Transformers

Acknowledgments

Slides adapted from

Anna Goldie, Shafiq Joty, Chris Manning

Contextualized Word Representations

Motivating Word Meaning and Context

Recall the adage we mentioned at the beginning of the course:

> *"You shall know a word by the company it keeps"* (J. R. Firth 1957: 11)

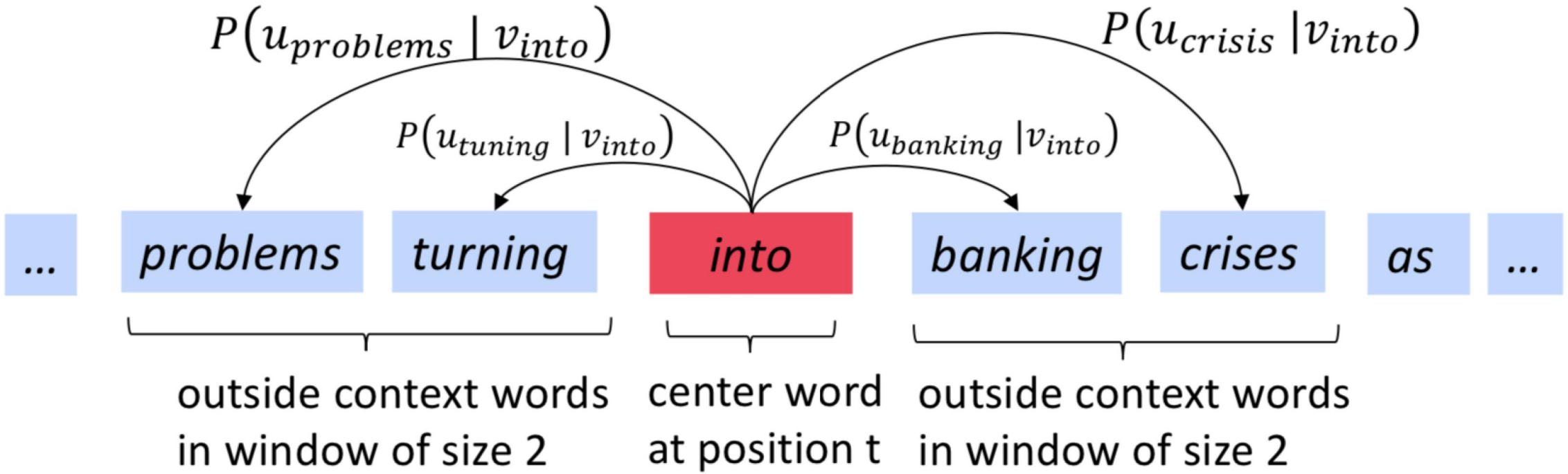
This quote is a summary of **distributional semantics**, and motivated **word2vec**. But:

> "… the complete meaning of a word is always contextual,
>
> and no study of meaning apart from a complete context
>
> can be taken seriously." (J. R. Firth 1935)

Consider *I* ***record*** *the* ***record***: the two instances of ***record*** mean different things.

[Thanks to Yoav Goldberg on Twitter for pointing out the 1935 Firth quote.]

Representation of a Word in Word2Vec



Representation of a Word in Word2Vec

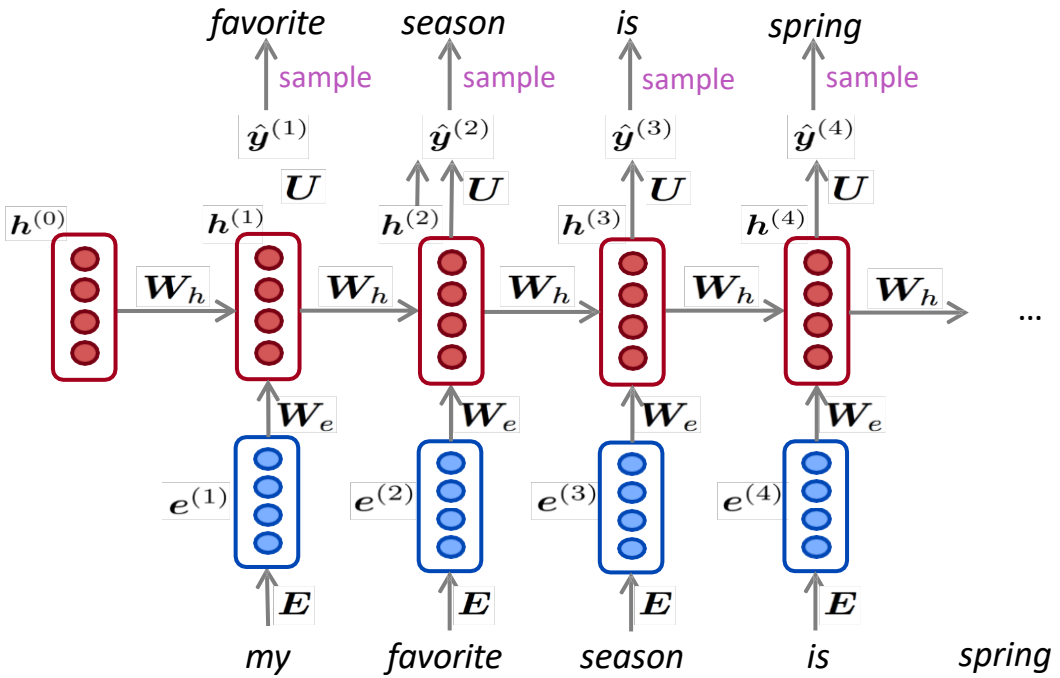
- **Word2vec has one representation vector for a word**

- Always the same representation for a **word type** regardless of the context in which a **word token** occurs
  - We might want very fine-grained word sense disambiguation

- We just have one representation for a word
  - but words have different **aspects**, including semantics, syntactic behavior, and register/connotations

Do We Already Have a Solution to This Problem?

- In an NLM, we immediately stuck word vectors (perhaps only trained on the corpus) through LSTM layers
- Those LSTM layers are trained to predict the next word
- Producing **context-specific word representations at each position**!



TagLM [Peters et al., 2017]

https://arxiv.org/pdf/1705.00108.pdf

- Idea: We want the meaning of a word in context, but standardly learn a task RNN only on a small task-labeled dataset (e.g., NER)

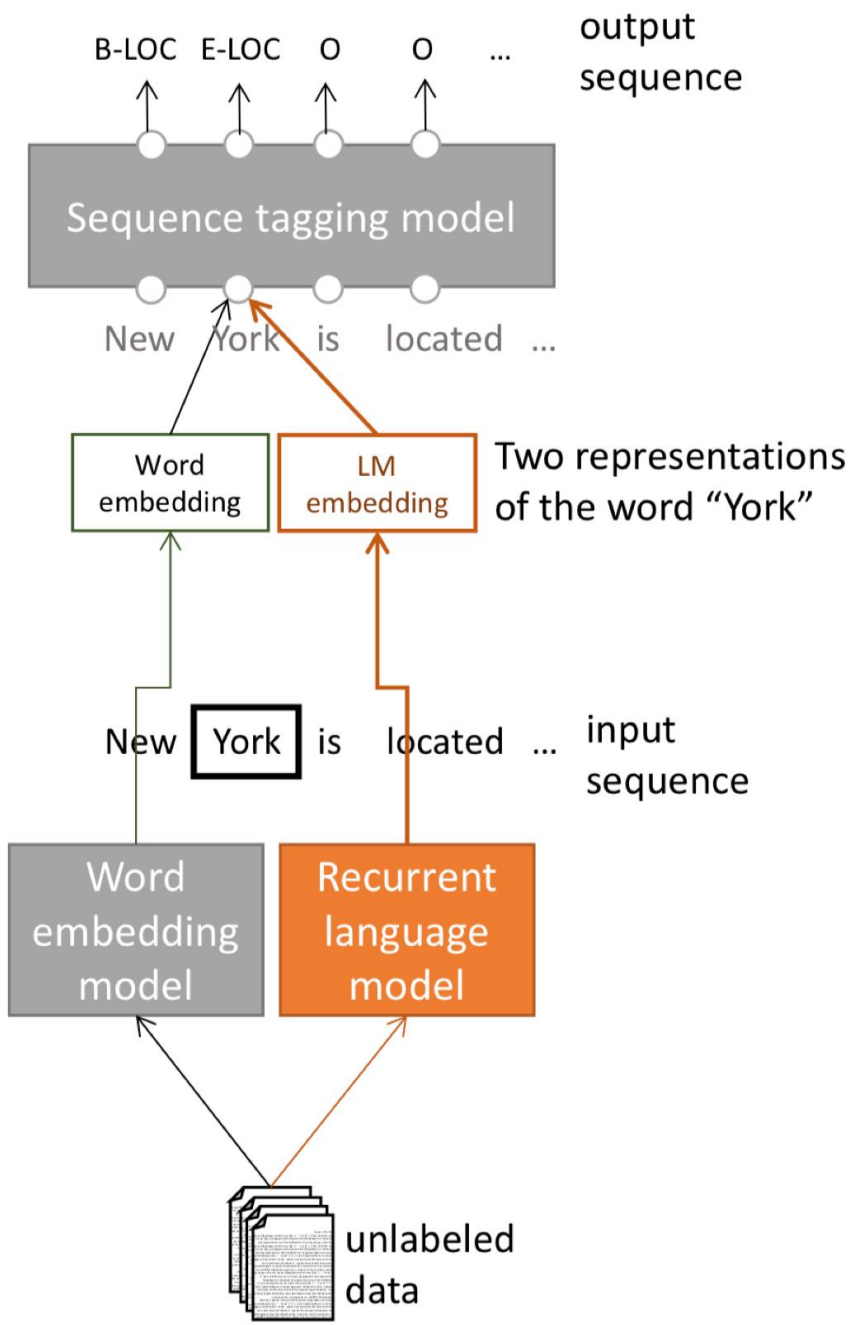
- Why not a semi-supervised approach where we train an NLM on a large unlabeled corpus, rather than just word vectors?

Tag LM

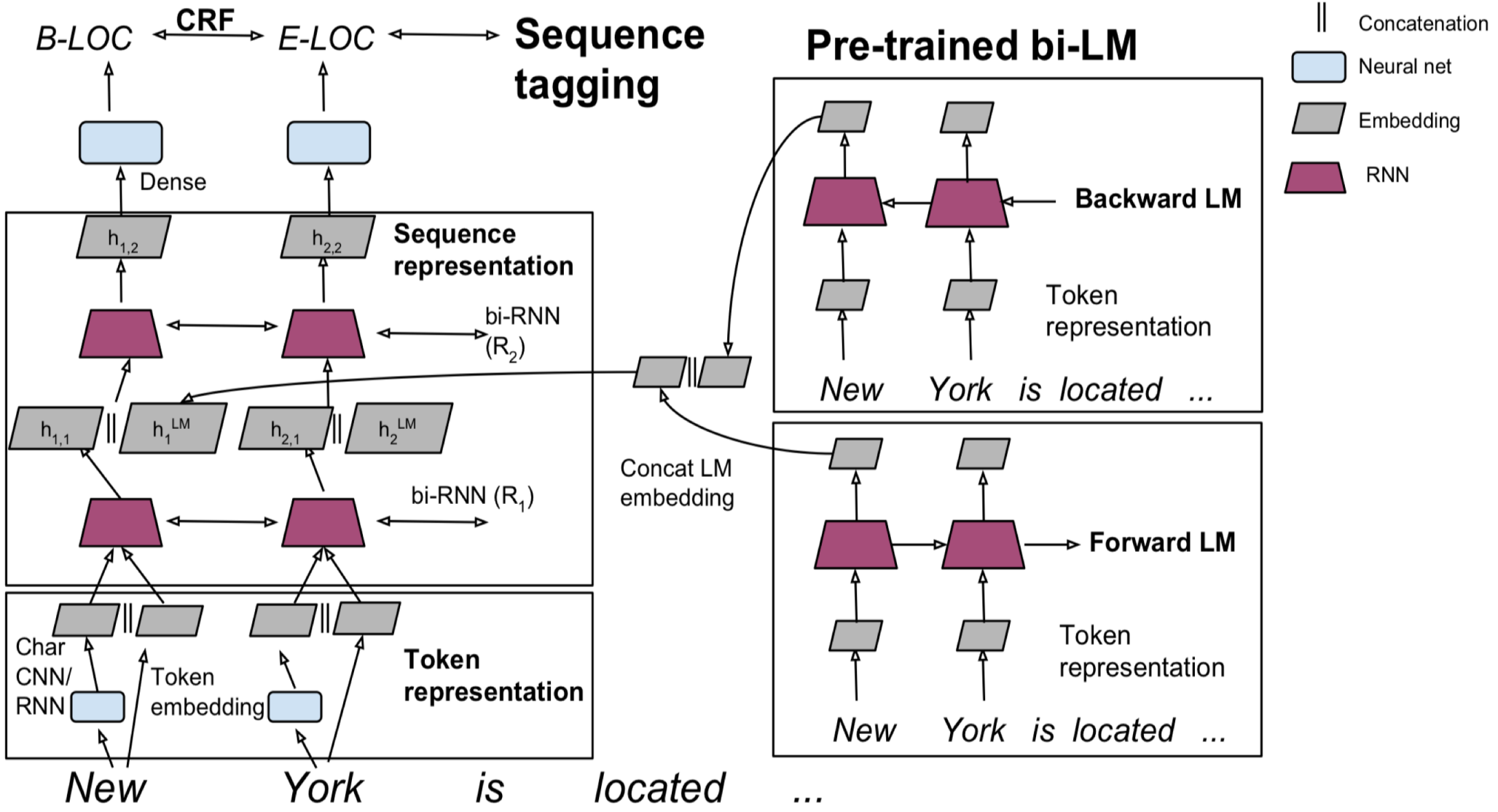
Step 3: Use both word embeddings and LM embeddings in the sequence tagging model.

Step 2: Prepare word embedding and LM embedding for each token in the input sequence.

Step 1: Pretrain word embeddings and language model.



Tag LM



$$\mathbf{h}_{k,1} = [\overrightarrow{\mathbf{h}}_{k,1}; \overleftarrow{\mathbf{h}}_{k,1}; \mathbf{h}_k^{LM}]$$

TagLM for CoNLL-2003 Named Entity Recognition

TagLM Peters	LSTM BiLM in BiLSTM tagger	2017	91.93
Ma + Hovy	BiLSTM + char CNN + CRF layer	2016	91.21
Tagger Peters	BiLSTM + char CNN + CRF layer	2017	90.87
Ratinov + Roth	Categorical CRF+Wikipeda+word cls	2009	90.80
Finkel et al.	Categorical feature CRF	2005	86.86
IBM Florian	Linear/softmax/TBL/HMM ensemble, gazettes++	2003	88.76
Stanford (Klein)	MEMM softmax Markov model	2003	86.07

CoVe [McCann et al., 2017]

https://dl.acm.org/doi/10.5555/3295222.3295377

- Similar idea to use a trained sequence model to learn contextualized representations for other NLP models

- Machine translation is meant to preserve meaning, so maybe that is a good objective?

- Use a 2-layer bi-LSTM that is the encoder

of a seq2seq + attention NMT system as the context provider

CoVe [McCann et al., 2017]

- The resulting CoVe vectors do outperform GloVe vectors on various tasks
  - But not as strong as simpler NLM training

Now abandoned…

- Maybe NMT is just harder than language modeling?
- Maybe some day this idea will return?

ELMo: Embeddings from Language Models

[Peters & al, 2018]

https://aclanthology.org/N18-1202

- Deep **contextualized** word representations
- Breakout version of **word token vectors** or **contextual word vectors**
- Learn word token vectors using long contexts, not context windows
  - here, whole sentence, could be longer
- Learn a deep Bi-NLM and use all its layers in the prediction

ELMo: Embeddings from Language Models

Structure

Each token t_k

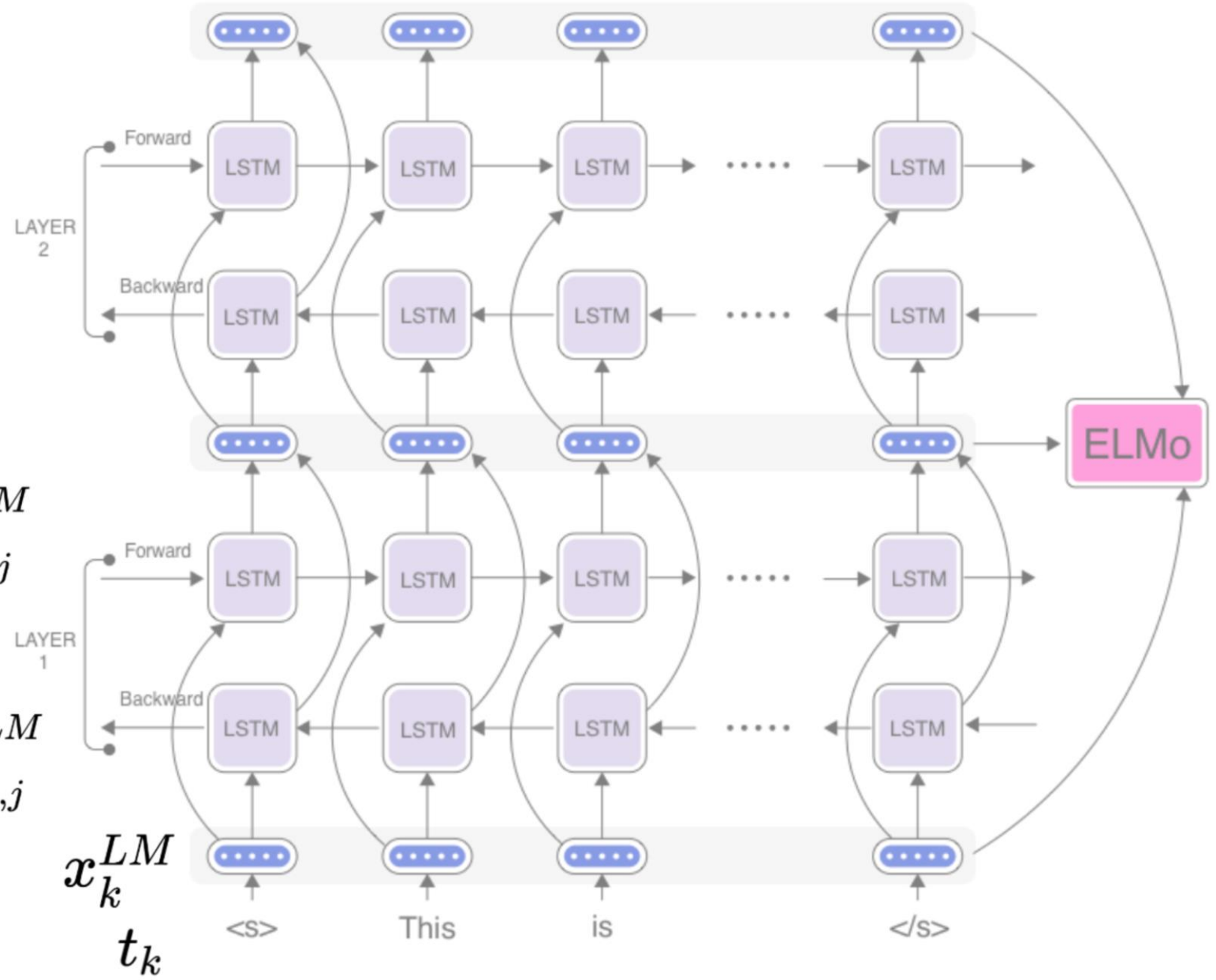
L-layer biLM computes 2L+1 representations

$\overrightarrow{h}^{LM}_{k,j}$

k is the k-th token

$\overleftarrow{h}^{LM}_{k,j}$

j is the j-th biLM layer



ELMo: Embeddings from Language Models

- Train a bidirectional LM

- Aim at performant but not overly large LM
  - Use 2 biLSTM layers
  - Use character CNN to build initial word representation (only)
    - 2048 char n-gram filters and 2 highway layers, 512 dim projection
  - User 4096-dimensional hidden/cell LSTM states with 512-dimensional projections to the next input
  - Use a residual connection
  - Tie the parameters of the token input and output (softmax) and tie these between the forward and the backward LMs

ELMo: Embeddings from Language Models

- ELMo learns task-specific combination of biLM representations
- An innovation improving on just using the top layer of the LSTM stack

$$
\begin{aligned}
R_k &= \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} \mid j = 1, \ldots, L\} \\
&= \{\mathbf{h}_{k,j}^{LM} \mid j = 0, \ldots, L\},
\end{aligned}
$$

$$
\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}
$$

- γ^{task} scales overall usefulness of ELMo to task;
- $\mathbf{s}^{\text{task}}$ are softmax-normalized mixture model weights

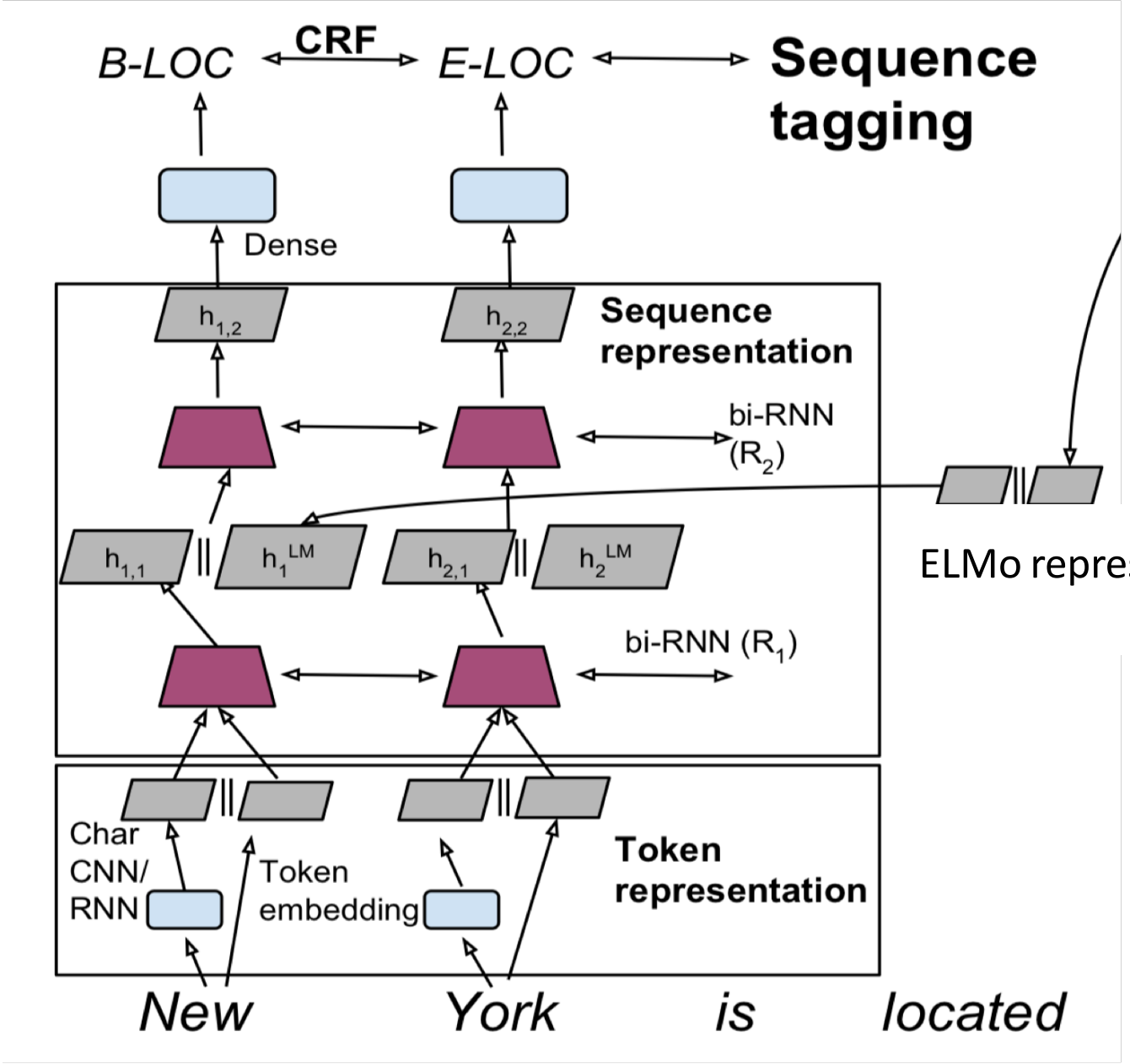
ELMo: Embeddings from Language Models

- First run a biLM to get representations for each word
- Then let the (whatever) end-task model use them
  - Freeze the weights of ELMo for the supervised model
  - Concatenate the ELMo weights into the task-specific model

