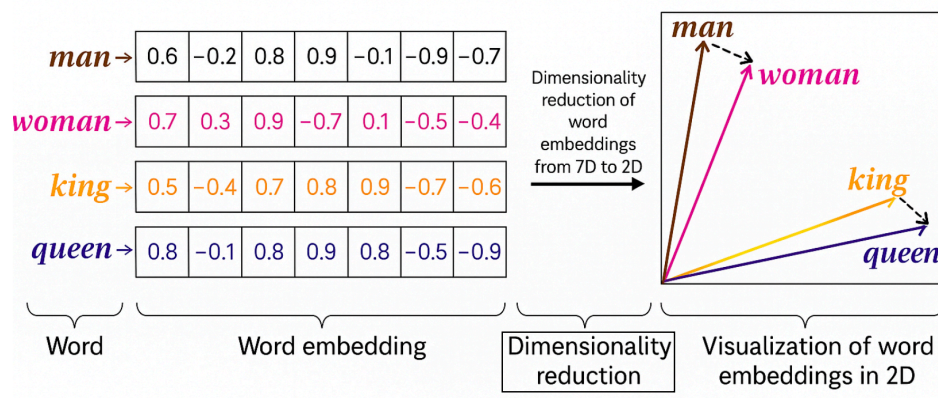
Embeddings and Word2Vec

[Efficient Estimation of Word Representations in Vector Space, Mikolov et al. 2013]

Based on Jay Almar Blog and others

Word Embeddings

- From Linguistic Corpus to Embedding Space



Vector Space Structure

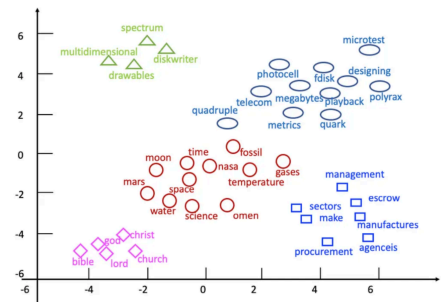
- Similarity

$$\text{cosine_similarity}\left(\begin{matrix} \text{Jay} \\ -0.4 & 0.8 \end{matrix}, \begin{matrix} \text{Person \#1} \\ -0.3 & 0.2 \end{matrix}\right) = 0.87 \quad \checkmark$$

$$\text{cosine_similarity}\left(\begin{matrix} \text{Jay} \\ -0.4 & 0.8 \end{matrix}, \begin{matrix} \text{Person \#2} \\ -0.5 & -0.4 \end{matrix}\right) = -0.20$$

- Operations

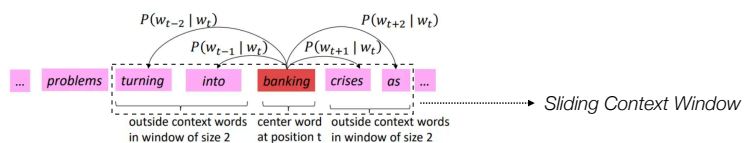
$$\text{king} - \text{man} + \text{woman} \approx \text{queen}$$



Learning the Embedding

- The Skip-Gram Model

- Maximize the probability of each context word given the center word



- Data Likelihood

$$P = \prod_{t=1}^T \prod_{-m \leq j \leq m, j \neq 0} P(w_{t+j} | w_t)$$

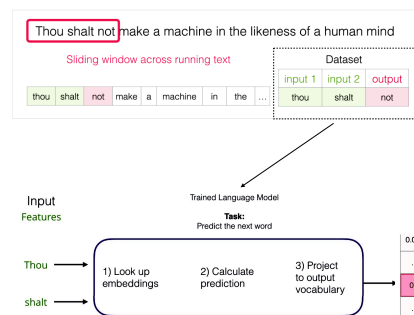
- Objective Function

$$J = -\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log P(w_{t+j} | w_t)$$

- Prediction

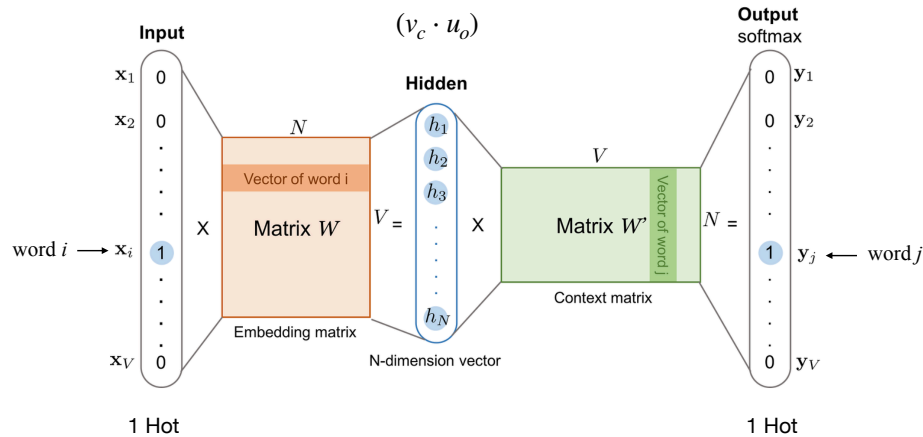
$$P(o|c) = \frac{\exp(v_c \cdot u_o)}{\sum_{w \in V} \exp(v_c \cdot u_w)}$$

output center vocabulary



Word2Vec

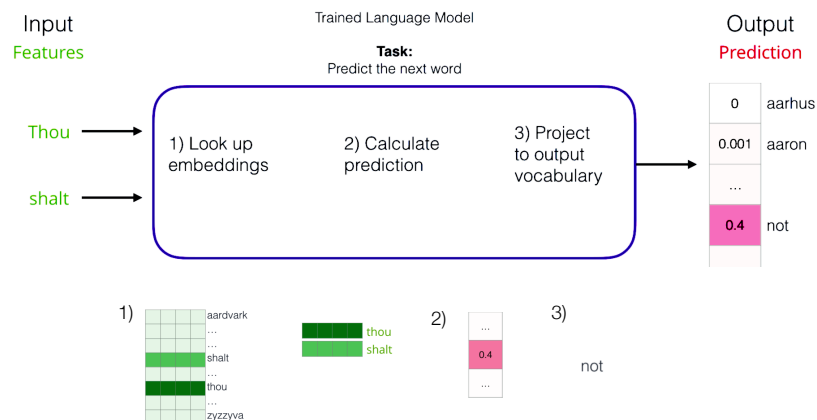
- Architecture (Skip-Gram)



Word2Vec Feed Forward

- Predicting the Neighboring Word

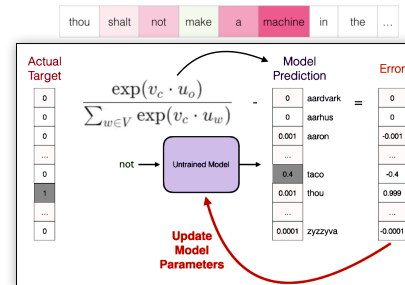
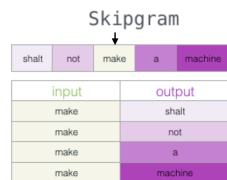
"Thou shalt not make a machine in the likeness of a human mind" ~Dune



Training Word2Vec

- Skipgram

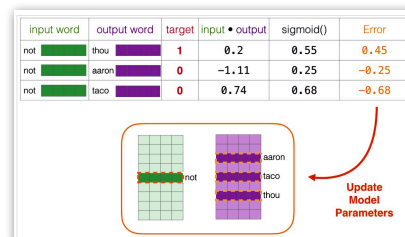
Thou shalt not make a machine in the likeness of a human mind



- Negative Sampling

Negative Sampling

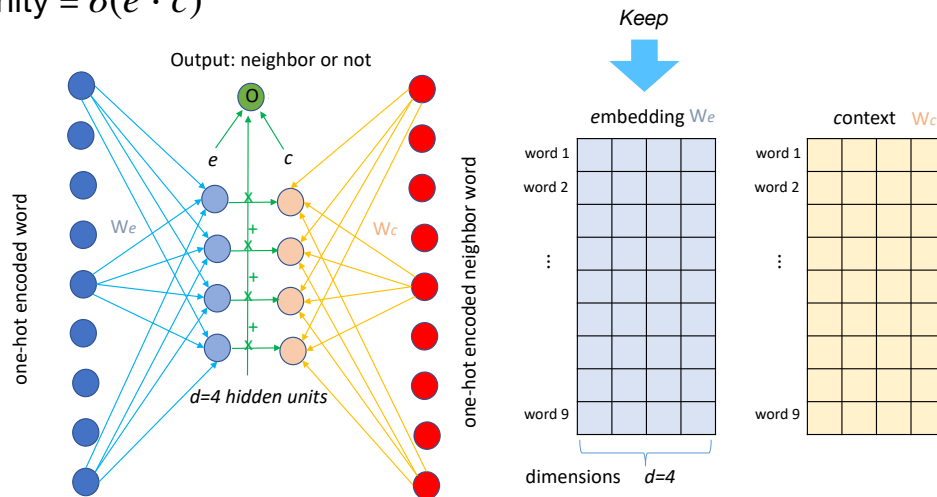
input word	output word	target
make	shalt	1
make	aaron	0
make	taco	0



