Lecture 12: An Introduction to Transformers

STATS 202: Statistical Learning and Data Science

Linh Tran

Department of Statistics
Stanford University

August 4, 2025

Announcements

- Midterm exams are available.
- Homework 4 due this Wednesday.
- Final project predictions due this Sunday night.
- Final project writeup is due next Wednesday.
  - Optional (will take max of Final project & Final exam).
- Final exam is a week from this Saturday
  - August 16 @ 7:00 P.M. - 10:00 P.M. @300-300
  - Practice exam to be released Friday
  - Accommodation requests should be made now
- No formal lectures next week
- Review this Friday

Outline

- Word embeddings
- Sequence models
- Context-aware
- Self-attention
- Transformers
- Encoder / Decoder models

Motivation

In NLP, we have three big challenges:

1. Our categories are extremely large (e.g. 100K)
2. Sequences can get extremely long (e.g 128K)
3. Categories are 'precise' (one wrong word/token can completely change meaning)

Question: How do we deal with these?

Word embeddings

How do we reduce the dimensionality of our categorical random variable?

Option 1: Collapse it

- Take similar words and group into the same category
- Very manual (i.e. unscalable) and task specific

Option 2: Project it

- Learn a function that projects the words into a smaller dimensional space (d)
- In practice: $d << K$

Some projected word embeddings

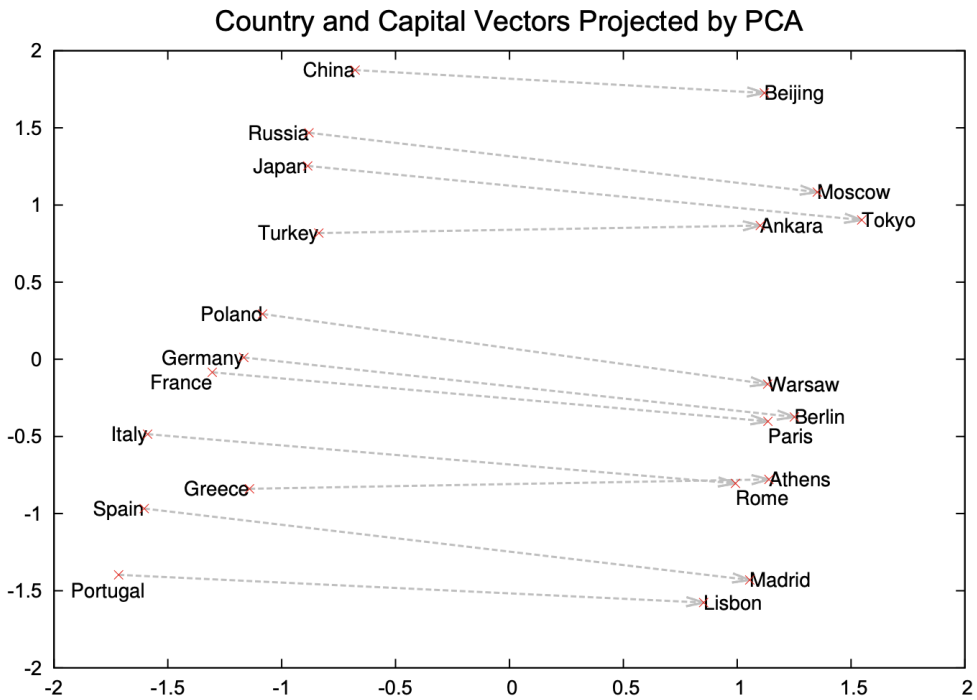


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

Word embeddings

How to 'learn' a projection.

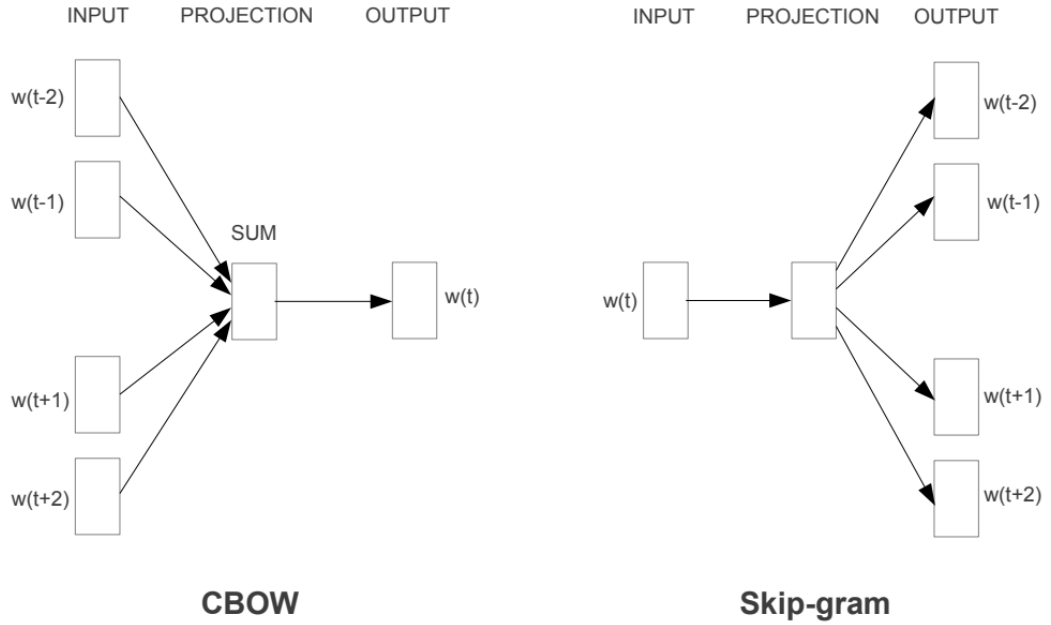
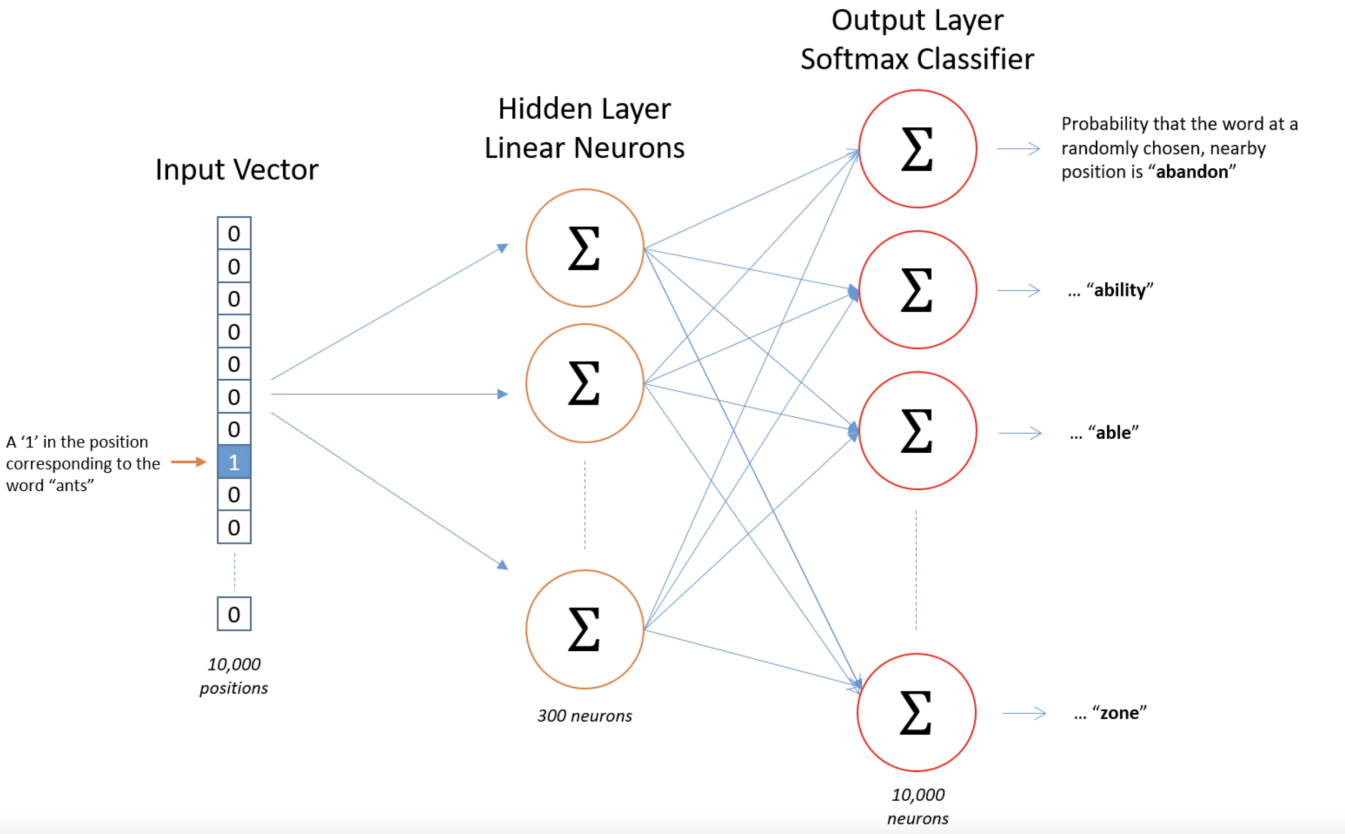


Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781. https://arxiv.org/abs/1301.3781

Learning word embeddings

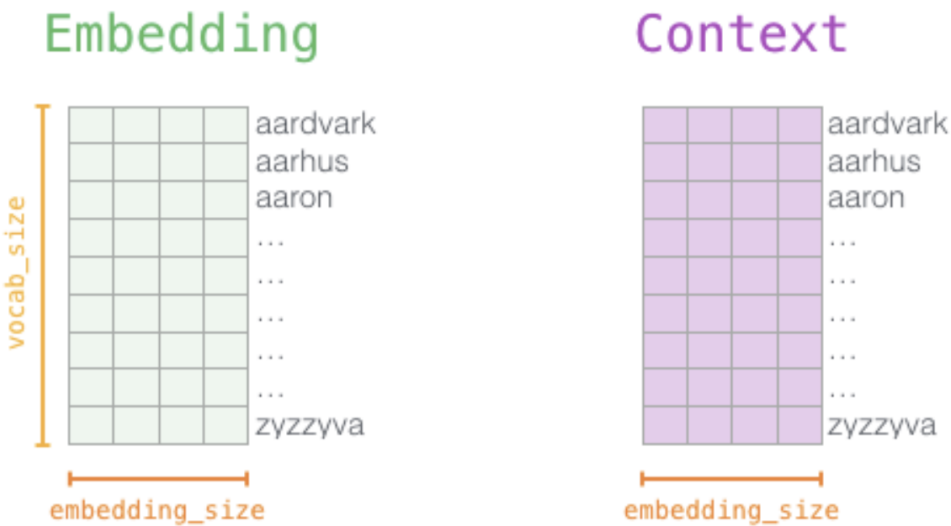
One approach:



Learning word embeddings

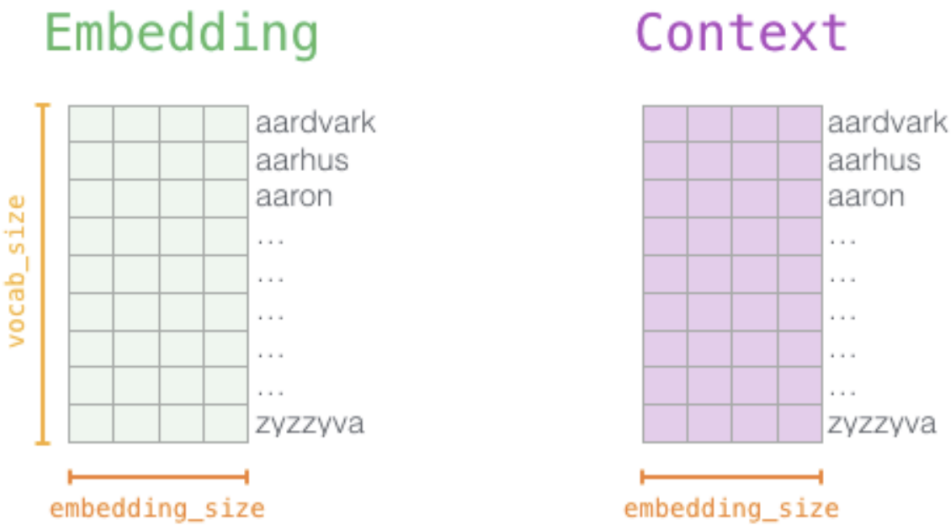
Another approach:

$$p(w_O|w_I) = \frac{\exp\left(\mathbf{v}'_{w_O}\mathbf{v}_{w_I}\right)}{\sum_{w=1}^{W}\exp\left(\mathbf{v}'_{w}\mathbf{v}_{w_I}\right)} \quad (1)$$



Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. Advances in Neural Information Processing Systems, 26, 3111-3119.

Learning word embeddings

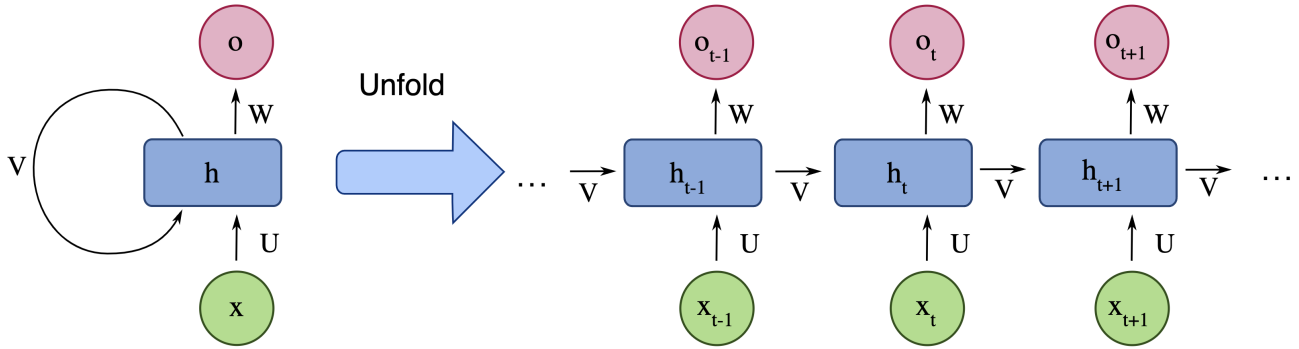


Notes:

- Uses hierarchical softmax.
- Performs negative sampling.
- Subsamples frequent words with $p(w_i) = \sqrt{\frac{t}{f(w_i)}} : t = 10^{-5}$.

Sequence models

How do we account for the sequential nature of text?



