The Transformer Model

Anoop Kunchukuttan
Microsoft AI Core, Hyderabad
anoop.kunchukuttan@gmail.com

For CS772 at IIT Bombay, Sep 2025

Course Instructor: Prof. Pushpak Bhattacharyya

Outline

- Recap
- Transformer: Introduction
- Transformer: Motivation
- Deep-dive into transformers
    - Key Ideas
    - Important components: Residual connections, Layer Normalization
    - Putting the whole thing together
- Self-study
- Assignment
- Additional Topics

Recap

- Discussed tasks which require processing text sequences
  - Language Modeling
  - Sequence Labeling Tasks
  - Sequence to Sequence Tasks
- Techniques
  - Feed-forward LMs
  - RNN LMs
  - LSTM LMs
  - Encoder-Decoder models
  - Attention Mechanism

Transformer – revolutionizing NLP and AI

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

*More than **200k** citations*

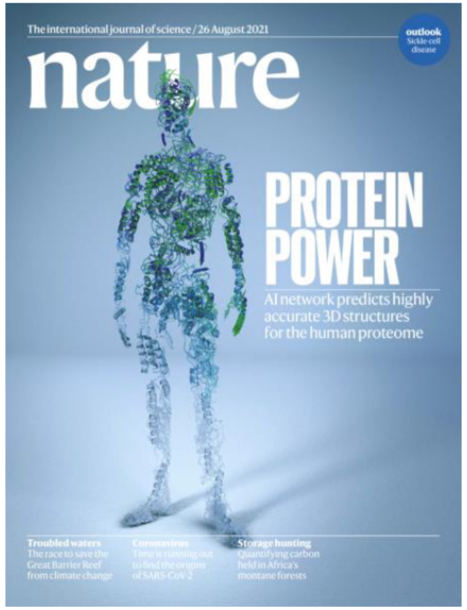
Made AI more scalable

Building complex systems from simple building blocks composed in a scalable way

Underpins all NLP models: BERT, GPT, Claude, Gemini, translation models, etc.

Large impact outside of NLP – vision transformers, protein folding research

Protein Folding



[Jumper et al. 2021] aka AlphaFold2!

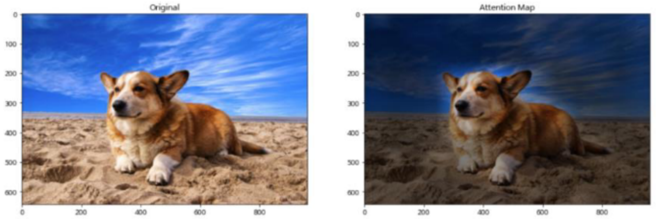
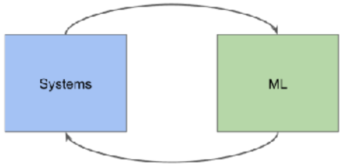
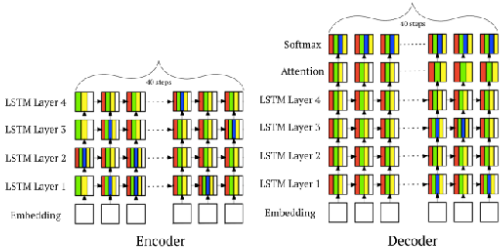


Image Classification

[Dosovitskiy et al. 2020]: Vision Transformer (ViT) outperforms ResNet-based baselines with substantially less compute.

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	**88.55** ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02	88.4/88.5*
ImageNet ReaL	**90.72** ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54	90.55
CIFAR-10	**99.50** ± 0.06	99.42 ± 0.03	99.15 ± 0.03	99.37 ± 0.06	–
CIFAR-100	**94.55** ± 0.04	93.90 ± 0.05	93.25 ± 0.05	93.51 ± 0.08	–
Oxford-IIIT Pets	**97.56** ± 0.03	97.32 ± 0.11	94.67 ± 0.15	96.62 ± 0.23	–
Oxford Flowers-102	99.68 ± 0.02	**99.74** ± 0.00	99.61 ± 0.02	99.63 ± 0.03	–
VTAB (19 tasks)	**77.63** ± 0.23	76.28 ± 0.46	72.72 ± 0.21	76.29 ± 1.70	–
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k





ML for Systems

[Zhou et al. 2020]: A Transformer-based compiler model (GO-one) speeds up a Transformer model!

Model (#devices)	GO-one (s)	HP (s)	METIS (s)	HDP (s)	Run time speed up over HP / HDP	Search speed up over HDP
2-layer RNNLM (2)	0.173	0.192	0.355	0.191	9.9% / 9.4%	2.95x
4-layer RNNLM (4)	0.210	0.239	0.503	0.251	13.8% / 16.3%	1.76x
8-layer RNNLM (8)	0.320	0.332	OOM	0.764	3.8% / 58.1%	27.8x
2-layer GNMT (2)	0.301	0.384	0.344	0.327	27.6% / 14.3%	30x
4-layer GNMT (4)	0.350	0.469	0.466	0.432	34% / 23.4%	58.8x
[illegible]	0.440	0.562	OOM	0.693	21.7% / 36.5%	7.35x
2-layer Transformer-XL (2)	0.223	0.268	0.37	0.262	20.1% / 17.4%	40x
4-layer Transformer-XL (4)	0.230	0.27	OOM	0.259	17.4% / 12.6%	26.7x
8-layer Transformer-XL (8)	0.350	0.46	OOM	0.425	23.9% / 16.7%	16.7x
[illegible]	0.229	0.312	OOM	0.301	26.6% / 23.9%	13.5x
Inception (2) b64	0.423	0.731	OOM	0.498	42.1% / 29.3%	21.0x
AmoebaNet (4)	0.394	0.44	0.426	0.418	26.1% / 6.1%	58.8x
2-stack 18-layer WaveNet (2)	0.317	0.376	OOM	0.354	18.6% / 11.7%	6.67x
4-stack 36-layer WaveNet (4)	0.659	0.988	OOM	0.721	50% / 9.4%	20x
GEOMEAN	-	-	-	-	**20.5% / 18.2%**	**15x**