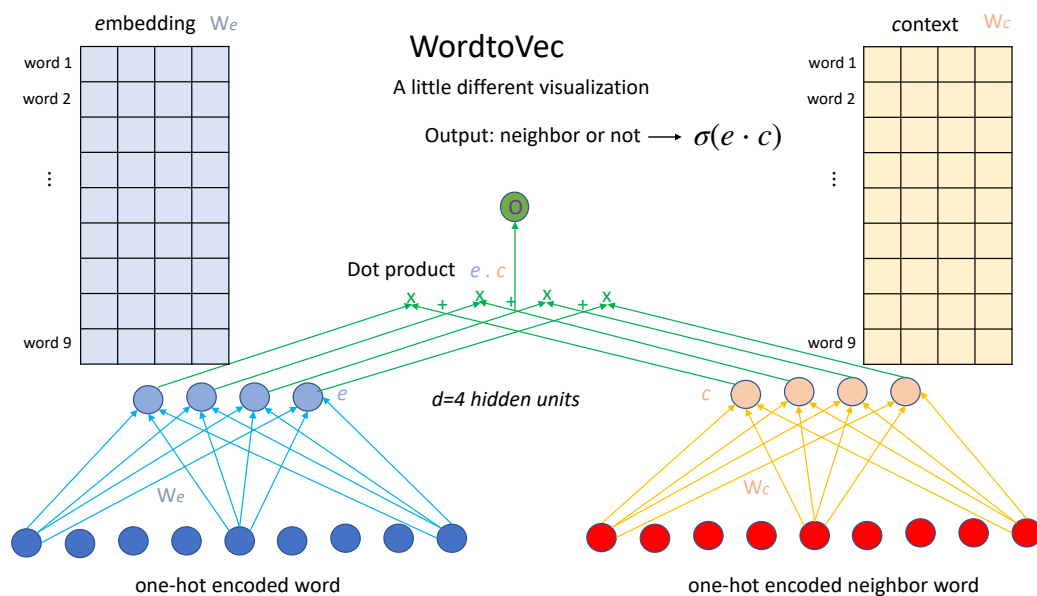
From Word2Vec to Transformers

Revisiting Word2Vec

- Affinity = $\sigma(e \cdot c)$

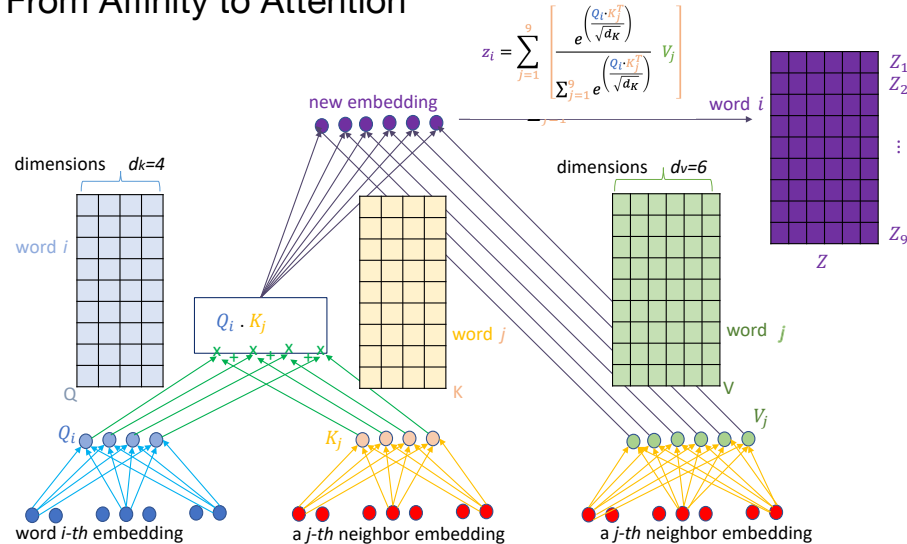


Changing the Point of View



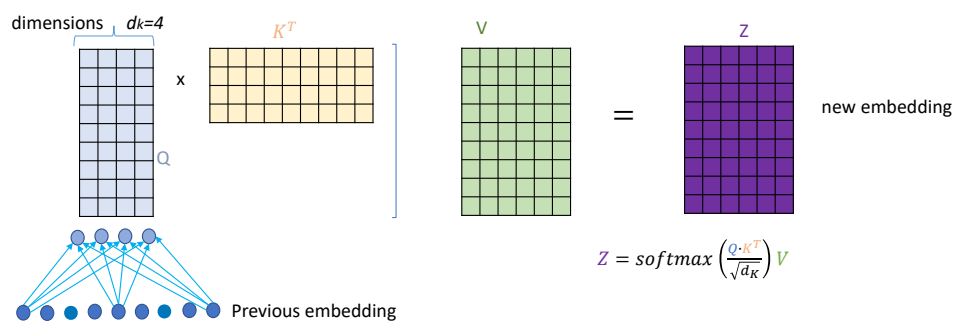
Telescoping the Embedding

- From Affinity to Attention



Query, Key and Value

Transformers: One Head



Attention Intuition

(Q K V)

Slides from Deep Learning Course (MIT)

Understanding Attention with Search

YouTube

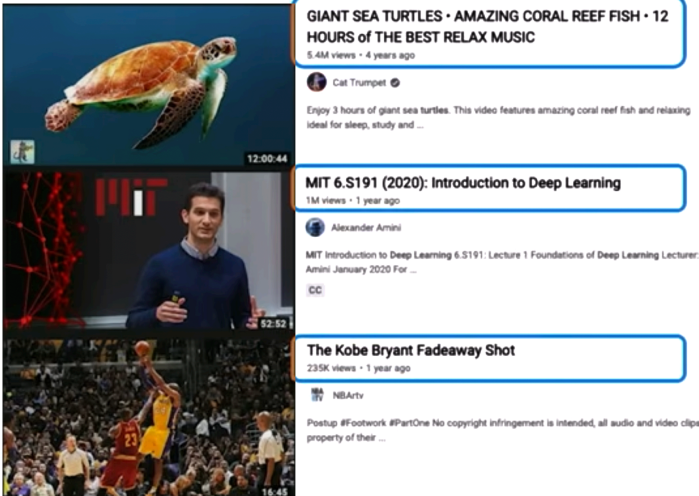
deep learning

Query (Q)

Key (K₁)

Key (K₂)

Key (K₃)



GIANT SEA TURTLES • AMAZING CORAL REEF FISH • 12 HOURS of THE BEST RELAX MUSIC

5.4M views • 4 years ago

Cat Trumpet

Enjoy 3 hours of giant sea turtles. This video features amazing coral reef fish and relaxing ideal for sleep, study and ...

MIT 6.S191 (2020): Introduction to Deep Learning

1M views • 1 year ago

Alexander Amini

MIT Introduction to Deep Learning 6.S191: Lecture 1 Foundations of Deep Learning Lecturer: Amini January 2020 For ...

CC

The Kobe Bryant Fadeaway Shot

235K views • 1 year ago

NBA

NBAArty

Postup #Footwork #PartOne No copyright infringement is intended, all audio and video clips property of their ...

Understanding Attention with Search 1.

YouTube × Query (Q)

MIT 6.S191 (2020): Introduction to Deep Learning

Alexander Amini

MIT Introduction to Deep Learning 6.S191: Lecture 1 Foundations of Deep Learning Lecturer: Amini January 2020 For ...

CC

Key (K₂)

How similar is the key to the query?

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

MIT Introduction to Deep Learning

Understanding Attention with Search 2.

YouTube × Query (Q)

MIT 6.S191 (2020): Introduction to Deep Learning

Alexander Amini

MIT Introduction to Deep Learning 6.S191: Lecture 1 Foundations of Deep Learning Lecturer: Amini January 2020 For ...