Recurrent neural network

Two options:
Elman network

$$\begin{aligned} h_t &= \sigma_h(\mathbf{W}_h \mathbf{x}_t + \mathbf{U}_h \mathbf{h}_{t-1} + \mathbf{b}_h) \\ y_t &= \sigma_y(\mathbf{W}_y \mathbf{h}_t + \mathbf{b}_y) \end{aligned}$$

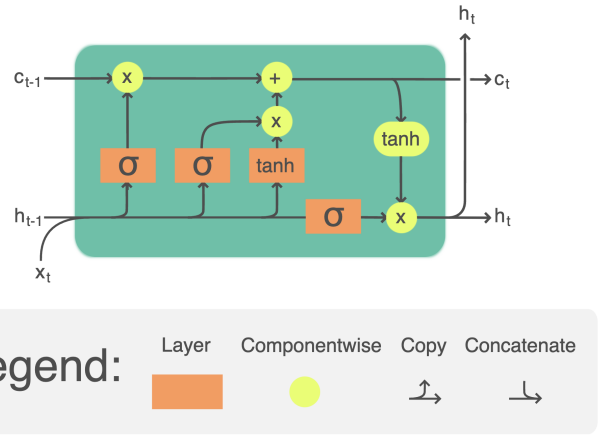
Jordan network

$$\begin{aligned} h_t &= \sigma_h(\mathbf{W}_h \mathbf{x}_t + \mathbf{U}_h \mathbf{y}_{t-1} + \mathbf{b}_h) \\ y_t &= \sigma_y(\mathbf{W}_y \mathbf{h}_t + \mathbf{b}_y) \end{aligned}$$

n.b. These RNNs can be stacked

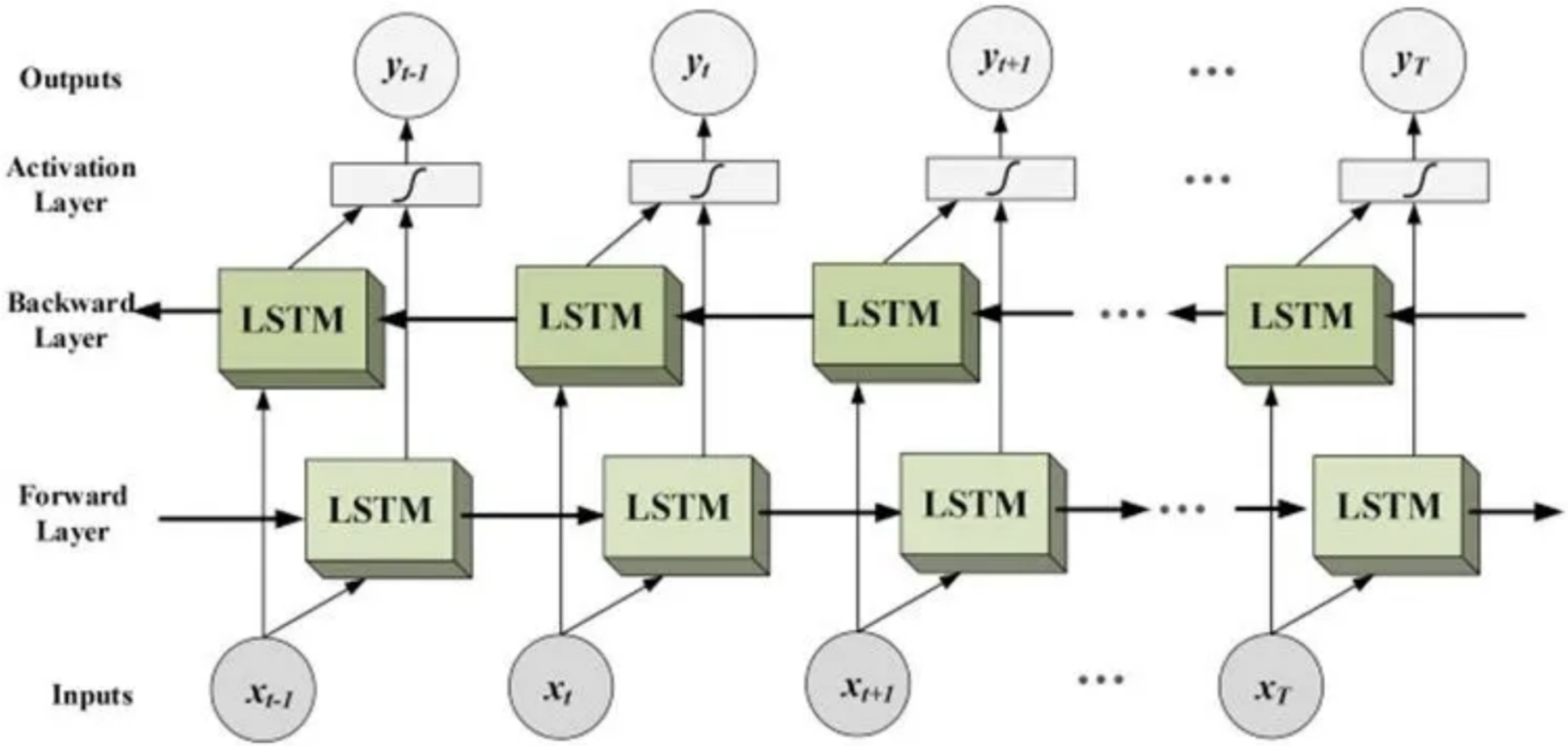
Sequence models

LSTMs address 'forgetting'



$$
\begin{aligned}
\mathbf{f}_t &= \sigma_g(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f) \\
\mathbf{i}_t &= \sigma_g(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{b}_i) \\
\mathbf{o}_t &= \sigma_g(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{b}_o) \\
\tilde{\mathbf{c}}_t &= \sigma_c(\mathbf{W}_c \mathbf{x}_t + \mathbf{U}_c \mathbf{h}_{t-1} + \mathbf{b}_c) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \\
\mathbf{h}_t &= \mathbf{o}_t \odot \sigma_h(\mathbf{c}_t)
\end{aligned}
$$

Bidirectionality



Good to use when we're not doing causal modeling (eg translation).

Context aware embeddings

Problem: words can have multiple meanings (e.g. bank)

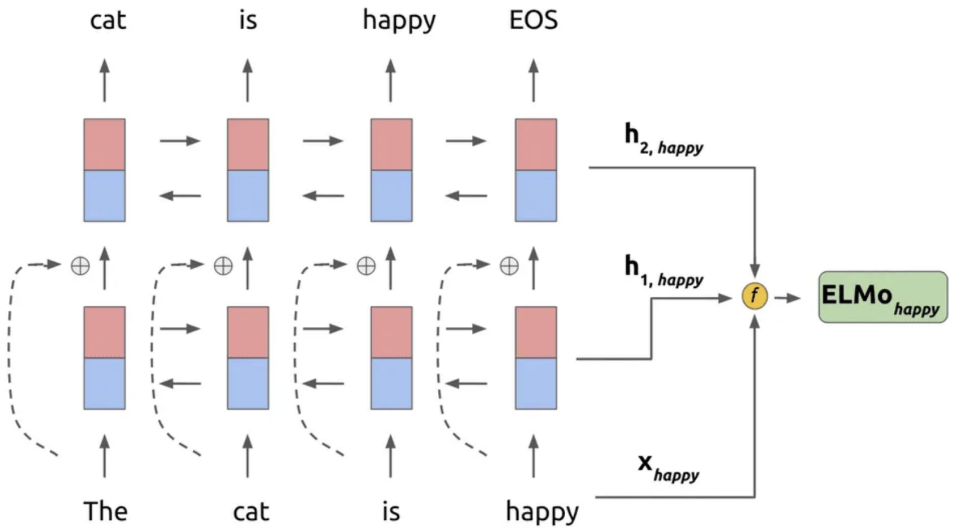
- How do we make the embeddings context aware?

Context aware embeddings

Problem: words can have multiple meanings (e.g. bank)

- How do we make the embeddings context aware?

