NUS | Computing
National University of Singapore

CS4248: Natural Language Processing

Lecture 10 — Transformers & LLMs

Course Logistics

- Assignments
  - Submission deadline for A3: Tue, Apr 2, 11.59 pm

- Project
  - Grades and comments for Intermediate Update posted
  - Optional consultation session – you can register here
  - Submission deadline: Thu, Apr 18, 11:59 pm
  - Considering participating in STePS

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

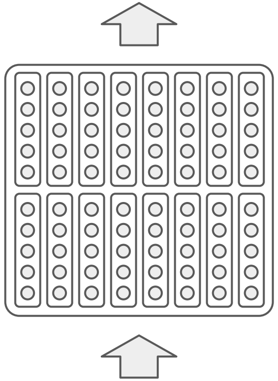
Supervised Training (RNN)

self-supervised

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”.[...]

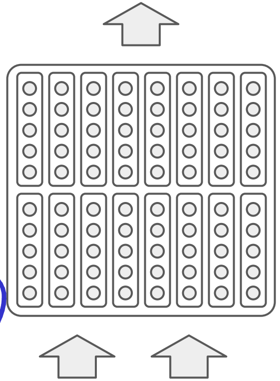


“some” encoder network

[...] 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 Systems

within a cloud

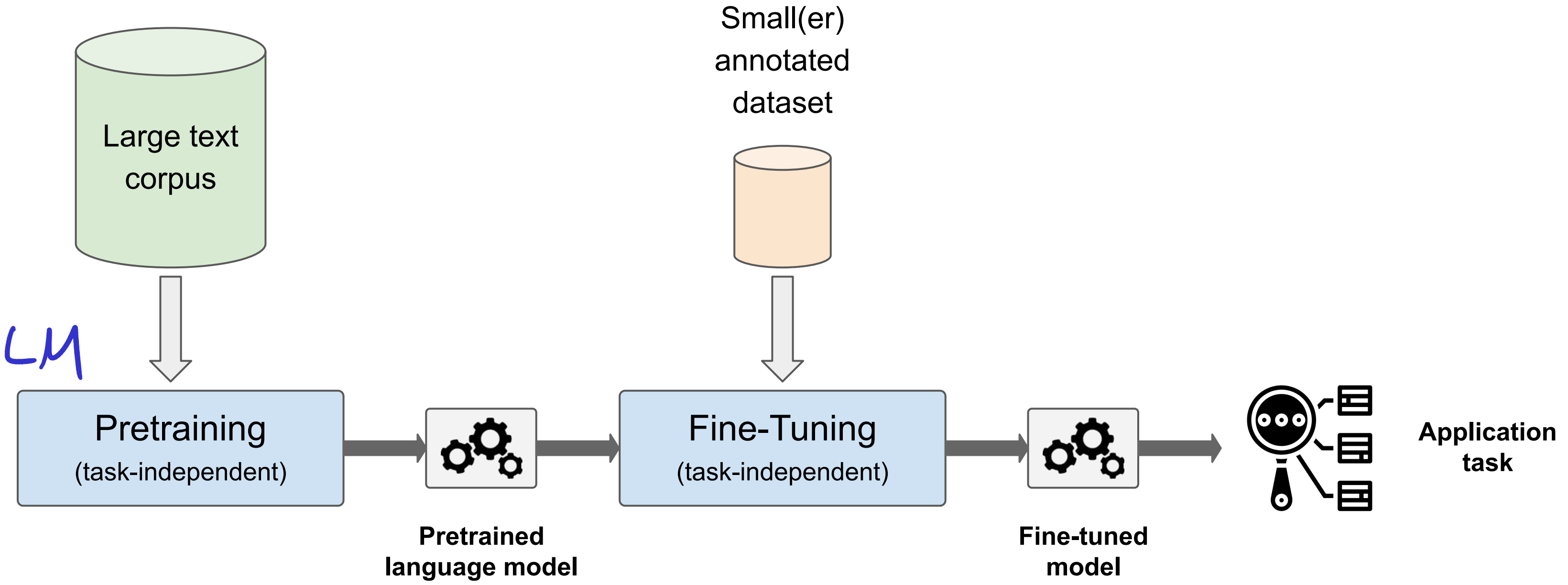


“some” encoder network

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

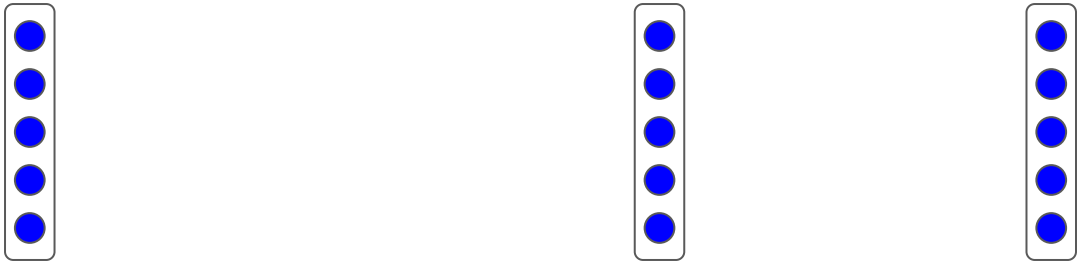


Transfer Learning with Word2Vec (or GloVe)

- 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

adj noun verb

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

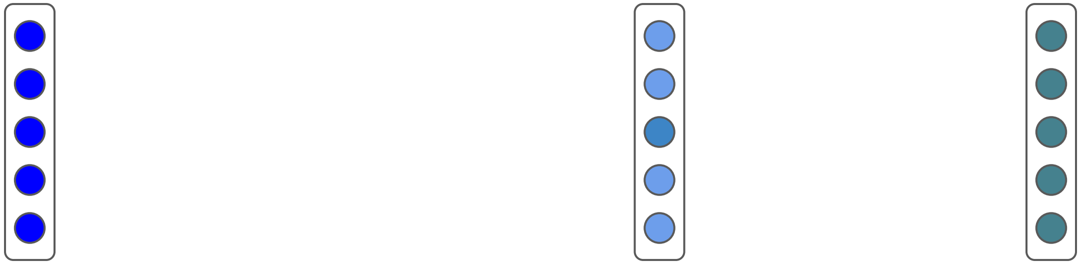


Problem: Same word vector for all occurrences of *“light”*!

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

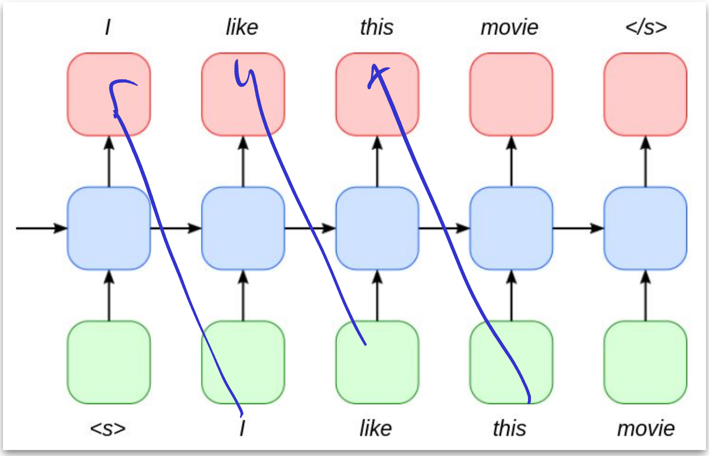
ELMo — Embeddings from Language Model

$h_t = f(h_{t-1}, x_t)$

- 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)

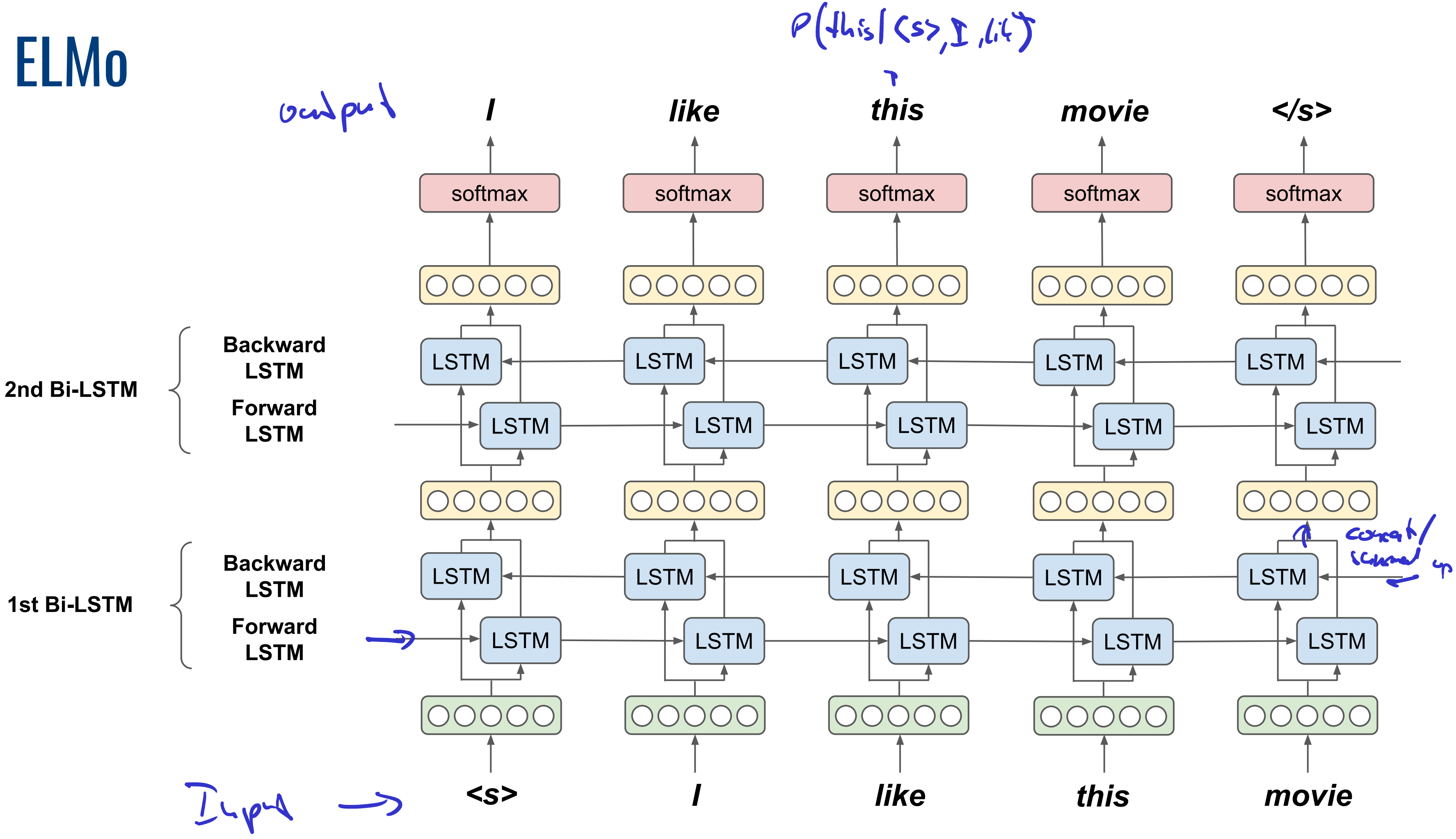
$P(\text{"this"} \mid \langle s \rangle, I, \text{like})$

Recall: Vanilla RNN Language Model



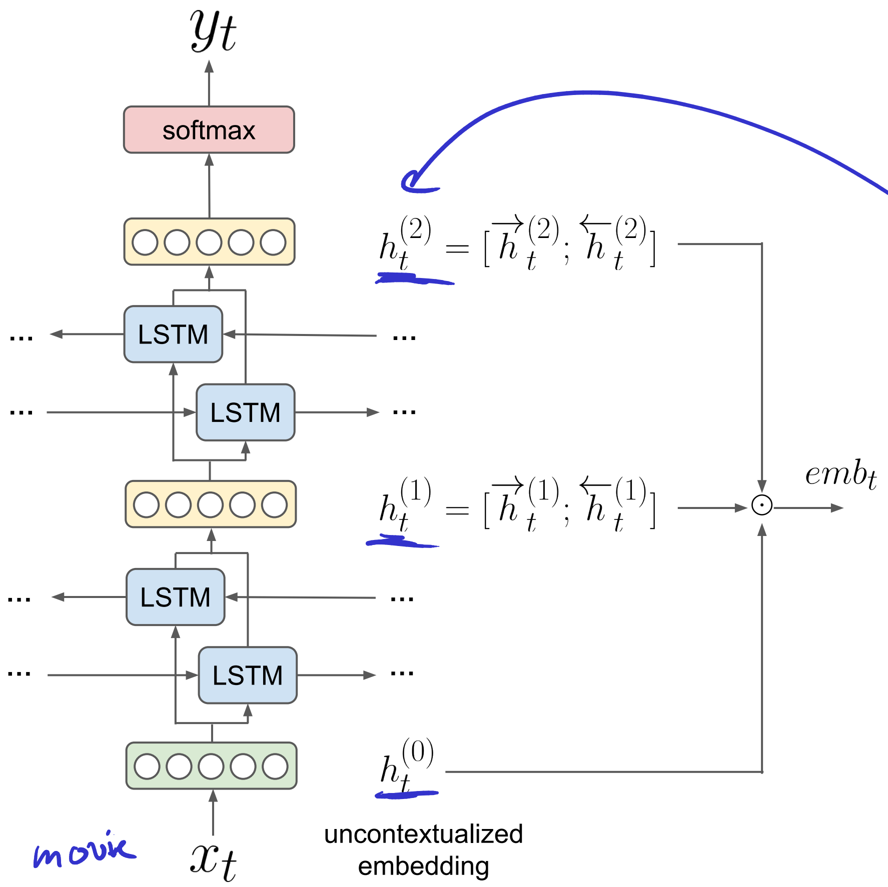
Source: [Deep Contextualized Word Representations]()

ELMo



P(this | <s>, I, like)
output
I
like
this
movie
</s>
softmax
softmax
softmax
softmax
softmax
2nd Bi-LSTM
Backward LSTM
Forward LSTM
LSTM
1st Bi-LSTM
Backward LSTM
Forward LSTM
concat / summed up
Input
<s>
I
like
this
movie

ELMo — Final Embeddings



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

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

Generalized approach: weighted sum

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

scaling factor

normalized weight

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 ± 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%

Source: [Deep Contextualized Word Representations]

ELMo — Evaluation

- Qualitative understanding what ELMo learns

≈ Word2Vec

	Source	Nearest Neighbors
GloVe	play	playing, game, games, played, players, plays, player, Play, football, multiplayer
biLM	Chico Ruiz made a spectacular play 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 play . → [illegible]
	Olivia De Havilland signed to do a Broadway play for Garson {...}	{...} they were actors who had been handed fat roles in a successful play , and had talent enough to fill the roles competently , with nice understatement .

