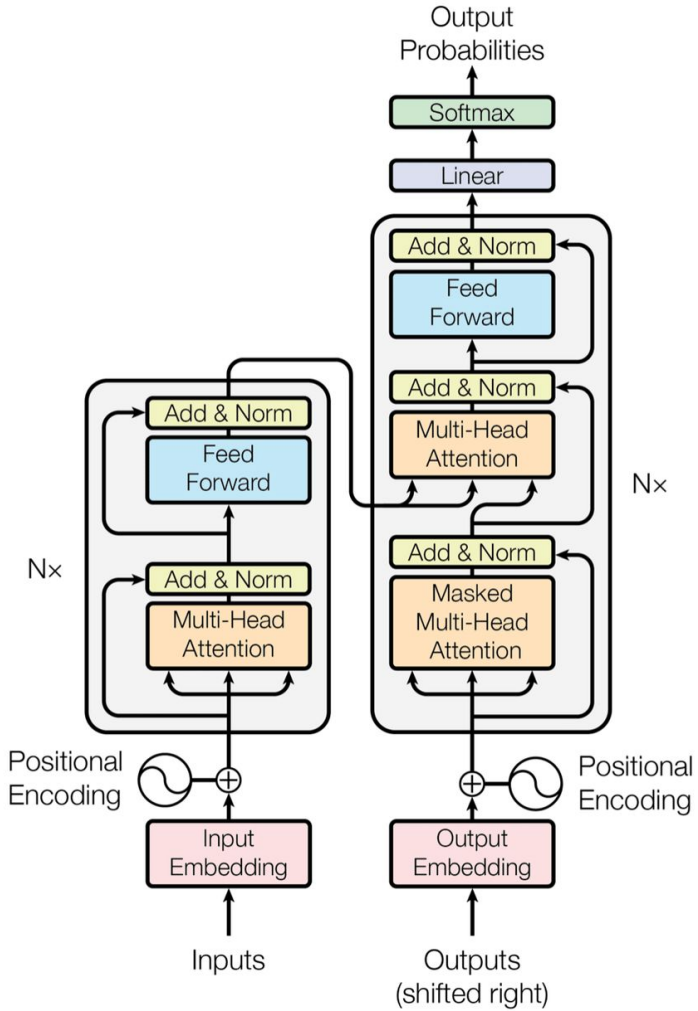
Transformers

Zhe, Germann, Ameya

Code Walkthrough

Code & Images from:
https://nlp.seas.harvard.edu/2018/04/03/attention.html
https://princeton-nlp.github.io/cos484/readings/the-annotated-transformer.pdf

Highly recommended!

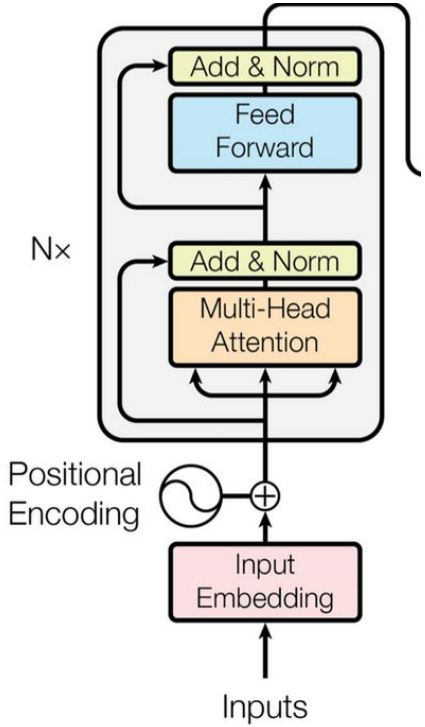

Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

```
class EncoderDecoder(nn.Module):
    """
    A standard Encoder-Decoder architecture. Base for this and many
    other models.
    """
    def __init__(self, encoder, decoder, src_embed, tgt_embed, generator):
        super(EncoderDecoder, self).__init__()
        self.encoder = encoder
        self.decoder = decoder
        self.src_embed = src_embed
        self.tgt_embed = tgt_embed
        self.generator = generator

    def forward(self, src, tgt, src_mask, tgt_mask):
        "Take in and process masked src and target sequences."
        return self.decode(self.encode(src, src_mask), src_mask,
                           tgt, tgt_mask)

    def encode(self, src, src_mask):
        return self.encoder(self.src_embed(src), src_mask)

    def decode(self, memory, src_mask, tgt, tgt_mask):
        return self.decoder(self.tgt_embed(tgt), memory, src_mask, tgt_mask)
```

```
class Generator(nn.Module):
    "Define standard linear + softmax generation step."
    def __init__(self, d_model, vocab):
        super(Generator, self).__init__()
        self.proj = nn.Linear(d_model, vocab)

    def forward(self, x):
        return F.log_softmax(self.proj(x), dim=-1)
```

