



**NUS**  
National University  
of Singapore

| **Computing**

# **CS4248: Natural Language Processing**

## **Lecture 10 — Transformers & LLMs**

# Announcements

## Projects

- Grades and comments for Intermediate Update posted
- (Extended for Week 12) Optional consultation session – you can register [here](#)

## Deadlines

- Standard Project Report Submission Deadline: Thu, Apr 18, 23:59 SGT
- STePS Projects:
  - STePS Public Poster Presentation: Wed, Apr 17, 15:00 – 20:00 SGT  
(Instructors and Project Mentors will let you know rough timing windows for oral viva)
  - Clarification Submission Deadline (if any): Thu, Apr 18, 23:59 SGT

# Recap of Week 09

## Context-Free Grammars (CFGs)

- Context-Free Grammars
  - Most common way to capture constituency and ordering → good fit for natural language!  
(in fact, context-free grammars were first used to study human languages to describe the structure of sentences)
  - Define what meaningful constituents are and how a constituent is formed out of other constituents
  - More powerful than RegExs as they can express recursive structure  
(in contrast, context free grammars can describe regular languages)

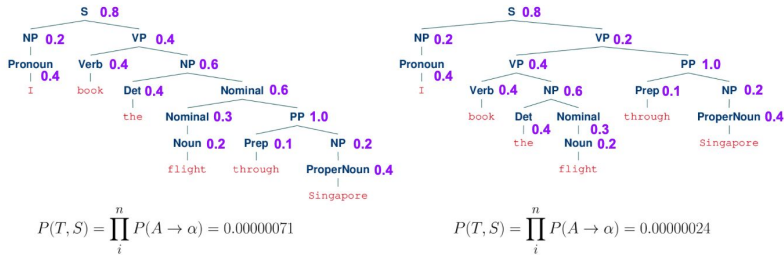
special start symbol → S → NP VP  
NP → Det Noun  
VP → Verb NP  
Det → a | the  
Noun → man | meal | flight  
Verb → saw | booked

set of rules or productions

- Non-terminal symbols**
- Symbols that can be replaced according to rules
  - For natural language grammars: phrase names, part of speech
- Terminal symbols**
- May be the output of a rule; cannot be changed/replaced further
  - For natural language grammars: words/tokens

## PCFG — Probability of a Parse Tree

- Probability of parse tree = product of probabilities of all rules
  - In practice, sum up log probabilities to avoid arithmetic underflow



## CYK — Walkthrough

I	book	the	flight	through	Singapore
[0,1] Pronoun, NP	[0,2] S	[0,3]	[0,4]	[0,5]	[0,6]
	[1,2] S, VP, Nominal, Noun, Verb	[1,3]	[1,4]	[1,5]	[1,6]
		[2,3] Det	[2,4] NP	[2,5]	[2,6]
			[3,4] Nominal, Noun	[3,5]	[3,6]
				[4,5] Prep	[4,6] PP
					[5,6] ProperNoun, NP

Cells for spans of length L > 1  
→ Check for each binary split if there is a production rule that can generate split

Example: Cell [0,2]  
→ only 1 binary split: [0,1] / [1,2]

Check each possible pair of non-terminals of binary split is the RHS of an existing production rule → Yes, add LHS to cell

LHS	RHS
—	Pronoun S
—	Pronoun VP
—	Pronoun Nominal
—	Pronoun Noun
—	Pronoun Verb
—	NP S
S	NP VP
—	NP Nominal
—	NP Noun
—	NP Verb

Only this rule exists in our grammar

## Evaluation of Parse Trees — Example

Tuples only present in correct tree: (NP, 3, 6), (Nominal, 4, 6)

Tuples resnt in both trees: (NP, 1, 1), (Pronoun, 1, 1), (VP, 2, 2), (Verb, 2, 2), (Det, 3, 3), (Nominal, 4, 4), (Noun, 4, 4), (Prep, 5, 5), (ProperNoun, 6, 6), (PP, 5, 6), (NP, 6, 6)

Tuples only present in computed tree: (VP, 2, 4), (NP, 3, 4)

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{11}{11 + 2} = 0.85$$

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{11}{11 + 2} = 0.85$$

$$f1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = 0.85$$

TP = # tuples in both trees

FP = # tuples only in the computed tree

FN = # tuples only in the correct tree

# Outline

- **Contextual Word Embeddings**

- Motivation
- ELMo

- **Transformers**

- Positional Encoding
- Core Layers
- Encoder & Decoder

- **Extended Concepts**

- Masking
- Restricted Attention

- **Transformer-based LLMs**

- Overview
- Encoder-only: BERT, RoBERTa
- Encoder-Decoder: T5, BART
- Decoder-only: GPT, LLaMA
- Opportunities & Challenges

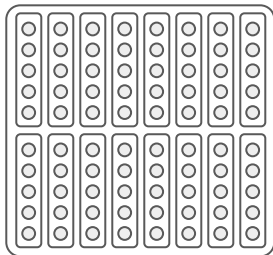
# Supervised Training (RNN)

Quick Quiz: Which model is easier to build? Why?

## Task A: Learning a Language Model

[...] Precipitation forms as smaller droplets coalesce via collision with other raindrops or ice crystals within a cloud. Short, intense periods of rain in scattered locations are called “showers”. [...]

“some” encoder network

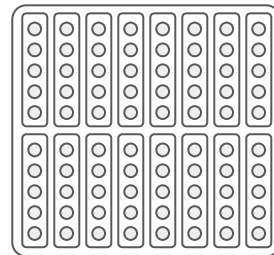


[...] Precipitation forms as smaller droplets coalesce via collision with other raindrops or ice crystals within a cloud. Short, intense periods of rain in scattered locations are called “showers”. [...]

## Task B: Learning a QA System

within a cloud

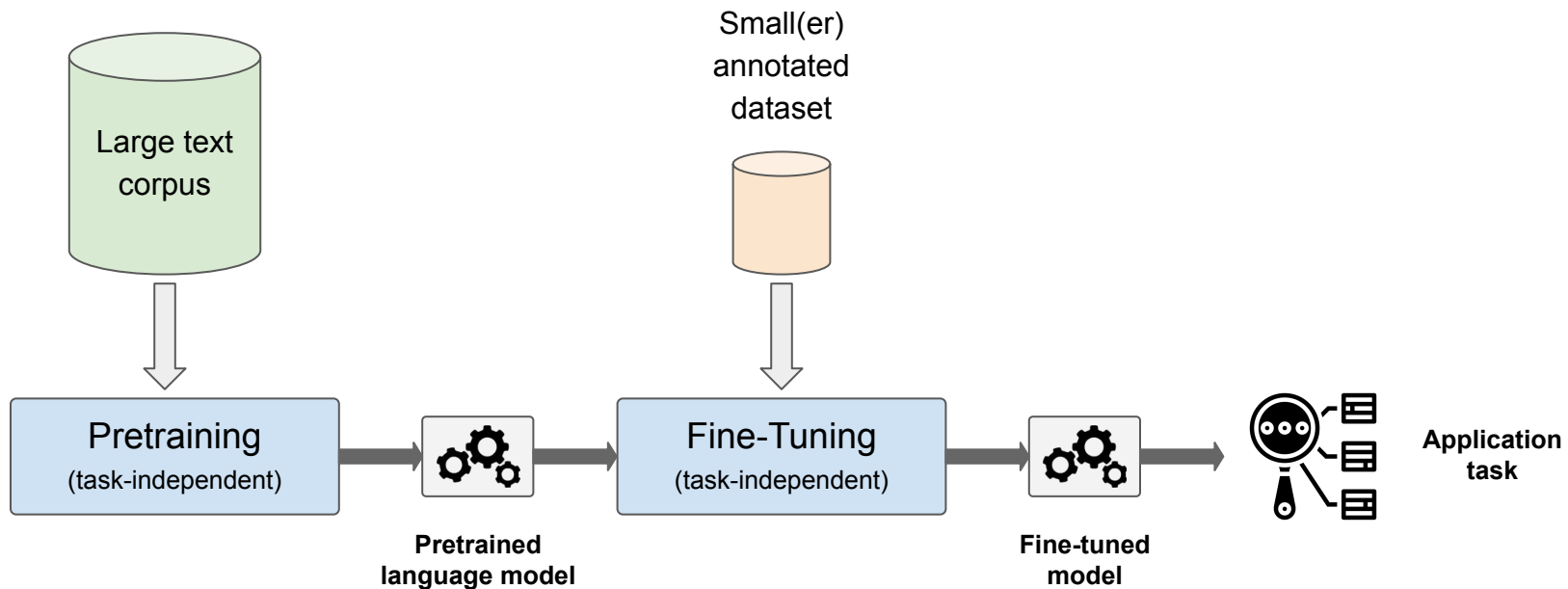
“some” encoder network



Where do water droplets collide with ice crystals to form precipitation?

[...] Precipitation forms as smaller droplets coalesce via collision with other raindrops or ice crystals **within a cloud**. Short, intense periods of rain in scattered locations are called “showers”. [...]

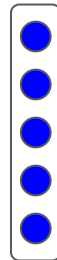
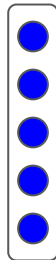
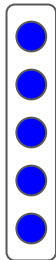
# Transfer Learning for NLP Models



# Transfer Learning with Word2Vec (or GloVe)

- Word2Vec: (almost) context-independent
  - BoW model → no consideration of word order
  - Limited window size → no consideration of whole sentence
  - Combining all the senses of a word into a single vector

*“A light wind will make the traffic light collapse and light up in flames.”*

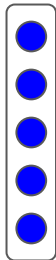


Problem: Same word vector for all occurrences of “light”!

# Goal: Contextualized Word Embeddings

- What we want
  - Word representations should vary depending on context
  - Context = whole sentence + word order

*“A light wind will make the traffic light collapse and light up in flames.”*



~ weak, soft mild



~ glow, brightness



~ ignite, burn, kindle



# Outline

- **Contextual Word Embeddings**

- Motivation
- **ELMo**

- **Transformers**

- Positional Encoding
- Core Layers
- Encoder & Decoder

- **Extended Concepts**

- Masking
- Restricted Attention

- **Transformer-based LLMs**

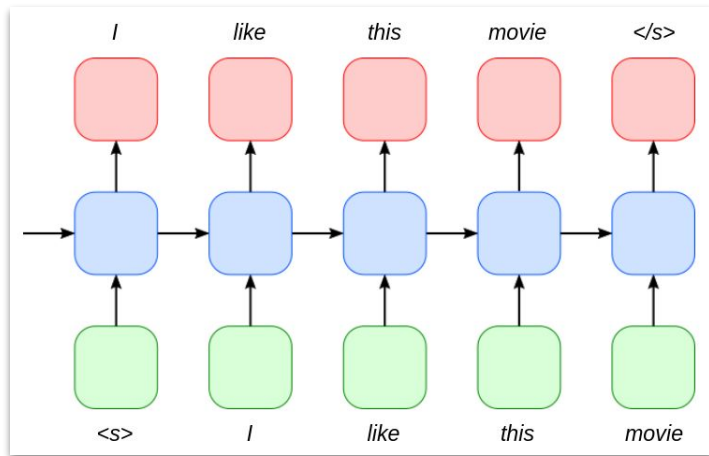
- Overview
- Encoder-only: BERT, RoBERTa
- Encoder-Decoder: T5, BART
- Decoder-only: GPT, LLaMA
- Opportunities & Challenges

# ELMo — Embeddings from Language Model

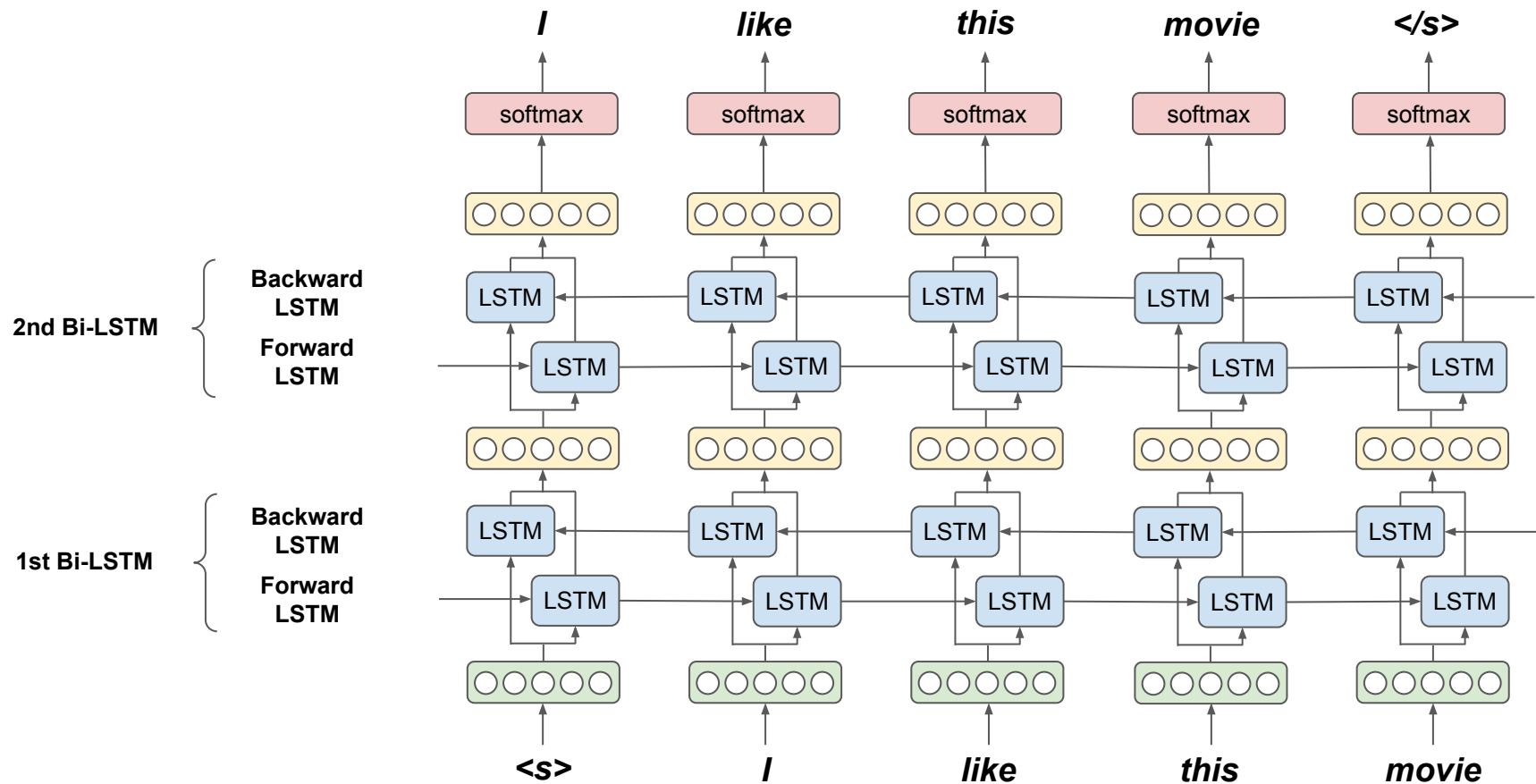
- ELMo = RNN-based Language model, but...

- LSTM instead of Vanilla RNN  
(better handling of long dependencies)
- Bi-LSTM — Bidirectional LSTM  
(forward and backward processing of sequence)
- Two Bi-LSTM layers  
(output of 1st layer = input of 2nd layer)

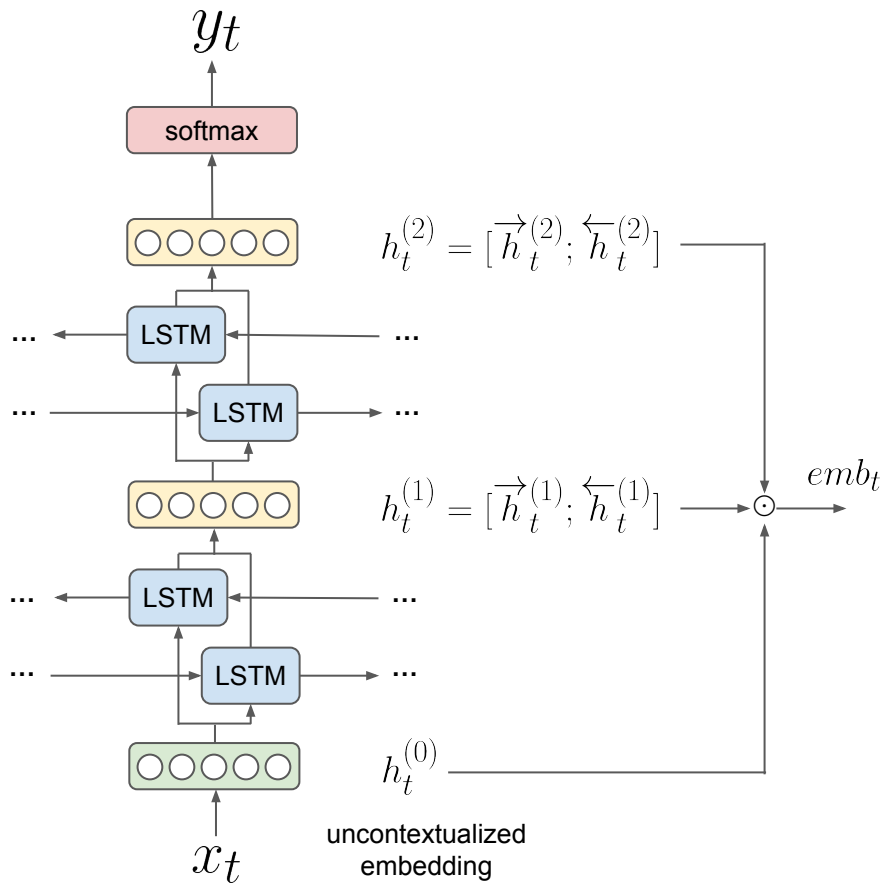
Recall: Vanilla RNN Language Model



# ELMo



# ELMo — Final Embeddings



Final embedding = “some” function of  $h_t^{(i)}$

Simplest case: top layer  $emb_t = h_t^{(2)}$

Generalized approach: weighted sum

$$emb_t = \gamma \sum_{j=0}^2 s_j h_t^{(j)} \quad , \quad \text{with} \quad \sum_{j=1}^2 s_j = 1$$

scaling factor  $\gamma$  and normalized weight  $s_j$  are derived from task-dependent values.