Calculate Embeddings from Language Models (ELMo)



An example of combining the bidirectional hidden representations and word representation for "happy" to get an Elmo-specific representation.

Context aware embeddings

Step 1: train to predict 'next' word

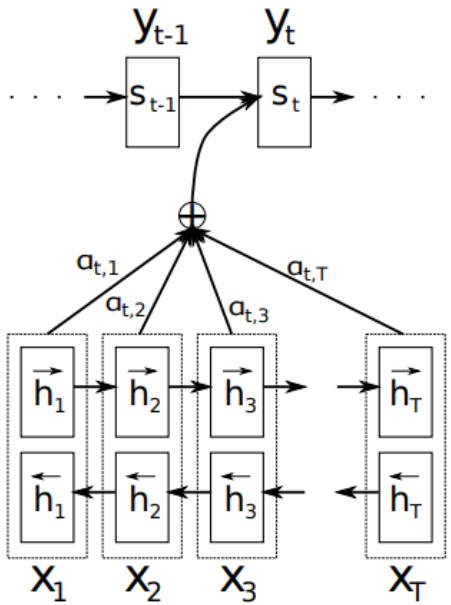
$$\begin{aligned}\mathcal{L}(\Theta) &= \sum_{k=1}^{N} \log p(t_k | t_1, \ldots, t_{k-1}; \Theta_x, \overrightarrow{\Theta}_{LSTM,\Theta_s}) \\ &\quad + \log p(t_k | t_{k+1}, \ldots, t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM,\Theta_s})\end{aligned}$$

Step 2: apply to a downstream task

$$\mathbf{h}_k^{task} = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}$$

Attention

Problem: LSTMs still don't carry history well
Solution: Build in weights for the hidden states



Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015).

Attention

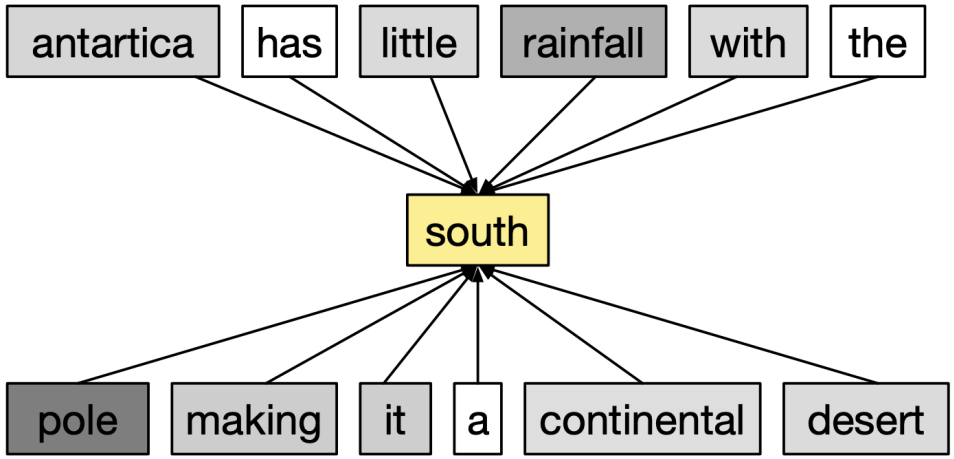
Mathematically (in context of machine translation),

$$
\begin{aligned}
p(y_i|y_1,\ldots,y_{i-1},\mathbf{x}) &= g(y_{i-1},\mathbf{s}_i,\mathbf{c}_i), \text{ where} \\
\mathbf{c}_i &= \sum_{j=1}^{T} \alpha_{ij}\mathbf{h}_j \\
\alpha_{ij} &= \frac{\exp(e_{ij})}{\sum_{k=1}^{T}\exp(e_{ik})} \\
e_{ij} &= a(\mathbf{s}_{i-1},\mathbf{h}_j)
\end{aligned}
$$

Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015).

Attention

Question: Do we even need RNNs/LSTMs?



Wang Ling, Lin Chu-Cheng, Yulia Tsvetkov, and Silvio Amir. (2015). Not all contexts are created equal: Better word representations with variable attention. EMNLP 2025.

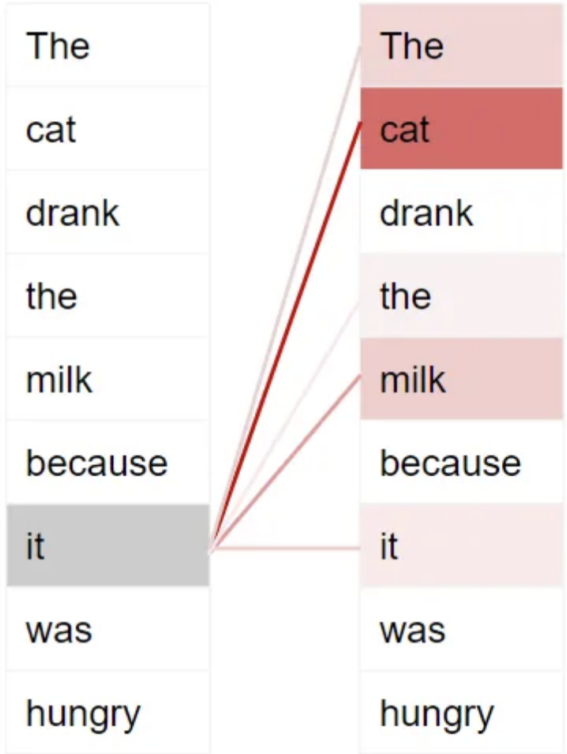
Attention

$$
\begin{aligned}
p(v_O|w_{[-b,b]-\{0\}}) &= \frac{\exp\left(\mathbf{v}_O^\top \mathbf{O}\mathbf{c}\right)}{\sum_{v\in V}\exp\left(\mathbf{v}_O^\top \mathbf{O}\mathbf{c}\right)} \\
\mathbf{c} &= \sum_{[-b,b]-\{0\}} a_i(\mathbf{w_i})\mathbf{w_i} \\
a_i(\mathbf{w_i}) &= \frac{\exp(k_{w,i}) + \mathbf{s}_i}{\sum_{[-b,b]-\{0\}}\exp(k_{w,j}) + \mathbf{s}_i} : k_{i,j} \in \mathbf{K} \in \mathcal{R}^{|V|\times 2b}
\end{aligned}
$$

Wang Ling, Lin Chu-Cheng, Yulia Tsvetkov, and Silvio Amir. (2015). Not all contexts are created equal: Better word representations with variable attention. EMNLP 2025.

Self attention

This can also be applied to other sequences, i.e.



Self attention

This can also be applied multiple times, i.e.

