Deep Learning
Transformers and GNNs

Recap

- We've seen
  - MLPs,
  - CNNs as scanning MLPs,
  - Recurrent nets as MLPs with time recurrence
  - Attention models
- What are the generalizations and extensions
  - Attention : Transformers
  - CNN : Graph networks
  - Autoencoders : Generative models – VAEs and GANs

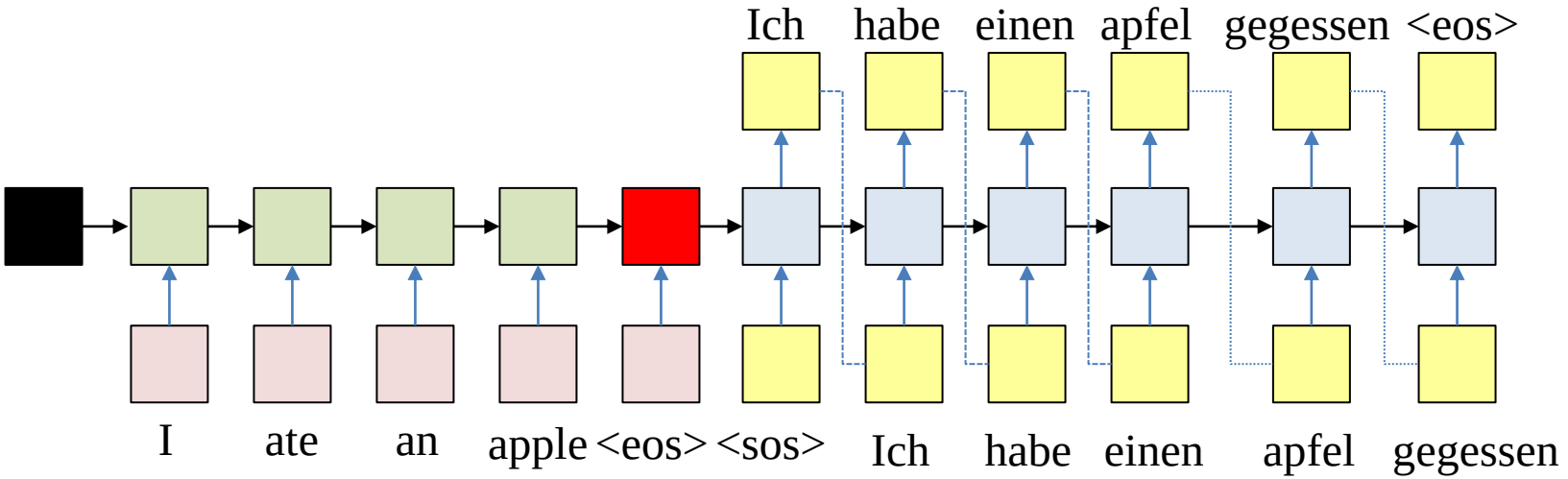
Topics for the week

- Transformers
- GNNs
- Representation learning
- Autoencoders

Topics for the week

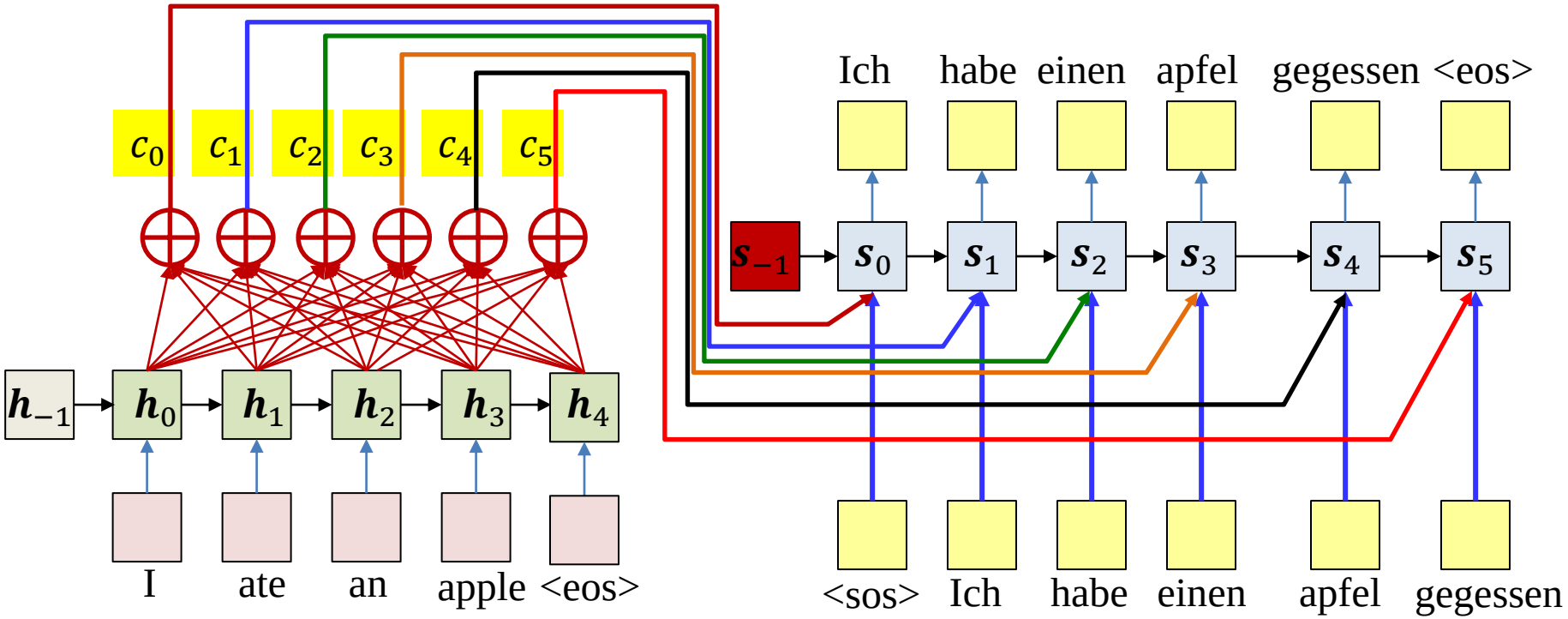
- Transformers
- GNNs
- Representation learning
- Autoencoders

Recap: Seq2Seq models



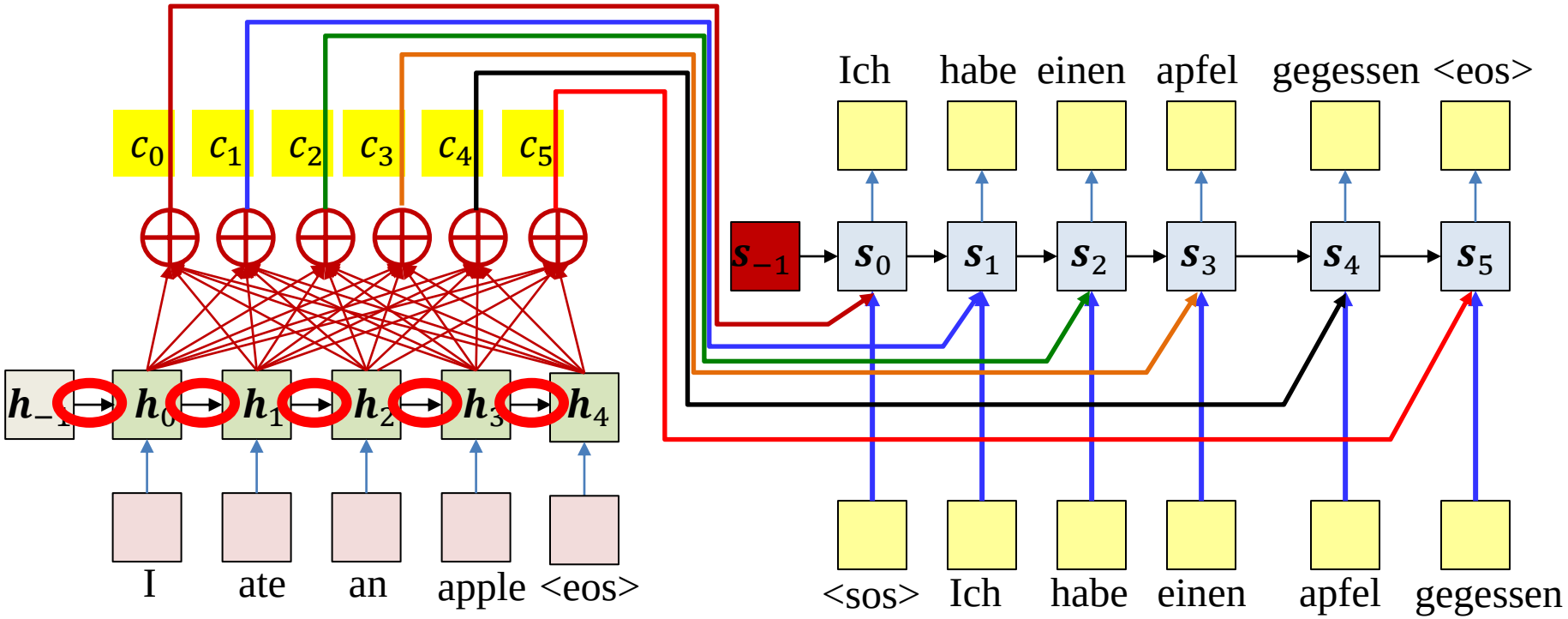
- The input sequence feeds into a recurrent structure
- The input sequence is terminated by an explicit <eos> symbol
  - The hidden activation at the <eos> "stores" all information about the sentence
- Subsequently a *second* RNN uses the hidden activation as initial state to produce a sequence of outputs

Recap: Attention Models



- Encoder recurrently produces hidden representations of input word sequence
- Decoder recurrently generates output word sequence
  - For each output word the decoder uses a weighted average of the hidden input representations as input “context”, along with the recurrent hidden state and the previous output word

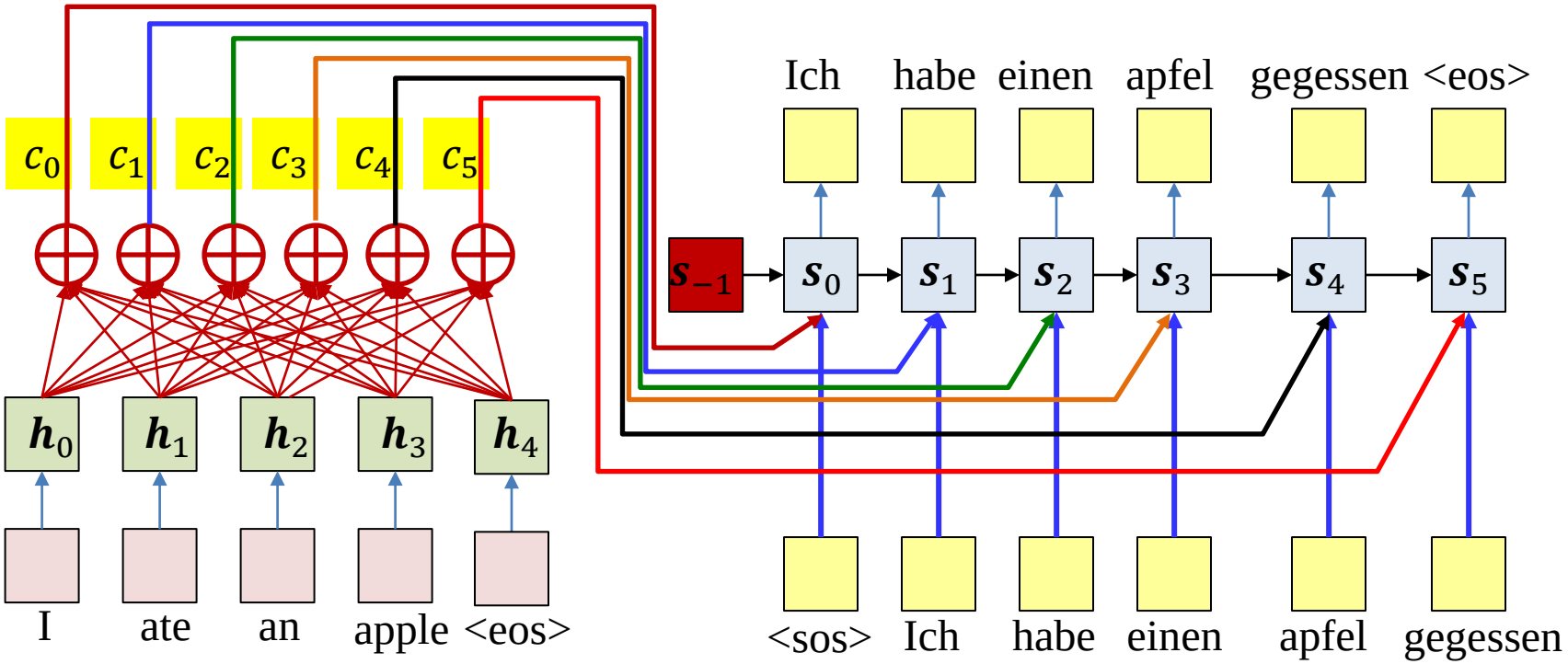
Recap: Attention Models



- Problem: Because of the recurrence, the hidden representation for any word is also influenced by *all* preceding words
  - The decoder is actually paying attention to the sequence, and not just the word

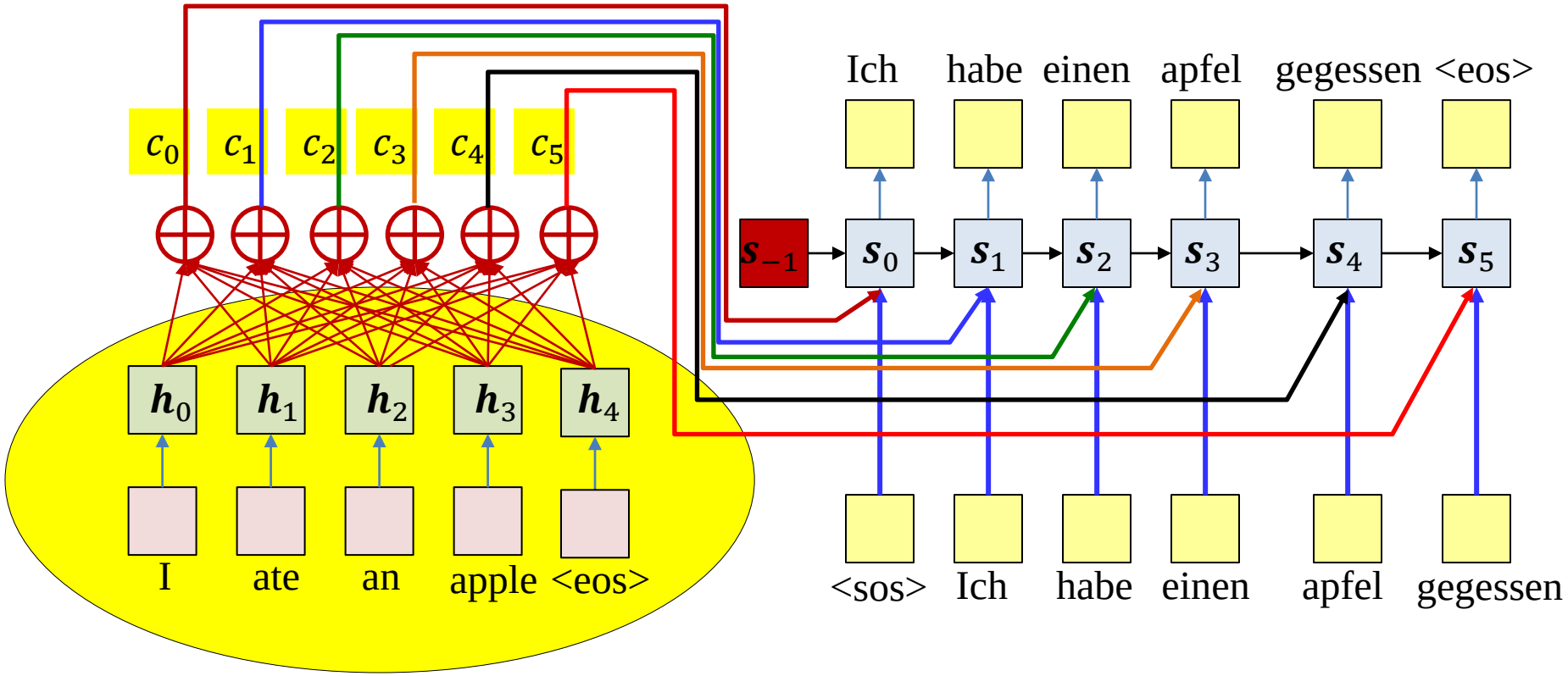
- If the decoder is automatically figuring out which words of the input to attend to at each time, is recurrence in the input even necessary?