# ELMo — Evaluation

- Improvement of NLP downstream tasks

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	$88.7 \pm 0.17$	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)	$91.93 \pm 0.19$	90.15	$92.22 \pm 0.10$	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	$54.7 \pm 0.5$	3.3 / 6.8%

# ELMo — Evaluation

- Qualitative understanding what ELMo learns

Source		Nearest Neighbors
GloVe	play	playing, game, games, played, players, plays, player, Play, football, multiplayer
biLM	Chico Ruiz made a spectacular <u>play</u> on Alusik 's grounder {...}	Kieffer , the only junior in the group , was commended for his ability to hit in the clutch , as well as his all-round excellent <u>play</u> .
	Olivia De Havilland signed to do a Broadway <u>play</u> for Garson {...}	{...} they were actors who had been handed fat roles in a successful <u>play</u> , and had talent enough to fill the roles competently , with nice understatement .

# Outline

- Contextual Word Embeddings
  - Motivation
  - ELMo
- **Transformers**
  - Positional Encoding
  - Core Layers
  - Encoder & Decoder
- Extended Concepts
  - Masking
  - Restricted Attention
- Transformer-based LLMs
  - Overview
  - Encoder-only: BERT, RoBERTa
  - Encoder-Decoder: T5, BART
  - Decoder-only: GPT, LLaMA
  - Opportunities & Challenges

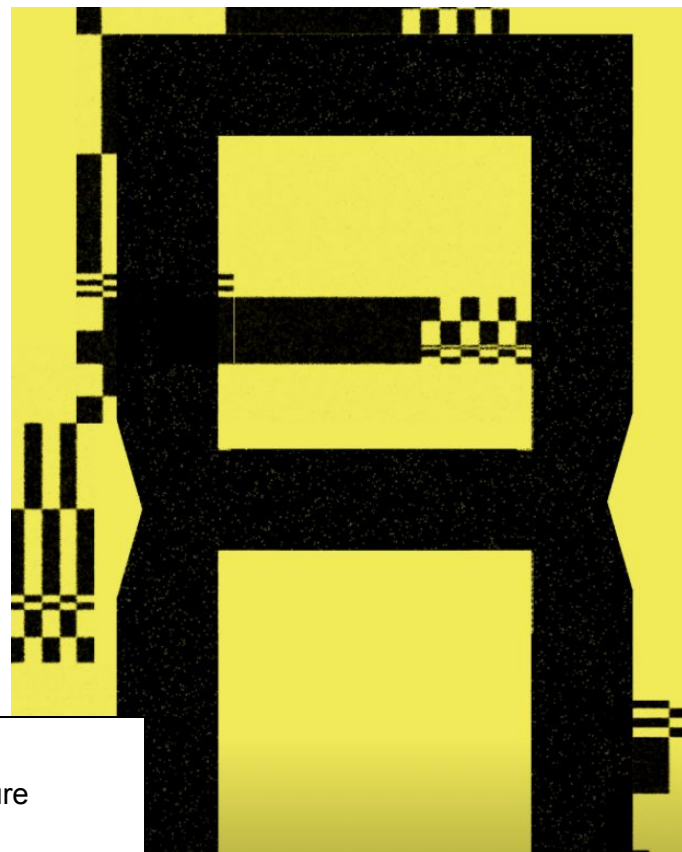
# Pre-Lecture Activity for Next Week

Read [8 Google Employees Invented Modern AI. Here's the Inside Story](#)  
(Wired Article)

Apply your own (self-)attention to the article.  
Quote a sentence of the article you think most  
or least strikes your attention. Tell us why.

**Side notes:**

- This task is meant as a warm-up to provide some context for the next lecture
- No worries if you get lost; we will talk about this in the next lecture
- You can just copy-&-paste others' answers but this won't help you learn better







# RNN — Problem: (Very) Long Sequences

- Training

- **Vanishing & Exploding Gradients** problem (not detailed here)

- Information capture

- Hidden state  $h_t$  must capture all information from  $h_0, h_1, \dots, h_{t-1}$
- Information dilutes over time → **bottleneck**

- Performance

- Processing is intrinsically sequential → **no parallelization**
- GPU-based performance gain depends on parallelization

→ **Attention**



→ **Transformer**

# Transformer — Architecture

- Encoder–decoder architecture without recurrences

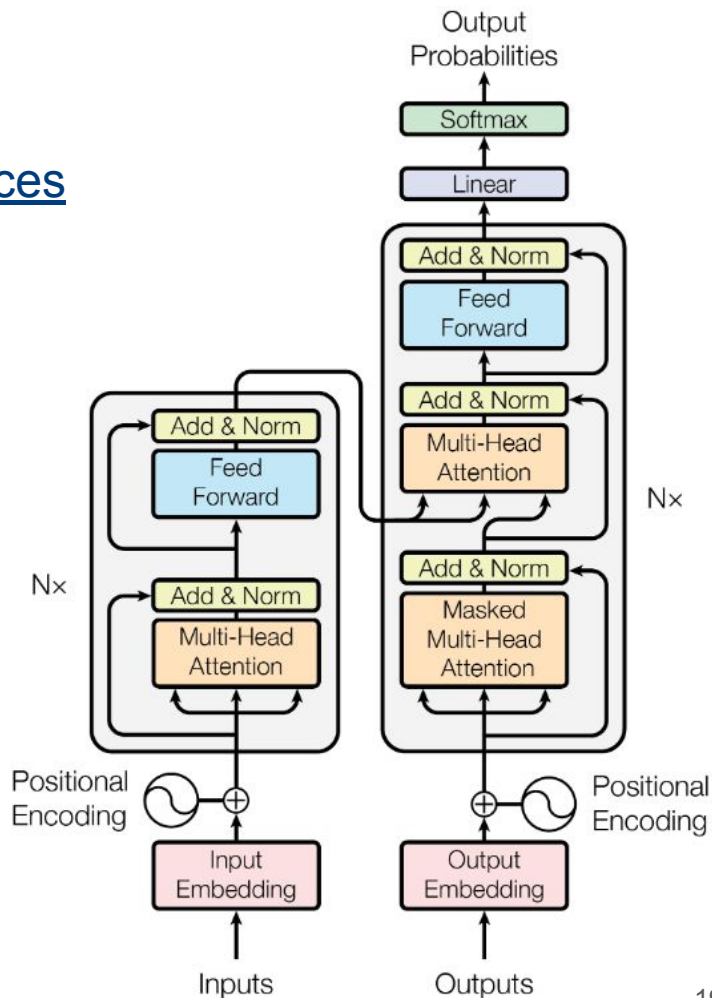
- No long-range dependencies → no bottleneck
- No sequential processing → easy to parallelize  
(note: this does not mean transformers are easier/faster to train!)

- Core concept: **Attention**

- Alignment scores between **all** word pairs

- Important: **Positional Embeddings**

- Preserve order of words in sequence



# Outline

- Contextual Word Embeddings
  - Motivation
  - ELMo
- **Transformers**
  - **Positional Encoding**
  - Core Layers
  - Encoder & Decoder
- Extended Concepts
  - Masking
  - Restricted Attention
- Transformer-based LLMs
  - Overview
  - Encoder-only: BERT, RoBERTa
  - Encoder-Decoder: T5, BART
  - Decoder-only: GPT, LLaMA
  - Opportunities & Challenges

# Positional Encodings

- Recall: RNNs process words sequentially

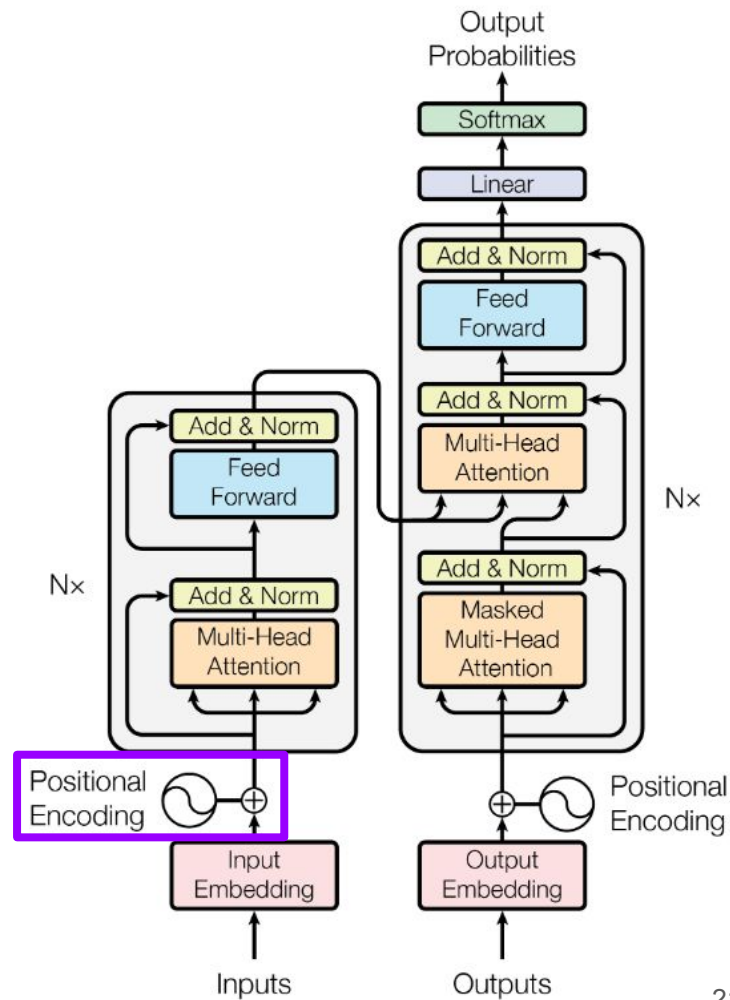
- Considers order of words
- Considers distance between words

- Transformers

- Process all words all at once
- No in-built mechanism to consider word order and word distances

## Can we somehow encode the position of words?

(as part of preprocessing the input for the transformer)



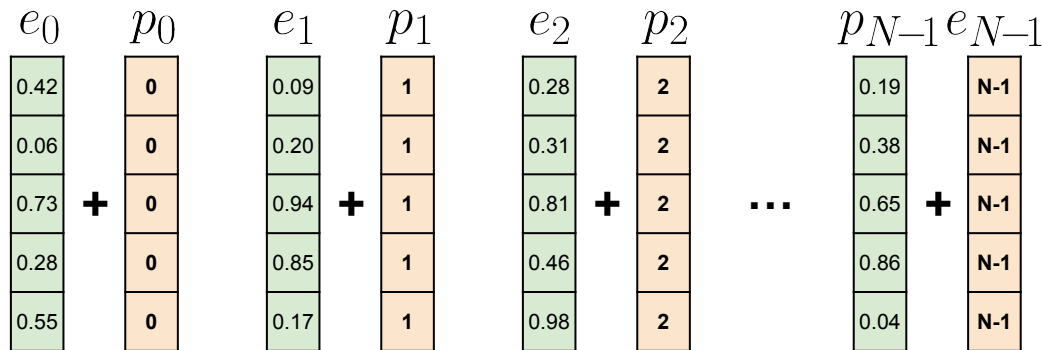


# What's Your Position?



# Positional Encodings — Naive Approach 1

- Set position embedding values to actual position



→ **Problem:** positional encodings quickly start “dominating” word embeddings

- Magnitude of positional embedding values depends on sequence length  $N$

# Positional Encodings — Naive Approach 2

- Set position embedding values to  $\frac{pos}{N-1}$

$e_0$	$p_0$	$e_1$	$p_1$	$e_2$	$p_2$	...	$p_{N-1}$	$e_{N-1}$
0.42	0	0.09	0.2	0.28	0.4		0.19	1
0.06	0	0.20	0.2	0.31	0.4		0.38	1
0.73	0	0.94	0.2	0.81	0.4		0.65	1
0.28	0	0.85	0.2	0.46	0.4		0.86	1
0.55	0	0.17	0.2	0.98	0.4		0.04	1

Example values for  $N=6$

- **Problem:** positional encodings depend on the length of the sequence length
- encoding of the same position will differ for sequences with different lengths



# Positional Encodings — Proposed Approach

- Set position embedding values to

$$PE_{(pos, 2i)} = \sin \left( \frac{pos}{10000^{2i/d_{model}}} \right)$$

$$PE_{(pos, 2i+1)} = \cos \left( \frac{pos}{10000^{2i/d_{model}}} \right)$$

	$p_0$		$p_{15}$		$p_{100}$	
$i = 0$	0.0		0.65		-0.51	
$i = 1$	1.0		0.93		-0.81	
$i = 2$	0.0	...	0.01	...	0.06	...
$i = 3$	1.0		1.0		1.0	
$i = 4$	0.0		0.0		0.0	

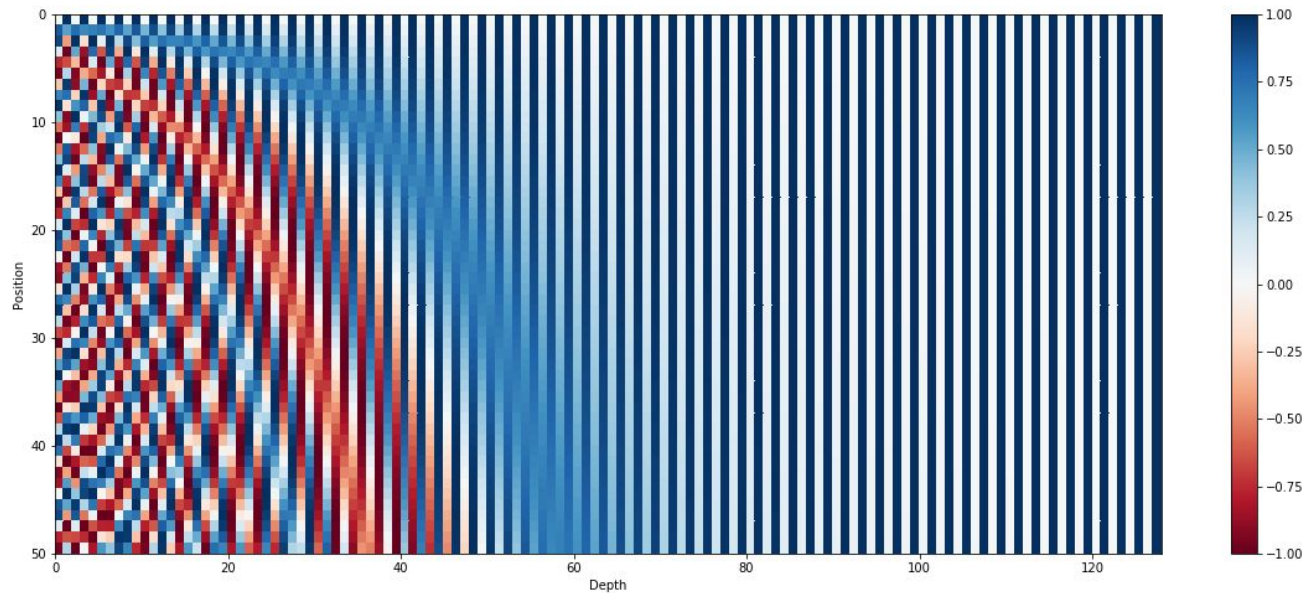
## Advantages:

- Unique encoding for each position
- All values are of interval  $[-1, 1]$
- Position encoding independent from  $N$

# Positional Encodings — Visualized

Representing a position/order (l) in binary and (r) in floats (positional encoding)

0 :	0	0	0	0
1 :	0	0	0	1
2 :	0	0	1	0
3 :	0	0	1	1
4 :	0	1	0	0
5 :	0	1	0	1
6 :	0	1	1	0
7 :	0	1	1	1



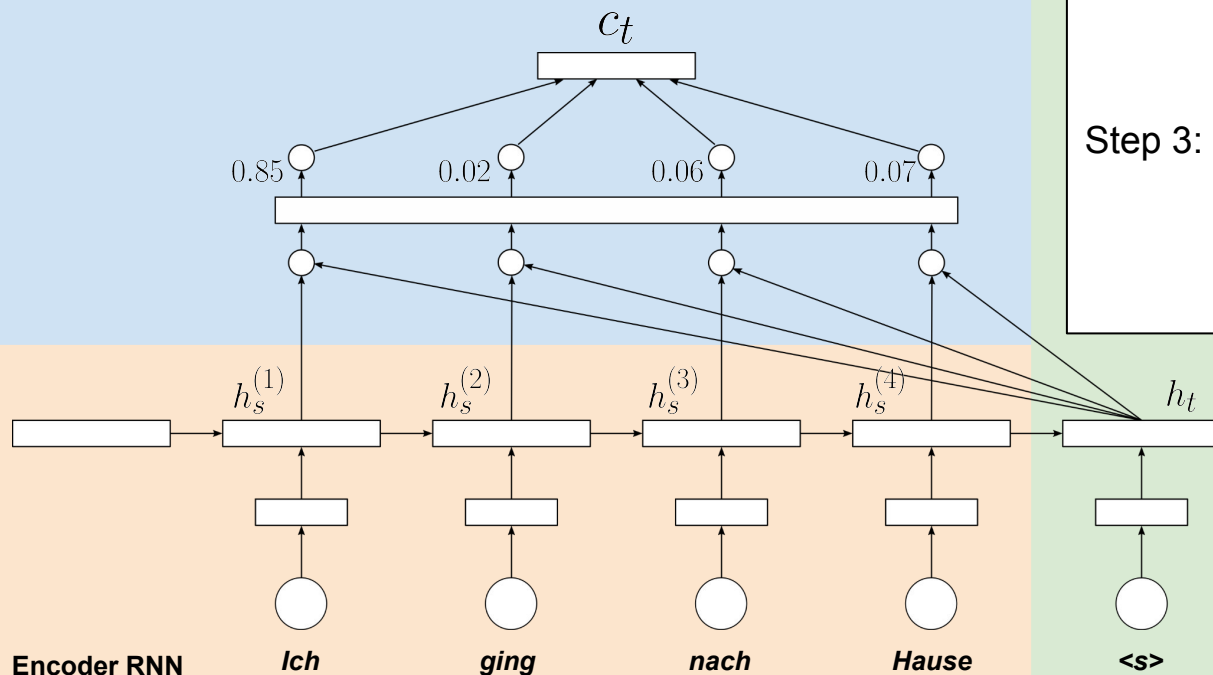
128-dimensional positional encoding for a sentence with length=50. Each row represents the embedding vector.