- The details depend on the task
  - Concatenating into an intermediate layer as for TagLM is typical

ELMo Used in a Sequence Tagger

$$\mathbf{h}_{k,1} = [\overrightarrow{\mathbf{h}}_{k,1}; \overleftarrow{\mathbf{h}}_{k,1}; \mathbf{h}_k^{LM}]$$



ELMo: Weighting of the Layers for Different Tasks

- The two biLSTM NLM layers have differentiated uses/meanings
  - The lower layer is better for lower-level syntax, etc.
    - Part-of-speech tagging, syntactic dependencies, NER
  - The higher layer is better for higher-level semantics
    - Sentiment, semantic role labeling, question answering, SNLI

ELMo for CoNLL-2003 Named Entity Recognition

ELMo	ELMo in BiLSTM	2018	92.22
TagLM Peters	LSTM BiLM in BiLSTM tagger	2017	91.93
Ma + Hovy	BiLSTM + char CNN + CRF layer	2016	91.21
Tagger Peters	BiLSTM + char CNN + CRF layer	2017	90.87
Ratinov + Roth	Categorical CRF+Wikipeda+word cls	2009	90.80
Finkel et al.	Categorical feature CRF	2005	86.86
IBM Florian	Linear/softmax/TBL/HMM ensemble, gazettes++	2003	88.76
Stanford (Klein)	MEMM softmax Markov model	2003	86.07

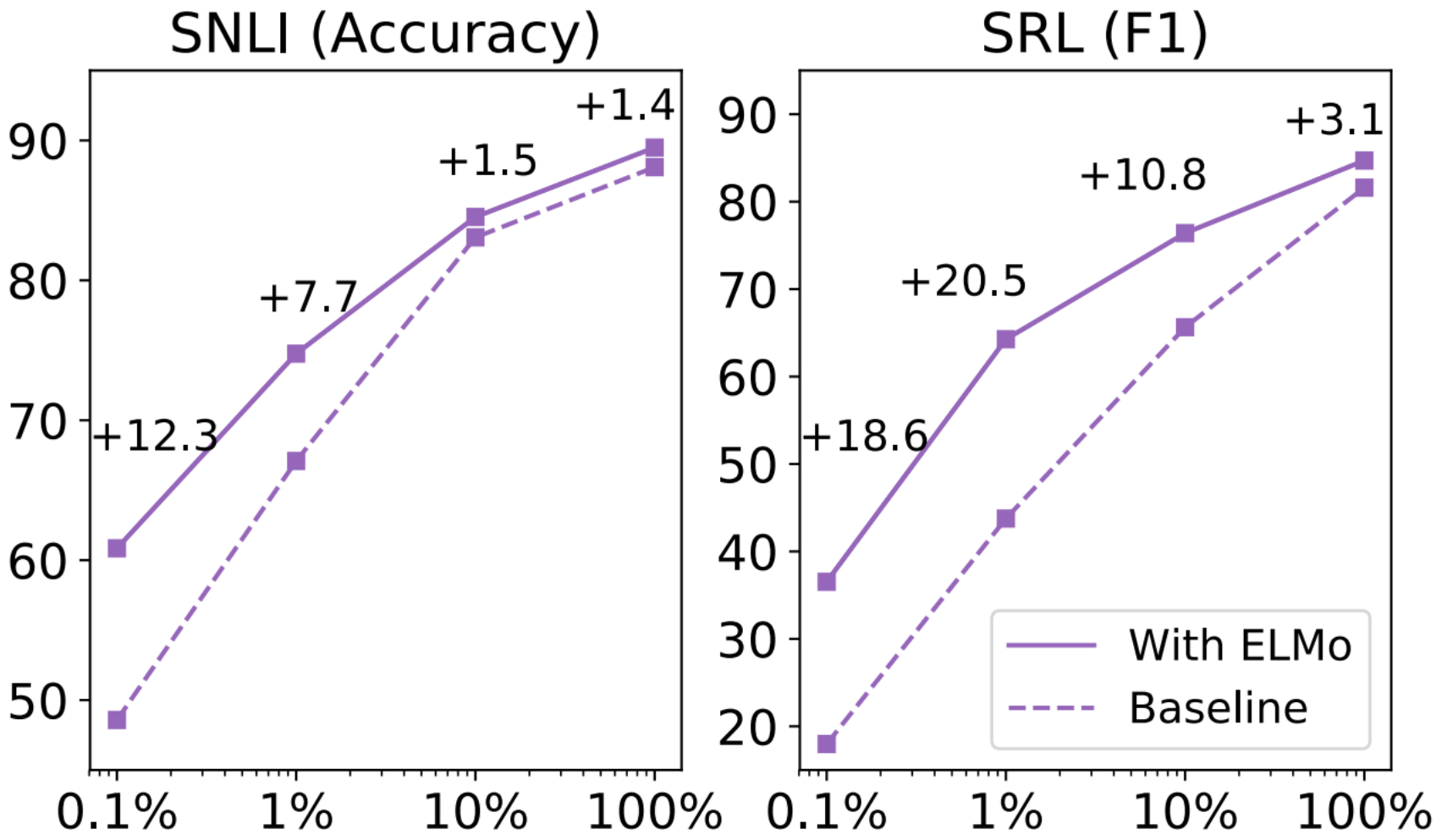
ELMo: Great for Other Tasks as Well

https://aclanthology.org/N18-1202

Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/ relative)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

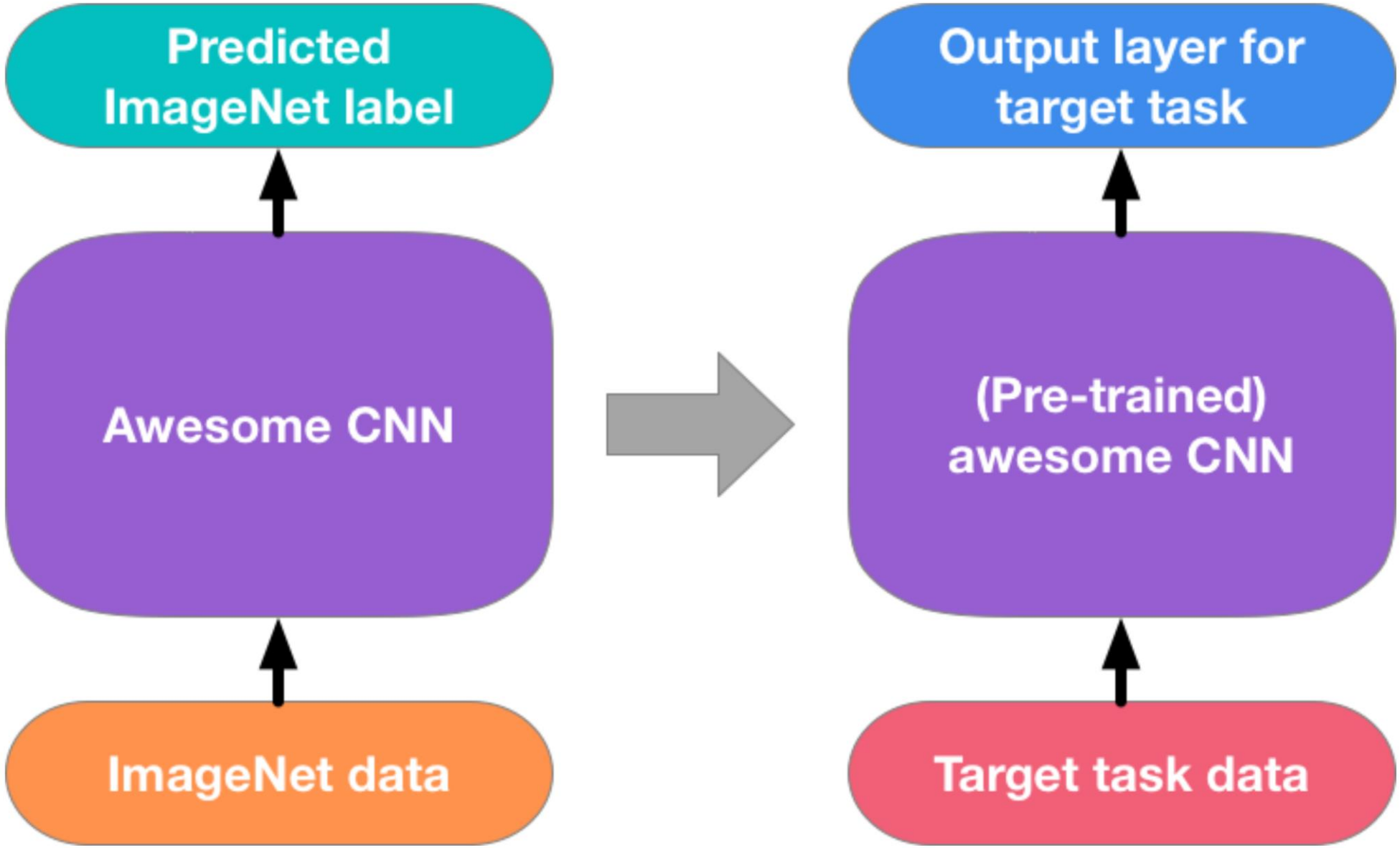
ELMo for Different Amounts of Target-Task Training Data

https://aclanthology.org/N18-1202

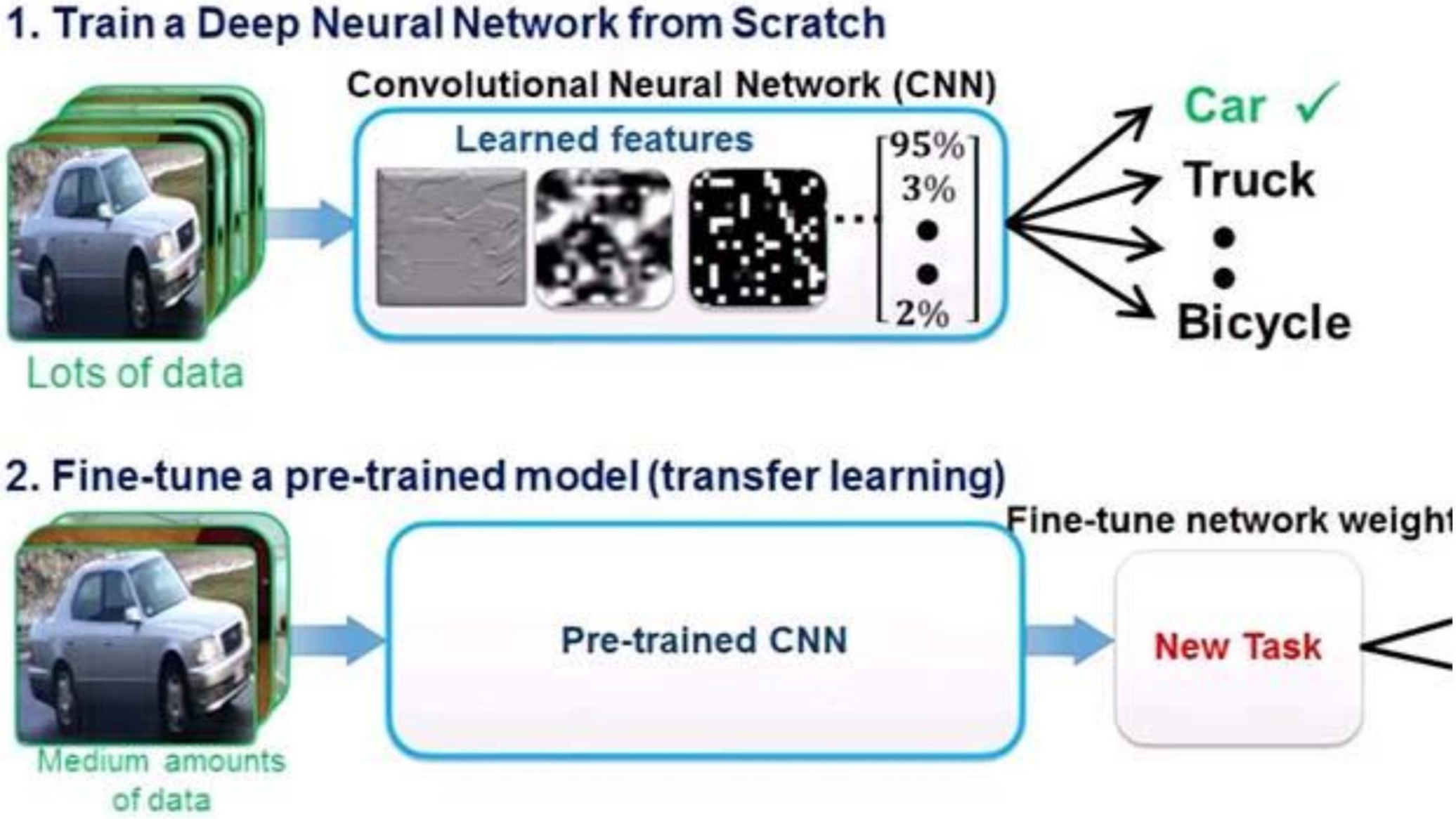


The “Pre-Train, Fine-Tune” Paradigm

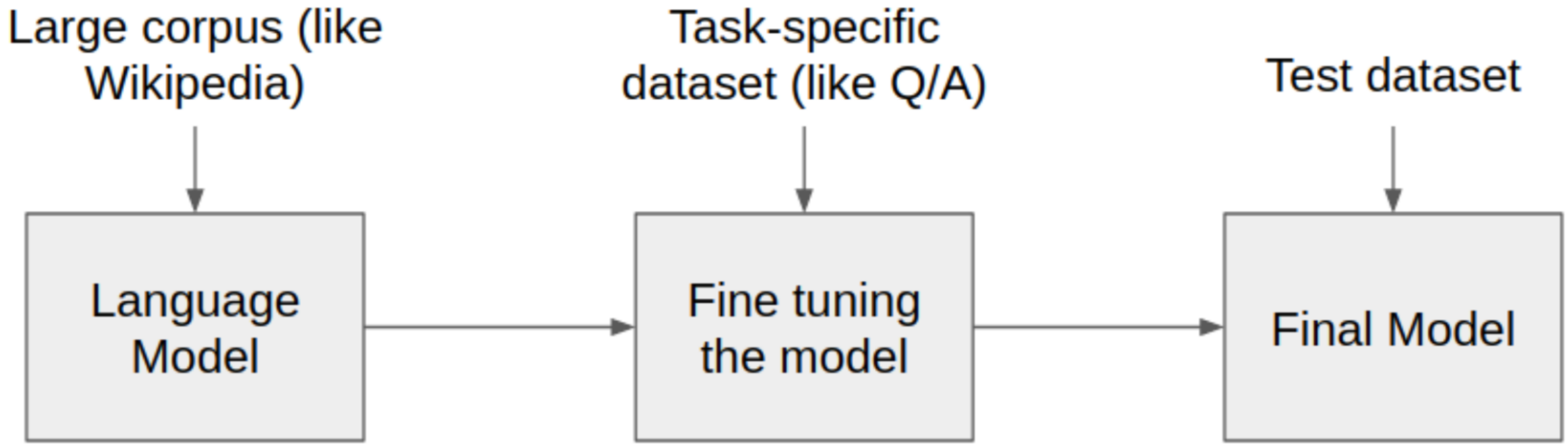
The “Pre-Train, Fine-Tune” Paradigm



The “Pre-Train, Fine-Tune” Paradigm



The “Pre-Train, Fine-Tune” Paradigm



- What do we transfer?
  - Model vs. Embeddings
- What do we pre-train
  - Encoder or Decoder or Encoder-Decoder?
- What is the pre-training objective
  - Language Model? Translation? Cloze? ….

Pre-Trained Word Vectors: The Early Years

[Collobert et al., 2011]

	POS WSJ (acc.)	NER CoNLL (F1)
State-of-the-art*	97.24	89.31
Supervised NN	**96.37**	**81.47**
Unsupervised pre-training followed by supervised NN**	**97.20**	**88.87**
+ hand-crafted features***	97.29	89.59

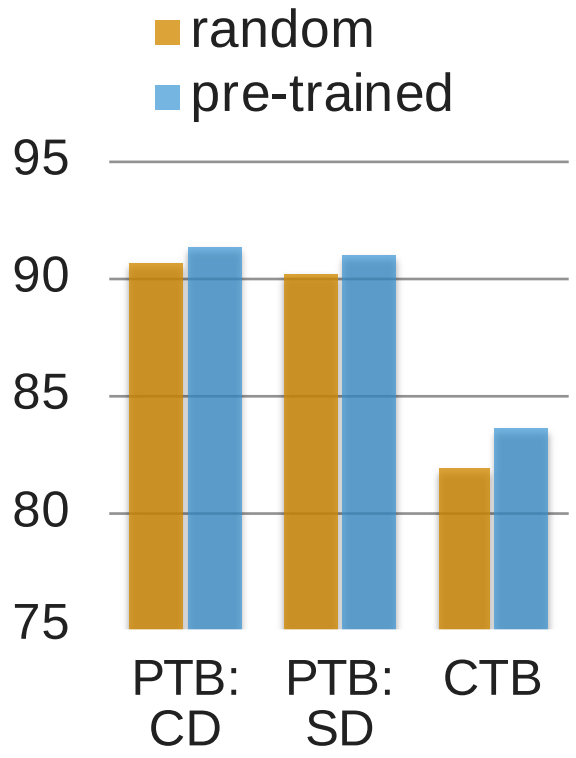
* Representative systems: POS: (Toutanova et al. 2003), NER: (Ando & Zhang 2005)

** 130,000-word embedding trained on Wikipedia and Reuters with 11 word window, 100 unit hidden layer – **for 7 weeks!** – then supervised task training

*** Features are character suffixes for POS and a gazetteer for NER

Pre-Trained Word Vectors: Current Sense (2014–)

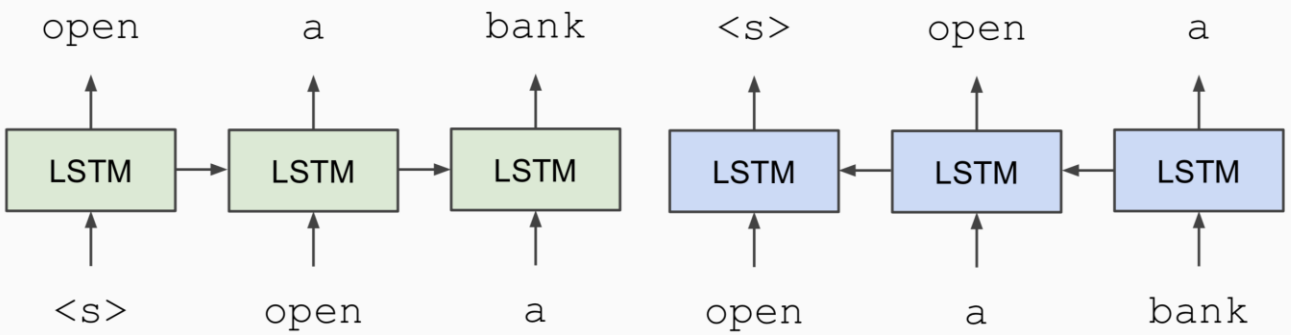
- We can just start with random word vectors and train them on our task
- However, in most cases, using pre-trained word vectors helps, because we can train them for **more words** on **much more data**



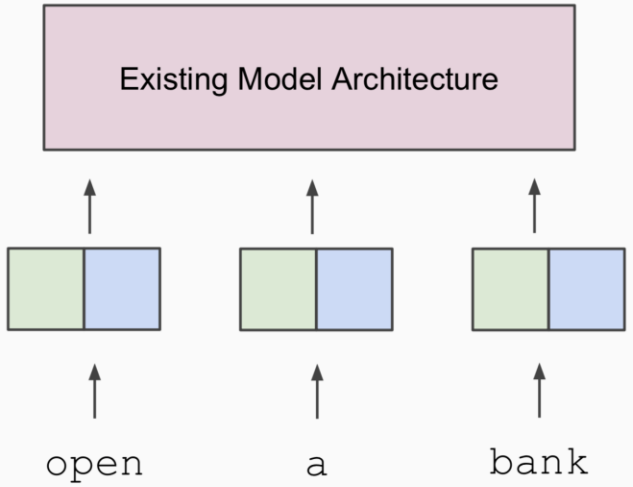
- Chen and Manning (2014)
  Dependency parsing
- Random: uniform(-0.01, 0.01)
- Pre-trained:
  - PTB (C & W): +0.7%
  - CTB (word2vec): +1.7%

ELMo: Revisited and Simplified

Train Separate Left-to-Right and Right-to-Left LMs



Apply as "Pre-trained Embeddings"