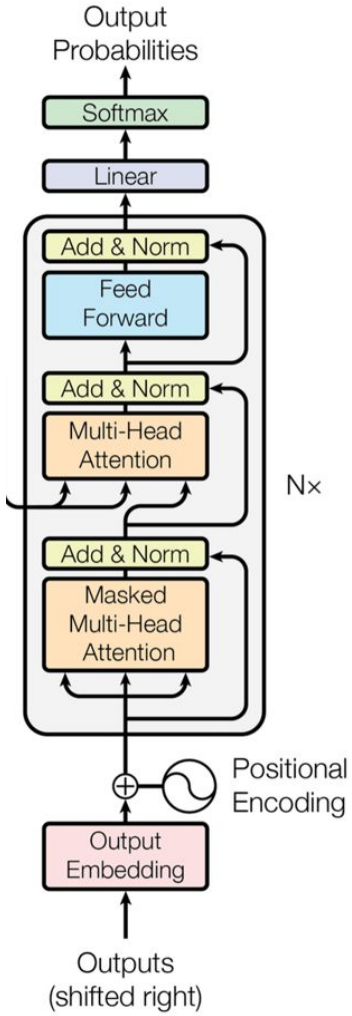
```python
class Encoder(nn.Module):
    "Core encoder is a stack of N layers"
    def __init__(self, layer, N):
        super(Encoder, self).__init__()
        self.layers = clones(layer, N)
        self.norm = LayerNorm(layer.size)

    def forward(self, x, mask):
        "Pass the input (and mask) through each layer in turn."
        for layer in self.layers:
            x = layer(x, mask)
        return self.norm(x)
```

```python
class EncoderLayer(nn.Module):
    "Encoder is made up of self-attn and feed forward (defined below)"
    def __init__(self, size, self_attn, feed_forward, dropout):
        super(EncoderLayer, self).__init__()
        self.self_attn = self_attn
        self.feed_forward = feed_forward
        self.sublayer = clones(SublayerConnection(size, dropout), 2)
        self.size = size

    def forward(self, x, mask):
        "Follow Figure 1 (left) for connections."
        x = self.sublayer[0](x, lambda x: self.self_attn(x, x, x, mask))
        return self.sublayer[1](x, self.feed_forward)
```