Source: [Deep Contextualized Word Representations]

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

RNN — Problem: (Very) Long Sequences

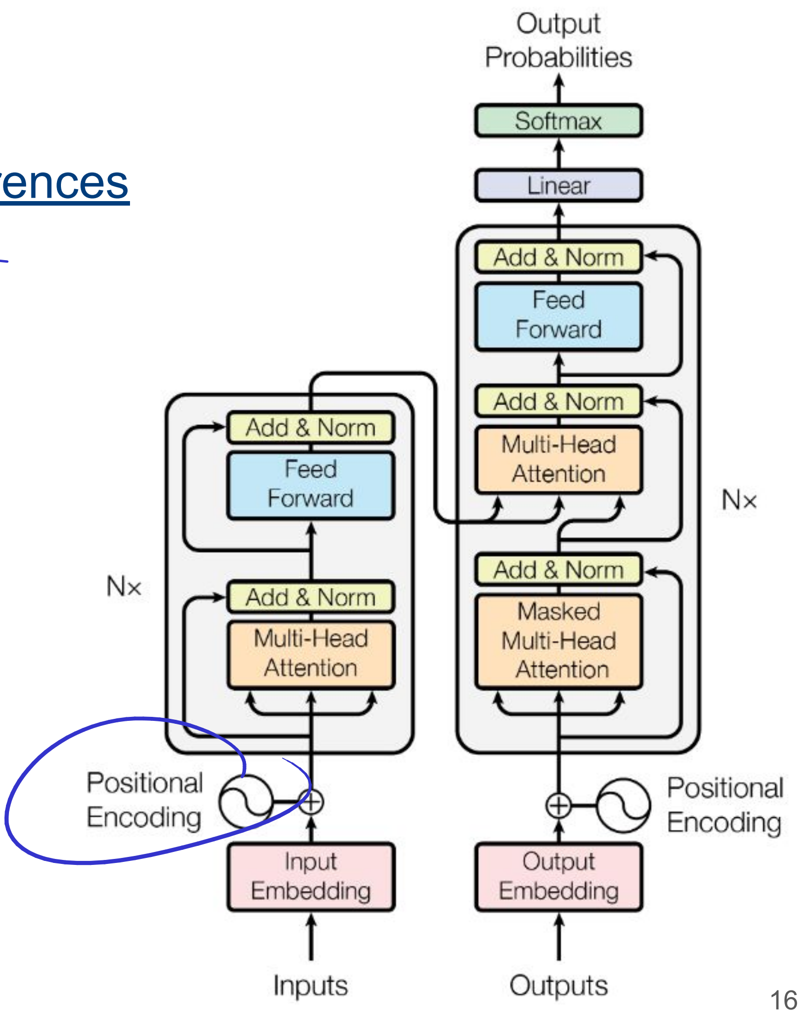
- Training
  - **Vanishing & Exploding Gradients** problem (not detailed here)
- Information capture
  - Hidden state h_t must capture all information from $h_0, h_1, ..., h_{t-1}$
  - Information dilutes over time ➜ **bottleneck**
- Performance
  - Processing is intrinsically sequential ➜ **no parallelization**
  - 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



Source: Attention is all You Need

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

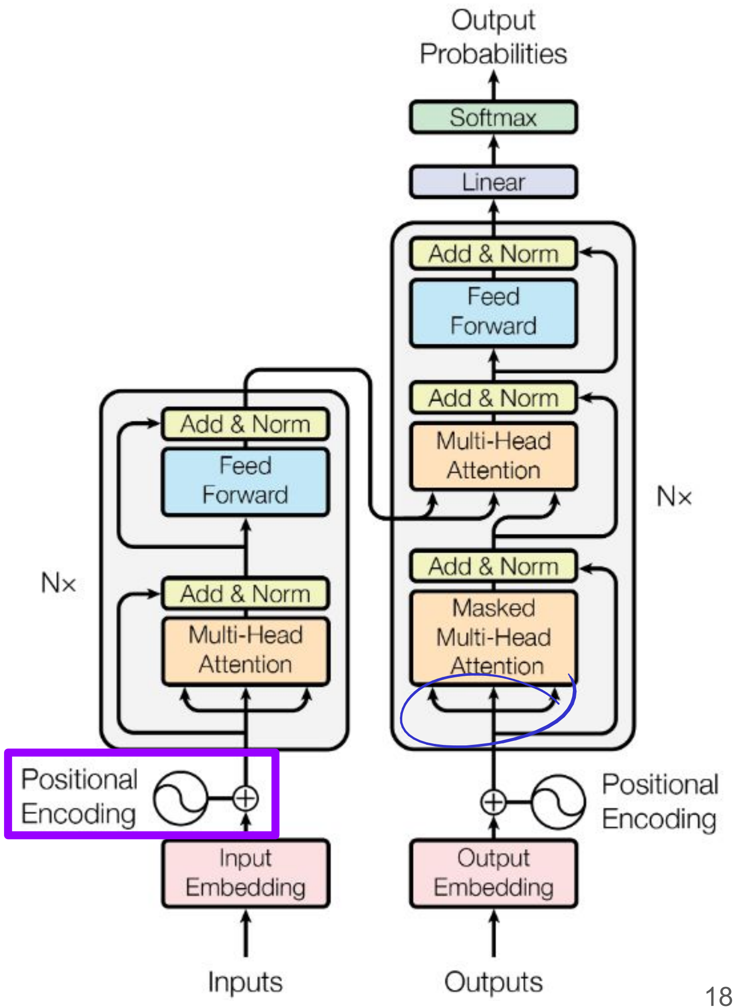
Positional Encodings

- 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

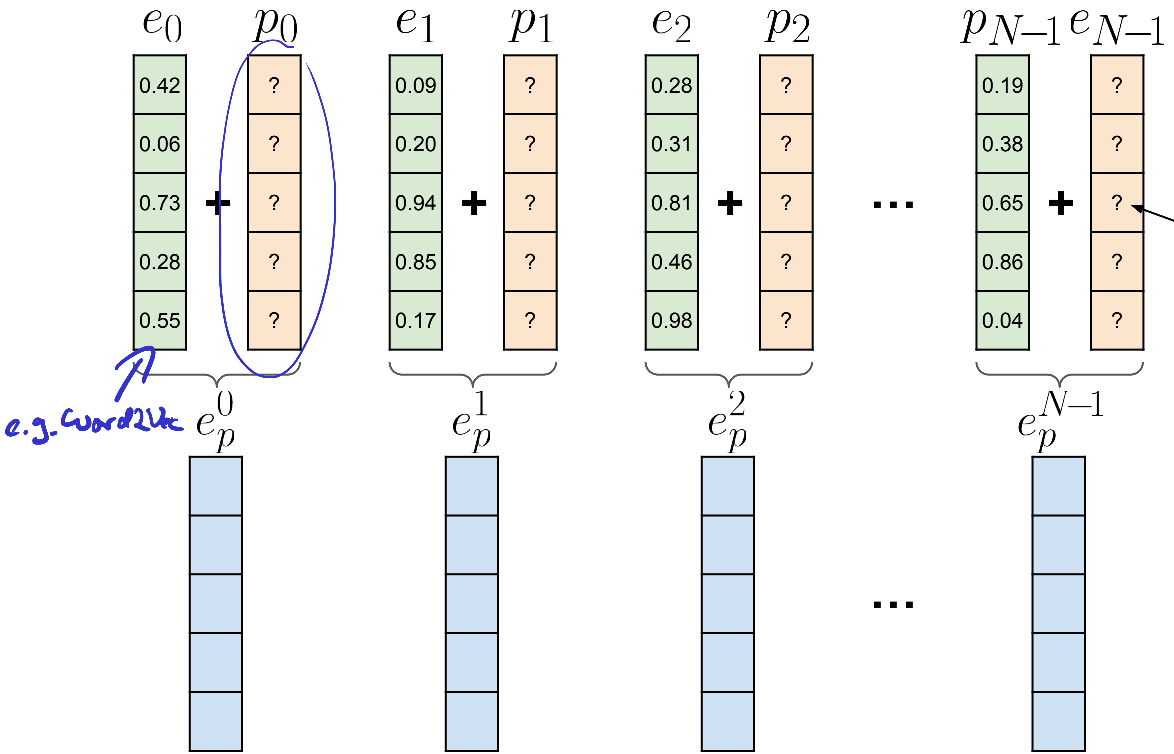
Can we somehow encode the position of words?

(as part of preprocessing the input for the transformer)



In-Lecture Activity (5 mins) — Positional Encodings

- Basic idea: Add "some" position embeddings p to initial word embeddings e



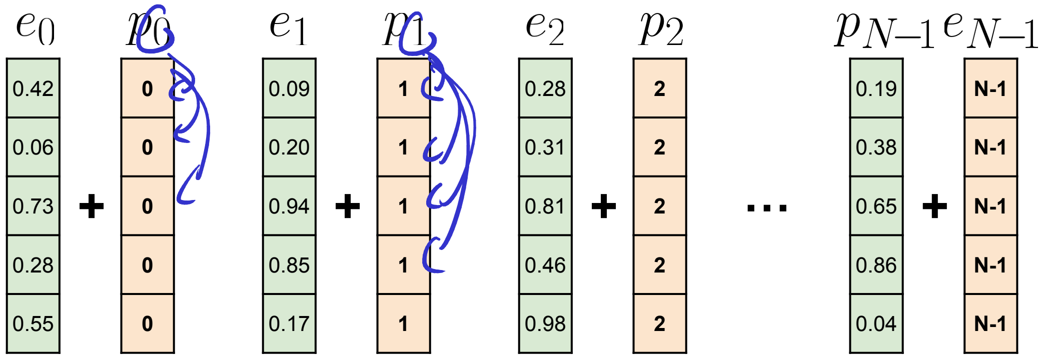
Questions:

- What are important requirements for "good" position embeddings?
- How could we compute them?

Post your solutions to Canvas > Discussions (individually or as a group; include all group members' names in the post)

Positional Encodings — Naive Approach 1

- Set position embedding values to actual position

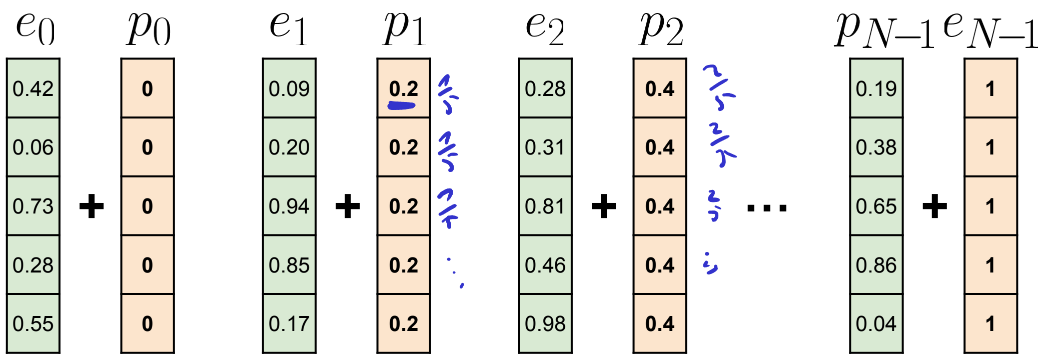


➜ **Problem:** positional encodings quickly start "dominating" word embeddings

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

Positional Encodings — Naive Approach 2

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



Example values for $N = \cancel{6}\ 5$

➜ **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

No N

$$\begin{cases} PE_{(pos,2i)} = \sin\left(\dfrac{pos}{10000^{2i/d_{model}}}\right) \\ PE_{(pos,2i+1)} = \cos\left(\dfrac{pos}{10000^{2i/d_{model}}}\right) \end{cases}$$

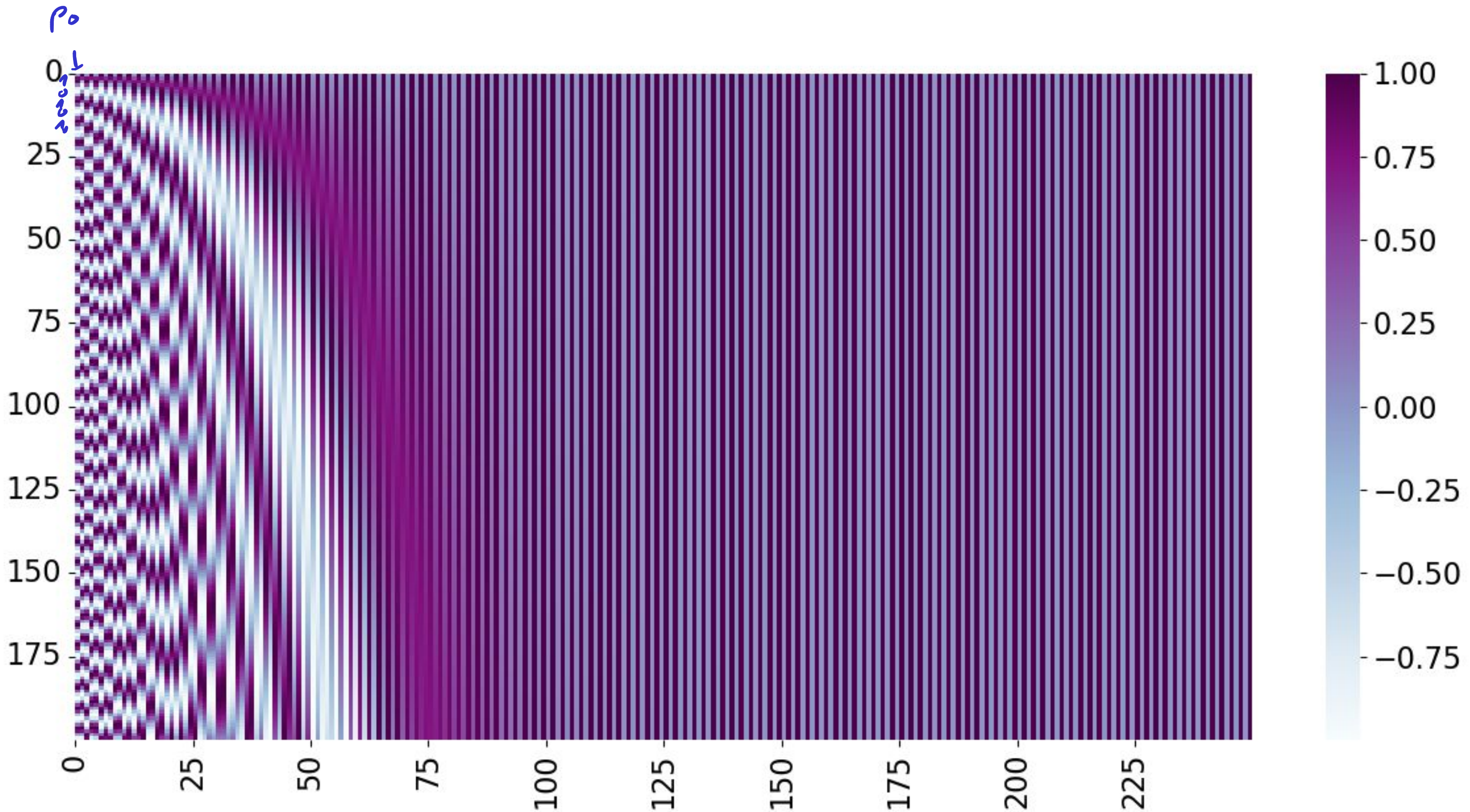
pos

	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 or of interval [-1, 1]
- Position encoding independent from N