# Outline

- Contextual Word Embeddings
  - Motivation
  - ELMo
- **Transformers**
  - Positional Encoding
  - **Core Layers: Attention**
  - Encoder & Decoder
- Extended Concepts
  - Masking
  - Restricted Attention
- Transformer-based LLMs
  - Overview
  - Encoder-only: BERT, RoBERTa
  - Encoder-Decoder: T5, BART
  - Decoder-only: GPT, LLaMA
  - Opportunities & Challenges

# RNN Attention (revisited)

Attention Layer



Step 1: Calculation of **Attention Scores**

$$e_i = \text{score}(h_t, h_s^{(i)}) = \begin{cases} h_t^\top h_s^{(i)} \\ \dots \end{cases}$$

Step 2: Calculation of **Attention Weights**

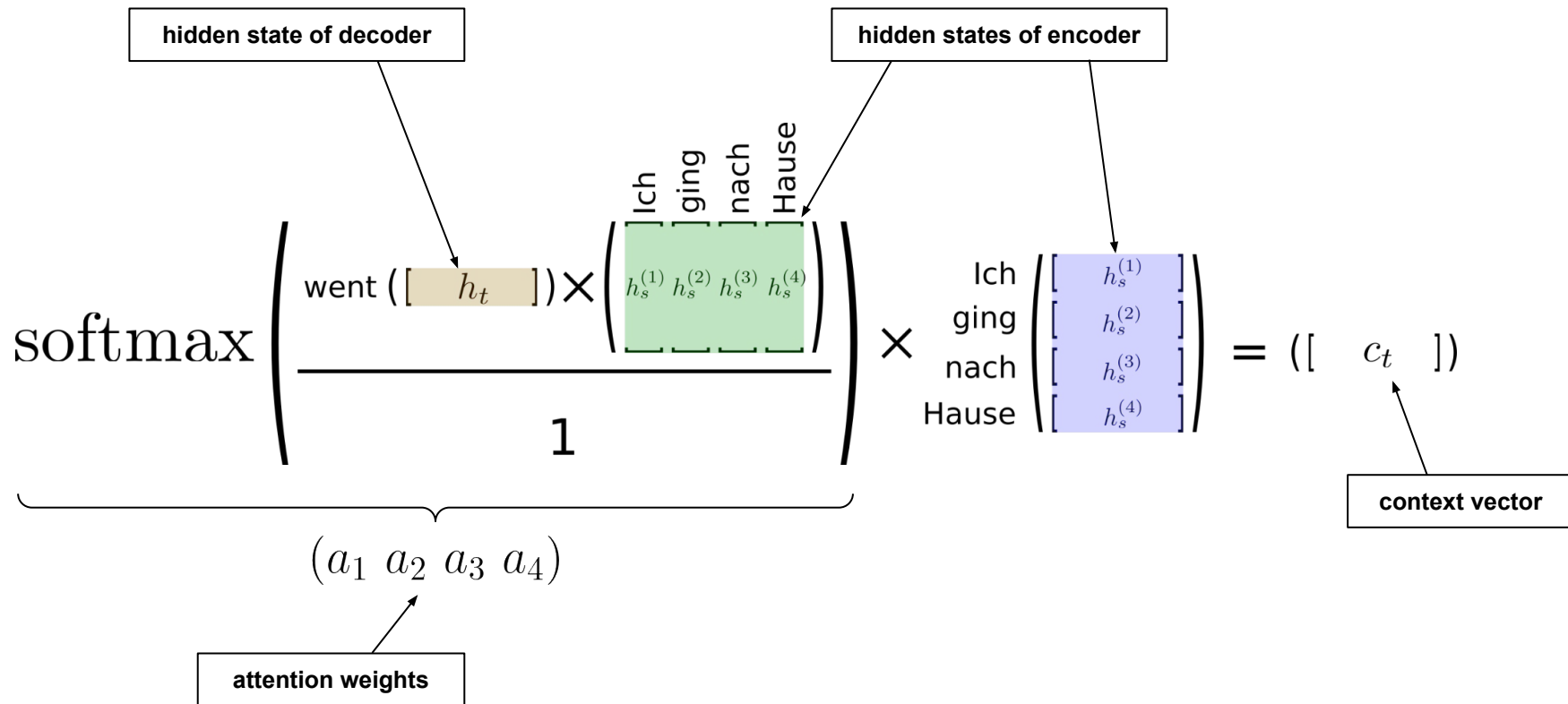
$$a_i = \frac{\exp(e_i)}{\sum_i \exp(e_i)}$$

Step 3: Calculation of **Context Vector**

$$c_t = \sum_i a_i \cdot h_s^{(i)}$$

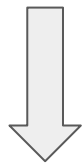
Decoder RNN

# RNN Attention (revisited)



# Attention — Generalized Definition

$$\text{softmax} \left( \frac{\text{went} \left( \begin{bmatrix} h_t \end{bmatrix} \right) \times \begin{pmatrix} \text{Ich} & \text{ging} & \text{nach} & \text{Hause} \\ h_s^{(1)} & h_s^{(2)} & h_s^{(3)} & h_s^{(4)} \end{pmatrix}}{1} \right) \times \begin{pmatrix} \text{Ich} \\ \text{ging} \\ \text{nach} \\ \text{Hause} \end{pmatrix} \begin{bmatrix} h_s^{(1)} \\ h_s^{(2)} \\ h_s^{(3)} \\ h_s^{(4)} \end{bmatrix} = \begin{bmatrix} c_t \end{bmatrix}$$



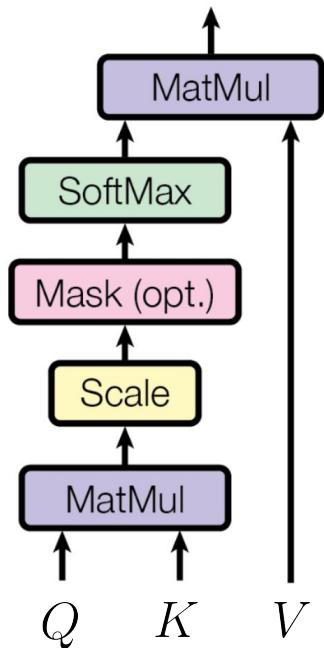
$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^\top}{\sqrt{d_k}} \right) V$$

## Scaled Dot-Product Attention

- Intuition: queries  $Q$ , keys  $K$ , values  $V$
- $k \in K, q \in Q$  are vectors of size  $d_k$
- scaling by  $\sqrt{d_k}$  leads to [more stable gradients](#)

# Scaled Dot-Product Attention

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

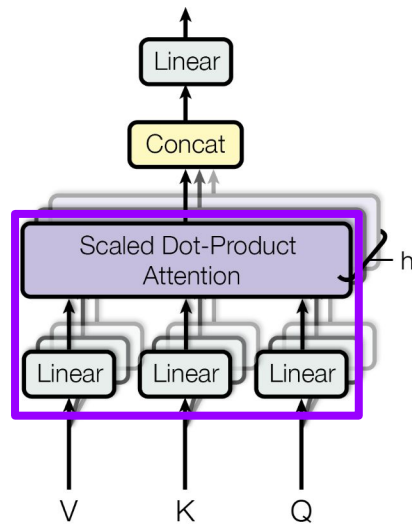


```
1 import torch
2 import torch.nn as nn
3
4
5 class Attention(nn.Module):
6     ### Implements Scaled Dot Product Attention
7
8     def __init__(self):
9         super().__init__()
10
11     def forward(self, Q, K, V, mask=None, dropout=None):
12         # All inputshapes: (batch_size B, seq_len L, model_size D)
13
14         # Perform Q*K^T (* is the dot product here)
15         # We have to use torch.matmul since we work with batches!
16         out = torch.matmul(Q, K.transpose(1, 2)) # => shape: (B, L, L)
17
18         # scale alignment scores
19         out = out / (Q.shape[-1] ** 0.5)
20
21         # Push through softmax layer
22         out = f.softmax(out, dim=-1)
23
24         # Multiply scaled alignment scores with values V
25         return torch.matmul(out, V)
```

# Attention Head

- Maps model size  $d_{model}$  to size of queries, keys, and values (by default: same size)
- Proposed:  $d_q = d_k = d_v = (d_{model}/h)$

Number of **heads**;  
see next slide



**Quick Quiz:** What do you think is the reason for dividing by the number of heads?

```
1 import torch
2 import torch.nn as nn
3
4
5 class AttentionHead(nn.Module):
6
7     def __init__(self, model_size, qkv_size):
8         super().__init__()
9         self.Wq = nn.Linear(model_size, qkv_size)
10        self.Wk = nn.Linear(model_size, qkv_size)
11        self.Wv = nn.Linear(model_size, qkv_size)
12        self.attention = Attention()
13
14    def forward(self, queries, keys, values):
15        # Computes scaled dot-product attention
16        return self.attention(self.Wq(queries),
17                               self.Wk(keys),
18                               self.Wv(values))
```



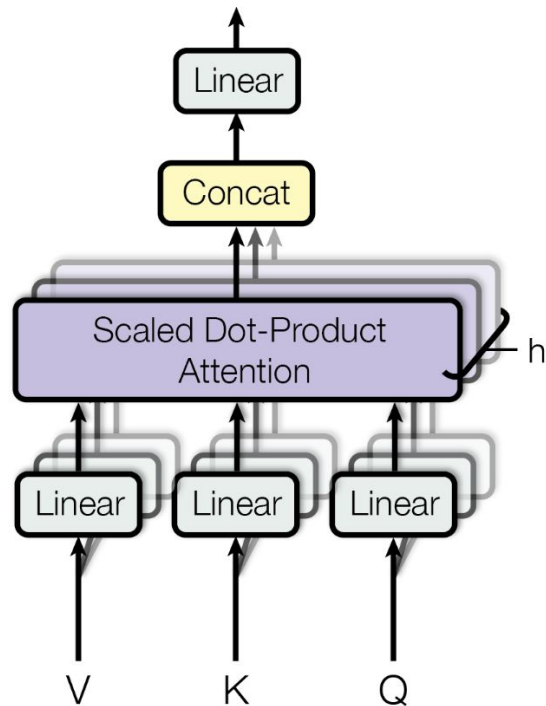
# Outline

- Contextual Word Embeddings
  - Motivation
  - ELMo
- Transformers
  - Positional Encoding
  - **Core Layers: Multi-Head Attention**
  - Encoder & Decoder
- Extended Concepts
  - Masking
  - Restricted Attention
- Transformer-based LLMs
  - Overview
  - Encoder-only: BERT, RoBERTa
  - Encoder-Decoder: T5, BART
  - Decoder-only: GPT, LLaMA
  - Opportunities & Challenges

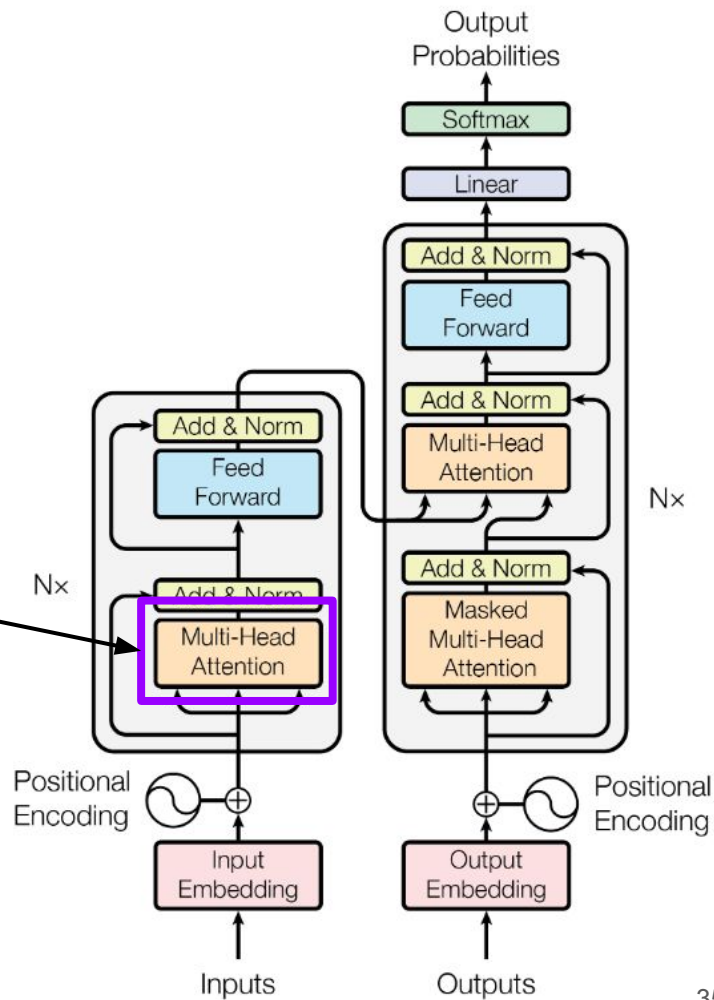
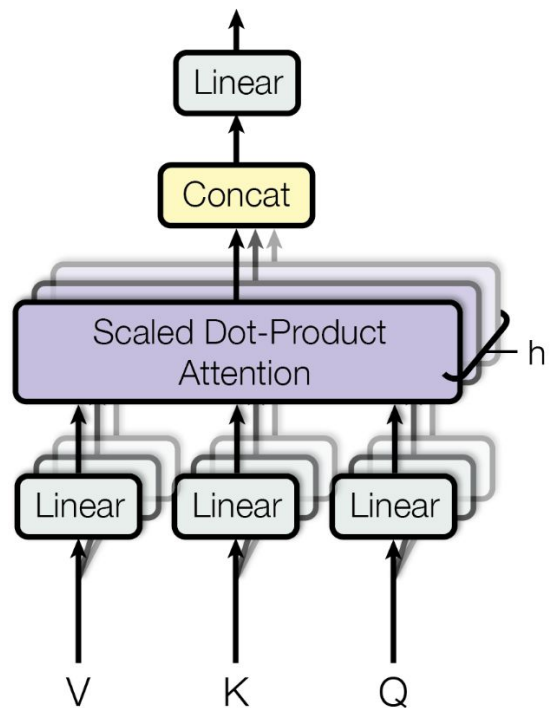
# Multi-Head Attention (MHA)

## Purpose / Intuition

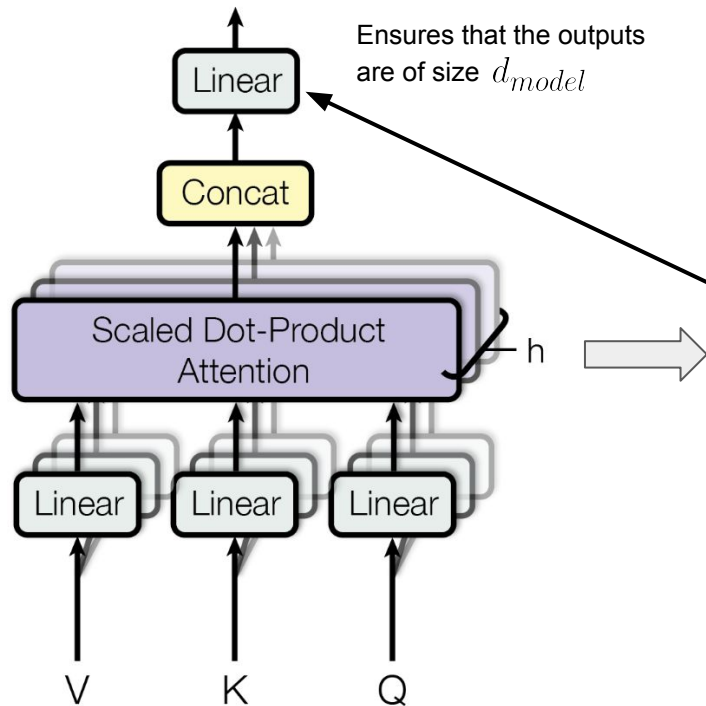
- A word may relate to multiple other words in a sentence
- A single Attention Head considers only one instance of relationship between pairs of words
- MHA allows to capture different relationships  
(note that each Attention Head comes with its own weight matrices!)
- Parameter: number of heads  $\rightarrow h$