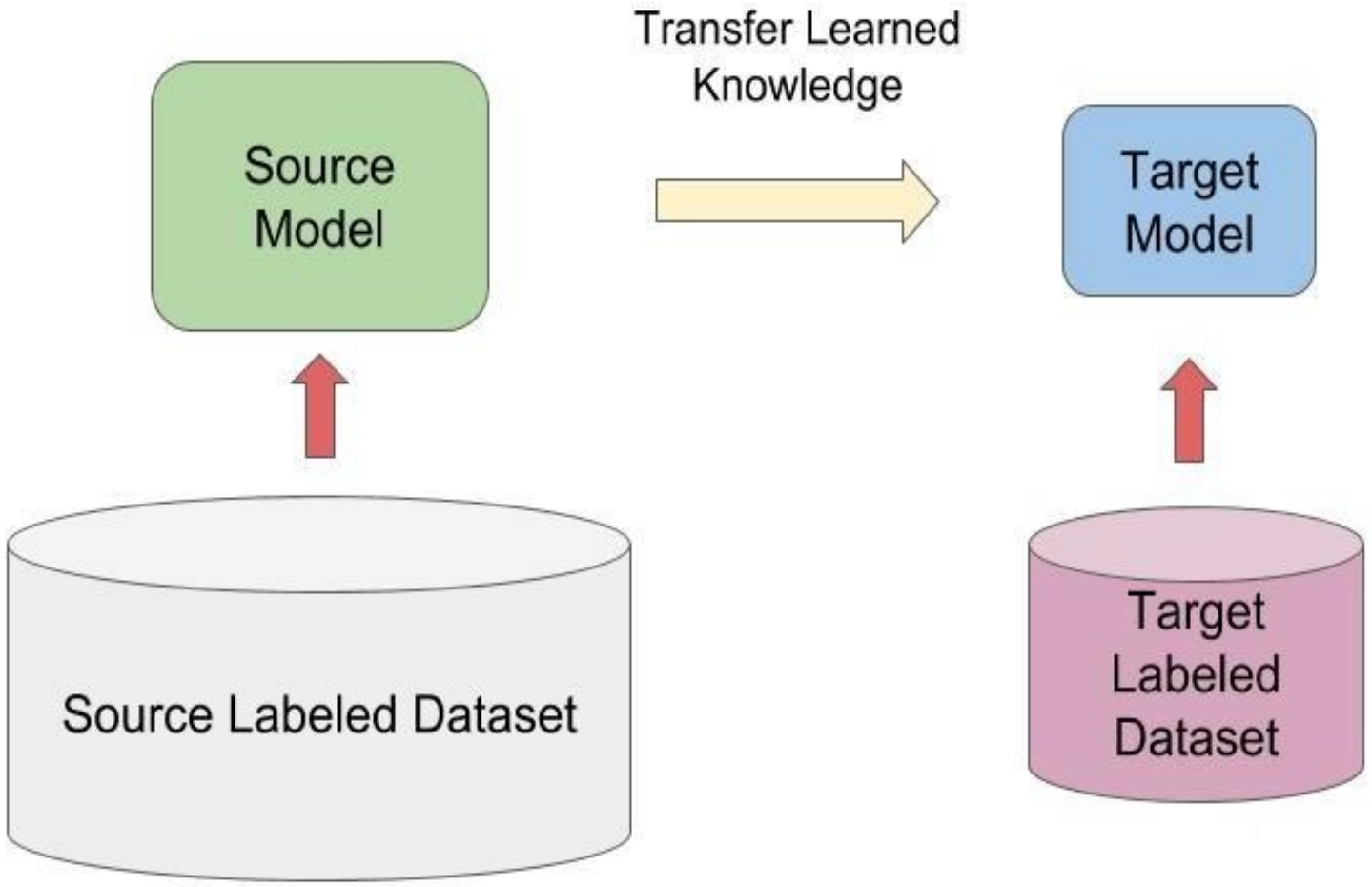


ULMFiT [Howard and Ruder, 2018]

https://aclanthology.org/P18-1031/

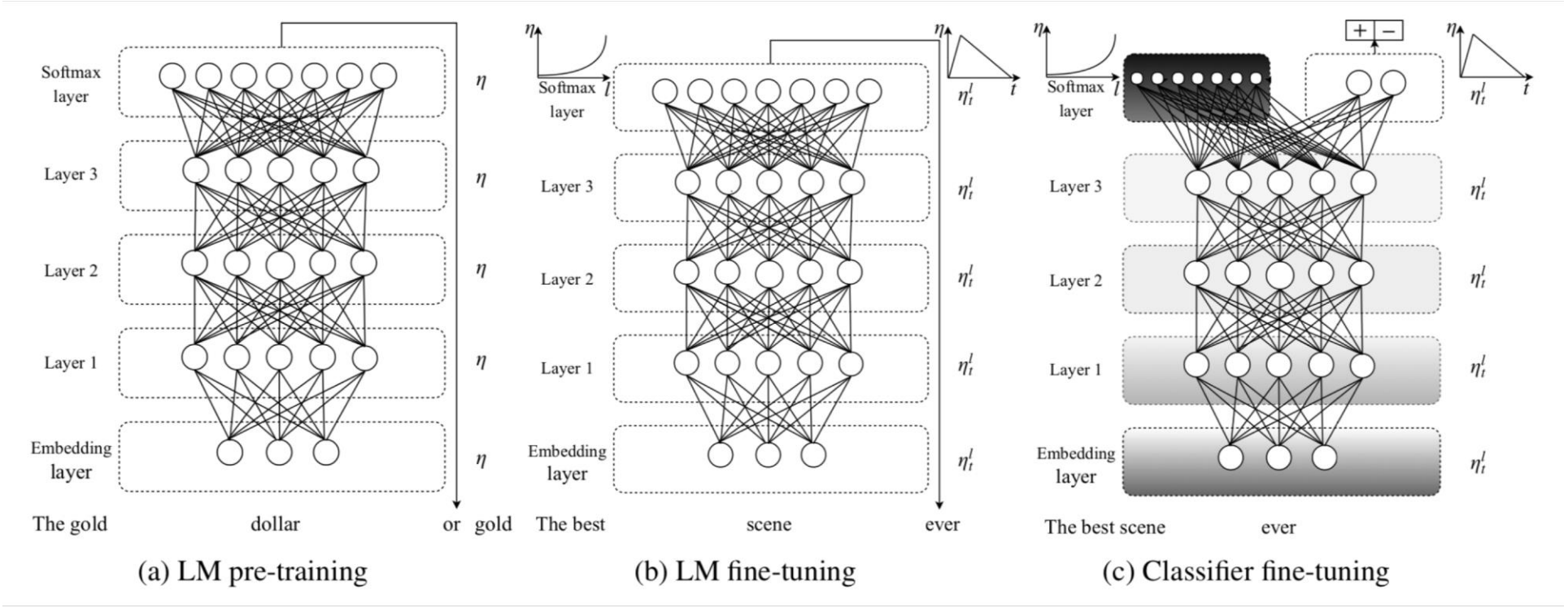
The general idea of transferring NLM knowledge

- Here applied to text classification



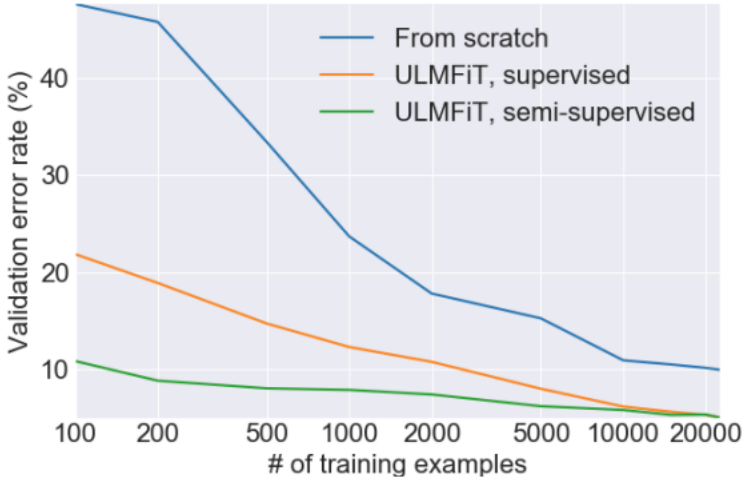
ULMFiT [Howard and Ruder, 2018]

1. Train LM on a large general-domain corpus (use biLM)
2. Fine-tune the LM on target task data
3. Fine-tune as a classifier on the target task



ULMFiT: Ablation Study

Sentiment Analysis on IMDb



Question Type Classification on TREC-6

News Topic Classification on AG

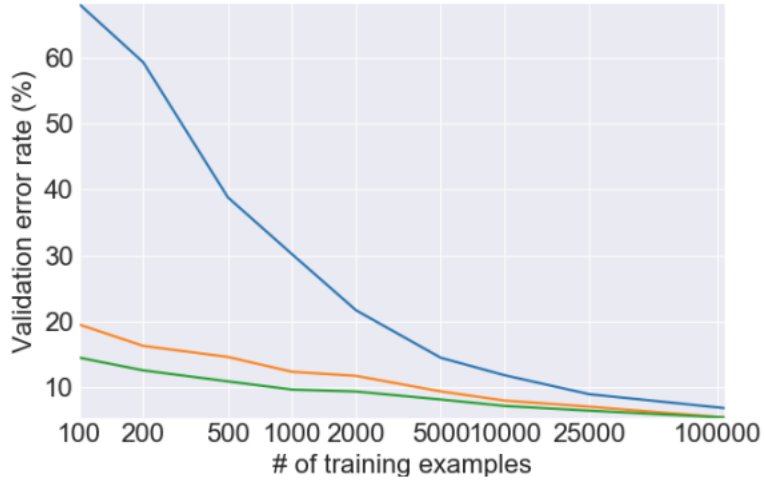


Figure 3: Validation error rates for supervised and semi-supervised ULMFiT vs. training from scratch with different numbers of training examples on IMDb, TREC-6, and AG (from left to right).

- Supervised: fine-tuning the LM on the labeled in-domain data only
- Semi-supervised: also use additional unlabeled in-domain data

Let's Scale It Up!

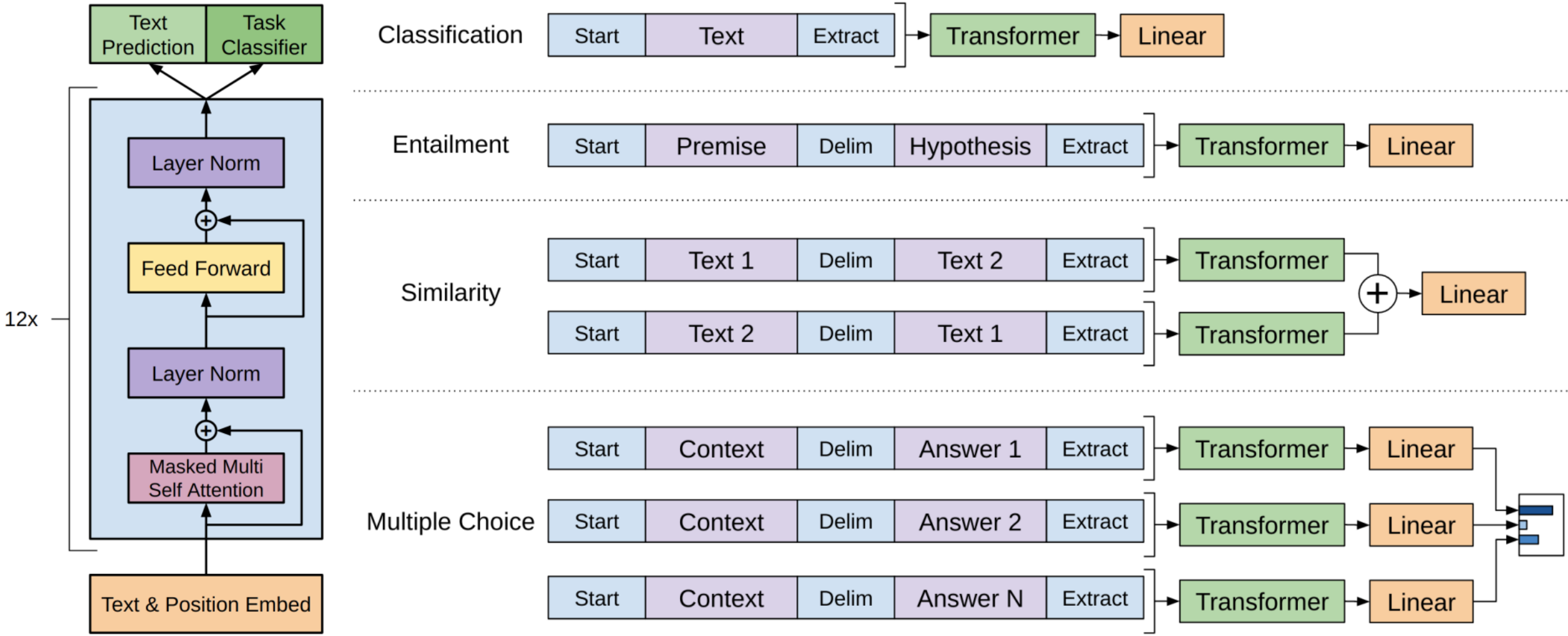
ULMfit	GPT	BERT	GPT-2
Jan 2018	June 2018	Oct 2018	Feb 2019
Training:	Training	Training	Training
1 GPU day	240 GPU days	256 TPU days	~2048 TPU v3 days according to a reddit thread
		~320–560 GPU days	

Fine-Tuning with GPT, GPT-2, GPT-3

Main Idea: Train a **Transformer Decoder** as conditional LM

Improving Language Understanding by Generative Pre-Training, OpenAI, 2018

Fine-Tuning with GPT, GPT-2, GPT-3



GPT-2 language Model (Cherry-Picked) Output

SYSTEM PROMPT (HUMAN-WRITTEN)	*In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.*
MODEL COMPLETION (MACHINE-WRITTEN, 10 TRIES)	The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science. Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved. Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow. Pérez and the others then ventured further into the valley. …

METRO
NEWS... BUT NOT AS YOU KNOW IT

NEWS SPORT ENTERTAINMENT SOAPS MORE TRENDING

UK WORLD WEIRD TECH

Elon Musk's OpenAI builds artificial intelligence so powerful it must be kept locked up for the good of humanity

Jasper Hamill Friday 15 Feb 2019 10:06 am

272 SHARES

Elon Musk's scientists have announced the creation of a terrifying artificial intelligence that's so smart they refused to release it to the public.

OpenAI's GPT-2 is designed to write just like a human and is an impressive leap forward capable of penning chillingly convincing text.

It was 'trained' by analysing eight million web pages and is capable of writing large tracts based upon a 'prompt' written by a real person.

But the machine mind will not be released in its fully-fledged form because of the risk of it being used for 'malicious purposes' such as generating fake news, impersonating people online, automating the production of spam or churning out 'abusive or faked content to post on social media'.

OpenAI wrote: 'Due to our concerns about malicious applications of the technology, we are not releasing the trained model.

35

Elon Musk
@elonmusk

Follow

Replying to @georgezachary

To clarify, I've not been involved closely with OpenAI for over a year & don't have mgmt or board oversight

8:19 PM - 16 Feb 2019

500 Retweets **14,573** Likes

229 500 15K

39

Transformer Models

All of these models are Transformer architecture models … so maybe we would better learn about Transformers?

ULMfit

Jan 2018

Training:

1 GPU day

GPT

June 2018

Training

240 GPU days

BERT

Oct 2018

Training

256 TPU days

~320–560 GPU days

GPT-2

Feb 2019

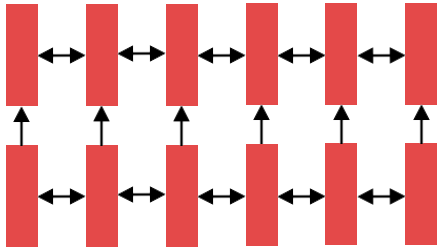
Training

~2048 TPU v3 days according to a reddit thread

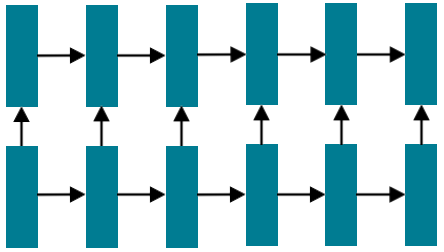
Motivation for the Transformer

2014-2017: Recurrent Models for (Most) NLP Tasks!

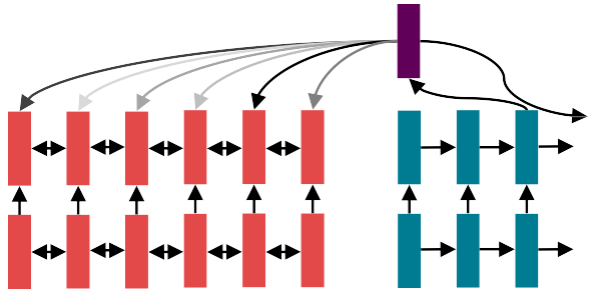
- Circa 2016, the de facto strategy in NLP was to **encode** sentences with a bidirectional LSTM: (for example, the source sentence in a translation)



- Define your output (parse, sentence, summary) as a sequence, and use an LSTM to generate it.

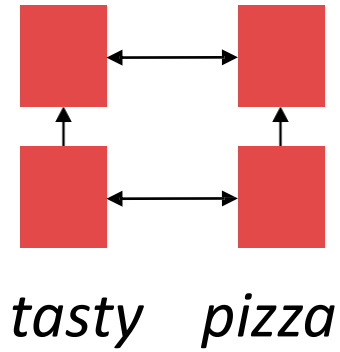


- Use attention to allow flexible access to memory

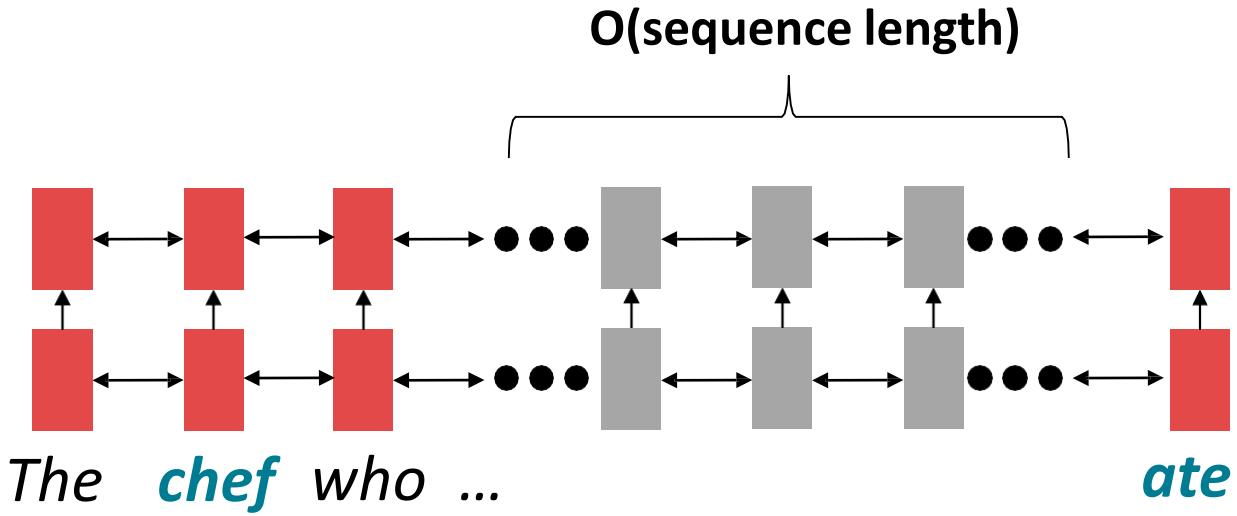


Issues with Recurrent Models: Linear Interaction Distance

- RNNs are unrolled "left-to-right".
- RNNs encode linear locality: a useful heuristic!
  - Nearby words often affect each other's meanings

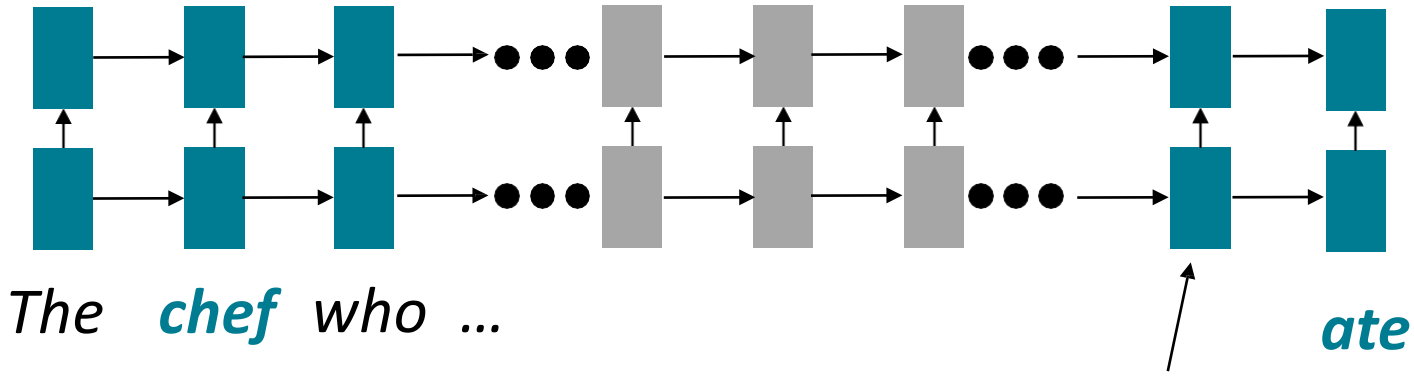


- **Problem:** RNNs take **O(sequence length)** steps for distant word pairs to interact.



Issues with Recurrent Models: Linear Interaction Distance

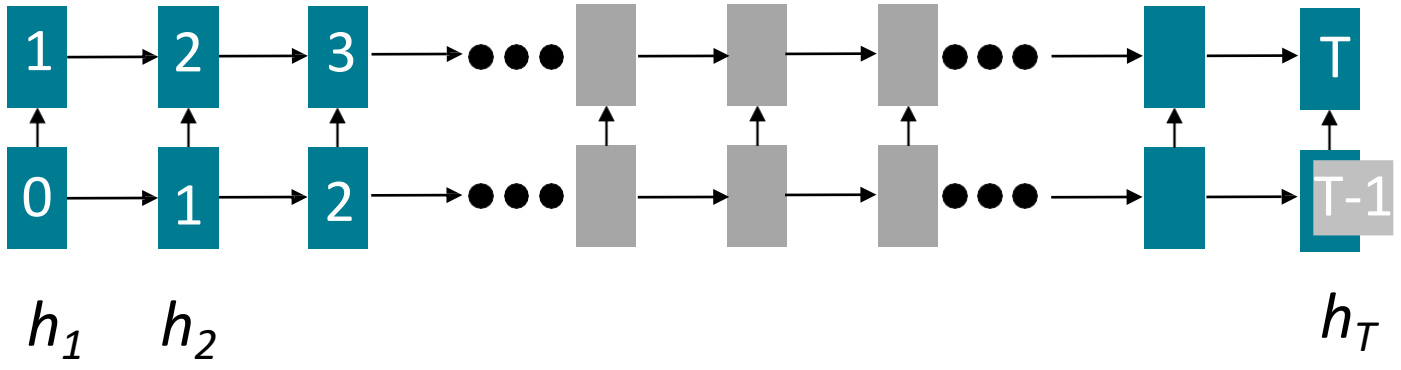
- **O(sequence length)** steps for distant word pairs to interact means:
  - Hard to learn long-distance dependencies (because of gradient problems!)
  - The linear order of the words is "baked in"
    - BUT we already know that sequential structure does not tell the whole story...



Info of ***chef*** has gone through O(sequence length) many layers!

Issues with Recurrent Models: Lack of Parallelizability

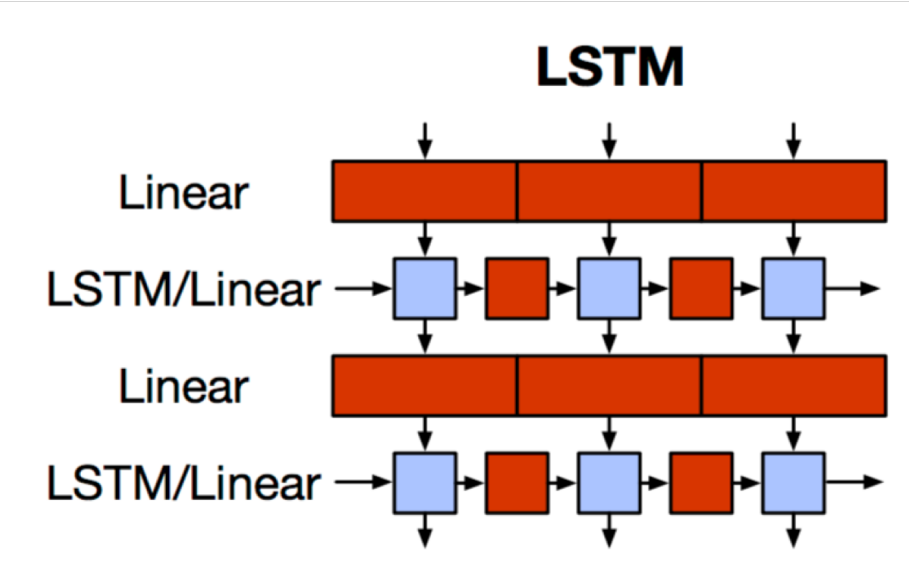
- Forward and backward passes have **O(seq length)** unparallelizable operations
  - GPUs (and TPUs) can perform many independent computations at once!
  - But future RNN hidden states cannot be computed in full before past RNN hidden states have been computed
  - Inhibits training on very large datasets!
  - Particularly problematic as the sequence length increases, as we can no longer batch many examples together due to memory limitations



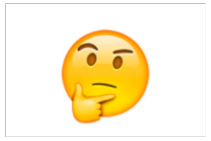
Numbers indicate min # of steps before a state can be computed

Motivation for Transformers

- We want **parallelization**, but RNNs are inherently sequential

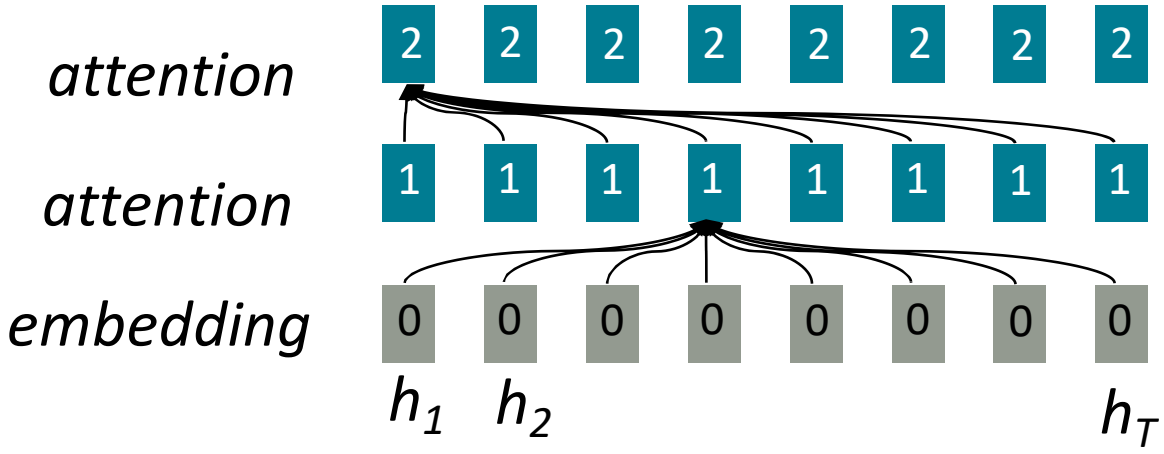


- Despite GRUs and LSTMs, RNNs still need attention mechanism to deal with long-range dependencies: the **path length** between states grows with sequence otherwise
- But if **attention** gives us access to any state… maybe we can just use attention and we do not need the RNN?