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

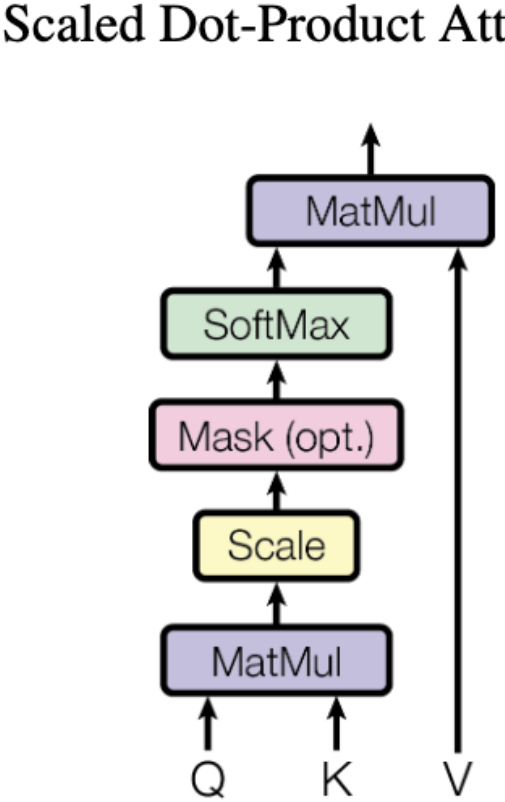
Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

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

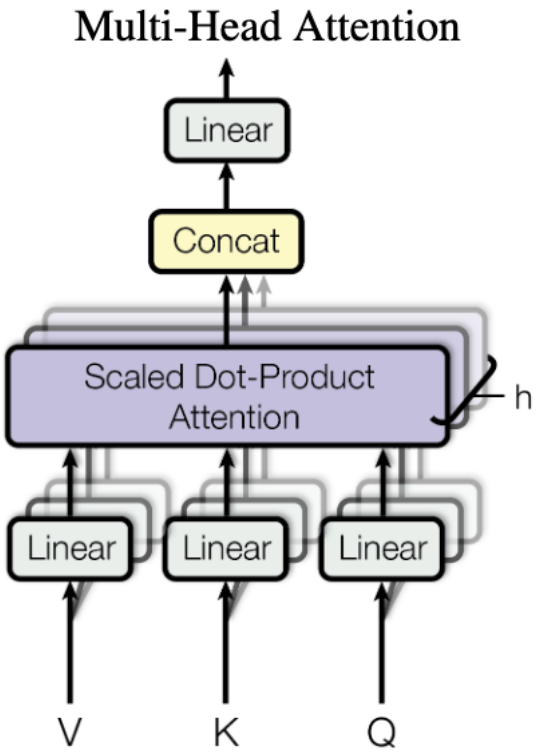
Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com



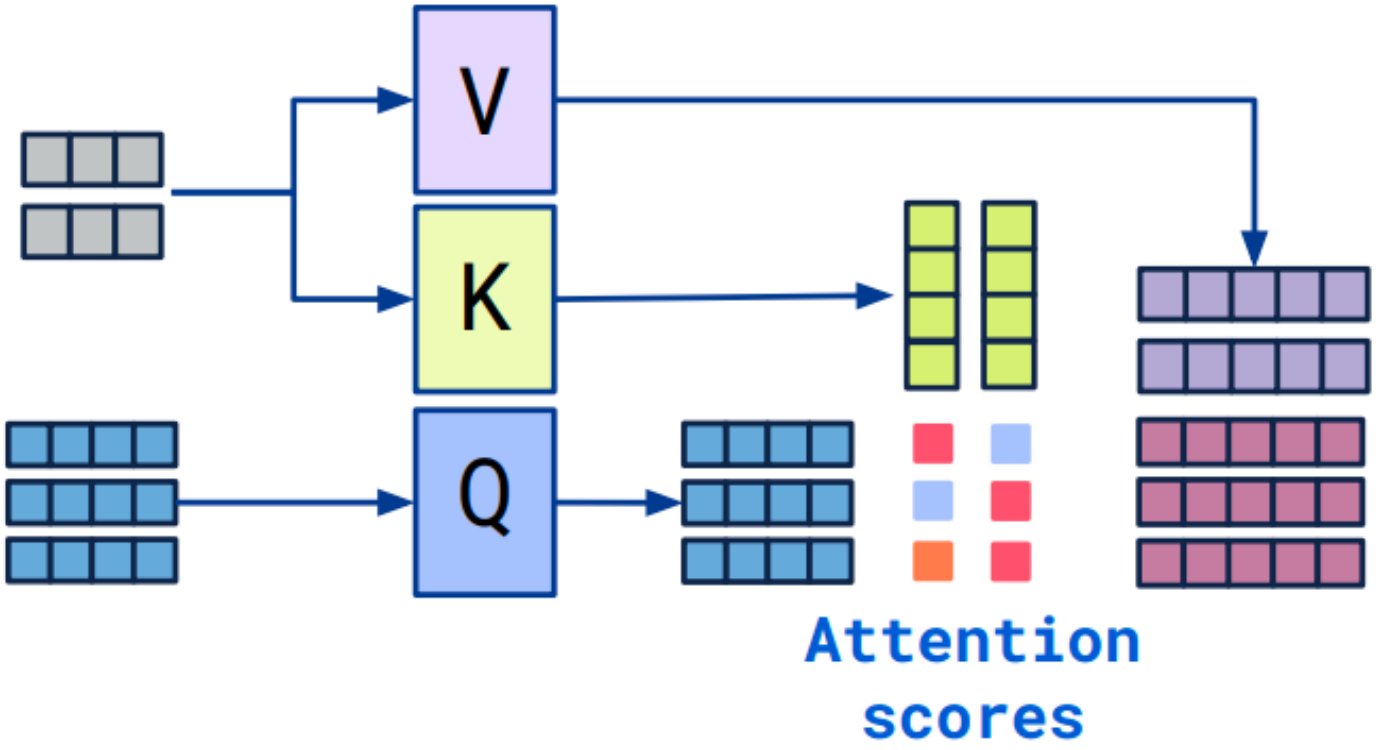
$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$

Transformers



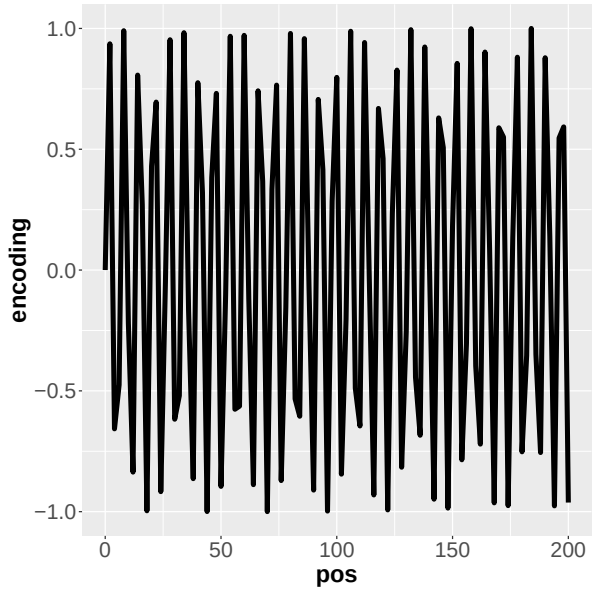
$$\begin{aligned} \text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) &= \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\mathbf{W}^O \\ \text{where head}_i &= \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V) \end{aligned}$$