CC

Key (K₂)

Value (V)

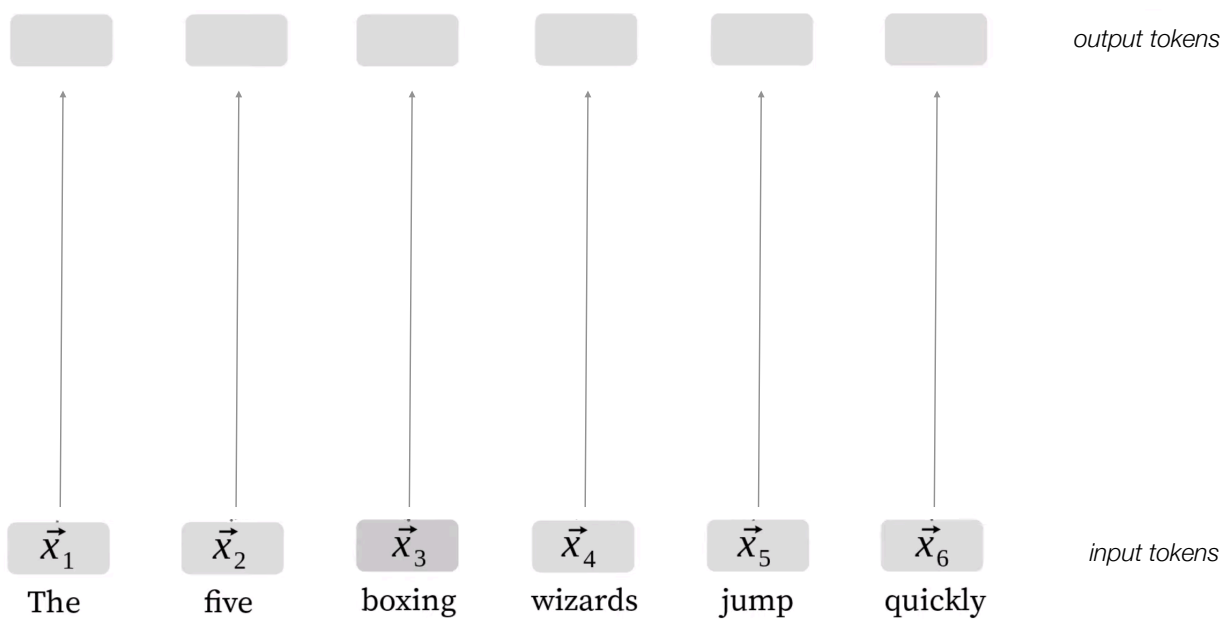
2. **Extract values based on attention:**
Return the values highest attention

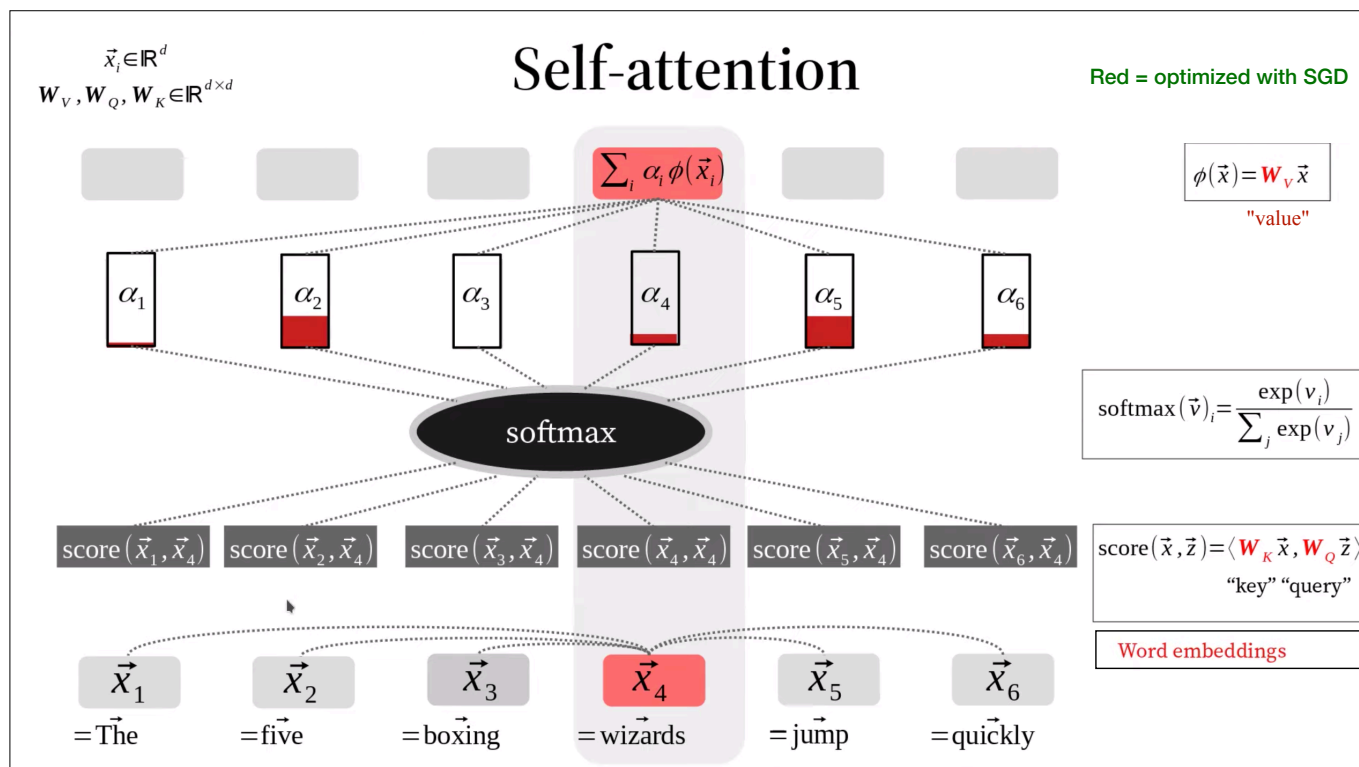
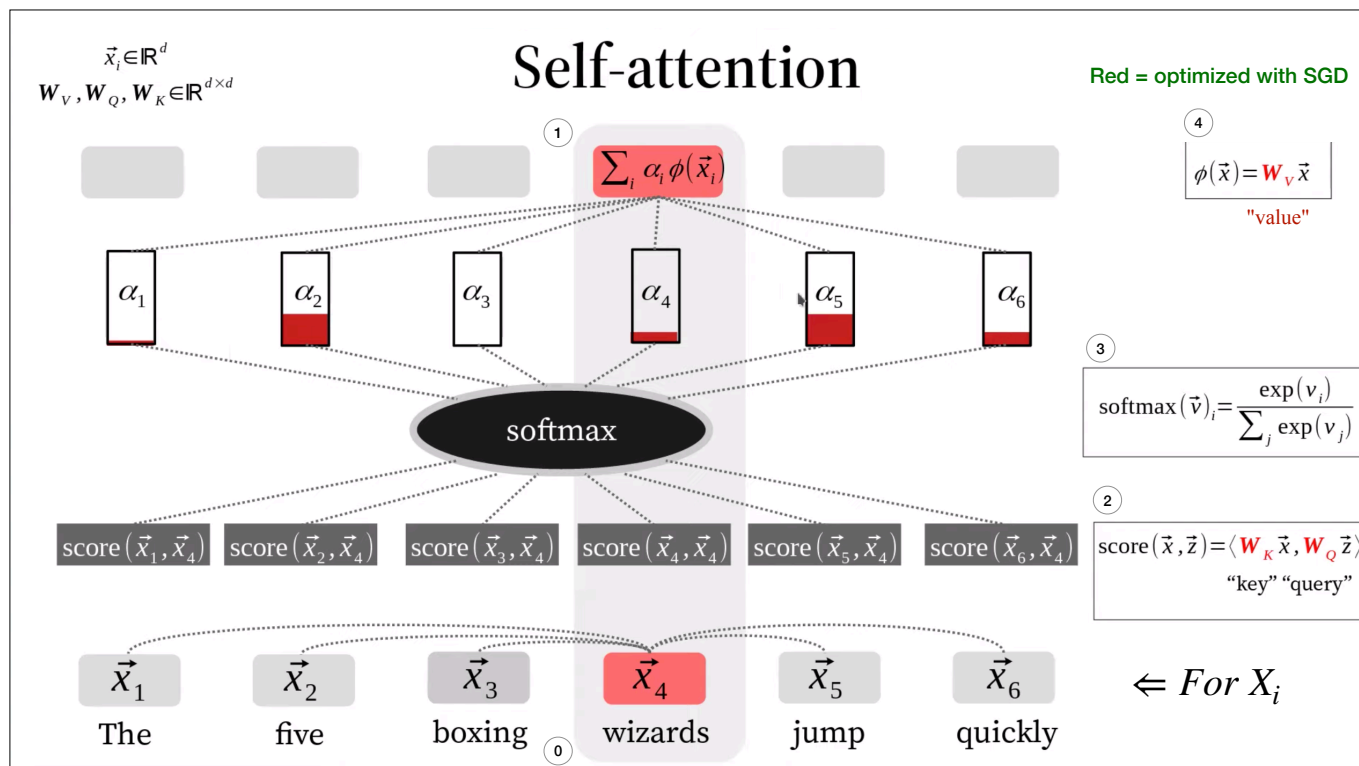
MIT Introduction to Deep Learning