# Multi-Head Attention



# Multi-Head Attention



```
1 import torch
2 import torch.nn as nn
3
4 class MultiHeadAttention(nn.Module):
5
6     def __init__(self, num_heads, model_size, qkv_size):
7         super().__init__()
8
9         # Define num_heads attention heads
10         self.heads = nn.ModuleList(
11             [ AttentionHead(model_size, qkv_size) for _ in range(num_heads) ]
12         )
13
14         # Linear layer to "unify" all heads into one
15         self.Wo = nn.Linear(num_heads * qkv_size, model_size)
16
17
18     def forward(self, query, key, value):
19         # Compute the outputs for all attention heads
20         out_heads = [ head(query, key, value) for head in self.heads ]
21
22         # Concatenate output of all attention heads
23         out = torch.cat(out_heads, dim=-1)
24
25         # Unify concatenated output to the model size
26         return self.Wo(out)
27
```

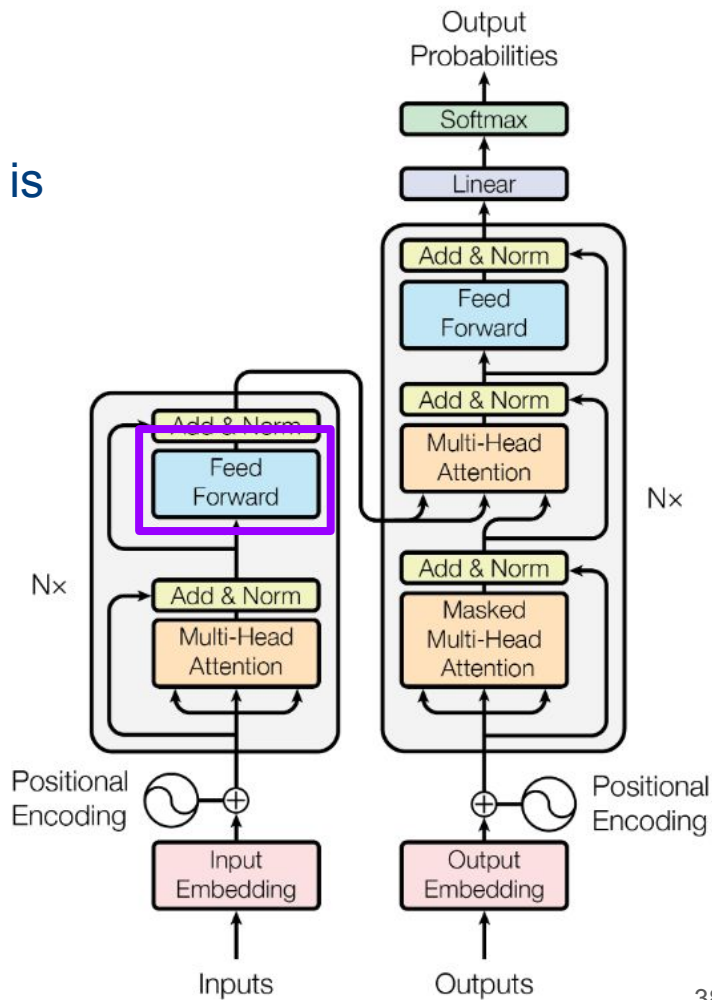
# Outline

- Contextual Word Embeddings
  - Motivation
  - ELMo
- **Transformers**
  - Positional Encoding
  - **Core Layers: Feed-Forward Layer**
  - Encoder & Decoder
- Extended Concepts
  - Masking
  - Restricted Attention
- Transformer-based LLMs
  - Overview
  - Encoder-only: BERT, RoBERTa
  - Encoder-Decoder: T5, BART
  - Decoder-only: GPT, LLaMA
  - Opportunities & Challenges

# Feed Forward Layer

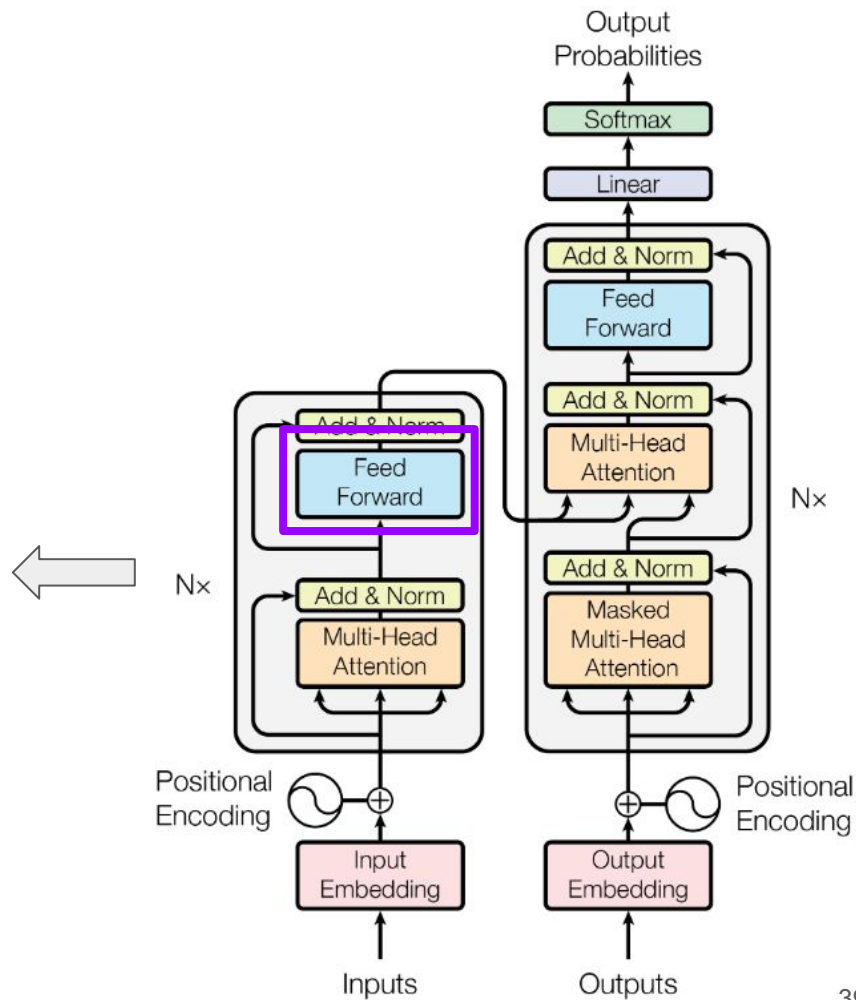
- The original paper doesn't say what its purpose is
- ...uh, increase capacity / complexity

Feed-forward layers constitute two-thirds of a transformer model's parameters, yet their role in the network remains under-explored.



# Feed Forward Layer

```
1 import torch
2 import torch.nn as nn
3
4
5 class FeedForward(nn.Module):
6
7     def __init__(self, model_size, hidden_size=2048):
8         super().__init__()
9
10        # Very simple 2-layer fully connected network
11        self.net = nn.Sequential(
12            nn.Linear(model_size, hidden_size),
13            nn.ReLU(),
14            nn.Linear(hidden_size, model_size),
15        )
16
17    def forward(self, X):
18        return self.net(X)
```



# Outline

- Contextual Word Embeddings
  - Motivation
  - ELMo
- **Transformers**
  - Positional Encoding
  - Core Layers
  - **Encoder & Decoder**
- Extended Concepts
  - Masking
  - Restricted Attention
- Transformer-based LLMs
  - Overview
  - Encoder-only: BERT, RoBERTa
  - Encoder-Decoder: T5, BART
  - Decoder-only: GPT, LLaMA
  - Opportunities & Challenges



# Encoder Layer

- Combines MHA and FF block  
(MHA: Multi-Head Attention, FF: Feed Forward)
- 3 additional concepts deployed

Oversimplified!

## (1) Residual Connections

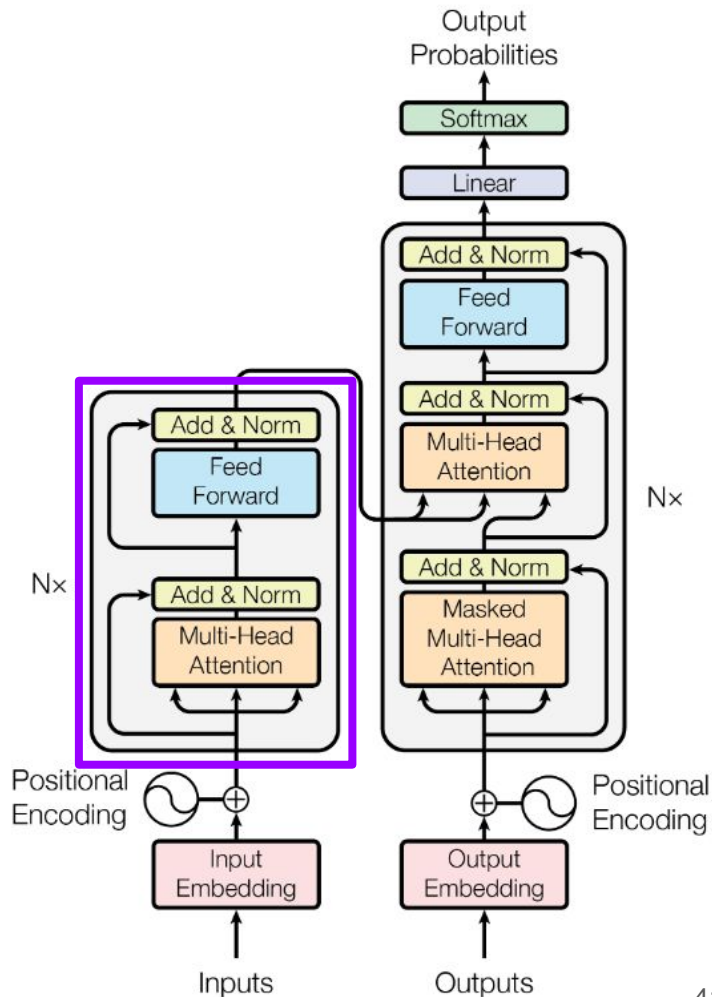
- Help mitigate the vanishing gradient problem

## (2) Dropout (after MHA/FF block; not shown)

- Regularization technique to prevent overfitting

## (3) Layer Normalization

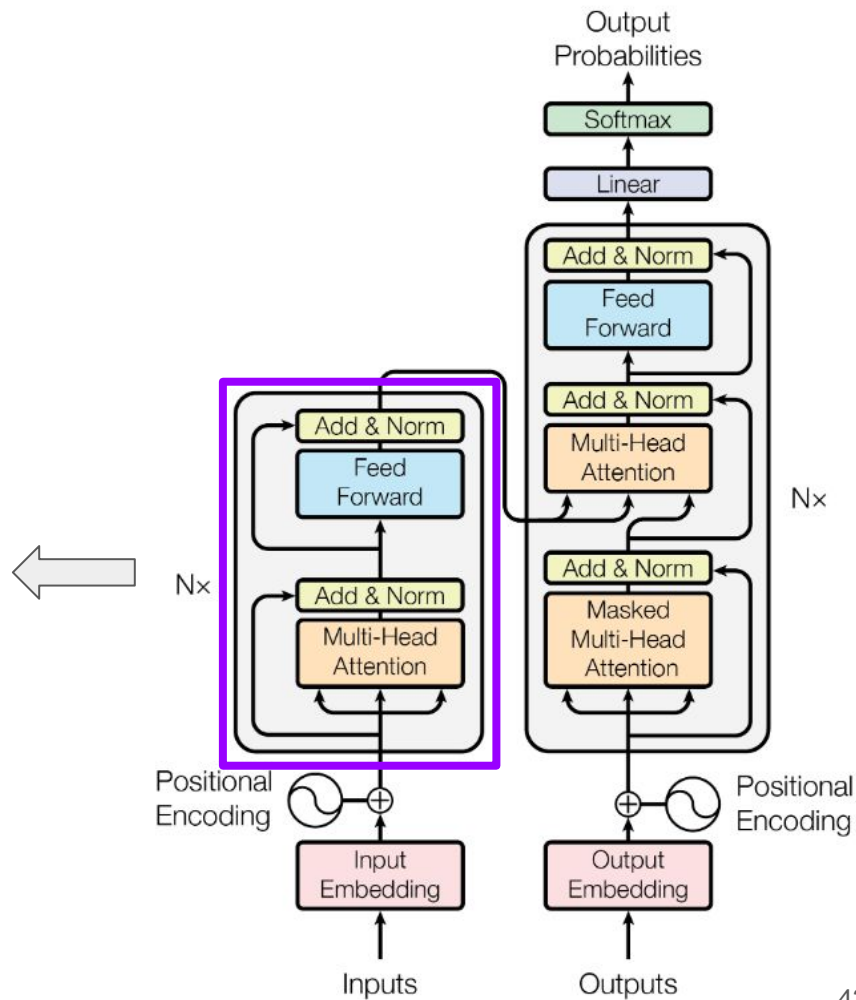
- Normalizes input across the features
- Improves the training stability and convergence



# Encoder Layer

```
1 import torch
2 import torch.nn as nn
3
4 class TransformerEncoderLayer(nn.Module):
5
6     def __init__(self, model_size, num_heads, ff_hidden_size, dropout):
7         super().__init__()
8
9         # Define sizes of Q/K/V based on model size and number of heads
10         qkv_size = max(model_size // num_heads, 1)
11
12         # MultiHeadAttention block
13         self.mha1 = MultiHeadAttention(num_heads, model_size, qkv_size)
14         self.dropout1 = nn.Dropout(dropout)
15         self.norm1 = nn.LayerNorm(model_size)
16
17         # FeedForward block
18         self.ff = FeedForward(model_size, ff_hidden_size)
19         self.dropout2 = nn.Dropout(dropout)
20         self.norm2 = nn.LayerNorm(model_size)
21
22     def forward(self, source):
23         # MultiHeadAttention block
24         out1 = self.mha1(source, source, source)
25         out1 = self.dropout1(out1)
26         out1 = self.norm1(out1 + source)
27
28         # FeedForward block
29         out2 = self.ff(out1)
30         out2 = self.dropout2(out2)
31         out2 = self.norm2(out2 + out1)
32
33         # Return final output
34         return out2
35
```

**Self-Attention**  
 $Q = K = V$



# Encoder — Self-Attention

- Example: German-to-English machine translation

$$\text{softmax} \left( \frac{\begin{matrix} \text{Ich} \\ \text{ging} \\ \text{nach} \\ \text{Hause} \end{matrix} \begin{bmatrix} Q \end{bmatrix} \times \begin{matrix} \text{Ich} \\ \text{ging} \\ \text{nach} \\ \text{Hause} \end{matrix} \begin{bmatrix} K^T \end{bmatrix}}{\sqrt{d_k}} \right) \times \begin{matrix} \text{Ich} \\ \text{ging} \\ \text{nach} \\ \text{Hause} \end{matrix} \begin{bmatrix} V \end{bmatrix} = \begin{matrix} \text{Ich} \\ \text{ging} \\ \text{nach} \\ \text{Hause} \end{matrix} \begin{bmatrix} \text{re-weighted } V \end{bmatrix}$$

word embeddings re-weighted based on attention weights

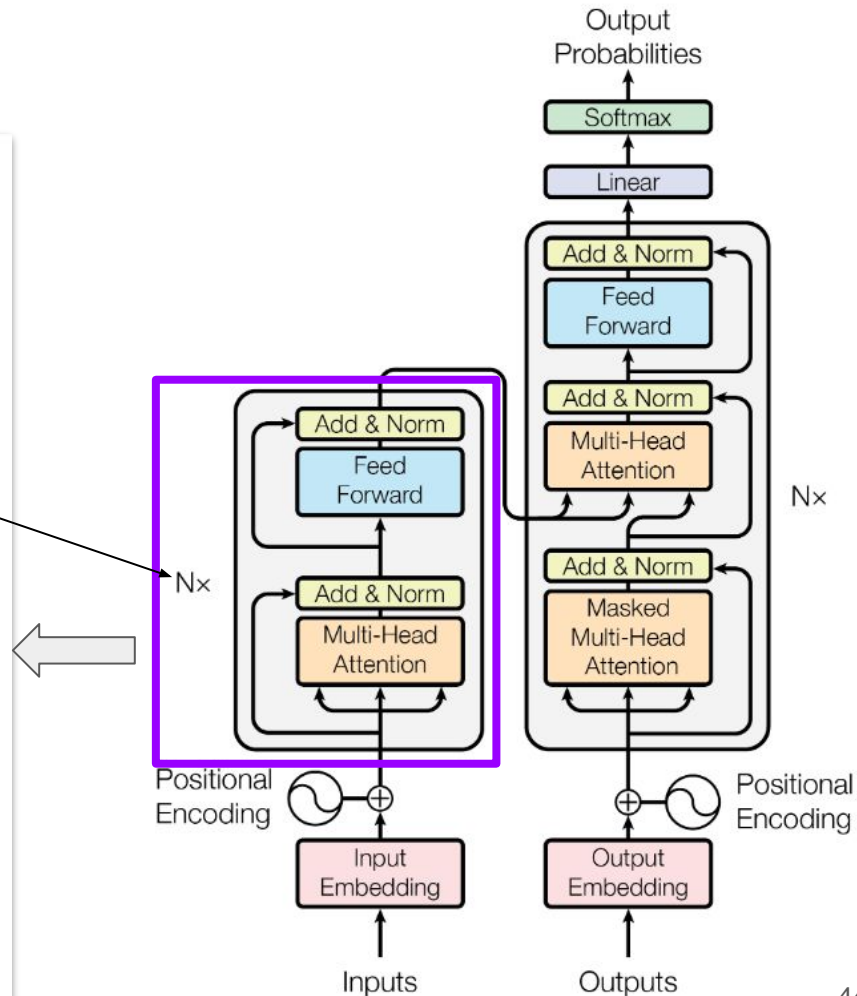
		Ich	ging	nach	Hause
Ich		●	●	●	●
ging		●	●	●	●
nach		●	●	●	●
Hause		●	●	●	●

**Self-Attention Matrix**

(rows sum up to 1!)