Cross attention

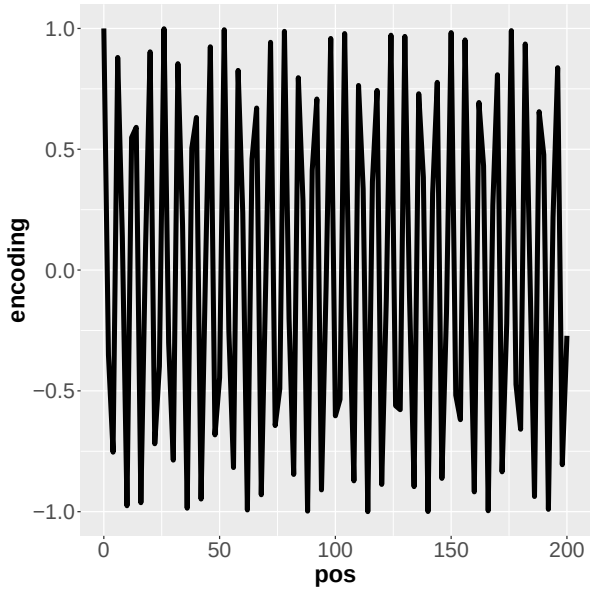


Positional encodings

Account for the positions of each token in the sequence



$\text{PE}_{(pos,2i)}$ where $i = 1$

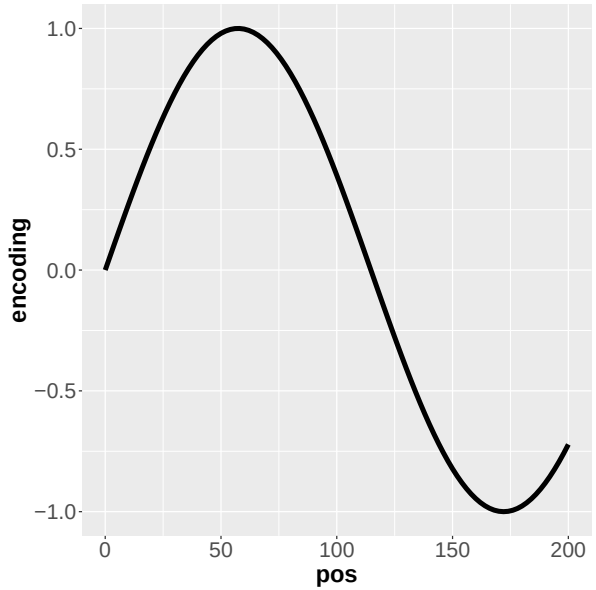


$\text{PE}_{(pos,2i+1)}$ where $i = 1$

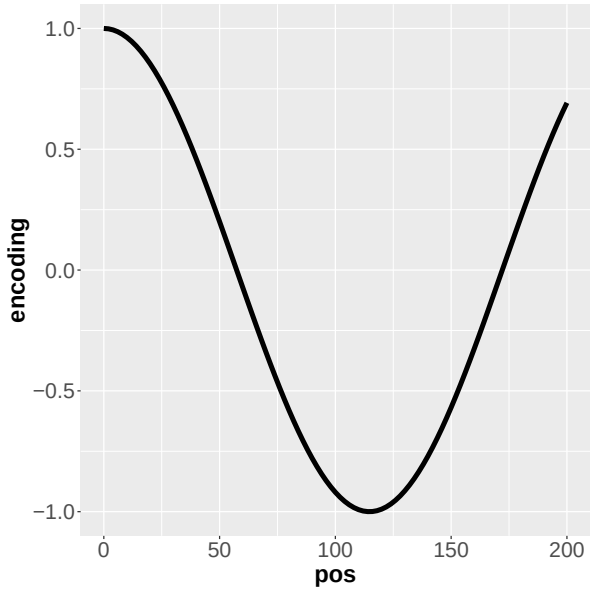
$$\text{PE}_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$
$$\text{PE}_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

Positional encodings

The functions vary over the dimension of the embeddings.



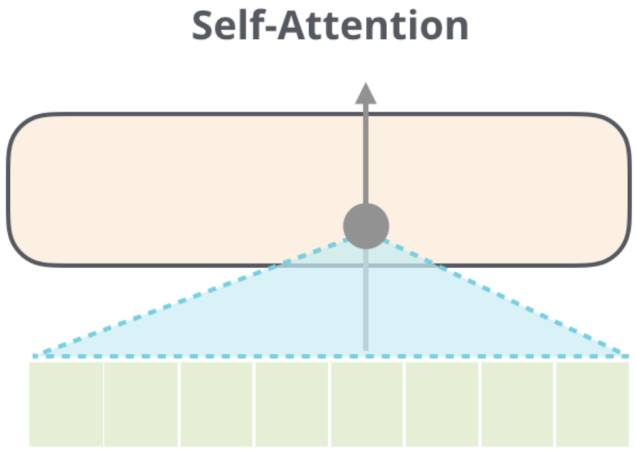
$\text{PE}_{(pos,2i)}$ where $i = 100$

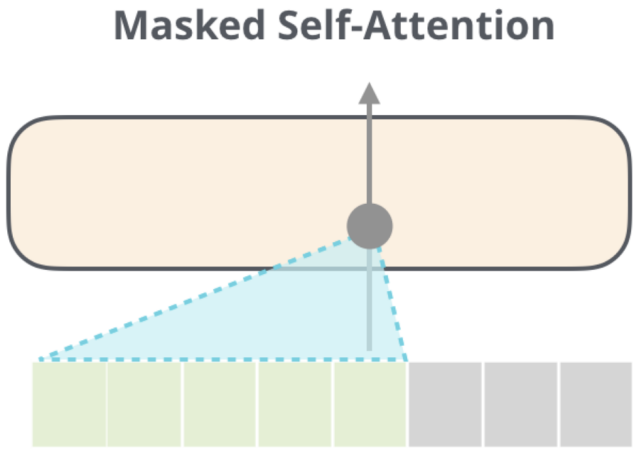


$\text{PE}_{(pos,2i+1)}$ where $i = 250$

$$\begin{aligned} \text{PE}_{(pos,2i)} &= \sin(pos/10000^{2i/d_{model}}) \\ \text{PE}_{(pos,2i+1)} &= \cos(pos/10000^{2i/d_{model}}) \end{aligned}$$

Masking





$$\text{MaskedAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} + m\right)\mathbf{V} : m \in \{0, -\infty\}$$

$$\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} + \mathbf{M} = \begin{bmatrix} a_{11} & -\infty & -\infty & -\infty \\ a_{21} & a_{22} & -\infty & -\infty \\ a_{31} & a_{32} & a_{33} & -\infty \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

Stabilizing training

Problem: When working with deep networks, small changes in one layer can significantly affect optimization in another.
Layer normalization:

1. Normalize each layer to have mean 0 and variance 1
2. For hidden node h_i, compute $h_i \leftarrow \frac{h_i - \mathbb{E}_n[x]}{\sqrt{\hat{\text{Var}}(x) + \epsilon}} * \gamma + \beta$
3. γ and β are learnable parameters
4. Reduces "covariate shift" (i.e. gradient dependencies) between layers