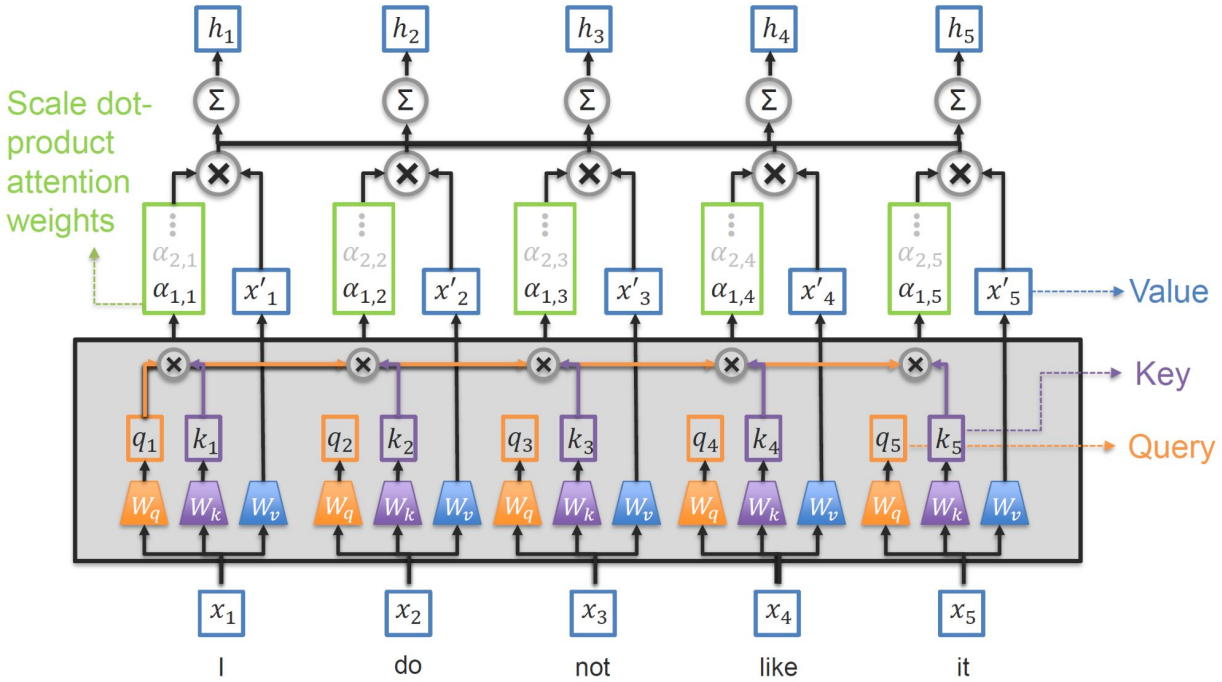
```python
class DecoderLayer(nn.Module):
    "Decoder is made of self-attn, src-attn, and feed forward (defined below)"
    def __init__(self, size, self_attn, src_attn, feed_forward, dropout):
        super(DecoderLayer, self).__init__()
        self.size = size
        self.self_attn = self_attn
        self.src_attn = src_attn
        self.feed_forward = feed_forward
        self.sublayer = clones(SublayerConnection(size, dropout), 3)

    def forward(self, x, memory, src_mask, tgt_mask):
        "Follow Figure 1 (right) for connections."
        m = memory
        x = self.sublayer[0](x, lambda x: self.self_attn(x, x, x, tgt_mask))
        x = self.sublayer[1](x, lambda x: self.src_attn(x, m, m, src_mask))
        return self.sublayer[2](x, self.feed_forward)
```

```python
class Decoder(nn.Module):
    "Generic N layer decoder with masking."
    def __init__(self, layer, N):
        super(Decoder, self).__init__()
        self.layers = clones(layer, N)
        self.norm = LayerNorm(layer.size)

    def forward(self, x, memory, src_mask, tgt_mask):
        for layer in self.layers:
            x = layer(x, memory, src_mask, tgt_mask)
        return self.norm(x)
```



Common Questions

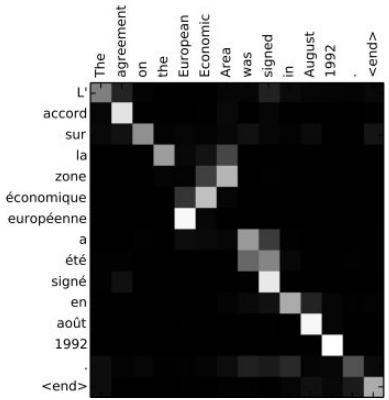
Transformer Self-Attention



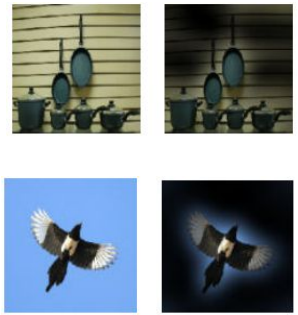
Multi-head Attention

- A single **key, query, value** matrix combination defines one attention head
  - Multiple KQVs define multiple attention heads
  - Results from all attention heads are concatenated as the final output
- Multiple attention heads allows the model to simultaneously attend to the same input sequence in different ways
- Heads are only differentiated by different random initialization of their underlying matrices
  - Attention heads can collapse into attending similar things