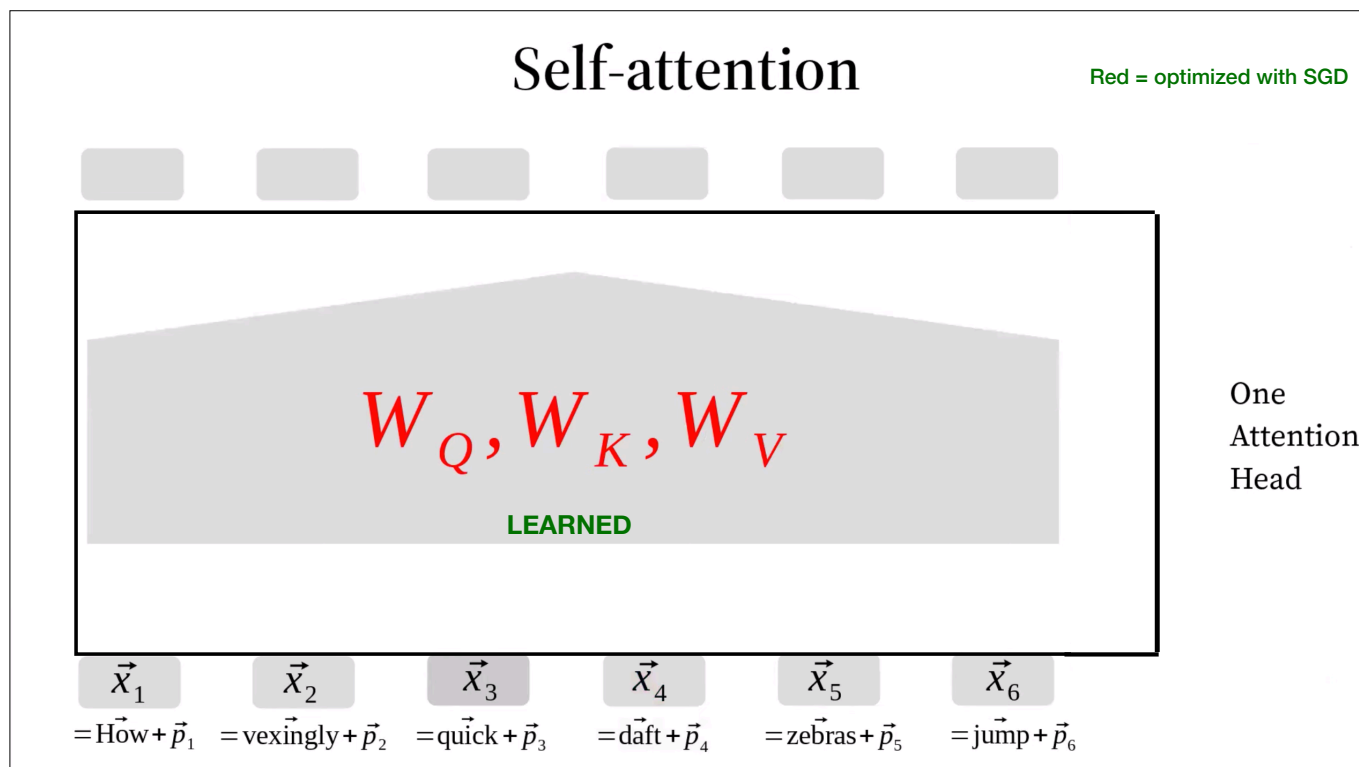
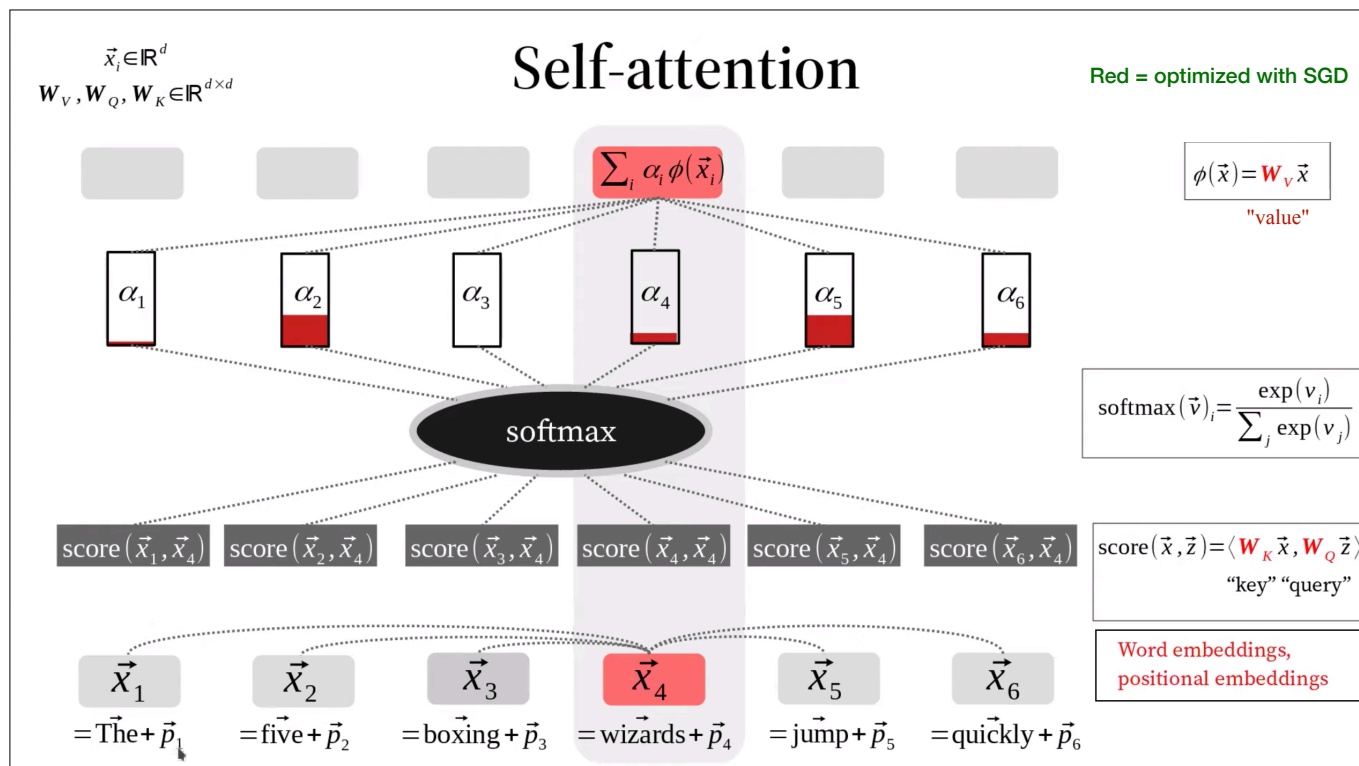
Self-Attention and Transformers

Based on Slides by Ben Edelman (Harvard)