If not Recurrence, Then What? How About (Self) Attention?

- To recap, **attention** treats each word's representation as a **query** to access and to incorporate information from **a set of values.**
  - With LSTM seq2seq, we saw attention from the **decoder** to the **encoder**;
  - **Self-attention** is **encoder-encoder** (or **decoder-decoder**) attention where each word attends to each other word **within the input (or output)**.



All words attend to all words in the previous layer; most arrows here are omitted

Transformers in NLP

Attention Is All You Need

https://papers.nips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

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

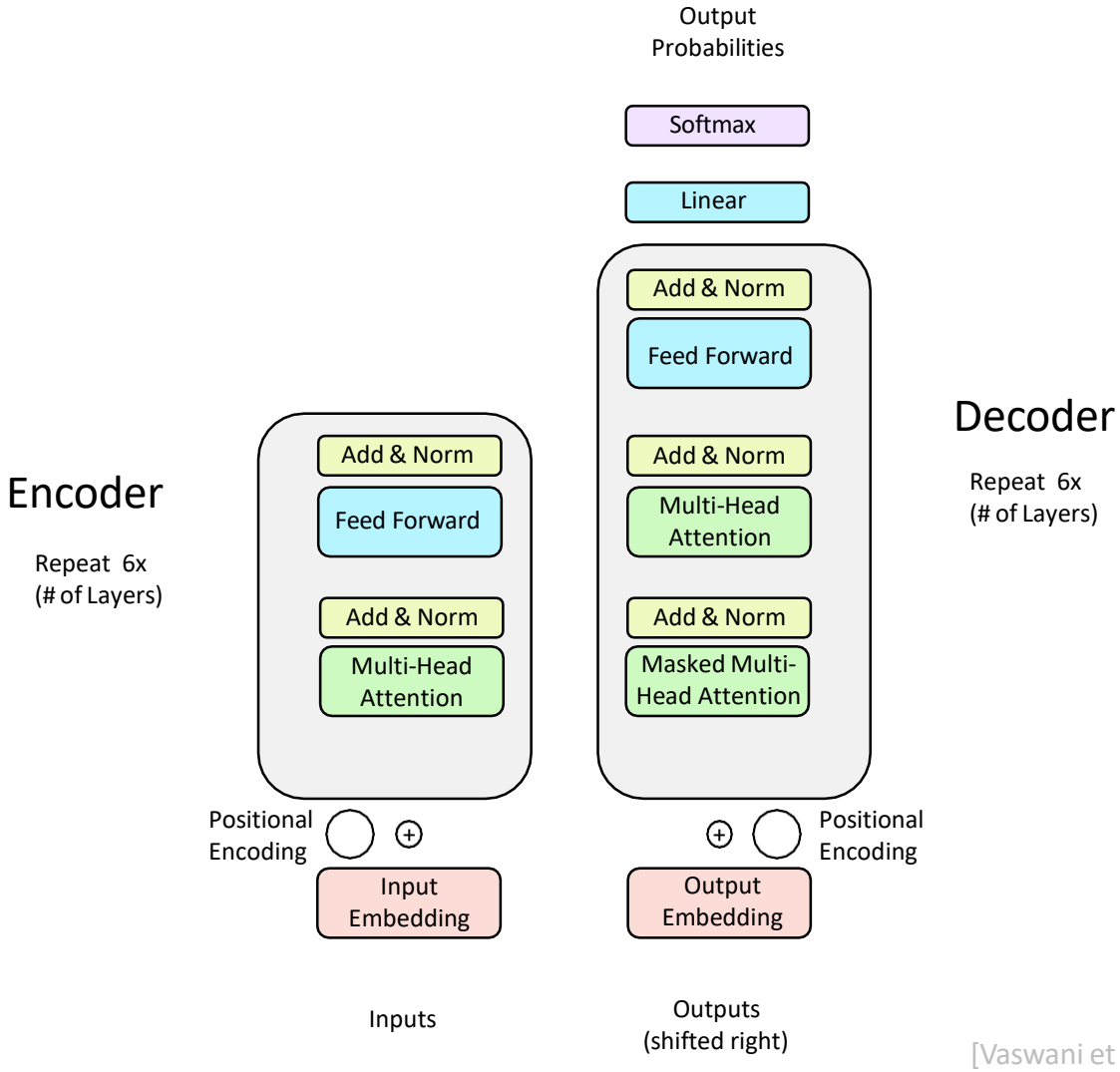
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Transformers Have Revolutionized the Field of NLP

Courtesy of Paramount Pictures



[Vaswani et al., 2017]

Great Results with Transformers: Machine Translation

Machine Translation results from the original Transformers paper

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**	$2.3 \cdot 10^{19}$	

[Test sets: WMT 2014 English-German and English-French]

[Vaswani et al., 2017]

Great Results with Transformers: Document Generation

Next, document generation!

(For perplexity, lower is better; for ROUGE-L, higher is better.)

Model	Test perplexity	ROUGE-L
seq2seq-attention, $L = 500$	5.04952	12.7
Transformer-ED, $L = 500$	2.46645	34.2
Transformer-D, $L = 4000$	2.22216	33.6
Transformer-DMCA, no MoE-layer, $L = 11000$	2.05159	36.2
Transformer-DMCA, MoE-128, $L = 11000$	1.92871	37.9
Transformer-DMCA, MoE-256, $L = 7500$	1.90325	38.8

The old standard from LSTMs!

Transformers dominating across the board.

[Liu et al., 2018]; WikiSum dataset

Preview: Great Results with (Pre-Trained) Transformers

Transformers' parallelizability allows for efficient pretraining, and have made them the de-facto standard.

On this popular aggregate benchmark, for example:

All top models are Transformer (and pretraining)-based.

Rank	Name	Model	URL	Score
1	DeBERTa Team - Microsoft	DeBERTa / TuringNLRv4		90.8
2	HFL iFLYTEK	MacALBERT + DKM		90.7
3	Alibaba DAMO NLP	StructBERT + TAPT		90.6
4	PING-AN Omni-Sinitic	ALBERT + DAAF + NAS		90.6
5	ERNIE Team - Baidu	ERNIE		90.4
6	T5 Team - Google	T5		90.3

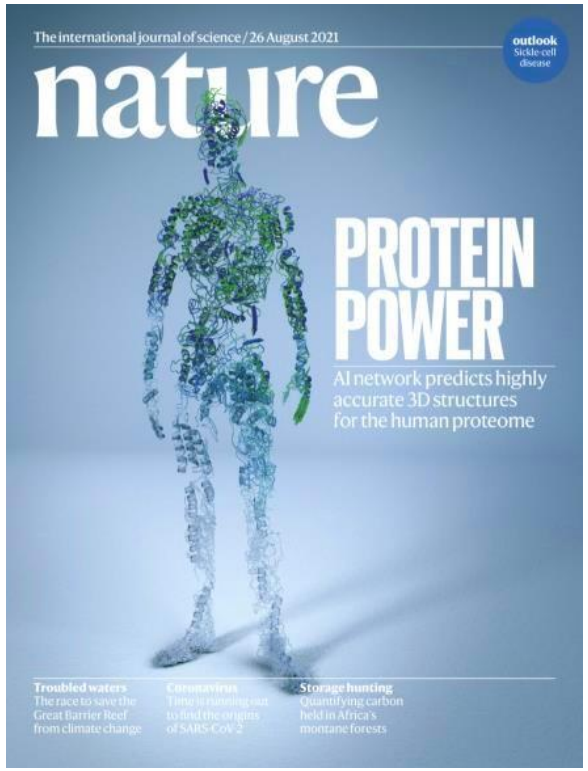
[Liu et al., 2018]

Transformers Beyond NLP

Transformers Even Show Promise Outside of NLP

Transformers Even Show Promise Outside of NLP

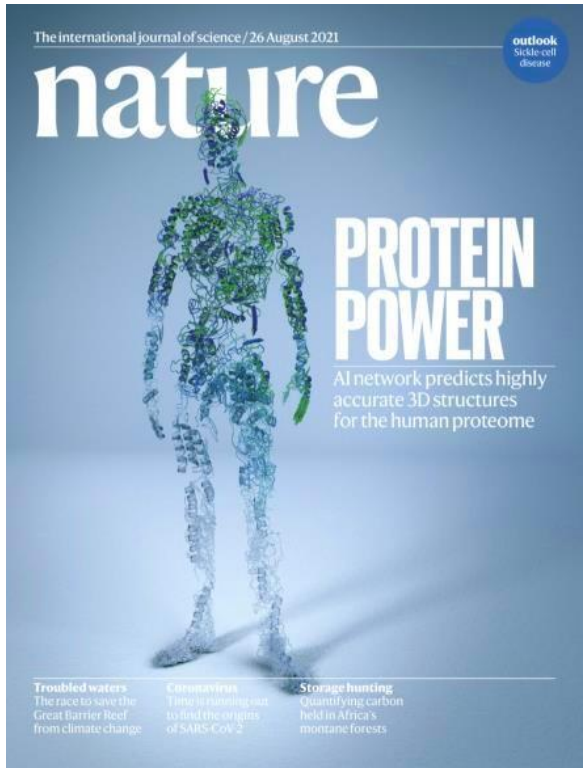
Protein Folding



[Jumper et al. 2021] aka AlphaFold2!

Transformers Even Show Promise Outside of NLP

Protein Folding



[Jumper et al. 2021] aka AlphaFold2!

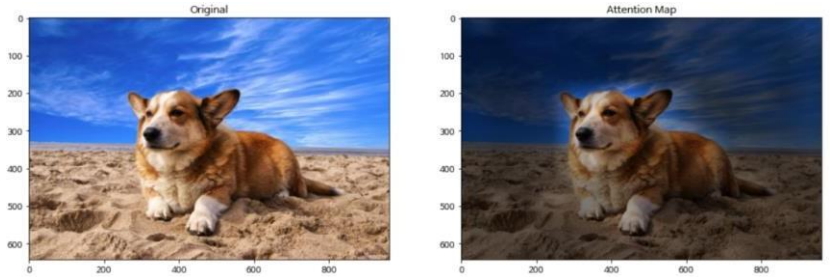


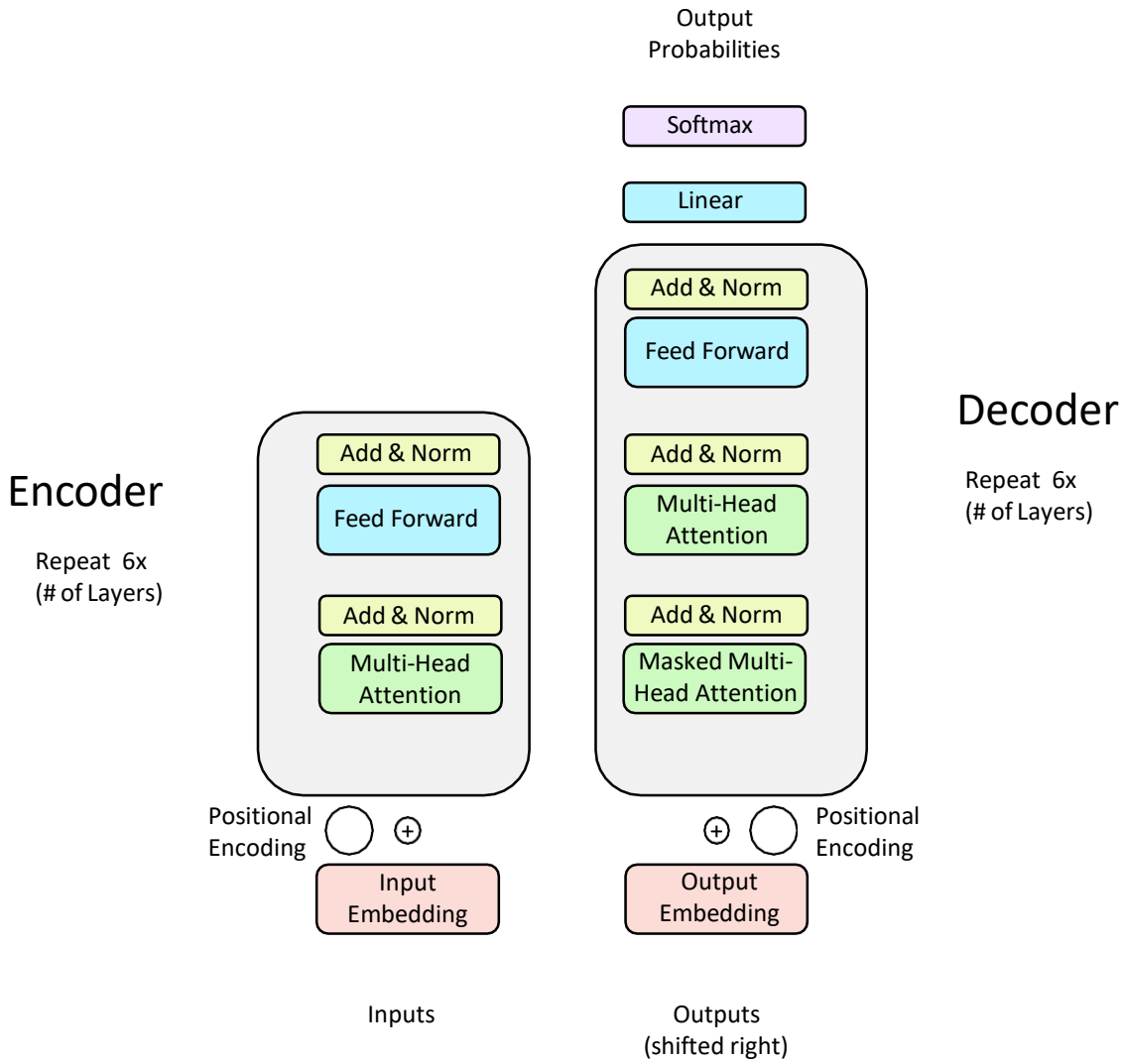
Image Classification

[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

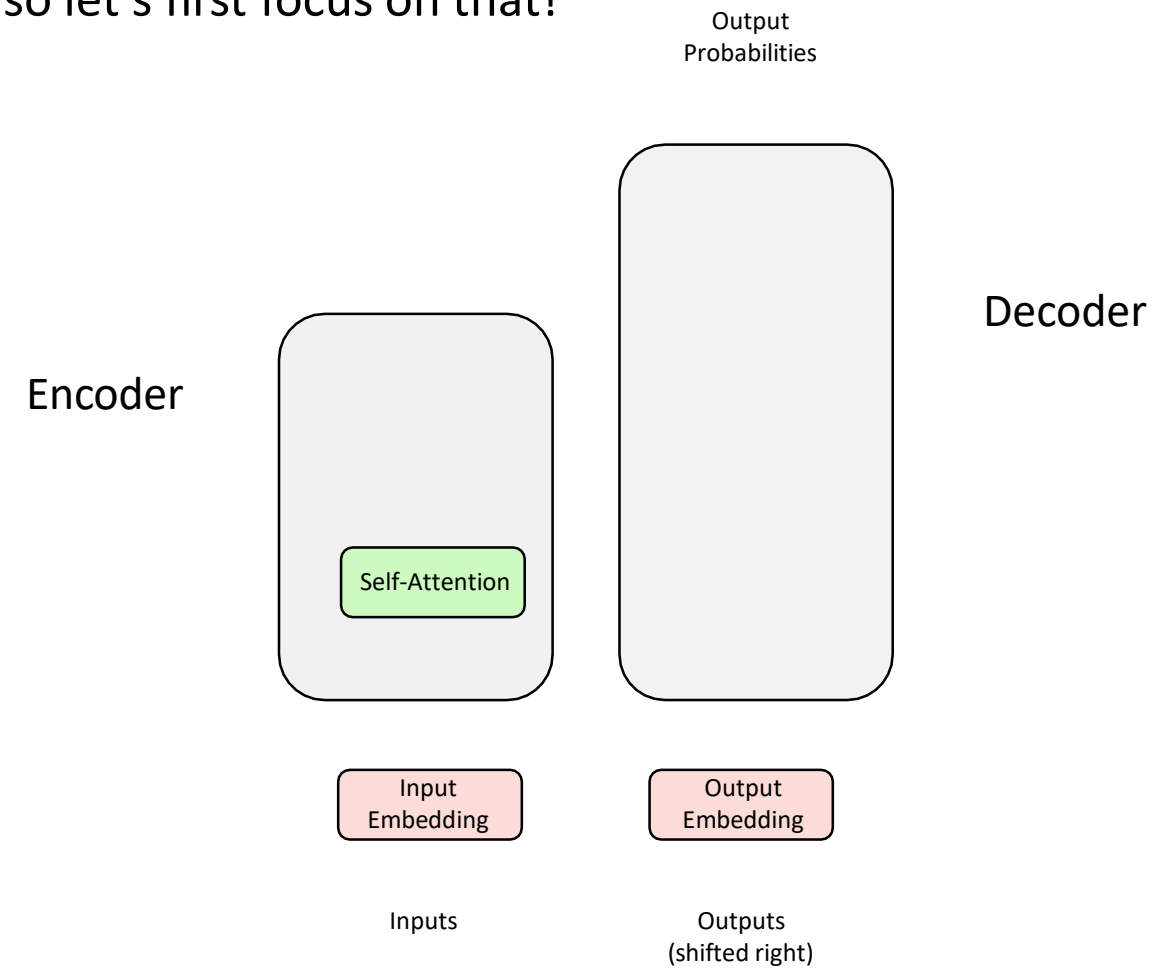
The Transformer Architecture

The Transformer Encoder-Decoder [Vaswani et al., 2017]



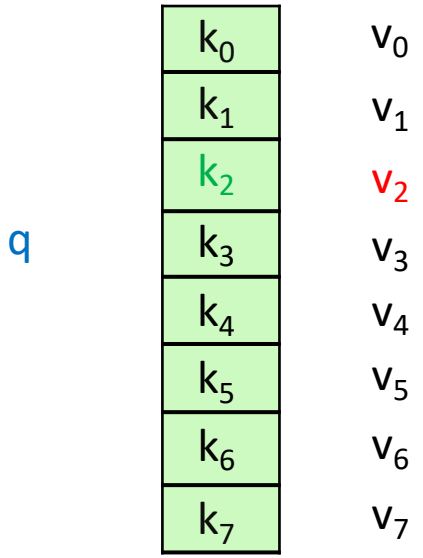
Encoder: Self-Attention

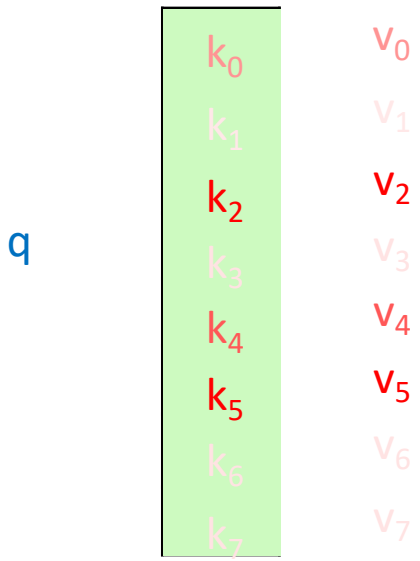
Self-Attention is the core building block of the Transformer, so let's first focus on that!



Intuition for the Attention Mechanism

- We can think of attention as a "fuzzy" or approximate hashtable:
  - To look up a **value**, we compare a **query** against **keys** in a table.
  - In a hashtable (shown on the bottom left):
    - Each **query** (hash) maps to exactly one **key**-**value** pair.
  - In (self-)attention (shown on the bottom right):
    - Each **query** matches each **key** to varying degrees.
    - We return a sum of **values** weighted by the **query**-**key** match.





Recipe for Self-Attention in the Transformer Encoder

- 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 the **query** and the **keys.**

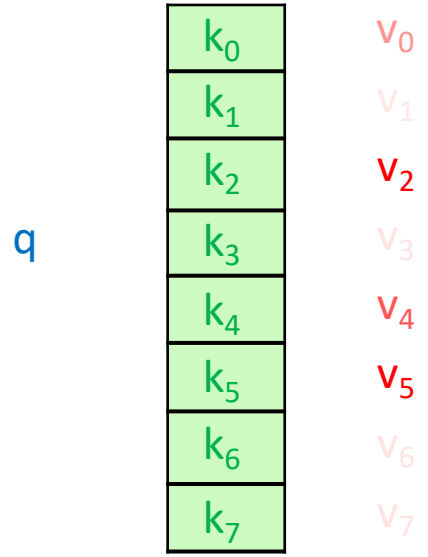
$$e_{ij} = q_i \cdot k_j$$

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

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

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

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



Recipe for (Vectorized) Self-Attention in the Transformer Encoder

- Step 1: With embeddings stacked in *X*, calculate **queries**, **keys**, and **values**.

$$Q = XW^Q \quad K = XW^K \quad V = XW^V$$

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

$$E = QK^T$$

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

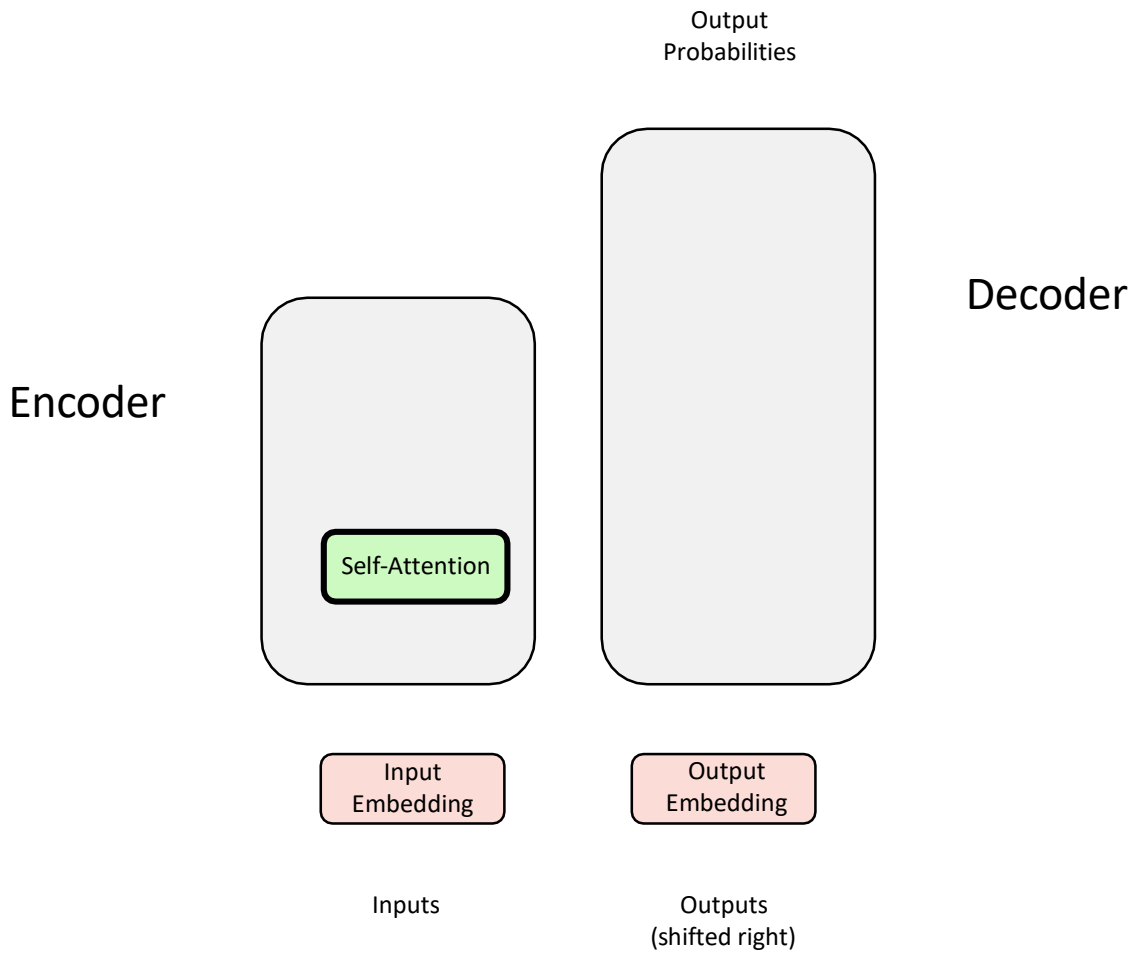
$$A = softmax(E)$$

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

$$Output = AV$$

$$Output = softmax\left(QK^T\right)V$$

What We Have So Far: (Encoder) Self-Attention!