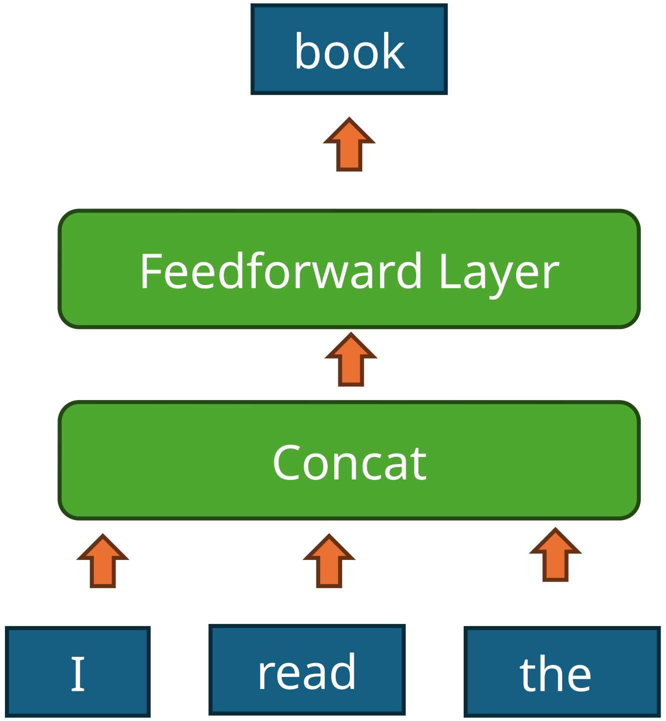
Absolutely fundamental topic

Lot of cutting-edge work is based on deep understanding of transformers

Understand the ins and outs – lots of resources available

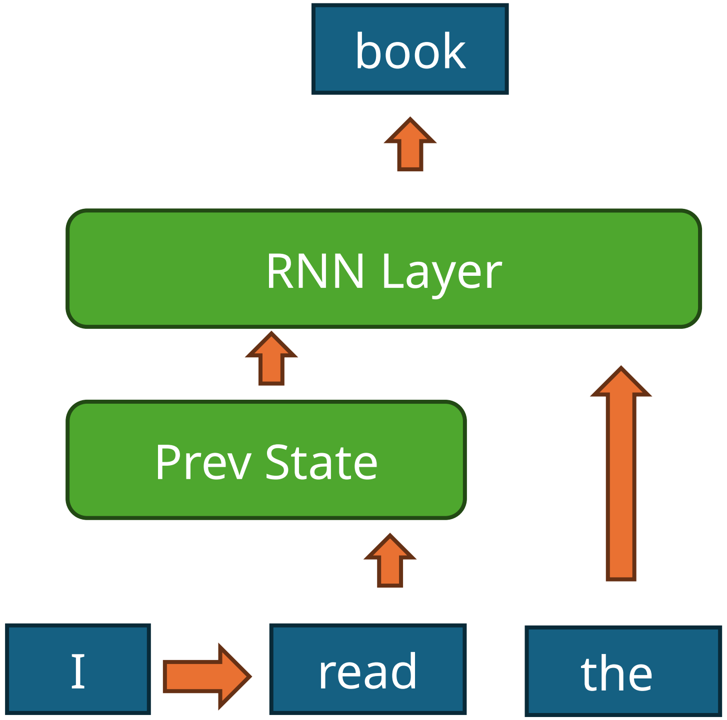
Sequence Modeling Approaches

Feedforward LLM



Not scalable to long contexts

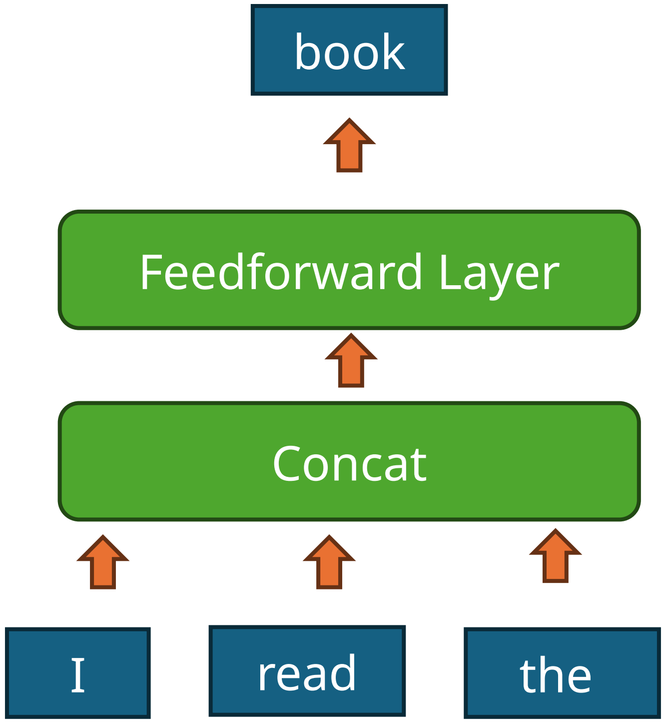
RNN/LSTM LLM



Context captured in a single state vector via recurrence

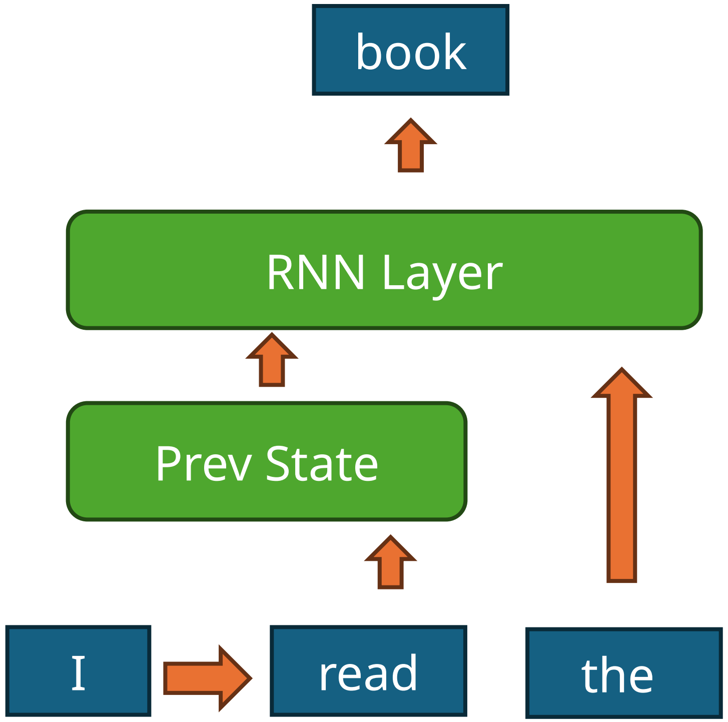
Limitations of RNN-based approaches

Feedforward LLM



Not scalable to long contexts

RNN/LSTM LLM



Context captured in a single state vector via recurrence

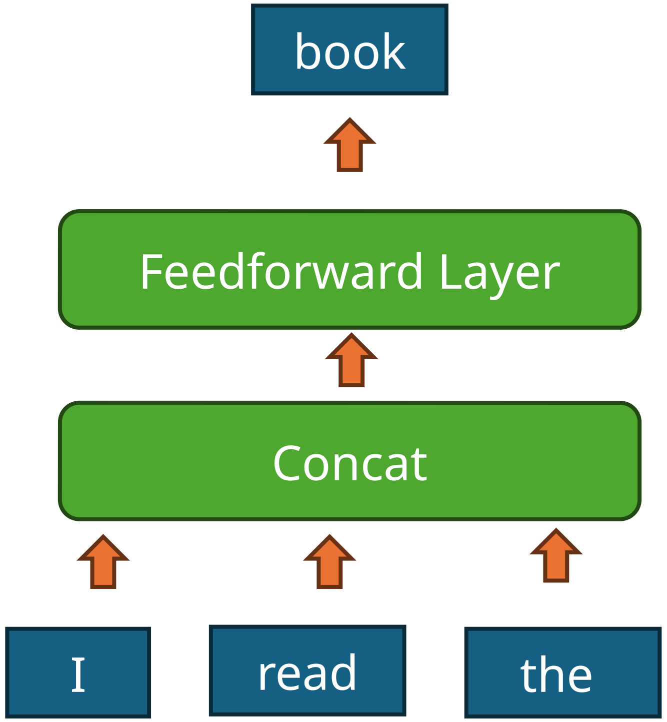
Long-distance dependency paths

Sequential Processing

Unidirectional Processing

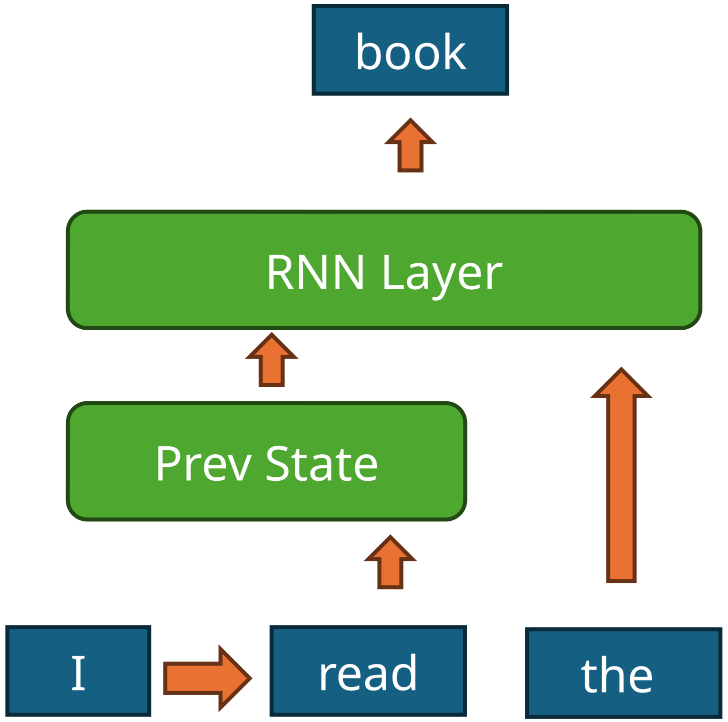
Can we use ideas from attention to overcome these limitations?

Feedforward LLM



Not scalable to long contexts

RNN/LSTM LLM



Context captured in a single state vector via recurrence

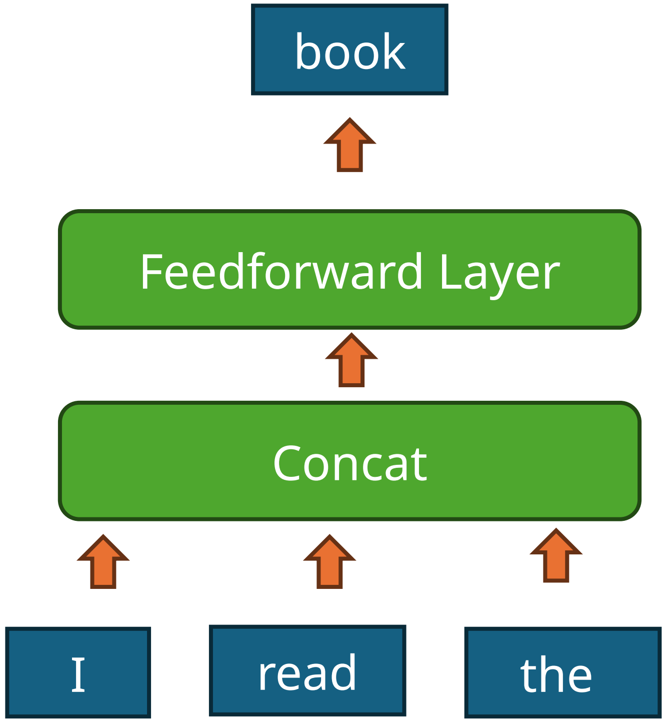
The RNN State vector is a representation of the entire context

From attention over encoders, we know that attention mechanism can also be used to build source representations relevant to target token

Can we use the same idea within the same sequence – aka SELF-attention?

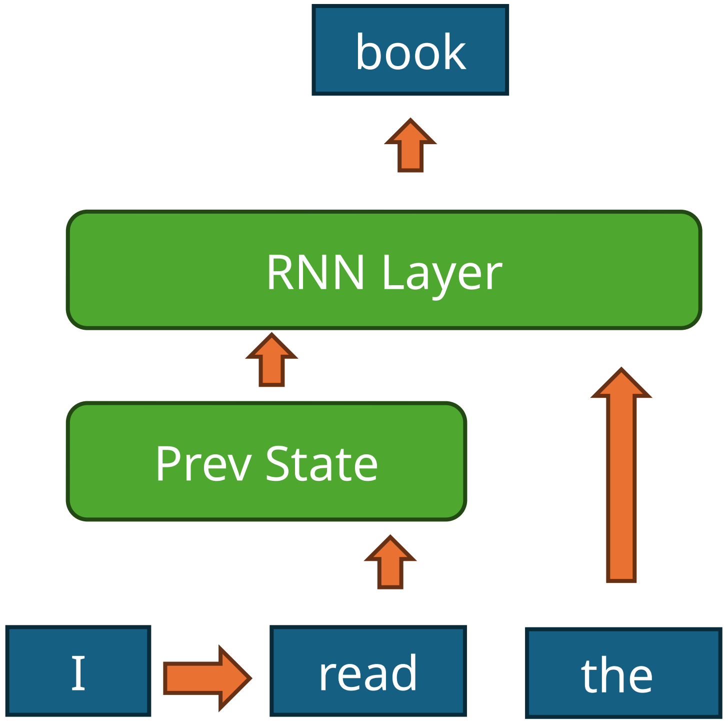
Can we use ideas from attention to overcome these limitations?