Non-recurrent encoder



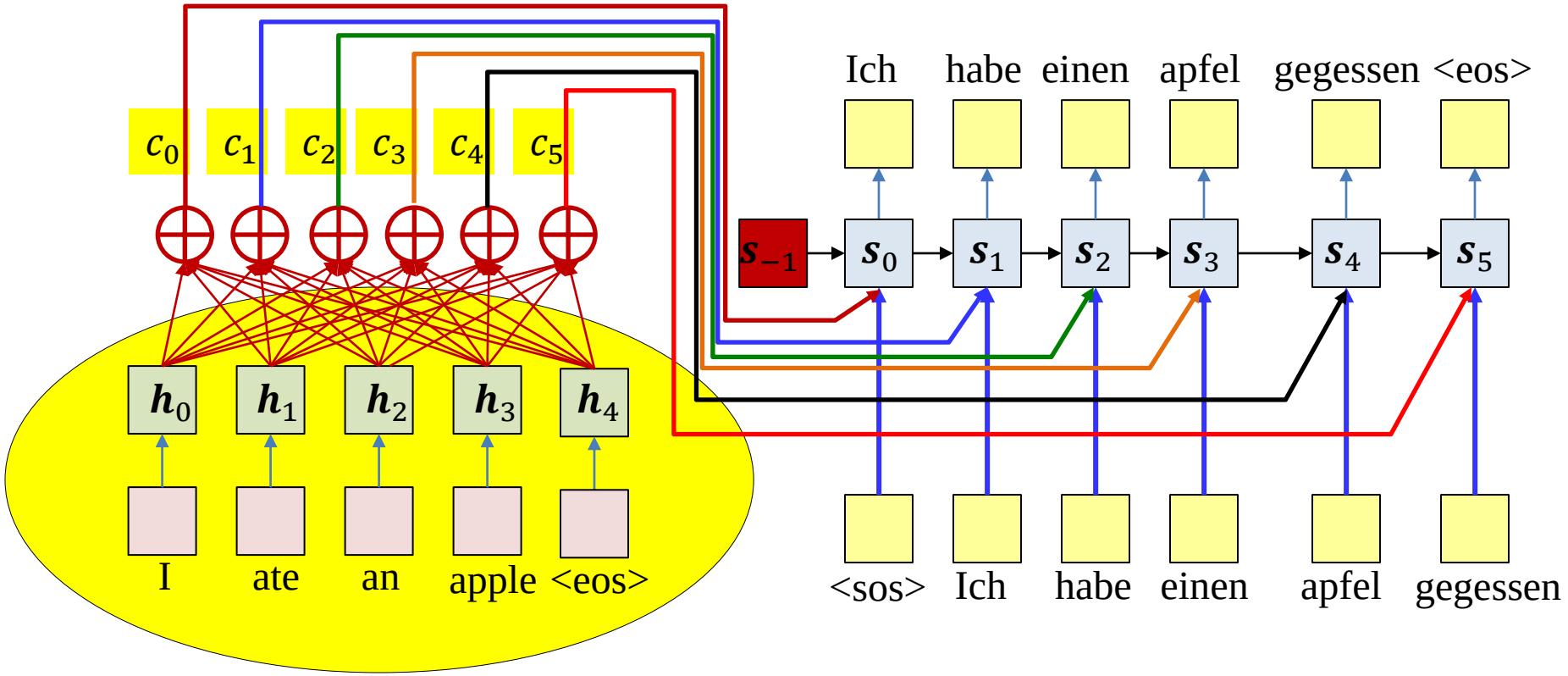
- Modification: Let us eliminate the recurrence in the encoder

Non-recurrent encoder



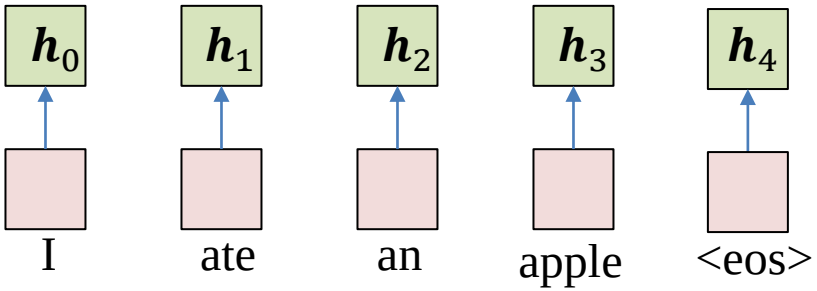
- But this will eliminate *context-specificity* in the encoder embeddings
  - The embedding for "an" must really depend on the remaining words
    - It could be translated to "ein", "einer", or "eines" depending on the context.
- Solution: Use the attention framework itself to introduce context-specificity in embeddings

Recap: Non-recurrent encoder



- The encoder in a sequence-to-sequence model can be composed without recurrence.
- Use the attention framework itself to introduce context-specificity in embeddings
  - "Self" attention

Self attention



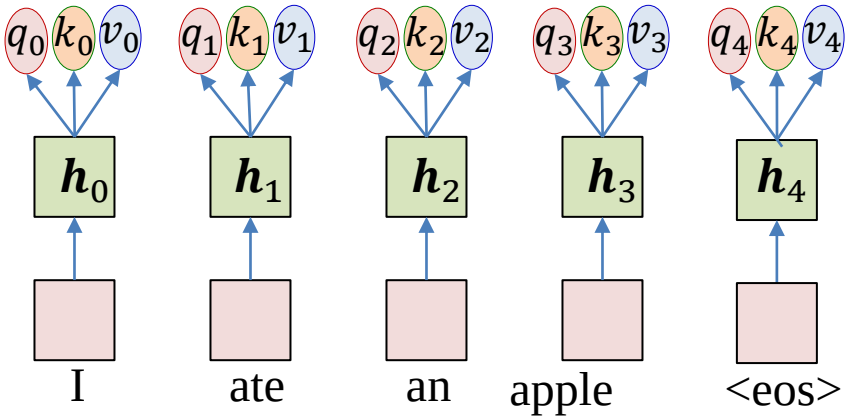
- First, for every word in the input sequence we compute an initial representation
  - E.g. using a single MLP layer

Self attention

$q_i = \boldsymbol{W}_q h_i$

$k_i = \boldsymbol{W}_k h_i$

$v_i = \boldsymbol{W}_v h_i$



- Then, from each of the hidden representations, we compute a query, a key, and a value.
  - Using separate linear transforms
  - The weight matrices W_q, W_k and W_v are learnable parameters

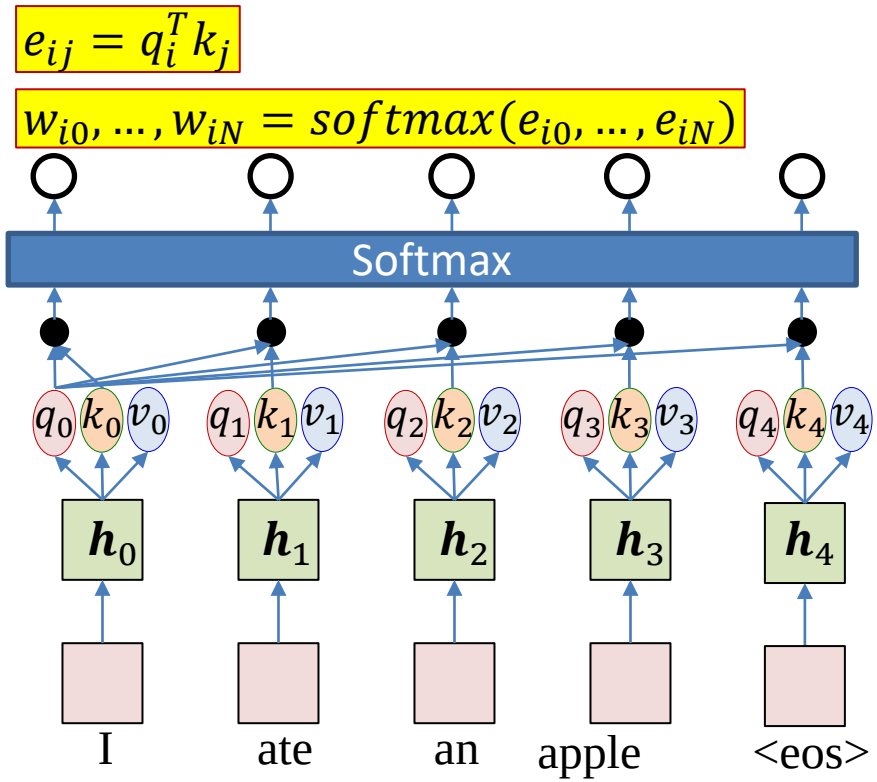
Self Attention

$q_i = \boldsymbol{W}_q h_i$

$k_i = \boldsymbol{W}_k h_i$

$v_i = \boldsymbol{W}_v h_i$

$w_{ij} = attn(q_i, k_{0:N})$



- For each word, we compute an attention weight between that word and all other words
  - The raw attention of the ith word to the jth word is a function of query q_i and key k_j
  - The raw attention values are put through a softmax to get the final attention weights

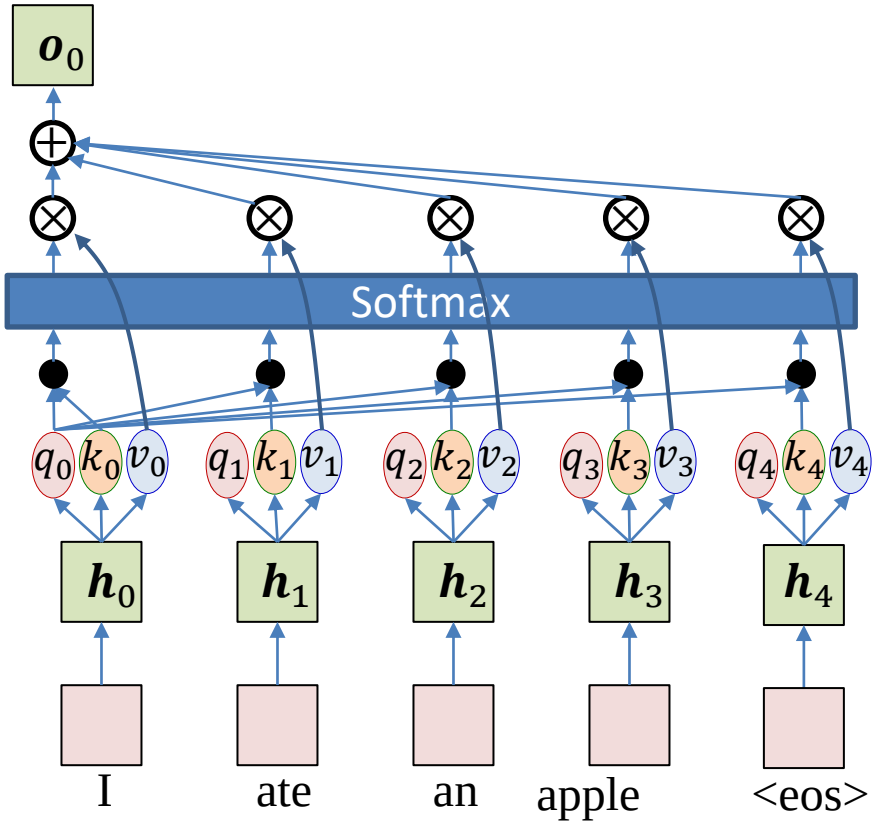
$q_i = \boldsymbol{W}_q h_i$

$k_i = \boldsymbol{W}_k h_i$

$v_i = \boldsymbol{W}_v h_i$

$w_{ij} = attn(q_i, k_{0:N})$

$$o_i = \sum_j w_{ij} v_j$$



- The updated representation for the word is the attention-weighted sum of the values for all words
  - Including itself

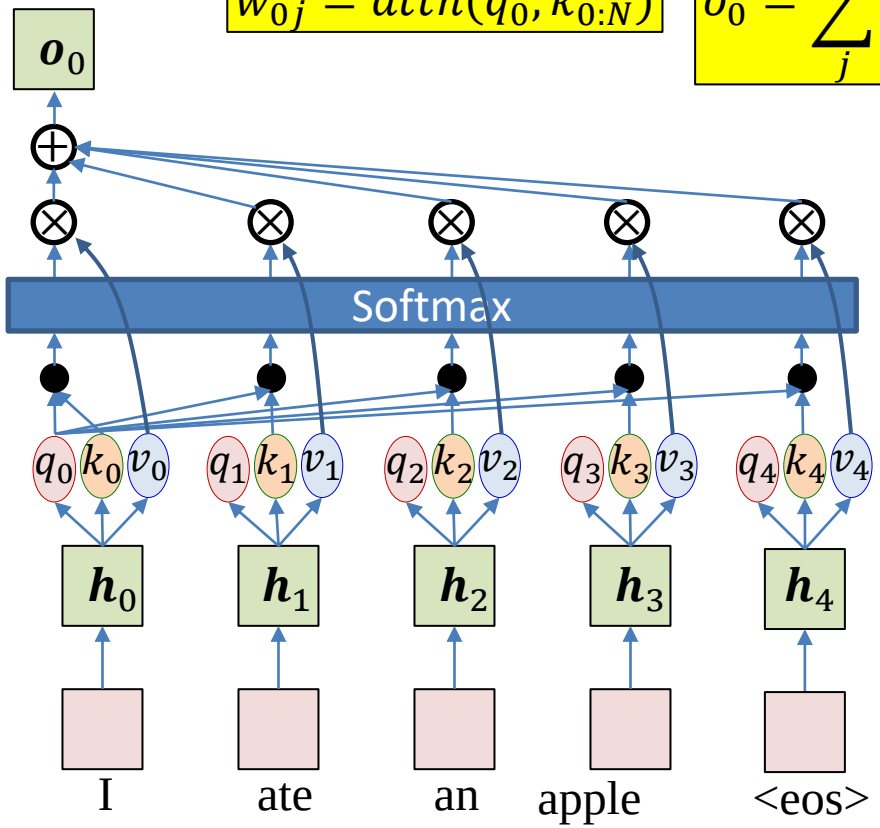
$q_i = \boldsymbol{W}_q h_i$

$k_i = \boldsymbol{W}_k h_i$

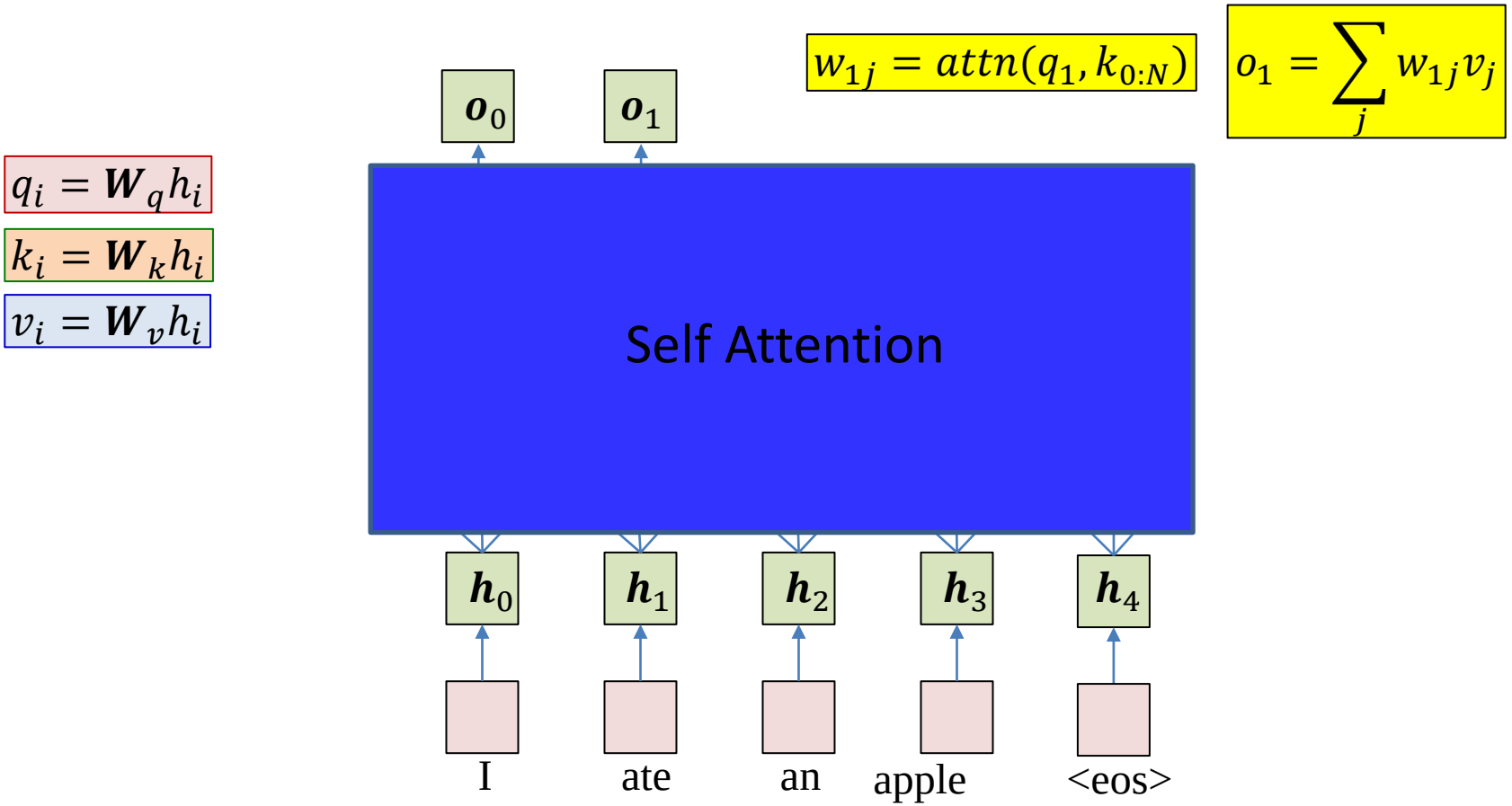
$v_i = \boldsymbol{W}_v h_i$

$w_{0j} = attn(q_0, k_{0:N})$

$o_0 = \sum_j w_{0j} v_j$



- Compute query-key-value sets for every word
- For each word
  - Using the query for that word, compute attention weights for all words using their keys
  - Compute updated representation for the word as attention-weighted sum of values of all words



- Compute query-key-value sets for every word
- For each word
  - Using the query for that word, compute attention weights for all words using their keys
  - Compute updated representation for the word as attention-weighted sum of values of all words

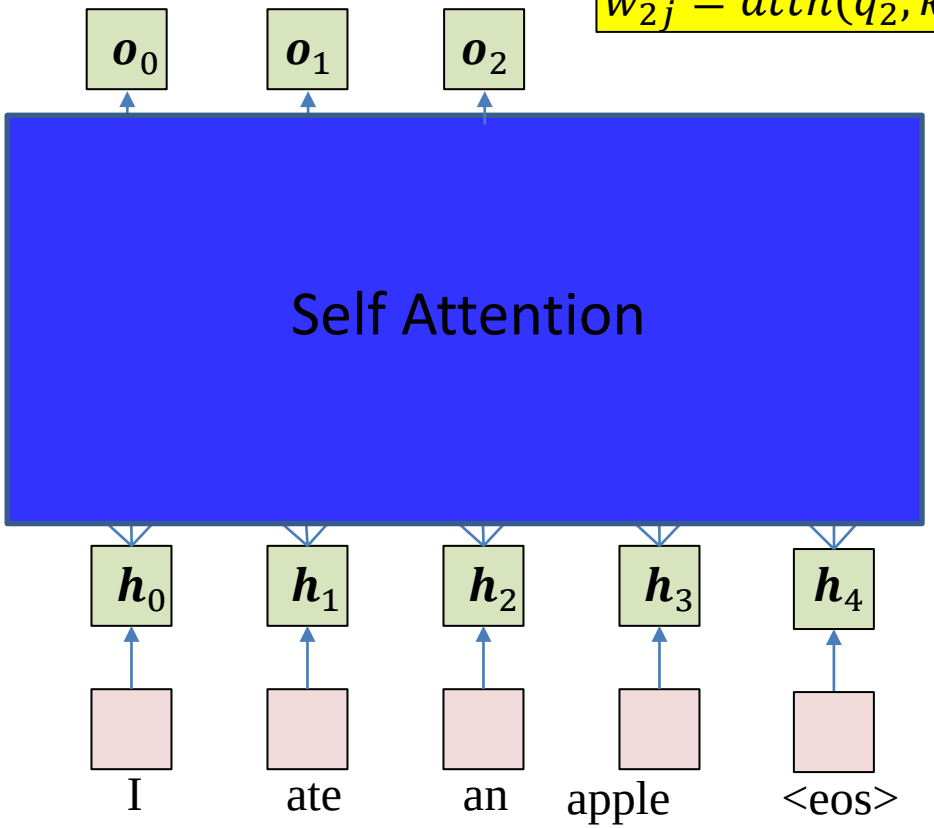
$q_i = \boldsymbol{W}_q h_i$

$k_i = \boldsymbol{W}_k h_i$

$v_i = \boldsymbol{W}_v h_i$

$w_{2j} = attn(q_2, k_{0:N})$

$o_2 = \sum_j w_{2j} v_j$

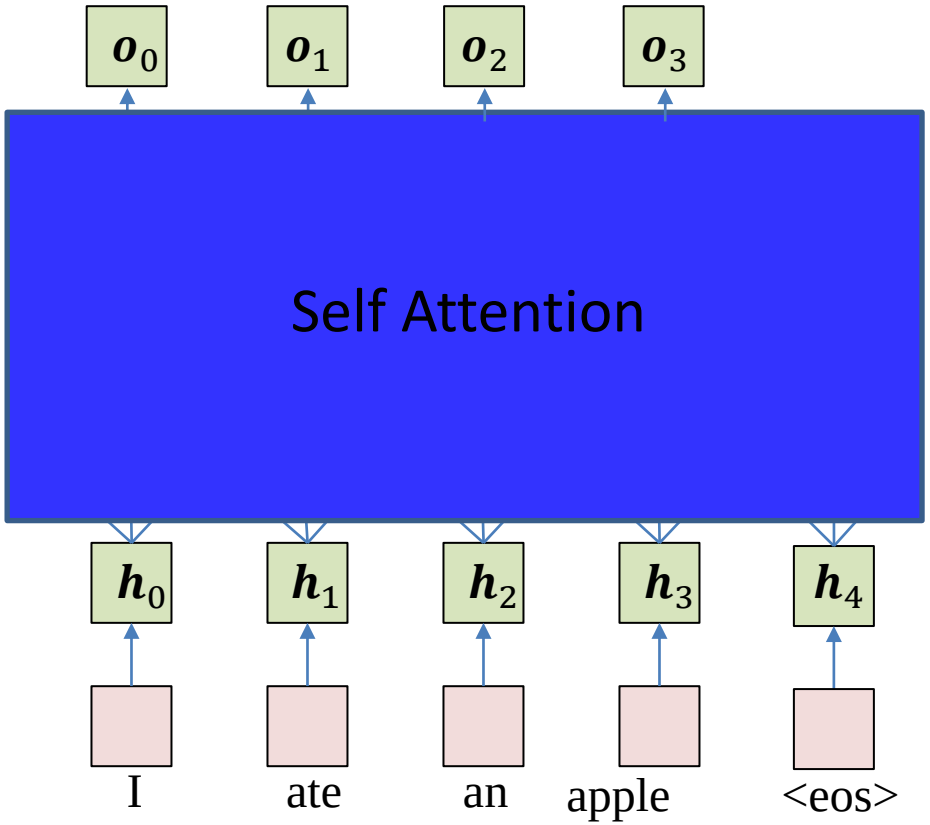


- Compute query-key-value sets for every word
- For each word
  - Using the query for that word, compute attention weights for all words using their keys
  - Compute updated representation for the word as attention-weighted sum of values of all words