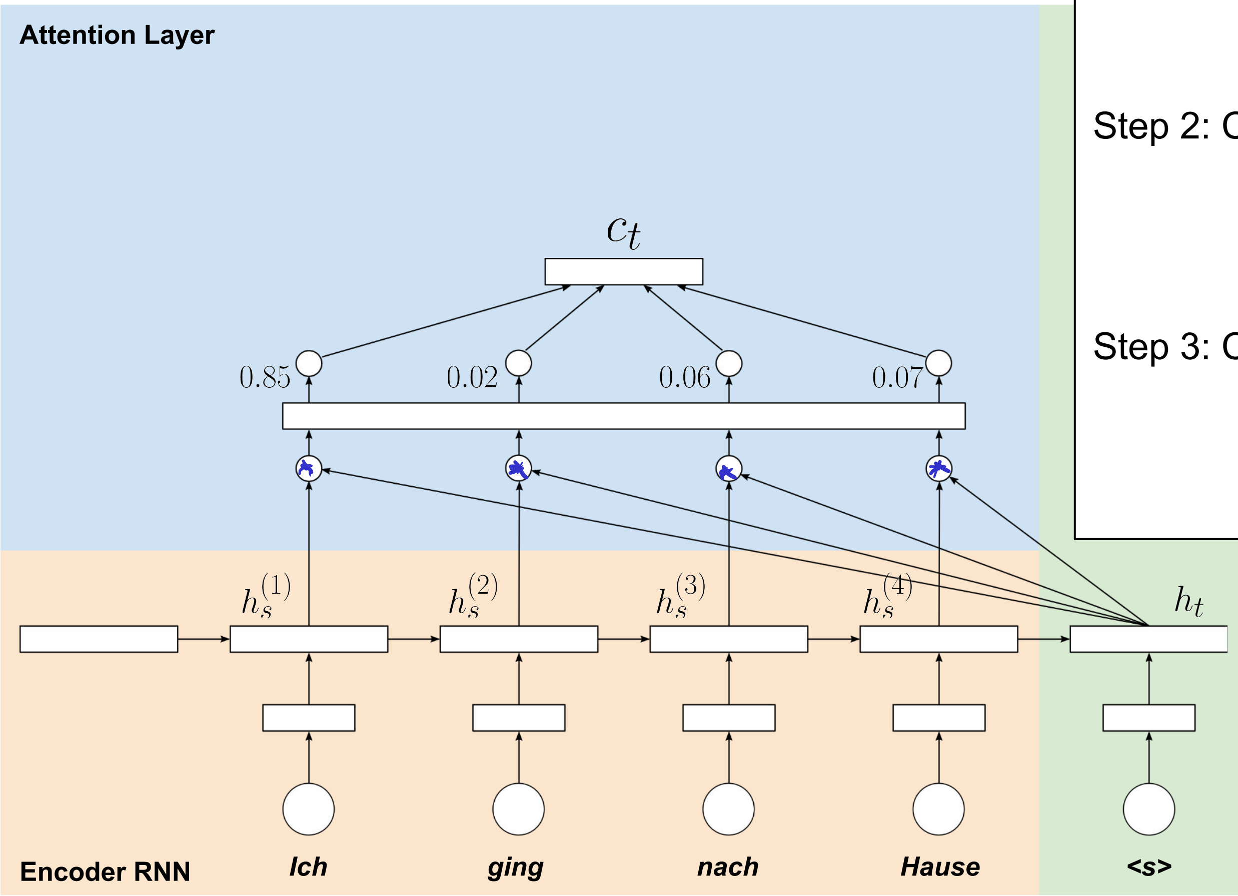
Positional Encodings — Visualized



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)



Step 1: Calculation of **Attention Scores**

$$e_i = score\left(h_t, h_s^{(i)}\right) = \begin{cases} h_t^{\mathrm{T}} h_s^{(i)} \\ \dots \text{dot prod} \ast \end{cases}$$

Step 2: Calculation of **Attention Weights**

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

Step 3: Calculation of **Context Vector**

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

$\sum a_i = 1$

RNN Attention (revisited)

hidden state of decoder

hidden states of encoder

$$\underbrace{\mathrm{softmax}\left(\frac{\text{went}\ (\,[\ h_t\ ]\,)\times\left(\begin{array}{cccc}\text{Ich}&\text{ging}&\text{nach}&\text{Hause}\\ \hline h_s^{(1)}&h_s^{(2)}&h_s^{(3)}&h_s^{(4)}\end{array}\right)}{1}\right)}_{(a_1\ a_2\ a_3\ a_4)}\times\begin{array}{r|c|}\text{Ich}&h_s^{(1)}\\ \text{ging}&h_s^{(2)}\\ \text{nach}&h_s^{(3)}\\ \text{Hause}&h_s^{(4)}\end{array} = (\,[\ \ c_t\ \ ]\,)$$

context vector

attention weights

Size of
hid dec
state

Attention — Generalized Definition

$$\mathrm{softmax}\left(\frac{\text{went}\left(\overset{Q}{[\;h_t\;]}\right)\times\left(\begin{array}{cccc} \text{Ich} & \text{ging} & \text{nach} & \text{Hause} \\ \overset{K}{[h_s^{(1)}} & h_s^{(2)} & h_s^{(3)} & h_s^{(4)}] \end{array}\right)}{1}\right)\times\begin{array}{r} \text{Ich} \\ \text{ging} \\ \text{nach} \\ \text{Hause} \end{array}\overset{V}{\left(\begin{array}{c} [\;h_s^{(1)}\;] \\ [\;h_s^{(2)}\;] \\ [\;h_s^{(3)}\;] \\ [\;h_s^{(4)}\;] \end{array}\right)} = ([\quad c_t \quad])$$

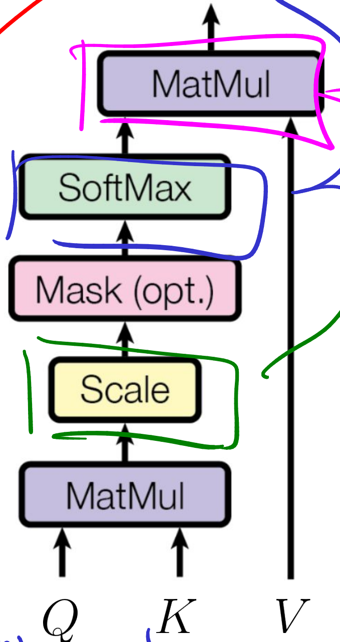
$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

Scaled Dot-Product Attention

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

Scaled Dot-Product Attention

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



RNN : current decoder h_t

all encoder h_t

```
import torch
import torch.nn as nn


class Attention(nn.Module):
    ### Implements Scaled Dot Product Attention

    def __init__(self):
        super().__init__()

    def forward(self, Q, K, V, mask=None, dropout=None):
        # All inputshapes: (batch_size B, seq_len L, model_size D)

        # Perform Q*K^T (* is the dot product here)
        # We have to use torch.matmul since we work with batches!
        out = torch.matmul(Q, K.transpose(1, 2)) # => shape: (B, L, L)

        # scale alignment scores
        out = out / (Q.shape[-1] ** 0.5)

        # Push throught softmax layer
        out = f.softmax(out, dim=-1)

        # Multiply scaled alignment scores with values V
        return torch.matmul(out, V)
```

masking

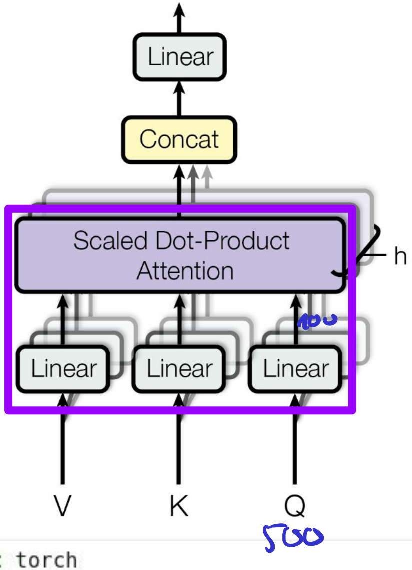
Source: Attention is all You Need

Attention Head

size of [illegible]
"hidden state" of trans

- 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



5 heads

500 100

```python
import torch
import torch.nn as nn


class AttentionHead(nn.Module):

    def __init__(self, model_size, qkv_size):
        super().__init__()
        self.Wq = nn.Linear(model_size, qkv_size)
        self.Wk = nn.Linear(model_size, qkv_size)
        self.Wv = nn.Linear(model_size, qkv_size)
        self.attention = Attention()

    def forward(self, queries, keys, values):
        # Computes scaled dot-product attention
        return self.attention(self.Wq(queries),
                              self.Wk(keys),
                              self.Wv(values))
```

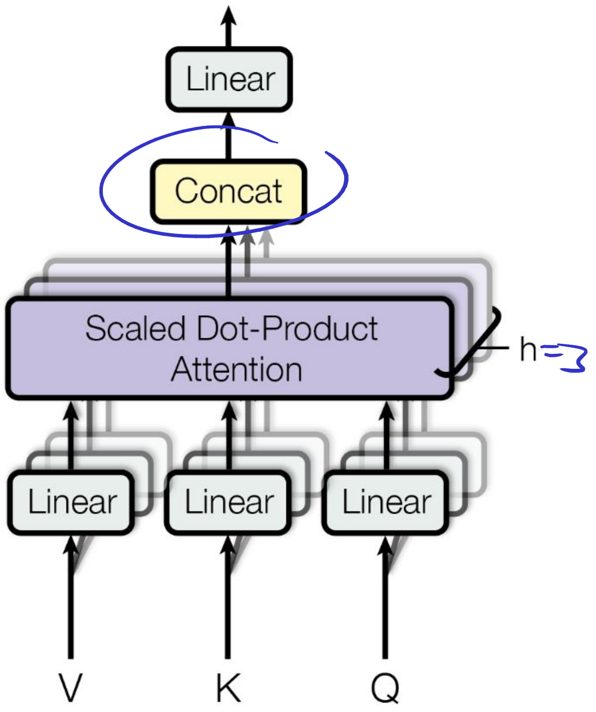
Quick Quiz: What do you think is the reason for dividing by the number of heads?

Outline

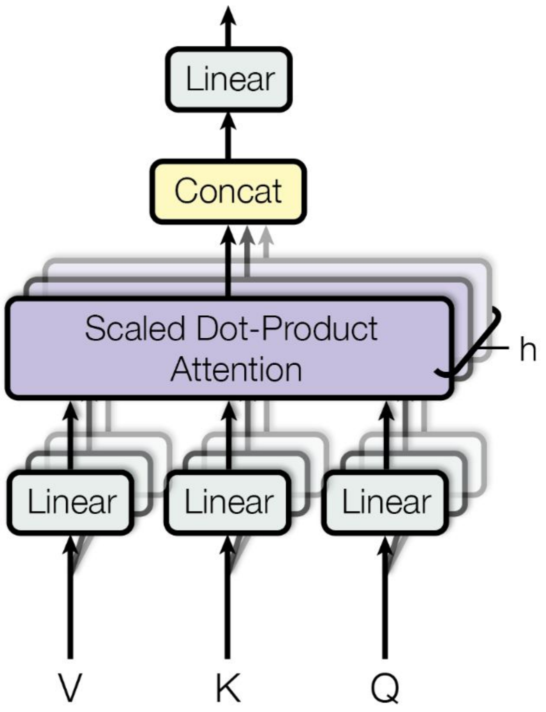
- Contextual Word Embeddings
  - Motivation
  - ELMo
- **Transformers**
  - Positional Encoding
  - **Core Layers: Multi-Head Attention**
  - Encoder & Decoder
- Extended Concepts
  - Masking
  - Restricted Attention
- Transformer-based LLMs
  - Overview
  - Encoder-only: BERT, RoBERTa
  - Encoder-Decoder: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