Attention Maps



https://arxiv.org/pdf/1409.0473.pdf



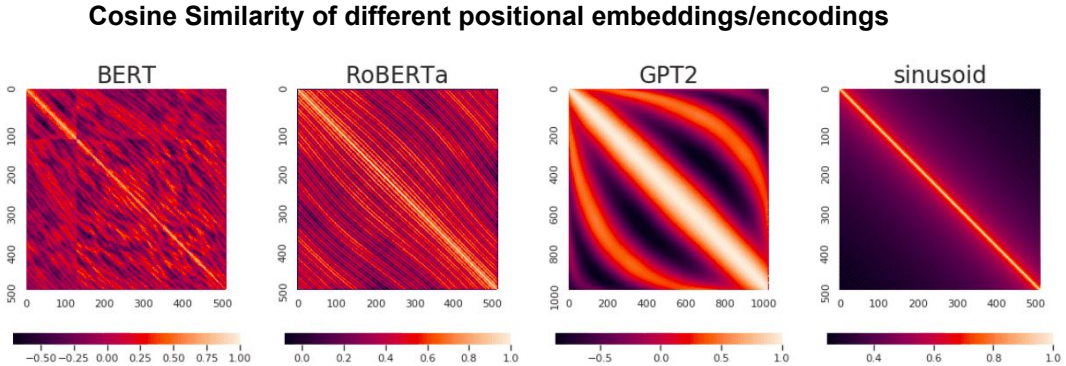
https://arxiv.org/pdf/2010.11929.pdf

Positional Encodings

- Without positional encoding, the model won't be able to distinguish the same token on different positions. Remember, a Transformer processes the entire input context parallelly.
- Naive ways of positional encodings do not work
  - Raw index of words: the magnitude of embedding increases over timestep and can dominate the resulting embedding
  - Fraction of length of input sentences: word at the same index in different-lengthed sentences will have different embeddings
- Frequency-based Positional Encoding
  - At each time t, you get a positional vector that is the same dimension as the word embedding itself. This vector is added to the word embedding.
  - $P(t + T) = M(T) P(t)$. The relationship between $P(t+T)$ and $P(t)$ does NOT change with t.
  - The series is chaotic: it has a cyclic orbit, but it never exactly repeats

Positional Embeddings

- Positional encodings are fixed functions. Positional embeddings are learned.



https://theaisummer.com/positional-embeddings/

Benefits of Eliminating Recurrence

✅

- Massively improved parallelism
- Less restrictions on informations available for enrichment at each timestep

❌

- Vanishing/exploding gradient
- Bottleneck

RNN vs Transformer: Memory

- LSTM has pretty good memory. According to a study (https://arxiv.org/pdf/1805.04623), it can remember 200 tokens of context on average and sharply distinguish 50 nearby tokens.
- Transformers use a very large context (384 tokens for BERT) in a sliding-window manner. As such, past information is available explicitly.
    - The attention matrices themselves can also be thought of as memory

RNN vs Transformer: Time Series Prediction

From
https://neuravest.net/how-transformers-with-attention-networks-boost-time-series-forecasting/

- We found RNN to be very difficult to train even after we've added LSTM (long/short memory). I believe that the main reason for our challenge was the model's inability to decide which information to save or discard when the input stream grew larger.
- RNN was not parallelizable and so training took significantly longer compared to CNN, which is based on aggregating the scores of independent learning paths and thus can be easily parallelized.
- The use of transformer architecture with attention mechanism enables the network to detect similar sequences, even though the specific image representations may be somewhat different. This in turn helps the models learn faster and generalize features better.

Masked Self-Attention Decoder

- Decoder is sequential. Each word is produced using previously decoded words as input
- When decoding at timestep t, the decoder should only attend to the t words that was already decoded
- In practice, we mask the attention score of timestep t + 1 and later to be 0, or mask the attention energy (attention score before applying SoftMax) to be -inf
- Full encoder context is always available