$q_i = \boldsymbol{W}_q h_i$

$k_i = \boldsymbol{W}_k h_i$

$v_i = \boldsymbol{W}_v h_i$

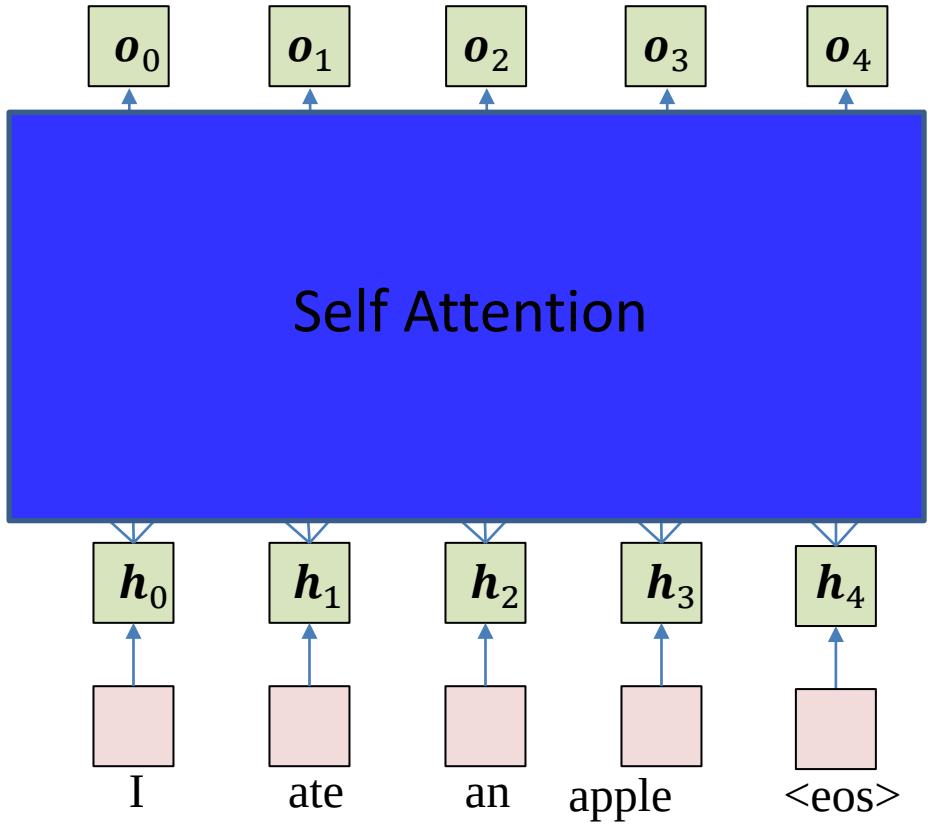


- Compute query-key-value sets for every word
- For each word
  - Using the query for that word, compute attention weights for all words using their keys
  - Compute updated representation for the word as attention-weighted sum of values of all words

$q_i = \boldsymbol{W}_q h_i$

$k_i = \boldsymbol{W}_k h_i$

$v_i = \boldsymbol{W}_v h_i$



- Compute query-key-value sets for every word
- For each word
  - Using the query for that word, compute attention weights for all words using their keys
  - Compute updated representation for the word as attention-weighted sum of values of all words

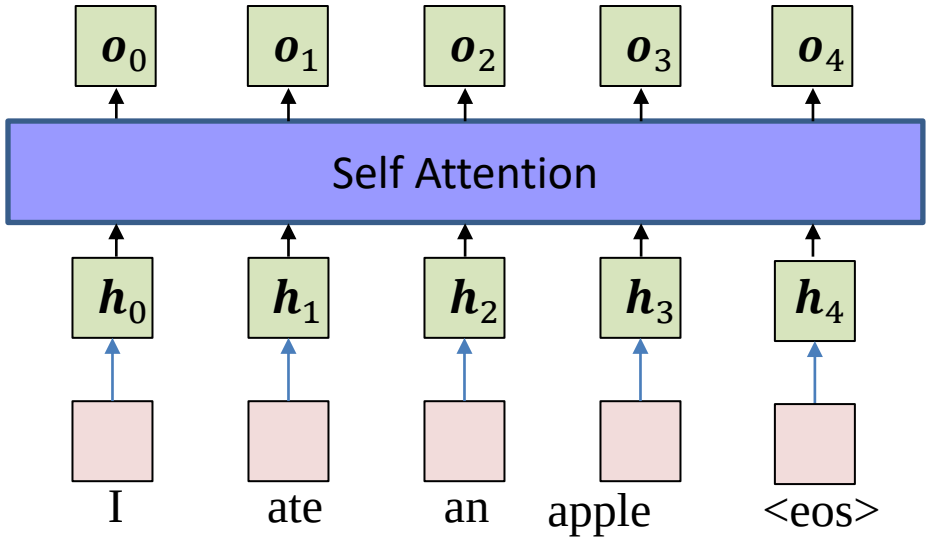
$q_i = \boldsymbol{W}_q h_i$

$k_i = \boldsymbol{W}_k h_i$

$v_i = \boldsymbol{W}_v h_i$

$w_{ij} = attn(q_i, k_{0:N})$

$o_i = \sum_j w_{ij} v_j$



- Compute query-key-value sets for every word
- For each word
  - Using the query for that word, compute attention weights for all words using their keys
  - Compute updated representation for the word as attention-weighted sum of values of all words

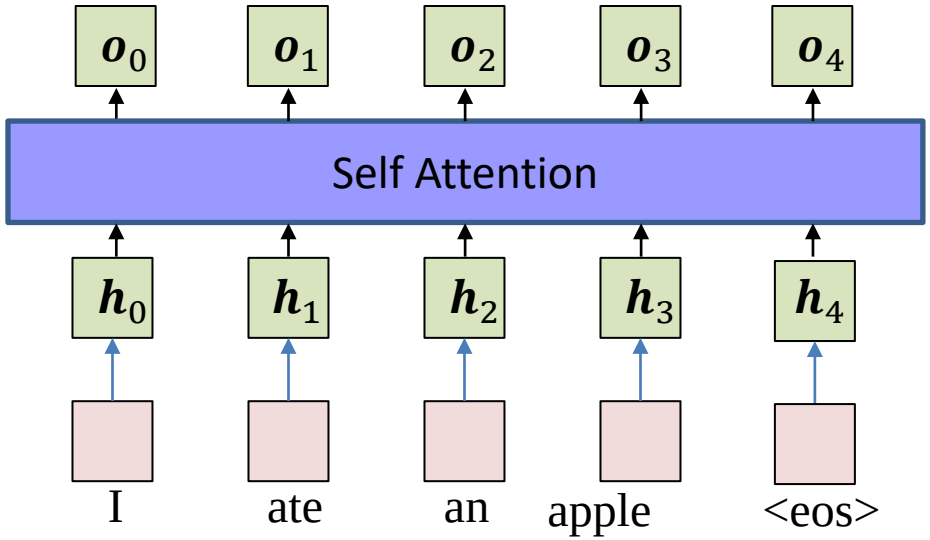
$$w_{ij} = attn(q_i, k_{0:N})$$

$$o_i = \sum_j w_{ij} v_j$$

$q_i = \boldsymbol{W}_q h_i$

$k_i = \boldsymbol{W}_k h_i$

$v_i = \boldsymbol{W}_v h_i$



This is a "single-head" self-attention block

$$q_i^a = \boldsymbol{W}_q^a h_i$$

$$k_i^a = \boldsymbol{W}_k^a h_i$$

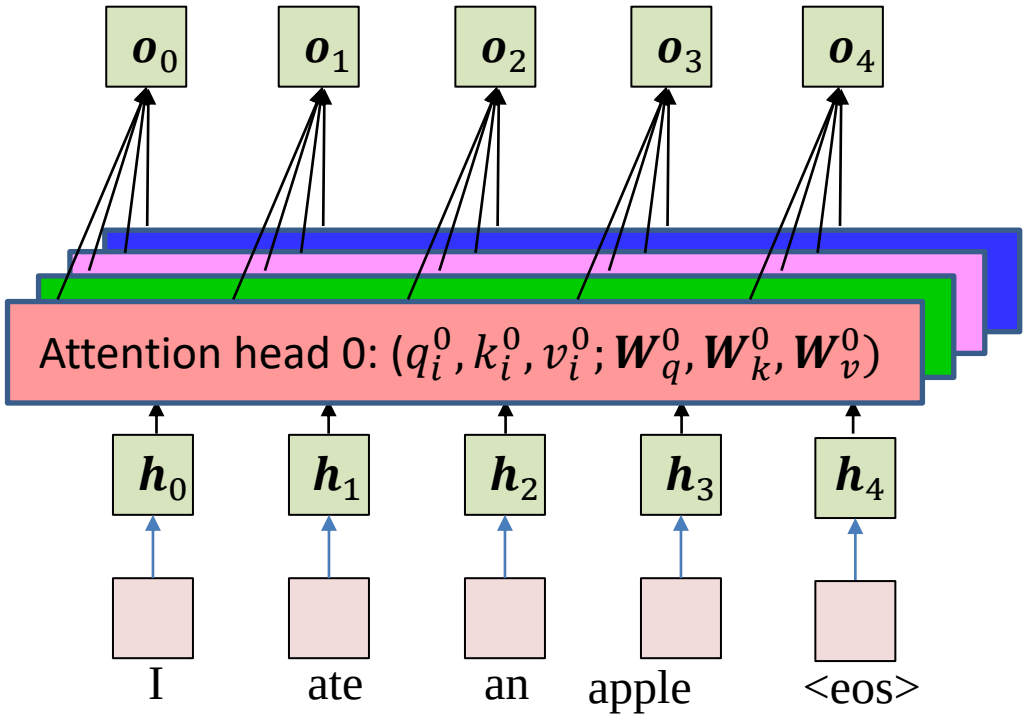
$$v_i^a = \boldsymbol{W}_v^a h_i$$

$$w_{ij}^a = attn(q_i^a, k_{0:N}^a)$$

$$o_i^a = \sum_j w_{ij}^a v_j^a$$

Concatenate

$$o_i = [o_i^1; o_i^2; o_i^3; \ldots; o_i^H]$$



- We can have *multiple* such attention "heads"
  - Each will have an independent set of queries, keys and values
  - Each will obtain an independent set of attention weights
    - Potentially focusing on a different aspect of the input than other heads
  - Each computes an independent output
- The final output is the concatenation of the outputs of these attention heads
- "MULTI-HEAD ATTENTION" (actually Multi-head *self* attention)

$$q_i^a = \boldsymbol{W}_q^a h_i$$

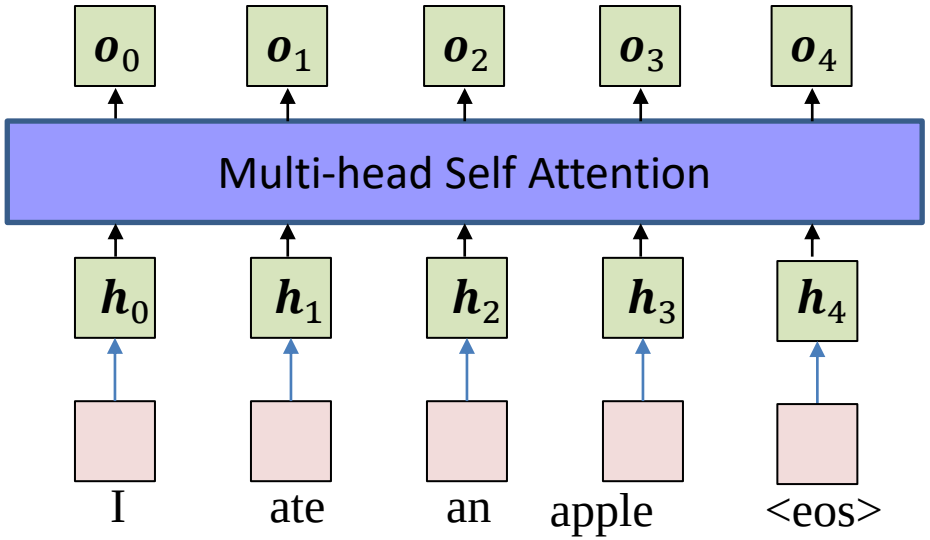
$$k_i^a = \boldsymbol{W}_k^a h_i$$

$$v_i^a = \boldsymbol{W}_v^a h_i$$

$$w_{ij}^a = attn(q_i^a, k_{0:N}^a)$$

$$o_i^a = \sum_j w_{ij}^a v_j^a$$

$$o_i = [o_i^1; o_i^2; o_i^3; \ldots; o_i^H]$$



- Multi-head self attention
  - Multiple self-attention modules in parallel

$$q_i^a = \boldsymbol{W}_q^a h_i$$

$$k_i^a = \boldsymbol{W}_k^a h_i$$

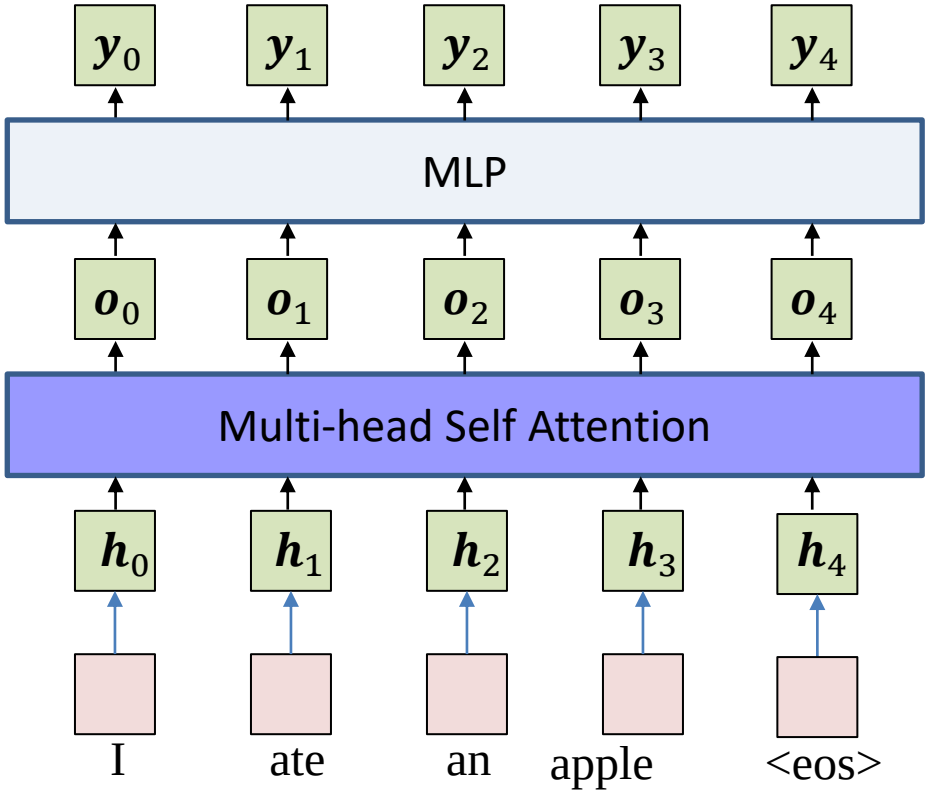
$$v_i^a = \boldsymbol{W}_v^a h_i$$

$$w_{ij}^a = attn(q_i^a, k_{0:N}^a)$$

$$o_i^a = \sum_j w_{ij}^a v_j^a$$

$$o_i = [o_i^1; o_i^2; o_i^3; \ldots; o_i^H]$$

$$y_i = MLP(o_i)$$



- Typically, the output of the multi-head self attention is passed through one or more regular feedforward layers
  - Affine layer followed by a non-linear activation such as ReLU

$$q_i^a = \boldsymbol{W}_q^a h_i$$

$$k_i^a = \boldsymbol{W}_k^a h_i$$

$$v_i^a = \boldsymbol{W}_v^a h_i$$

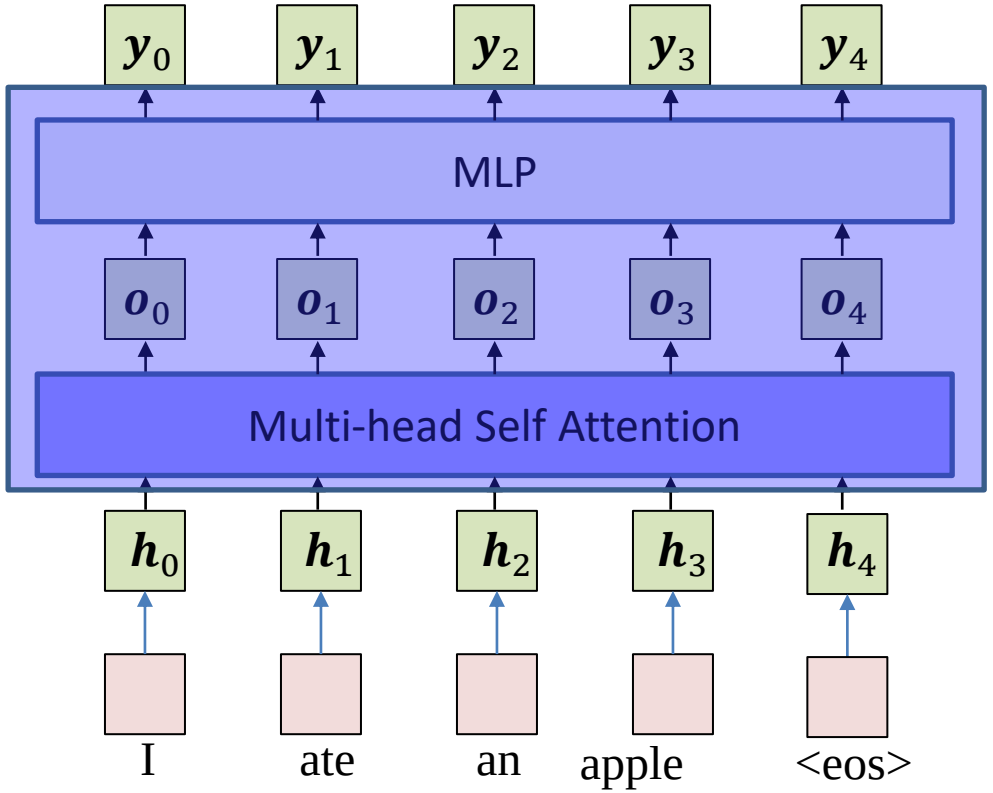
$$w_{ij}^a = attn(q_i^a, k_{0:N}^a)$$

$$o_i^a = \sum_j w_{ij}^a v_j^a$$

$$o_i = [o_i^1; o_i^2; o_i^3; \ldots; o_i^H]$$

$$y_i = MLP(o_i)$$

MULTI-HEAD SELF ATTENTION BLOCK



- The entire unit, including multi-head self-attention module followed by MLP is a ***multi-head self-attention block***