Self-attention





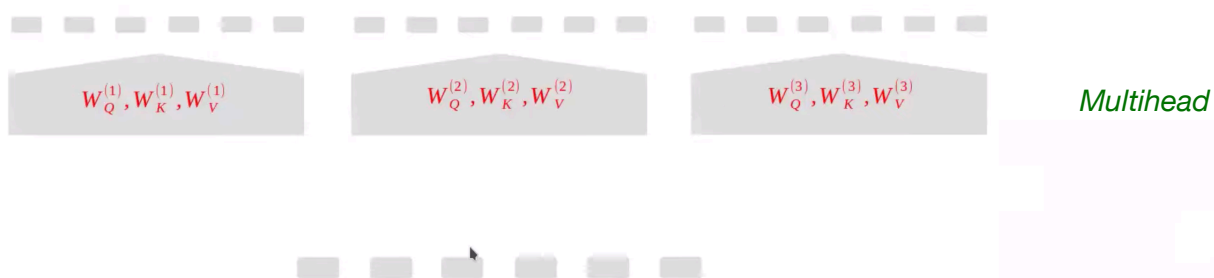


W_Q, W_K, W_V

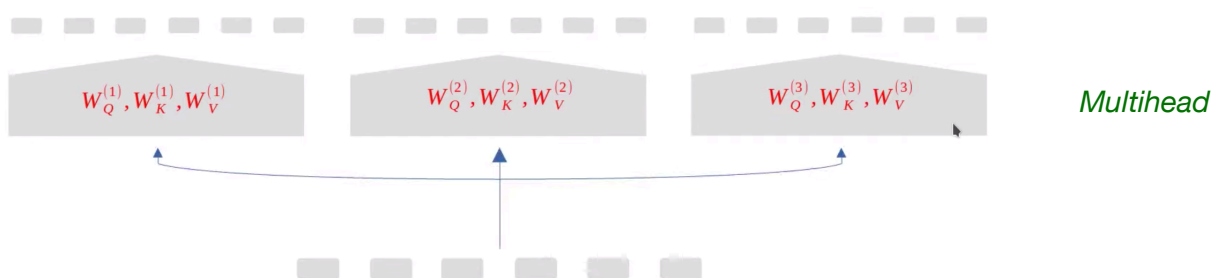
LEARNED

One
Attention
Head

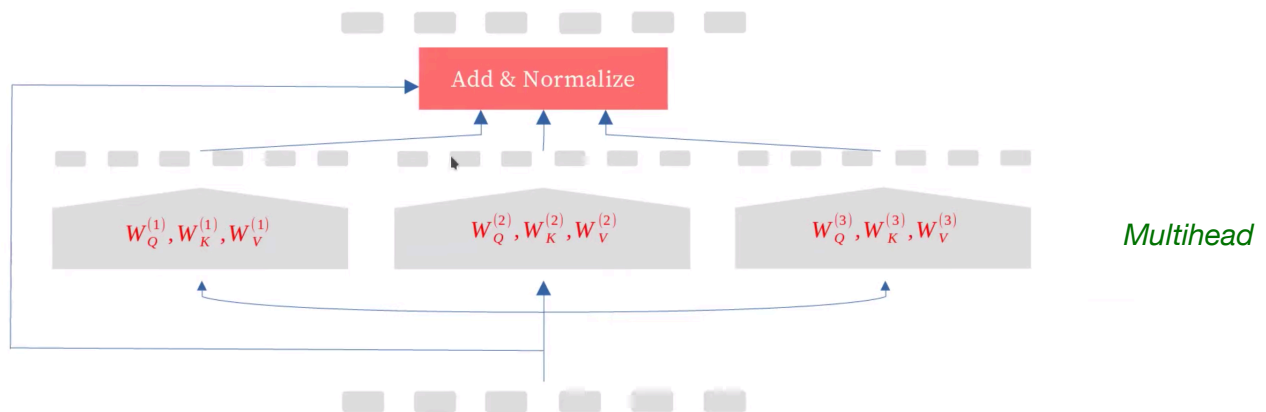
Transformer



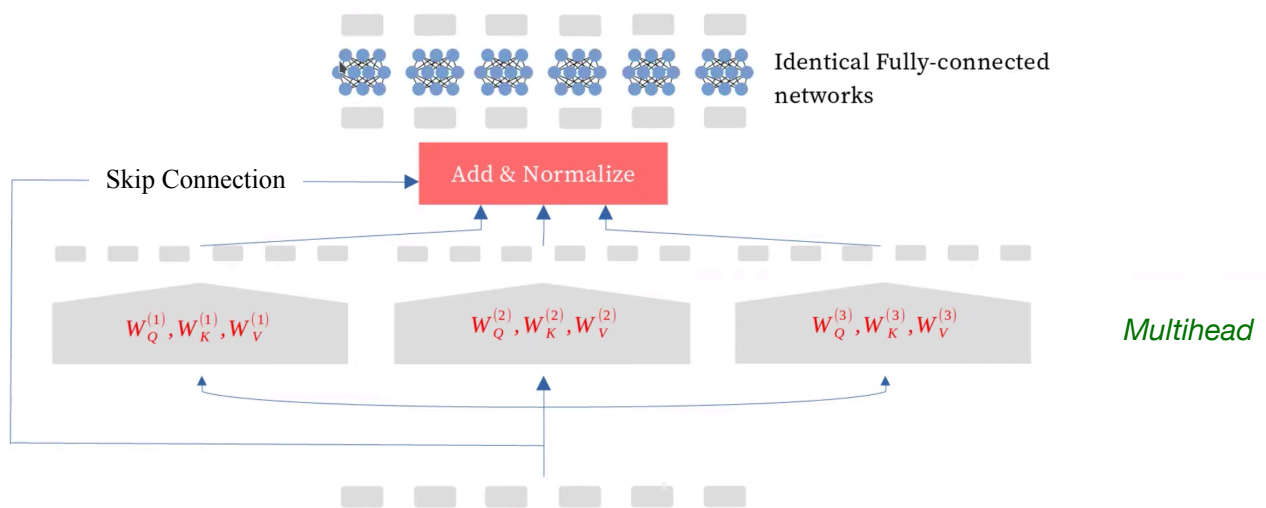
Transformer



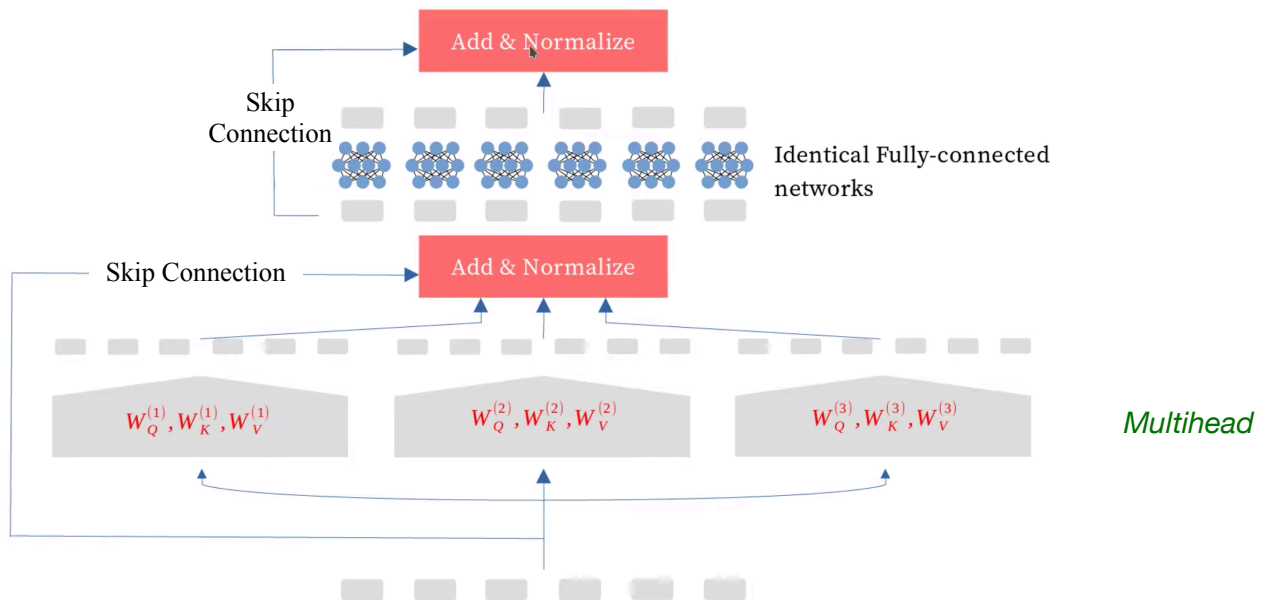
Transformer



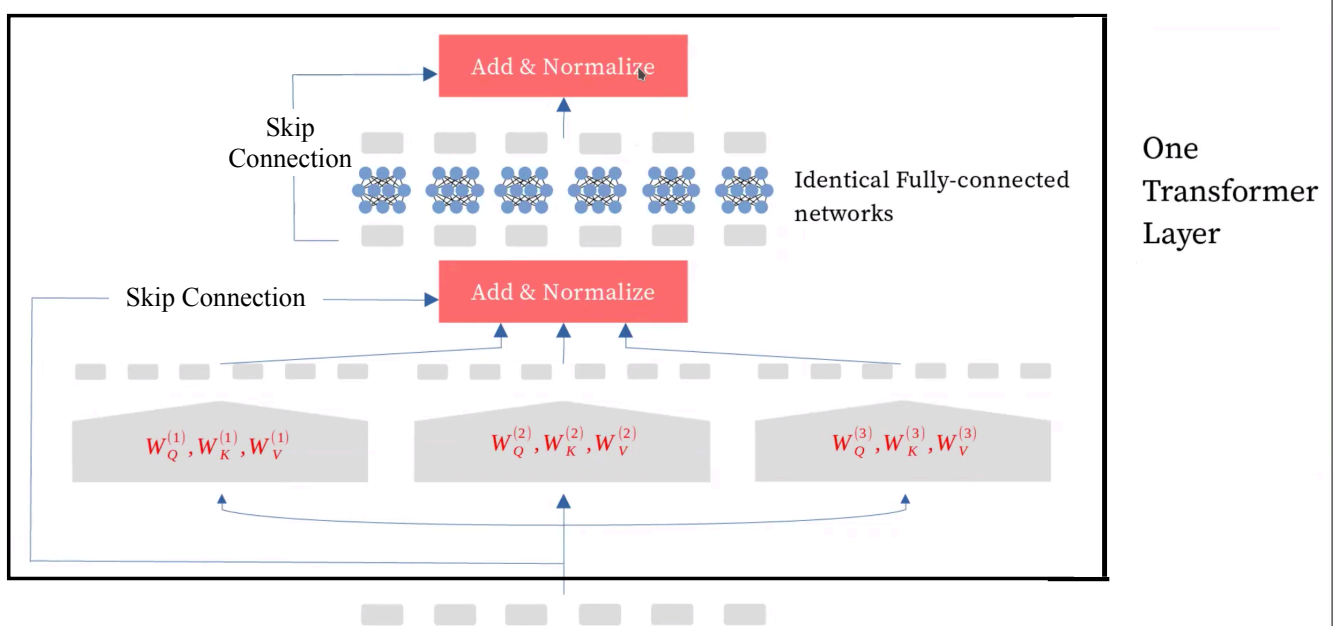
Transformer



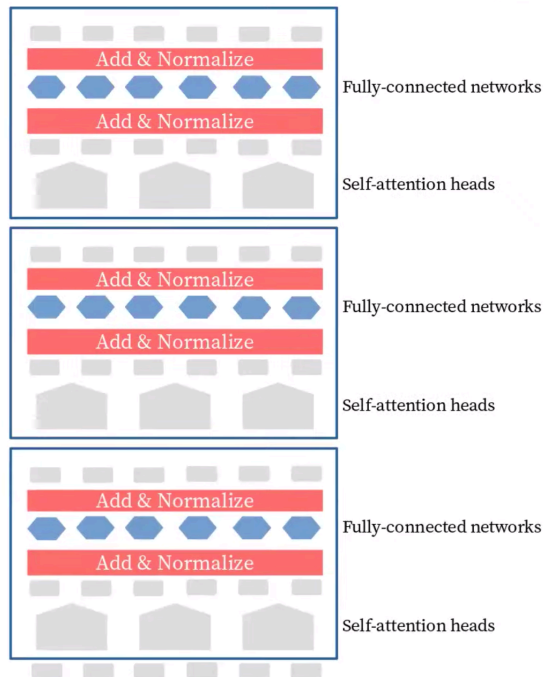