Feedforward LLM



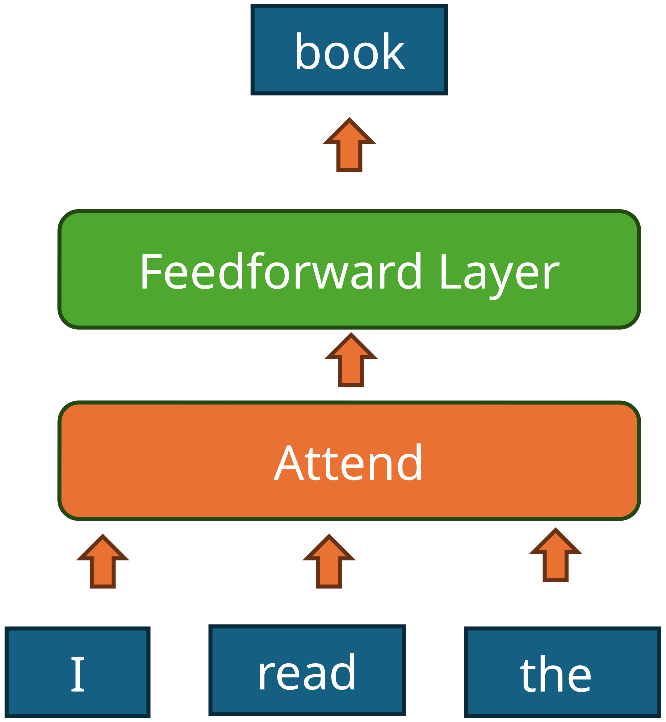
Not scalable to long contexts

RNN/LSTM LLM



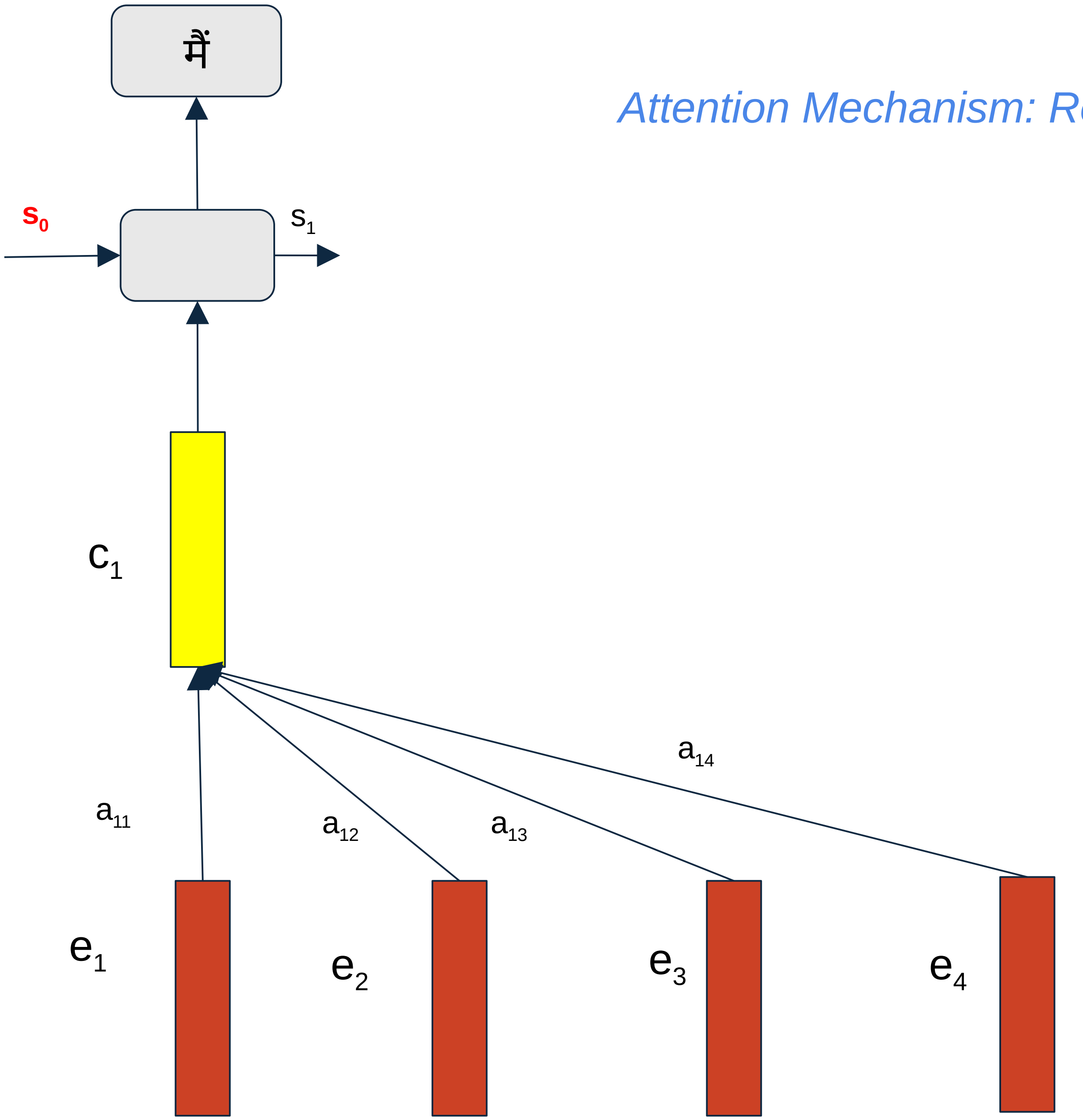
Context captured in a single state vector via recurrence

Self-Attention



Context captured in a single summary vector via attention

Attention Mechanism: Recap



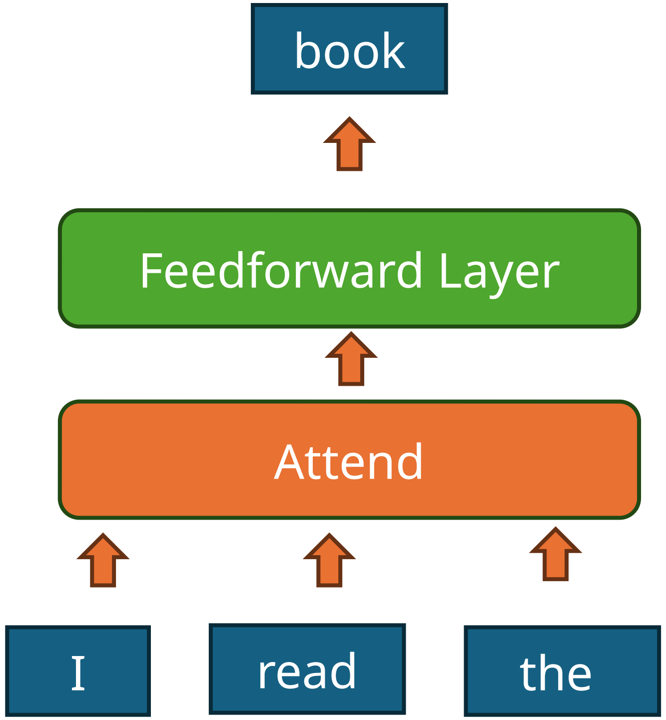
$$\alpha_{ij} = g(s_{j-1}, e_i, \mathbf{emb}(y_{j-1}))$$

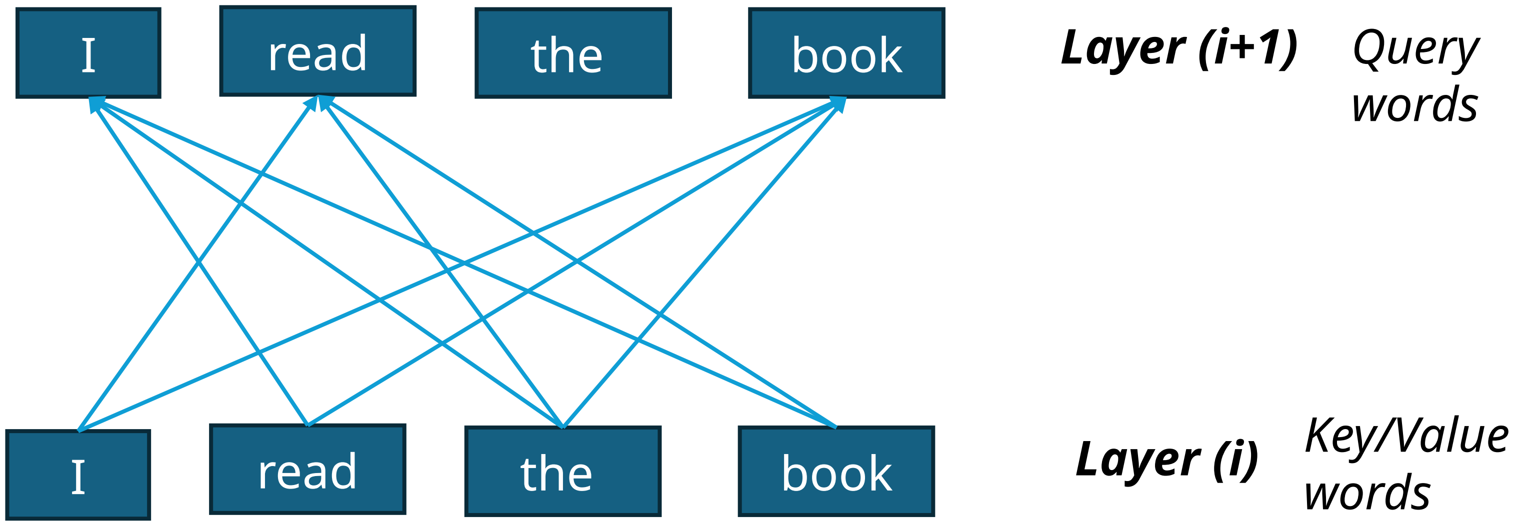
$$a_{ij} = \frac{\exp(\alpha_{ij})}{\sum_{i=1}^{i=N} \exp(\alpha_{kj})}$$

$$c_j = \sum_{i=1}^{i=N} a_{ij} e_i$$

For generation of i^{th} output character:
c_i : context vector
a_{ij} : annotation weight for the j^{th} annotation vector
e_j: j^{th} annotation vector

Self-Attention





Context captured in a single summary vector via attention

Attention over words in the same sequence

Attention over what?
Representations from previous layer

Dependency path length is minimized to 1
- All words impact the target word

All words can be processed in parallel

Bidirectional context can be naturally modelled
Models can be more interpretable

A method to obtain contextualized distributed representations

Is Attention All You Need?

There is quite a bit of machinery & trick in getting this to work at scale

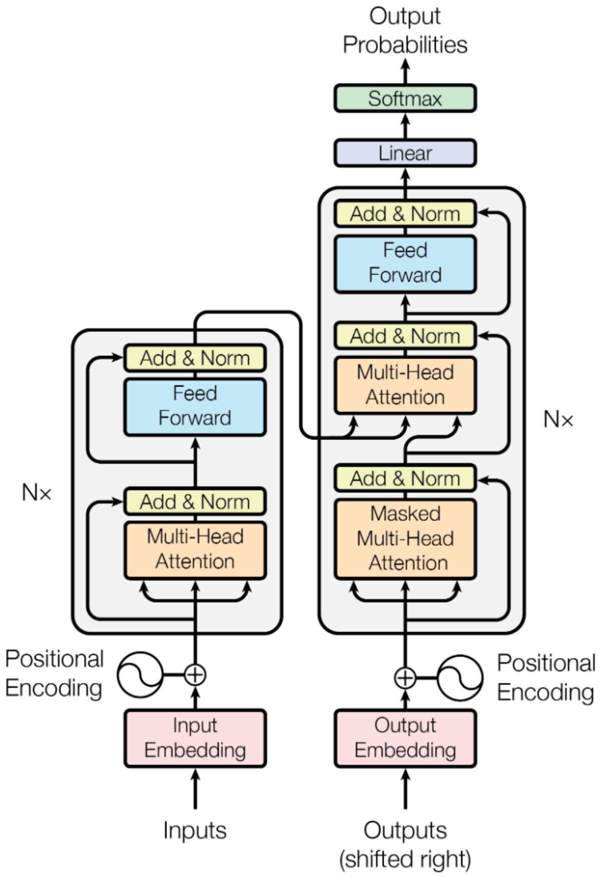


Figure 1: The Transformer - model architecture.

Let's dive into the details

Transformer - Overview

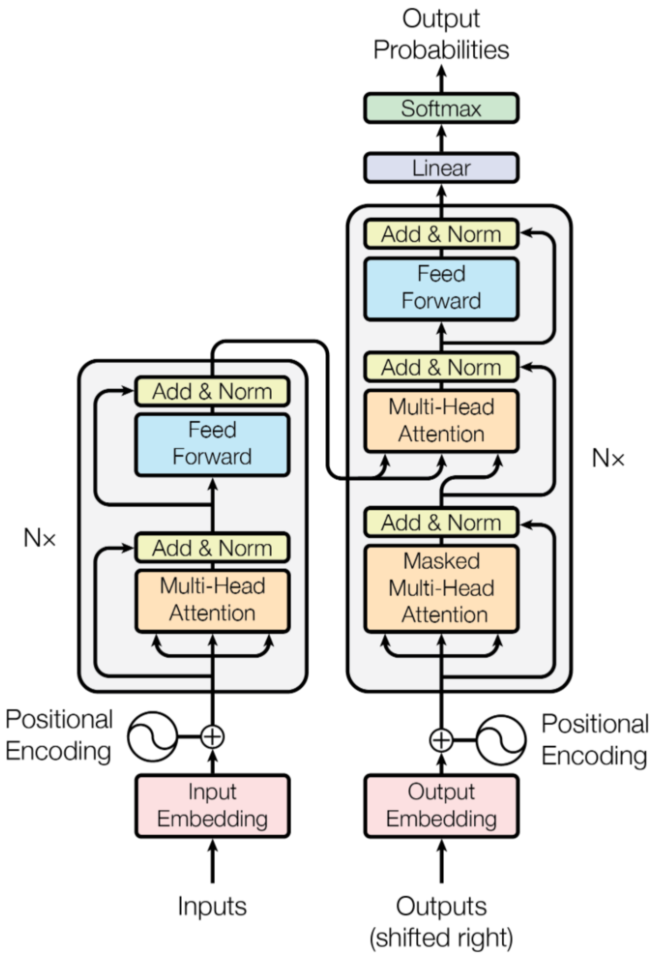


Figure 1: The Transformer - model architecture.