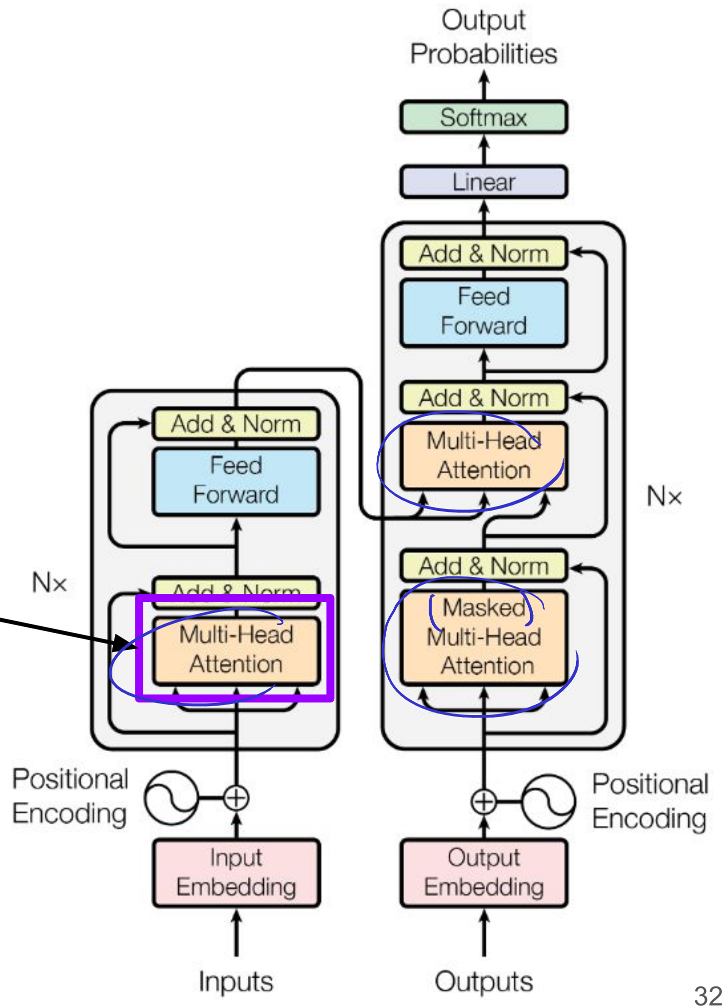
Multi-Head Attention (MHA)

- Multi-Head Attention — 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 ➜ h

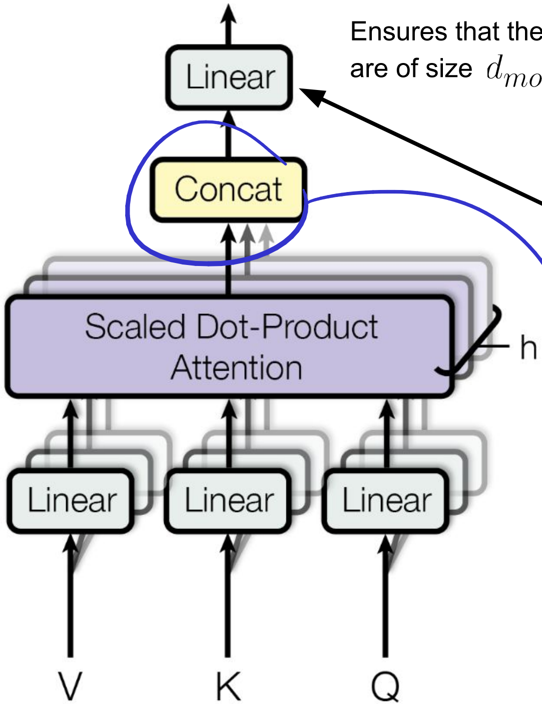


Multi-Head Attention





Multi-Head Attention



Ensures that the outputs are of size d_{model}

```python
import torch
import torch.nn as nn


class MultiHeadAttention(nn.Module):

    def __init__(self, num_heads, model_size, qkv_size):
        super().__init__()

        # Define num_heads attention heads
        self.heads = nn.ModuleList(
            [ AttentionHead(model_size, qkv_size) for _ in range(num_heads) ]
        )

        # Linear layer to "unify" all heads into one
        self.Wo = nn.Linear(num_heads * qkv_size, model_size)

    def forward(self, query, key, value):
        # Compute the outputs for all attention heads
        out_heads = [ head(query, key, value) for head in self.heads ]

        # Concatenate output of all attention heads
        out = torch.cat(out_heads, dim=-1)

        # Unify concatenated output to the model size
        return self.Wo(out)
```

model_size

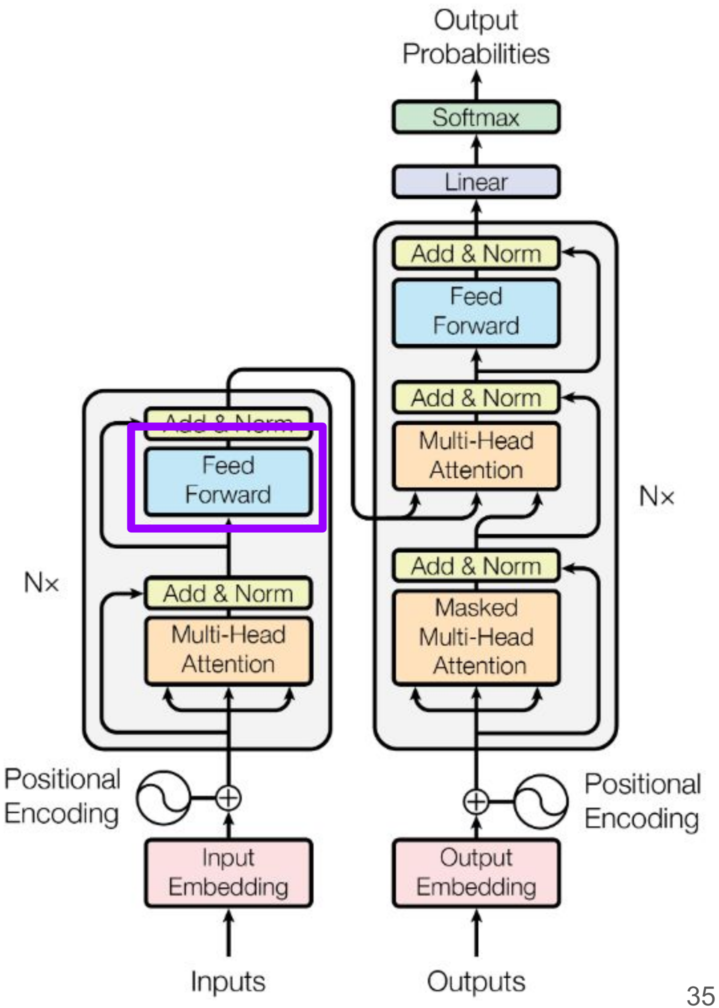
Outline

- 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

- Feed Forward Layer — purpose
  - The original paper doesn’t say
  - …uh, increase capacity / complexity

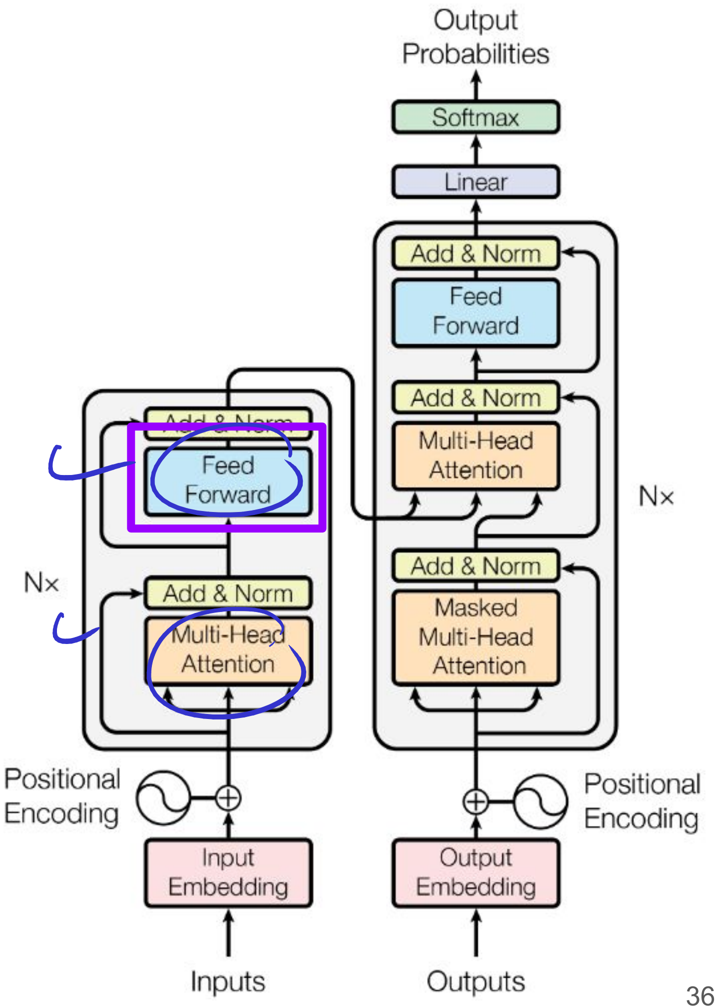
> Feed-forward layers constitute two-thirds of a transformer model’s parameters, yet their role in the network remains under-explored.



Source: [Transformer Feed-Forward Layers Are Key-Value Memories (2021)]

Feed Forward Layer

```python
import torch
import torch.nn as nn


class FeedForward(nn.Module):

    def __init__(self, model_size, hidden_size=2048):
        super().__init__()

        # Very simple 2-layer fully connected network
        self.net = nn.Sequential(
            nn.Linear(model_size, hidden_size),
            nn.ReLU(),
            nn.Linear(hidden_size, model_size),
        )

    def forward(self, X):
        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 / Block

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

Oversimplified!

(1) Residual Connections

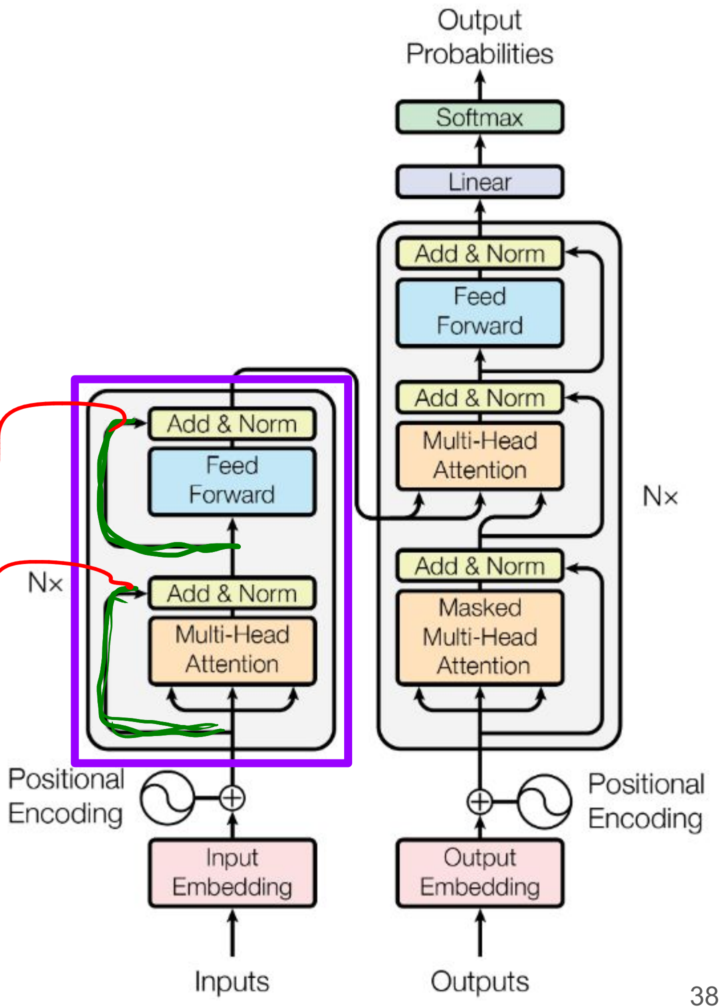
- Help mitigate the vanishing gradient problem

(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

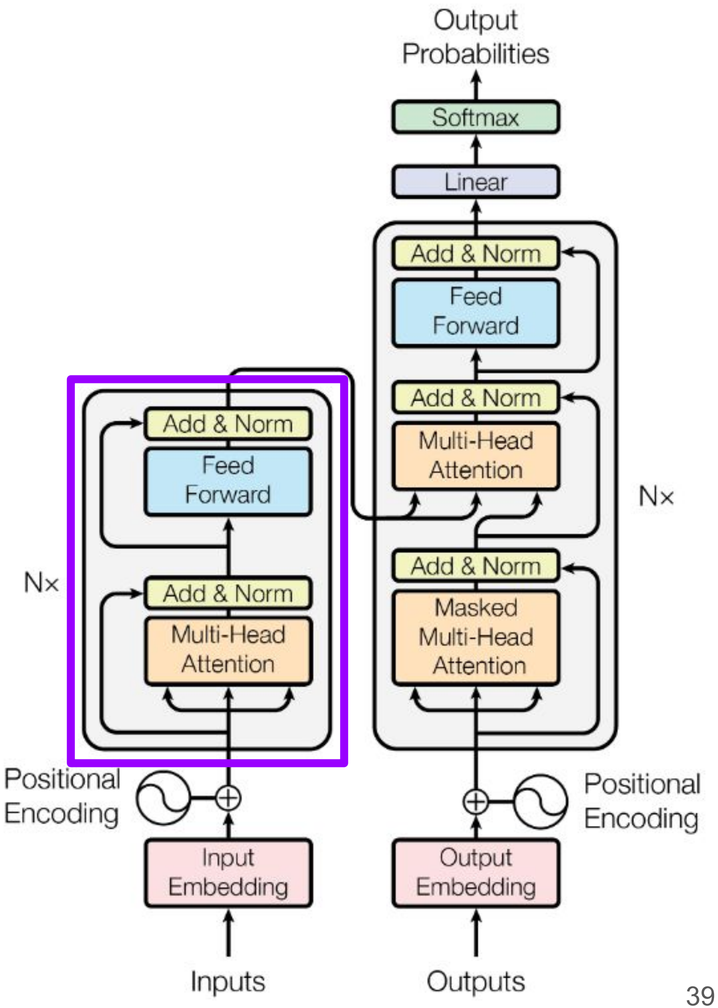


Encoder Layer / block

```python
import torch
import torch.nn as nn


class TransformerEncoderLayer(nn.Module):

    def __init__(self, model_size, num_heads, ff_hidden_size, dropout):
        super().__init__()

        # Define sizes of Q/K/V based on model size and number of heads
        qkv_size = max(model_size // num_heads, 1)

        # MultiHeadAttention block
        self.mha1 = MultiHeadAttention(num_heads, model_size, qkv_size)
        self.dropout1 = nn.Dropout(dropout)
        self.norm1 = nn.LayerNorm(model_size)