MULTI-HEAD SELF ATTENTION BLOCK

$$q_i^a = \boldsymbol{W}_q^a h_i$$

$$k_i^a = \boldsymbol{W}_k^a h_i$$

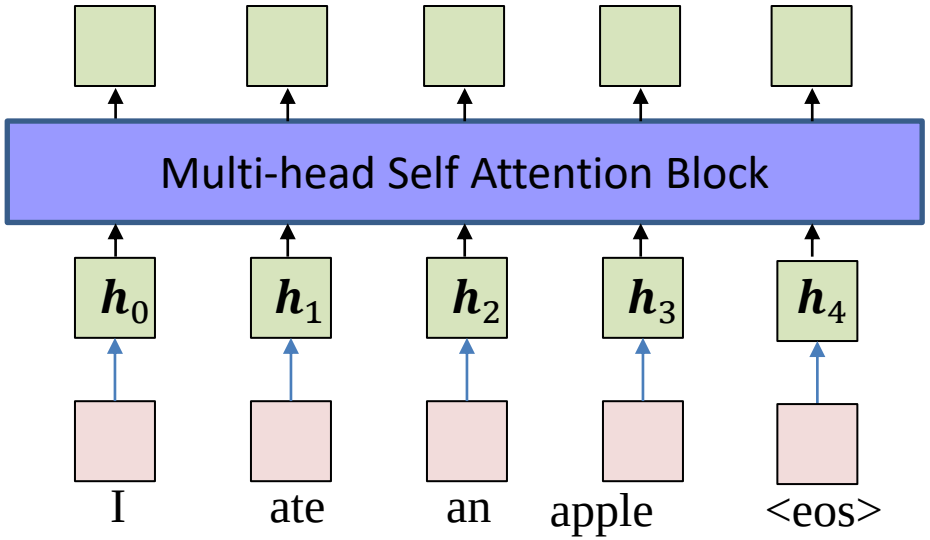
$$v_i^a = \boldsymbol{W}_v^a h_i$$

$$w_{ij}^a = attn(q_i^a, k_{0:N}^a)$$

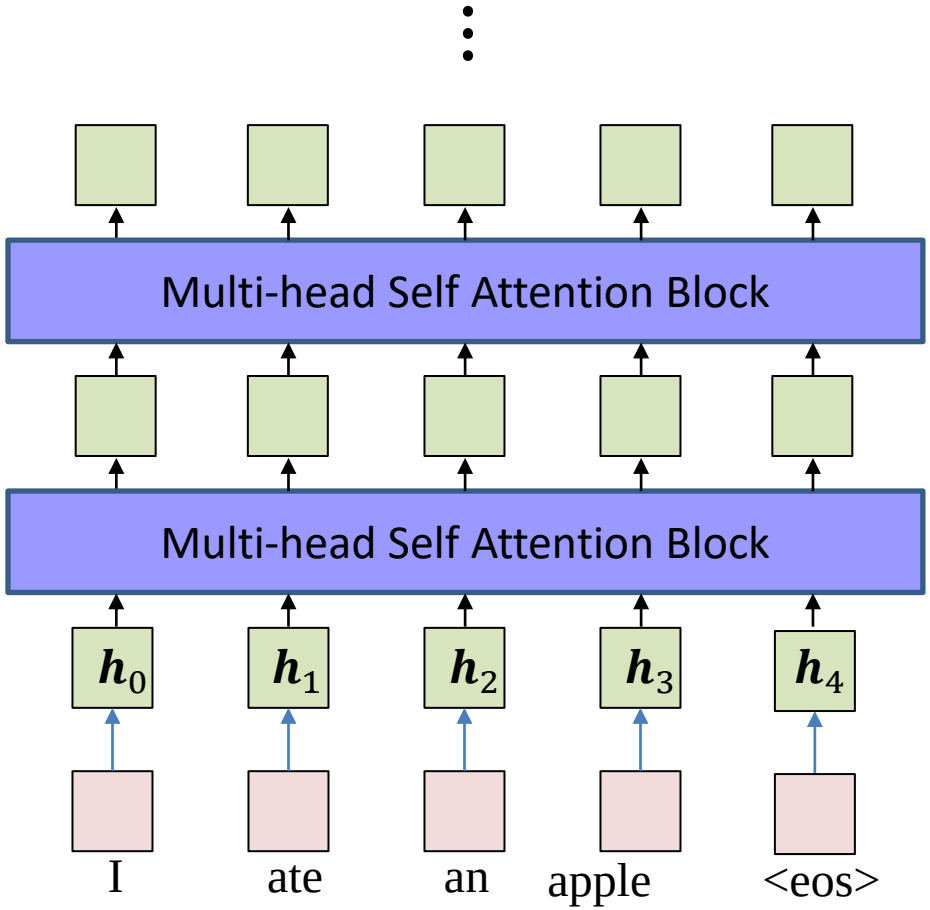
$$o_i^a = \sum_j w_{ij}^a v_j^a$$

$$o_i = [o_i^1; o_i^2; o_i^3; \ldots; o_i^H]$$

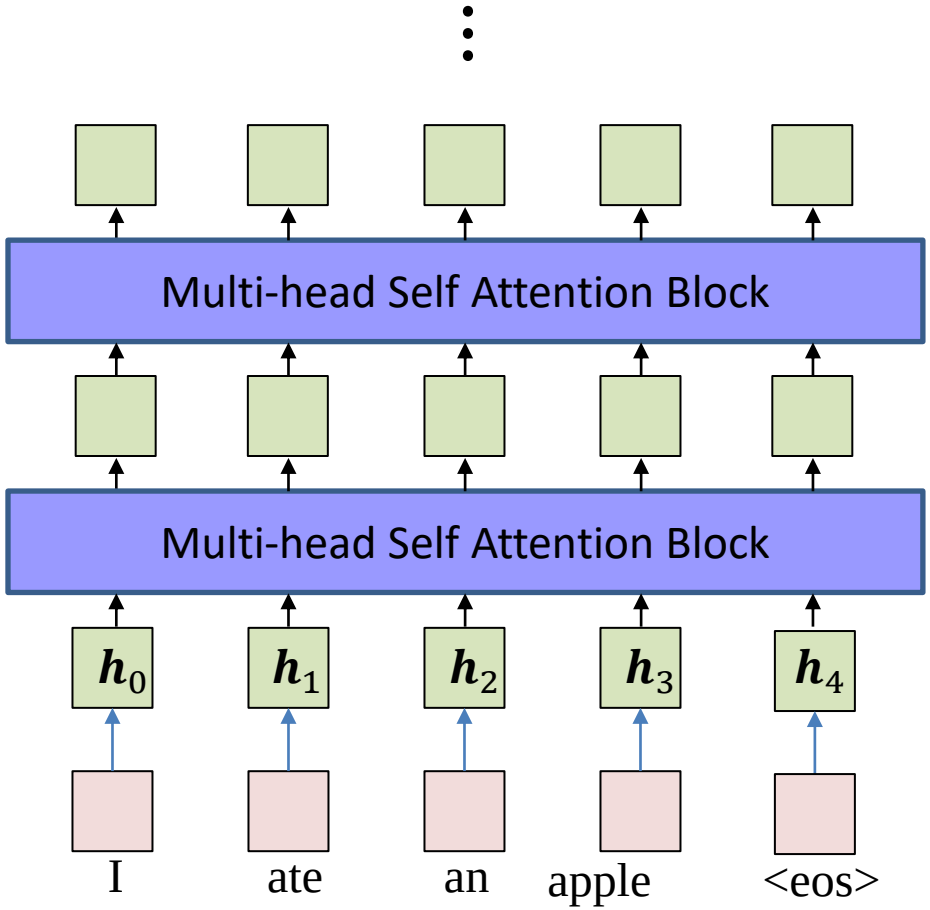
$$y_i = MLP(o_i)$$



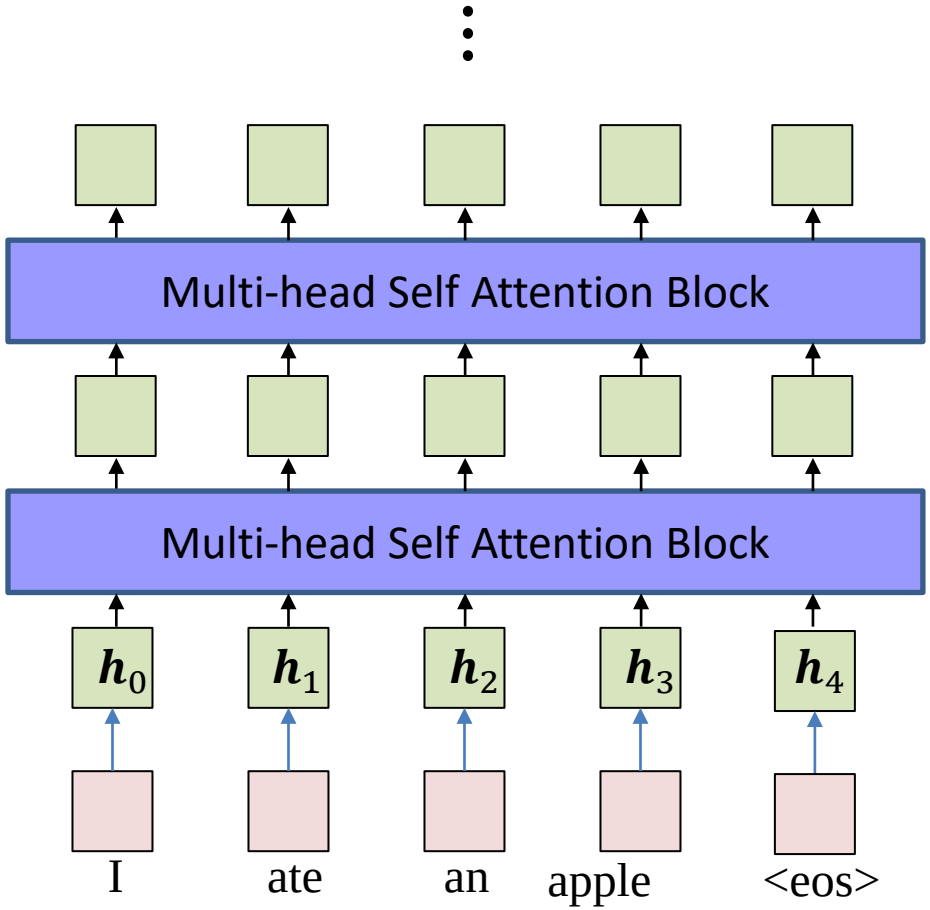
- The entire unit, including multi-head self-attention module followed by MLP is a ***multi-head self-attention block***



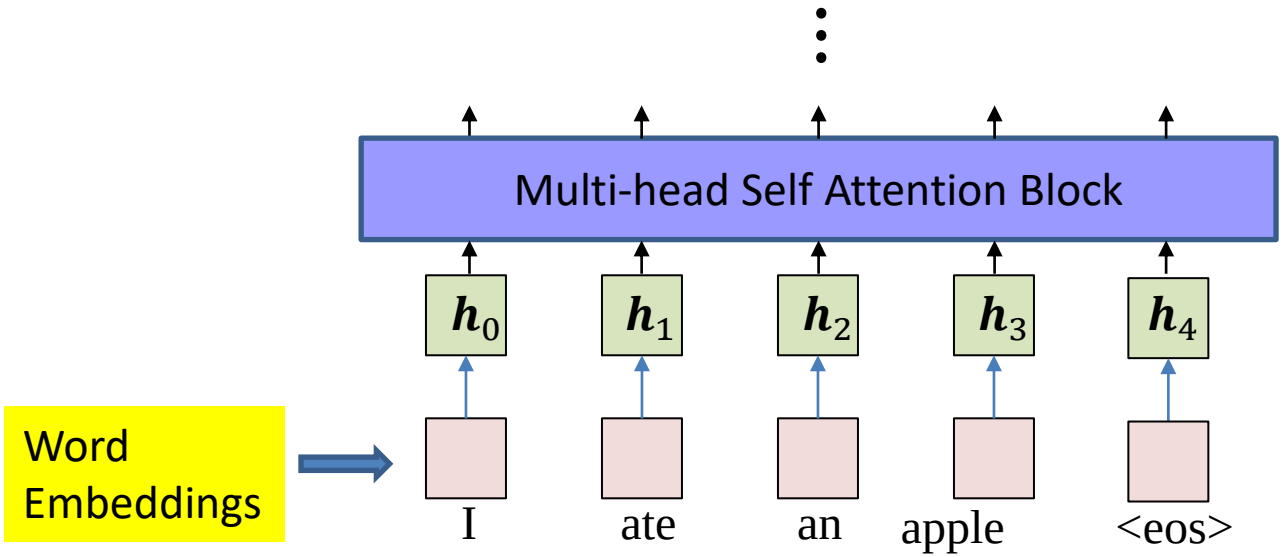
- The encoder can include many layers of such blocks
- No need for recurrence…



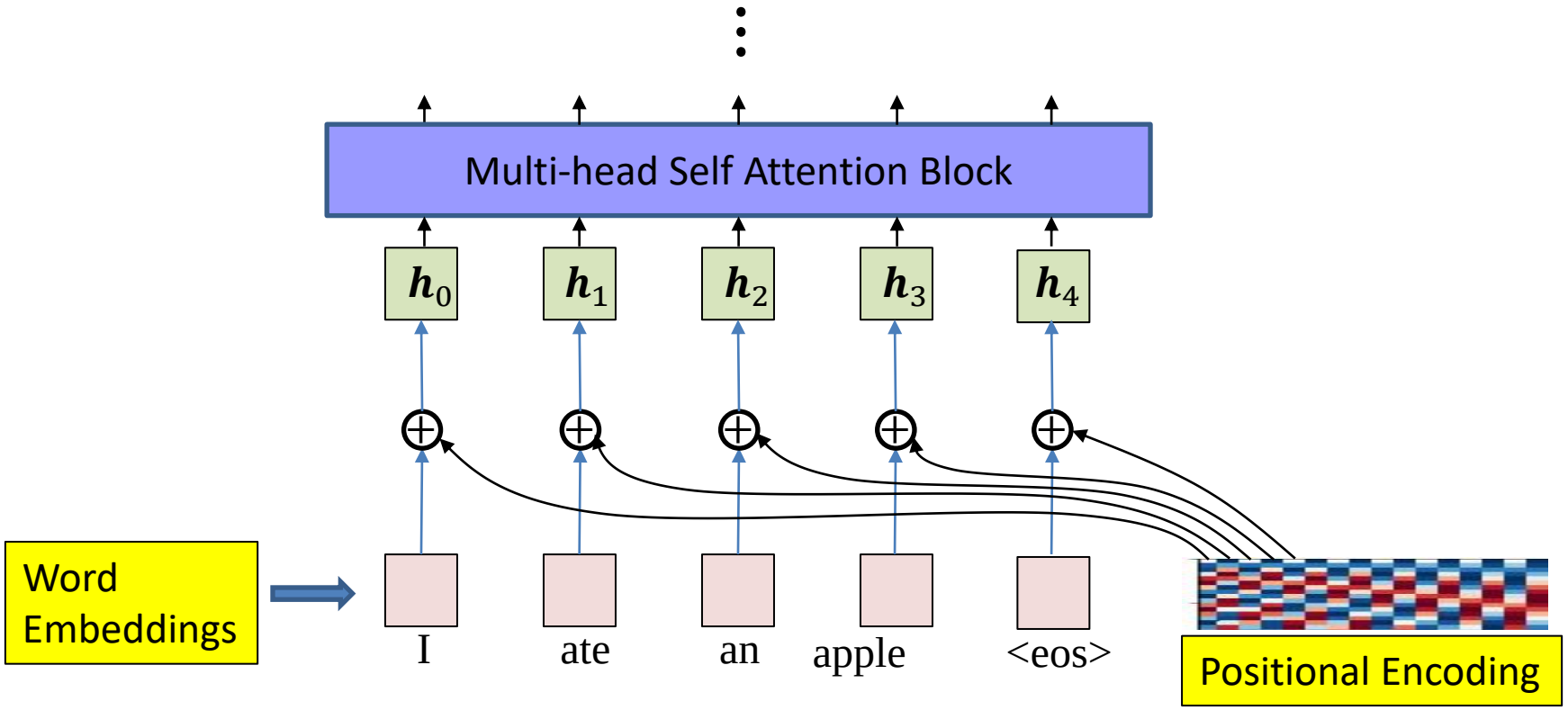
- Recap: The encoder in a sequence-to-sequence model can replace recurrence through a series of “multi-head self attention” blocks



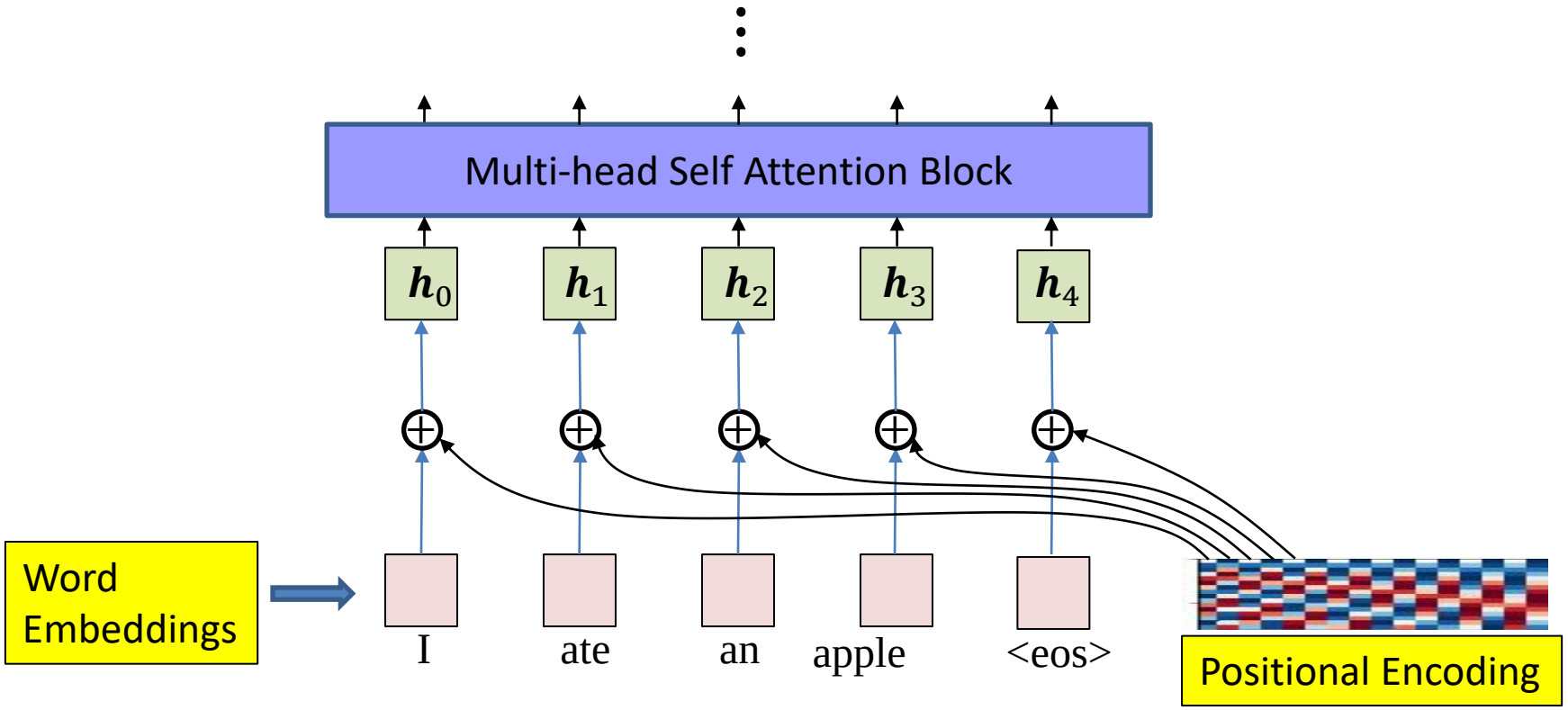
- Recap: The encoder in a sequence-to-sequence model can replace recurrence through a series of "multi-head self attention" blocks
- But this still ignores *relative position*
  - A context word one word away is different from one 10 words away
  - The attention framework does not take distance into consideration