# Complete Encoder

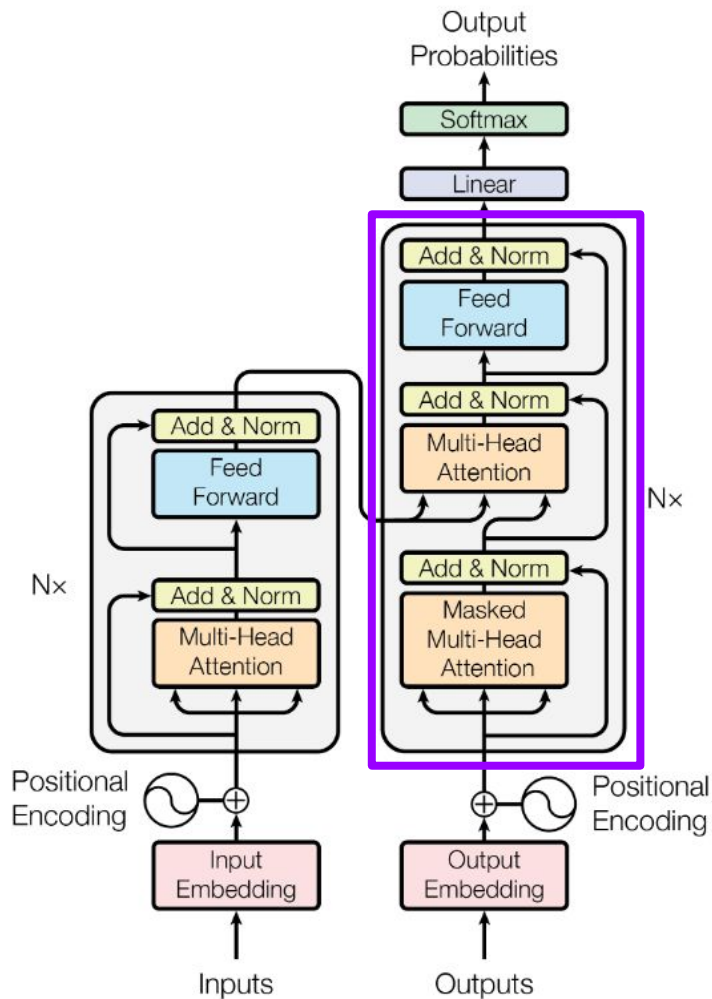
```
1 import torch
2 import torch.nn as nn
3
4
5 class TransformerEncoder(nn.Module):
6
7     def __init__(self,
8                 num_layers=6,           # Common default values
9                 model_size=512,        # used in original paper
10                 num_heads=8,
11                 ff_hidden_size=2048,
12                 dropout= 0.1):
13         super().__init__()
14
15         # Define num_layers (N) encoder layers
16         self.layers = nn.ModuleList(
17             [ TransformerEncoderLayer(model_size,
18                                     num_heads,
19                                     ff_hidden_size,
20                                     dropout)
21               for _ in range(num_layers)
22             ]
23         )
24
25     def forward(self, source):
26         # Push through each encoder layer
27         for l in self.layers:
28             source = l(source)
29         return source
```



# Decoder Layer

- The same components as Encoder Layer

- Multi-Head Attention but 2 MHA blocks  
(one for output, once for input/output)
- Feed Forward Layer
- The same additional concepts  
(residual connections, dropout, layer normalization)
- Multiple layers for complete decoder



# Decoder Layer

```

1 import torch
2 import torch.nn as nn
3
4
5 class TransformerDecoderLayer(nn.Module):
6
7     def __init__(self, model_size, num_heads, ff_hidden_size, dropout):
8         super().__init__()
9
10        # Define sizes of Q/K/V based on model size and number of heads
11        qkv_size = max(model_size // num_heads, 1)
12
13        # 1st MultiHeadAttention block (decoder input only)
14        self.mha1 = MultiHeadAttention(num_heads, model_size, qkv_size)
15        self.dropout1 = nn.Dropout(dropout)
16        self.norm1 = nn.LayerNorm(model_size)
17
18        # 2nd MultiHeadAttention block (encoder & decoder)
19        self.mha2 = MultiHeadAttention(num_heads, model_size, qkv_size)
20        self.dropout2 = nn.Dropout(dropout)
21        self.norm2 = nn.LayerNorm(model_size)
22
23        self.ff = FeedForward(model_size, ff_hidden_size)
24        self.dropout3 = nn.Dropout(dropout)
25        self.norm3 = nn.LayerNorm(model_size)
26
27    def forward(self, target, memory):
28        # 1st MultiHeadAttention block
29        out1 = self.mha1(target, target, target)
30        out1 = self.dropout1(out1)
31        out1 = self.norm1(out1 + target)
32        # 2nd MultiHeadAttention block
33        out2 = self.mha2(out1, memory, memory)
34        out2 = self.dropout2(out2)
35        out2 = self.norm2(out2 + out1)
36        # FeedForward block
37        out3 = self.ff(out2)
38        out3 = self.dropout3(out3)
39        out3 = self.norm3(out3 + out2)
40        # Return final output
41        return out3

```

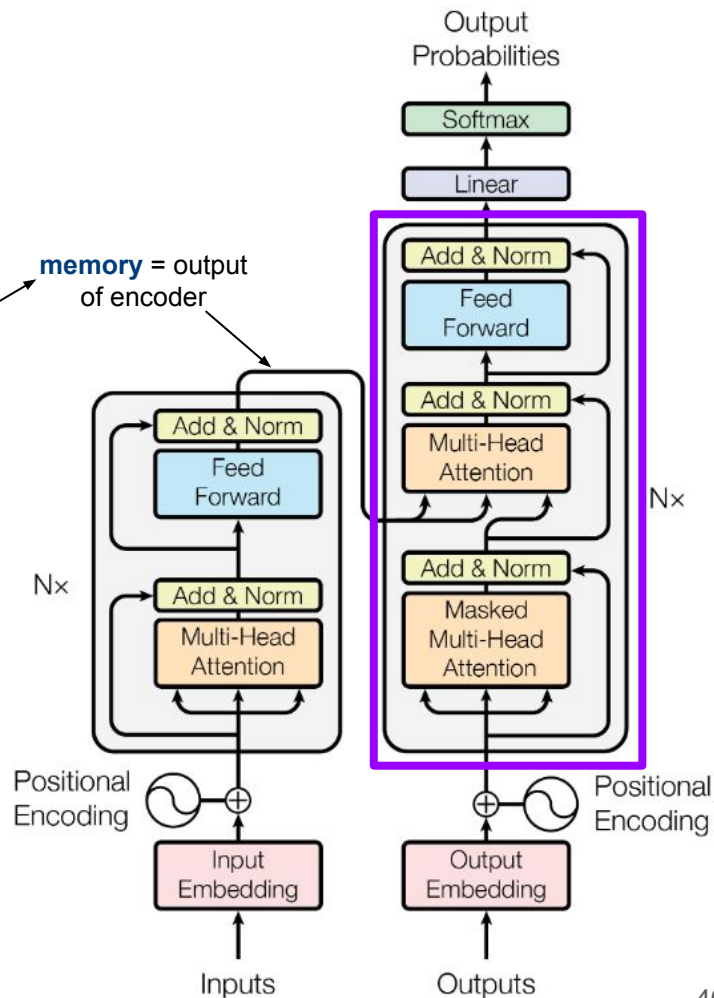
**Self-Attention**

$$Q = K = V$$

**Source-Target Attention**

$$Q \neq K = V$$

**memory** = output  
of encoder



# Decoder — Attentions

- Example: German-to-English machine translation

$$\text{softmax} \left( \frac{\begin{matrix} \text{went} \\ \text{home} \end{matrix} \begin{pmatrix} Q \end{pmatrix} \times \begin{pmatrix} \text{went} \\ \text{home} \end{pmatrix} \begin{pmatrix} K^T \end{pmatrix}}{\sqrt{d_k}} \right) \times \begin{matrix} \text{went} \\ \text{home} \end{matrix} \begin{pmatrix} V \end{pmatrix} = \begin{matrix} \text{went} \\ \text{home} \end{matrix} \begin{pmatrix} \phantom{V} \end{pmatrix}$$

**Self-Attention**

$$Q = K = V$$

$$\text{softmax} \left( \frac{\begin{matrix} \text{went} \\ \text{home} \end{matrix} \begin{pmatrix} Q \end{pmatrix} \times \begin{pmatrix} \text{Ich} \\ \text{ging} \\ \text{nach} \\ \text{Hause} \end{pmatrix} \begin{pmatrix} K^T \end{pmatrix}}{\sqrt{d_k}} \right) \times \begin{matrix} \text{Ich} \\ \text{ging} \\ \text{nach} \\ \text{Hause} \end{matrix} \begin{pmatrix} V \end{pmatrix} = \begin{matrix} \text{Ich} \\ \text{ging} \\ \text{nach} \\ \text{Hause} \end{matrix} \begin{pmatrix} \phantom{V} \end{pmatrix}$$

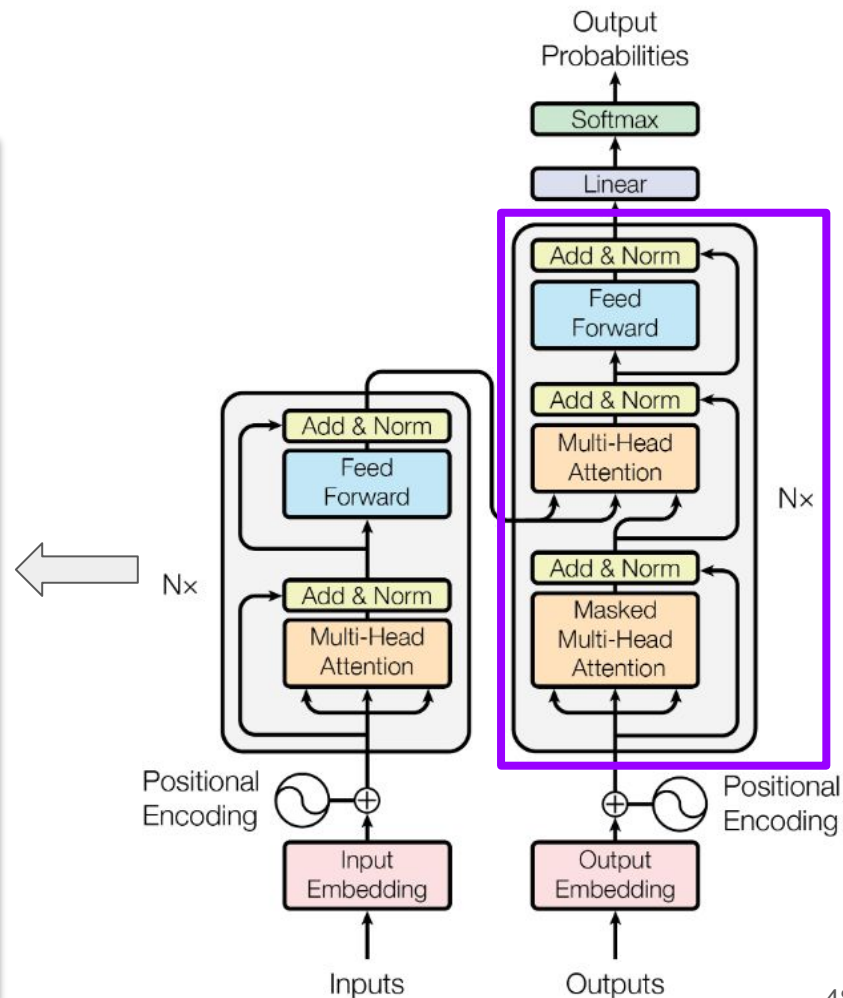
**Cross-Attention**

$$Q \neq K = V$$



# Complete Decoder

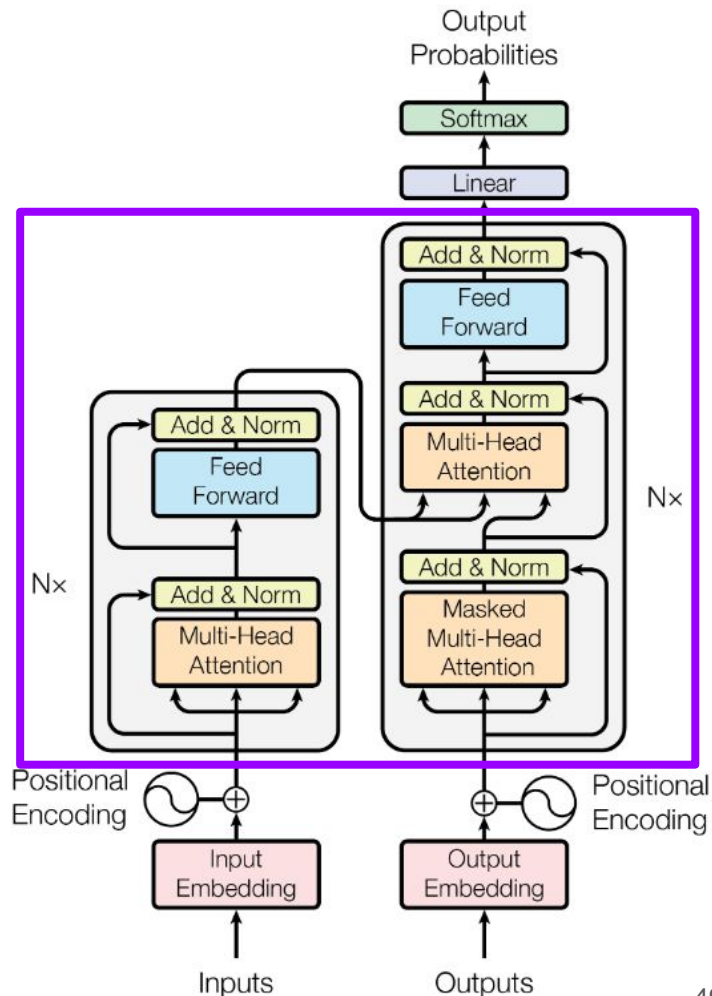
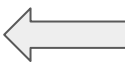
```
1 import torch
2 import torch.nn as nn
3
4
5 class TransformerDecoder(nn.Module):
6
7     def __init__(self,
8                 num_layers=6,           # Common default values
9                 model_size=512,        # used in original paper
10                 num_heads=8,
11                 ff_hidden_size=2048,
12                 dropout= 0.1):
13         super().__init__()
14
15         # Define num_layers (N) decoder layers
16         self.layers = nn.ModuleList(
17             [ TransformerDecoderLayer(model_size,
18                                     num_heads,
19                                     ff_hidden_size,
20                                     dropout)
21               for _ in range(num_layers)
22             ]
23         )
24
25     def forward(self, target, memory):
26         # Push through each decoder layer
27         for l in self.layers:
28             target = l(target, memory)
29         return target
```





# Complete Transformer

```
1 import torch
2 import torch.nn as nn
3
4
5 class Transformer(nn.Module):
6
7     def __init__(self,
8                 num_encoder_layers=6, # Common default values
9                 num_decoder_layers=6, # used in original paper
10                 model_size=512,
11                 num_heads=8,
12                 ff_hidden_size=2048,
13                 dropout= 0.1):
14         super().__init__()
15
16         # Define encoder
17         self.encoder = TransformerEncoder(
18             num_layers=num_encoder_layers,
19             model_size=model_size,
20             num_heads=num_heads,
21             ff_hidden_size=ff_hidden_size,
22             dropout=dropout
23         )
24
25         # Define decoder
26         self.decoder = TransformerDecoder(
27             num_layers=num_decoder_layers,
28             model_size=model_size,
29             num_heads=num_heads,
30             ff_hidden_size=ff_hidden_size,
31             dropout=dropout
32         )
33
34     def forward(self, source, target):
35         memory = self.encoder(source)
36         return self.decoder(target, memory)
```