- Encoder-Decoder model for **sequence-to-sequence modeling**
- **Stacked** layers of transformer blocks in both encoder and decoder
- Encoder and decoder blocks are largely **similar**
  - Decoder has an additional attention module
- Each block is a combination of **self-attention** and feed-forward modules
- **Other key ideas**
  - Multi-head Attention
  - Positional Embeddings
  - Layer Norm
  - Residual Connections

Self-Attention Explored

- Step 1: For each word x_i , calculate its **query**, **key**, and **value**.

  $q_i = W^Q x_i \quad k_i = W^K x_i \quad v_i = W^V x_i$

- Step 2: Calculate attention score between **query** **and** **keys.**

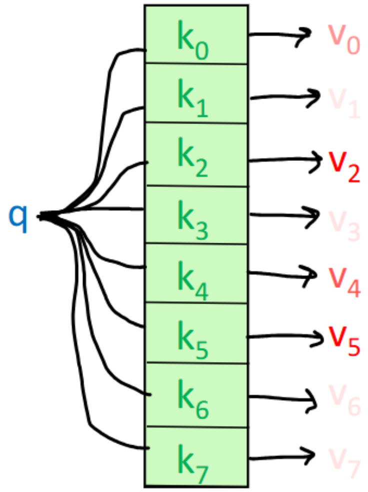
  $e_{ij} = q_i \cdot k_j$

- Step 3: Take the softmax to normalize attention scores.

  $$\alpha_{ij} = softmax(e_{ij}) = \frac{exp(e_{ij})}{\sum_k exp(e_{ik})}$$

- Step 4: Take a weighted sum of **values**.

  $$Output_i = \sum_j \alpha_{ij} v_j$$



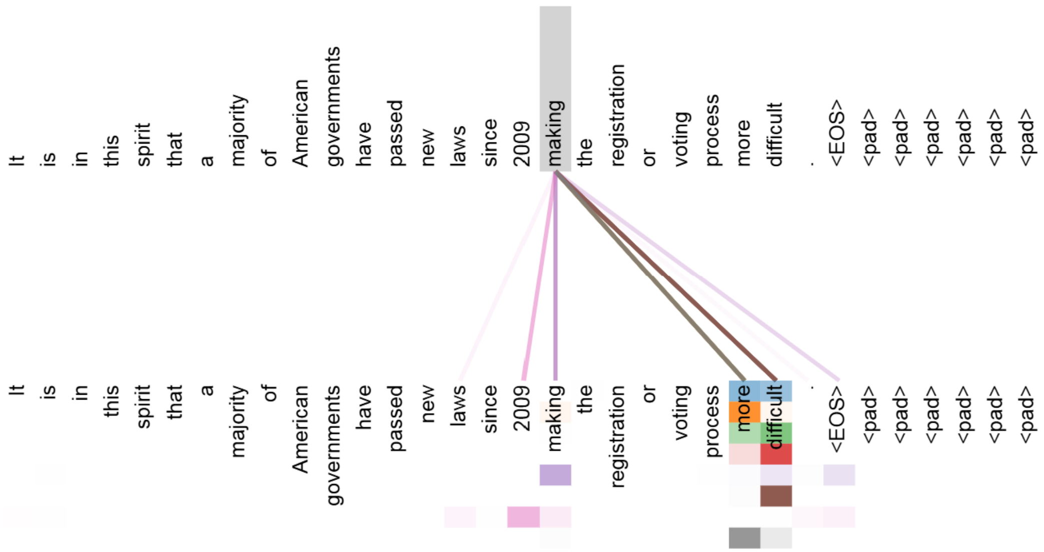
W^Q, W^K, W^V allow you to look at same input vector from 3 different perspectives

Modify attention by scaling dot-product

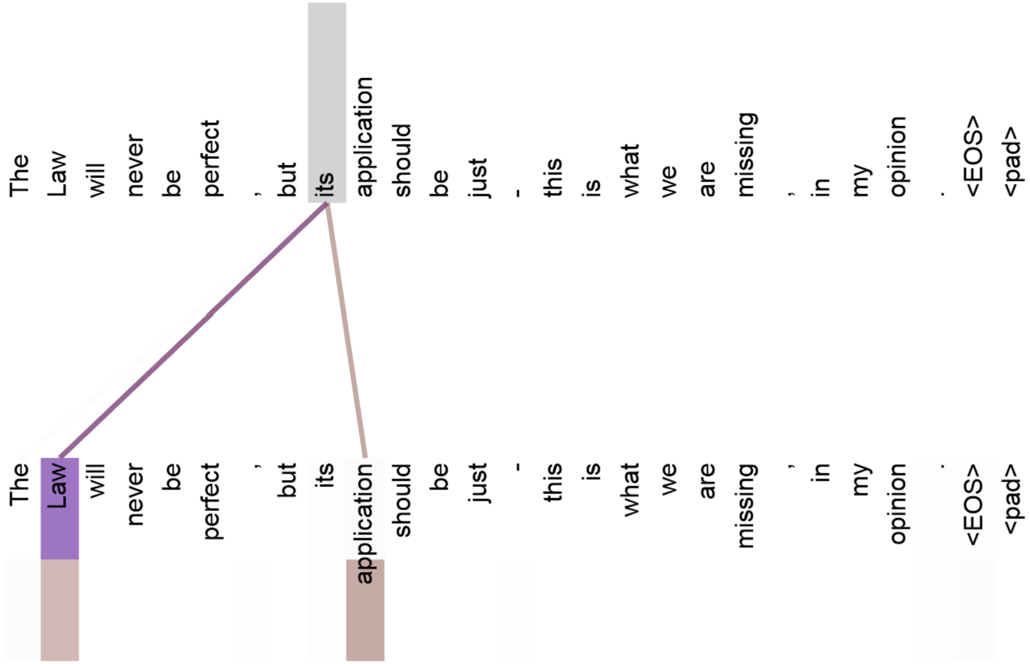
$e_{ij} = (q_i . k_j)/\sqrt{d_k}$ ⟶ *To avoid dot product taking extreme values, as its variance scales with dimensionality d_k*

Source: Stanford CS224n

Self-Attention Visualized



Capturing dependency relationships



Capturing anaphora

The model learns to focus on the right context dynamically when processing a sequence

Multi-head Attention - Motivation

We want to look at word and their relations from different perspectives

e.g.

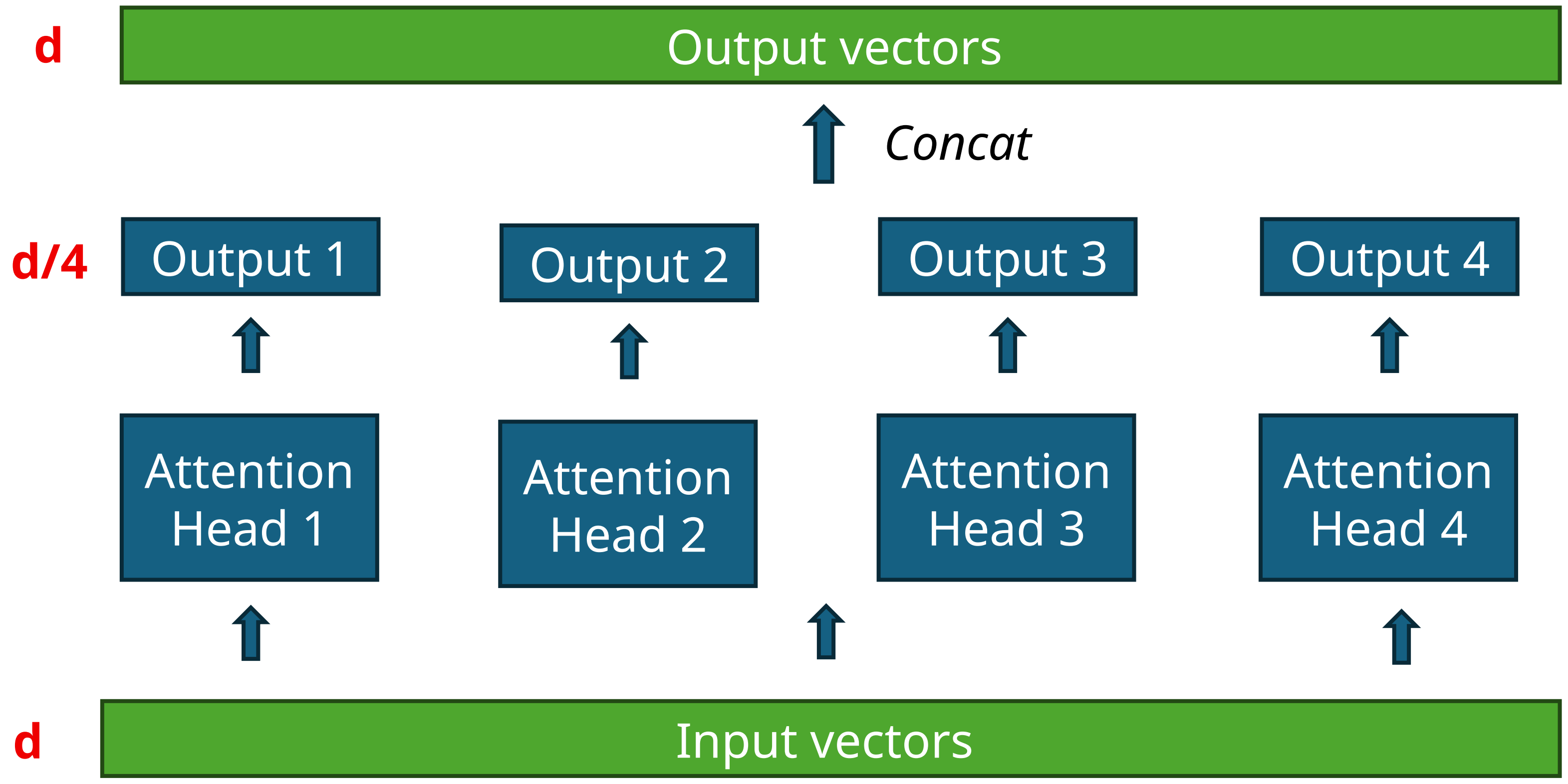
- Dependency relations
- Anaphora
- Structural relations
- Semantic similarity

How can we capture different kinds of relations between words?