Transformer-based Models

Transformer variants

Generally, the Transformer architecture can be used in three different ways

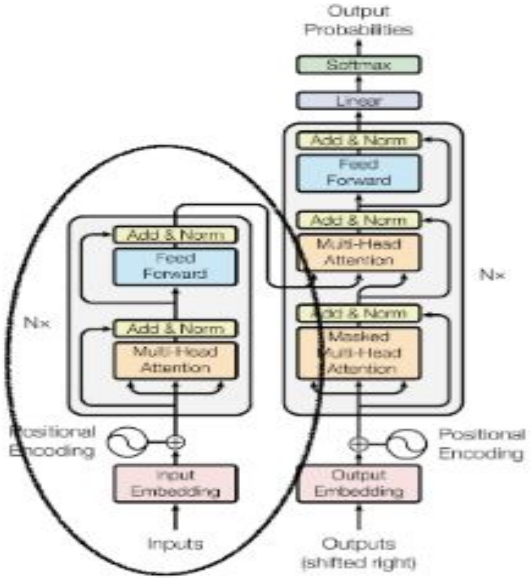
- **Encoder-Decoder**: The full Original Transformer architecture is used. This is typically used in sequence-to-sequence modeling tasks like machine translation
- **Encoder only**: Only the encoder is used and the outputs are utilized as a representation for the input sequence. This is usually used for classification or sequence labeling problems.
- **Decoder only**: Only the decoder is used, and the encoder-decoder cross-attention module is also removed. It can be used for sequence generation, such as language modeling.

From https://arxiv.org/pdf/2106.04554.pdf

Encoder only models

- Bert
- RoBERTa
- BigBird
- etc...

BERT



System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	Average
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set.[8] BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components.

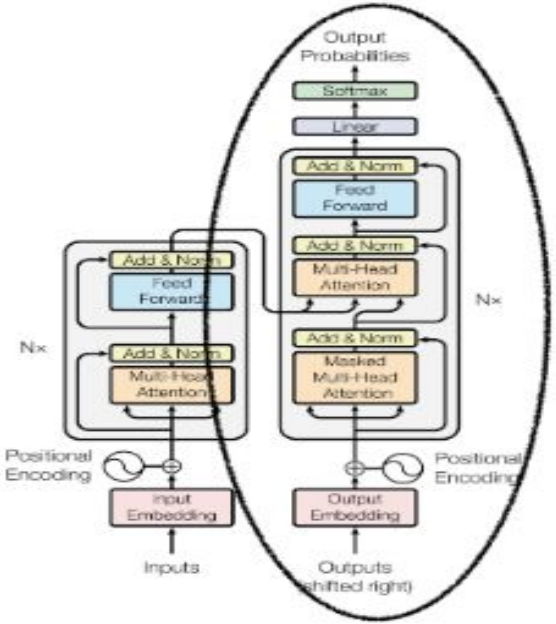
- Bert: Only uses encoder of transformer to derive word and sentence embeddings
- Trained to "fill in the blanks"
- This is *representation learning* (more next lecture)

Decoder only models

These models rely on the decoder part of the original transformer and use an attention mask so that at each position, the model can only look at the tokens before the attention heads.

GPT models are one of the most famous ones.

GPT



Alec Radford et. al., Improving Language Understanding by Generative Pre-Training

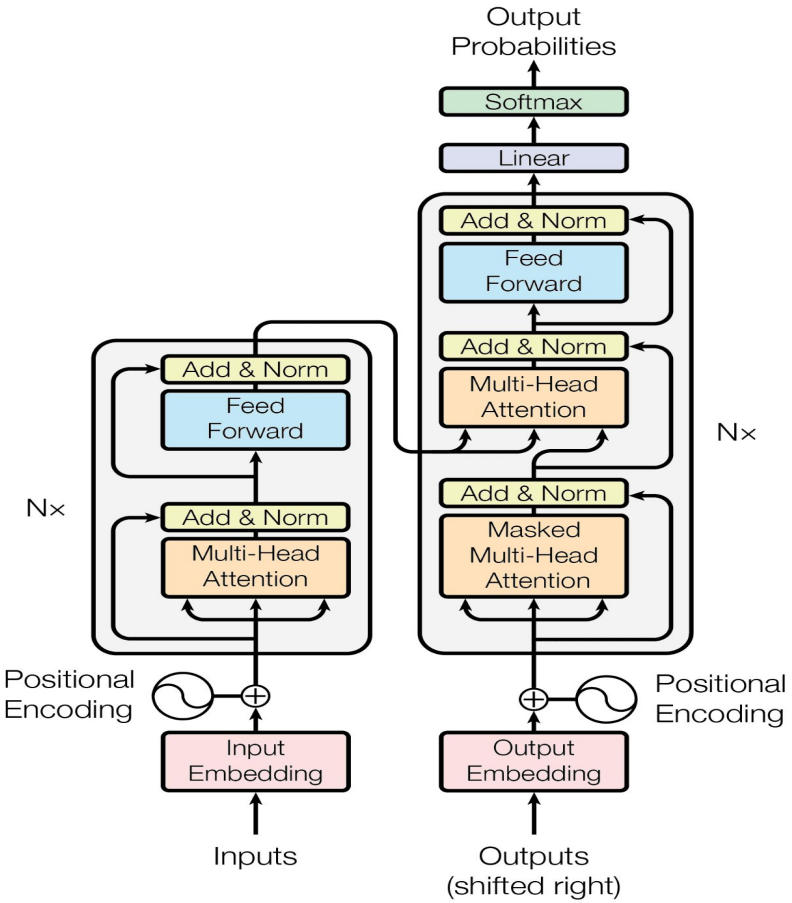
Table 5: Analysis of various model ablations on different tasks. Avg. score is a unweighted average of all the results. (*mc*= Mathews correlation, *acc*=Accuracy, *pc*=Pearson correlation)