# Outline

- Contextual Word Embeddings
  - Motivation
  - ELMo
- Transformers
  - Positional Encoding
  - Core Layers
  - Encoder & Decoder
- **Extended Concepts**
  - **Masking**
  - Restricted Attention
- Transformer-based LLMs
  - Overview
  - Encoder-only: BERT, RoBERTa
  - Encoder-Decoder: T5, BART
  - Decoder-only: GPT, LLaMA
  - Opportunities & Challenges

# Masking — Purpose

- Masking: Ignore attention between “invalid” words — most commonly
  - Padding in batches with sequences of different lengths
  - “Hidden” words in models for Language Modeling
  - “Future” words in models for text generation
- Masking padded words

<i>best</i>	<i>movie</i>	<i>ever</i>	<PAD>	<PAD>
<i>i</i>	<i>really</i>	<i>liked</i>	<i>only</i>	<i>the</i>
<i>top</i>	<i>movie</i>	<PAD>	<PAD>	<PAD>
<i>such</i>	<i>a</i>	<i>dumb</i>	<i>and</i>	<i>silly</i>
<i>could</i>	<i>have</i>	<i>been</i>	<i>much</i>	<i>worse</i>
<i>the</i>	<i>story</i>	<i>was</i>	<i>not</i>	<i>that</i>



Masking matrix  $M$

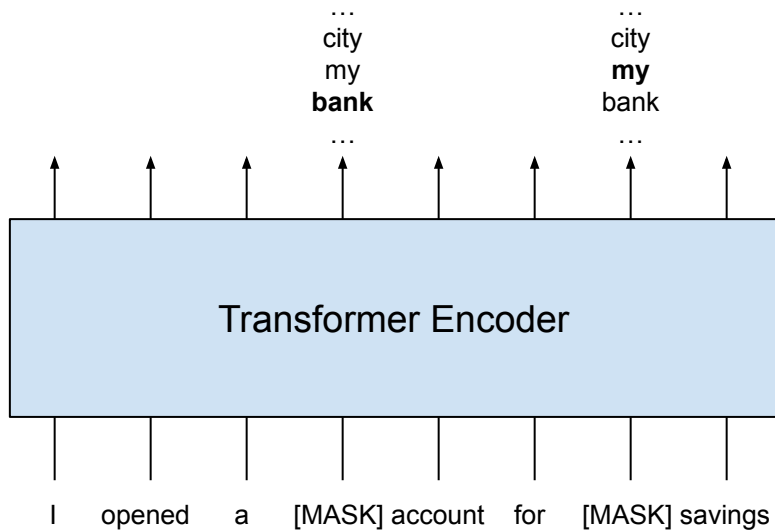
$$\begin{pmatrix} 0 & 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\infty & -\infty & -\infty \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\begin{aligned} a_{ij} + 0 &= a_{ij} \\ a_{ij} + (-\infty) &= -\infty \end{aligned}$$

Becomes 0 probability  
after Softmax!

# Masking for Language Modeling

- Masked Language Model — basic idea
  - Mask a random number of words in a input sequence (e.g., BERT: 15%)
  - Train model — transformer encoder — to predict masked words



Masking matrix  $M$

$$\begin{pmatrix} \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & -\infty & 0 & 0 & -\infty & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \end{pmatrix}$$

# Masking for Text Generation

## Decoder is autoregressive

- Output is generated word-by-word
- During training, decoder gets complete output sequence  
(i.e., the decoder could “cheat” and look at subsequent words)
- Ignore attention between a word and “future” words
- Only affects self-attention MHA block

## Example

- German-to-English machine translation

	<S>	I	went	home
<S>	0	$-\infty$	$-\infty$	$-\infty$
I	0	0	$-\infty$	$-\infty$
went	0	0	0	$-\infty$
home	0	0	0	0

# Outline

- Contextual Word Embeddings
  - Motivation
  - ELMo
- Transformers
  - Positional Encoding
  - Core Layers
  - Encoder & Decoder
- **Extended Concepts**
  - Masking
  - **Restricted Attention**
- Transformer-based LLMs
  - Overview
  - Encoder-only: BERT, RoBERTa
  - Encoder-Decoder: T5, BART
  - Decoder-only: GPT, LLaMA
  - Opportunities & Challenges

## Attention — Performance Considerations

- **Attentions is all you need...but it doesn't come for free**
  - Pro: no sequential processing required → easy parallelize
  - Cons: number of operations for attention:  $N^2$  ( $N$  = sequence length)

$$\underbrace{\text{softmax} \left( \frac{\begin{matrix} \text{Ich} \\ \text{ging} \\ \text{nach} \\ \text{Hause} \end{matrix} \begin{bmatrix} Q \end{bmatrix} \times \begin{matrix} \text{Ich} & \text{ging} & \text{nach} & \text{Hause} \\ \begin{bmatrix} K^T \end{bmatrix} \end{matrix}}{\sqrt{d_k}} \right)} \times \begin{matrix} \text{Ich} \\ \text{ging} \\ \text{nach} \\ \text{Hause} \end{matrix} \begin{bmatrix} V \end{bmatrix} = \begin{matrix} \text{Ich} \\ \text{ging} \\ \text{nach} \\ \text{Hause} \end{matrix} \begin{bmatrix} \phantom{V} \end{bmatrix}$$

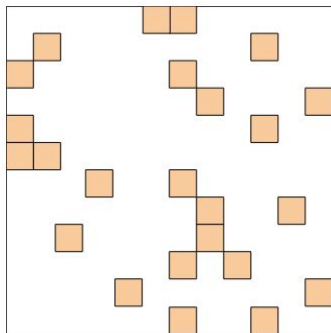
$\begin{matrix} & \text{Ich} & \text{ging} & \text{nach} & \text{Hause} \\ \text{Ich} & \text{blue} & \text{light blue} & \text{white} & \text{light blue} \\ \text{ging} & \text{light blue} & \text{dark blue} & \text{light blue} & \text{light blue} \\ \text{nach} & \text{light blue} & \text{light blue} & \text{dark blue} & \text{light blue} \\ \text{Hause} & \text{light blue} & \text{light blue} & \text{light blue} & \text{dark blue} \end{matrix}$

# Attention — Performance Considerations

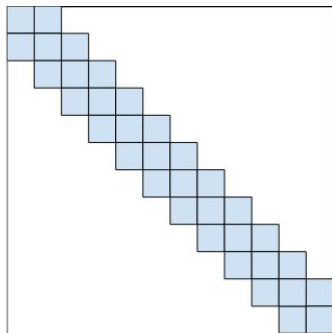
## Alternative: “restricted” attention

- Does not compute attention between all pairs of words
- Main goal: make number of operations to be in  $O(N)$

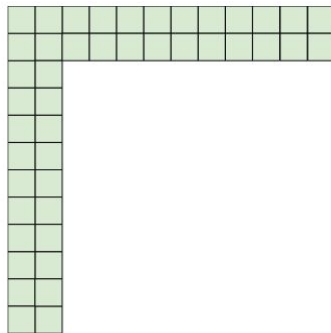
### Example:



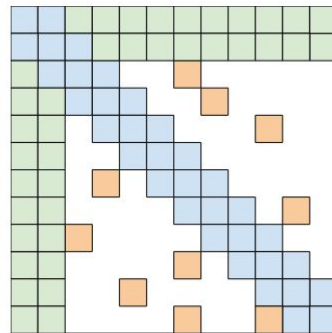
(a) Random attention



(b) Window attention



(c) Global Attention



(d) BIGBIRD



# Outline

- Contextual Word Embeddings
  - Motivation
  - ELMo
- Transformers
  - Positional Encoding
  - Core Layers
  - Encoder & Decoder
- Extended Concepts
  - Masking
  - Restricted Attention
- Transformer-based LLMs
  - **Overview**
  - Encoder-only: BERT, RoBERTa
  - Encoder-Decoder: T5, BART
  - Decoder-only: GPT, LLaMA
  - Opportunities & Challenges

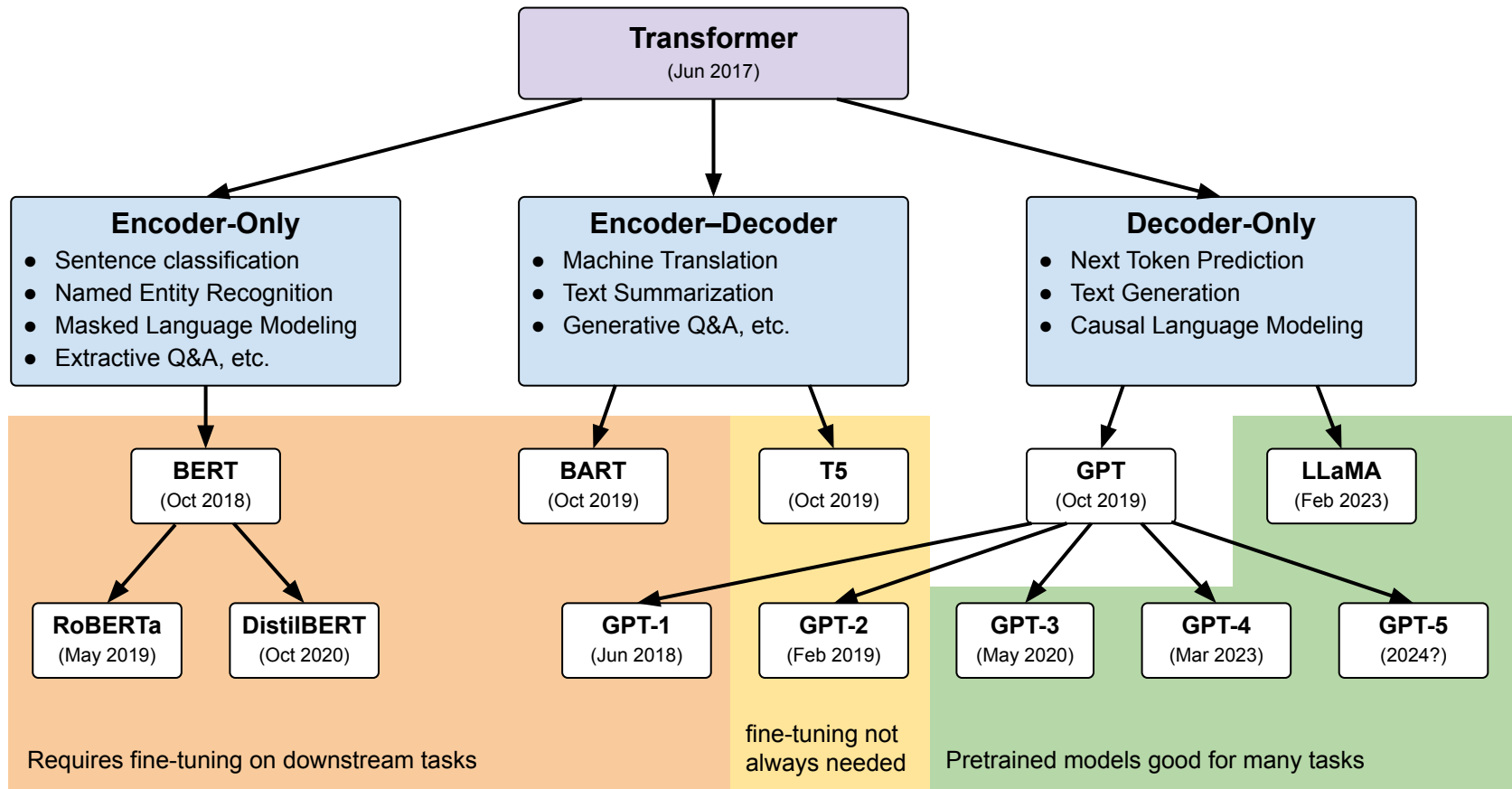


# Architecture Order: Doric Ionic Corinthian



# Break

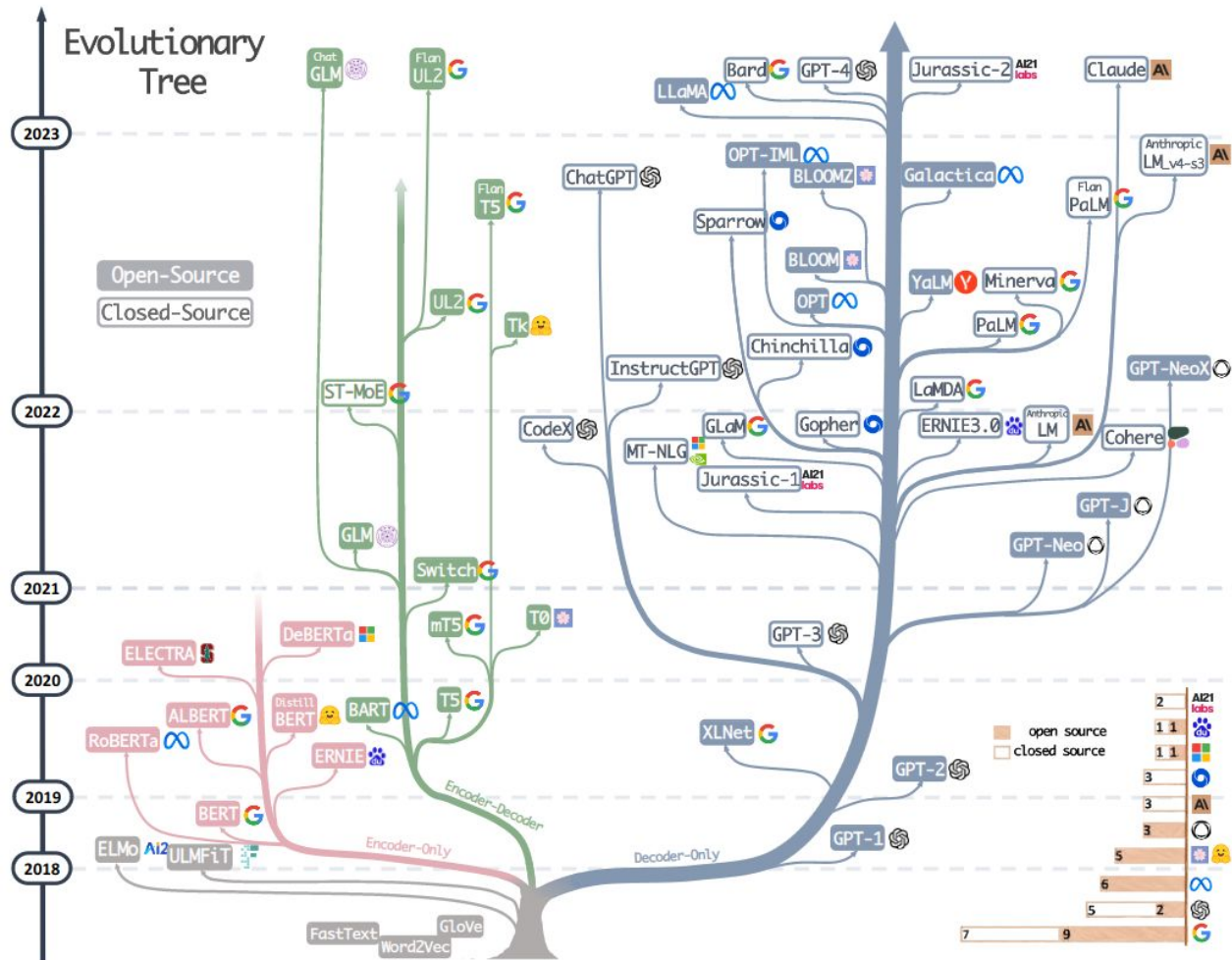
# Architectures



# The LLM Craze

Observation: **Decoder-only dominates!**

- Simpler architecture & setup
- More cheaply to train (relatively)
- More suitable for text generation
- Good zero-shot generalization



Source: [Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond \(2023\)](#)

# Outline

- Contextual Word Embeddings
  - Motivation
  - ELMo
- Transformers
  - Positional Encoding
  - Core Layers
  - Encoder & Decoder
- Extended Concepts
  - Masking
  - Restricted Attention
- Transformer-based LLMs
  - Overview
  - **Encoder-only: BERT, RoBERTa**
  - Encoder-Decoder: T5, BART
  - Decoder-only: GPT, LLaMA
  - Opportunities & Challenges

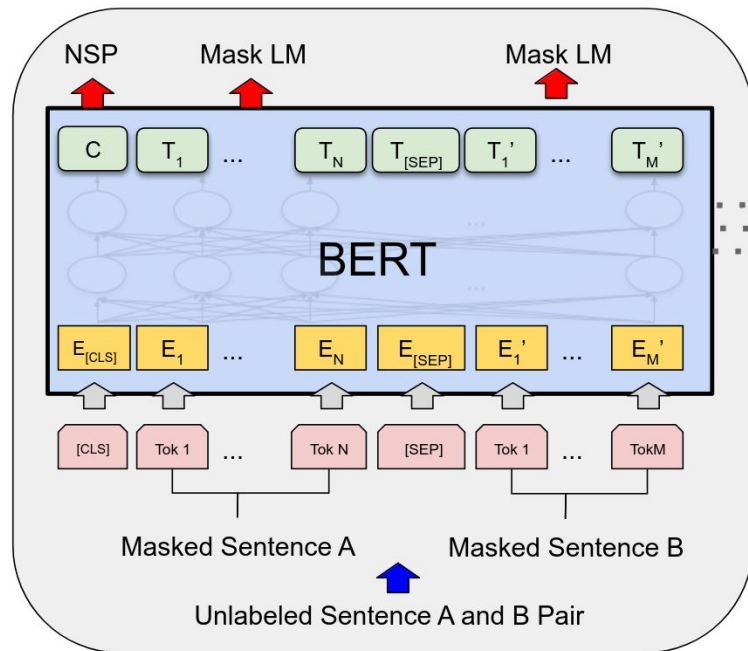
# BERT (Bidirectional Encoder Representations from Transformers)