Attention is Not Quite All You Need!

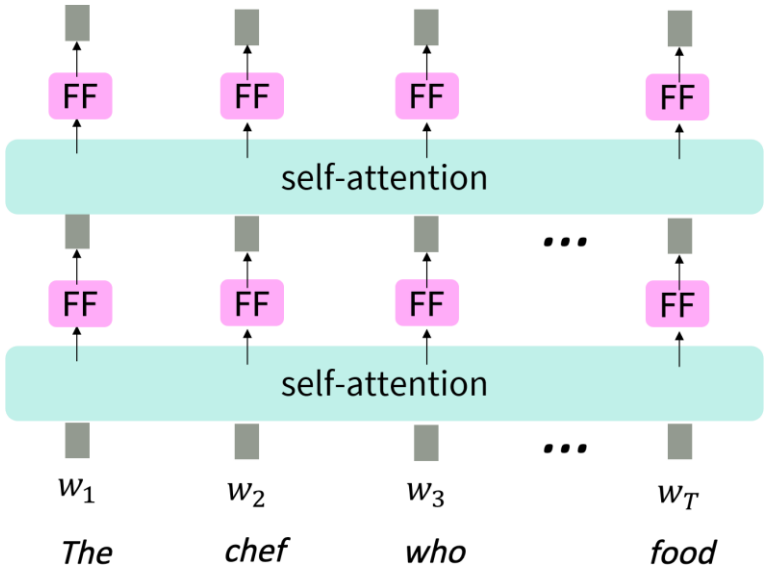
- **Problem:** Since there are no element-wise non-linearities, self-attention is simply performing a re-averaging of the value vectors.
- **Question:** Why is this a problem?

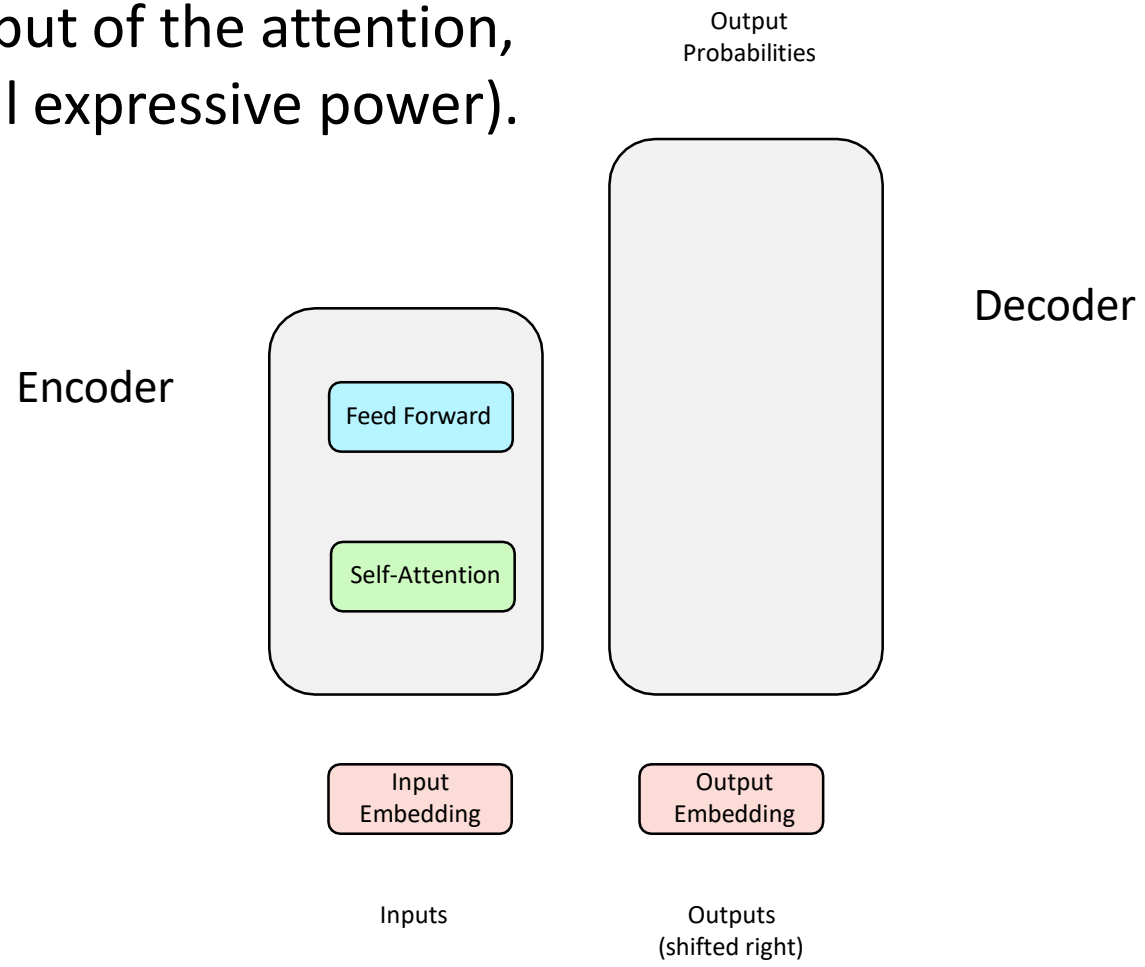
Attention is Not Quite All You Need!

- **Problem:** Since there are no element-wise non-linearities, self-attention is simply performing a re-averaging of the value vectors.
- **Easy fix:** Apply a feedforward layer to the output of the attention, providing non-linear activation (and additional expressive power).

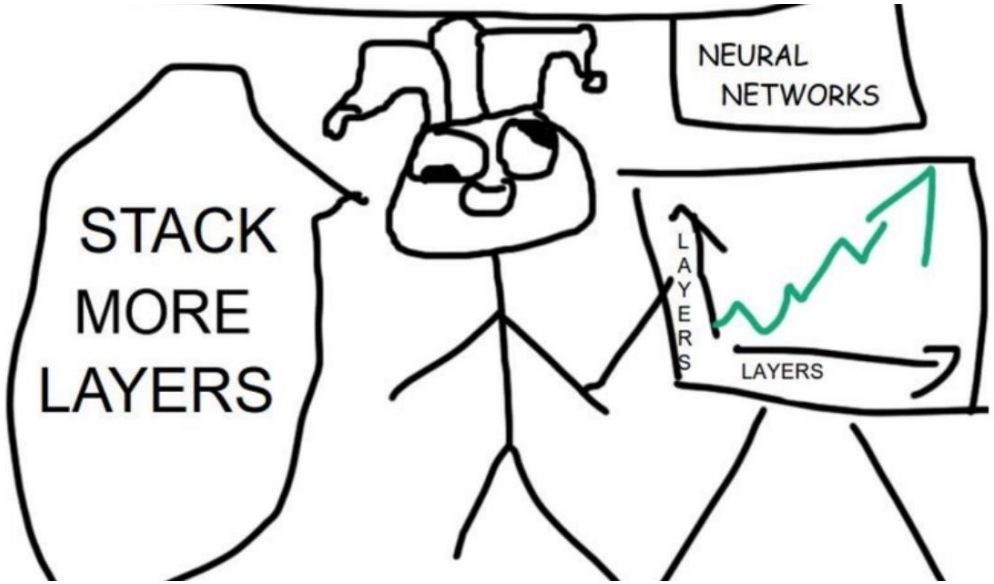
Equation for a Feed-Forward Layer

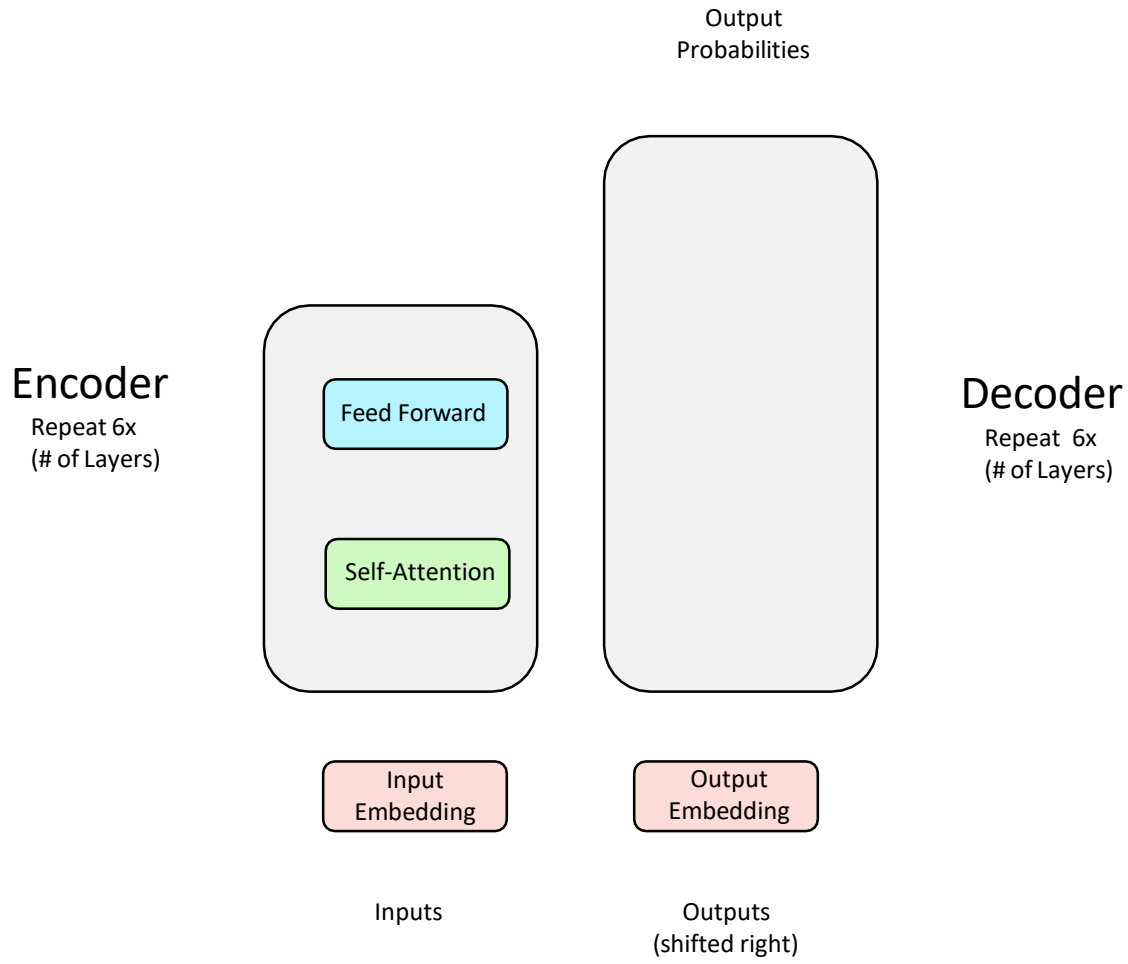
$$m_i = MLP(\text{output}_i)$$
$$= W_2 * \text{ReLU}(W_1 \times \text{output}_i + b_1) + b_2$$





How Do We Make This Work for Deep Networks?





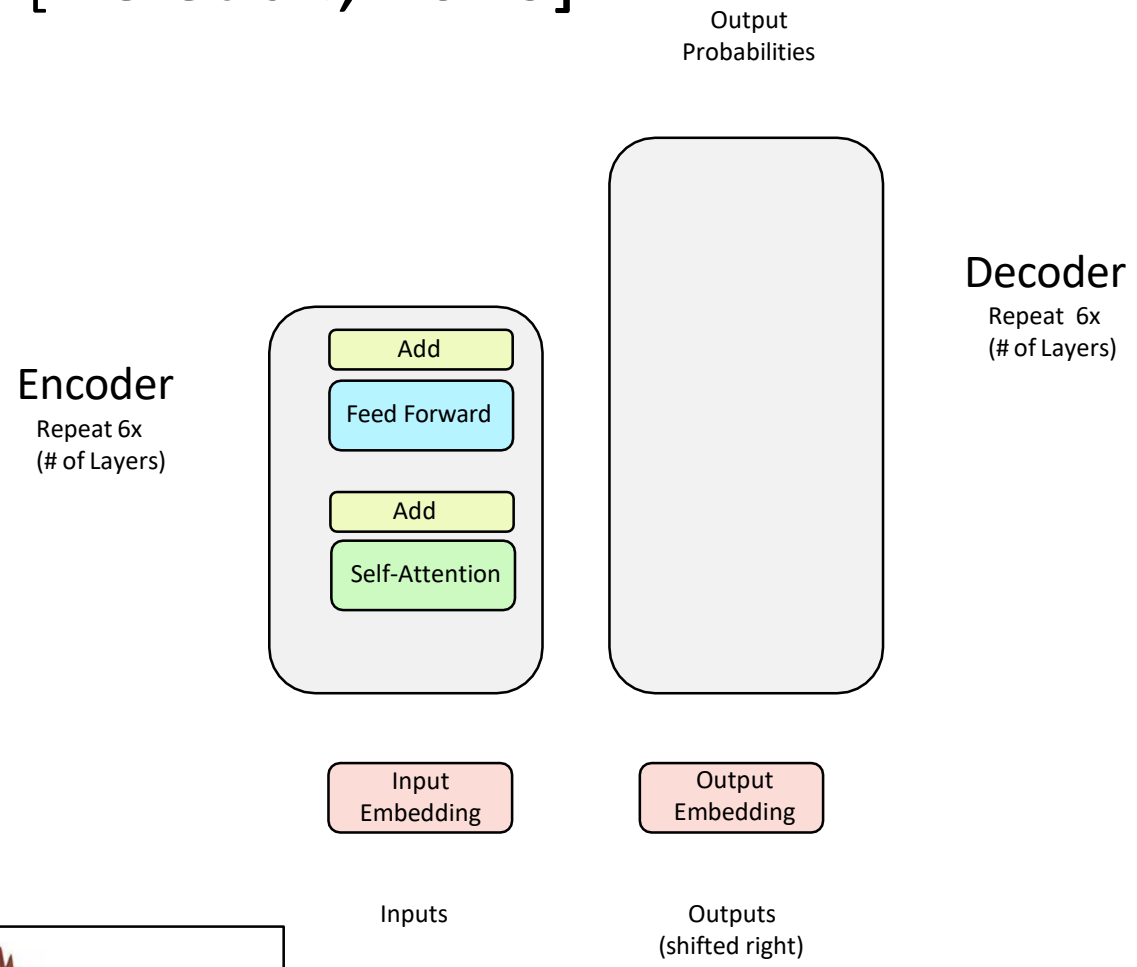
Training Trick #1: Residual Connections
Training Trick #2: LayerNorm
Training Trick #3: Scaled Dot Product Attention

Training Trick #1: Residual Connections [He et al., 2016]

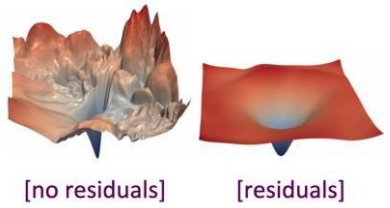
- Residual connections are a simple but powerful technique from computer vision.
- Deep networks are surprisingly bad at learning the identity function!
- Therefore, directly passing "raw" embeddings to the next layer can actually be very helpful!

$$x_\ell = F(x_{\ell-1}) + x_{\ell-1}$$

- This prevents the network from "forgetting" or distorting important information as it is processed by many layers.



Residual connections are also thought to smooth the loss landscape and make training easier!



[Loss landscape visualization, Li et al., 2018, on a ResNet]

Training Trick #2: Layer Normalization [Ba et al., 2016]

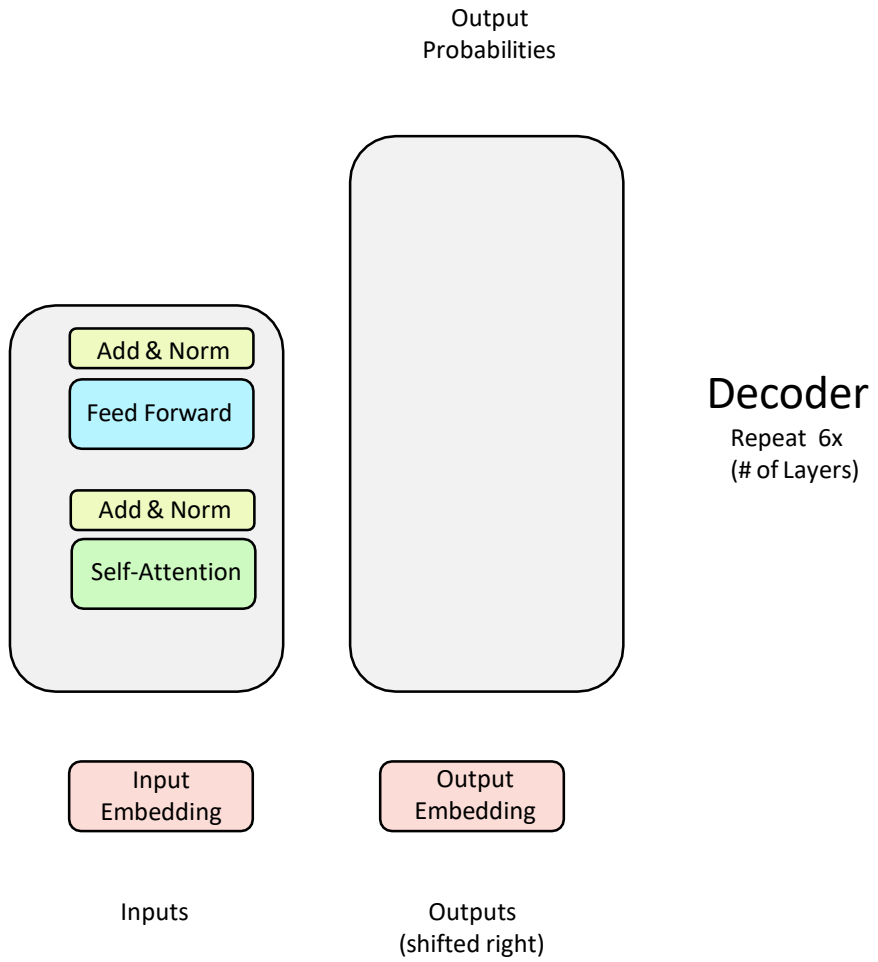
- **Problem:** Difficult to train the parameters of a given layer because its input from the layer beneath keeps shifting.
- **Question:** Why is this a problem?

Training Trick #2: 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 the uninformative 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: $\sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2}$

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

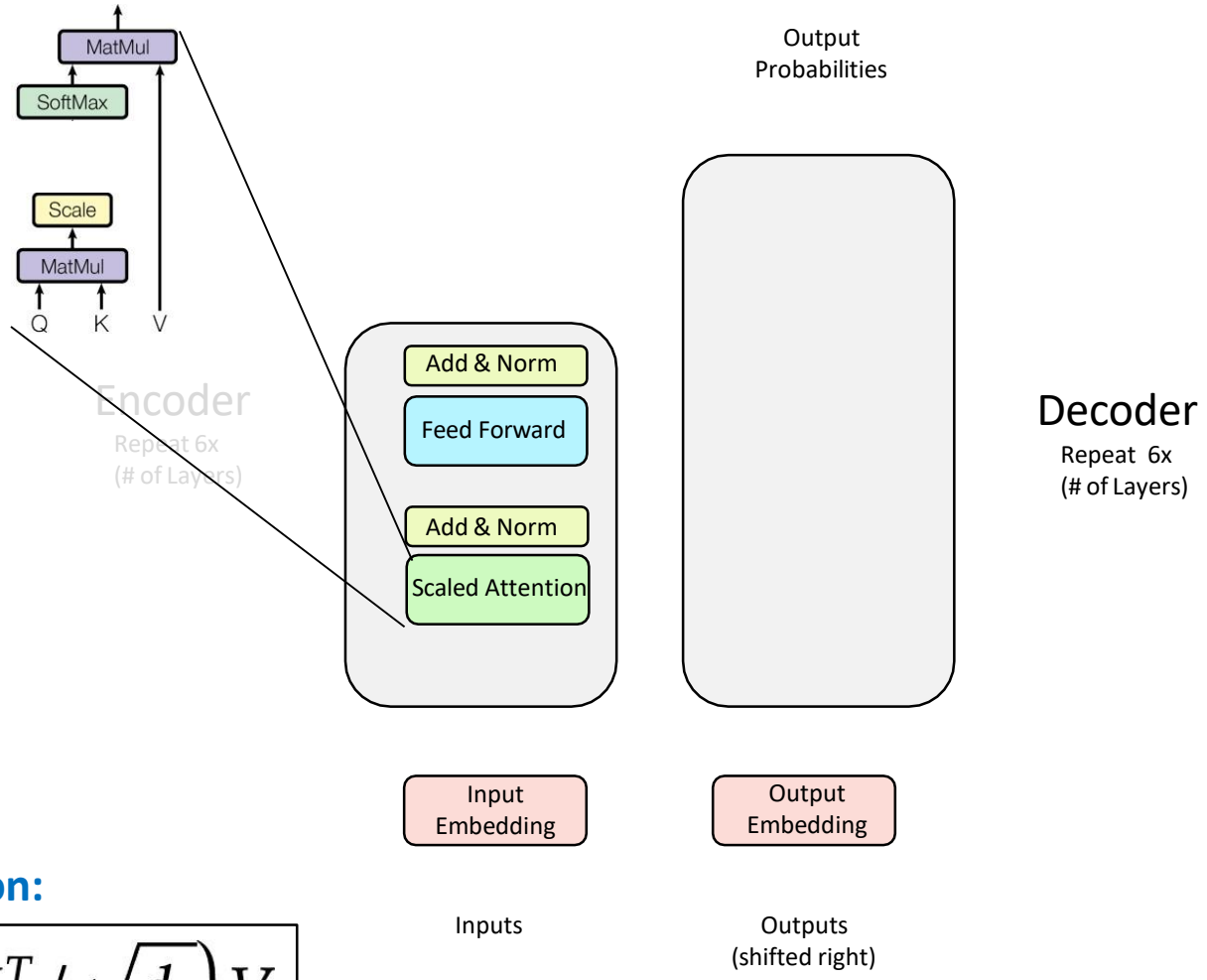


Training Trick #3: Scaled Dot Product Attention

- After LayerNorm, the mean and the variance of the vector elements is 0 and 1, respectively.
- However, the dot product still tends to take on extreme values, as its variance scales with dimensionality d_k

Quick Statistics Review:

- Mean of sum = sum of means = $d_k * 0 = 0$
- Variance of sum = sum of variances = $d_k * 1 = d_k$
- To set the variance to 1, simply divide by $\sqrt{d_k}$!