- Note that the inputs are actually word *embeddings*



- Note that the inputs are actually word *embeddings*
- We add a "positional" encoding to them to capture the relative distance from one another



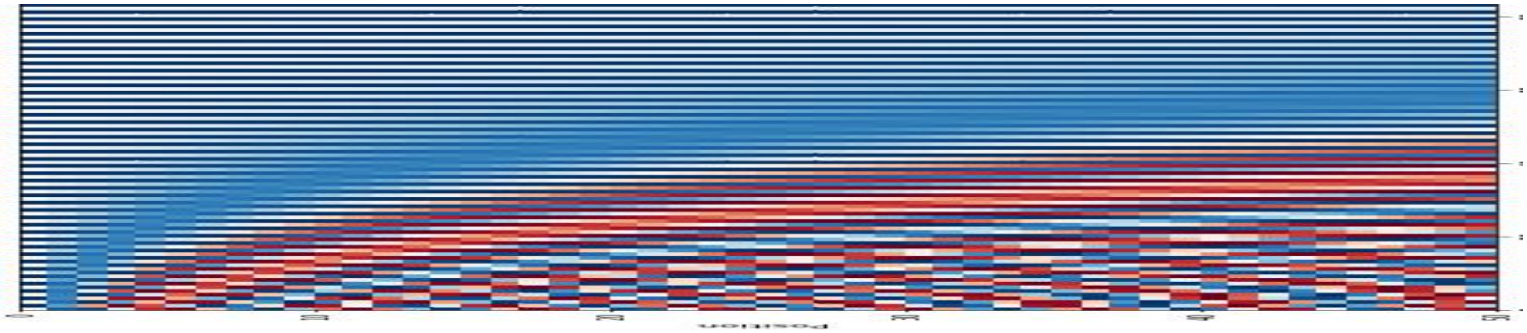
- **Positional Encoding:** A sequence of vectors $P_0, \ldots, P_N$, to encode position
  - Every vector is unique (and uniquely represents time)
  - Relationship between P_t and $P_{t+\tau}$ only depends on the distance between them

$$P_{t+\tau} = M_\tau P_t$$

- The linear relationship between P_t and $P_{t+\tau}$ enables the net to learn shift-invariant "gap" dependent relationships

Positional Encoding

regenerate



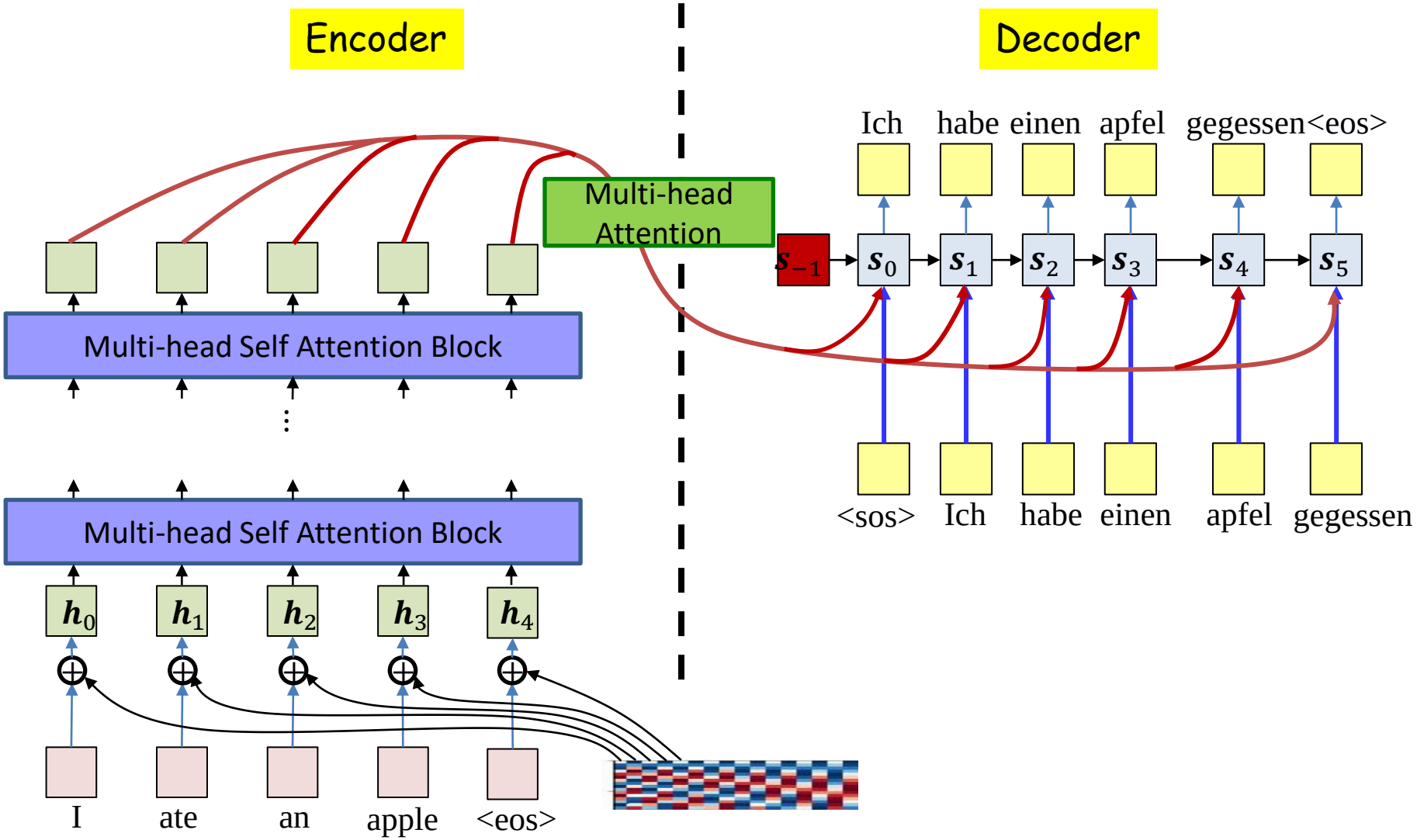
$$P_t = \begin{bmatrix} \sin \omega_1 t \\ \cos \omega_1 t \\ \sin \omega_2 t \\ \cos \omega_2 t \\ \vdots \\ \sin \omega_{d/2} t \\ \sin \omega_{d/2} t \end{bmatrix}$$

$$\omega_l = \frac{1}{10000^{2l/d}}$$

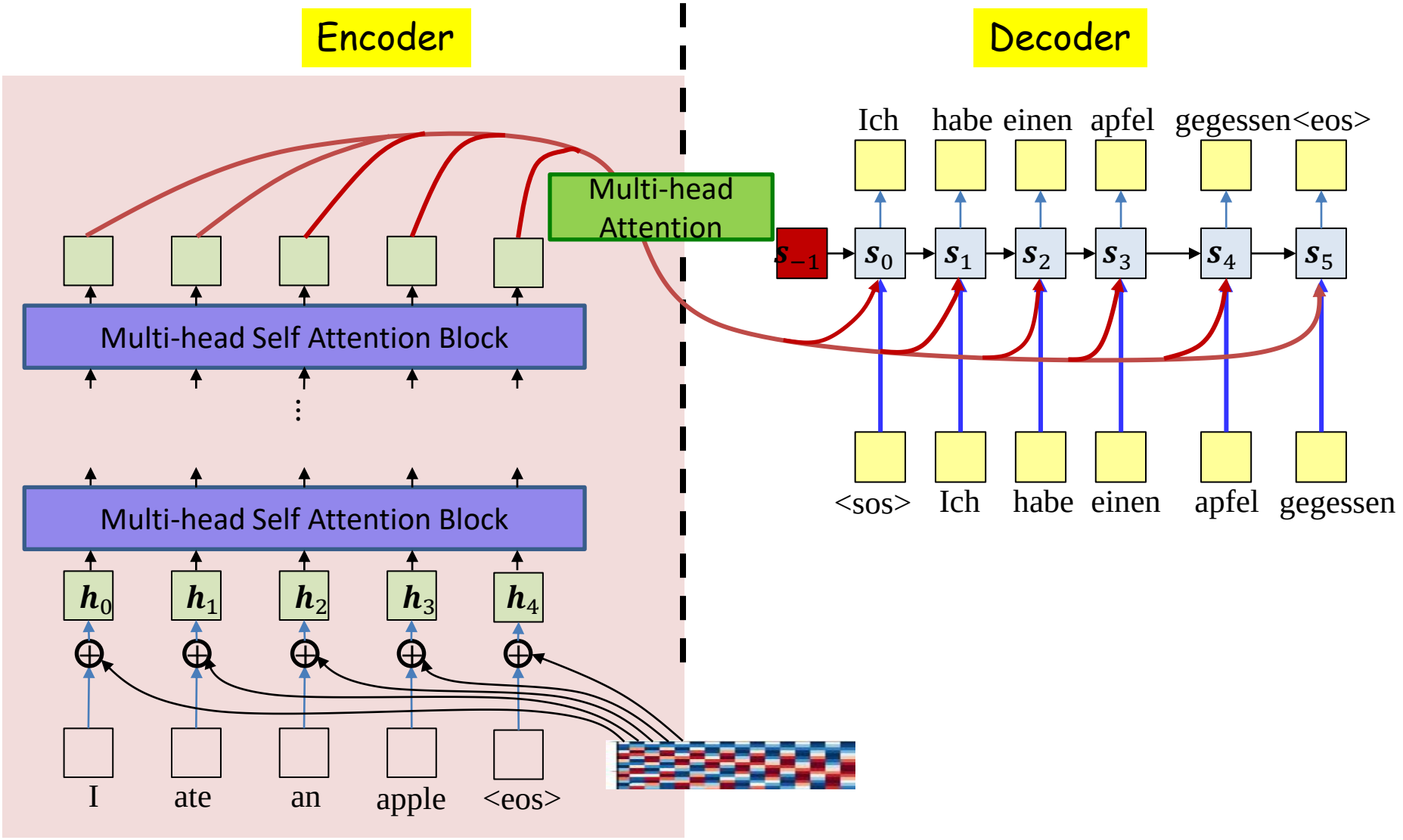
$$P_{t+\tau} = M_\tau P_t$$

$$M_\tau = diag\left(\begin{bmatrix} \cos \omega_l \tau & \sin \omega_l \tau \\ -\sin \omega_l \tau & \cos \omega_l \tau \end{bmatrix}, l = 1 \dots d/2\right)$$

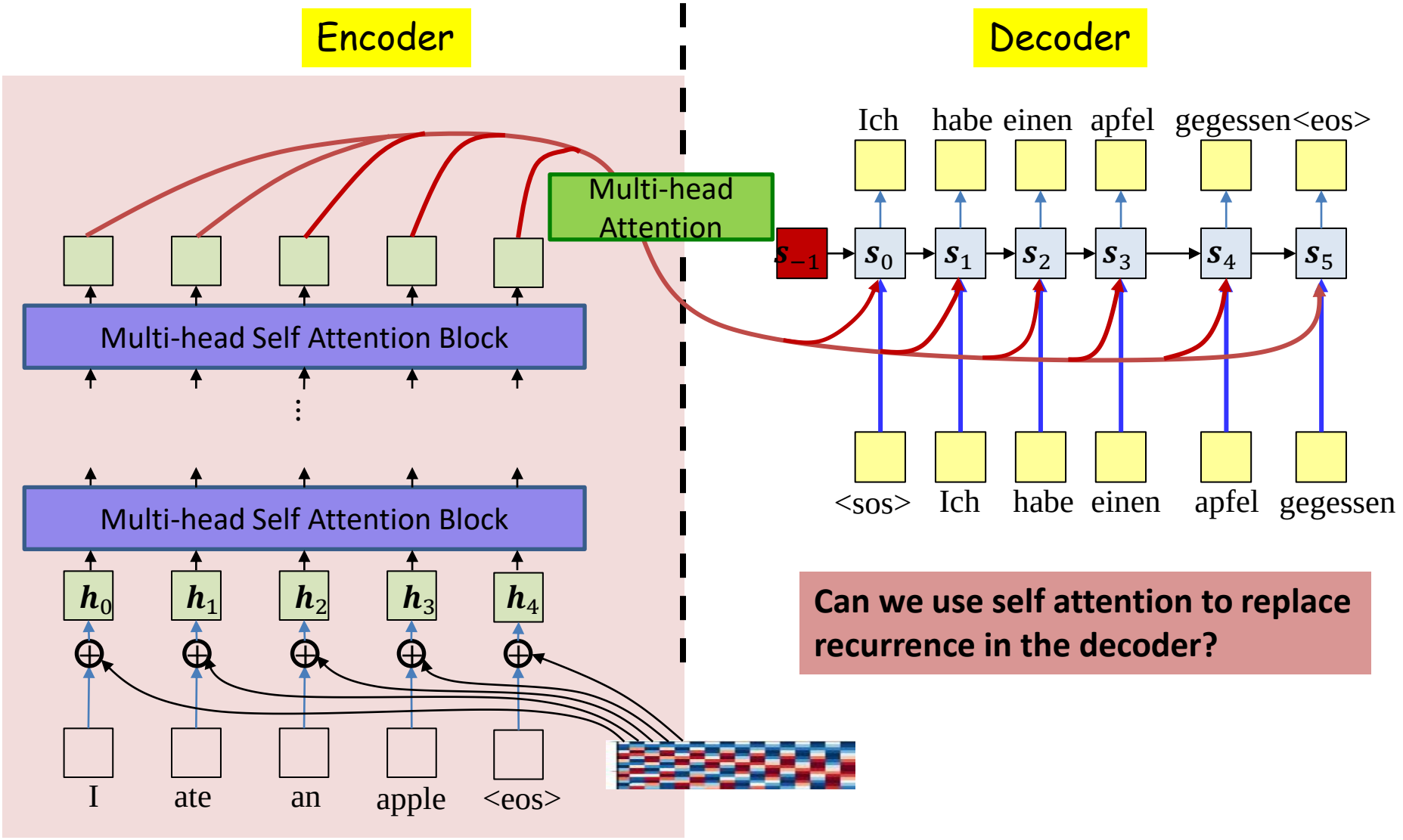
- A vector of sines and cosines of a harmonic series of frequencies
  - Every $2l$-th component of P_t is $\sin \omega_l t$
  - Every $2l + 1$-th component of P_t is $\cos \omega_l t$
- Never repeats
- Has the linearity property required



- The linear relationship between P_t and $P_{t+\tau}$ enables the net to learn shift-invariant "gap" dependent relationships

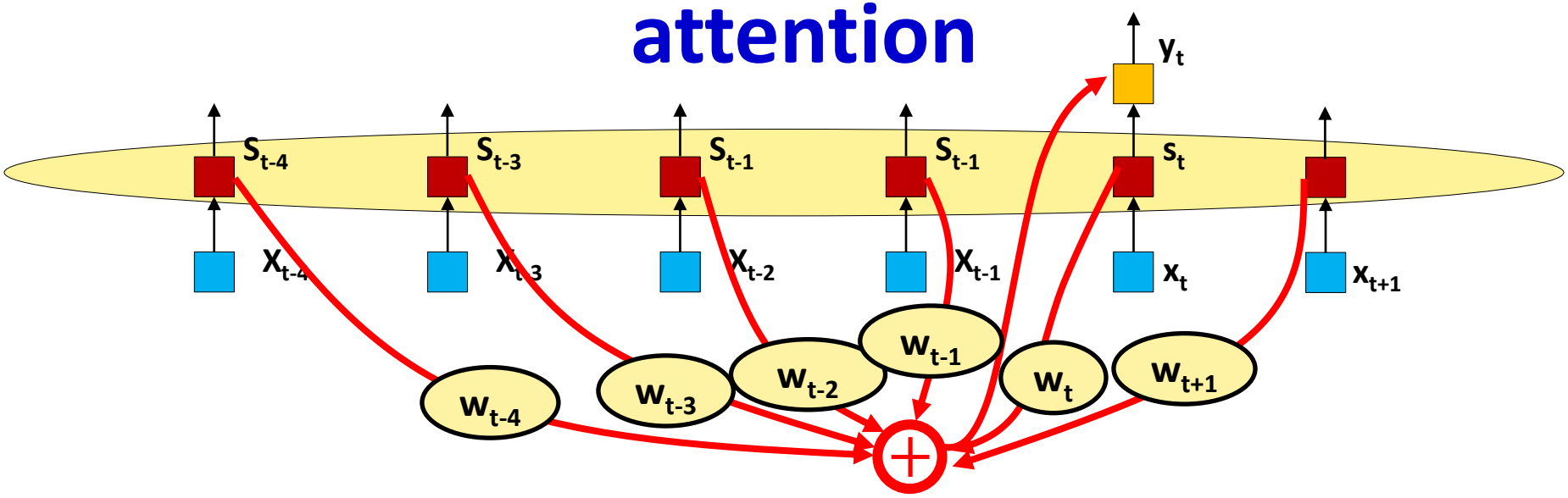


- The self-attending encoder!!



- The self-attending encoder!!