        # FeedForward block
        self.ff = FeedForward(model_size, ff_hidden_size)
        self.dropout2 = nn.Dropout(dropout)
        self.norm2 = nn.LayerNorm(model_size)

    def forward(self, source):
        # MultiHeadAttention block
        out1 = self.mha1(source, source, source)
        out1 = self.dropout1(out1)
        out1 = self.norm1(out1 + source)

        # FeedForward block
        out2 = self.ff(out1)
        out2 = self.dropout2(out2)
        out2 = self.norm2(out2 + out1)

        # Return final output
        return out2
```

Self-Attention

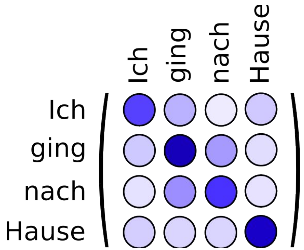
$$Q = K = V$$



Encoder — Self-Attention

- Example: German-to-English machine translation

$$\underbrace{\mathrm{softmax}\left(\frac{\begin{array}{r}\text{Ich}\\\text{ging}\\\text{nach}\\\text{Hause}\end{array}\left(Q\right)\times\begin{array}{c}\begin{array}{cccc}\text{Ich}&\text{ging}&\text{nach}&\text{Hause}\end{array}\\\left(K^{T}\right)\end{array}}{\sqrt{d_k}}\right)}\times\begin{array}{r}\text{Ich}\\\text{ging}\\\text{nach}\\\text{Hause}\end{array}\left(V\right)=\begin{array}{r}\text{Ich}\\\text{ging}\\\text{nach}\\\text{Hause}\end{array}\left(\quad\right)$$



Self-Attention Matrix

(rows sum up to 1!)

word embeddings re-weighted based on attention weights

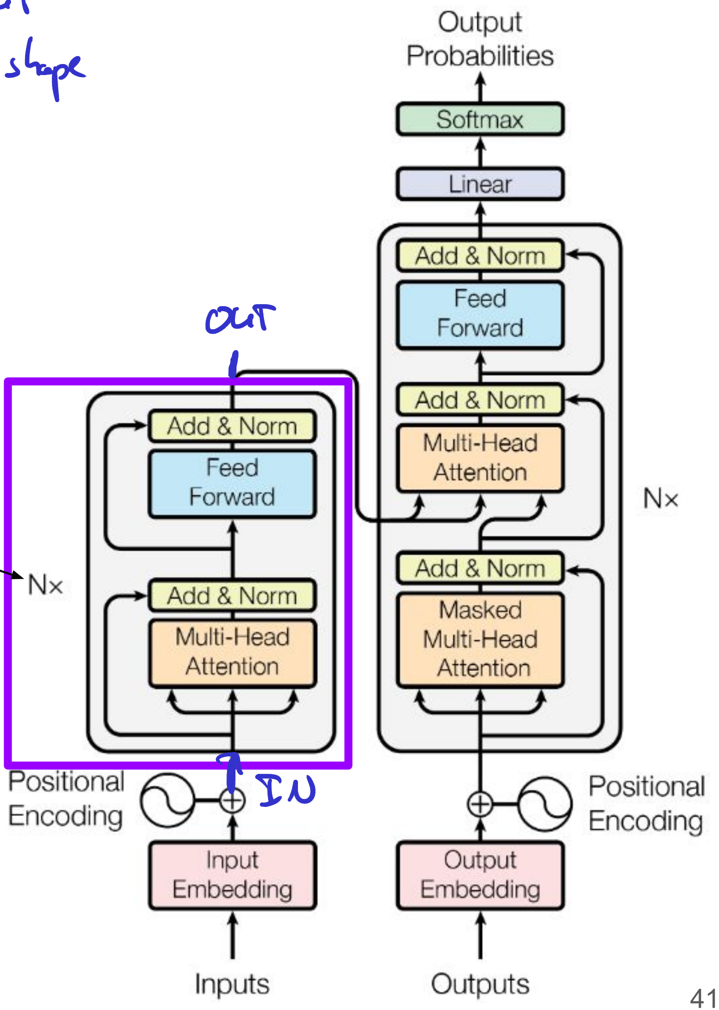
Complete Encoder

IN and OUT
have the same shape

```python
import torch
import torch.nn as nn


class TransformerEncoder(nn.Module):

    def __init__(self,
                 num_layers=6,          # Common default values
                 model_size=512,        # used in original paper
                 num_heads=8,
                 ff_hidden_size=2048,
                 dropout= 0.1):
        super().__init__()

        # Define num_layers (N) encoder layers
        self.layers = nn.ModuleList(
            [ TransformerEncoderLayer(model_size,
                                      num_heads,
                                      ff_hidden_size,
                                      dropout)
              for _ in range(num_layers)
            ]
        )

    def forward(self, source):
        # Push through each encoder layer
        for l in self.layers:
            source = l(source)
        return source
```

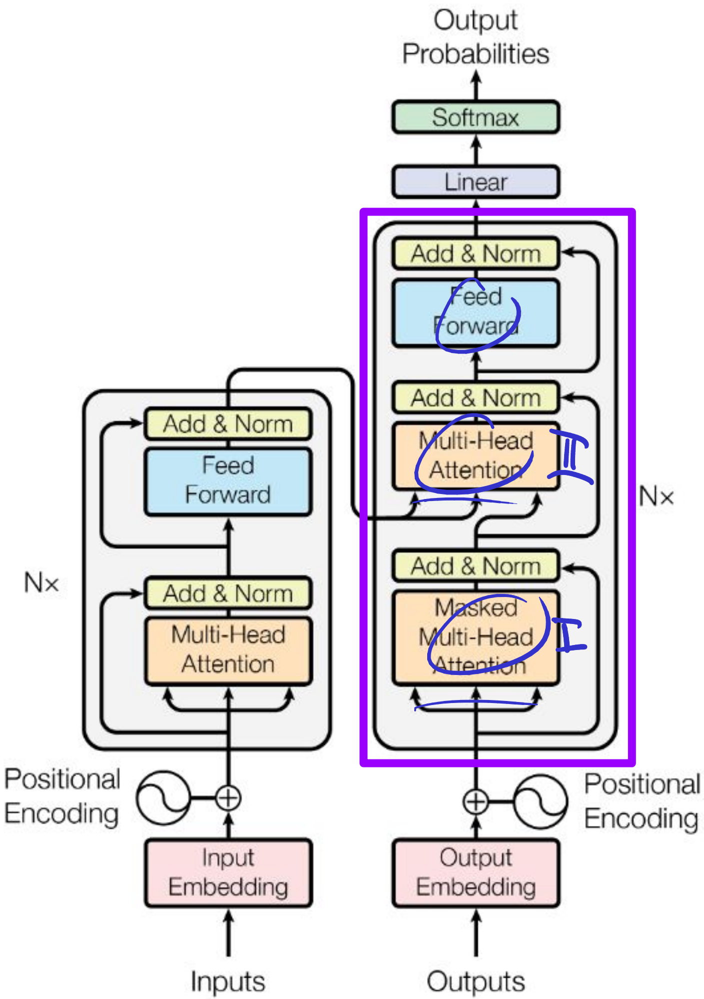
6x

e.g. German sentence



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

```python
import torch
import torch.nn as nn


class TransformerDecoderLayer(nn.Module):

    def __init__(self, model_size, num_heads, ff_hidden_size, dropout):
        super().__init__()

        # Define sizes of Q/K/V based on model size and number of heads
        qkv_size = max(model_size // num_heads, 1)

        # 1st MultiHeadAttention block (decoder input only)
        self.mha1 = MultiHeadAttention(num_heads, model_size, qkv_size)
        self.dropout1 = nn.Dropout(dropout)
        self.norm1 = nn.LayerNorm(model_size)

        # 2nd MultiHeadAttention block (encoder & decoder)
        self.mha2 = MultiHeadAttention(num_heads, model_size, qkv_size)
        self.dropout2 = nn.Dropout(dropout)
        self.norm2 = nn.LayerNorm(model_size)

        self.ff = FeedForward(model_size, ff_hidden_size)
        self.dropout3 = nn.Dropout(dropout)
        self.norm3 = nn.LayerNorm(model_size)

    def forward(self, target, memory):
        # 1st MultiHeadAttention block
        out1 = self.mha1(target, target, target)
        out1 = self.dropout1(out1)
        out1 = self.norm1(out1 + target)
        # 2nd MultiHeadAttention block
        out2 = self.mha2(out1, memory, memory)
        out2 = self.dropout2(out2)
        out2 = self.norm2(out2 + out1)
        # FeedForward block
        out3 = self.ff(out2)
        out3 = self.dropout3(out3)
        out3 = self.norm3(out3 + out2)
        # Return final output
        return out3
```

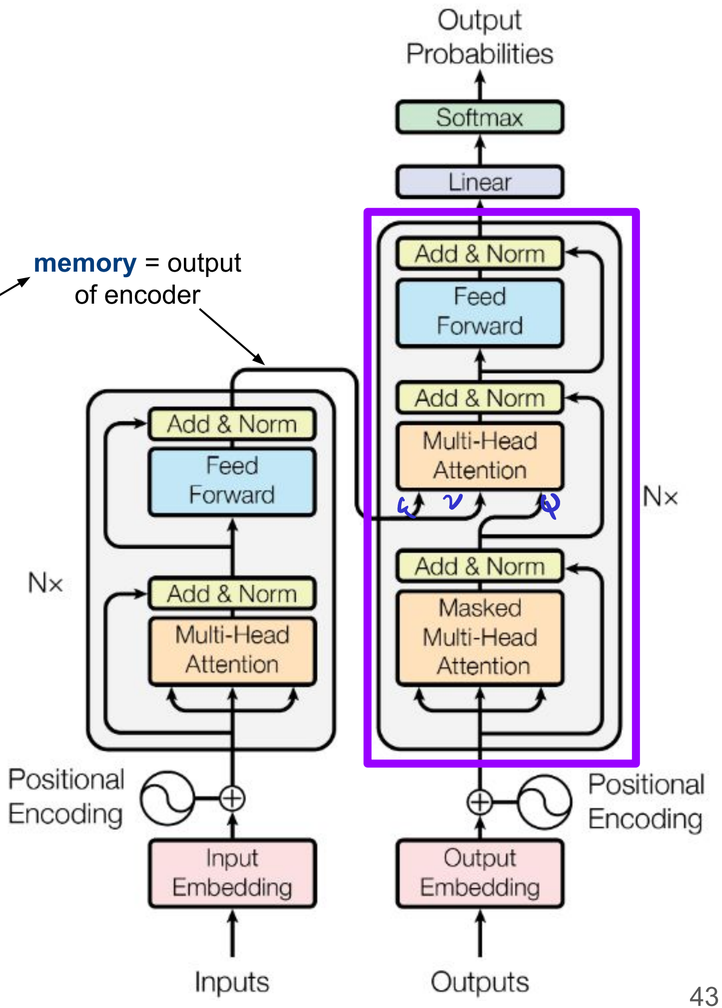
Self-Attention

$$Q = K = V$$

Source–Target Attention

$$Q \neq K = V$$

memory = output of encoder



Decoder — Attentions

- Example: German-to-English machine translation

$$\mathrm{softmax}\left(\frac{\begin{array}{c|c} \text{I} & \\ \text{went} & Q \\ \text{home} & \end{array} \times \begin{array}{ccc} \text{I} & \text{went} & \text{home} \\ \hline & K^T & \end{array}}{\sqrt{d_k}}\right) \times \begin{array}{c|c} \text{I} & \\ \text{went} & V \\ \text{home} & \end{array} = \begin{array}{c|c} \text{I} & \\ \text{went} & \\ \text{home} & \end{array}$$

Self-Attention
$Q = K = V$

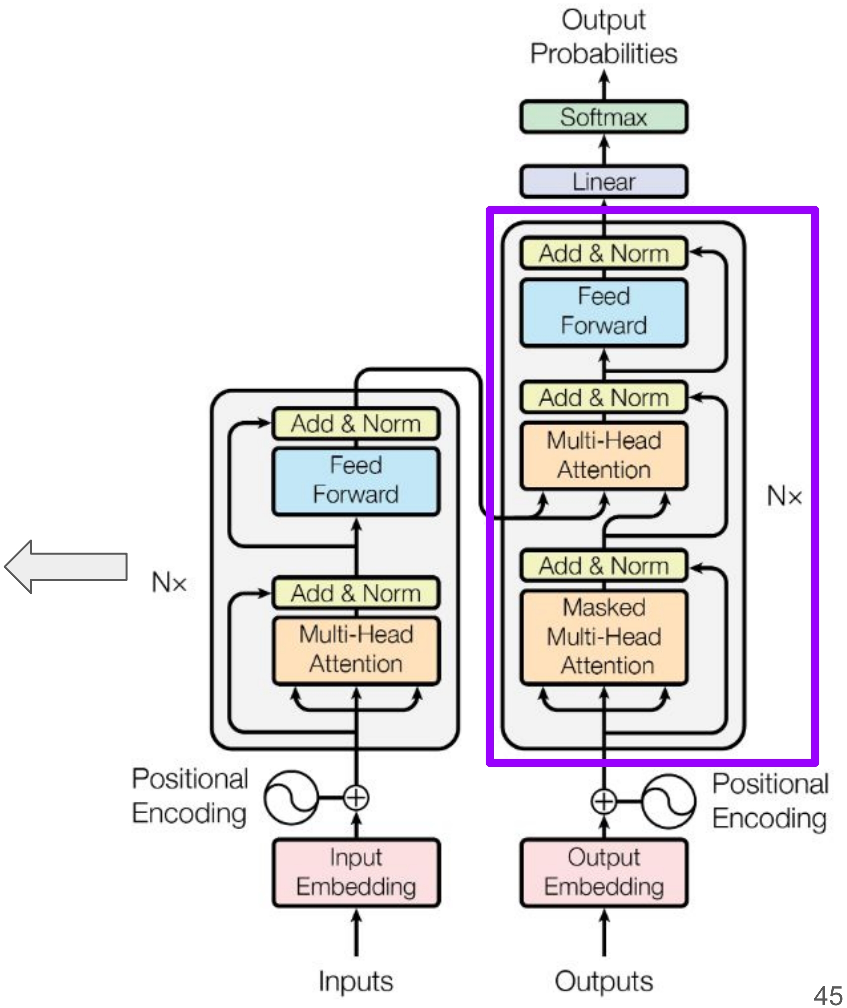
$$\mathrm{softmax}\left(\frac{\begin{array}{c|c} \text{I} & \\ \text{went} & Q \\ \text{home} & \end{array} \times \begin{array}{cccc} \text{Ich} & \text{ging} & \text{nach} & \text{Hause} \\ \hline & K^T & & \end{array}}{\sqrt{d_k}}\right) \times \begin{array}{c|c} \text{Ich} & \\ \text{ging} & V \\ \text{nach} & \\ \text{Hause} & \end{array} = \begin{array}{c|c} \text{Ich} & \\ \text{ging} & \\ \text{nach} & \\ \text{Hause} & \end{array}$$

Cross-Attention
$Q \neq K = V$

Complete Decoder

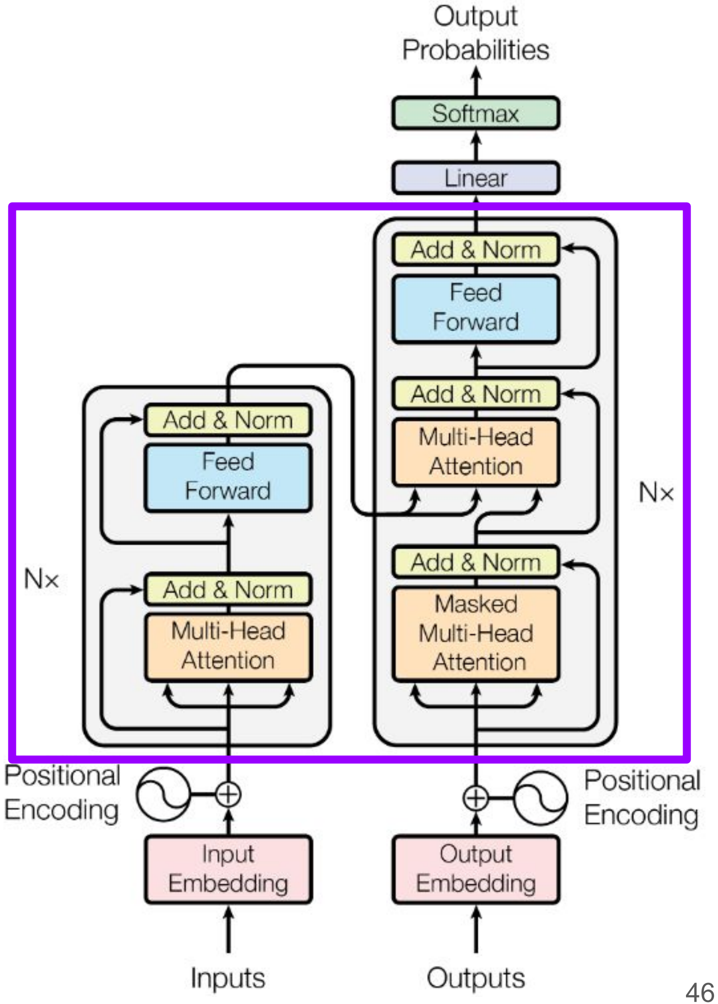
```python
import torch
import torch.nn as nn