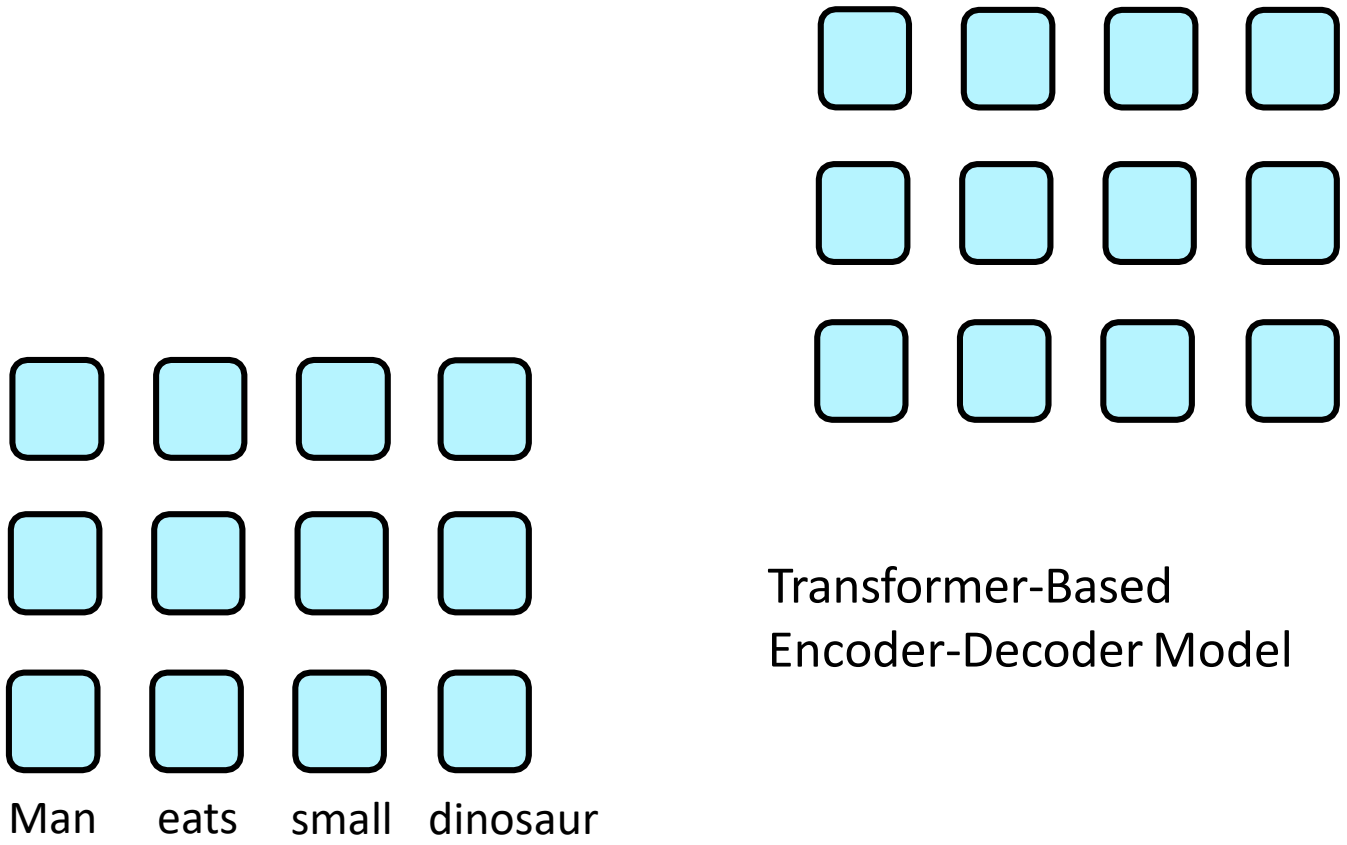
Updated Self-Attention Equation:

$$Output = softmax\left(QK^T / \sqrt{d_k}\right)V$$

Major Issue!

- We are almost done with the Encoder, but we have a major problem! What is it?

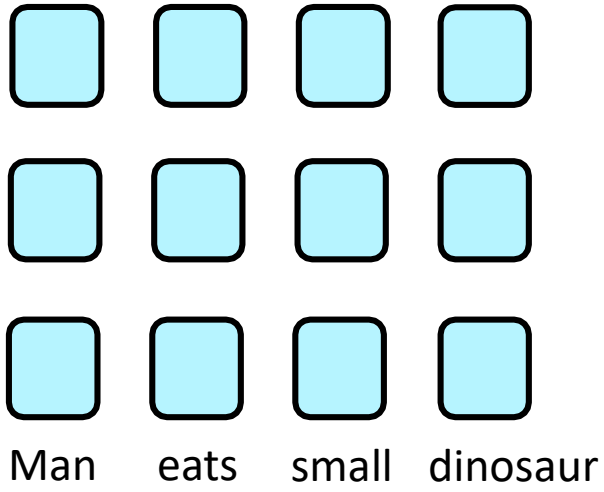
$$Output = softmax\left(QK^T / \sqrt{d_k}\right)V$$

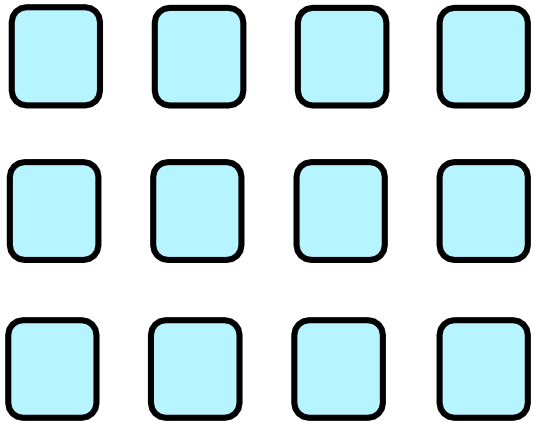


Major Issue!

- We are almost done with the Encoder, but we have a major problem! What is it?
- Consider this sentence:
  - "Man eats small dinosaur."
- **We do not model word order!**

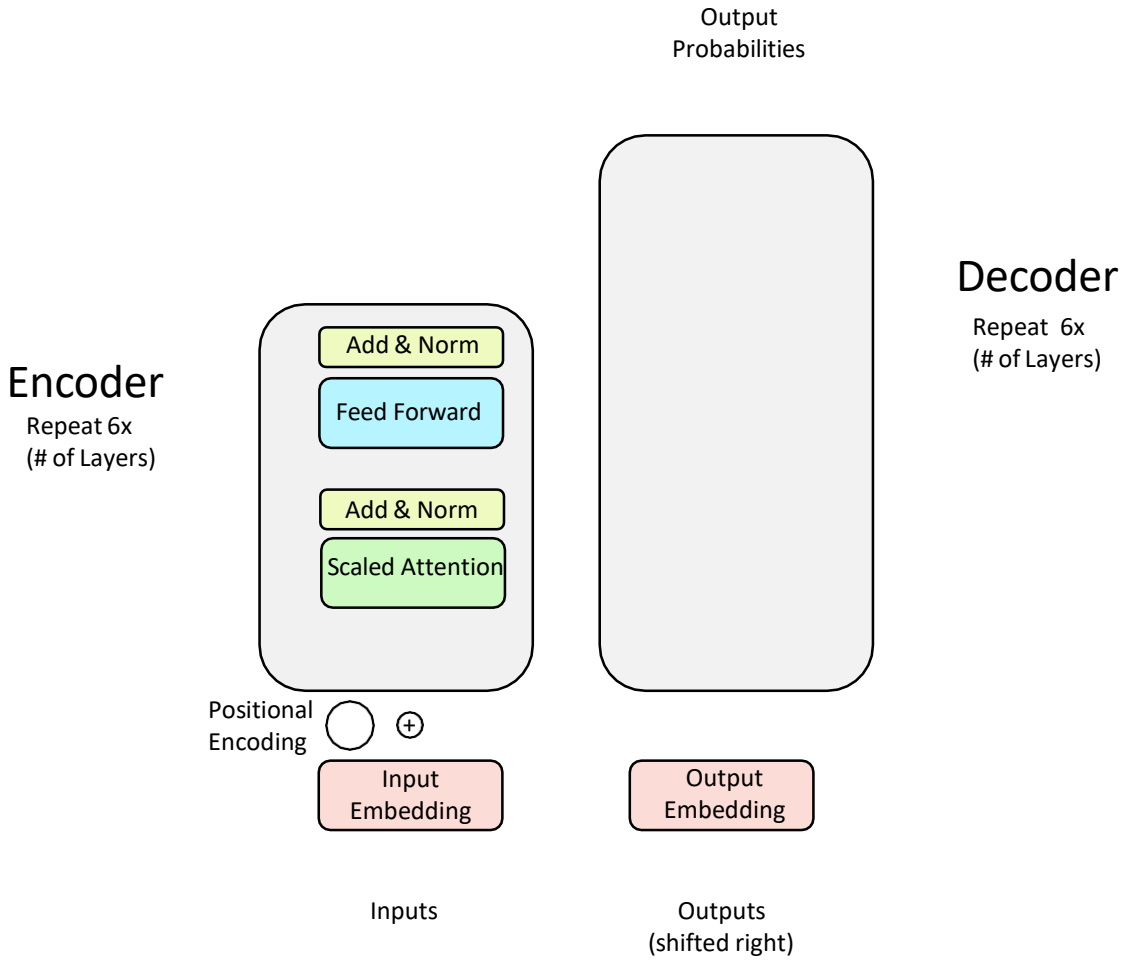
$$Output = softmax\left(QK^T / \sqrt{d_k}\right)V$$





Transformer-Based
Encoder-Decoder Model

Solution: Inject Order Information Through Positional Encodings!



Fixing the First Self-Attention Problem: Sequence Order

- Since self-attention doesn't build in order information, we need to encode the order of the sentence in our keys, queries, and values.
- Consider representing each **sequence index** as a **vector**

$$p_i \in \mathbb{R}^d, \text{ for } i \in \{1,2,\ldots,T\} \text{ are position vectors}$$

- Don't worry about what the p_i are made of yet!
- Easy to incorporate this info into our self-attention block: just add the p_i to our inputs!
- Let $\tilde{v}_i\ \tilde{k}_i, \tilde{q}_i$ be our old values, keys, and queries.

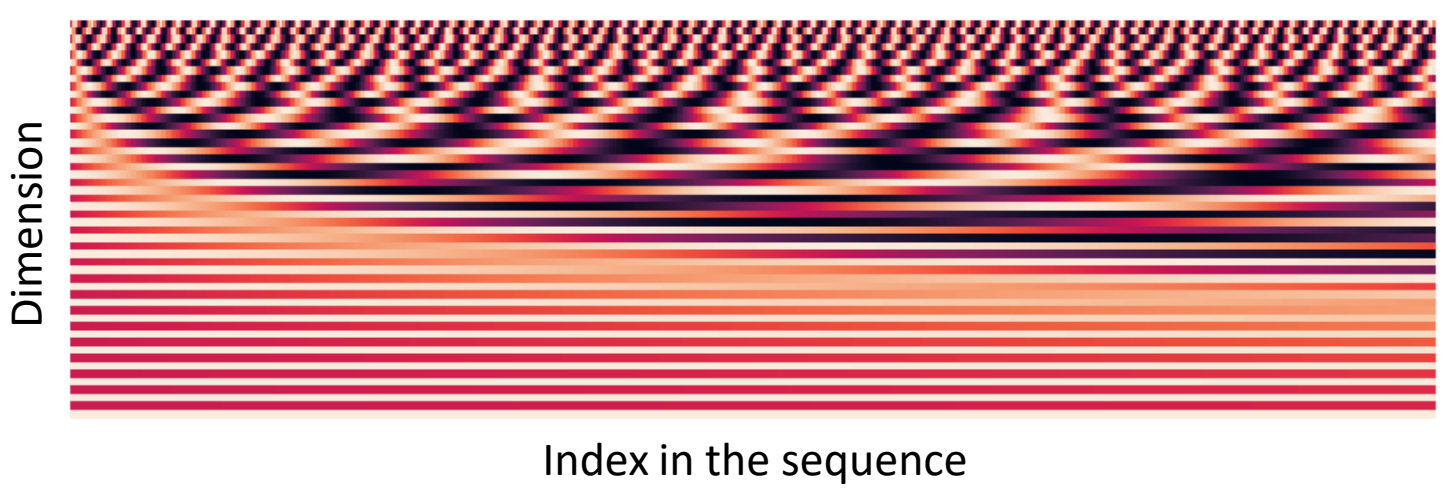
$$v_i = \tilde{v}_i + p_i$$
$$q_i = \tilde{q}_i + p_i$$
$$k_i = \tilde{k}_i + p_i$$

In deep self-attention networks, we do this at the first layer! You could concatenate them as well, but people mostly just add…

Position Representation Vectors Through Sinusoids

- **Sinusoidal position representations:** concatenate sinusoidal functions of varying periods:

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



- Pros:
  - Periodicity indicates that **maybe the "absolute position" is not as important**
  - Maybe we can extrapolate to longer sequences as periods restart
- Cons:
  - **Not learnable**; also the extrapolation does not really work

Image: https://timodenk.com/blog/linear-relationships-in-the-transformers-positional-encoding/

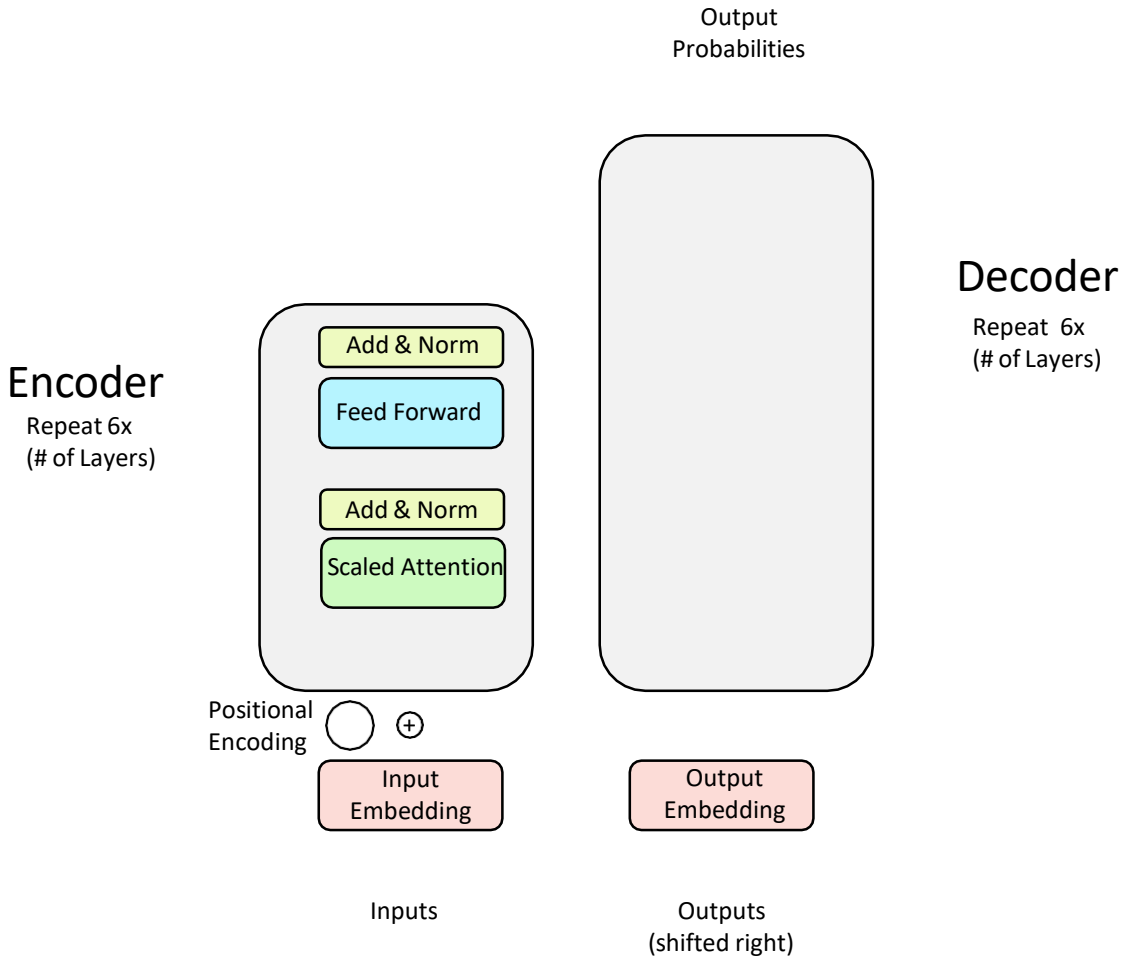
Position Representation vectors learned from scratch

- **Learned absolute position representations:** Let all p_i be learnable parameters!
  Learn a matrix $p \in \mathbb{R}^{d \times T}$, *and let each* p_i *be a column of that matrix!*

- Pros:
  - Flexibility: each position gets to be learned to fit the data
- Cons:
  - Definitely cannot extrapolate to indices outside $1, \ldots, T$.
- Most systems use this!

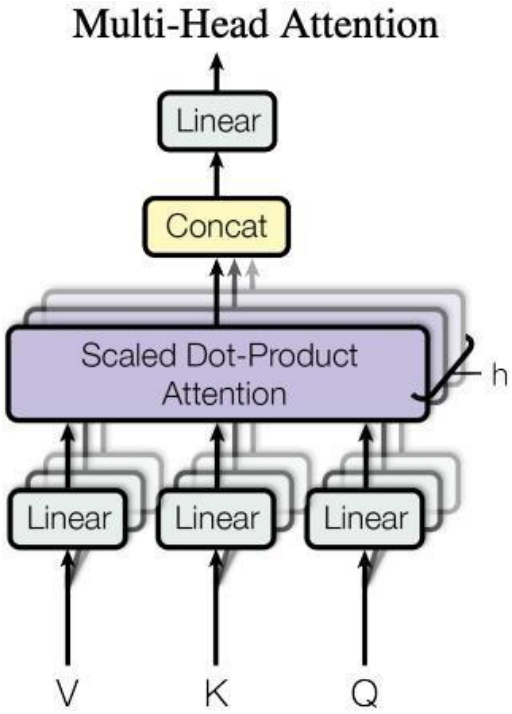
- Sometimes people try more flexible representations of position:
  - Relative linear position attention [Shaw et al., 2018]
  - Dependency syntax-based position [Wang et al., 2019]

Solution: Inject Order Information through Positional Encodings!



Multi-Headed Self-Attention: *k* Heads are Better Than 1!

- **High-Level Idea:** Let us perform self-attention multiple times in parallel and combine the results.



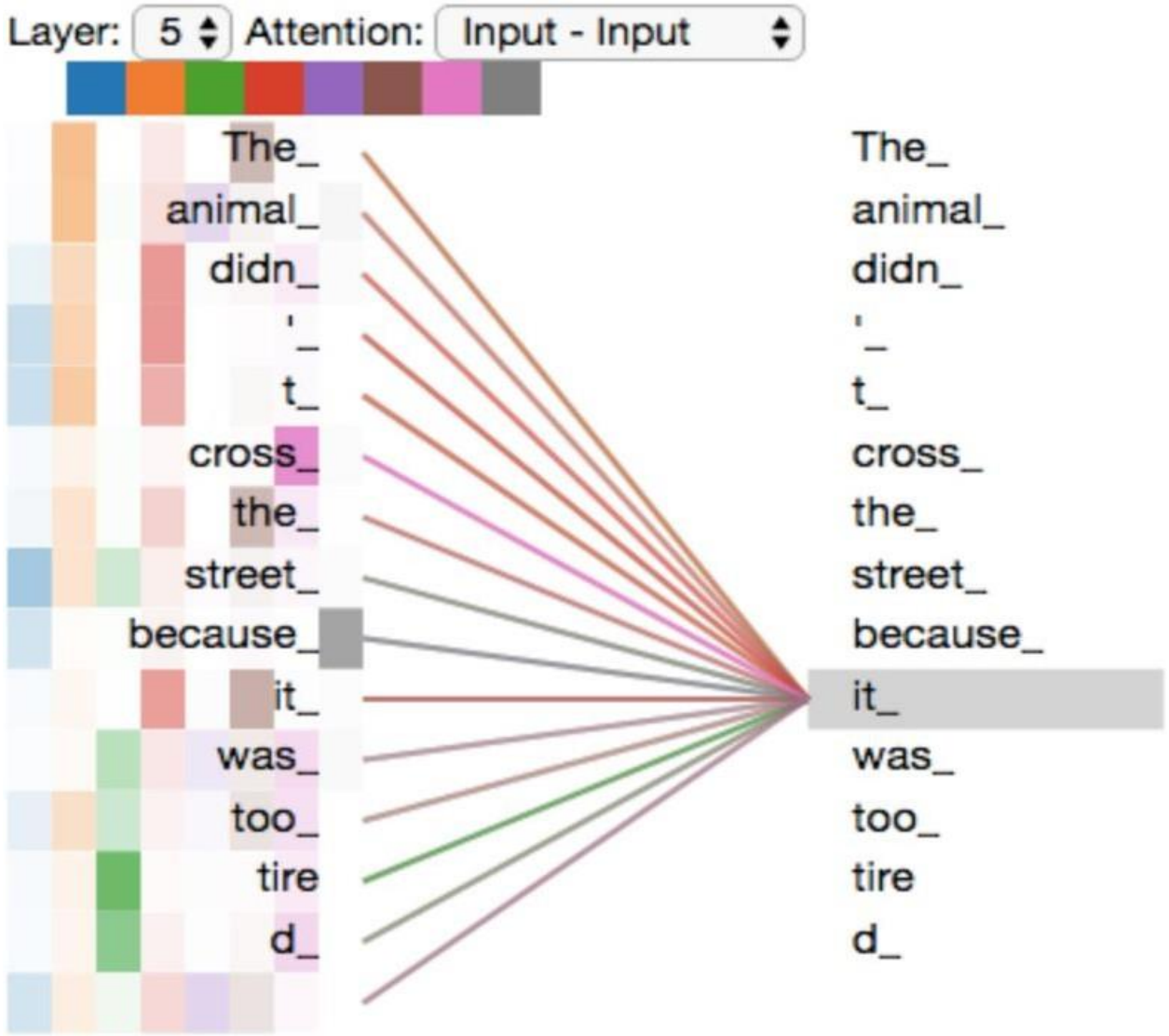
[Vaswani et al. 2017]

Wizards of the Coast, Artist: Todd Lockwood

The Transformer Encoder: Multi-Headed Self-Attention

- What if we want to look in multiple places in the sentence at once?
  - For word i, self-attention "looks" where $x_i^\top Q^\top K x_j$ is high, but maybe we want to focus on different j for different reasons?
- We'll define **multiple attention "heads"** through multiple Q,K,V matrices
- Let, $Q_\ell, K_\ell, V_\ell \in \mathbb{R}^{d \times \frac{d}{h}}$, where h is the number of attention heads, and ℓ ranges from 1 to h.
- Each attention head performs attention independently:
  - $\text{output}_\ell = \text{softmax}\left(XQ_\ell K_\ell^\top X^\top\right) * XV_\ell$, where $\text{output}_\ell \in \mathbb{R}^{d/h}$
- Then the outputs of all the heads are combined!
  - $\text{output} = Y[\text{output}_1; \ldots; \text{output}_h]$, where $Y \in \mathbb{R}^{d \times d}$

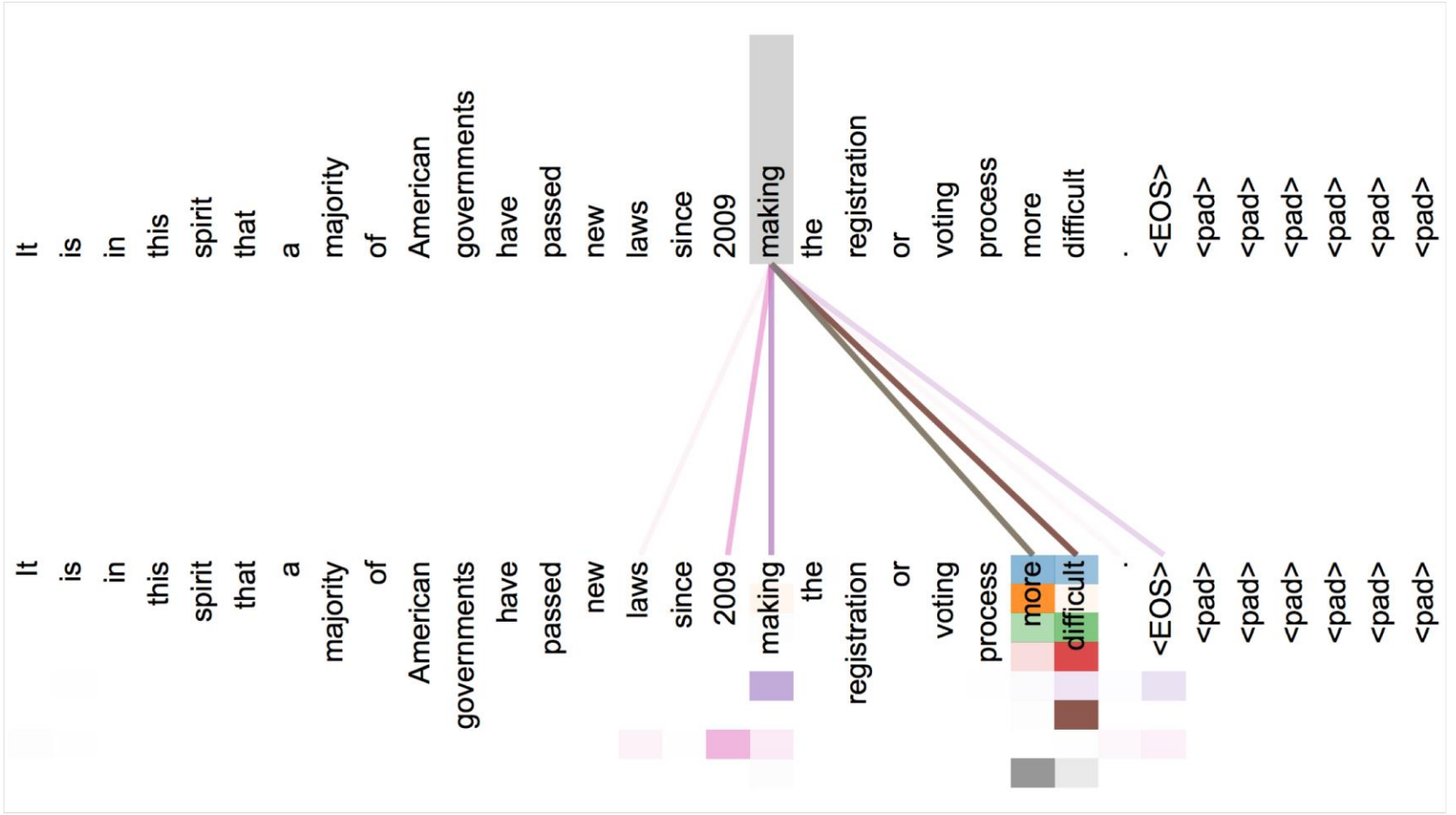
The Transformer Encoder: Multi-Headed Self-Attention