⇩

Use Multiple Heads

Multi-head Attention (MHA) - Implementation



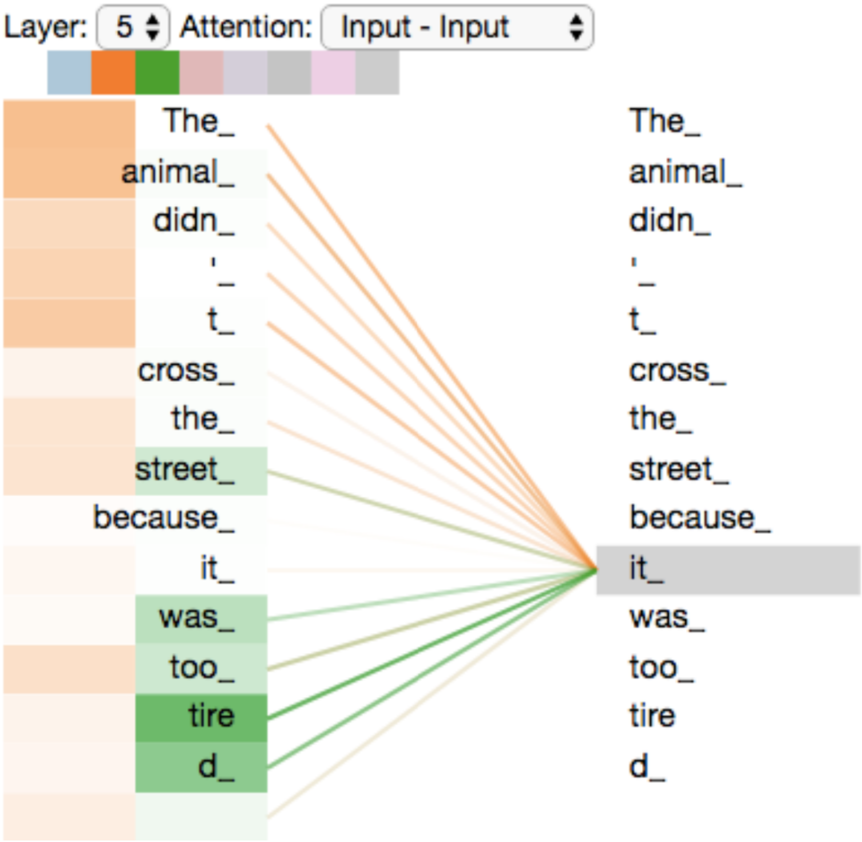
Each token is looked at by multiple heads independently.

Each head has its in own W^Q, W^K, W^V parameters.

Each attention head generates part of the final output vector

Final output vector is a concatenation of each head's output

Multi-head Attention (MHA) - Visualized



As we encode the word "it", one attention head (orange) is focusing most on "the animal",

Another (green) is focusing on "tired" -- in a sense, the model's representation of the word "it" bakes in some of the representation of both "animal" and "tired".

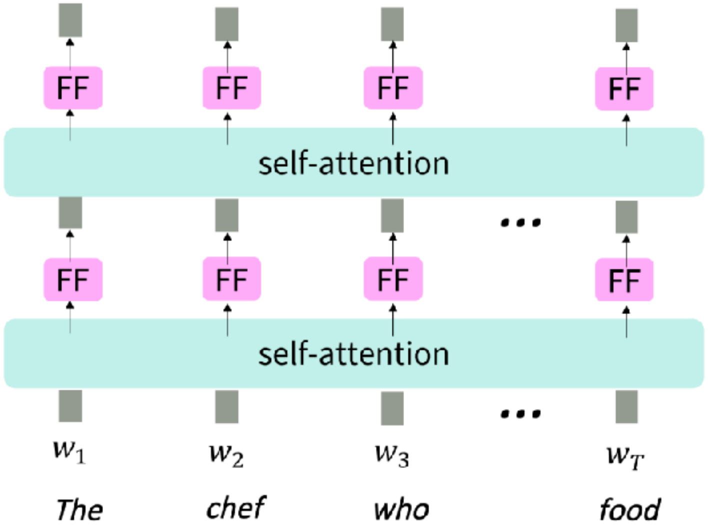
One is capturing anaphora, the other is capturing dependency

Source: [Illustrated Transformer]()

Feedforward Module

- Attention Module helps build distributed representations of words
- However, its expressive power is limited
  - Simple averaging of vectors ➔ linear operation
  - No non-linearies ➔ limited learning (remember Perceptron!)

$$
\begin{aligned}
m_i &= MLP(\text{output}_i) \\
&= W_2 * \text{ReLU}(W_1 \times \text{output}_i + b_1) + b_2
\end{aligned}
$$



Up projection (~4x) ➔ non-linearity ➔ Down-projection

Adds lots of parameters, typically a large fraction of parameter budget goes to feedforward module

Note:

Self-attention looks at the entire sequence at a time,

Generates a distributed word representation

The feed-forward module looks each representation independently – applied in parallel

Problem – No Structural/Positional Awareness

The **dog** barks at the man

The man barks at the **dog**

Dog has the same representation irrespective of its role in the sentence

*Same for **man***

The old **boy** taught the little **boy**

*The two **boys** have the same representation, although they have different roles*

The self-attention and feed-forward computations are position invariant

Positional Embeddings

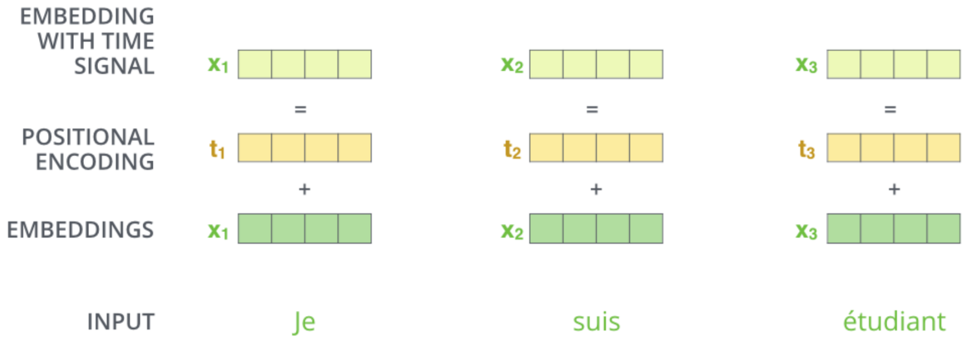
We need to encoder position information into our word representations

Add a position-specific embedding to each token

Typically, down at the embedding layer

$$\widehat{x_i} = x_i + p_i$$

where p_i is the unique embedding vector to identify position i

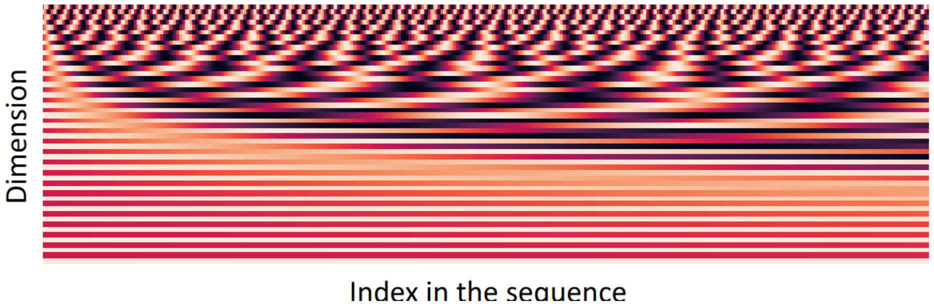


Learned Embeddings

Positional embeddings are additional parameters that are learr

Sinusoidal Embeddings

$$p_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$



Where are we?

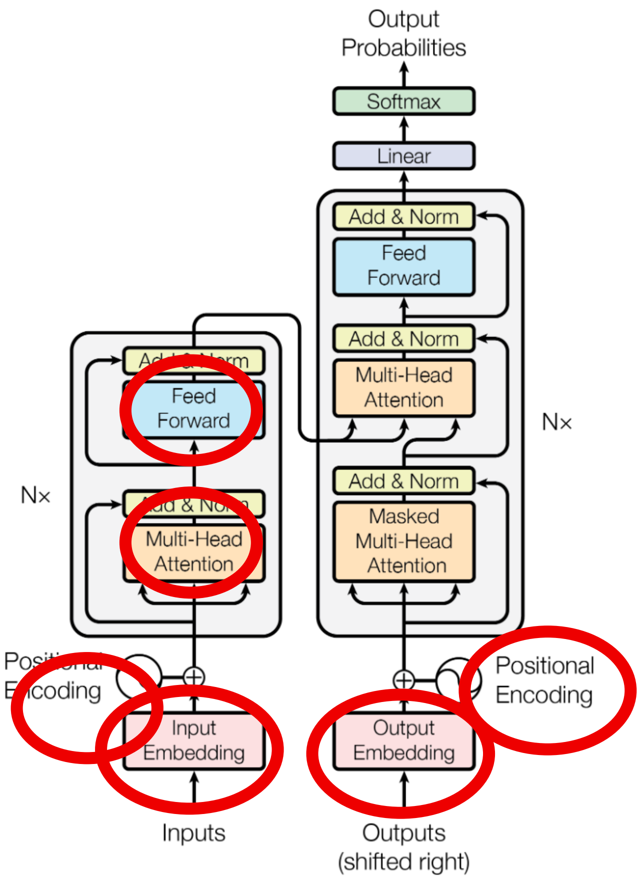


Figure 1: The Transformer - model architecture.

We have covered the most important concepts

Let's fill in the rest

Let’s look at the decoder side of things

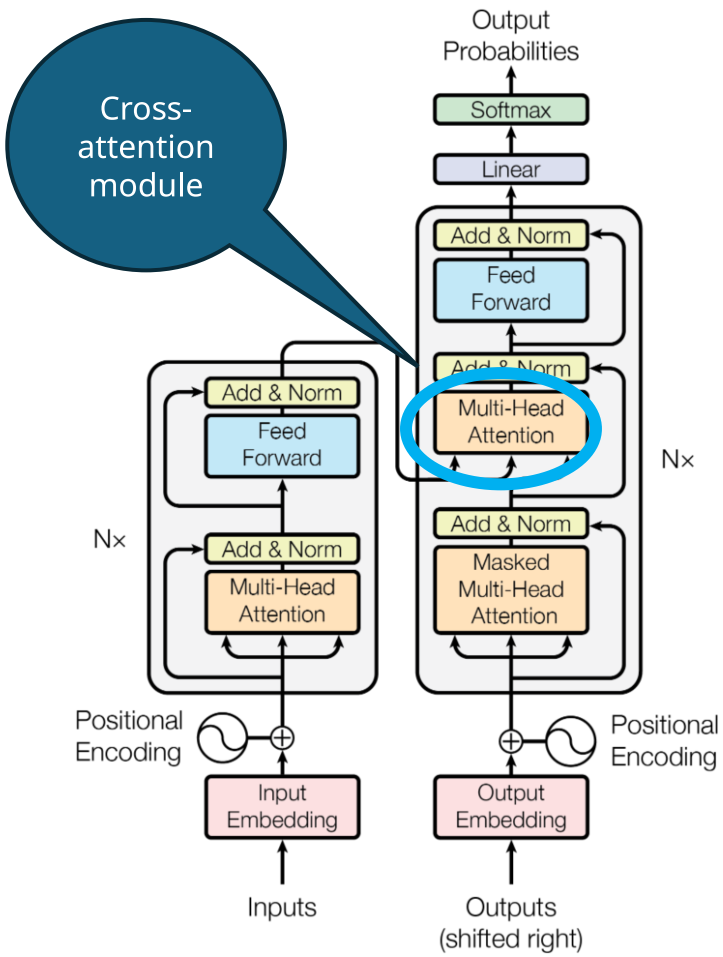


Figure 1: The Transformer - model architecture.