class TransformerDecoder(nn.Module):

    def __init__(self,
                 num_layers=6,        # Common default values
                 model_size=512,      # used in original paper
                 num_heads=8,
                 ff_hidden_size=2048,
                 dropout= 0.1):
        super().__init__()

        # Define num_layers (N) decoder layers
        self.layers = nn.ModuleList(
            [ TransformerDecoderLayer(model_size,
                                      num_heads,
                                      ff_hidden_size,
                                      dropout)
              for _ in range(num_layers)
            ]
        )

    def forward(self, target, memory):
        # Push through each decoder layer
        for l in self.layers:
            target = l(target, memory)
        return target
```

6



Complete Transformer

```python
import torch
import torch.nn as nn


class Transformer(nn.Module):

    def __init__(self,
                 num_encoder_layers=6, # Common default values
                 num_decoder_layers=6, # used in original paper
                 model_size=512,
                 num_heads=8,
                 ff_hidden_size=2048,
                 dropout= 0.1):
        super().__init__()

        # Definer encoder
        self.encoder = TransformerEncoder(
            num_layers=num_encoder_layers,
            model_size=model_size,
            num_heads=num_heads,
            ff_hidden_size=ff_hidden_size,
            dropout=dropout
        )

        #Define decoder
        self.decoder = TransformerDecoder(
            num_layers=num_decoder_layers,
            model_size=model_size,
            num_heads=num_heads,
            ff_hidden_size=ff_hidden_size,
            dropout=dropout
        )

    def forward(self, source, target):
        memory = self.encoder(source)
        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

a_{ij} = dot(movie, <PAD>)

- 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

"some" special

- Masking padded words

a_{ij}

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

Masking matrix M

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

nothing

$$a_{ij} + 0 = a_{ij}$$

$$a_{ij} + (-\infty) = -\infty$$

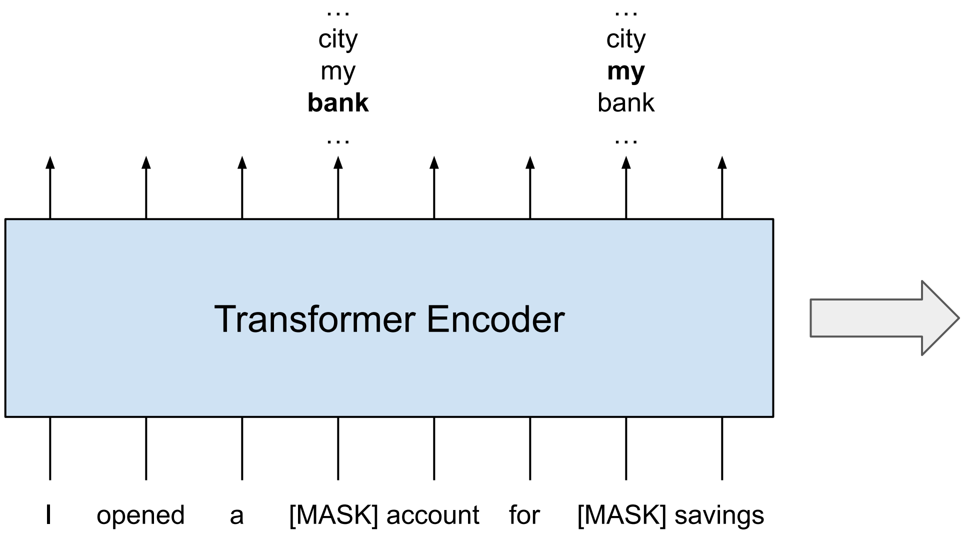
0 after Softmax!

Masking for Language Modeling

$a_{ij} = \mathrm{dot}(\text{opened}, [\text{MASK}])$

$c_{ij} + (-\infty) = -\infty \xRightarrow{\text{softmax}} 0$

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



Masking matrix M

$$\begin{pmatrix} \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & .. \\ 0 & 0 & 0 & \underline{-\infty} & 0 & 0 & \underline{-\infty} & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & .... & \cdots & \cdots & \cdots \end{pmatrix}$$

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

$$\begin{array}{cc} & \begin{array}{cccc} \text{<S>} & \text{I} & \text{went} & \text{home} \end{array} \\ \begin{array}{c} \text{<S>} \\ \text{I} \\ \text{went} \\ \text{home} \end{array} & \begin{pmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

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

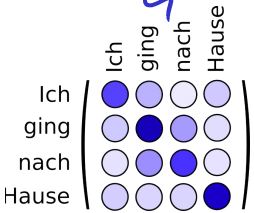
Attention — Performance Considerations

- 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)

4 4

$$\underbrace{\operatorname{softmax}\left(\frac{\begin{array}{r}\text{Ich}\\\text{ging}\\\text{nach}\\\text{Hause}\end{array}\begin{pmatrix} Q \end{pmatrix} \times \begin{array}{c}\text{Ich ging nach Hause}\\ \begin{pmatrix} K^T \end{pmatrix}\end{array}}{\sqrt{d_k}}\right)} \times \begin{array}{r}\text{Ich}\\\text{ging}\\\text{nach}\\\text{Hause}\end{array}\begin{pmatrix} V \end{pmatrix} = \begin{array}{r}\text{Ich}\\\text{ging}\\\text{nach}\\\text{Hause}\end{array}\begin{pmatrix} \ \end{pmatrix}$$

4

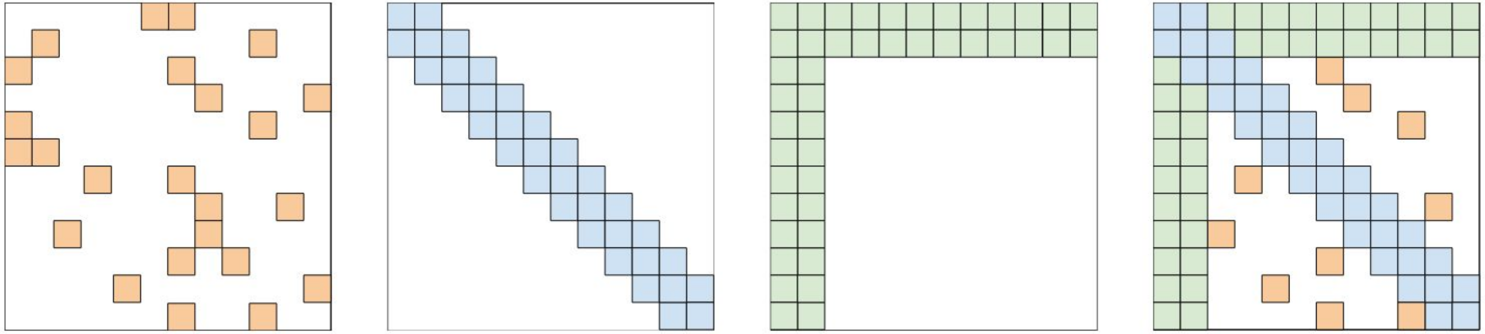


4 => 16 ops

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:



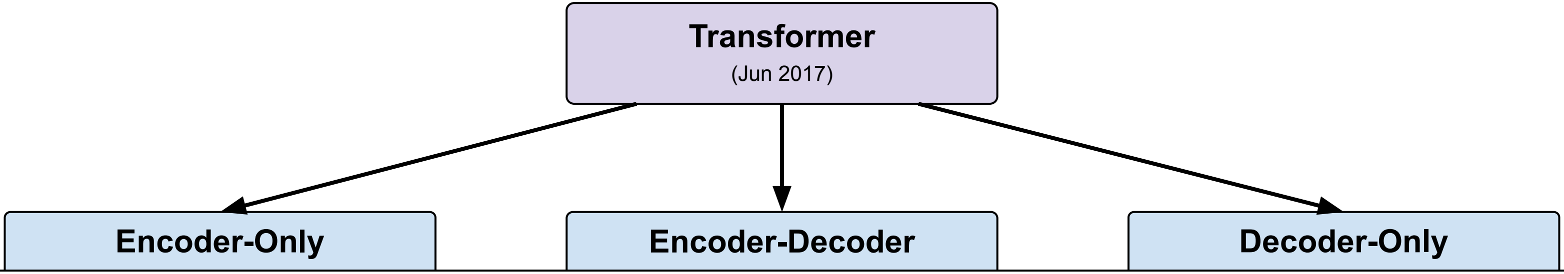
(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

Source: [Big Bird: Transformers for Longer Sequences (2021)]

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

Architectures



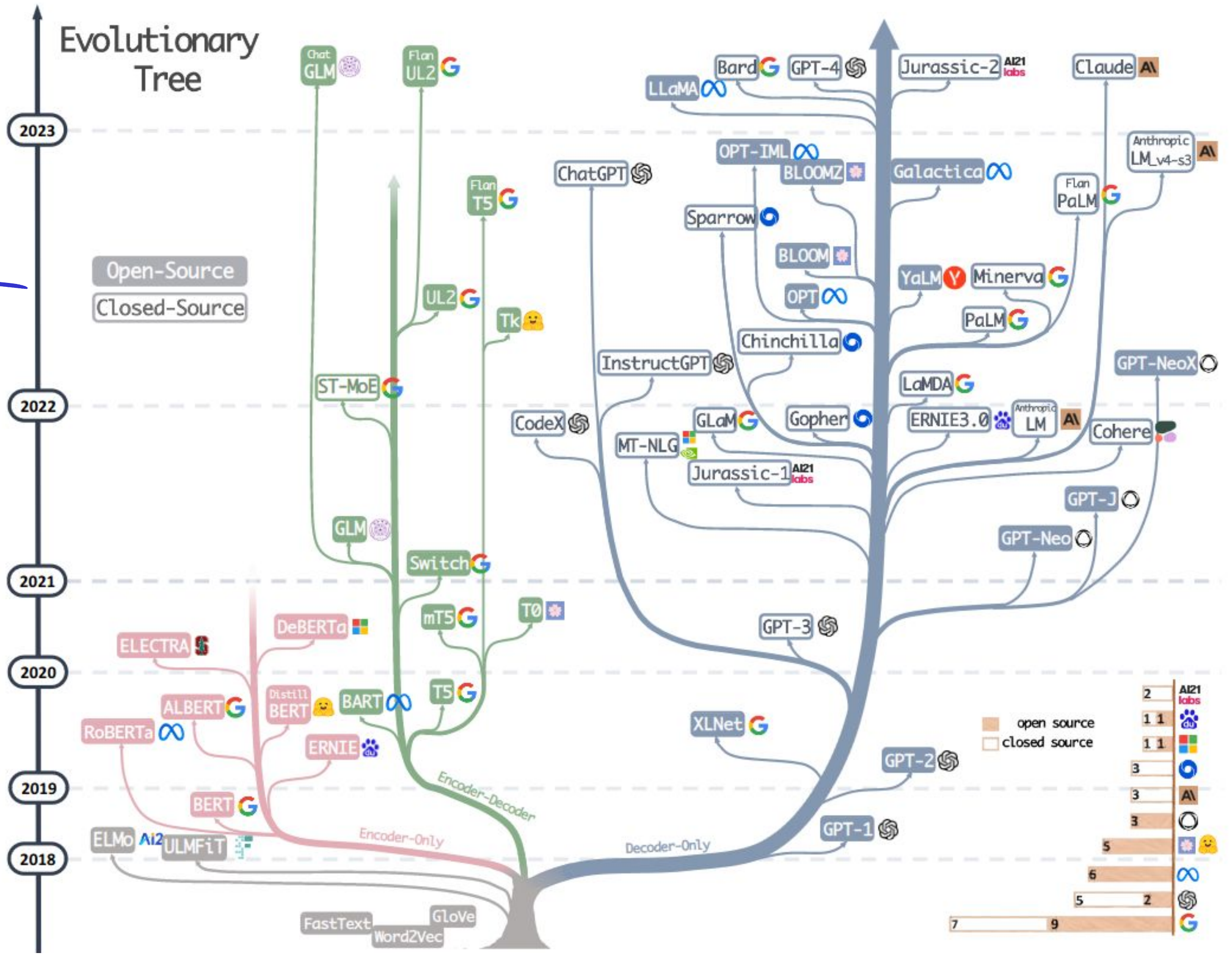
In-Lecture Activity (~~10~~ 8 mins incl. break)