Transformer



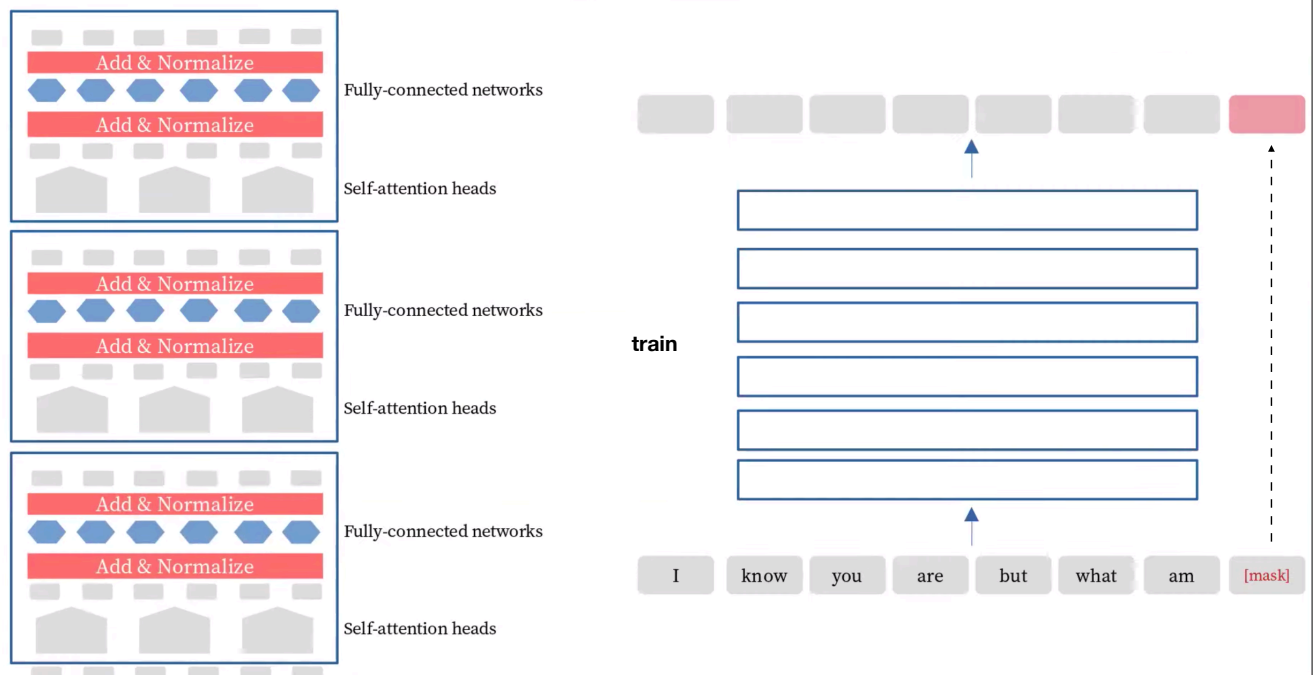
Transformer

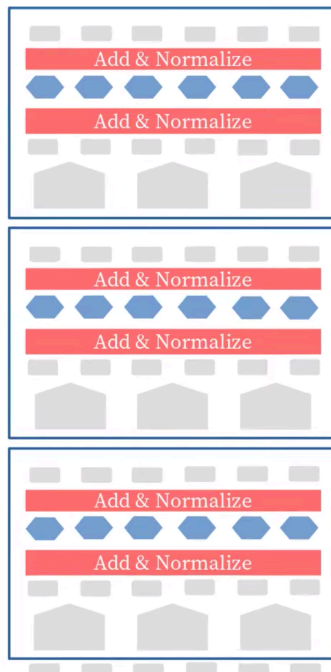


Transformer



Transformer

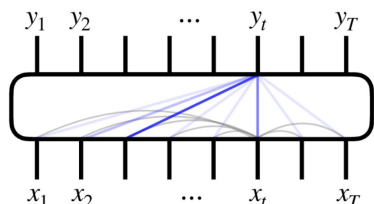




But why do they work?

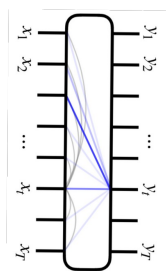
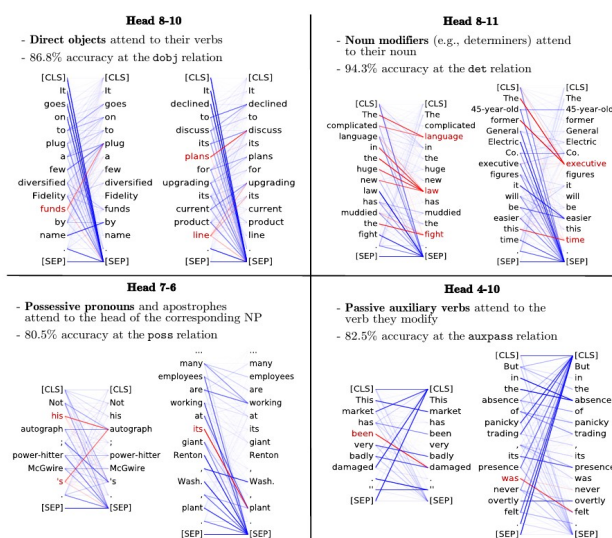
Biases of Attention

Based on Slides by Ben Edelman (Harvard)



What inductive biases does self-attention have?