The Transformer - model architecture

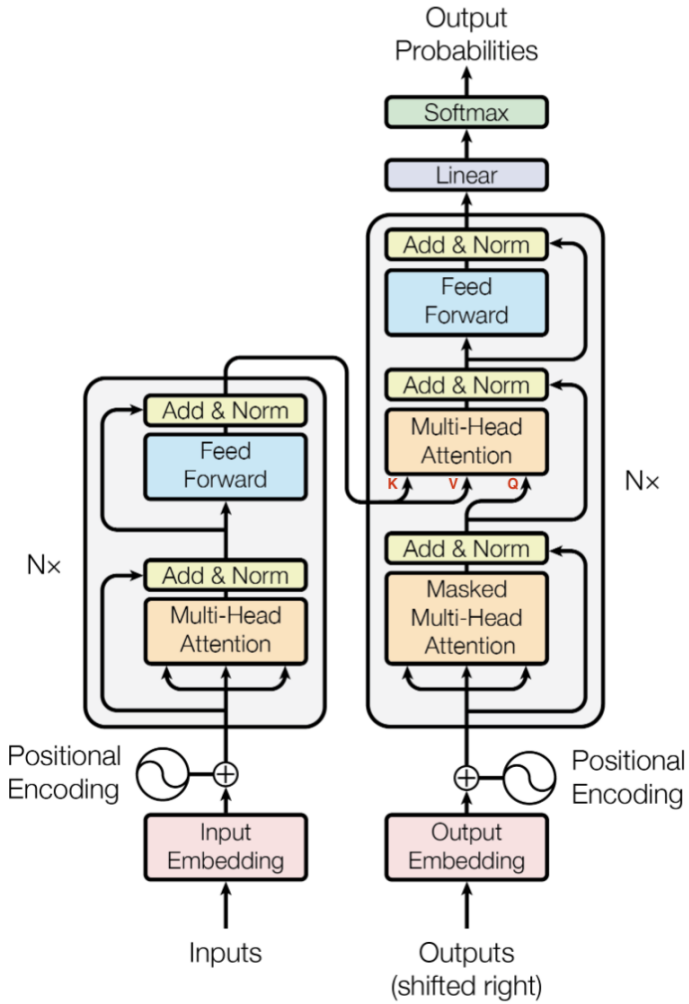
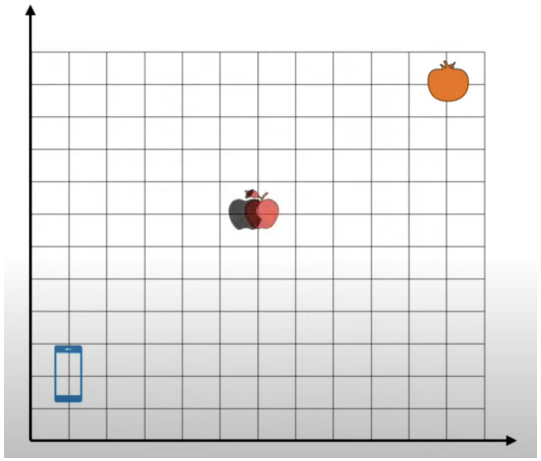
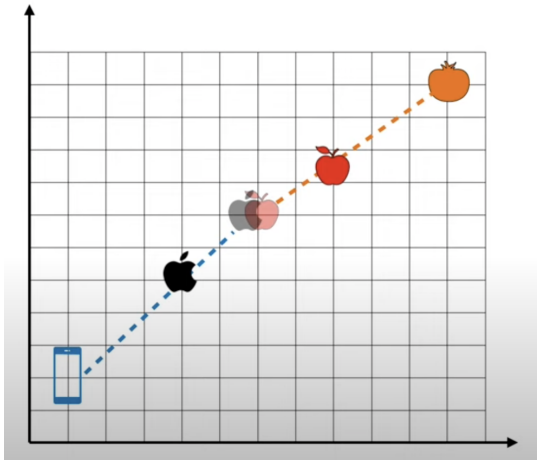


Figure 1: The Transformer - model architecture.

n.b. Visualization available *here*.

Contextualized embeddings





Results

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

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	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**	**41.0**	$2.3 \cdot 10^{19}$	

- Performance: slightly better than other state of the art
- Cost: Much cheaper to train

Encoder models - BERT

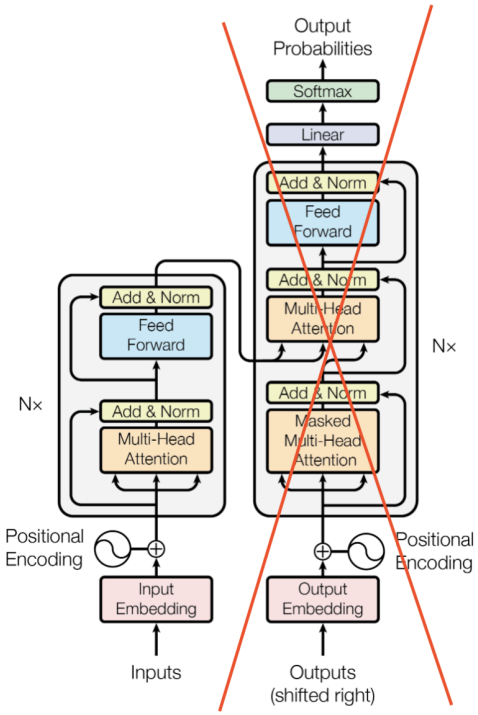
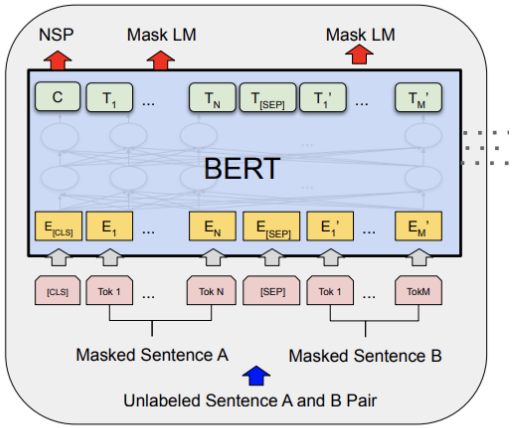
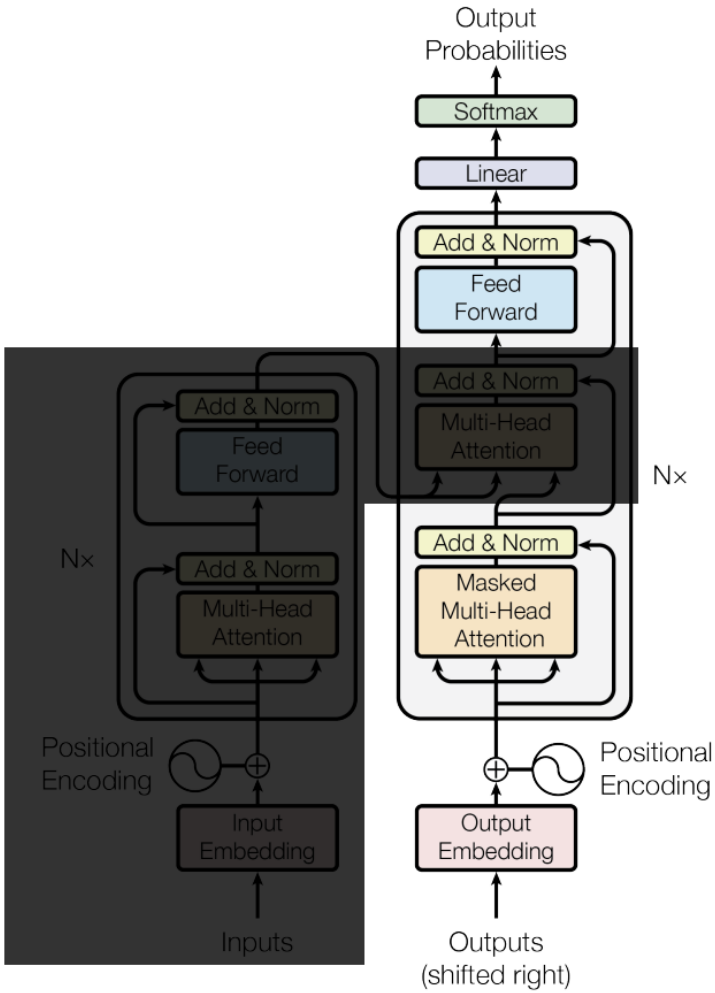


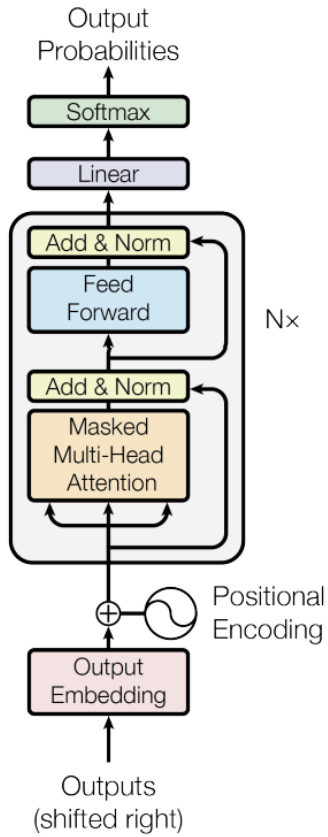
Figure 1: The Transformer - model architecture.