- Question: What is the intuition behind using different LLM architectures
  - Encoder-only vs. encoder-decoder vs. decoder-only
  - Post your RegEx to Canvas > Discussions
    (individually or as a group; include all group members' names in the post)

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

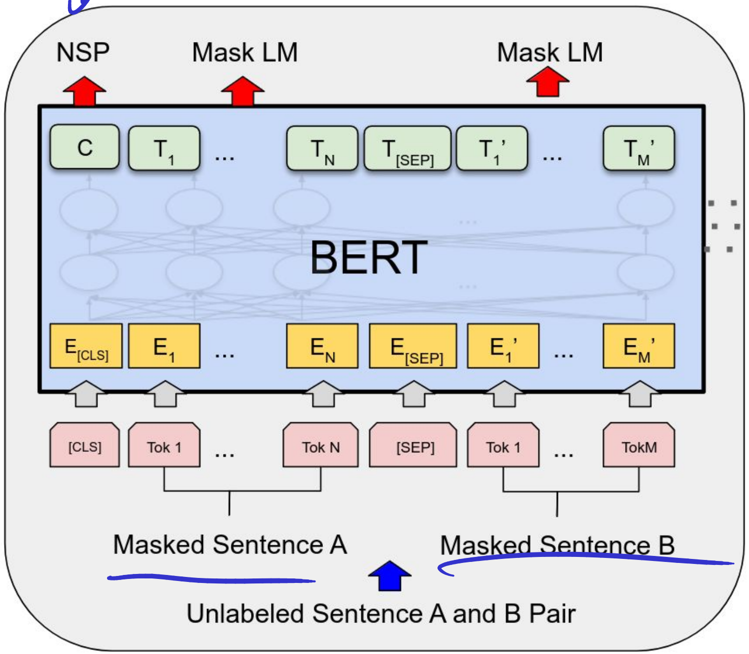
- Contextual Word Embeddings
  - Motivation
  - ELMo
- Transformers
  - Positional Encoding
  - Core Layers
  - Encoder & Decoder
- Extended Concepts
  - Masking
  - Restricted Attention
- **Transformer-based LLMs**
  - Overview
  - **Encoder-only: BERT, RoBERTa**
  - Encoder-Decoder: T5, BART
  - Decoder-only: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

BERT (Bidirectional Encoder Representations from Transformers)

- BERT
  - Uses only the Transformer Encoder
  - Self-supervised training

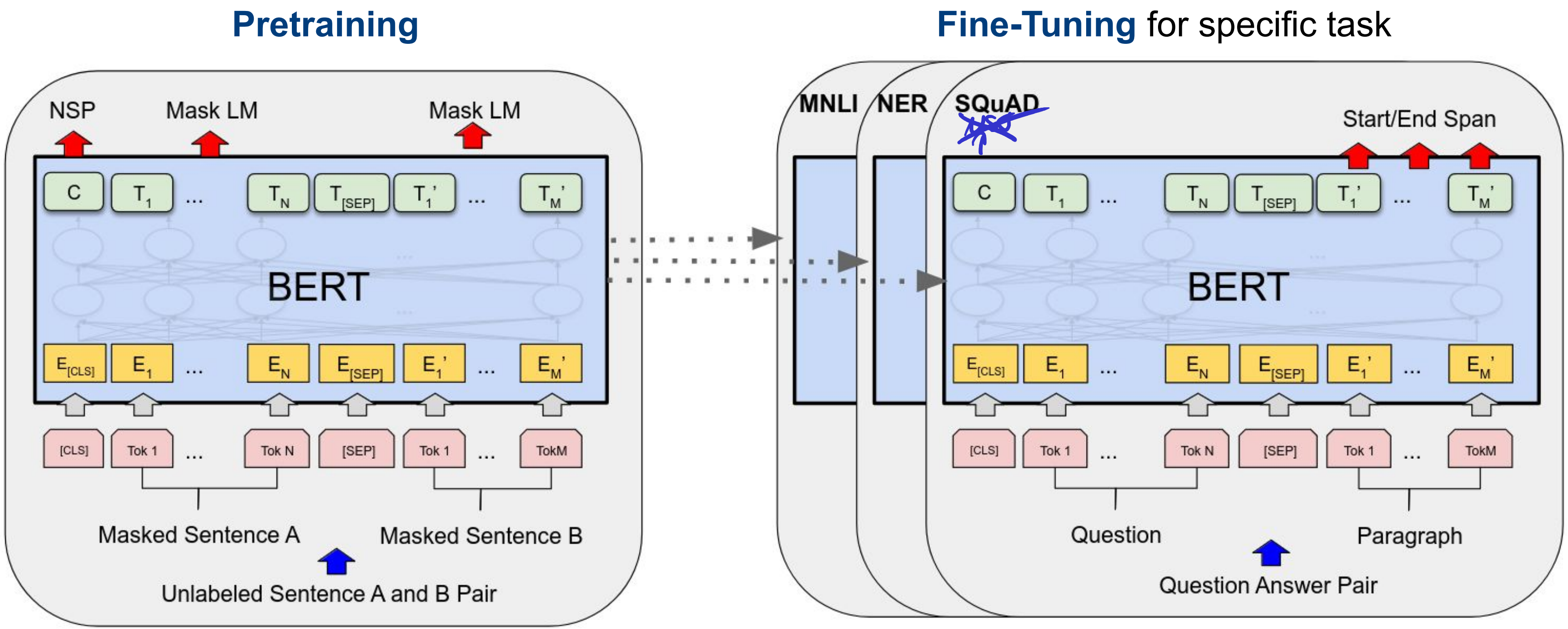
- Train on 2 learning objectives
  - MLM: Masked Language Model
    (predicted the words masked in the input sentences)
  - NSP: Next Sentence Prediction
    (predict if 2nd sentence was indeed followed 1st sentence)

Y/N



Source: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (2018)

BERT (Bidirectional Encoder Representations from Transformers)



Source: [BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (2018)]

RoBERTa (A Robustly Optimized Bidirectional Encoder Representations from Transformers)

- RoBERTa ≈ 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 Trans-former, MLM & NSP	BERT without NSP, Using Dynamic Masking	BERT Distillation	BERT with reduced para-meters & SOP (not NSP)

Source: [Humboldt-Universität zu Berlin website]

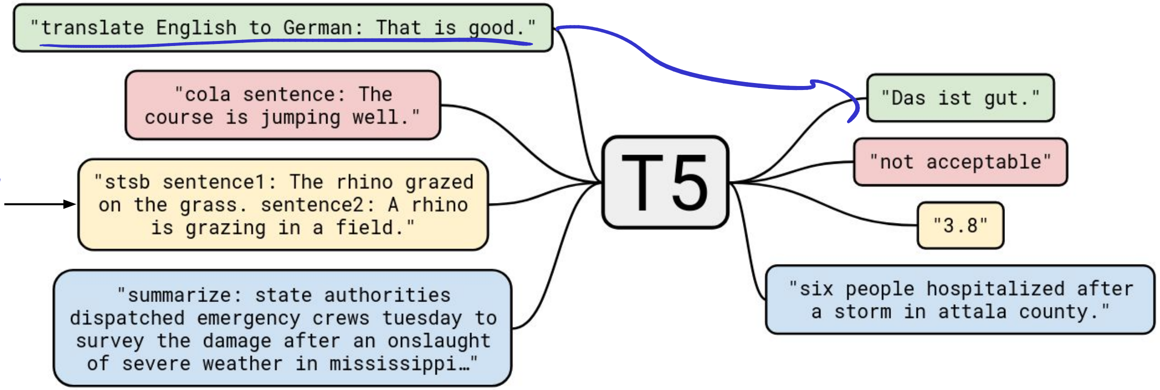
Outline

- 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)

- T5 — 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)

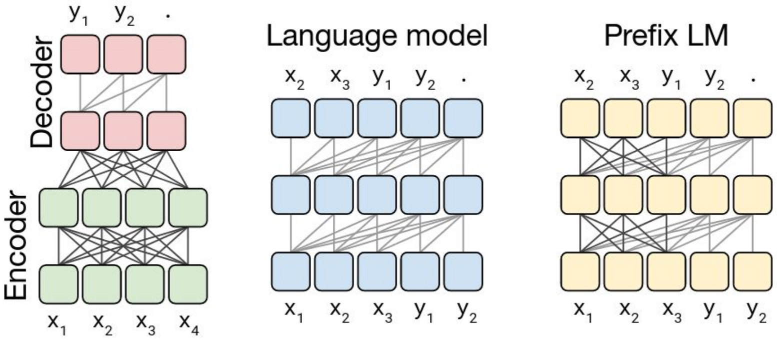
Example: Semantic Text Similarity Benchmark (STSB) training data sample reformulated as a text-to-text task



Source: Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer (2019)

T5 (Text-to-Text Transfer Transformer)

- T5 — evaluation
  - The authors evaluated multi-task learning approach for different architectures
  - Best results: encoder-decoder architecture

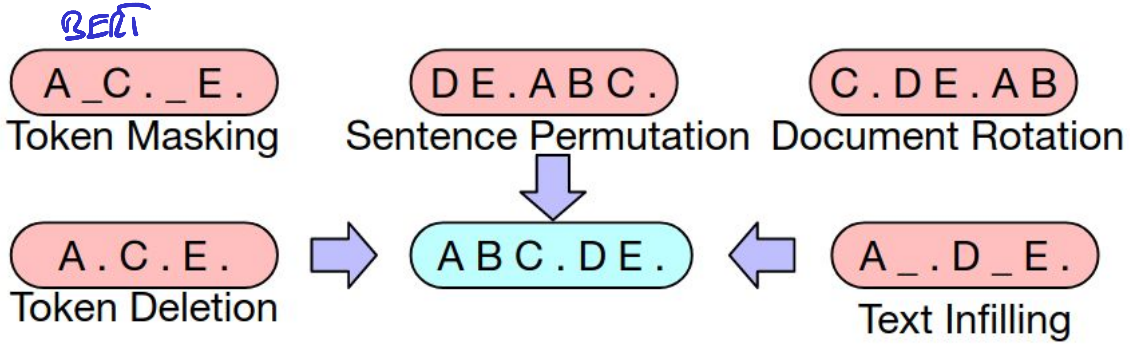


Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	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

Source: [Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer (2019)]

BART (Bidirectional and Auto-Regressive Transformers)

- BART — core concepts
  - Basic encoder-decoder Transformer architecture
  - Trained by corrupting documents and then optimizing a reconstruction loss ➜ **denoising**
    (the cross-entropy between the decoder's output and the original document)
  - Various transformation techniques to corrupt input documents

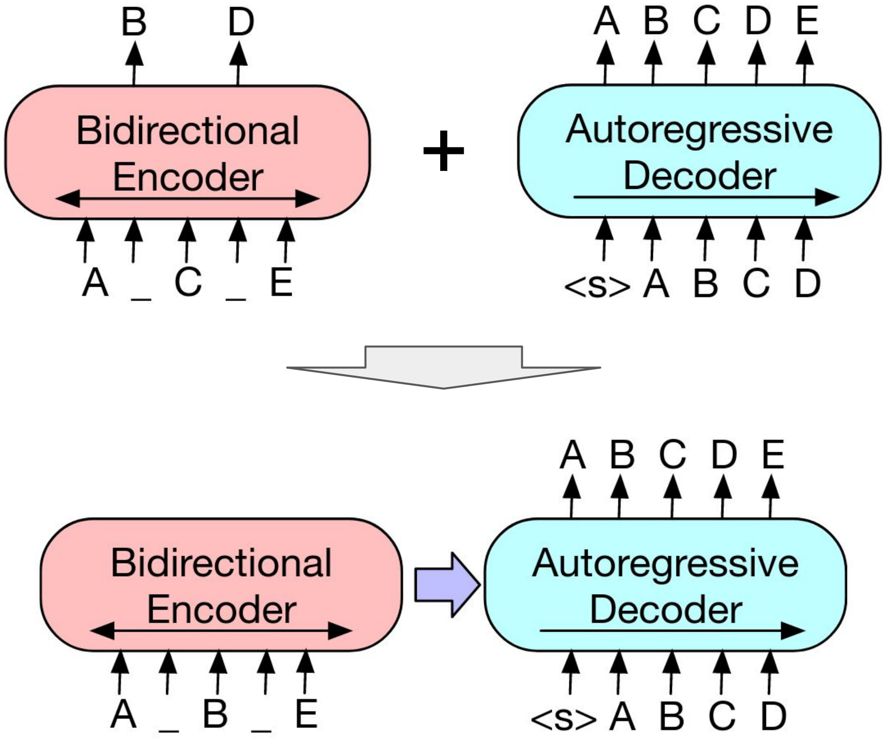


Source: [BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension (2019)]

BART ≈ 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

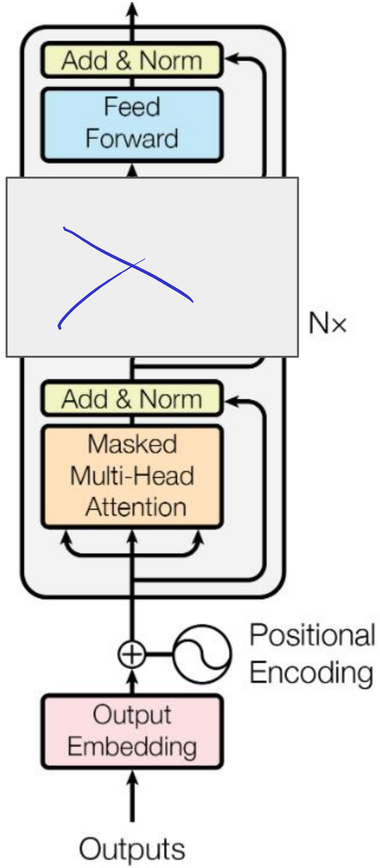
Source: [BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension (2019)]

