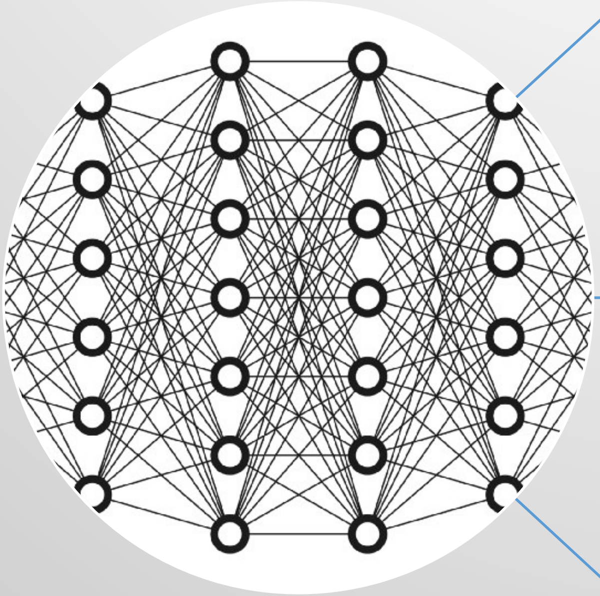
Natural Language Processing

Dr. Karen Mazidi

Part Six:
Deep Learning



Topics

- Transformers: encoders and decoders

Quizzes

- no quiz on this material

Homework

- Homework: Text classification

Encoders and decoders

- Used in machine translation and other applications
- Aka sequence-to-sequence model

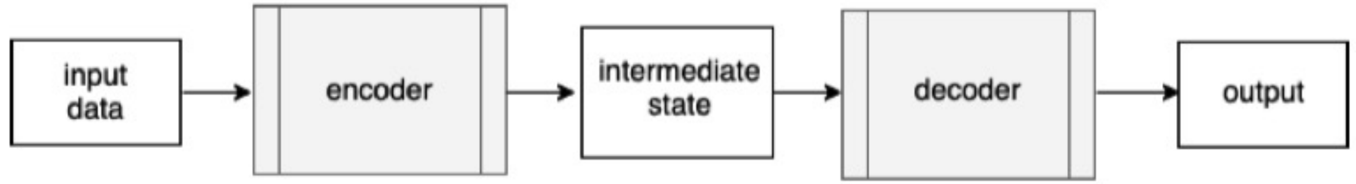


Figure 26.1: Encoder-Decoder Architecture

- Original language goes through and encoder to an intermediate tensor representation
- Then through the decoder to output in the target language

Sequence-to-sequence models

- The inputs and outputs do not have to be the same length
  - How are you <==> como estas
- Encoders and decoders are neural networks with RNN, LSTM, or GRU layers
- The decoder learns to predict the next token offset by one timestep in the future
  - This is called teacher forcing

Example

- Short English sentences into French
- Learning takes place on a character level, not word

Code 26.2.1 — Sequence to sequence. Build the encoder

```
# Define an input sequence and process it.
encoder_inputs = layers.Input(shape=(None, num_encoder_tokens))
encoder = layers.LSTM(latent_dim, return_state=True)
encoder_outputs, state_h, state_c = encoder(encoder_inputs)

# We discard ‘encoder_outputs‘ and only keep the states.
encoder_states = [state_h, state_c]
```

Example

Code 26.2.2 — Sequence to sequence. Build the decoder

```
# Set up the decoder, using ‘encoder_states‘ as initial state.
decoder_inputs = layers.Input(shape=(None, num_decoder_tokens))
# We set up our decoder to return full output sequences,
# and to return internal states as well. We don’t use the
# return states in the training model, but we will use them in inference.
decoder_lstm = layers.LSTM(latent_dim, return_sequences=True,
       return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_inputs,
                                     initial_state=encoder_states)
decoder_dense = layers.Dense(num_decoder_tokens, activation=’softmax’)
decoder_outputs = decoder_dense(decoder_outputs)
```

Example

- Put the model together

Code 26.2.3 — Sequence to sequence. Build the model

```
# Define the model that will turn
# 'encoder_input_data' & 'decoder_input_data' into 'decoder_target_data'
model = models.Model([encoder_inputs, decoder_inputs], decoder_outputs)
```

Example

- Results after training 100 epochs
- The character-level approach demonstrates how this is not at all like human language learning, this system just learning to map inputs to outputs
- In contrast, human speech varies our mappings: how are you? How’s it going? Etc.