Many of the module are the same as encoder, but **2 differences because of the nature of the decoder**

Difference 1
The decoder has to also look at the encoder representation (from last lecture)

➔ Extra cross-attention module
➔ Similar to self-attention, look at top encoder layer instead of previous layer

Let’s look at the decoder side of things

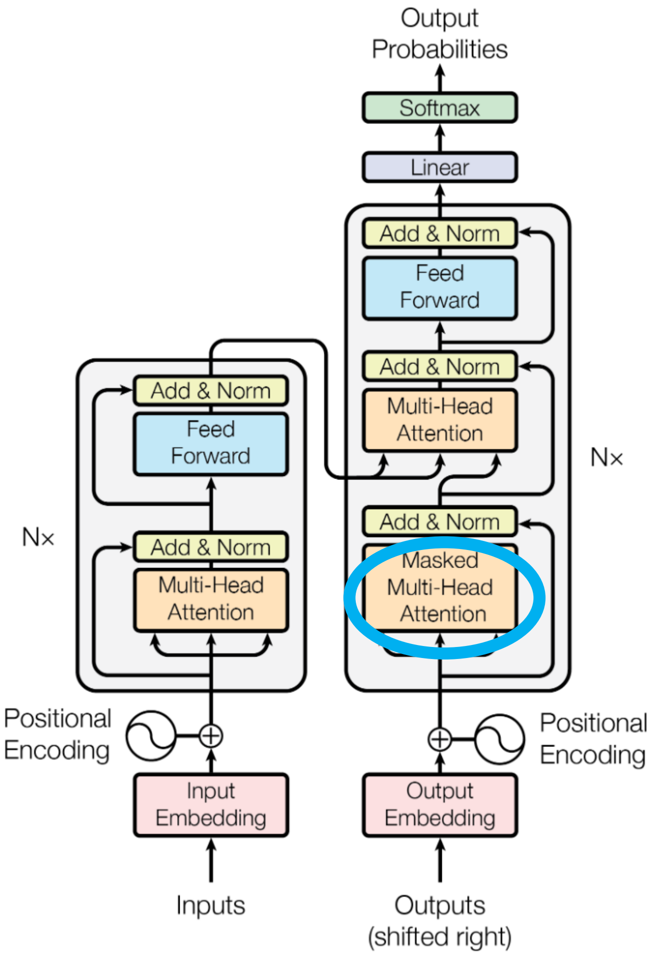


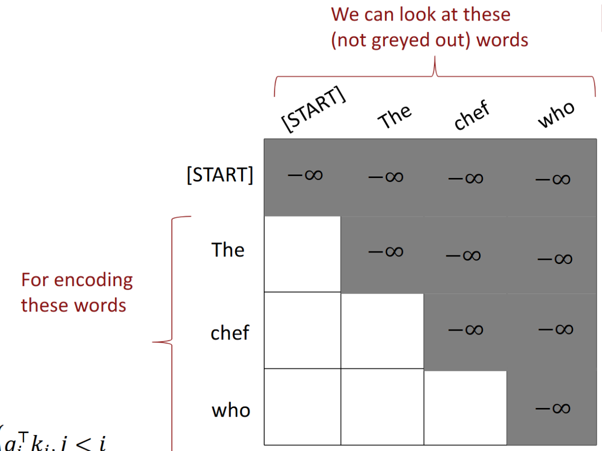
Figure 1: The Transformer - model architecture.

Many of the module are the same as encoder, but **2 differences because of the nature of the decoder**

Difference 2

Decoder is generation module, generating from left to right – so self-attention does not have access to future tokens

Self-attention layer uses masked attention



$\{q_i^\top k_j, j < i$

To enable parallelization, we **mask out attention** to future words by setting attention scores to $-\infty$.

$$e_{ij} = \begin{cases} q_i^\top k_j, j < i \\ -\infty, j \geq i \end{cases}$$

Now for the final pieces to train deep networks

Layer Normalization

Residual Connections

Layer Normalization [Ba et al., 2016]

- **Problem:** Difficult to train the parameters of a given layer because its input from the layer beneath keeps shifting.
- **Solution:** Reduce variation by **normalizing** to zero mean and standard deviation of one within each **layer**.

Mean: $\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l$ Standard Deviation: $\sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2}$

$$\boxed{x^{\ell\prime} = \frac{x^{\ell} - \mu^{\ell}}{\sigma^{\ell} + \epsilon}}$$

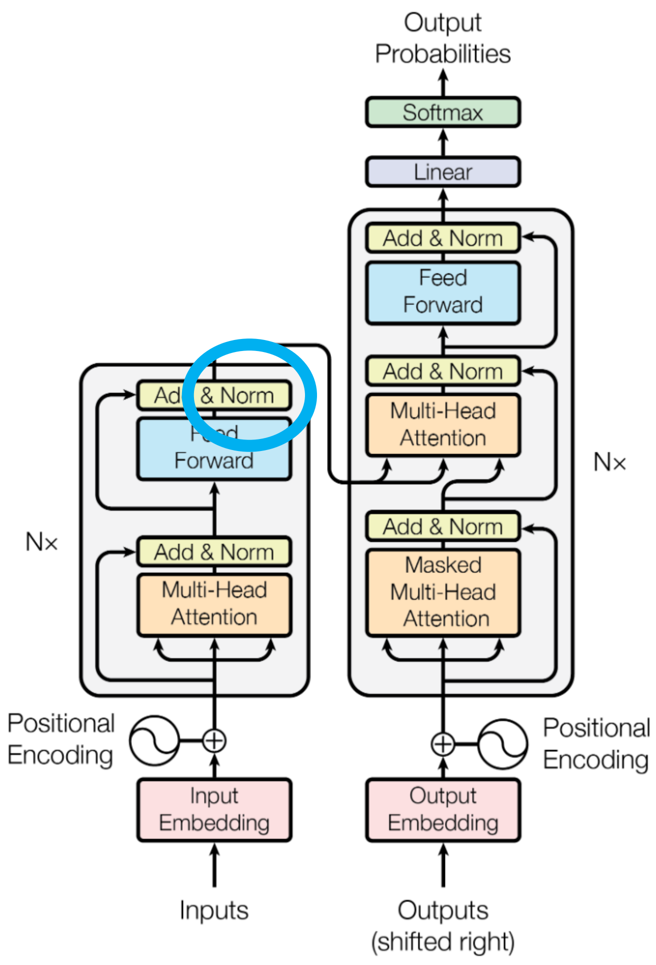


Figure 1: The Transformer - model architecture.

Residual Connections [He et al., 2016]

So far we have looked at modules in a single layer – now let's look between layers

Difficult to train deep networks because of

- Vanishing gradient (again! - now between layers)
- Degradation – deep networks find it difficult to learn representations from scratch

Residual module: add previous layer's output to current layer's output

$$\text{Output} = F(x) + x$$

Network is now learning residuals F(x)

Direct connection for gradient flow
(similar to what LSTMs did – without the gating)

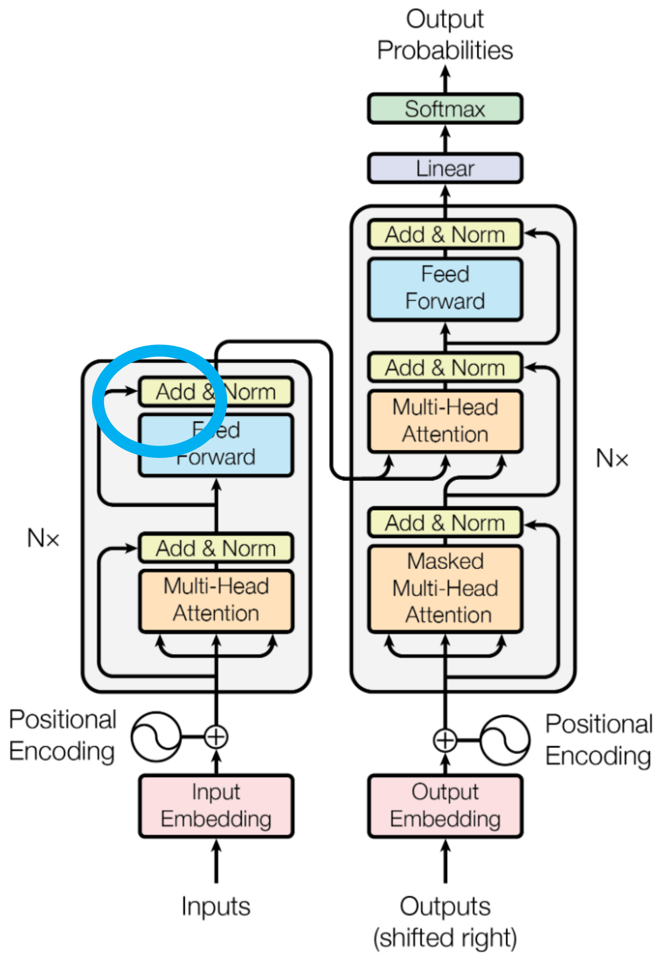
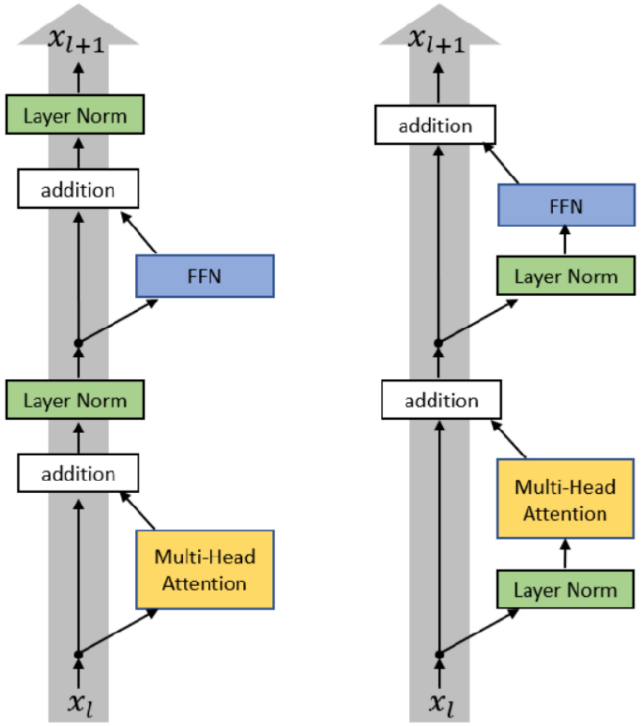


Figure 1: The Transformer - model architecture.