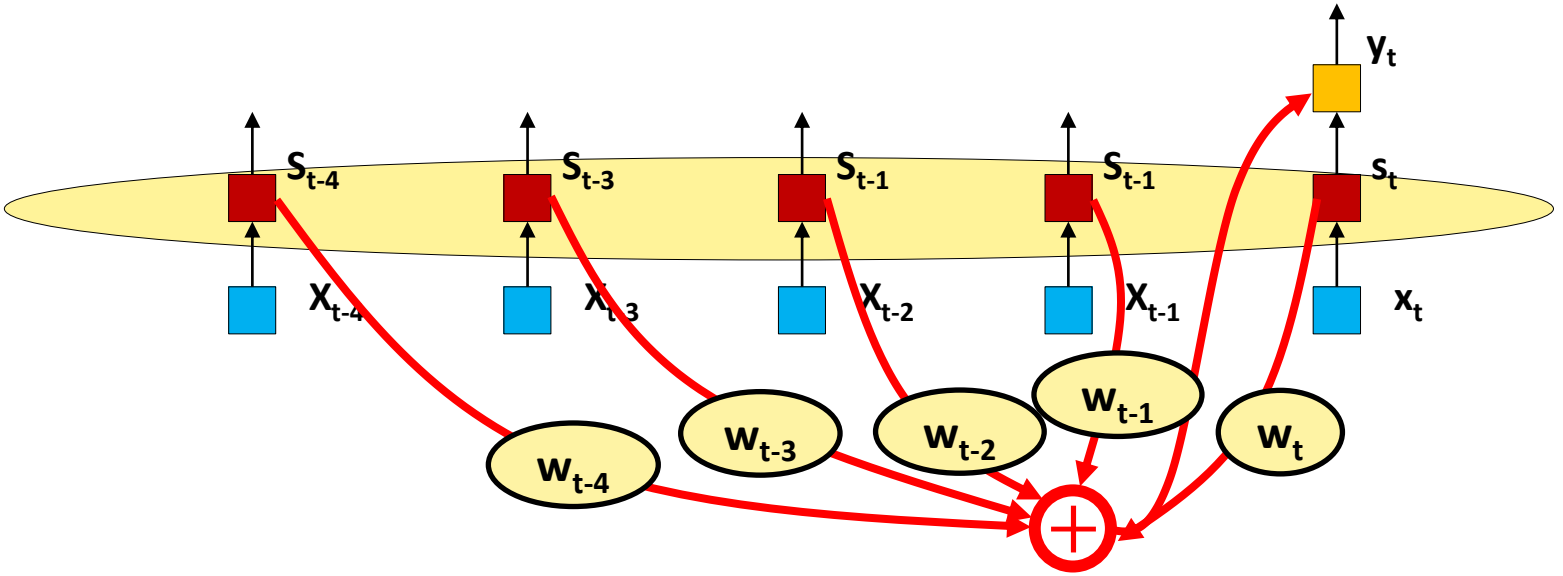
Self attention and masked self attention



- **Self attention in encoder:** Can use input embedding at time t+1 and further to compute output at time t, because all inputs are available

Self attention and masked self attention

- **Self attention in decoder:** Decoder is sequential
  - Each word is produced using the previous word as input
  - Only embeddings until time t are available to compute the output at time t
- The attention will have to be “masked”, forcing attention weights for t+1 and later to 0



Masked self-attention block

$q_i = \boldsymbol{W}_q h_i$

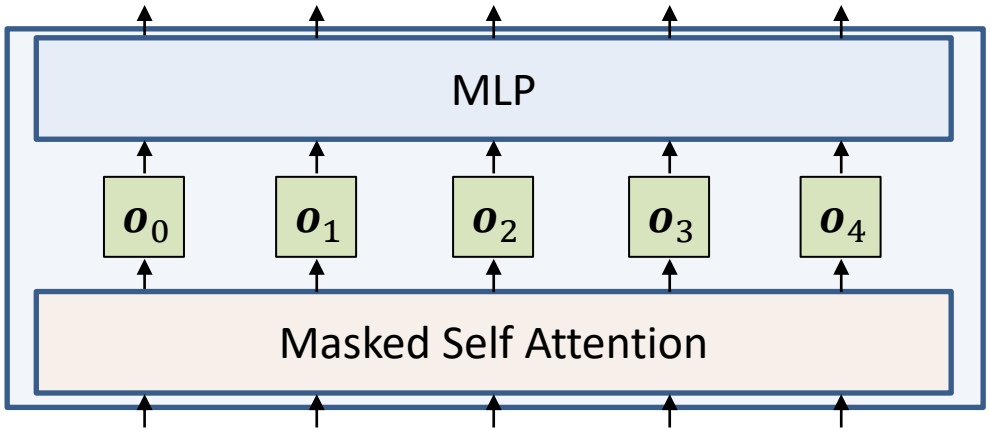
$k_i = \boldsymbol{W}_k h_i$

$v_i = \boldsymbol{W}_v h_i$

$e_{ij} = q_i^T k_j$

$w_{i0}, \ldots, w_{ii} = softmax(e_{i0}, \ldots, e_{ii})$

$o_i = \sum_{j=0}^{i-1} w_{ij} v_j$



- The "masked self attention ***block***" includes an MLP after the masked self attention
  - Like in the encoder

Masked multi-head self-attention

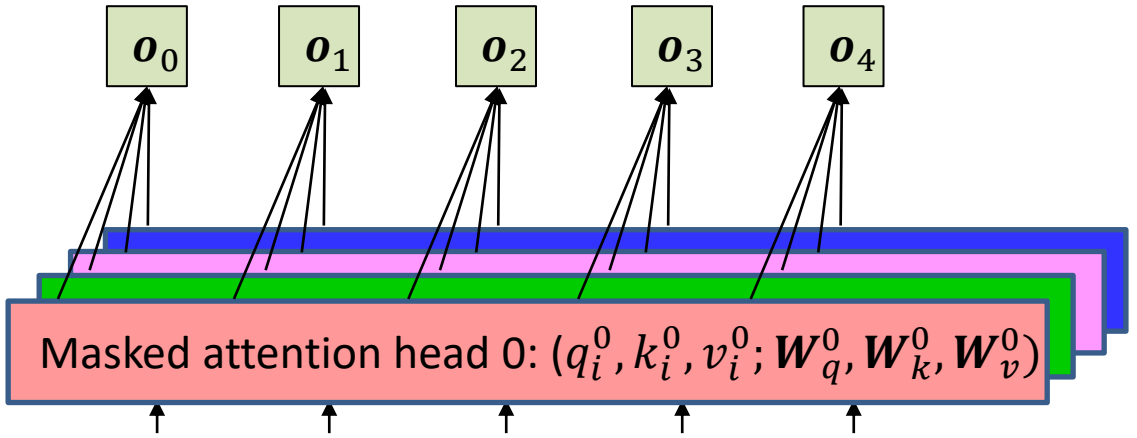
$$q_i^a = \boldsymbol{W}_q^a h_i$$

$$k_i^a = \boldsymbol{W}_k^a h_i$$

$$v_i^a = \boldsymbol{W}_v^a h_i$$

$$w_{ij}^a = attn(q_i^a, k_{0:i-1}^a)$$

$$o_i^a = \sum_j w_{ij}^a v_j^a$$



- The "masked ***multi-head*** self attention ***block***" includes multiple masked attention heads
  - Like in the encoder

Masked multi-head self-attention block

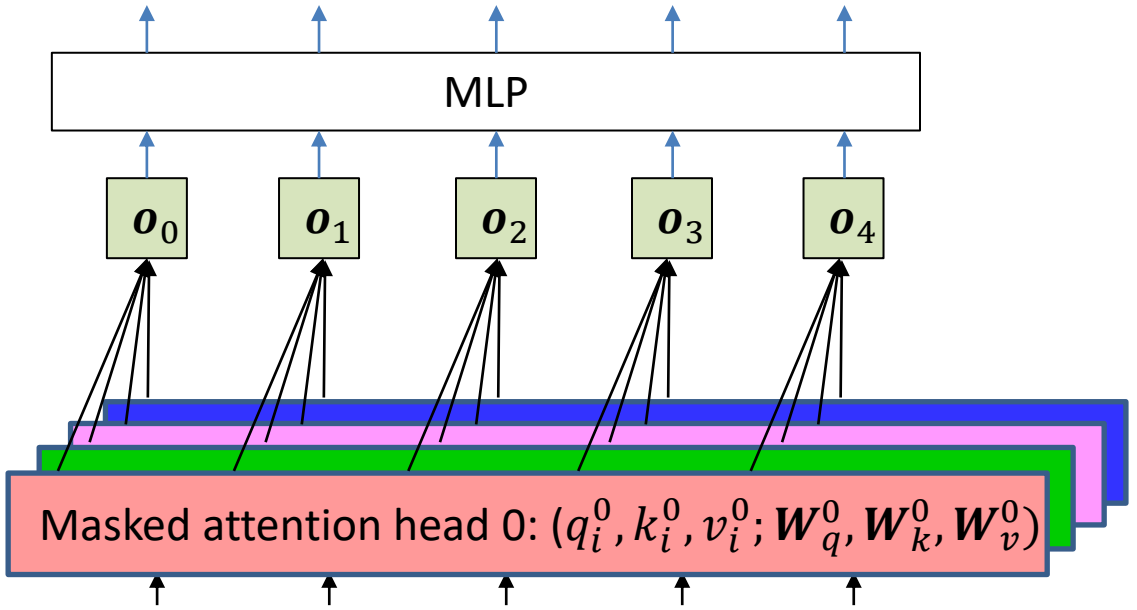
$$q_i^a = \boldsymbol{W}_q^a h_i$$

$$k_i^a = \boldsymbol{W}_k^a h_i$$

$$v_i^a = \boldsymbol{W}_v^a h_i$$

$$w_{ij}^a = attn(q_i^a, k_{0:i-1}^a)$$

$$o_i^a = \sum_j w_{ij}^a v_j^a$$



- The "masked ***multi-head*** self attention ***block***" includes multiple masked attention heads
  - Like in the encoder

Masked multi-head self-attention block

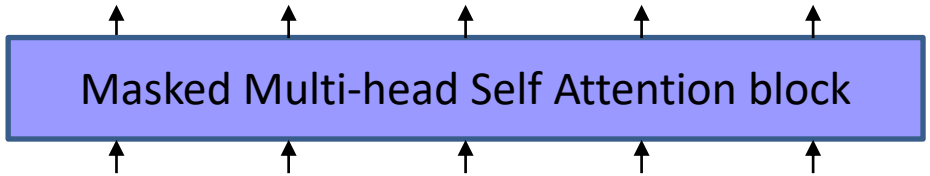
$$q_i^a = \boldsymbol{W}_q^a h_i$$

$$k_i^a = \boldsymbol{W}_k^a h_i$$

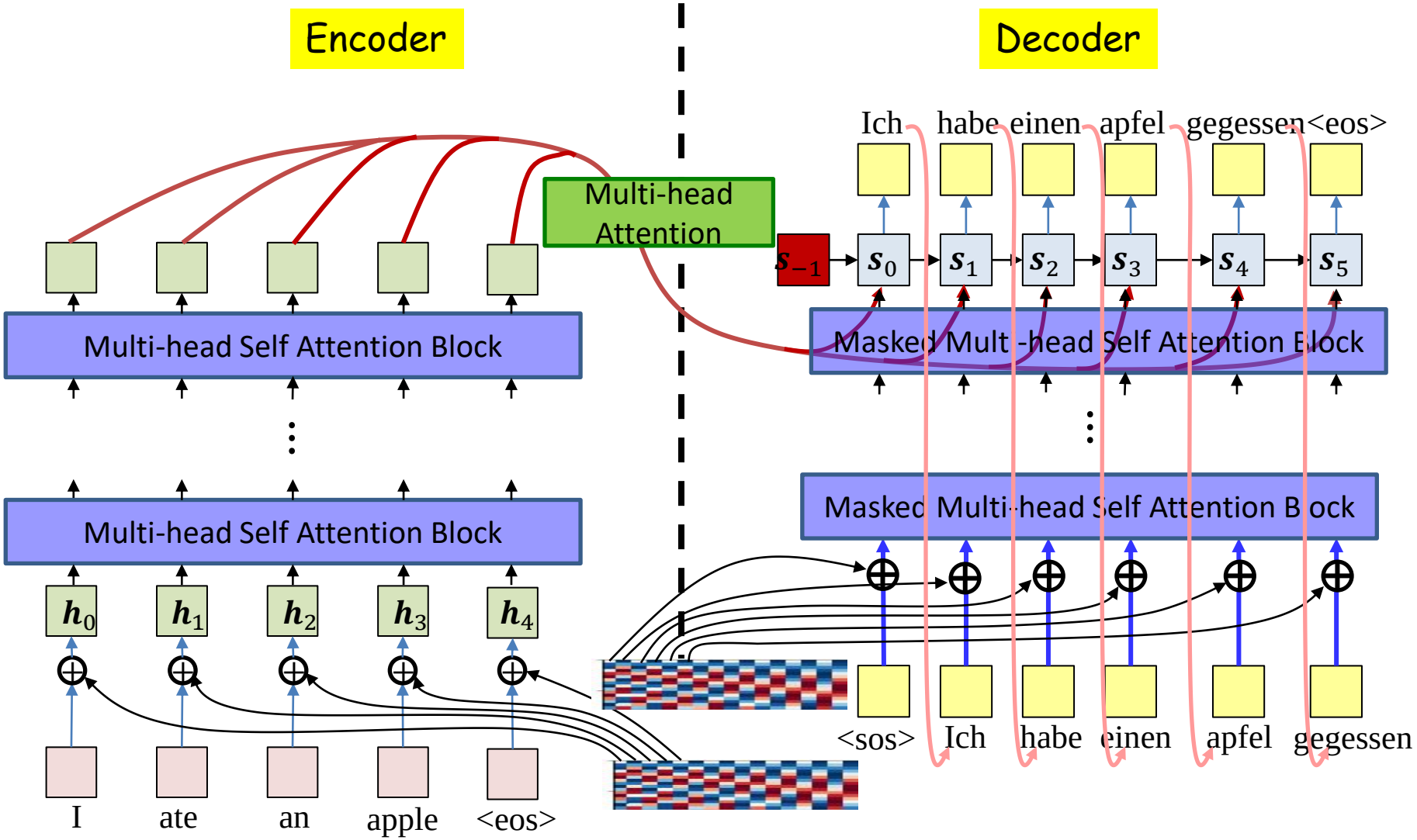
$$v_i^a = \boldsymbol{W}_v^a h_i$$

$$w_{ij}^a = attn(q_i^a, k_{0:i-1}^a)$$

$$o_i^a = \sum_j w_{ij}^a v_j^a$$

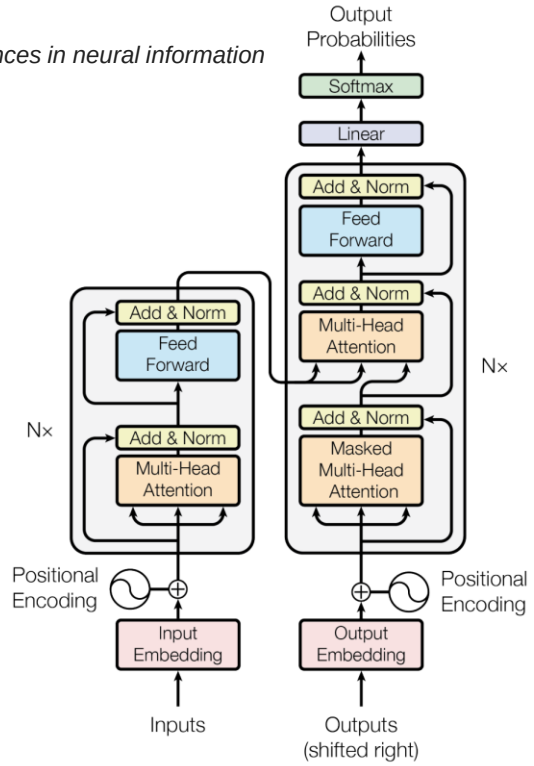


- The "masked ***multi-head*** self attention ***block***" includes multiple masked attention heads, followed by an MLP
  - Like in the encoder


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
h_0 h_1 h_2 h_3 h_4
s_{-1} s_0 s_1 s_2 s_3 s_4 s_5
I ate an apple <eos>
<sos> Ich habe einen apfel gegessen
Ich habe einen apfel gegessen<eos>

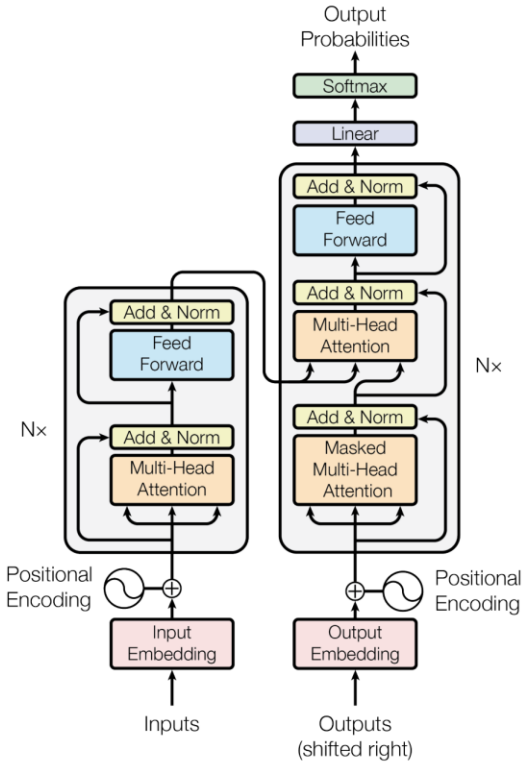
Transformer: Attention is all you need

Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems*. 2017.



- Transformer: A sequence-to-sequence model that replaces recurrence with positional encoding and multi-head self attention
  - "Attention is all you need"

Transformer



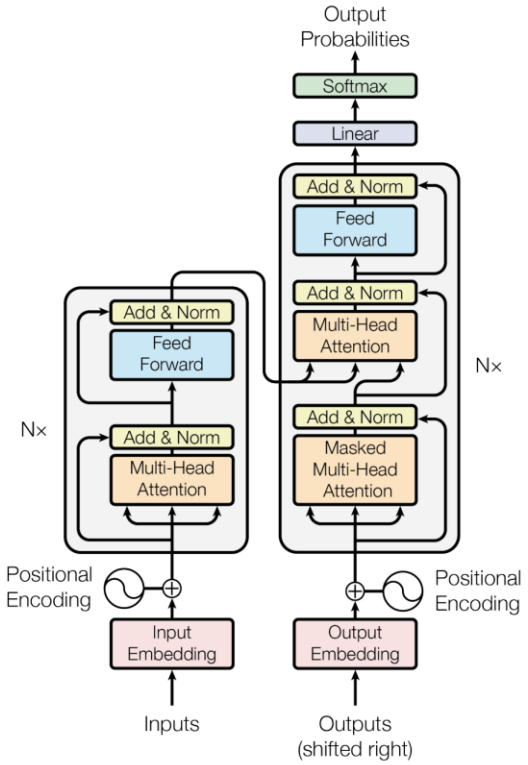
From "Attention is all you need"

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

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [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	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

- Transformer: tremendous decrease in model computation for similar performance as state-of-art translation models
- The last row in the table shows transformer performance
- The final two columns show computational cost.