Code Examples

- seq-2-seq for machine translation

Attention is all you need

- 2017 NeurIPS paper: https://arxiv.org/abs/1706.03762

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

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

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

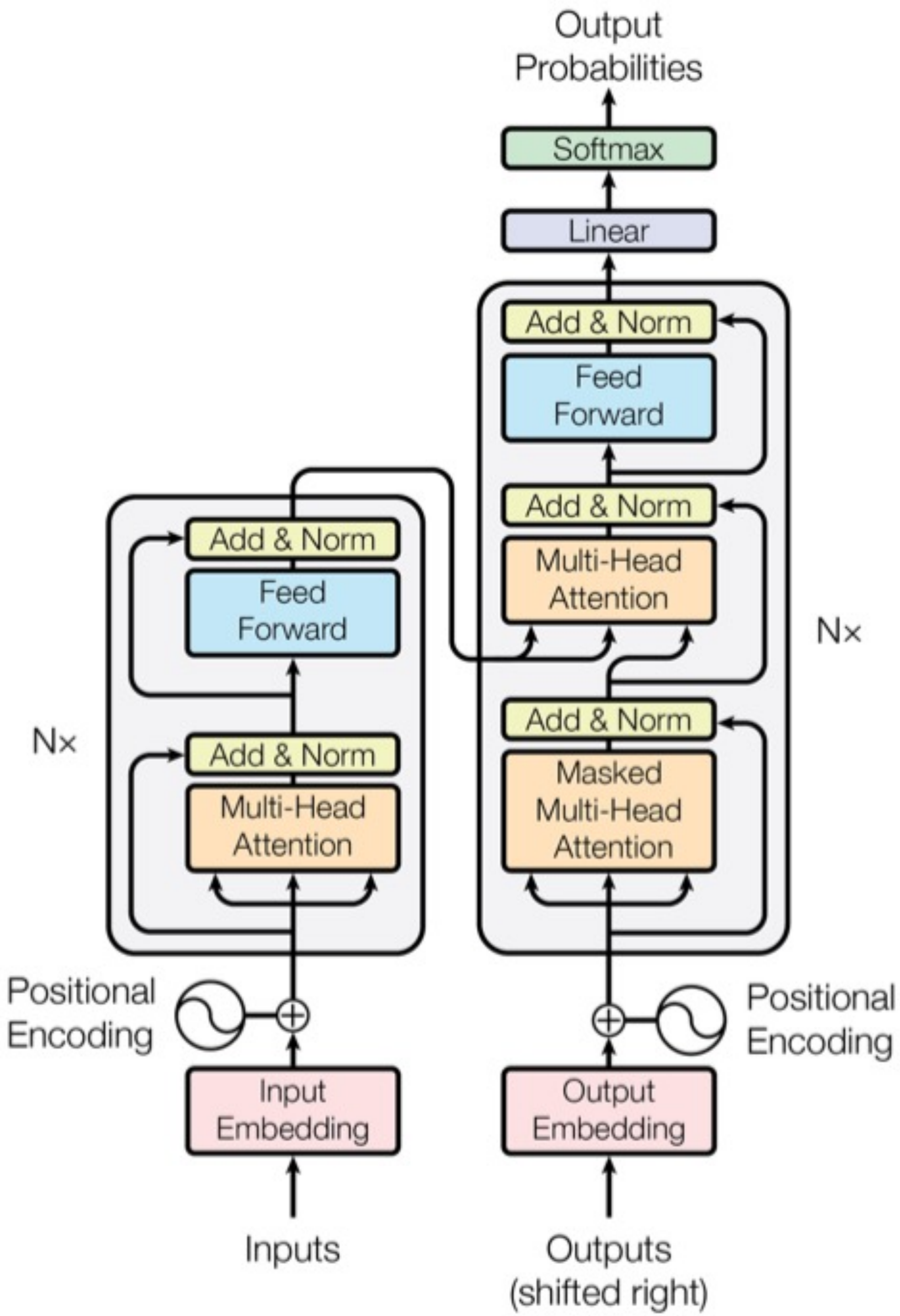
- Prior sequence transduction models based on encoder-decoder architecture connected via an attention mechanism
- Authors simplify the architecture by removing recurrence and convolutions, using only attention
- This model is called the Transformer
- Transformer improved results on 2 machine translation tasks, and trained faster
- Transformer also generalized to another task: English constituency parsing