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 ↳ next-word prediction



- 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)

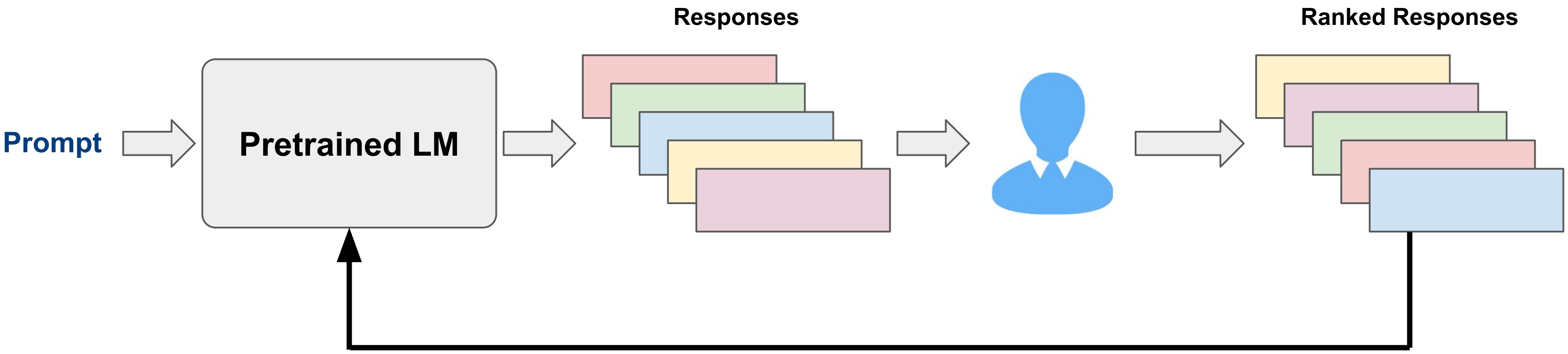
- GPT-3 models

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

Source: Language Models are Few-Shot Learners (2020)

GPT — RLHF (Reinforcement Learning from Human Feedback)

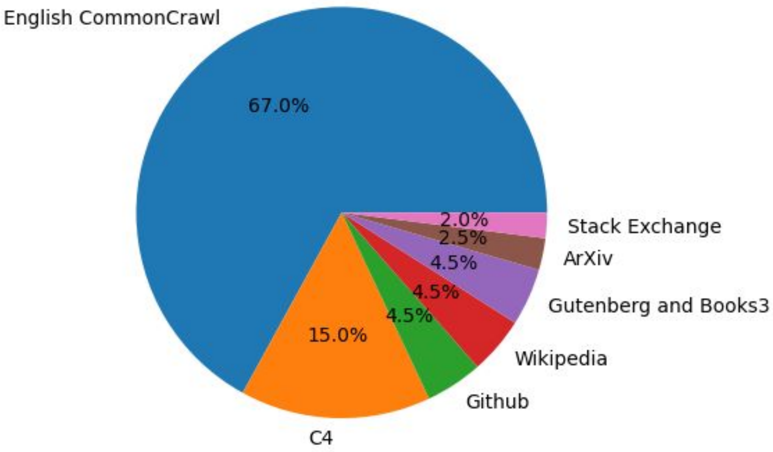
- RLHF — two common setups
  - Use human-generated responses to prompts to fine-tune the pretrained model
  - Generate multiple response for same prompt; human ranks response; use ranking for fine-tuning



LLaMA (Large Language Model Meta AI)

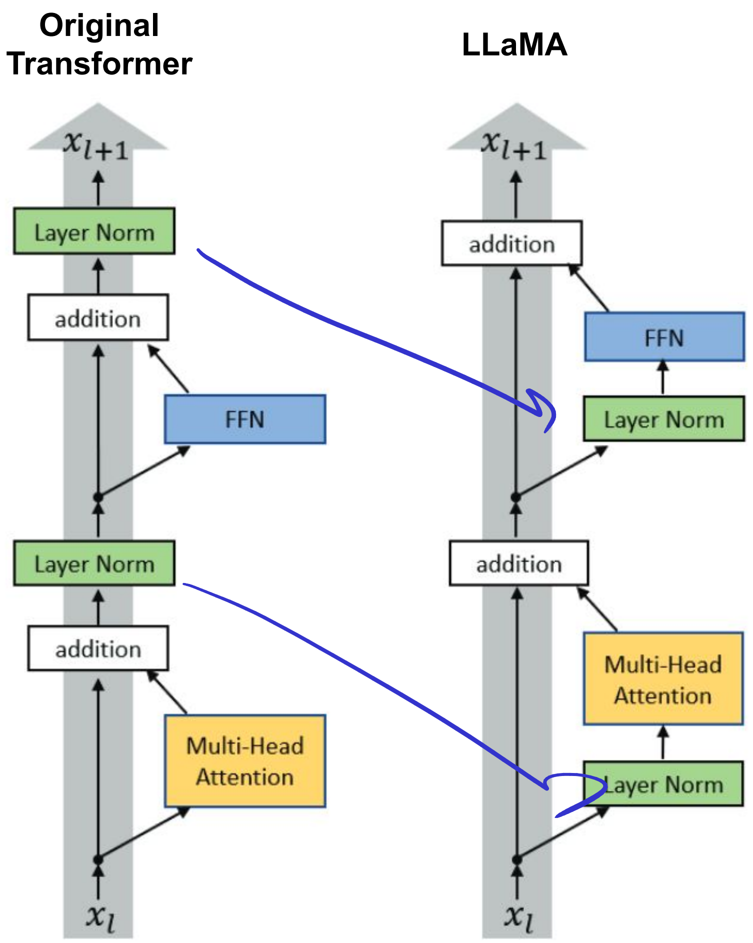
- De ~~En~~coder-only architecture very similar to GPT (any many others!) — main tweaks
  - Pre-normalization: layer normalization is put **inside** the residual blocks
  - SwiGLU (Swish-Gated Linear Unit) activation: non-monotonic, parameterized activation function
  - Rotary Positional Embeddings: encode word positions by rotating word embedding vectors

- Open LLM
  - Trained exclusively on publicly available data



LLaMA — Pre-Normalization

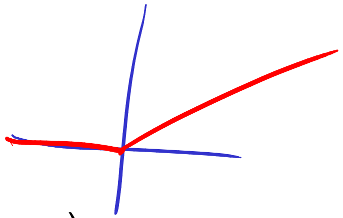
- Post vs. pre-normalization
  - Post: layer normalization **between** residual blocks (original transformer)
  - Pre: layer normalization **inside** residual blocks (LLaMA, etc.)
  - Observed benefit of pre-normalization:
    - "Well-behaved" gradients at initialization
    - Significantly faster training



Source: [On Layer Normalization in the Transformer Architecture (2020)]

LLaMA — SwiGLU (Swish-Gated Linear Unit)

ReLU



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)

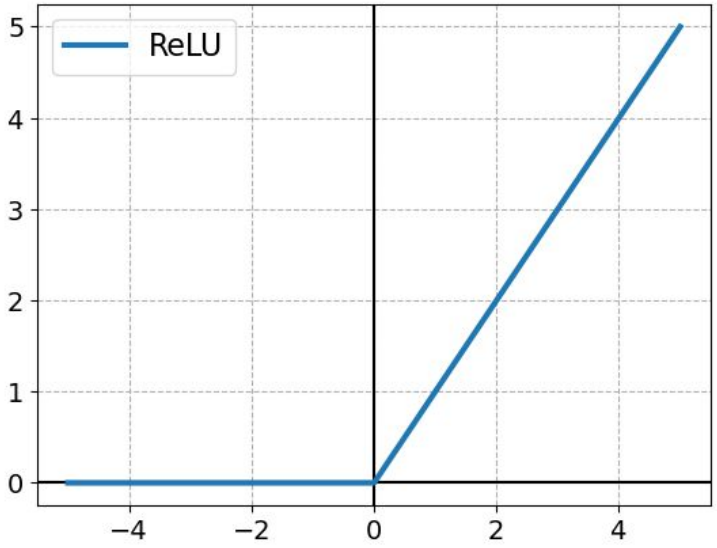
- Simple parameterized activation function
- Approach: "try and see what works best"

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

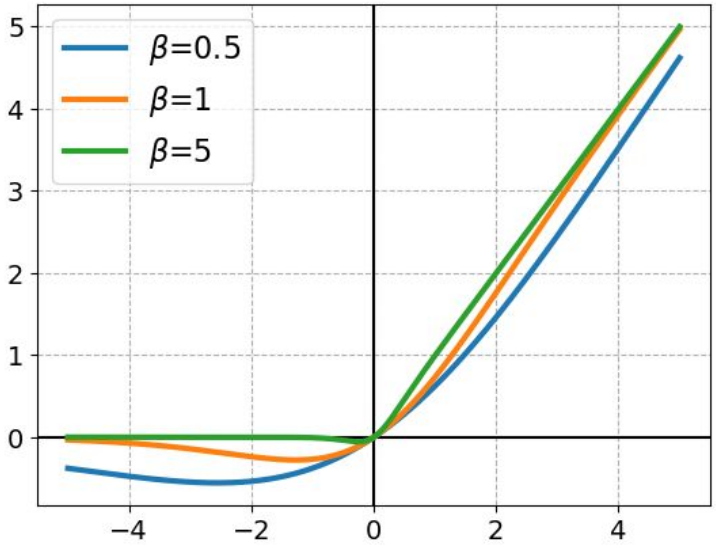
$$SwiGLU(x) = (xW + b) \otimes Swish_{\beta}(xV + c)$$

LLaMA — SwiGLU (Swish-Gated Linear Unit)

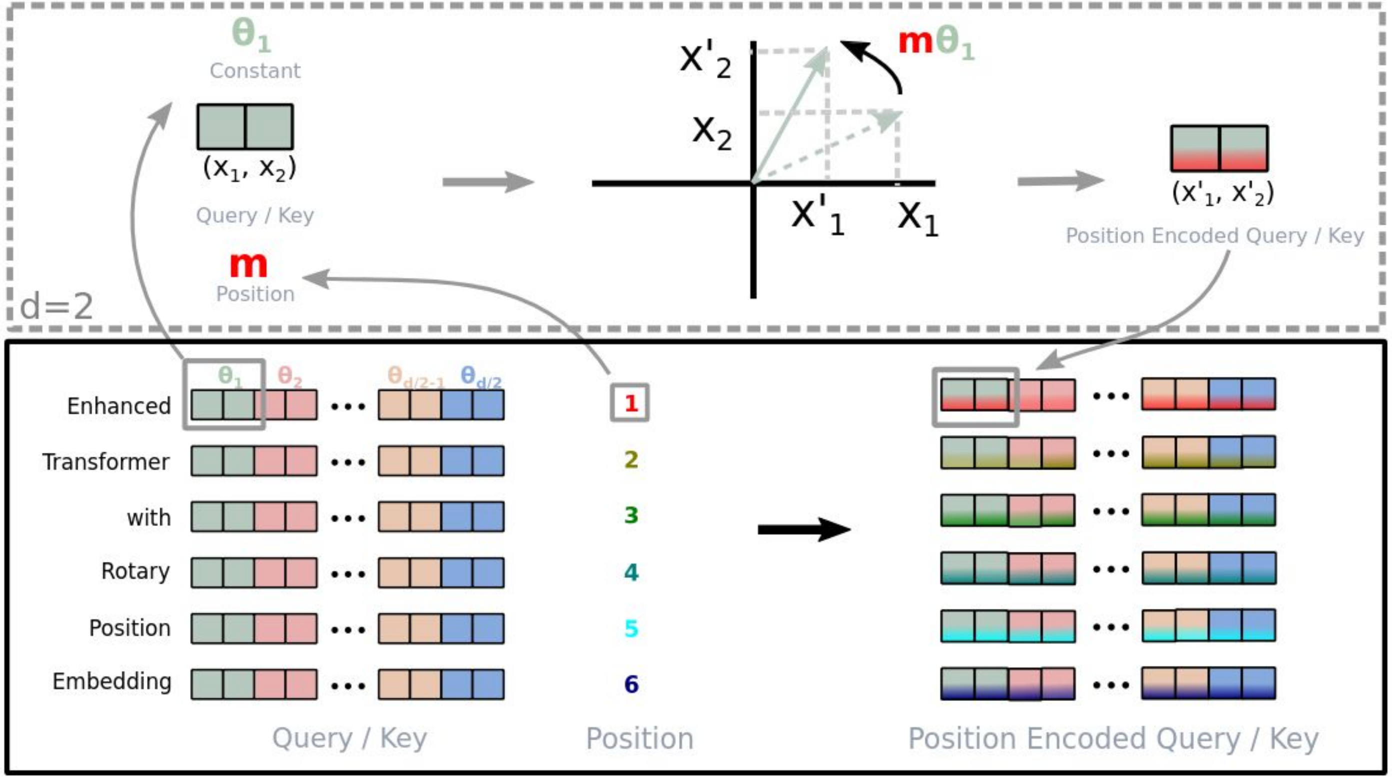
ReLU (Linear Rectified Unit)



Swish



LLaMA — Rotary Positional Embeddings



Source: [RoFormer: Enhanced Transformer with Rotary Position Embedding (2021)]

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 <u>really</u> 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

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

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

ChatGPT Poses Dangers for Online Dating Apps

Hallucinations, Plagiarism, and ChatGPT

Cybercriminals are using ChatGPT to create malware

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

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

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

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.

Source: [GLU Variants Improve Transformer (2020)]

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

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 for 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 his won't help you learn better

Solutions to Quick Quizzes

- Slide 4
  - Learning a language model is arguably easier since it is a self-supervised task
  - The annotations / labels are (almost) the same as the inputs ➜ (relatively) easier to set up
  - Note: this does not mean that it's also easier to get good results for Task A
- Slide 29
  - This make the total number of trainable parameters independent from the number of heads