- BERT

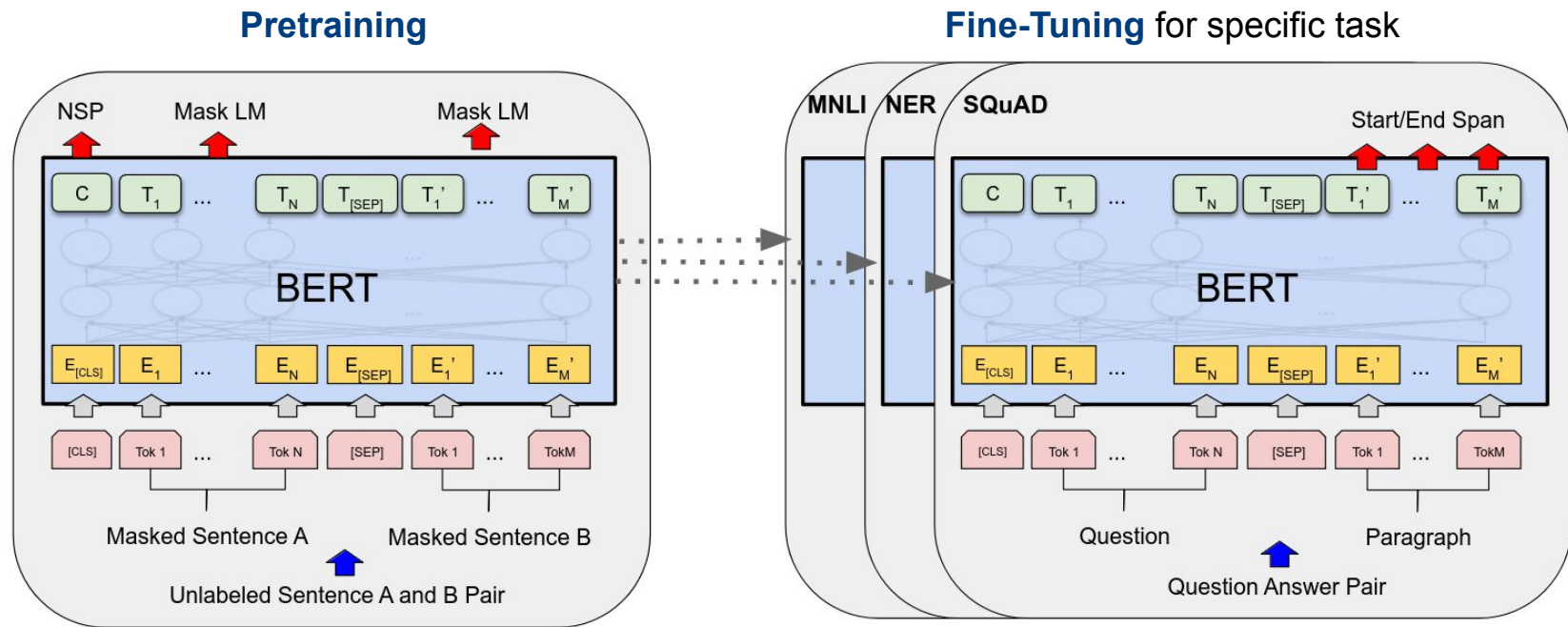
- Uses only the Transformer Encoder
- Self-supervised training

- Train on 2 learning objectives

- MLM: Masked Language Model  
(predict the masked words in input sentences)
- NSP: Next Sentence Prediction  
(predict if the second sentence was indeed followed by the first sentence)



# BERT (Bidirectional Encoder Representations from Transformers)





# RoBERTa (A Robustly Optimized Bidirectional Encoder Representations from Transformers)

- RoBERTa  $\approx$  BERT scaled up

- Same architecture, similar training setup (MLM only), but longer training, using more data
- Dynamic masking: masking done during training time  
(BERT uses “static” masking: masking done during preprocessing)

- Other BERT variants

- DistilBERT
- ALBERT

Comparison	BERT October 11, 2018	RoBERTa July 26, 2019	DistilBERT October 2, 2019	ALBERT September 26, 2019
Parameters	Base: 110M Large: 340M	Base: 125 Large: 355	Base: 66	Base: 12M Large: 18M
Layers / Hidden Dimensions / Self-Attention Heads	Base: 12 / 768 / 12 Large: 24 / 1024 / 16	Base: 12 / 768 / 12 Large: 24 / 1024 / 16	Base: 6 / 768 / 12	Base: 12 / 768 / 12 Large: 24 / 1024 / 16
Training Time	Base: 8 x V100 x 12d Large: 280 x V100 x 1d	1024 x V100 x 1 day (4-5x more than BERT)	Base: 8 x V100 x 3.5d (4 times less than BERT)	[not given] Large: 1.7x faster
Performance	Outperforming SOTA in Oct 2018	88.5 on GLUE	97% of BERT-base's performance on GLUE	89.4 on GLUE
Pre-Training Data	BooksCorpus + English Wikipedia = 16 GB	BERT + CCNews + OpenWebText + Stories = 160 GB	BooksCorpus + English Wikipedia = 16 GB	BooksCorpus + English Wikipedia = 16 GB
Method	Bidirectional Transformer, MLM & NSP	BERT without NSP, Using Dynamic Masking	BERT Distillation	BERT with reduced parameters & SOP (not NSP)

# Outline

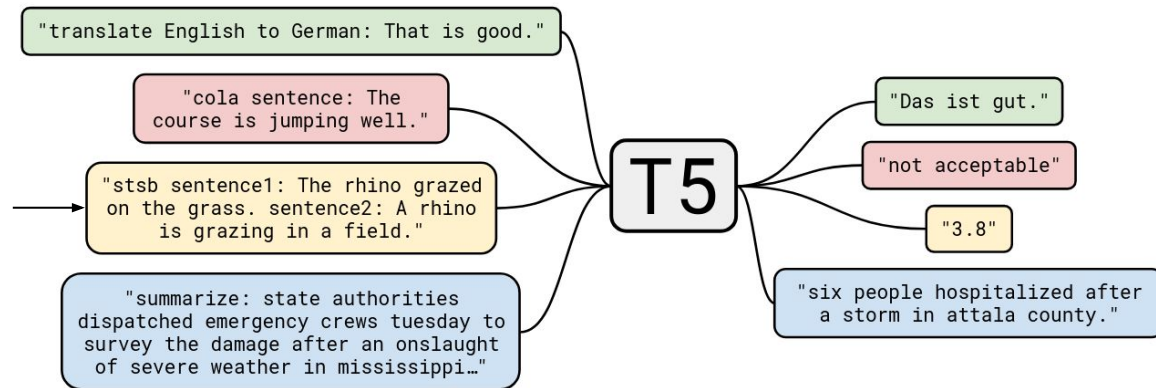
- Contextual Word Embeddings
  - Motivation
  - ELMo
- Transformers
  - Positional Encoding
  - Core Layers
  - Encoder & Decoder
- Extended Concepts
  - Masking
  - Restricted Attention
- Transformer-based LLMs
  - Overview
  - Encoder-only: BERT, RoBERTa
  - **Encoder-Decoder: T5, BART**
  - Decoder-only: GPT, LLaMA
  - Opportunities & Challenges

# T5 (Text-to-Text Transfer Transformer)

## Core Concepts

- Basic encoder–decoder Transformer architecture
- Multi-task learning: training of model on multiple tasks at the same time  
(e.g., machine translation, coreference resolution, text summarization, [sentence acceptability judgment](#), sentiment analysis)
- Each task is (re-)formulated as text-to-text task to match encoder–decoder architecture  
(including task-specific prefixes)

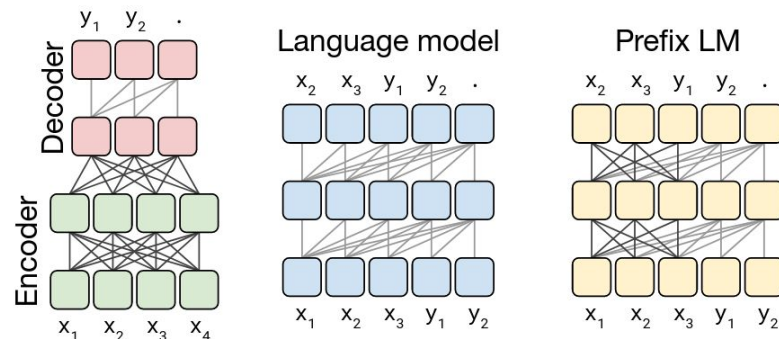
Example: [Semantic Text Similarity Benchmark](#) (STSB)  
training data sample reformulated as a text-to-text task



# T5 (Text-to-Text Transfer Transformer)

## Evaluation

- The authors evaluated the multi-task learning approach on different architectures
- Best results: encoder–decoder architecture

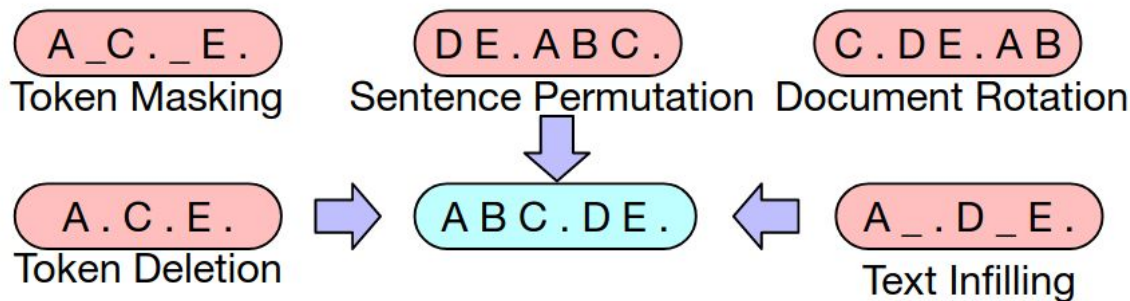


Architecture	Objective	Params	Cost	GLUE	CNN4	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	$M$	<b>83.28</b>	<b>19.24</b>	<b>80.88</b>	<b>71.36</b>	<b>26.98</b>	<b>39.82</b>	<b>27.65</b>
Enc-dec, shared	Denoising	$P$	$M$	82.81	18.78	<b>80.63</b>	<b>70.73</b>	26.72	39.03	<b>27.46</b>
Enc-dec, 6 layers	Denoising	$P$	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	$P$	$M$	74.70	17.93	61.14	55.02	25.09	35.28	25.86
Prefix LM	Denoising	$P$	$M$	81.82	18.61	78.94	68.11	26.43	37.98	27.39

# BART (Bidirectional and Auto-Regressive Transformers)

## Core Concepts

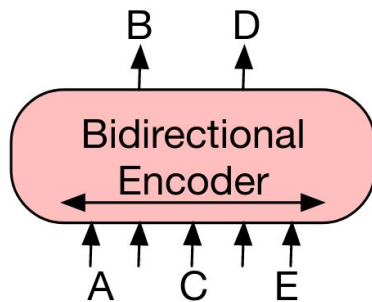
- Basic encoder–decoder Transformer architecture
- Trained by corrupting documents and then optimizing a reconstruction loss → **denoising**  
(Denoising: Minimising the cross-entropy between the decoder's output and the original document)
- Various transformation techniques to corrupt input documents



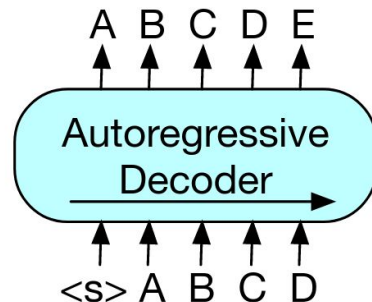
# BART $\approx$ BERT + GPT

## BERT

- Random tokens are replaced with masks (e.g., [MASK])
- Input is encoded bidirectionally (not suitable for text generation)



+

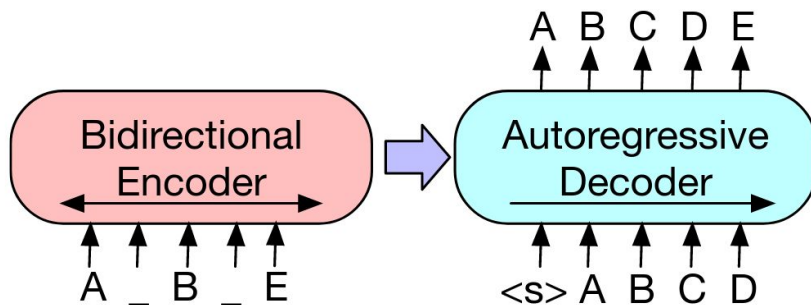


## GPT

- Auto-regressively word prediction (suitable for text generation)
- Words can only condition on leftward context (cannot learn bidirectional interactions)

## BART

- Arbitrary noise transformation (not just BERT-like masking)
- Bidirectional encoding + auto-regression word prediction



# Outline

- Contextual Word Embeddings
  - Motivation
  - ELMo
- Transformers
  - Positional Encoding
  - Core Layers
  - Encoder & Decoder
- Extended Concepts
  - Masking
  - Restricted Attention
- Transformer-based LLMs
  - Overview
  - Encoder-only: BERT, RoBERTa
  - Encoder-Decoder: T5, BART
  - **Decoder-only: GPT, LLaMA**
  - Opportunities & Challenges

# GPT (Generative Pretrained Transformer)

- GPT

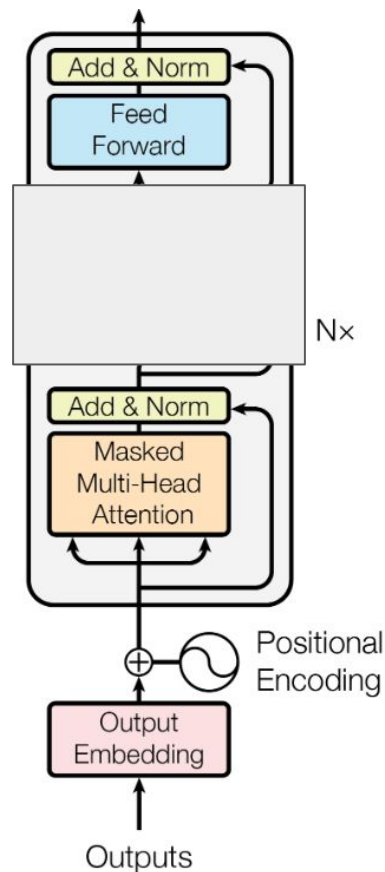
- Uses only the Transformer Decoder without the encoder attention block (alternatively: encoder with “do not look ahead” masking)
- Self-supervised training

- Learning objectives

- Auto-regressive Language Model

- (Very) oversimplified history of GPT

- GPT-1/2/3: text only, “just” making it larger; GPT-4: multimodal
- GPT-3+: **reinforcement learning from human feedback** (RLHF)





# GPT (Generative Pretrained Transformer)

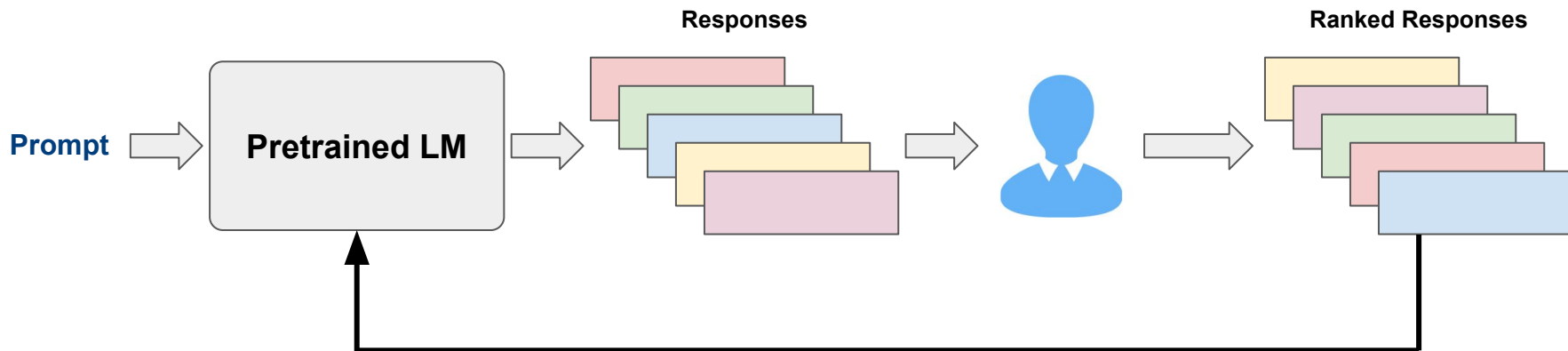
- GPT-3 models

Model Name	$n_{\text{params}}$	$n_{\text{layers}}$	$d_{\text{model}}$	$n_{\text{heads}}$	$d_{\text{head}}$	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	$6.0 \times 10^{-4}$
GPT-3 Medium	350M	24	1024	16	64	0.5M	$3.0 \times 10^{-4}$
GPT-3 Large	760M	24	1536	16	96	0.5M	$2.5 \times 10^{-4}$
GPT-3 XL	1.3B	24	2048	24	128	1M	$2.0 \times 10^{-4}$
GPT-3 2.7B	2.7B	32	2560	32	80	1M	$1.6 \times 10^{-4}$
GPT-3 6.7B	6.7B	32	4096	32	128	2M	$1.2 \times 10^{-4}$
GPT-3 13B	13.0B	40	5140	40	128	2M	$1.0 \times 10^{-4}$
GPT-3 175B or “GPT-3”	175.0B	96	12288	96	128	3.2M	$0.6 \times 10^{-4}$