Background

- self-attention relates different positions of a single sequence in order to compute a representation of the sequence
- self-attention has been successfully used in a variety of tasks:
  - reading comprehension
  - abstractive summarization
  - textual entailment
  - learning task-independent sentence representaitons

Model architecture

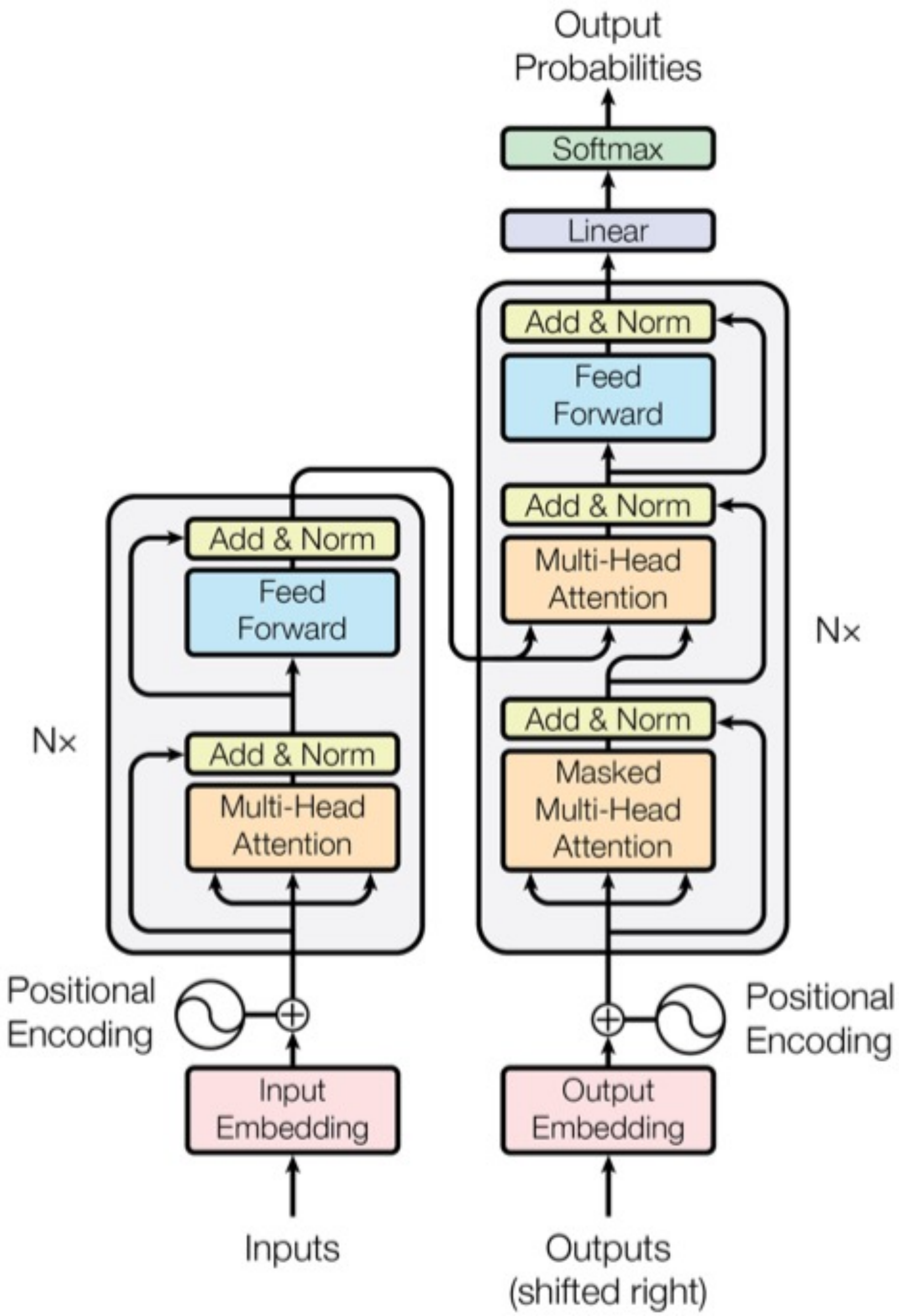
- Transformer uses stacked self-attention and point-wise fully connected layers for both the encoder (left) and decoder (right)