Layer Normalization Revisited (Xiong et al., 2020)



Post-norm **Pre-norm**

Original transformer paper used Post-Norm implementation

Training instability due to improper gradient flow

All modern transformers use Pre-norm

Summarizing Transformers

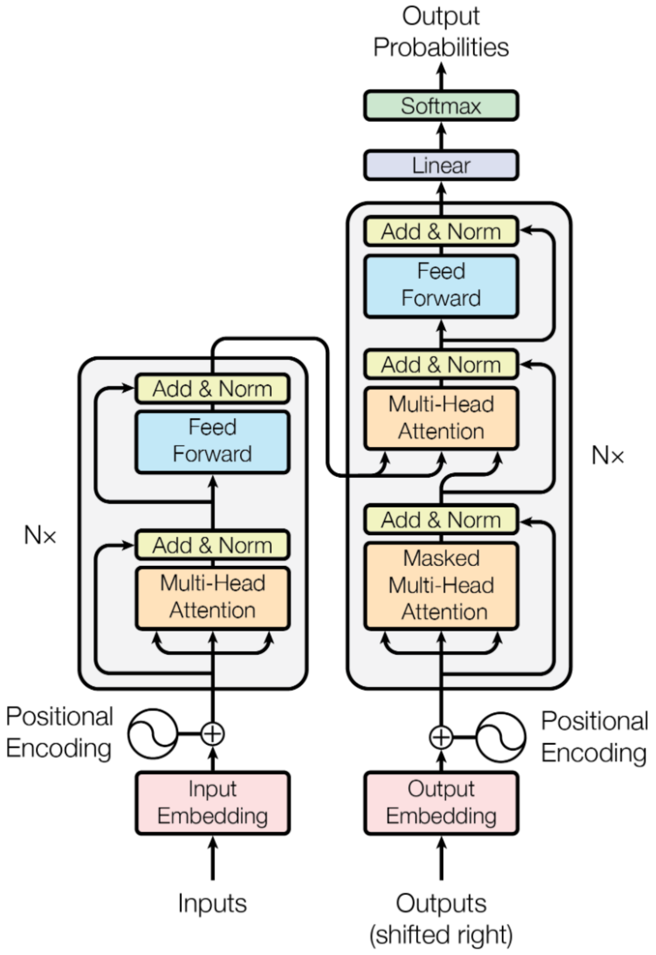


Figure 1: The Transformer - model architecture.

Self-Attention: enables distributed representations that can be learnt parallelly and without gradient vanishing issues.

Multi-head Attention: help look at different perspectives of a token

Positional Encoding: encode positional information

Feedforward Module: the brains of the networks

Scaling to deep networks: residual connections and layer normalization

Transformer Variants

- Transformer Encoder ➔ BERT, XLM-R, ModernBERT and all other encoder-only models.
  - Use only the encoder part of the transformer
- Transformer Decoder ➔ GPT and all modern generative LLMs
  - Use the decoder part only – masked self-attention, but cross-attention is not needed.

Dive Deep!

Understand the math for each component and understand the paper: **The Annotated Transformer**

Read through network architecture of transformers in **Fairseq** or **HuggingFace** code

Implement a minimalist transformer model: **nanogpt**

Go through the reading list and dive into the suggested readings

Exercise: Can you count the number of parameters in the network given its architecture?

Exercise: What is the time and space-complexity of processing a sequence?

Reading List

1. Vaswani, A., et al. (2017). **Attention Is All You Need**. In *Advances in Neural Information Processing Systems* (pp. 5998–6008). (Link).
2. Alammar, J. (2018). **The Illustrated Transformer**. [Blog post]. (Link)
3. Rush, A. M. (2018). **The Annotated Transformer**. (Link)
4. Anna Goldie. (2024). Transformers. (Link)
5. Mitesh Khapra and Arun Prakash. **Lecture 16: Transformers/CS6910**. (Link)
6. Ba, J. et al. (2016). **Layer Normalization**. arXiv preprint arXiv:1607.06450. (Link)
7. He, K. et al. (2016). **Deep Residual Learning for Image Recognition**. CVPR. (Link)
8. Xiong, R., et al. (2020). **On Layer Normalization in the Transformer Architecture**. ICML. (Link)
9. Devlin, J., et al. (2018). **BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding**. NAACL. (Link)
10. Radford, A., Narasimhan, K., Salimans, T. and Sutskever, I. (2018). **Improving Language Understanding by Generative Pre-Training**. (Link)
11. Fleetwood (2024). **You could have designed state of the art positional encoding.** Blog Post. (Link)

Assignment

Build your transliteration model

- **Transliteration**: converting text from one script to another
- Sequence to sequence model
- Build a transformer-based transliteration model
- Task: Indic scripts to Roman and vice-versa

Assignment Details to be provided by the end of the week

Additional Topics

Subword Vocabulary

The Vocabulary Problem

- **The input & output embedding layers are finite**
  - How to handle an open vocabulary?
  - How to translate named entities?

- **Softmax computation at the output layer is expensive**
  - Proportional to the vocabulary size

$$softmax(o_{jk}) = \frac{\exp(o_{jk})}{\sum\limits_{m=0}^{m=T} \exp(o_{jm})}$$

Subword-level Translation

Original sentence: प्रयागराज में 43 दिनों तक चलने वाला माघ मेला आज से शुरू हो गया है

Possible inputs to NMT system:

- प्रयाग @@राज में 43 दि @@नों तक चल @@ने वाला माघ मेला आज से शुरू हो गया है
- प्र या ग रा ज _में _ 43 _ दि नों _ त क _ च ल ने _ वा ला _मा घ मे ला _ आज _ से _ शुरू _ हो _ गया _ है

Obvious Choices: Character, Character n-gram, Morphemes ➔ They all have their flaws!

The New Subword Representations: Byte-Pair Encoding, Unigram (implemented in SentencePiece package)

Learn a fixed vocabulary & segmentation model from training data

Segment Training Data based on vocabulary

Train NMT system on the segmented model

{प्रयाग, राज, में दि, नों, तक, चल, ने}

vocabulary

{प्रयाग राज}
{च ल}
{चल, ने}

Segmentation model

प्रयाग @@राज में 43 दि @@नों तक चल @@ने वाला माघ मेला आज से शुरू हो गया है

- Every word can be expressed as a concatenation of subwords
- A small subword vocabulary has good representative power
    - 4k to 64k depending on the size of the parallel corpus
- Most frequent words should not be segmented

Byte Pair Encoding

Byte Pair Encoding is a greedy compression technique (Gage, 1994)

Number of BPE merge operations=3
Vocab: A B C D E F

P_1=AD P_2=EE P_3=P_1D

Words to encode	*Iterations* ①	②	③	④
BADD	BADD	BP_1D	BP_1D	BP_3
FAD	FAD	FP_1	FP_1	FP_1
FEEDE	FEEDE	FEEDE	FP_2DE	FP_2DE
ADDEEF	ADDEEF	P_1DEEF	P_1DP_2F	$P_3$$P_2$F

Data-dependent segmentation

- Inspired from compression theory
- MDL Principle *(Rissansen, 1978)* ⇒ Select segmentation which maximizes data likelihood

Problems with subword level translation

Unwanted splits:

नाराज़ ✉ ना राज़ no secret

Problem is exacerbated for:

- Named Entities
- Rare Words
- Numbers

Explore multiple subword segmentation

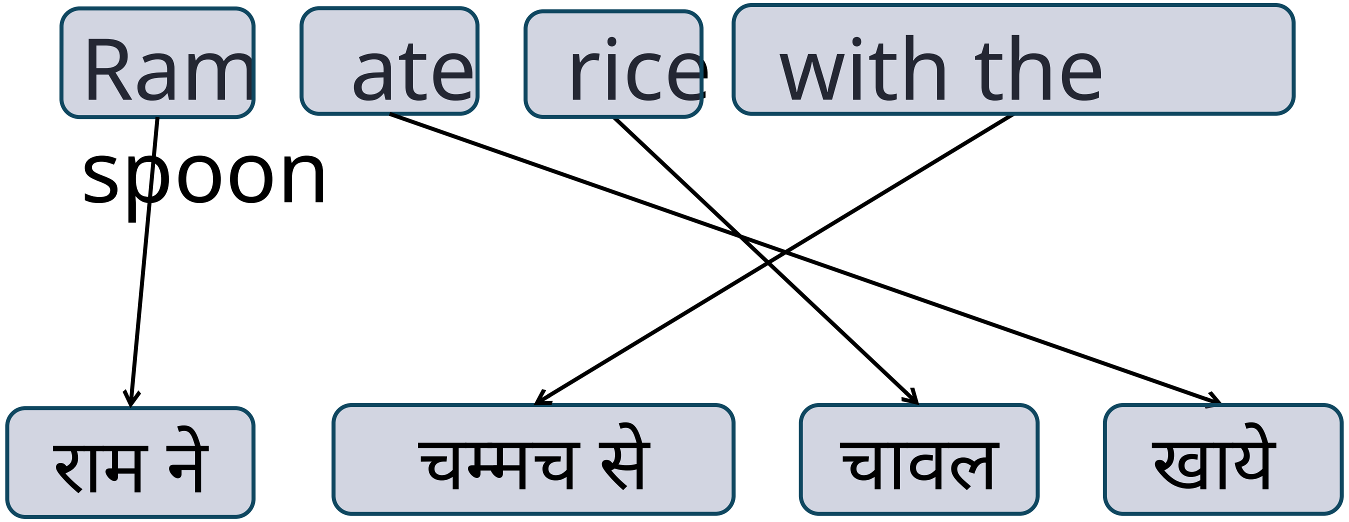
- BPE dropout
- Unigram + subword-regularization

Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates. Taku Kudo. ACL 2018.
BPE-Dropout: Simple and Effective Subword Regularization Ivan Provilkov, Dmitrii Emelianenko, Elena Voita. ACL 2020

Decoding

Decoding

Searching for the best translations in the space of all translations



Decoding Strategies

- Exhaustive Search: *Score each and every possible translation – Forget it!* ➔ $O(V^N)$
- Sampling ➔ $O(NV)$
- Greedy ➔ $O(NV)$
- Beam Search ➔ $O(kNV)$

Greedy Decoding

w_1	0.03
w_2	0.7
w_3	0.05
w_3	0.1
w_4	0.08
w_5	0.04

Select best word using the distribution $P(y_j|y_{<j},\boldsymbol{x})$

Sampling Decoding

w_1	0.03
w_2	0.7
w_3	0.05
w_3	0.1
w_4	0.08
w_5	0.04

Sample next word using the distribution $P(y_j|y_{<j},\boldsymbol{x})$

Generate one word at a time sequentially

Not used to find best translation, but these methods have their uses ➔ for efficiency reas

Greedy Search is not optimal

	t_1		t_2
w_1	**0.5**	w_1	0.1
w_2	0.4	w_2	0.2
w_3	0.05	w_3	**0.3**
w_3	0.02	w_3	0.1
w_4	0.01	w_4	0.1
w_5	0.02	w_5	0.2