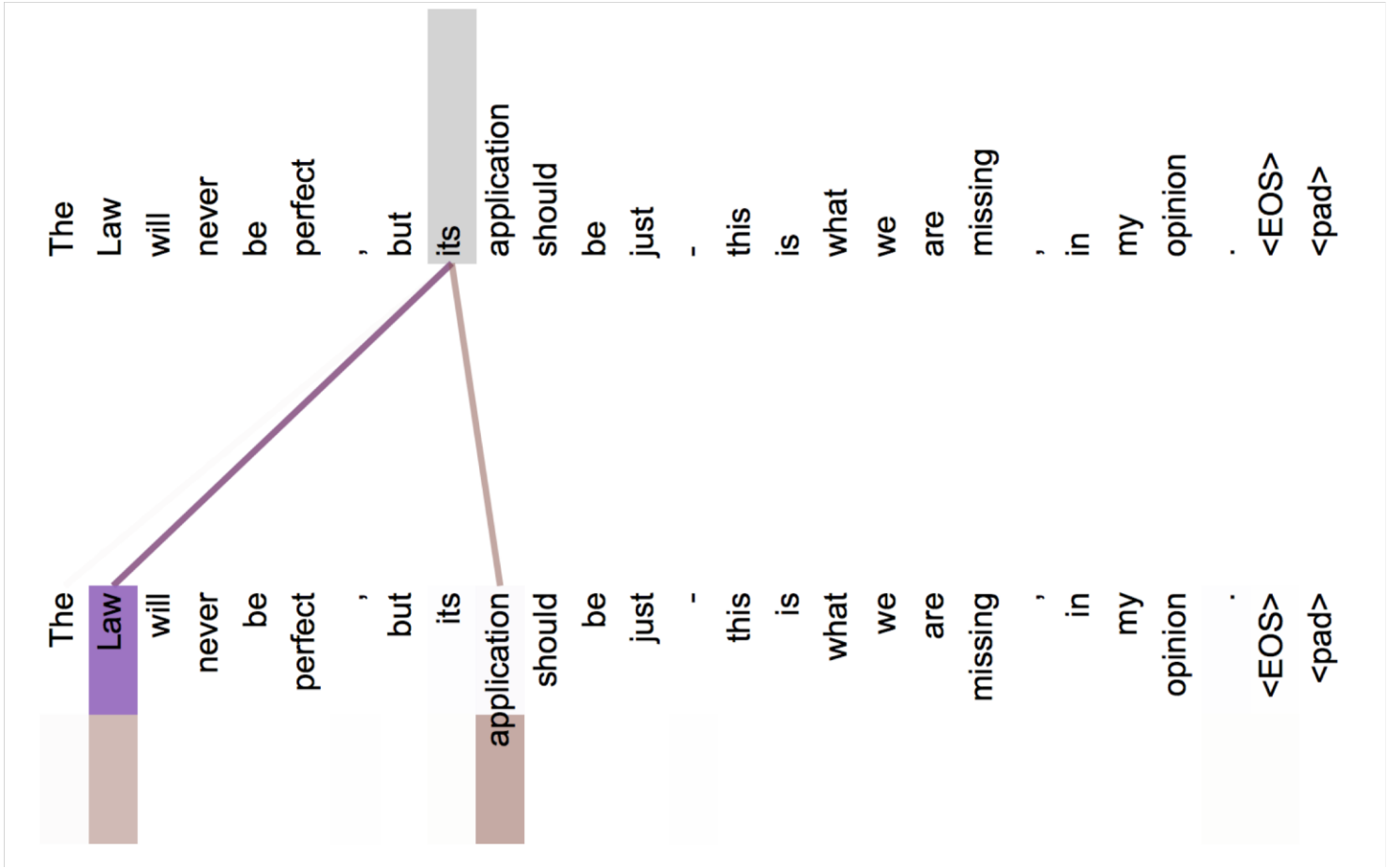
Credit to https://jalammar.github.io/illustrated-transformer/

Attention Visualization in Layer 5

- Words start to pay attention to other words in sensible ways

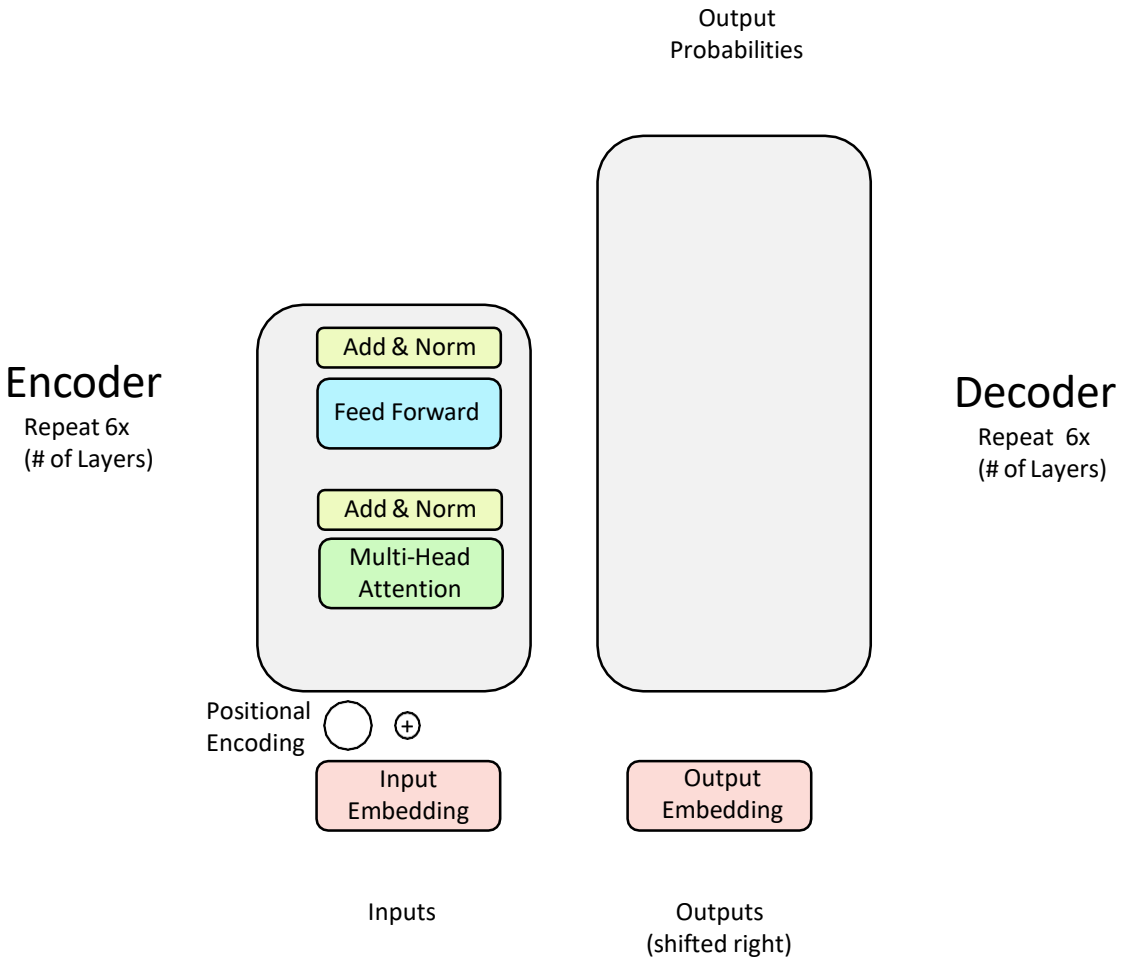


Attention Visualization: Implicit Anaphora Resolution



In the 5th layer, isolated attentions from just the word ‘its’ for attention heads 5 and 6. Note that the attentions are very sharp for this word.

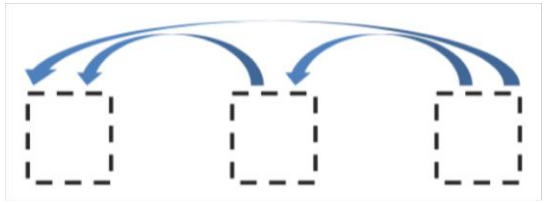
We Completed the Encoder! Time for the Decoder...



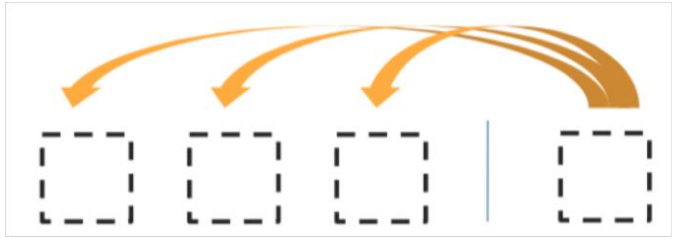
The Decoder

Two sublayer changes in the decoder:

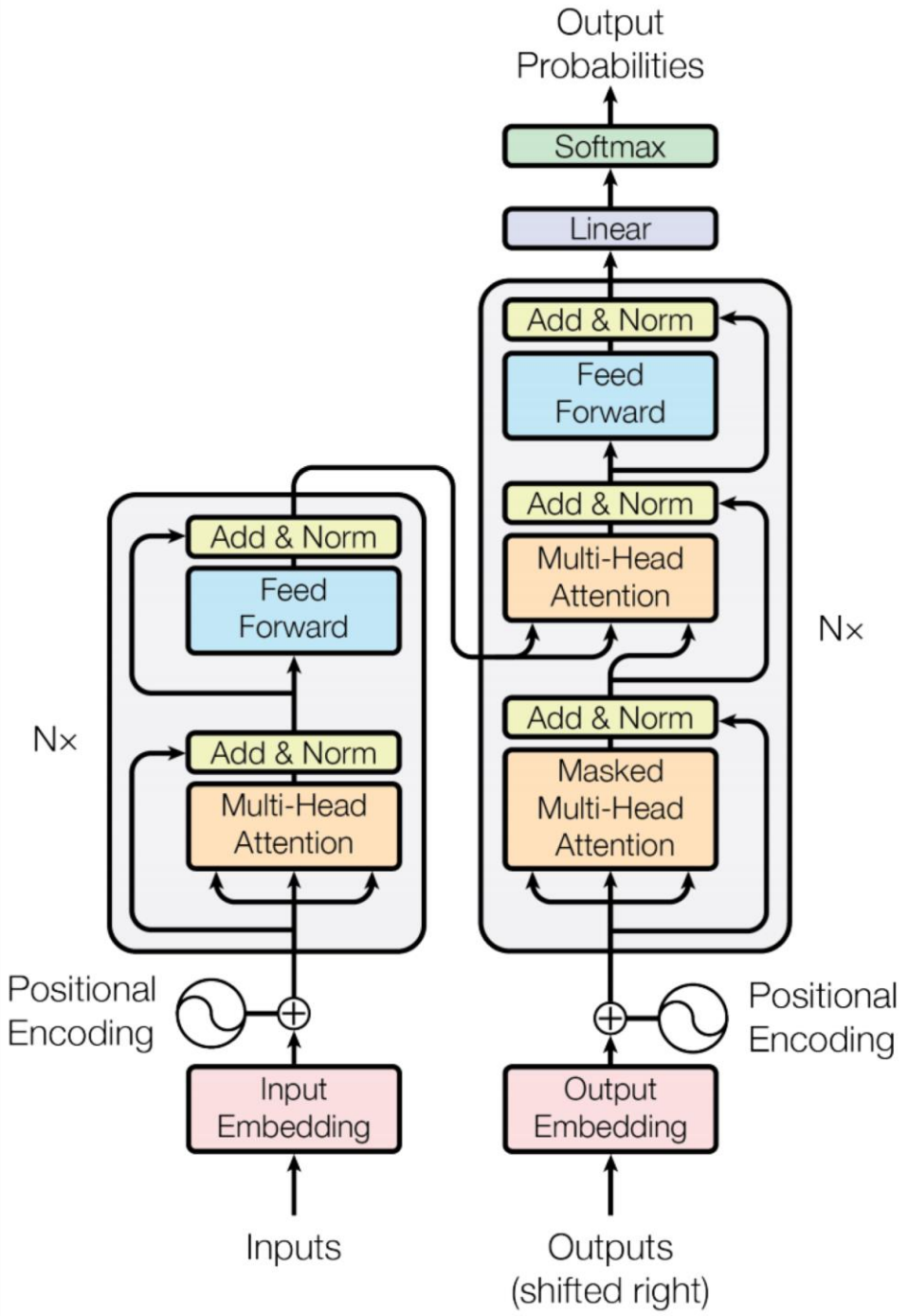
- **Masked decoder self-attention** on previously generated outputs:



- **Encoder-Decoder Attention**, where the queries come from the previous decoder layer and the keys and the values come from the output of the encoder

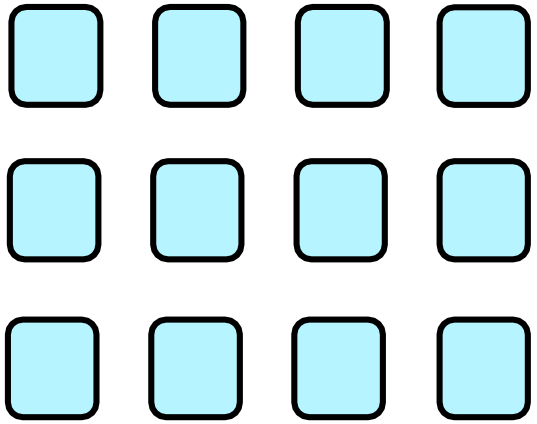


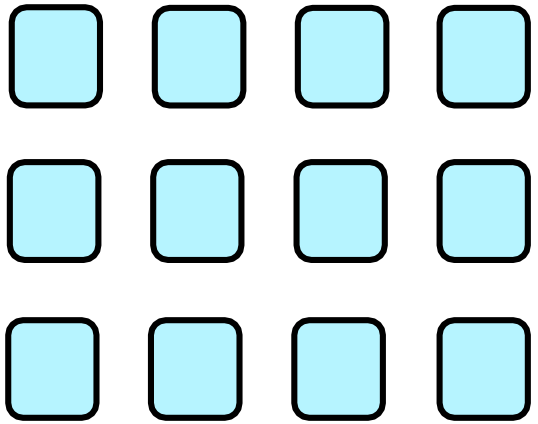
Blocks repeated 6 times also



Decoder: Masked Multi-Head Self-Attention

- **Problem:** How do we keep the decoder from "cheating"? If we have a language modeling objective, can't the network just look ahead and "see" the answer?

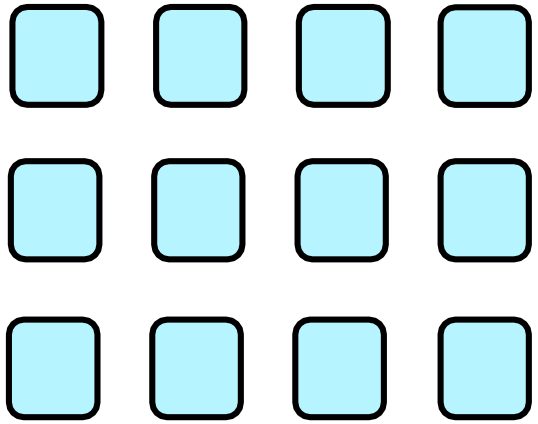


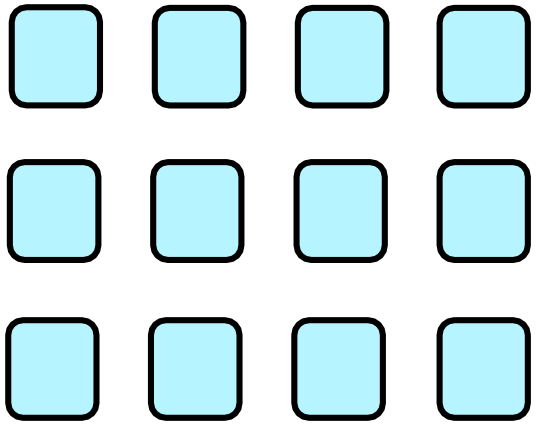


Transformer-Based
Encoder-Decoder Model

Decoder: Masked Multi-Head Self-Attention

- **Problem:** How do we keep the decoder from "cheating"? If we have a language modeling objective, can't the network just look ahead and "see" the answer?
- **Solution:** Masked Multi-Head Attention. At a high-level, we hide (mask) information about future tokens from the model.



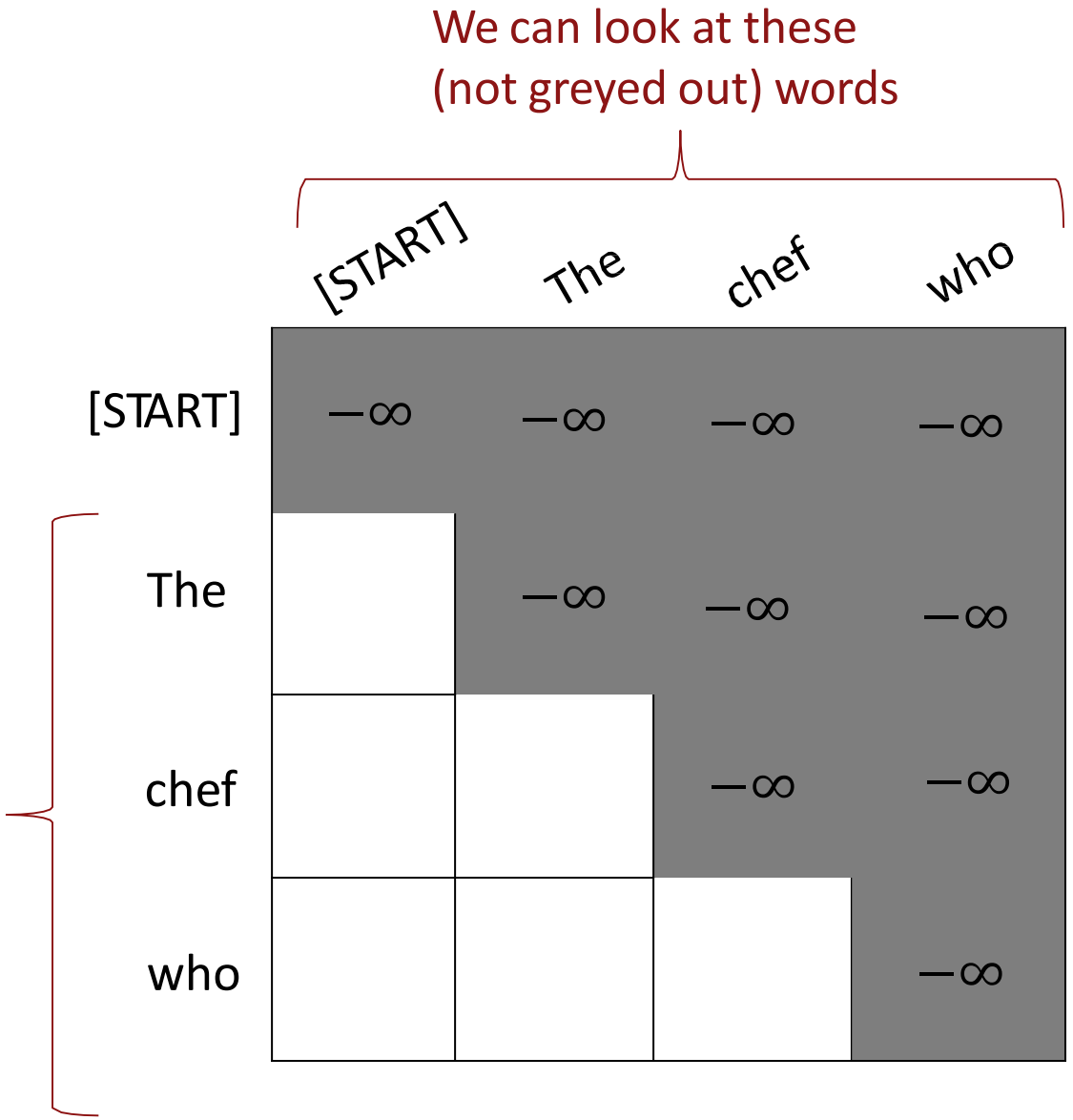


Transformer-Based Encoder-Decoder Model

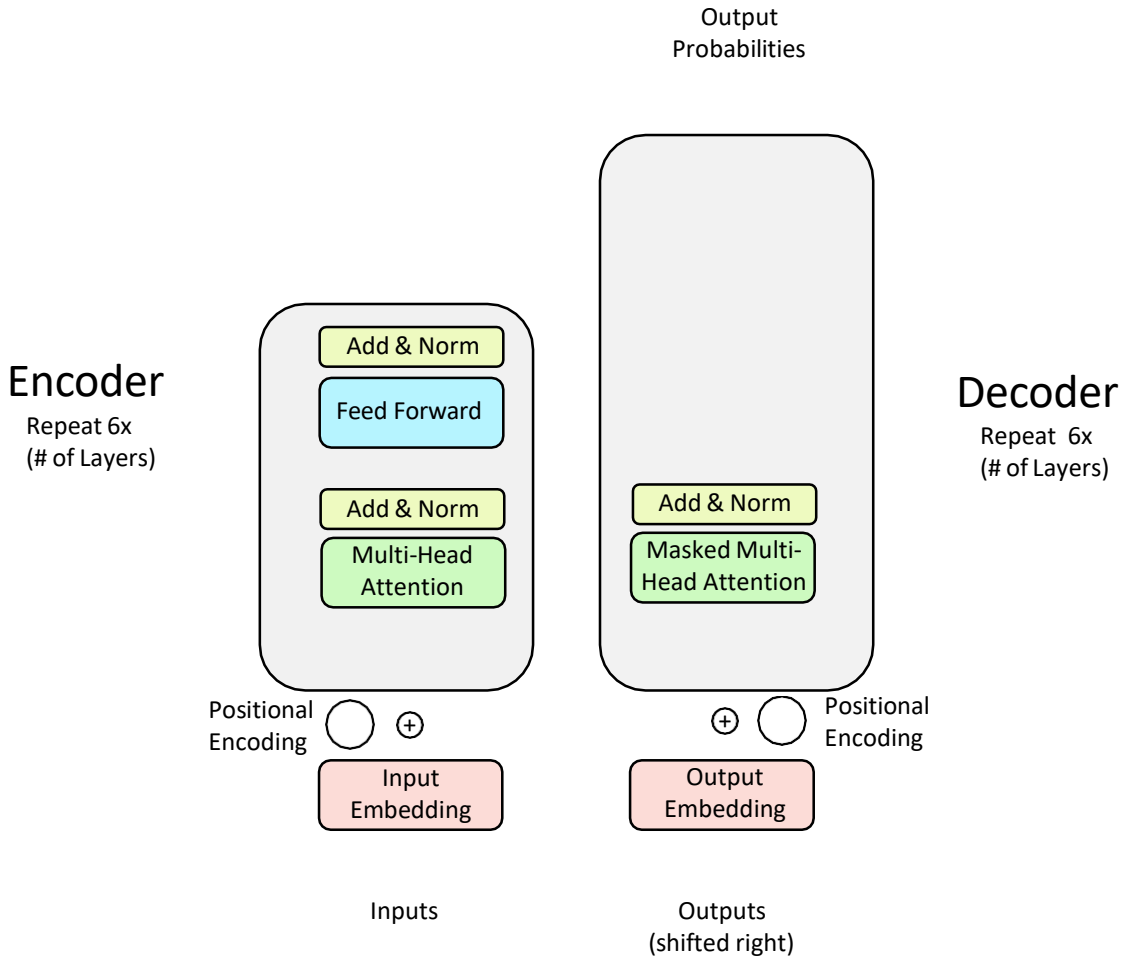
Masking the Future in Self-Attention

- To use self-attention in **decoders**, we need to ensure we cannot peek at the future.
- At every timestep, we could change the set of **keys and queries** to include only past words. (Inefficient!)
- To enable parallelization, we **mask out attention** to future words by setting attention scores to $-\infty$.