Transformer



From "Attention is all you need"

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

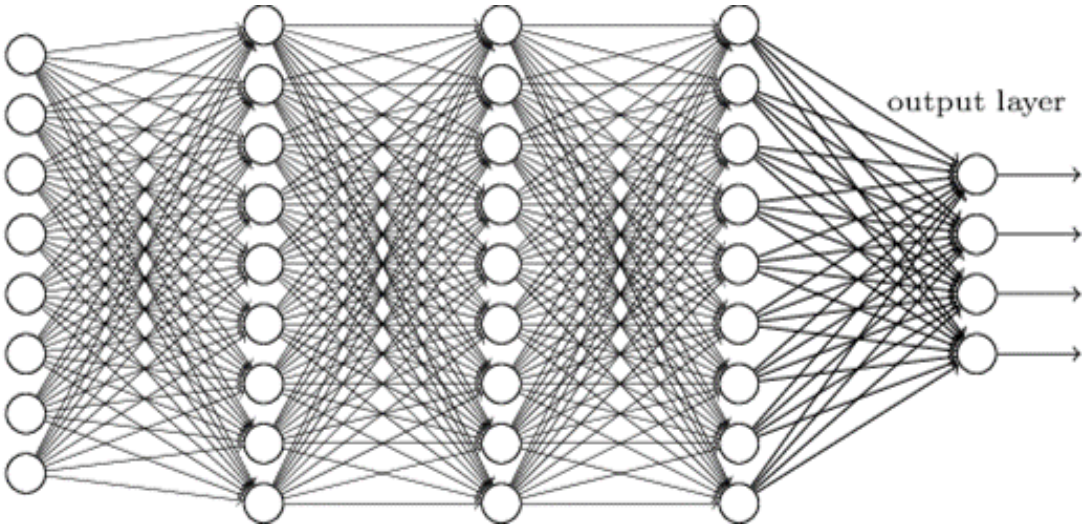
Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
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	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Why so good?

Why so fast?

- Transformer: tremendous decrease in model computation for similar performance as state-of-art translation models
- The last row in the table shows transformer performance
- The final two columns show computational cost.

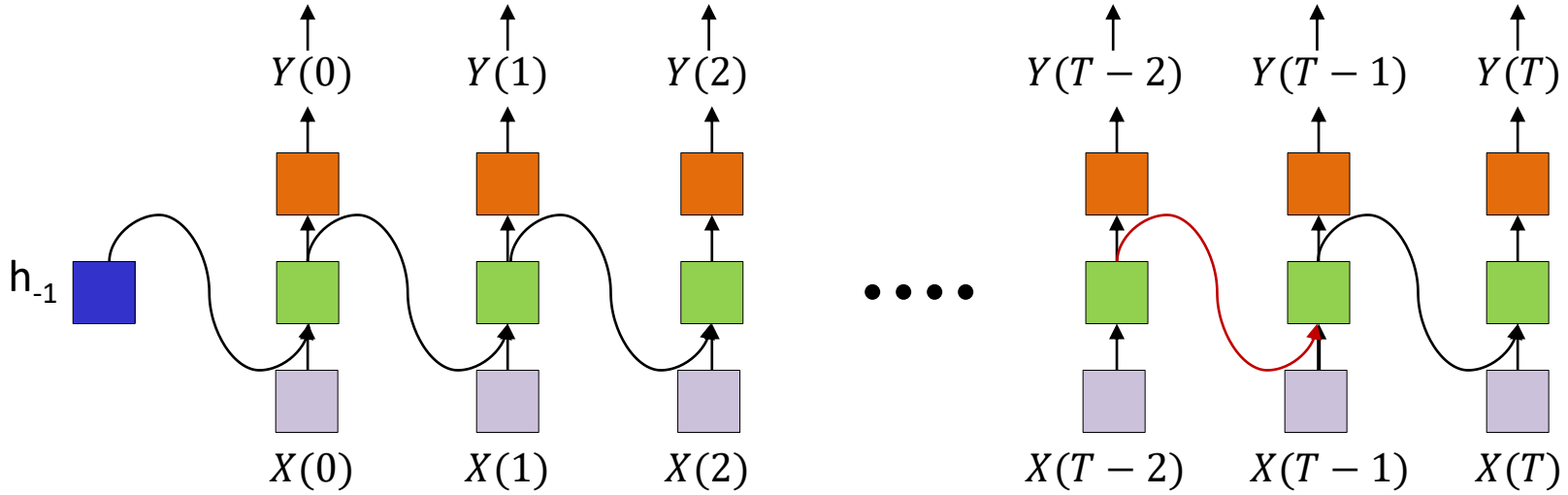
Recap: Vanishing/exploding gradients



$$\nabla_{f_k} Div = \nabla D . \nabla f_N . W_N . \nabla f_{N-1} . W_{N-1} \ldots \nabla f_{k+1} W_{k+1}$$

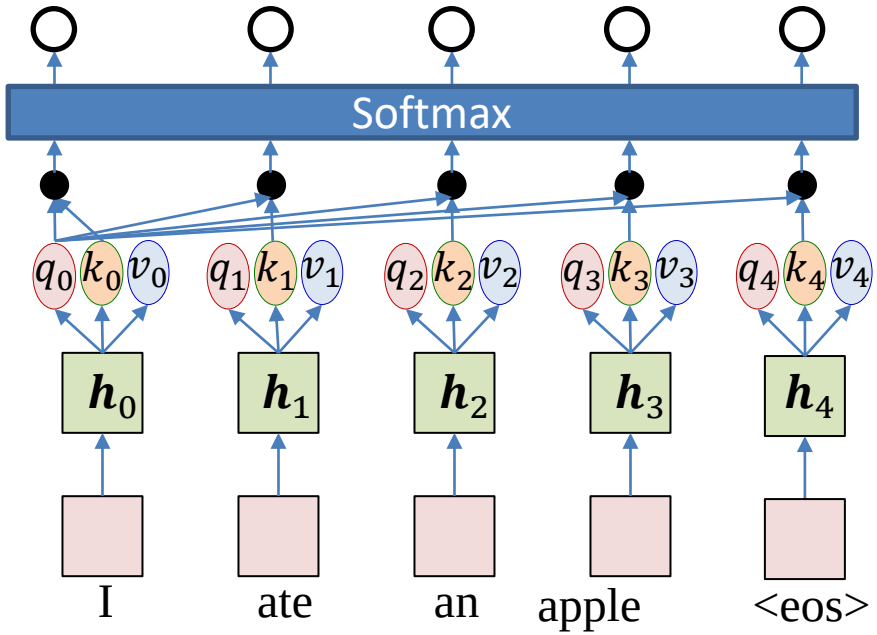
- RNNs are just very deep networks
- LSTMs mitigate the problem at the cost of 3x more matrix multiplications
- Transformers get rid of it! To encode a full sentence, they have way fewer layers than an unrolled RNN.
- The same goes with the vanishing memory issue to an extent.

Processing order



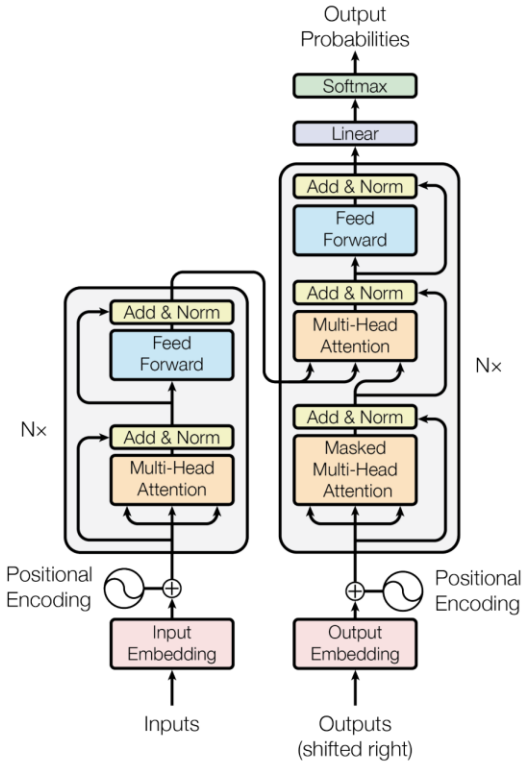
- Computing $Y(T)$ requires $Y(T-1)$…
- Which requires $Y(T-2)$, etc…
- RNN inputs must be processed *in order* → slow implementation

Processing order



- q_n, k_n, v_n can be computed separately.
- n^2 $< q_n, k_n >$ dot products to compute.
- Self attention is easy to compute in parallel → Faster implementations

Transformer



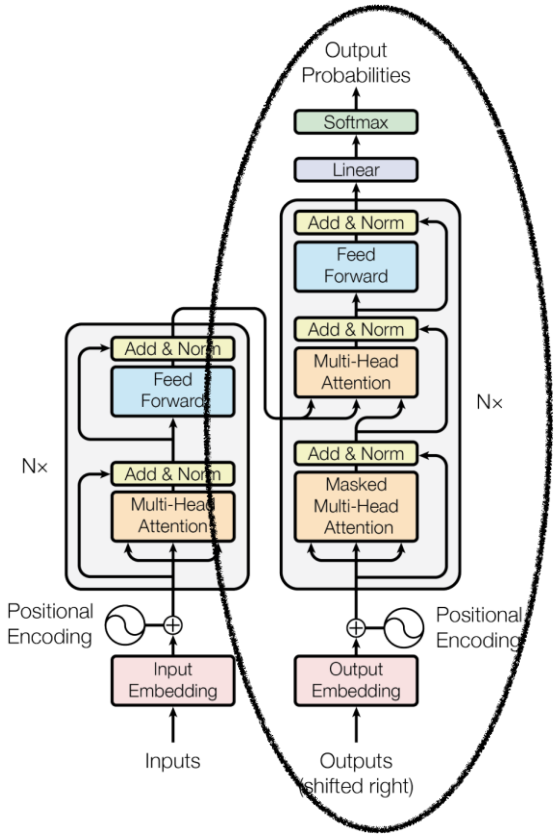
From "Attention is all you need"

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

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [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	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

- Transformer: tremendous decrease in model computation for similar performance as state-of-art translation models
- The last row in the table shows transformer performance
- The final two columns show computational cost.

GPT



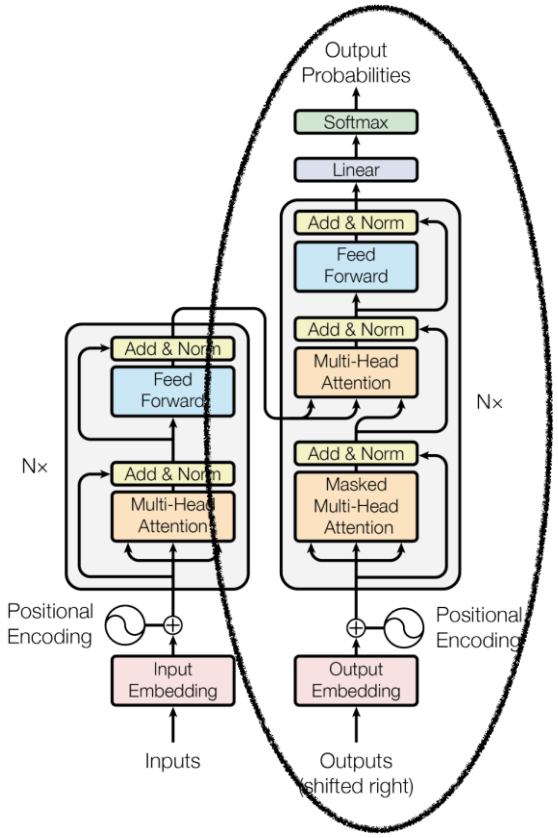
Alec Radford et. al., Improving Language Understanding by Generative Pre-Training

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

GPT



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

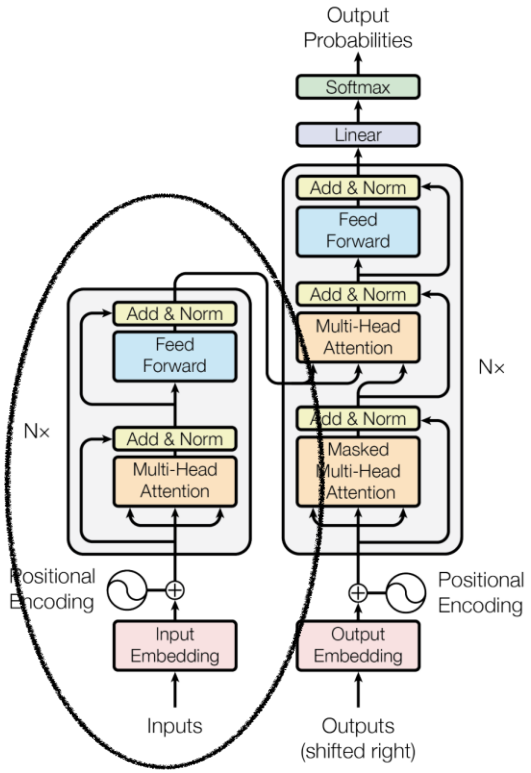
Table 5: Analysis of various model ablations on different tasks. Avg. score is a unweighted average of all the results. (*mc*= Mathews correlation, *acc*=Accuracy, *pc*=Pearson correlation)

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

- Add *Task conditioning*: put the nature of your task in the input (not just LM)
- Parameters x1000

→ **GPT-3** : Generalizes to **more tasks**, not just more inputs!

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)

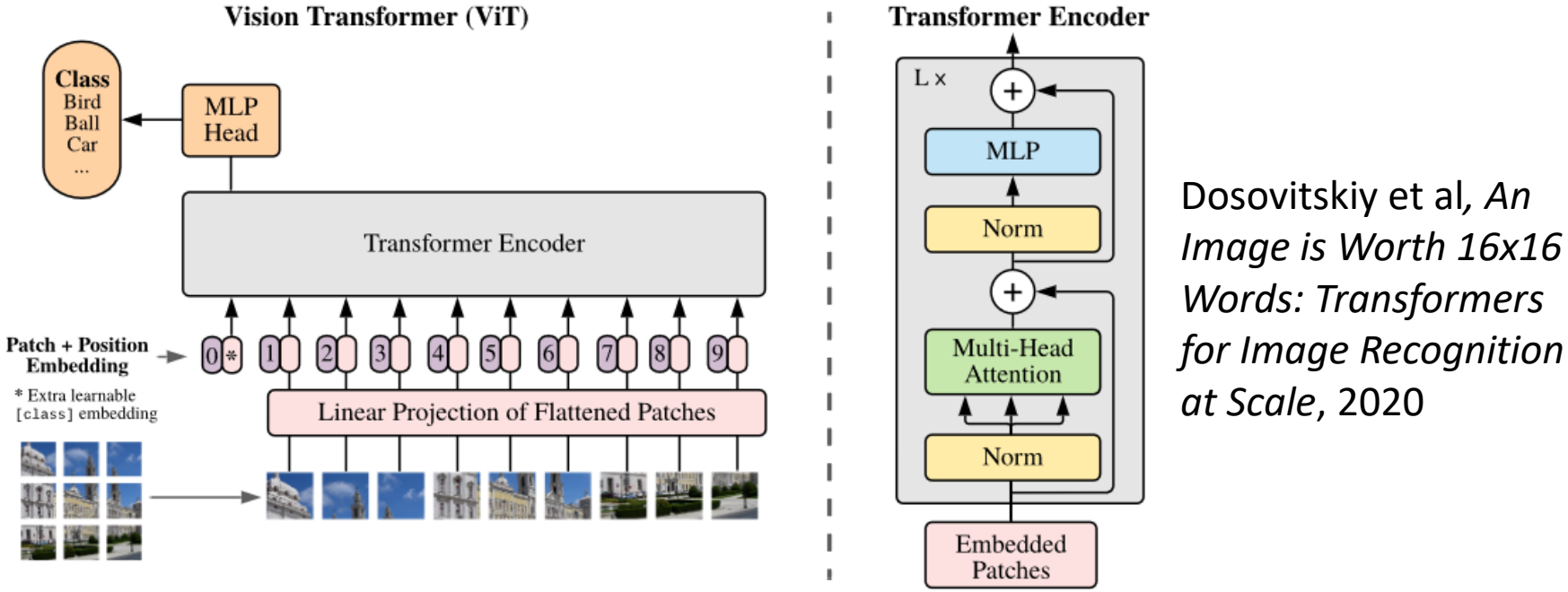
Attention is all you need

- Self-attention can effectively replace recurrence in sequence-to-sequence models
  - "Transformers"
  - Requires "positional encoding" to capture positional information
- Can also be used in regular sequence analysis settings as a substitute for recurrence
- Currently *the* state of the art in most sequence analysis/prediction…

Attention is all you need

- Self-attention can effectively replace recurrence in sequence-to-sequence models
  - "Transformers"
  - Requires "positional encoding" to capture positional information
- Can also be used in regular sequence analysis settings as a substitute for recurrence
- Currently *the* state of the art in most sequence analysis/prediction… and even computer vison problems!

Vision Transformers



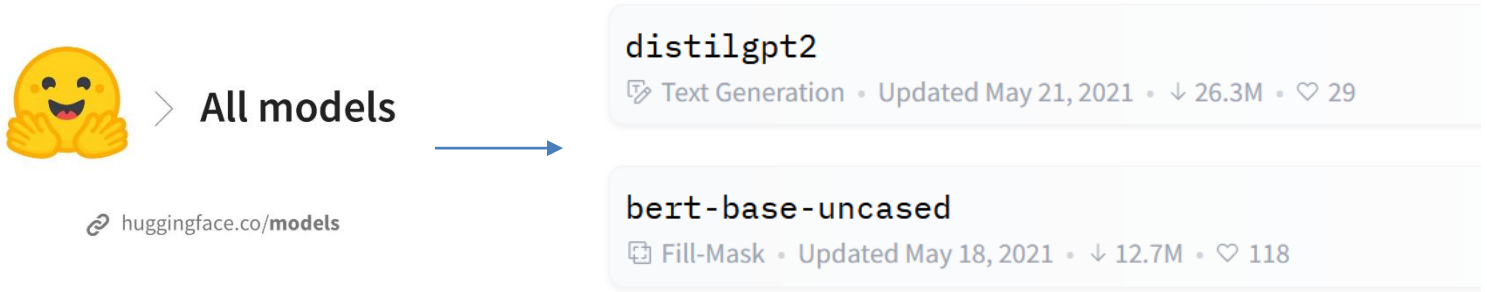
Dosovitskiy et al, *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*, 2020

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

→ Sequential and image problems are similar when using transformers

Impact of Transformers

- Transformers have played a major role in the "uniformization" of DL-based tasks:
  - Find a pretrained "BERT-like" transformer (Text, Image, Speech)
  - Fine-tune on your task – or not! (Prompting…)
- This has helped democratize Deep Learning considerably



- **But**…

Caveat 1

- Not all transformers are the same: Big/small, fast/slow, mono-/multilingual, contrastive/ generative, regressive/autoencoding…
- Pick the right one!