Model architecture

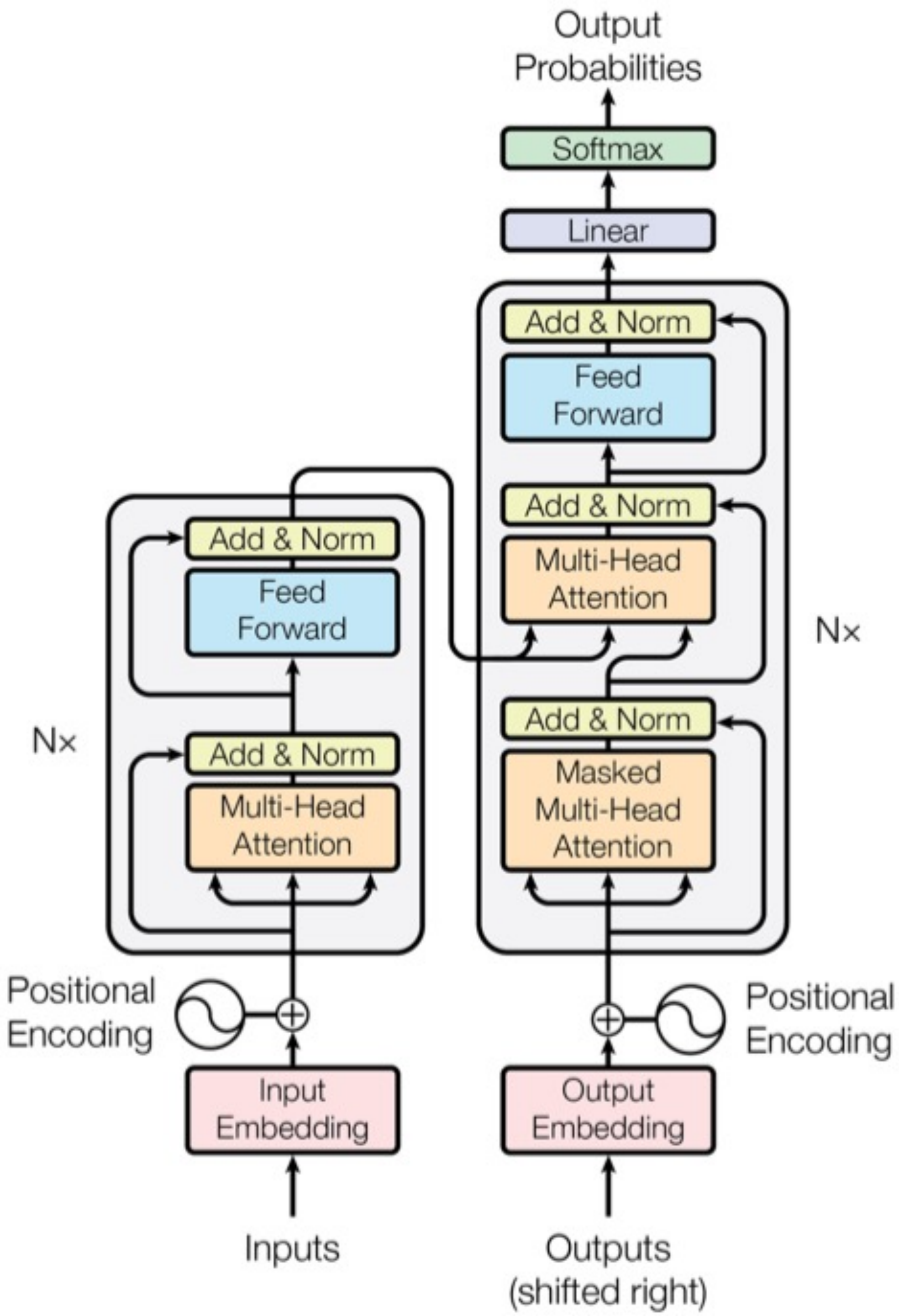
- the encoder (left) is a stack of N=6 identical layers
- each layer has two sub-layers
  - a multi-head self-attention mechanism
  - a simple position-wise connected FFNN
  - a residual connection surrounds the two sub-layers, followed by layer normalization
  - all sub-layers as well as embedding layers output dimension = 512