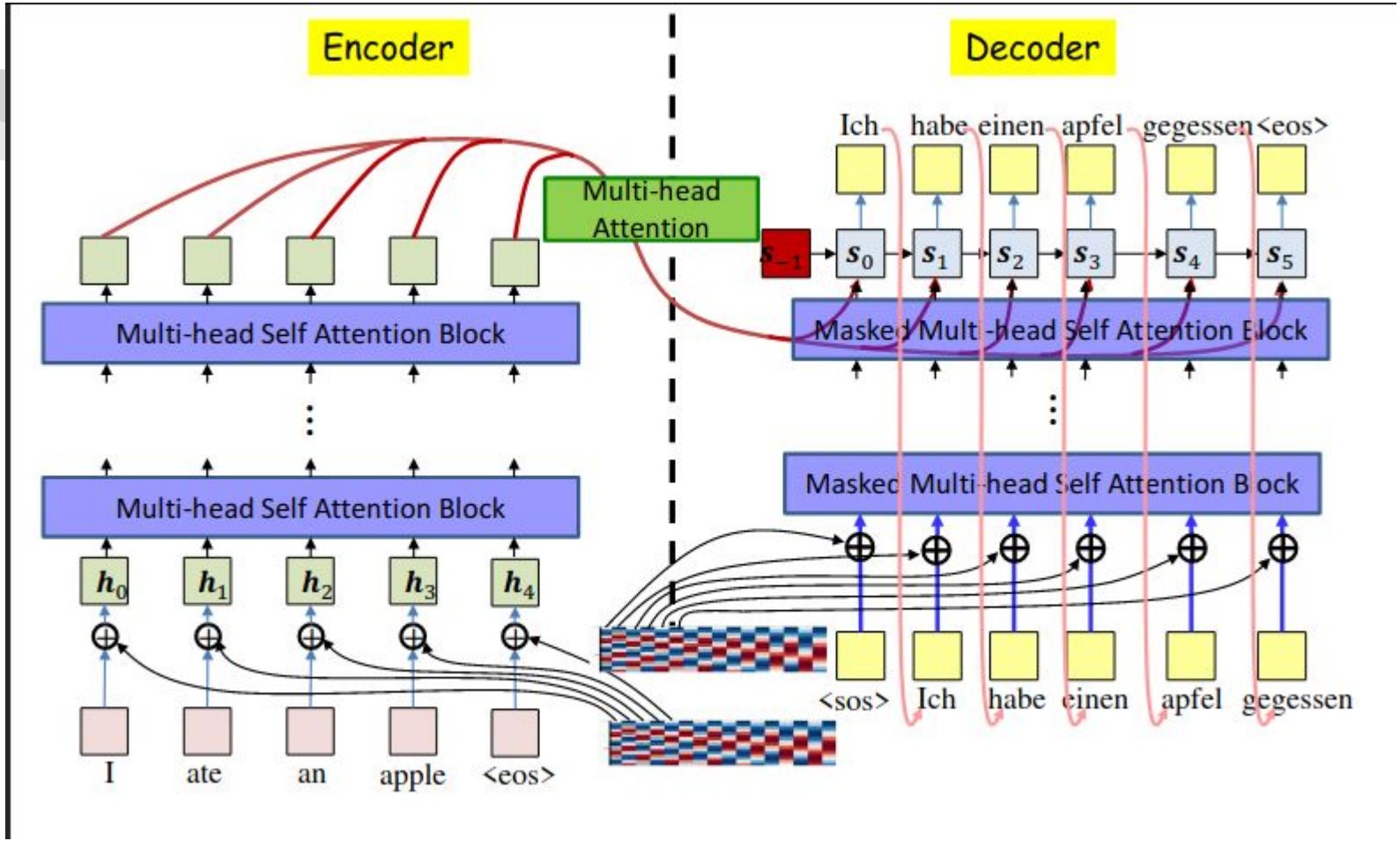
Method	Avg. Score	CoLA (mc)	SST2 (acc)	MRPC (F1)	STSB (pc)	QQP (F1)	MNLI (acc)	QNLI (acc)	RTE (acc)
Transformer w/ aux LM (full)	74.7	45.4	91.3	82.3	82.0	**70.3**	**81.8**	**88.1**	**56.0**
Transformer w/o pre-training	59.9	18.9	84.0	79.4	30.9	65.5	75.7	71.2	53.8
Transformer w/o aux LM	**75.0**	**47.9**	**92.0**	**84.9**	**83.2**	69.8	81.1	86.9	54.4
LSTM w/ aux LM	69.1	30.3	90.5	83.2	71.8	68.1	73.7	81.1	54.6