Probability of best sequence w_1w_3 *=0.15*

	t_1		t_2
w_1	0.5	w_1	0.1
w_2	**0.4**	w_2	0.45
w_3	0.05	w_3	0.2
w_3	0.02	w_3	0.15
w_4	0.01	w_4	0.08
w_5	0.02	w_5	0.02

Probability of best sequence w_2w_2 *=0.18*

Beam Search

A compromise solution between greedy decoding and exhaustive search

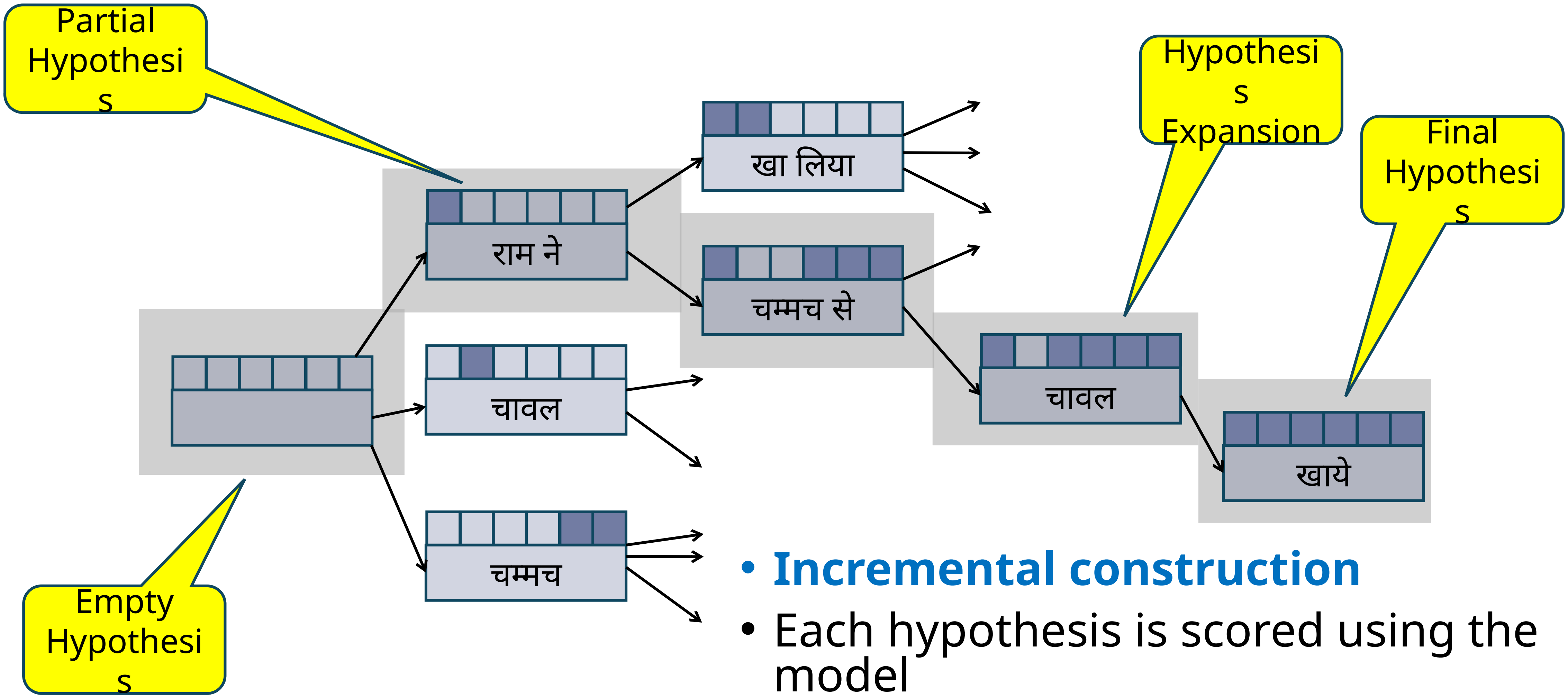
- *Explores more translation candidates than greedy search*
- *More efficient than exhaustive search*

2 Core Ideas:

- ***Incremental*** *construction & scoring of translation candidate (one decoder time step at a time)*
- *At each decoder time step,* ***keep the k-most probable partial translations***
  - *➔ these will be used for candidates expansion*

- ***Not guaranteed to find optimal solution***

http://www.phontron.com/slides/nlp-programming-en-13-search.pdf



- **Incremental construction**
- Each hypothesis is scored using the model
- Hypotheses are maintained in a priority queue

Backtranslation

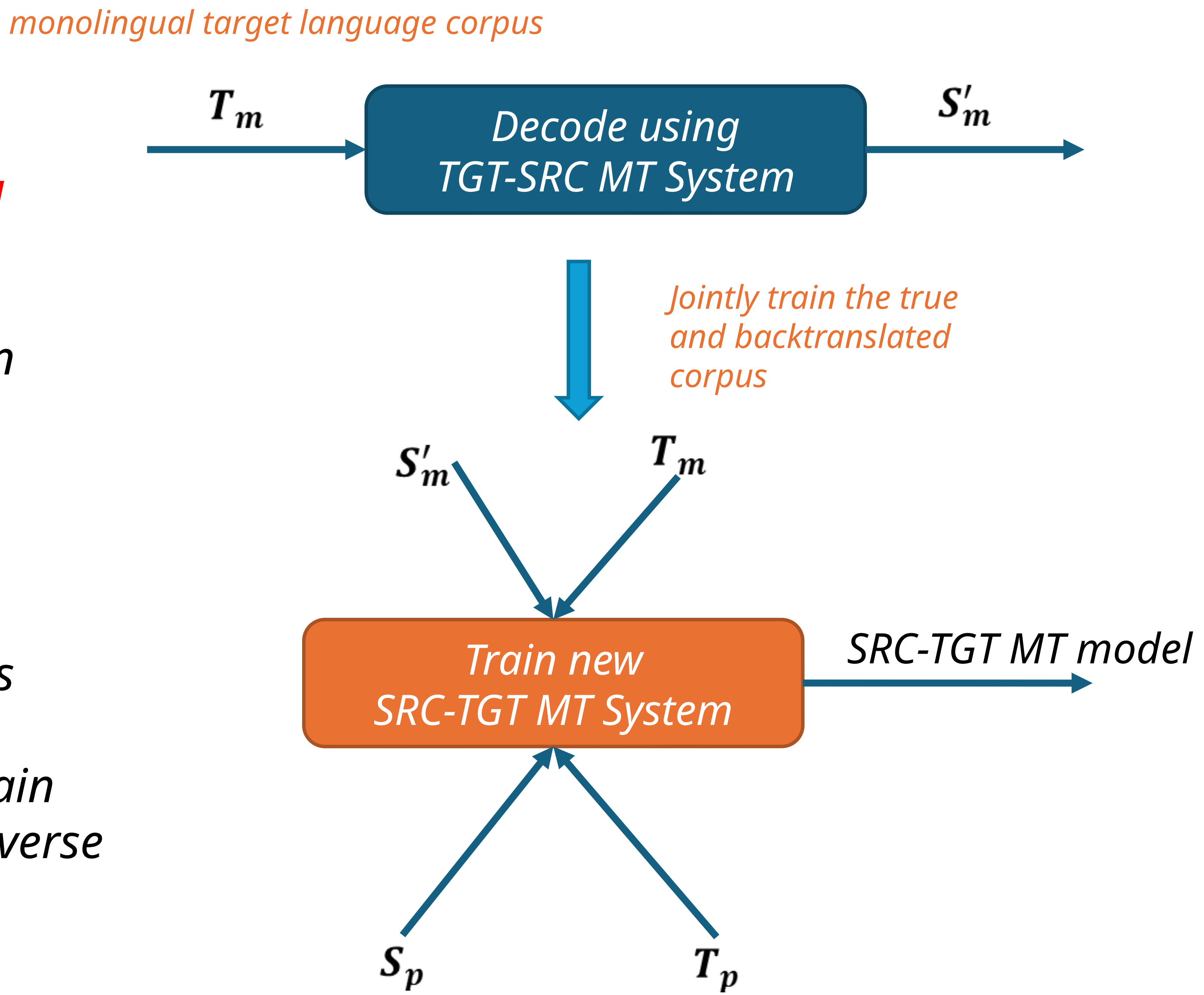
The models discussed so far do not use monolingual data

Can monolingual data help improve NMT models?

Backtranslation

Create pseudo-parallel corpus using Target to source model (Backtranslated corpus)

Need to find the right balance between true and backtranslated corpus



Why is backtranslation useful?

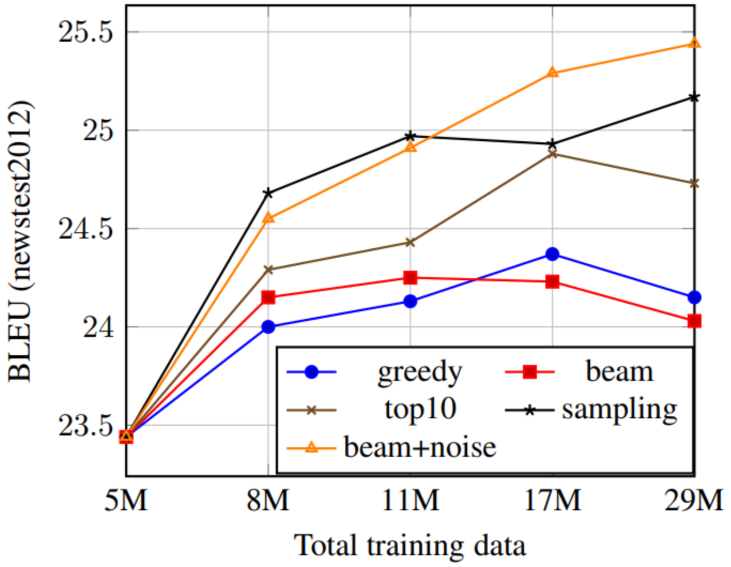
- *Target side language model improves*
  - *target side is clean*
- *Adaptation to target language domain*
- *Prevent overfitting by exposure to diverse corpora*

Particularly useful for low-resource languages

Sennrich, Rico, Barry Haddow, and Alexandra Birch. "Improving neural machine translation models with monolingual data." ACL 2016

Make backtranslation more diverse

- *Sampling*
- *Restricted Sampling*
- *Beam+noising*



Make it easy for the model to distinguish between natural & synthetic input

Tagged Backtranslation ➔

add a special token indicating that the input is synthetic

Noise type	Example sentence
[no noise]	Raise the child, love the child.
P3BT	child Raise the, love child the.
NoisedBT	Raise child ___ love child, the.
TaggedBT	<BT> Raise the child, love the child.
TaggedNoisedBT	<BT> Raise, the child the ___ love.

Tagged BT and Noised BT serve the same purpose ➔ distinguishing inputs

Sergey Edunov, Myle Ott, Michael Auli, David Grangier . Understanding Back-Translation at Scale. EMNLP 2018.

Isaac Caswell, Ciprian Chelba, David Grangier . Tagged Back-Translation. WMT 2019

Thank You!

Write to me at anoop.kunchukuttan@gmail.com in case you have any questions