Layer normalization normalizes the distributions of intermediate layers, enabling smoother gradients, faster training, and better generalization.



Model architecture

- the decoder (right) also has N=6 identical layers
- adds a 3rd sublayer in each decoded layer to perform multi-head attention over the output of the encoder stack
- also has residual connections around sub layers, followed by layer normalization
- the output embeddings are offset by one position so that predictions for position i can only depend on known outputs at positions < i

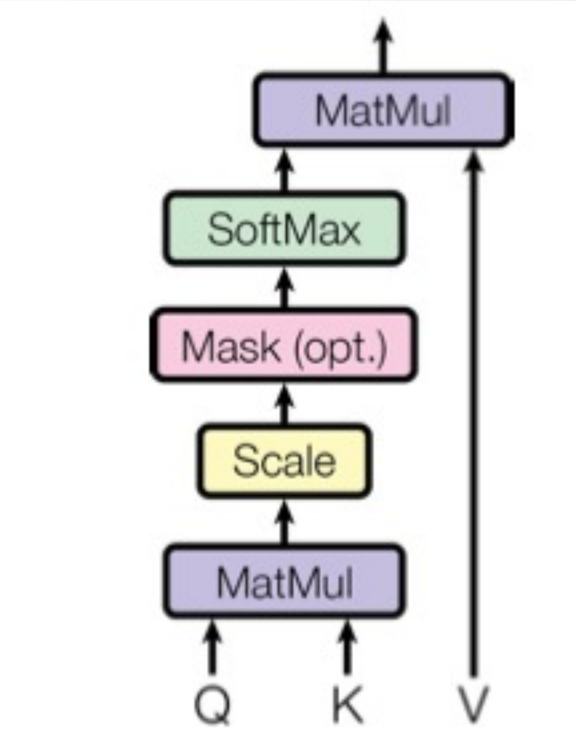


Attention

- can be implemented as mapping a query vector to key-value pair vectors
- the output is a weighted sum of the values
- the weight for each value is computed by a compatibility function of the query with the corresponding key