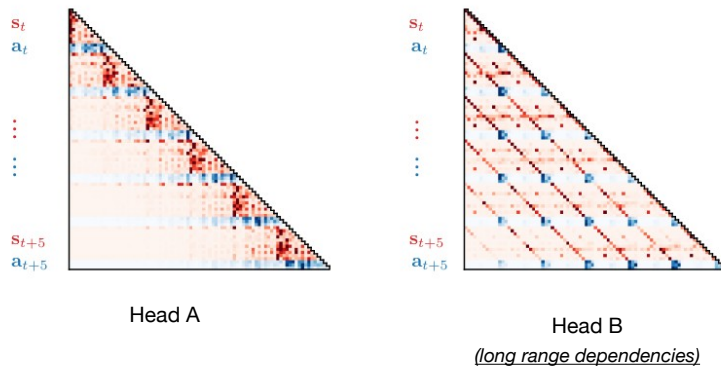
Inductive biases of attention



Attention weights are sparse
(or close to uniform)

Source: "What Does BERT Look At? An Analysis of BERT's Attention"
Clark, Khandelwal, Levy, Manning, 2019

Inductive biases of attention



Attention weights are sparse
(or close to uniform)

Source: "Offline Reinforcement Learning as One Big
Sequence Modeling Problem"
Janner, Li, Levine

Inductive biases of attention

"Inductive Biases and Variable Creation in Self-Attention Mechanisms"

<https://arxiv.org/abs/2110.10090>

With Surbhi Goel, Sham Kakade, & Cyril Zhang

Theory

Experiments

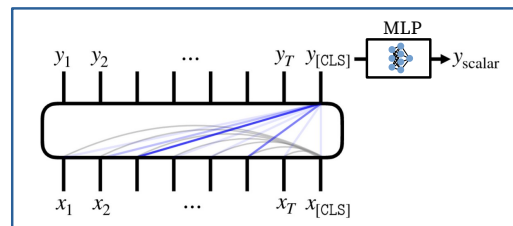
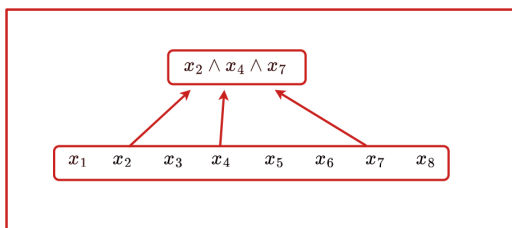
Generalization:

Representation:

Optimization:

Main result: Sparse variable creation

The class of **s-sparse functions of length- T inputs**
can be learned by
the class of **Transformers layers with weight norms $2^{O(s)}$**
with **sample complexity scaling as $\log(T)$** ← **optimal**



Inductive biases of attention

1

Generalization:

Can we prove
Transformers won't
overfit?

2

Representation:

What functions can
Transformers
approximate?

3

Optimization:

Do Transformers
learn these functions
in practice?

Theory

Experiments

1 / 2 - Useful properties of softmax

For generalization bound:

$l_1 \rightarrow l_\infty$ smoothness

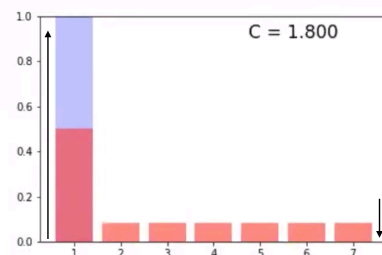
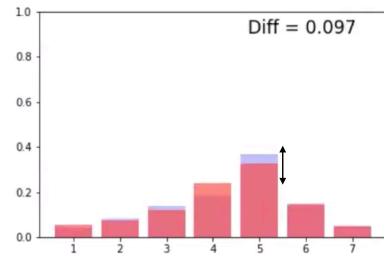
$$\|\text{softmax}(\tilde{z}) - \text{softmax}(\tilde{x})\|_1 \leq 2 \|\tilde{z} - \tilde{x}\|_\infty$$

For representation results:

Exponential saturation

$$\|\text{softmax}(c \tilde{e}_i) - \tilde{e}_i\|_1 \leq \frac{2T}{\exp(c)}$$

$$\text{softmax}(\tilde{v})_i = \frac{\exp(v_i)}{\sum_j \exp(v_j)}$$

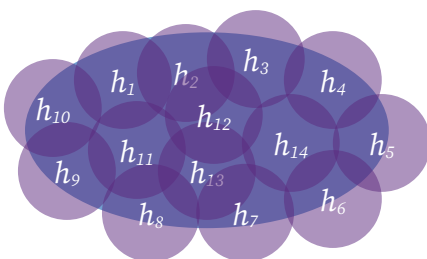


1 - Generalization result

When learning any bounded distribution using a Transformer layer,

$$|\text{error on training set} - \text{error on distribution}| = \tilde{O}\left(\sqrt{\frac{\text{poly}(C) \cdot \log(T)}{m}}\right).$$

Technique: ∞ -covering numbers



Norms of attention
& MLP weight matrices

Number of
training samples

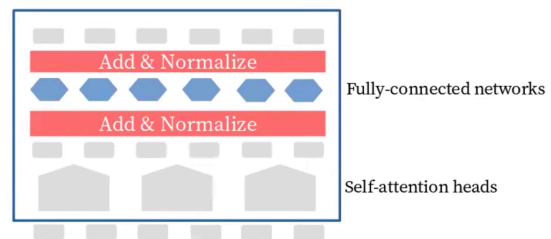
Sequence length

2 - Main representation result

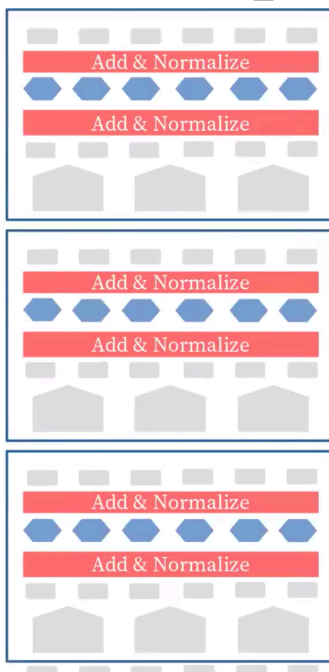
Transformer layers can approximate arbitrary sparse Boolean functions, with norms scaling as a function of the sparsity.

Intuition: Softmax allows sparse variable selection;
fully-connected networks allow arbitrary functions to be applied

Examples: Sparse conjunctions, sparse disjunctions, etc.



2 - Sparse variable creation



Generalization

+

Representation

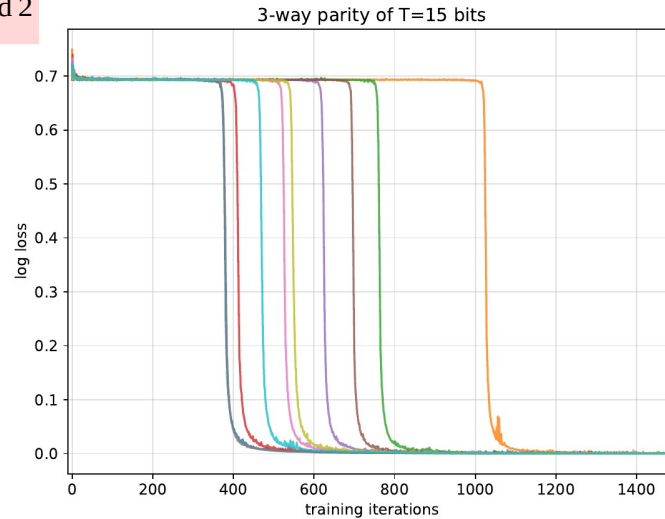
Sample complexity for learning sparse functions scales like:

$$\approx f(\text{\#sparsity}) \cdot \text{polylog}(T)$$

This form is optimal Matches sparse regression.

3 - Transformer layers learn sparse parities

$$\vec{x} \sim \text{unif}([0,1]^T)$$
$$y = x_{i_1} + x_{i_2} + x_{i_3} \bmod 2$$



Related: "Grokking:
Generalization Beyond Overfitting
on Small Algorithmic Datasets"
Power, Burda, Edwards,
Babuschkin, Misra

A Glimpse into the Future

Based on Slides by Ben Edelman (Harvard)

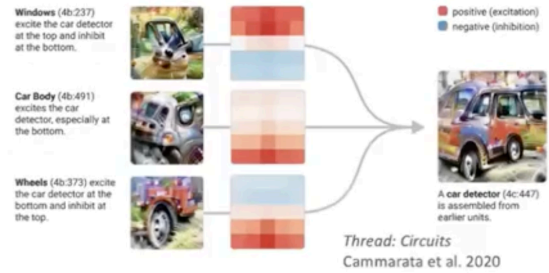
1. Feature learning

Features

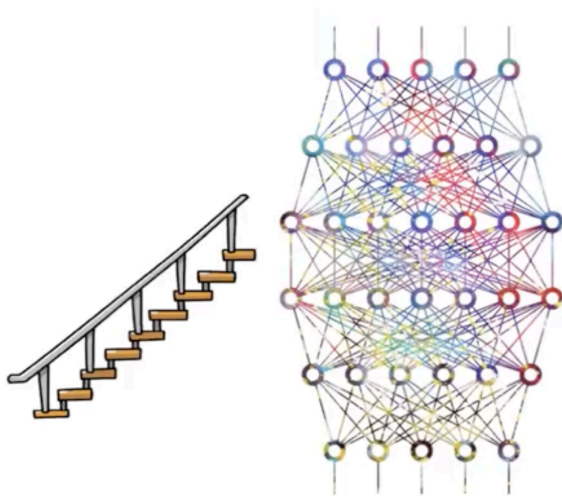


Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks
Lee et al., 2011

Circuits



1. Feature learning



One story:

During the course of training, the network improves its features in a gradual, greedy fashion—each improvement is simple.