# GPT — RLHF (Reinforcement Learning from Human Feedback)

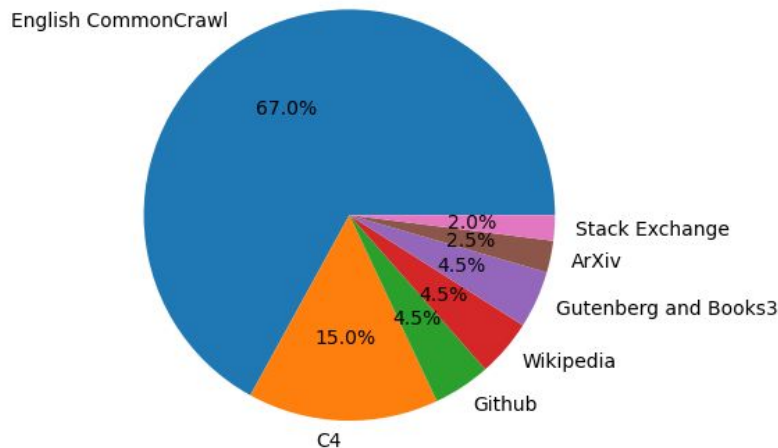
- RLHF — two common setups

- Use human-generated responses to prompts to fine-tune the pretrained model
- Generate multiple response for same prompt; human ranks response; use ranking for fine-tuning



# LLaMA (Large Language Model Meta AI)

- Decoder-only architecture very similar to GPT (any many others!) — main tweaks
  - Pre-normalization: layer normalization is put **inside** the residual blocks
  - SwiGLU (Swish-Gated Linear Unit) activation: non-monotonic, parameterized activation function
  - Rotary Positional Embeddings: encode word positions by rotating word embedding vectors
- Open LLM
  - Trained exclusively on publicly available data

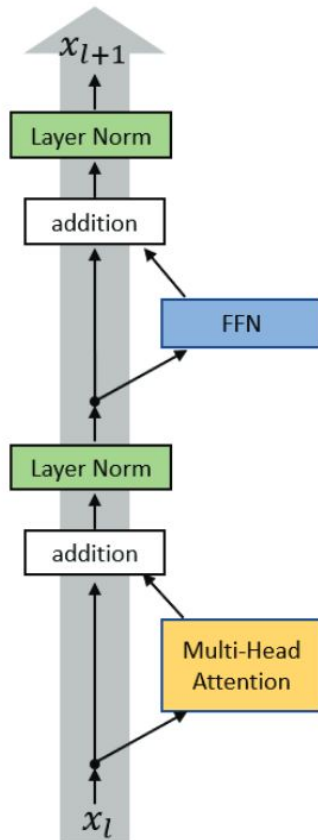


# LLaMA — Pre-Normalization

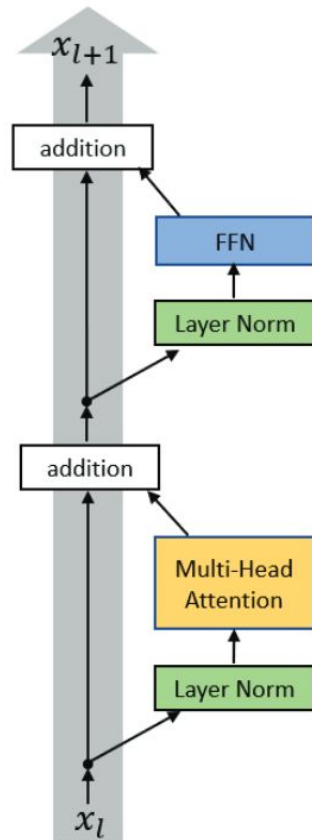
- Post vs. pre-normalization

- Post: layer normalization **between** residual blocks (original transformer)
- Pre: layer normalization **inside** residual blocks (LLaMA, etc.)
- Observed benefit of pre-normalization:
  - Well-behaved gradients at initialization
  - Significantly faster training

Original Transformer



LLaMA



# LLaMA — SwiGLU (Swish-Gated Linear Unit)

## GLU – Gated Linear Unit ([paper](#))

- Gating proposed in LSTM [paper](#) (1997!)
- Parameterized activation function
- Many other variants proposed

$$GLU(x) = (xW + b) \otimes \sigma(xV + c)$$

## Swish ([paper](#))

- Simple parameterized activation function
- Approach: "try and see what works best"

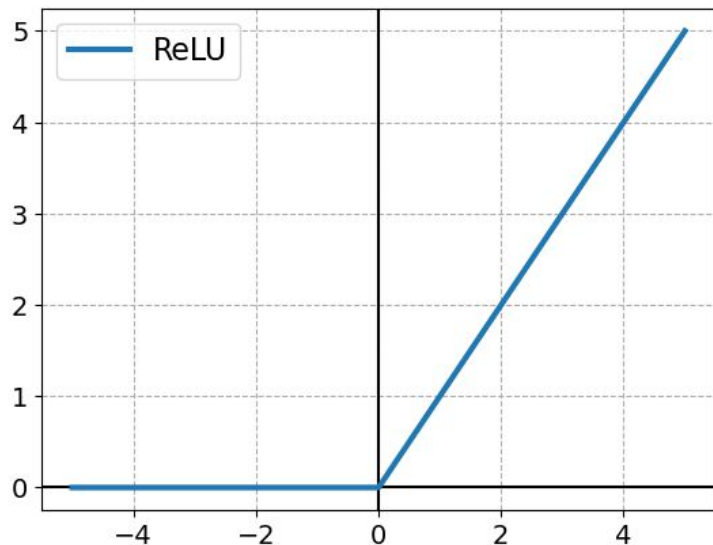
$$Swish(x) = x \otimes \sigma(\beta x)$$



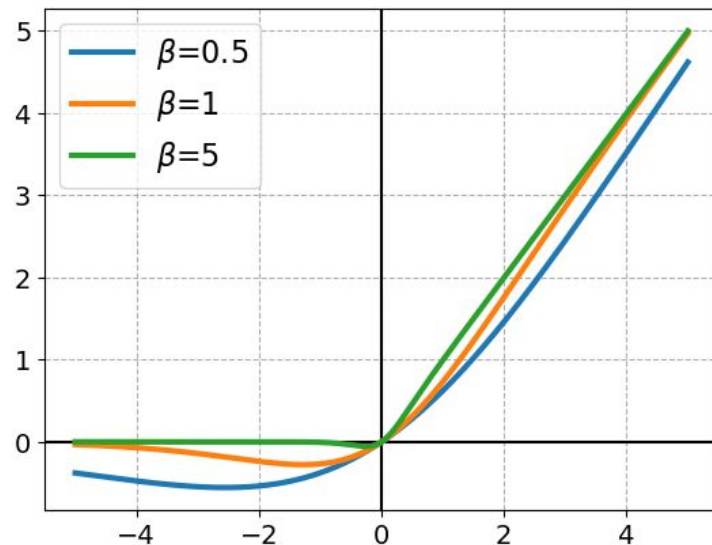
$$SwiGLU(x) = (xW + b) \otimes Swish_{\beta}(xV + c)$$

# LLaMA — SwiGLU (Swish-Gated Linear Unit)

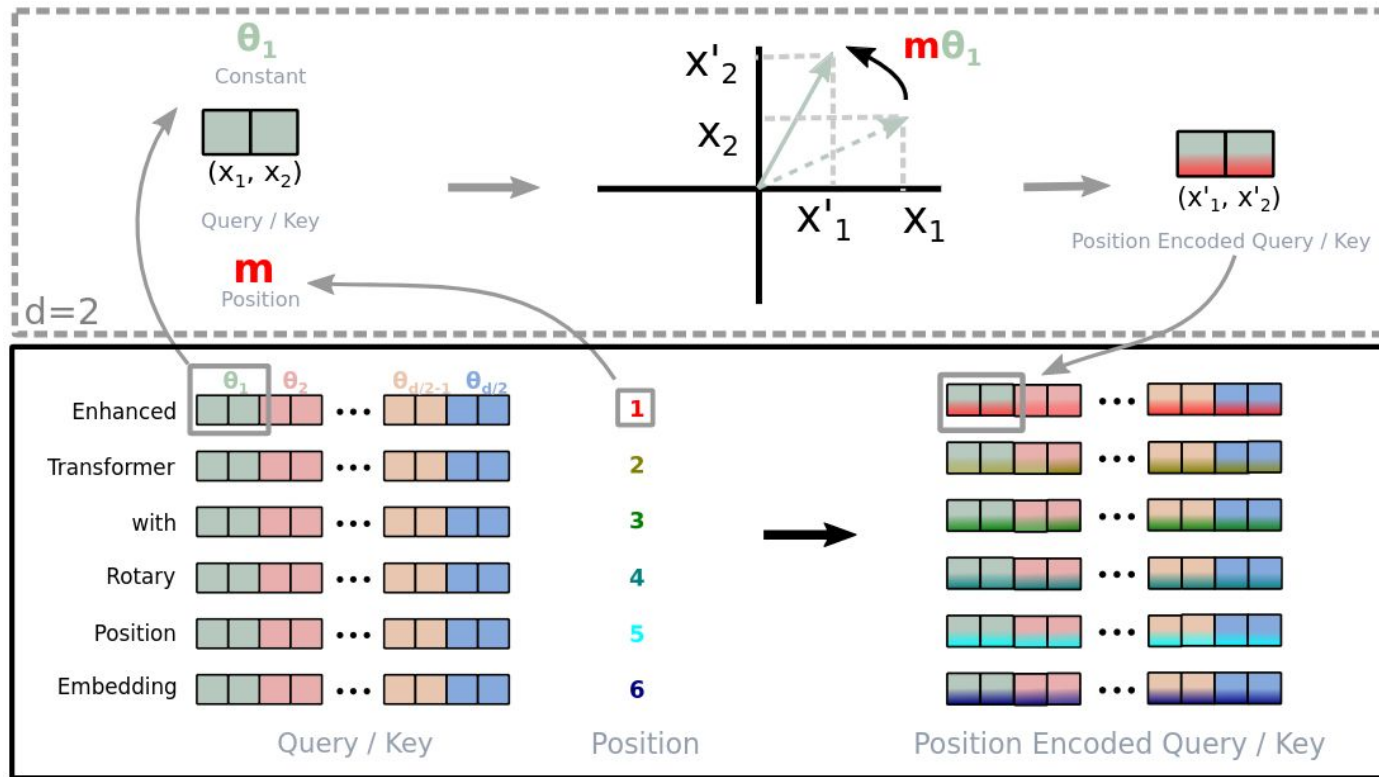
ReLU (Linear Rectified Unit)



Swish



# LLaMA — Rotary Positional Embeddings (RoPE)



# Outline

- Contextual Word Embeddings
  - Motivation
  - ELMo
- Transformers
  - Positional Encoding
  - Core Layers
  - Encoder & Decoder
- Extended Concepts
  - Masking
  - Restricted Attention
- Transformer-based LLMs
  - Overview
  - Encoder-only: BERT, RoBERTa
  - Encoder-Decoder: T5, BART
  - Decoder-only: GPT, LLaMA
  - **Opportunities & Challenges**



# The Future of Large Language Models — Opportunities

Language models are an old idea — What changed?

- New architectures (here: Transformers)
- More computing power
- More and diverse data
- More resources (i.e., money, manpower)



→ Exploding size/scale of models



Size of models has crossed  
some kind of threshold

→ **LLMs show Emergent Abilities**



Abilities that were not explicitly programmed into  
the model but emerge from the training process

# The Future of Large Language Models — Opportunities

## Emergent abilities

- **Language Generation** (coherent and fluent text in a variety of styles and genres, from news articles to poetry)
- **Question Answering** (answering complex questions by extracting information from large amounts of text data)
- **Translation** (translating text between different languages with high accuracy)
- **Summarization** (generate concise summaries of long documents, allowing for efficient information extraction and consumption)
- **Dialogue Generation** (engage in natural and coherent conversations with humans)
- **Common Sense Reasoning** (basic degree of common sense reasoning; predicting outcome of simple scenarios)

➔ **Question: Can a language model really do these tasks?**

# The Future of Large Language Models — Challenges

**EXPLAINER: What is ChatGPT and why are schools blocking it?**

## ChatGPT

The impact of Large Language Models on Law Enforcement

Will ChatGPT take my job? Here are 20 professions that could be replaced by AI

**Criminals will soon use ChatGPT to make scams more convincing, experts warn; only 'a matter of time' before S'pore hit**

**Hallucinations, Plagiarism, and ChatGPT**

**ChatGPT Poses Dangers for Online Dating Apps**

**Letters | How universities can start to grapple with ChatGPT's capabilities**

**Cybercriminals are using ChatGPT to create malware**

**Hollywood: Writers Guild considering ChatGPT-written scripts, no AI credits**

A fake news frenzy: why ChatGPT could be disastrous for truth in journalism

**Pause Giant AI Experiments: An Open Letter**

# The Future of Large Language Models — Challenges

ChatGPT invented a sexual harassment scandal and named a real law prof as the accused

**1,100+ notable signatories just signed an open letter asking 'all AI labs to immediately pause for at least 6 months'**

**Italy orders ChatGPT blocked citing data protection concerns**

AI can be racist, sexist and creepy. What should we do about it?

**GPT-4 kicks AI security risks into higher gear**

**Europol sounds alarm as crooks tap into ChatGPT-4**

**GPT-5 expected this year, could make ChatGPT indistinguishable from a human**

What Have Humans Just Unleashed?

Call it tech's optical-illusion era: Not even the experts know exactly what will come next in the AI revolution.

**Experts Warn of Nightmare Internet Filling With Infinite AI-Generated Propaganda**

**Did a Robot Write This? We Need Watermarks to Spot AI**

# The Future of Large Language Models — Challenges

Exclusive: OpenAI Used Kenyan Workers on Less Than \$2 Per Hour to Make ChatGPT Less Toxic

**Australian Mayor Threatens to Sue OpenAI for Defamation by Chatbot**

**Artists sue AI company for billions, alleging "parasite" app used their work for free**

**ChatGPT banned on Q&A site over 'substantially harmful' answers**

\$120bn wiped off Google after Bard AI chatbot gives wrong answer

**Chat-GPT Pretended to Be Blind and Tricked a Human Into Solving a CAPTCHA**

**Microsoft tries to justify A.I.'s tendency to give wrong answers by saying they're 'usefully wrong'**

**ChatGPT lies about scientific results, needs open-source alternatives, say researchers**

**AI isn't close to becoming sentient – the real danger lies in how easily we're prone to anthropomorphize it**

# The Future of Large Language Models — Challenges

...and the biggest questions: **Why** does this all seem to work?

We have extended the GLU family of layers and proposed their use in Transformer. In a transfer-learning setup, the new variants seem to produce better perplexities for the de-noising objective used in pre-training, as well as better results on many downstream language-understanding tasks. These architectures are simple to implement, and have no apparent computational drawbacks. [We offer no explanation as to why these architectures seem to work; we attribute their success, as all else, to divine benevolence. ]

# Summary

- Transformer architecture

- Encoder-decoder architecture
- Core concept: attention (self-attention + cross attention)
- Additional concepts: positional encoding, masking

- Large Language Models (LLMs)

- Currently dominated by transformer architecture
- Main categorization: encoder-only, encoder-decoder, decoder-only (with decoder-only models right now dominating the field)
- Still continuously growing model zoo of LLMs

→ Last lecture: LLMs – problems, challenges, strategies

# Pre-Lecture Activity for Next Week

- Assigned Task

- Do a web search and answer the question stated below
- Post you answer(s) to the question into the Discussion on Canvas (please cite or quote your sources)

*“What is the relationship between information retrieval  
and natural language processing?”*

**Side notes:**

- This task is meant as a warm-up to provide some context for the next lecture
- No worries if you get lost; we will talk about this in the next lecture
- You can just copy-&-paste others' answers but this won't help you learn better



# Outlook for Next Week: Classification Applications

fake news

olivetti STUDIO 46

MYRIAM REDONDO  
**VERIFICACIÓN  
DIGITAL**  
PARA PERIODISTAS  
MANUAL CONTRA BULOS Y  
DESINFORMACIÓN INTERNACIONAL



EDITORIAL MOC

**FAKE  
NEWS**  
LA VERDAD DE  
LAS NOTICIAS  
Jordi Garcia  
JORDI EVOLE

Photo credit: [Franganillo@Unsplash](https://unsplash.com/photos/8G3v3v3v3v3)