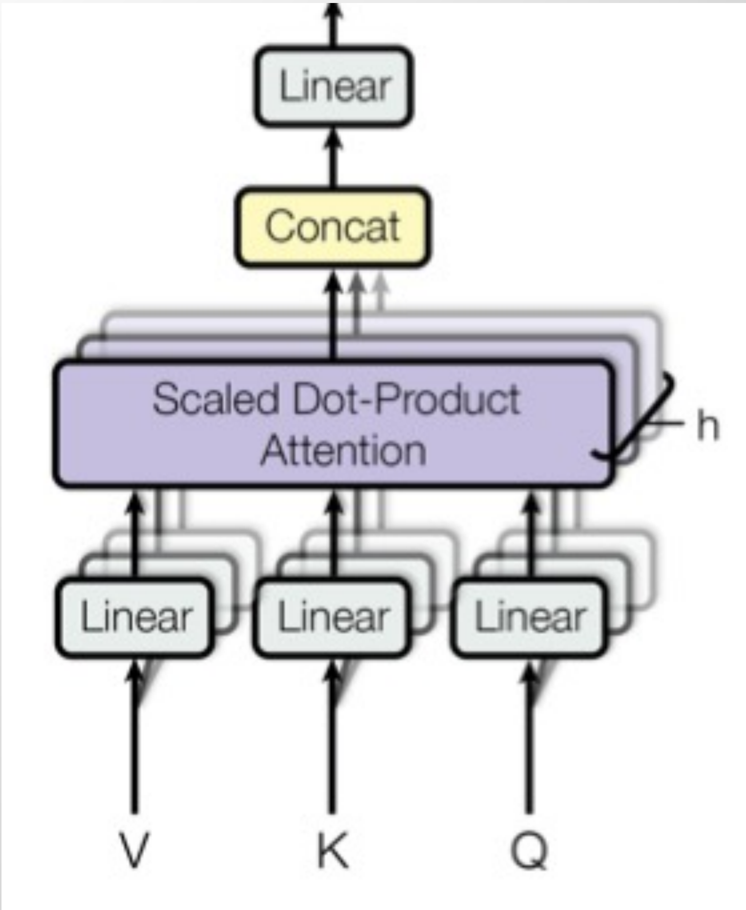
Scaled dot-product attention

- input is query Q and key K
- compute the dot products of the query with all keys, divide each by sqrt(dim(k)), apply a softmax to obtain the weights on the values
- this was implemented by combining queries packed into a matrix, and also packing K and V
- dot-product implementations are computationally efficient due to highly optimized matmul code
- the dot products are scaled to prevent them growing large in magnitude



Multi-head attention

- instead of performing a single attention function, the authors linearly projected the queries, keys and values h times with different, learned linear projections to different dimensions
- these h projects are performed in parallel
- the outputs are concatenated and again projected, resulting in final values
- multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions
- authors used h=8 parallel attention layers, “heads”