But are there situations where there isn't a sequence of locally simple improvements leading to a desired circuit?

2. Learning combinatorial functions

- GPT-3 [Brown et al. '20]
- Codex [Chen et al. '21]
- Minerva [Lewkowycz et al. '22]
-

```

1 import datetime
2
3 def parse_expenses(expenses_string):
4     """Parse the list of expenses and return the list of triples (date, value, currency).
5     Ignore lines starting with #.
6     Parse the date using datetime.
7     Example expenses_string:
8     2016-01-02 -> $4.81 USD
9     2016-01-03 2.59 DKK
10    2016-01-03 -2.72 EUR
11    """
12    expenses = []
13    for line in expenses_string.splitlines():
14        if line.startswith("#"):
15            continue
16        date, value, currency = line.split(" ")
17        expenses.append((datetime.datetime.strptime(date, "%Y-%m-%d"),
18                        float(value),
19                        currency))
20    return expenses

```

Question: A particle moves so that it is at $(3 \sin \frac{t}{4}, 3 \cos \frac{t}{4})$ at time t . Find the speed of the particle, measured in unit of distance per unit of time.

Model output: The speed is the magnitude of the velocity vector. Therefore, we must first find the velocity vector, which is the derivative of the position vector. We have

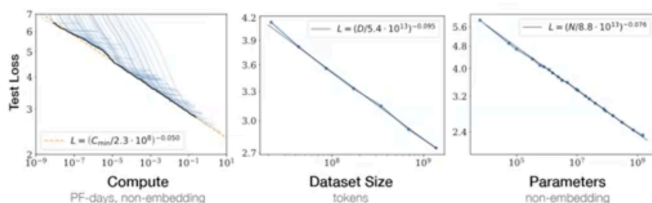
$$\mathbf{v} = \frac{d}{dt} \left(3 \sin \frac{t}{4}, 3 \cos \frac{t}{4} \right) = 3 \left(\cos \frac{t}{4}, -\sin \frac{t}{4} \right) \frac{1}{4} = \left(\frac{3}{4} \cos \frac{t}{4}, -\frac{3}{4} \sin \frac{t}{4} \right).$$

Therefore, the magnitude of the velocity vector is

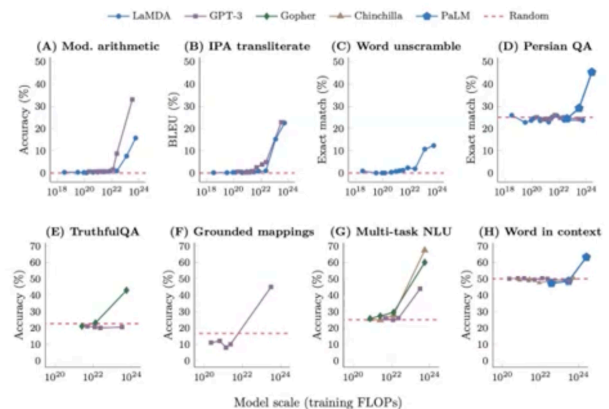
$$|\mathbf{v}| = \sqrt{\left(\frac{3}{4} \cos \frac{t}{4} \right)^2 + \left(-\frac{3}{4} \sin \frac{t}{4} \right)^2} = \sqrt{\frac{9}{16} \cos^2 \frac{t}{4} + \frac{9}{16} \sin^2 \frac{t}{4}} \\ = \frac{3}{4} \sqrt{\cos^2 \frac{t}{4} + \sin^2 \frac{t}{4}} = \frac{3}{4}.$$

Therefore, the speed of the particle is $\boxed{\frac{3}{4}}$.

3. Emergence



Scaling Laws for Neural Language Models
Kaplan et al. 2020



Emergent Abilities of Large Language Models
Wei et al. 2022

Mysteries of contemporary deep learning

1. How do neural networks learn to construct useful features?
2. How do neural networks learn to “reason” / compute “combinatorial” functions?
3. Why are there sometimes emergent breakthroughs in capabilities as resources are scaled up?

In some “combinatorial” tasks, like learning sparse parities, features are only useful when they are learned together and to a sufficient extent. In other words: the network needs to learn from scratch to compute a certain circuit. In these situations, we may see a “phase transition” in the loss curve, even though there is hidden progress inside the black box.

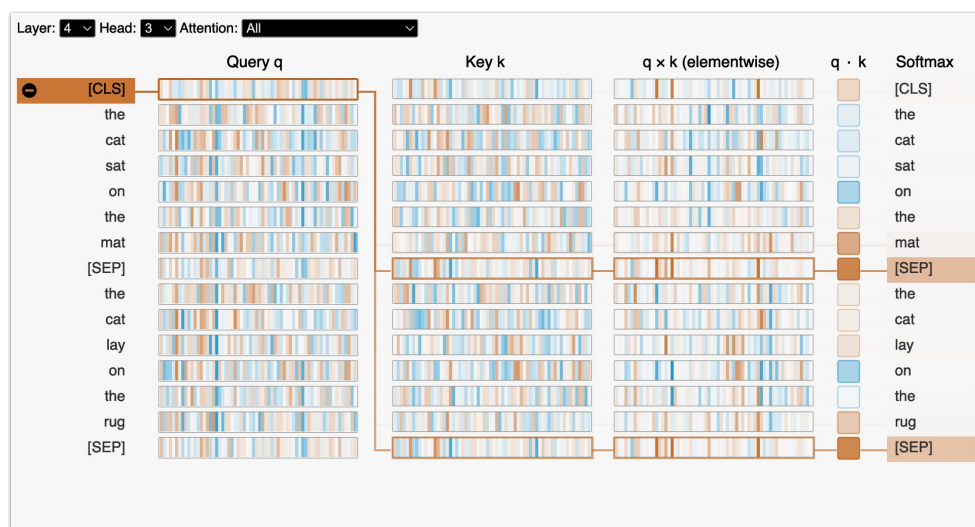
Still mysteries

1. How do neural networks learn to hierarchically construct useful features?
2. How do neural networks learn to “reason” / compute complex “combinatorial” functions?
3. Why are there sometimes emergent breakthroughs in capabilities as resources like network size are scaled up?

Seeing and Believing

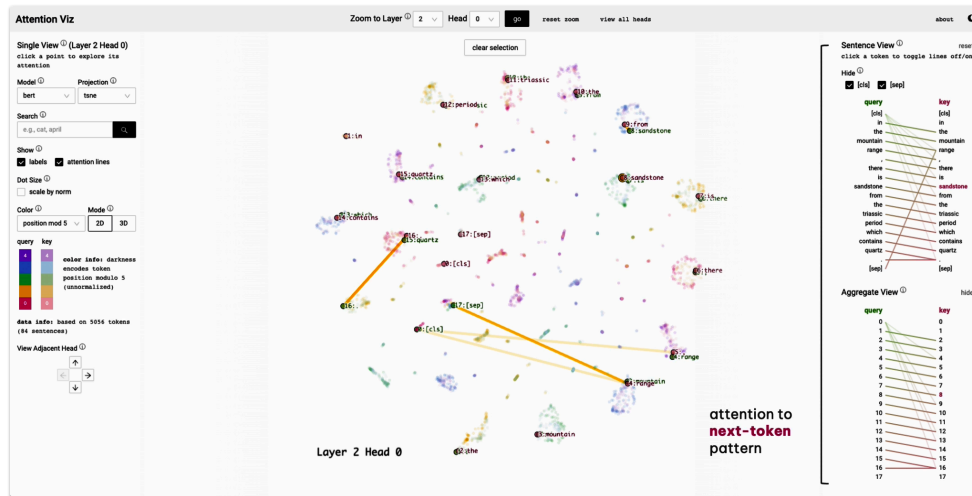
- Visualization Tools
 - Bertviz
 - AttentionViz
 - Induction Heads

Bertviz



- <https://github.com/jessevig/bertviz>

AttentionViz



- <https://catherinesyeh.github.io/attn-docs/>