- GPT uses only the decoder of the transformer as an LM
  - "Transformer w/o aux LM"
- Large performance improvement in many tasks

Encoder-Decoder

- Vanilla Transformer
- Bart
- T5
- etc...

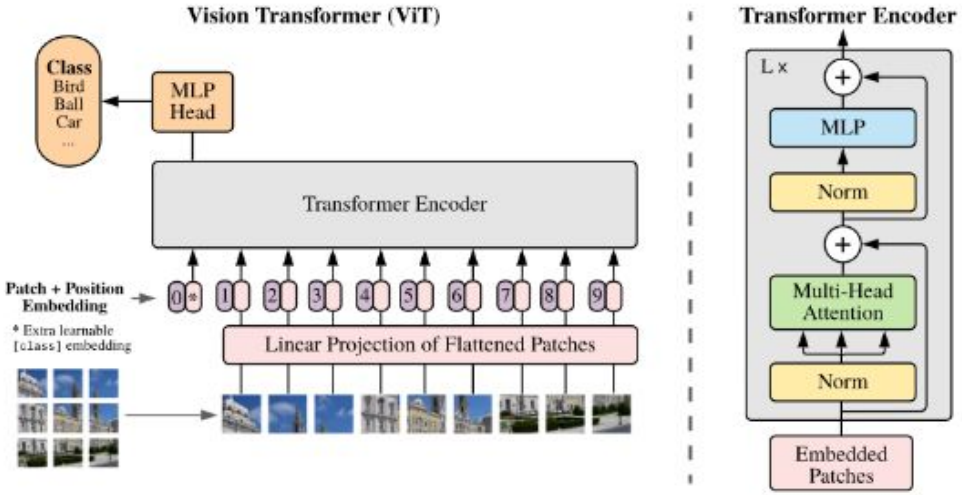



Encoder
Decoder
Multi-head Attention
Multi-head Self Attention Block
Multi-head Self Attention Block
Masked Multi-head Self Attention Block
Masked Multi-head Self Attention Block
s_{-1}
s_0
s_1
s_2
s_3
s_4
s_5
h_0
h_1
h_2
h_3
h_4
I
ate
an
apple
<eos>
<sos>
Ich
habe
einen
apfel
gegessen
Ich
habe
einen
apfel
gegessen
<eos>

Vision Transformers

- ViT
- Swin Transformer
- ViViT
- Image Transformer
- etc...

Vision Transformers



Dosovitskiy et al, *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*, 2020

- Divide your image in patches with pos. encodings
- Apply Self-Attention!

→ Sequential and image problems are similar when using transformers

Great variety of Transformer variants

There is a great variety of transformers model for different modalities such as: Text, images, Audio, Multimodal. Depending on the tasks, many architectures have been proposed for various tasks like text classification, question answering, image classification, object detection, speech recognition, visual question answering, etc...

Many pretrained transformer based models are available here:
https://huggingface.co/models

Q & A