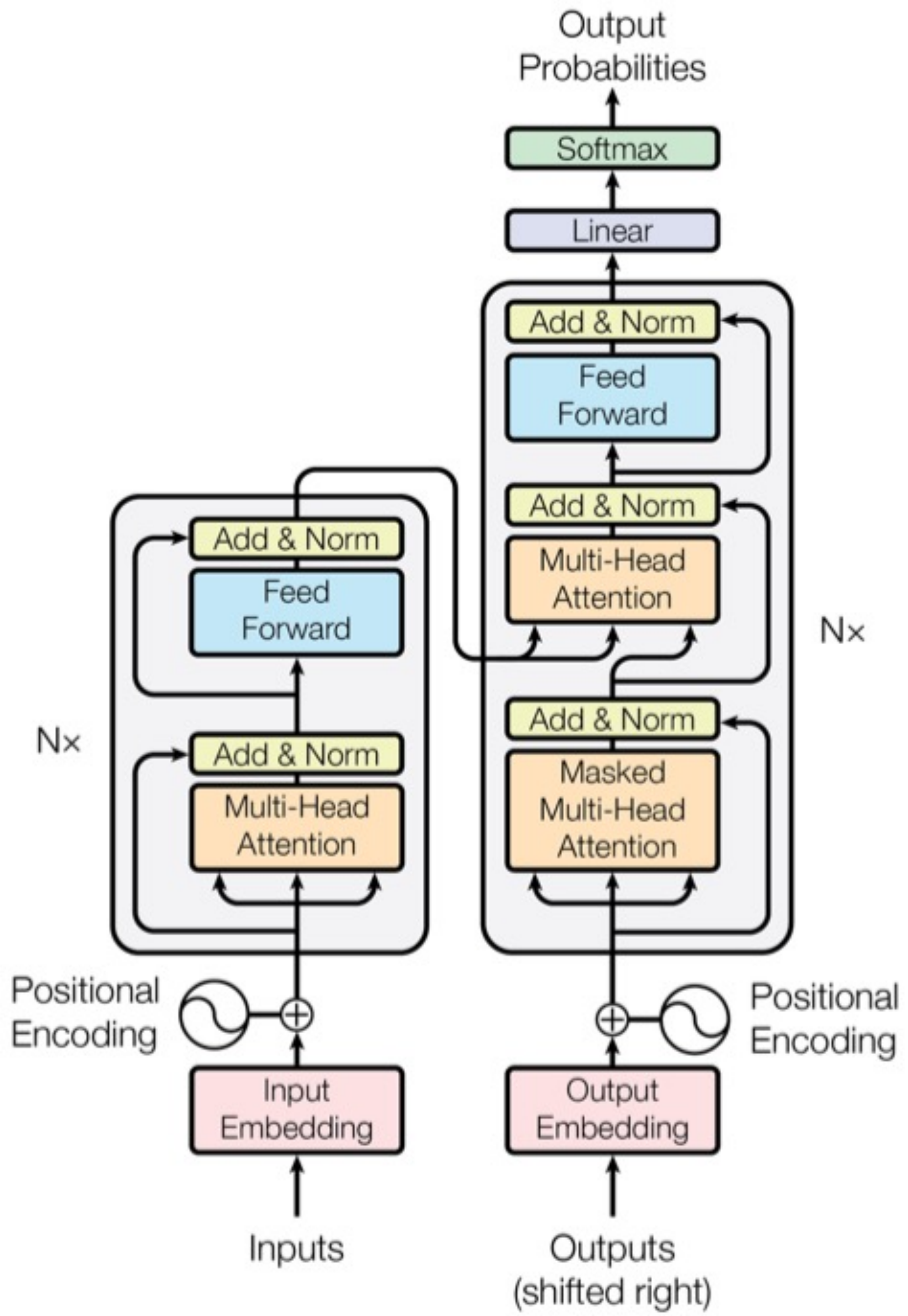
Transformer

- the Transformer uses multi-head attention in 3 ways:

1. in ‘encoder-decoder’ layers, the queries come from the previous decoder layer, and the keys and values come from the output of the encoder; this allows every position in the decoder to attend over all positions in the input sequence
2. in the encoder self-attention layers, the keys, values, and queries come from the output of the previous layer in the encoder; each position in the encoder can attend to all positions in the previous layer of the encoder
3. self-attention layers in the decoder allow each position in the decoder to attend to all positions in the decoder up to and including that position; illegal connections are masked out

Model architecture

- the Feed Forward (blue) layers consist of two FFNNs with ReLU activation in between
- uses different parameters from layer to layer
  - think of this as two "convolutions" with kernel size 1
- uses learned embeddings
- learned linear transformation and softmax convert the decoder output to predicted next-token probabilities
- adds positional encodings to the input embeddings at the bottom of the stacks



Complexity

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Why self-attention

1. improves total computational complexity per layer
2. is more easily parallelized
3. shorter lengths to travel to capture long-range dependencies

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Training

- WMT 2014 English-German data (4.5 million sentence pairs) and English-French data (36 million sentence pairs)
- one machine with 8 NVIDIA P100 GPUs
  - about 12 hours for the base models
  - 3.5 days for the big models
- Adam optimizer
- regularization:
  - dropout for each sub-layer
  - dropout for the sums of the embeddings and positional encodings
  - label smoothing of 0.1, which hurts perplexity but improves accuracy and BLEU score

BLEU

- BiLingual Evaluation Understudy
- automatically evaluate machine translation results, comparing machine output to reference translations
- range [0, 1] where 1 is a perfect match (even human translators don’t achieve this)

BLEU Score	Interpretation
< 10	Almost useless
10 - 19	Hard to get the gist
20 - 29	The gist is clear, but has significant grammatical errors
30 - 40	Understandable to good translations
40 - 50	High quality translations
50 - 60	Very high quality, adequate, and fluent translations
> 60	Quality often better than human

Results

German:

- the big Transformer model improved BLEU score by > 2.0, creating a new SOTA
- base model also improved SOTA at a faction of the training cost of competitive models

French:

- the big Transformer model also improved SOTA

Results

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Constituency parsing

- despite having do task-specific tuning of the model, the model performs well

Conclusion

- future work is to apply to the model to other tasks
- Code available here: https://github.com/tensorflow/tensor2tensor

AllenNLP GPT-2 language model

- https://demo.allennlp.org/next-token-lm

GPT-4

- https://openai.com/research/gpt-4
- Technical report: https://cdn.openai.com/papers/gpt-4.pdf

- Response from tech community: https://futureoflife.org/open-letter/pause-giant-ai-experiments/

Essential points to note

- Sequence-to-sequence models use a neural network to encode text, then use another neural network to decode that encoding
- this can be used for machine translation, question answering systems, text summarization and more
- transformers use an attention mechanism to enable seq-2-seq models without recurrence

Next class

Presentations