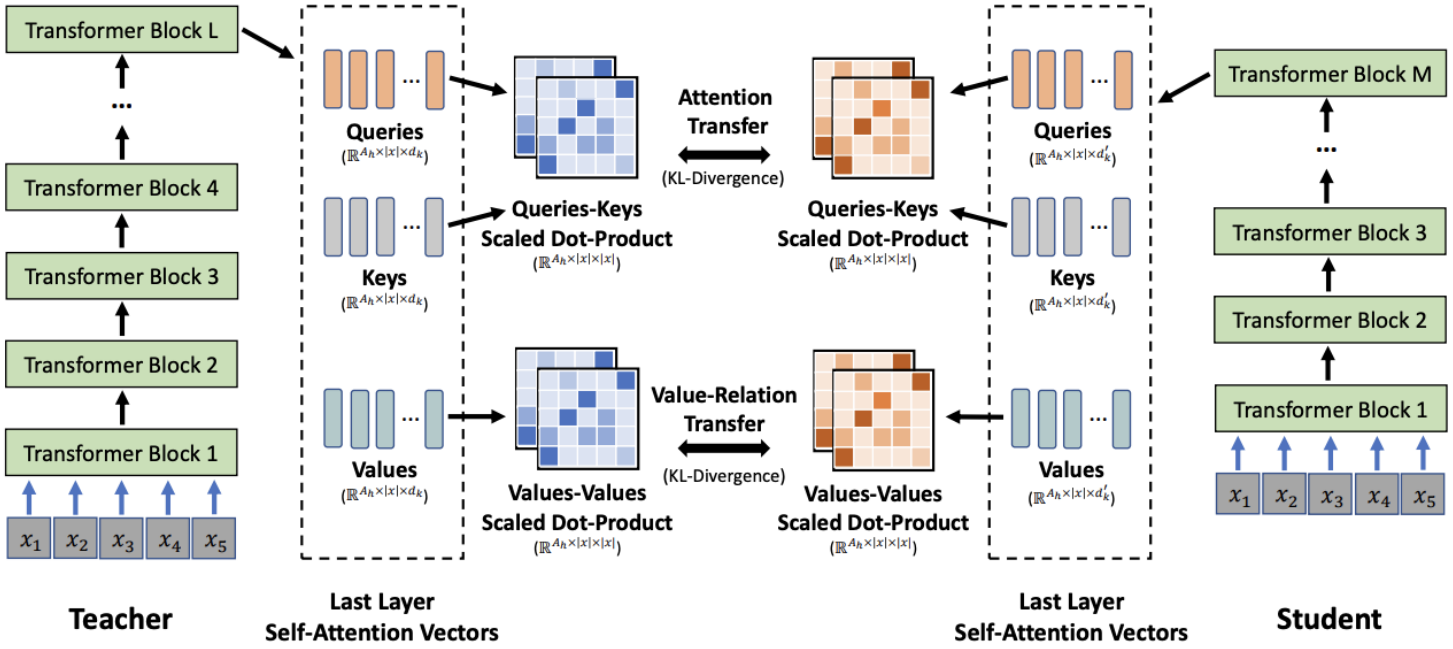
Pre-training

Decoder models - GPT





Examples - MiniLM



Examples - MiniLM

```
BertModel(
  (embeddings): BertEmbeddings(
    (word_embeddings): Embedding(30522, 384, padding_idx=0)
    (position_embeddings): Embedding(512, 384)
    (token_type_embeddings): Embedding(2, 384)
    (LayerNorm): LayerNorm((384,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (encoder): BertEncoder(
    (layer): ModuleList(
      (0-5): 6 x BertLayer(
        (attention): BertAttention(
          (self): BertSelfAttention(
            (query): Linear(in_features=384, out_features=384, bias=True)
            (key): Linear(in_features=384, out_features=384, bias=True)
            (value): Linear(in_features=384, out_features=384, bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output): BertSelfOutput(
            (dense): Linear(in_features=384, out_features=384, bias=True)
            (LayerNorm): LayerNorm((384,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (intermediate): BertIntermediate(
          (dense): Linear(in_features=384, out_features=1536, bias=True)
          (intermediate_act_fn): GELUActivation()
        )
        (output): BertOutput(
          (dense): Linear(in_features=1536, out_features=384, bias=True)
          (LayerNorm): LayerNorm((384,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
  )
  (pooler): BertPooler(
    (dense): Linear(in_features=384, out_features=384, bias=True)
    (activation): Tanh()
  )
)
```

Examples - GPT2

```
GPT2Model(
  (wte): Embedding(50257, 768)
  (wpe): Embedding(1024, 768)
  (drop): Dropout(p=0.1, inplace=False)
  (h): ModuleList(
    (0-11): 12 x GPT2Block(
      (ln_1): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (attn): GPT2Attention(
        (c_attn): Conv1D()
        (c_proj): Conv1D()
        (attn_dropout): Dropout(p=0.1, inplace=False)
        (resid_dropout): Dropout(p=0.1, inplace=False)
      )
      (ln_2): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (mlp): GPT2MLP(
        (c_fc): Conv1D()
        (c_proj): Conv1D()
        (act): NewGELUActivation()
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
  )
  (ln_f): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
)
```

Examples - Llama 2 7B

```
LlamaModel(
  (embed_tokens): Embedding(32000, 4096)
  (layers): ModuleList(
    (0-31): 32 x LlamaDecoderLayer(
      (self_attn): LlamaSdpaAttention(
        (q_proj): Linear(in_features=4096, out_features=4096, bias=False)
        (k_proj): Linear(in_features=4096, out_features=4096, bias=False)
        (v_proj): Linear(in_features=4096, out_features=4096, bias=False)
        (o_proj): Linear(in_features=4096, out_features=4096, bias=False)
        (rotary_emb): LlamaRotaryEmbedding()
      )
      (mlp): LlamaMLP(
        (gate_proj): Linear(in_features=4096, out_features=11008, bias=False)
        (up_proj): Linear(in_features=4096, out_features=11008, bias=False)
        (down_proj): Linear(in_features=11008, out_features=4096, bias=False)
        (act_fn): SiLU()
      )
      (input_layernorm): LlamaRMSNorm()
      (post_attention_layernorm): LlamaRMSNorm()
    )
  )
  (norm): LlamaRMSNorm()
)
```

Examples - Mistral 7B

```
MistralModel(
  (embed_tokens): Embedding(32768, 4096)
  (layers): ModuleList(
    (0-31): 32 x MistralDecoderLayer(
      (self_attn): MistralSdpaAttention(
        (q_proj): Linear(in_features=4096, out_features=4096, bias=False)
        (k_proj): Linear(in_features=4096, out_features=1024, bias=False)
        (v_proj): Linear(in_features=4096, out_features=1024, bias=False)
        (o_proj): Linear(in_features=4096, out_features=4096, bias=False)
        (rotary_emb): MistralRotaryEmbedding()
      )
      (mlp): MistralMLP(
        (gate_proj): Linear(in_features=4096, out_features=14336, bias=False)
        (up_proj): Linear(in_features=4096, out_features=14336, bias=False)
        (down_proj): Linear(in_features=14336, out_features=4096, bias=False)
        (act_fn): SiLU()
      )
      (input_layernorm): MistralRMSNorm()
      (post_attention_layernorm): MistralRMSNorm()
    )
  )
  (norm): MistralRMSNorm()
)
```

Direct calculations

Prof Tom Yeh's work on AI by hand:

- Simplies complex topics for presentation.
- Has the Transformer's architecture *here*

n.b. Interactive visualization available *here*.