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



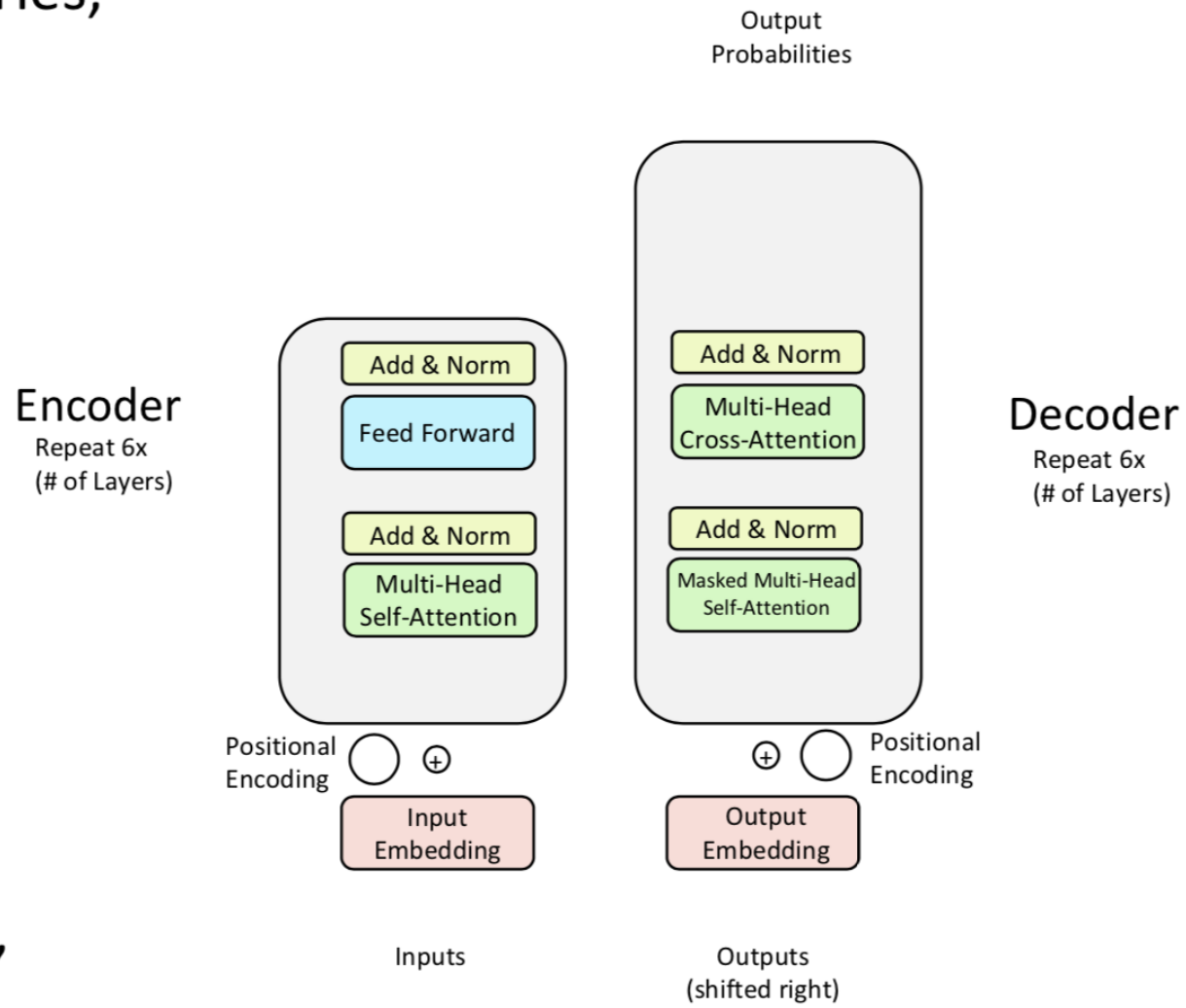
Decoder: Masked Multi-Headed Self-Attention



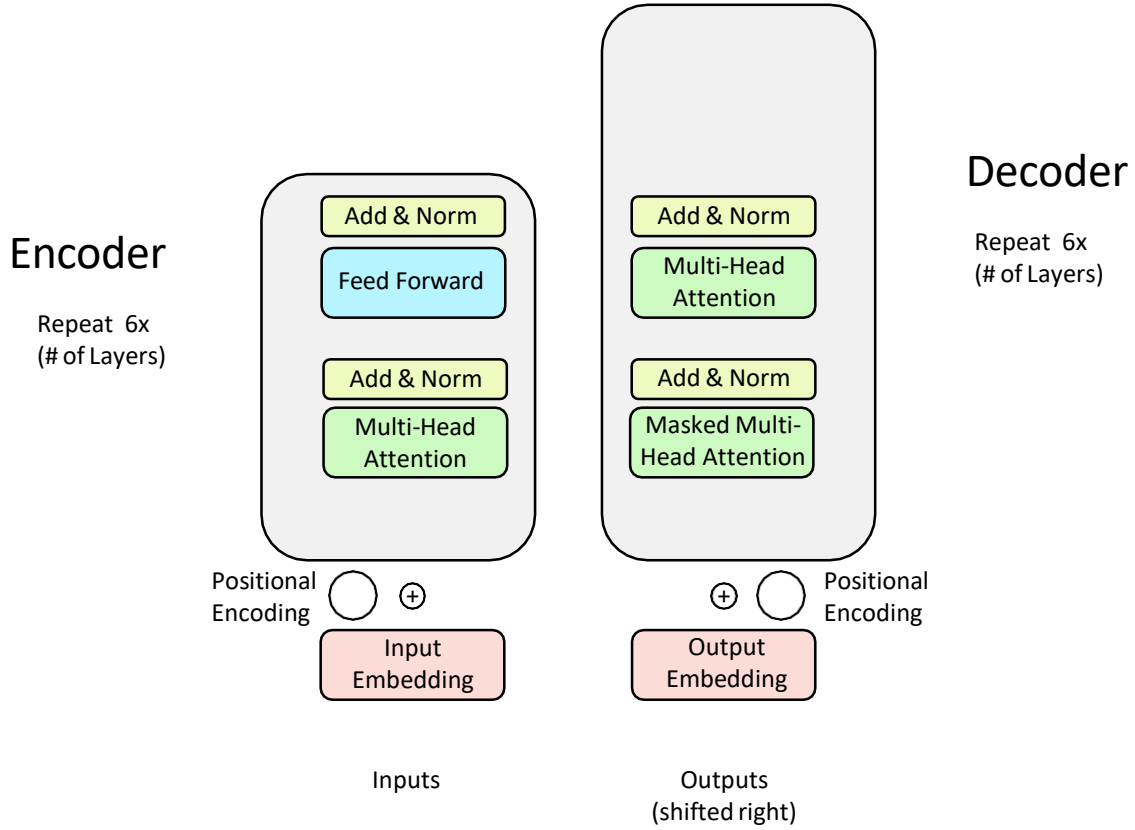
Encoder-Decoder Attention

- We saw that self-attention is when keys, queries, and values come from the same source.

- Let $h_1, \ldots, h_T$ be **output** vectors **from** the Transformer **encoder**; $x_i \in \mathbb{R}^d$
- Let $z_1, \ldots, z_T$ be input vectors from the Transformer **decoder**, $z_i \in \mathbb{R}^d$
- Then keys and values are drawn from the **encoder** (like a memory):
  - $k_i = Kh_i, v_i = Vh_i$.
- And the queries are drawn from the **decoder**, $q_i = Qz_i$.

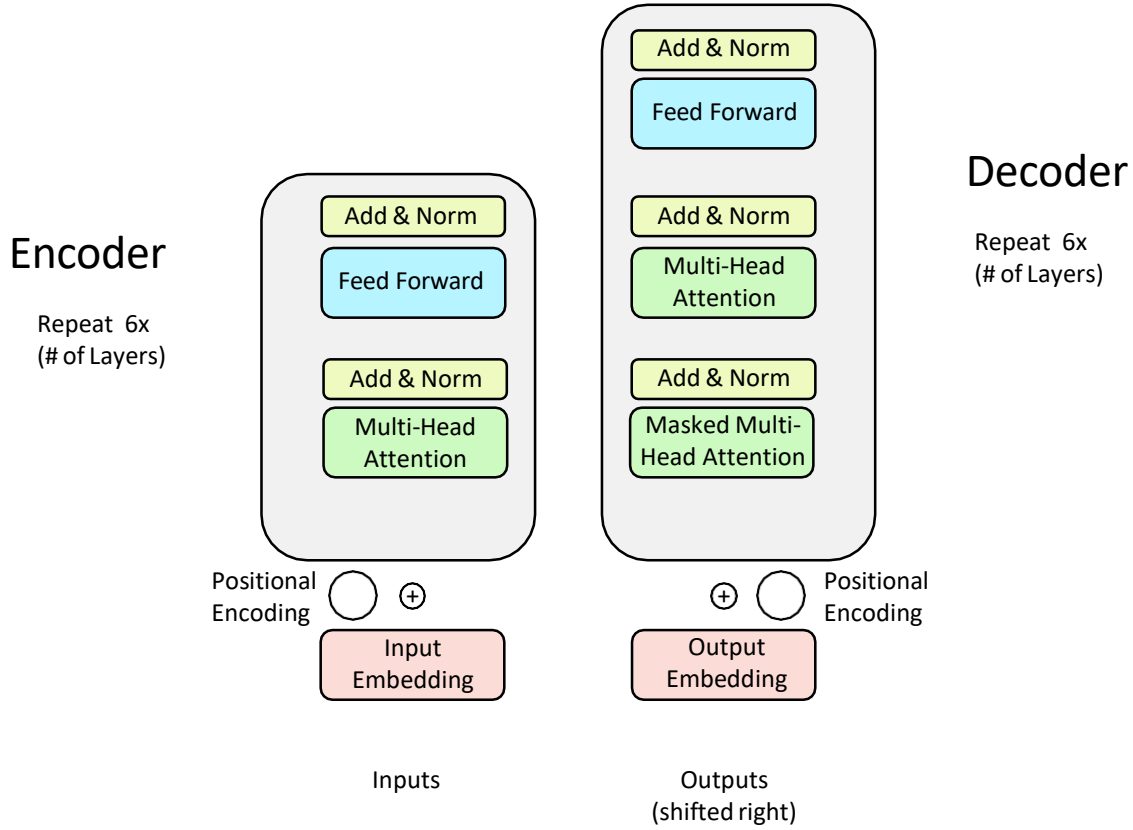


Decoder: Finishing Touches!



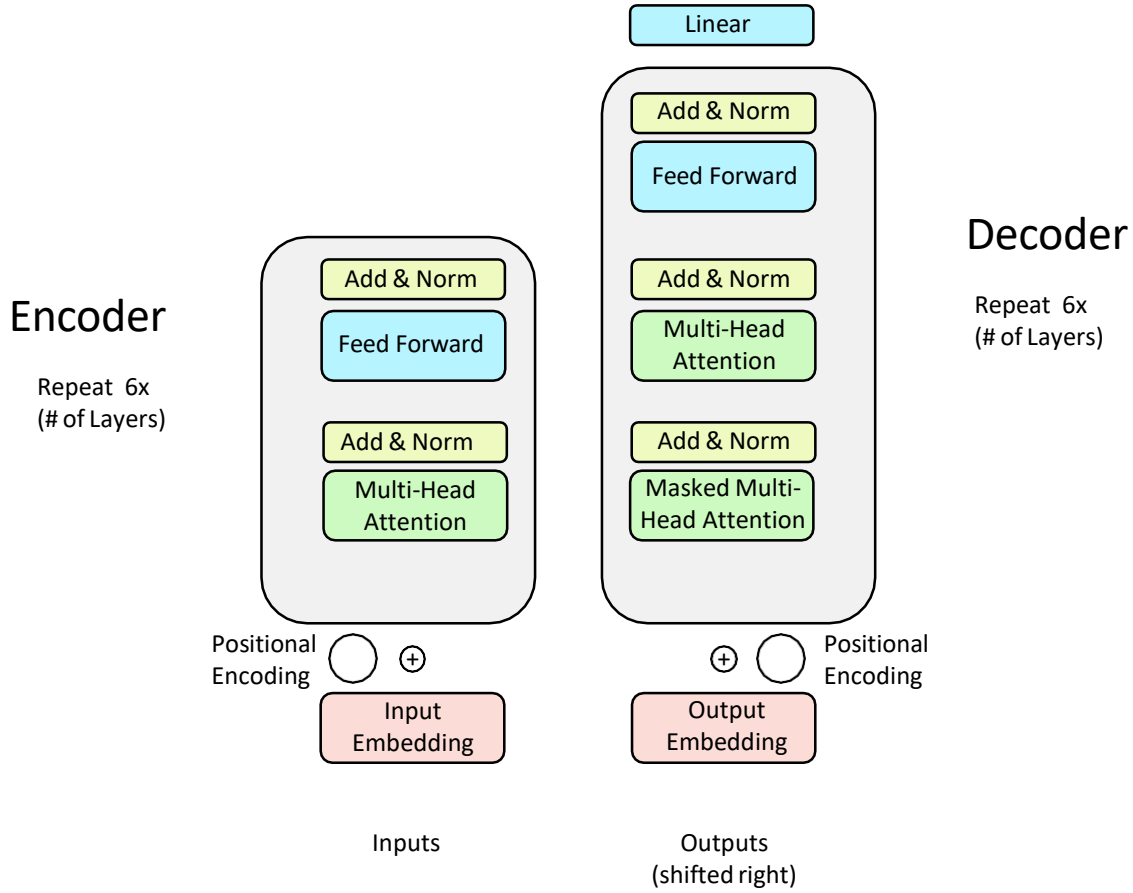
Decoder: Finishing Touches!

- Add a feed forward layer (with residual connections and layer norm)



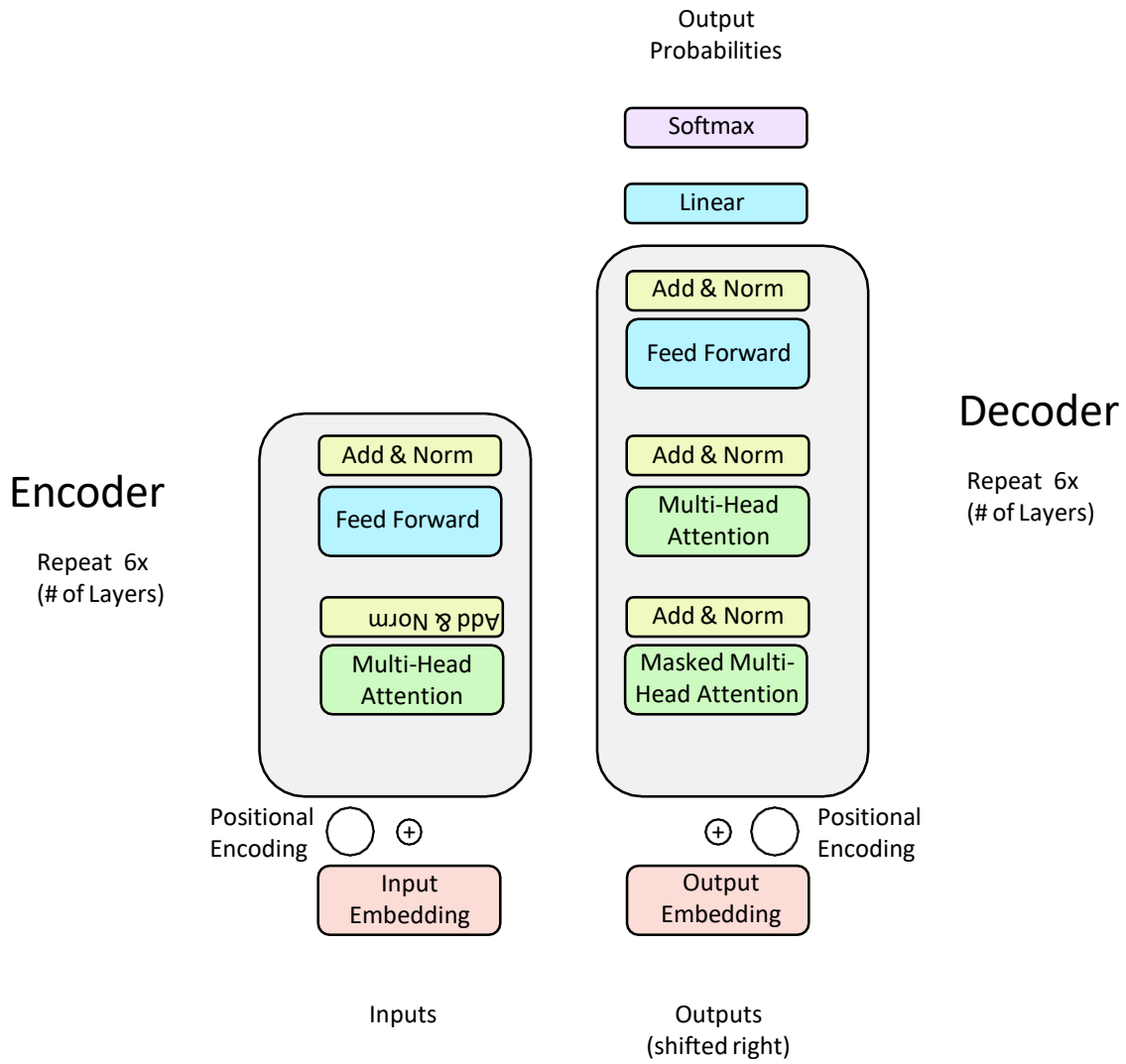
Decoder: Finishing Touches!

- Add a feed forward layer (with residual connections and layer norm)
- Add a final linear layer to project the embeddings into a much longer vector of length vocab size (logits)

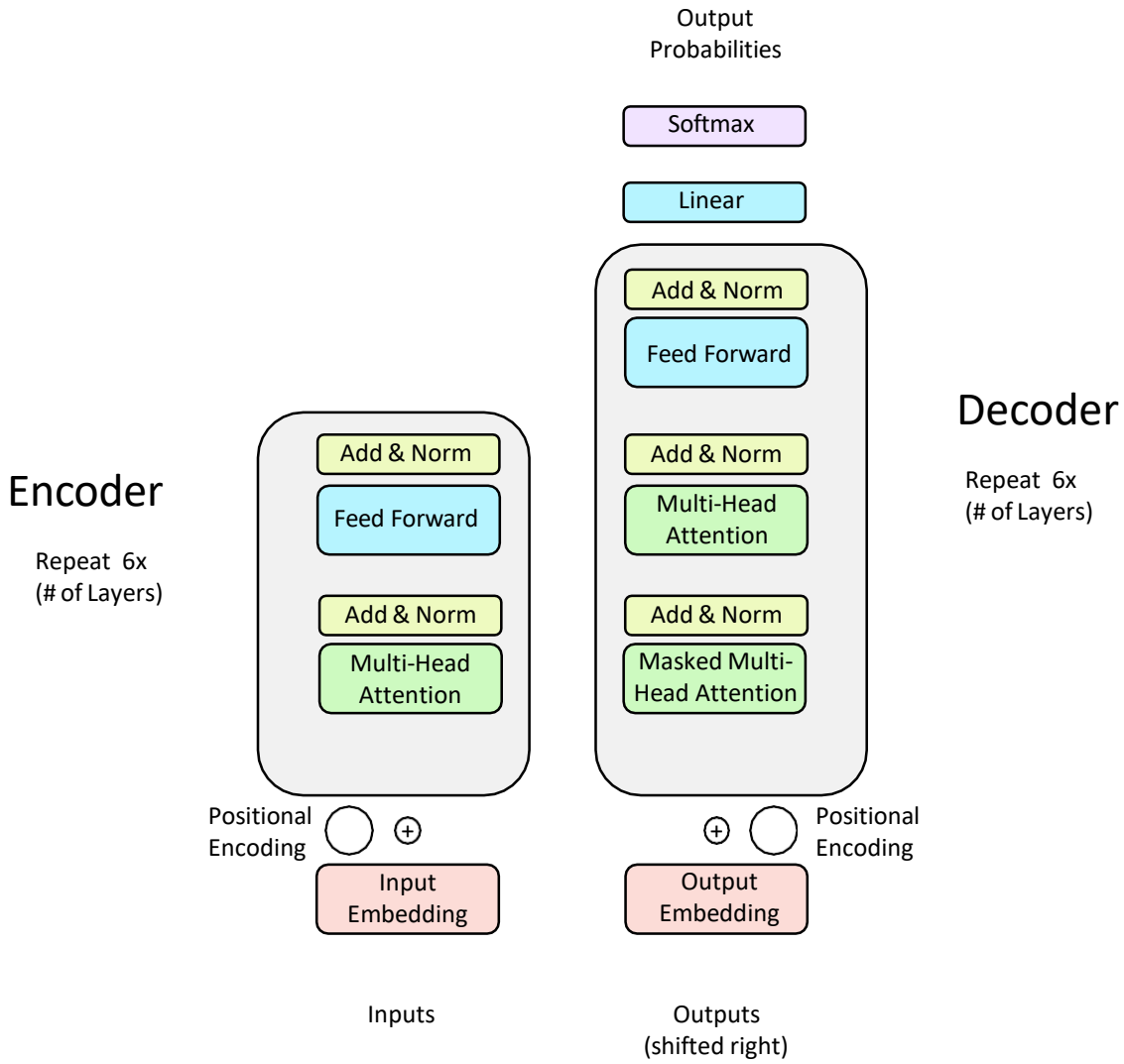


Decoder: Finishing Touches!

- Add a feed forward layer (with residual connections and layer norm)
- Add a final linear layer to project the embeddings into a much longer vector of length vocab size (logits)
- Add a final softmax to generate a probability distribution of possible next words!



Recap of the Transformer Architecture



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

The Illustrated Transformer

Discussions: Hacker News (65 points, 4 comments), Reddit r/MachineLearning (29 points, 3 comments)

Translations: Arabic, Chinese (Simplified) 1, Chinese (Simplified) 2, French 1, French 2, Italian, Japanese, Korean, Persian, Russian, Spanish 1, Spanish 2, Vietnamese

Watch: MIT's Deep Learning State of the Art lecture referencing this post

Featured in courses at Stanford, Harvard, MIT, Princeton, CMU and others

In the previous post, we looked at Attention – a ubiquitous method in modern deep learning models. Attention is a concept that helped improve the performance of neural machine translation applications. In this post, we will look at **The Transformer** – a model that uses attention to boost the speed with which these models can be trained. The Transformer outperforms the Google Neural Machine Translation model in specific tasks. The biggest benefit, however, comes from how The Transformer lends itself to parallelization. It is in fact Google Cloud's recommendation to use The Transformer as a reference model to use their Cloud TPU offering. So let's try to break the model apart and look at how it functions.

https://nlp.seas.harvard.edu/annotated-transformer/

The Annotated Transformer

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

- *v2022: Austin Huang, Suraj Subramanian, Jonathan Sum, Khalid Almubarak, and Stella Biderman.*
- *Original: Sasha Rush.*