Caveat 2

- Transformers are not always the right choice.
- They often require more parameters than LSTMs at equal performance

→ Tricky on small hardware (phones, IoT, etc)

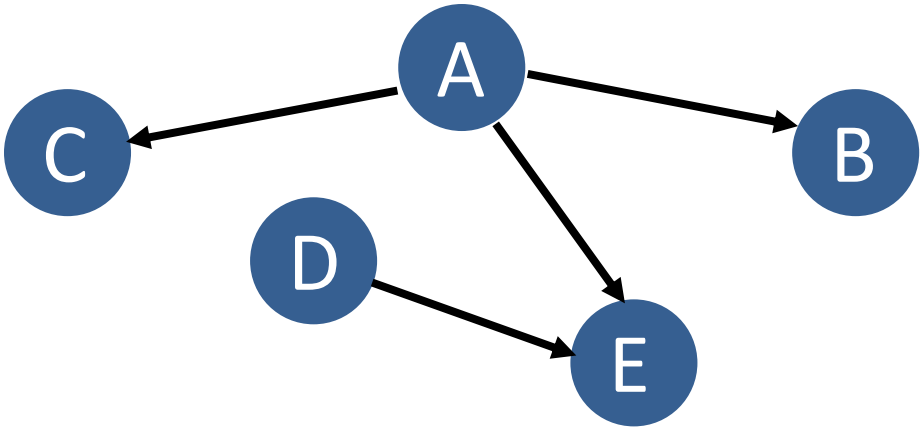
Topics for the week

- Transformers
- GNNs

Input structure

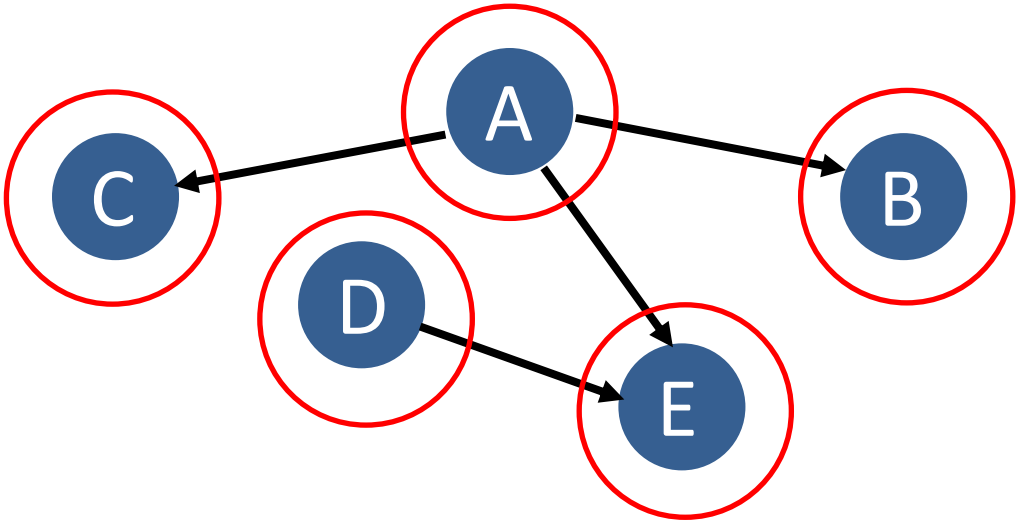
- We've seen models for
  - Static, fixed-sized inputs
    - MLP
  - Shift invariant pattern recognition
    - CNN
  - Inputs arranged in a sequence
    - RNN
- What about Graph-structured inputs?

What is a graph?



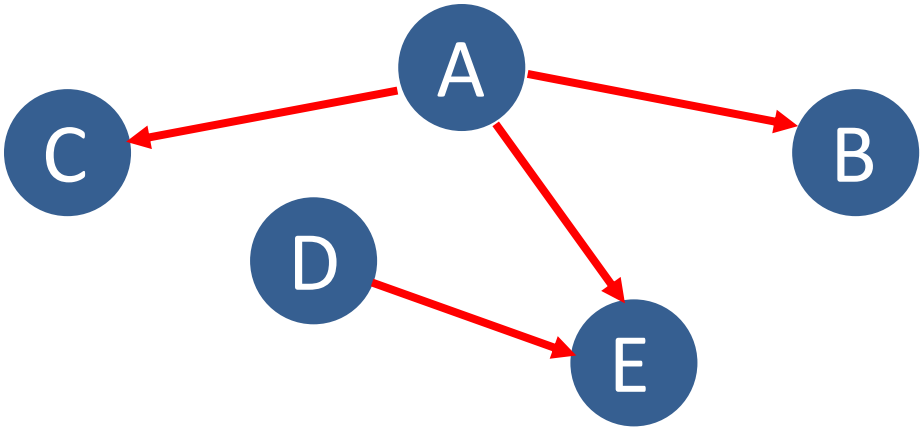
- A structured representation of sets of entities with pair-wise interactions

What is a graph?



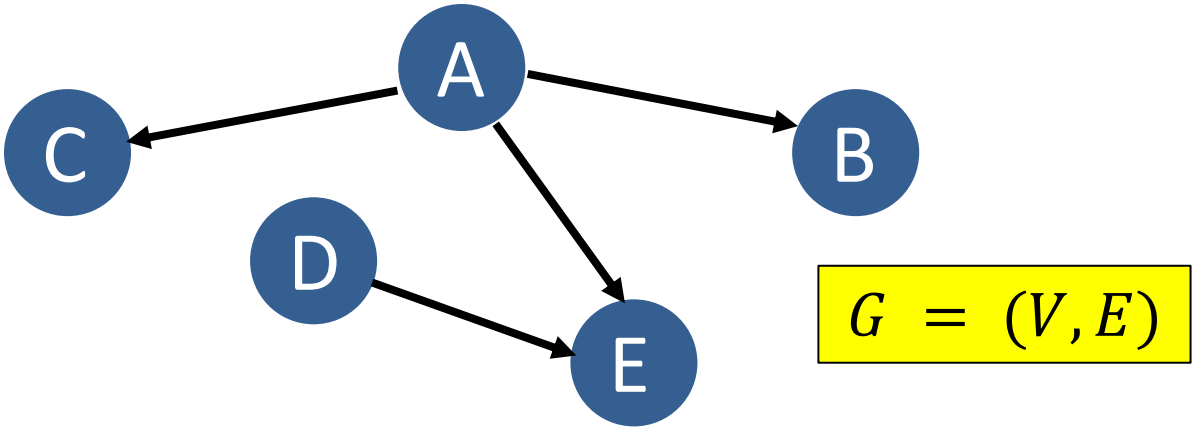
- **Objects/Data points** : Nodes, Vertices (V)

What is a graph?



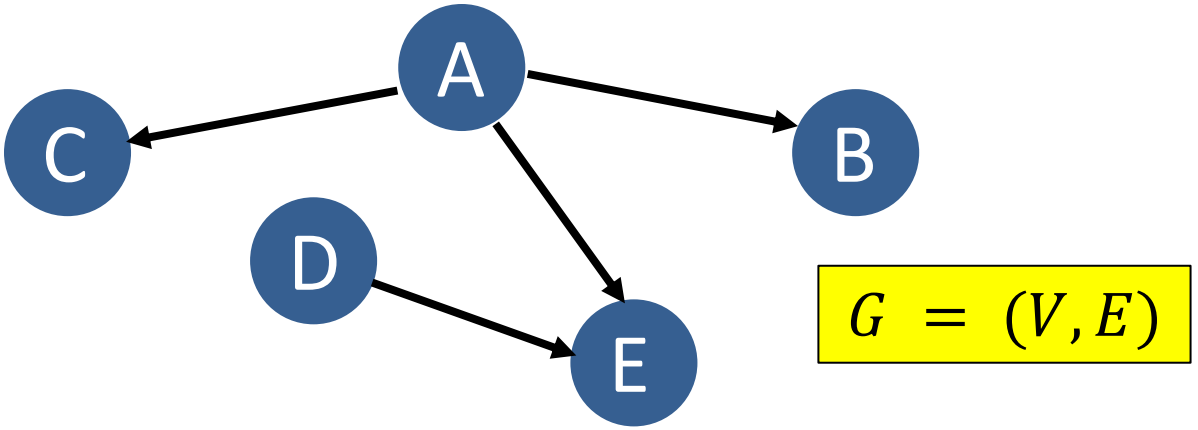
- **Objects/Data points** : Nodes, Vertices (V)
- **Interactions/Relations** : Links, Edges (E)

What is a graph?



- **Objects/Data points** : Nodes, Vertices (V)
- **Interactions/Relations** : Links, Edges (E)
- **System** : Network, Graphs (G)

What is a graph?



- **Objects/Data points** : Nodes, Vertices (V)
- **Interactions/Relations** : Links, Edges (E)
- **System** : Network, Graphs (G)
- **Node (and edge) Attributes** : Feature vectors (X)

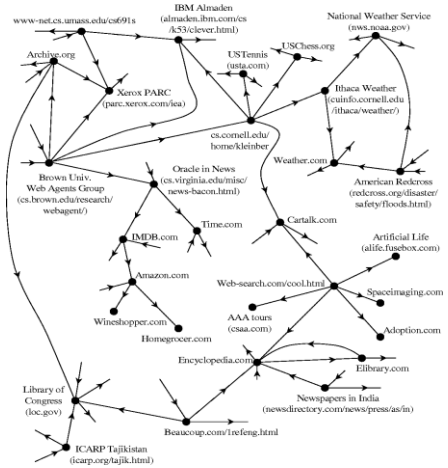
Representing data as graphs

Social Networks

- Many types of data are naturally represented as graphs
  - Social networks: Nodes are people, links are connections

Representing data as graphs

Social Networks

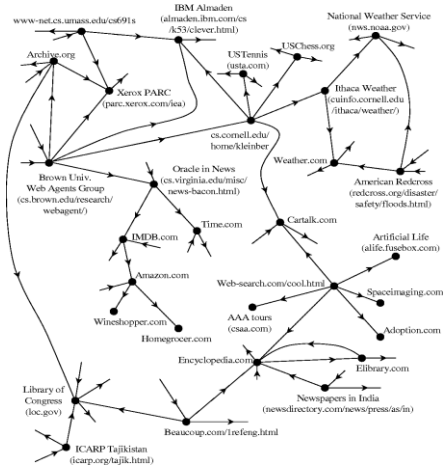


World Wide Web or
Citation Networks

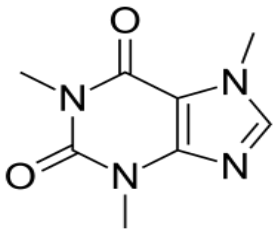
- Many types of data are naturally represented as graphs
  - Social networks: Nodes are people, links are connections
  - World wide web: Sites point to one another
  - Citation networks: Papers cite one another

Representing data as graphs

Social Networks



World Wide Web or
Citation Networks



Molecules

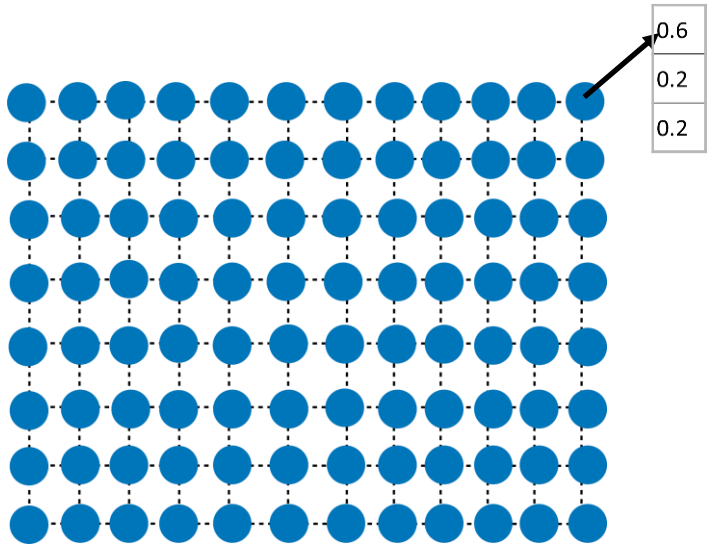
- Many types of data are naturally represented as graphs
  - Social networks: Nodes are people, links are connections
  - World wide web: Sites point to one another
  - Citation networks: Papers cite one another
  - Molecules: Atoms and their connections

Representing data as graphs

- Any data with relational structure can be represented as a graph
  - Data instances with pair-wise relations

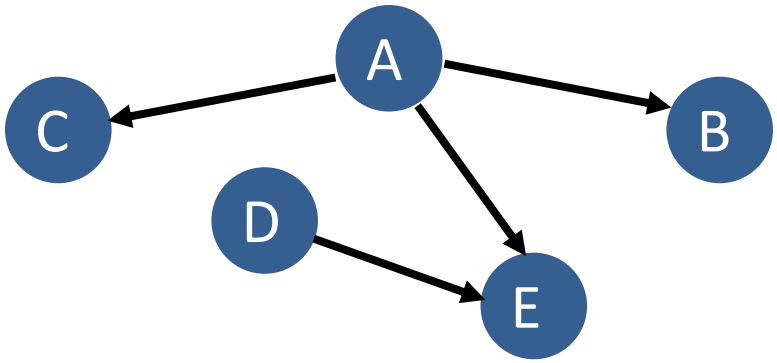
- Like the examples we just saw
- And other, more surprising instances

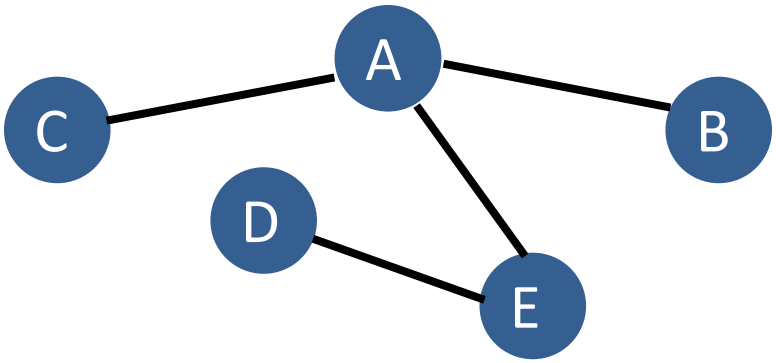
Surprising graphs



- Images are graphs!
  - Nodes are pixel positions
    - Pixel values are node attributes
  - Relations are adjacency
    - Each node is connected to the four adjacent nodes

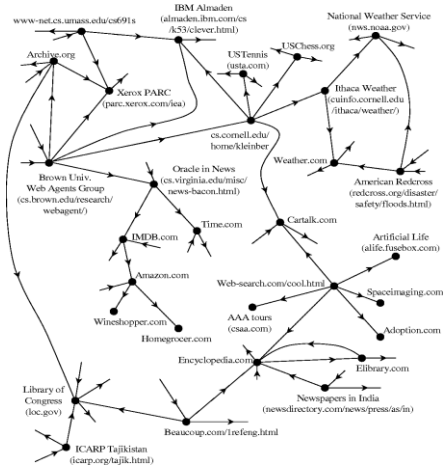
Directed and undirected edges

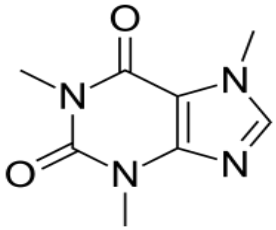




- Edges in a graph can be directed or undirected
  - An undirected edge is actually an edge that points both ways
- We will assume directed edges, but everything generalizes to undirected graphs

Recap: Data can be represented as graphs

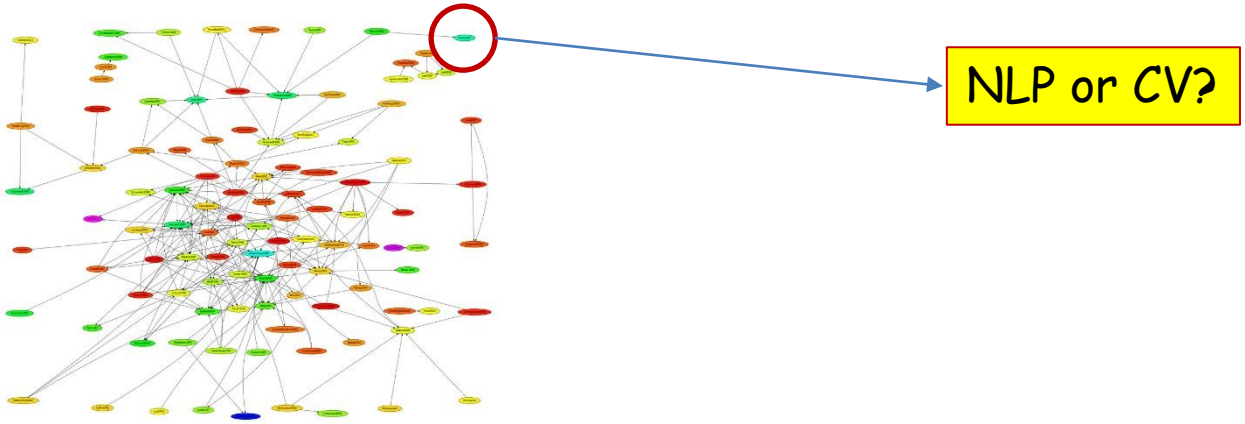




- Many types of data can be represented as graphs
  - With nodes representing instances and edges representing pair-wise relationships
  - Even images and time series can be viewed as graphs

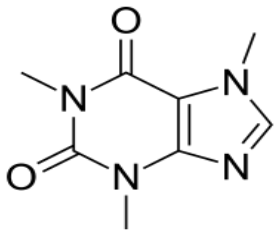
- Classification and prediction tasks can also be performed on graphs

Tasks on a graph



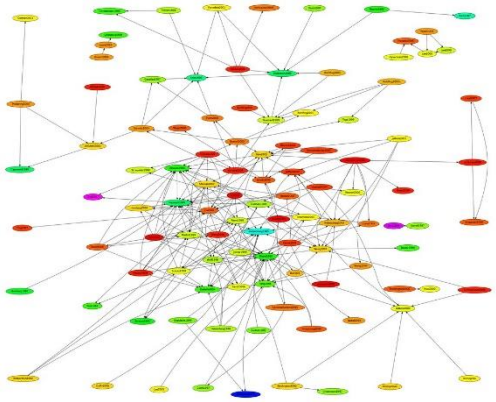
- **Node Classification** : Topic Classification

Tasks on a graph



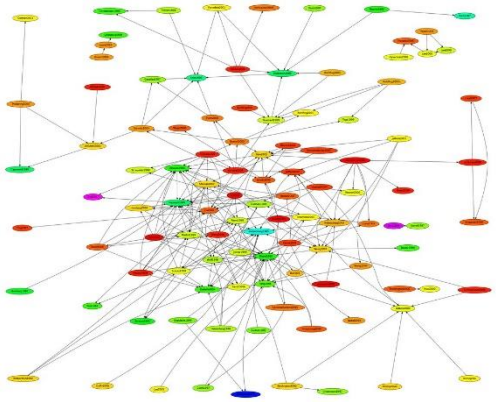
- **Node Classification** : Topic Classification
- **Link Prediction** : Recommendation Systems, predicting bond types

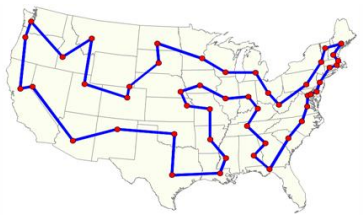
Tasks on a graph



- **Node Classification** : Topic Classification
- **Link Prediction** : Recommendation Systems, predicting bond types
- **Graph Classification** : Image Classification

Tasks on a graph

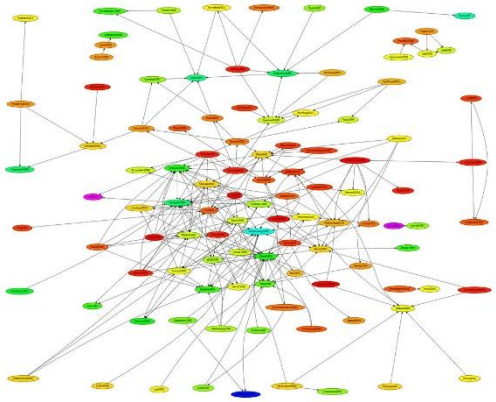


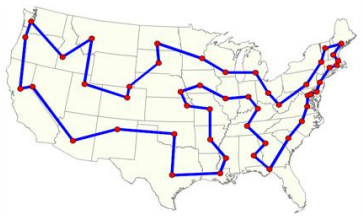


- **Node Classification** : Topic Classification
- **Link Prediction** : Recommendation Systems
- **Graph Classification** : Image Classification

- Also, various combinatorial optimization problems, e.g. travelling salesman problem

Tasks on a graph





- **Node Classification** : Topic Classification
- **Link Prediction** : Recommendation Systems
- **Graph Classification** : Image Classification

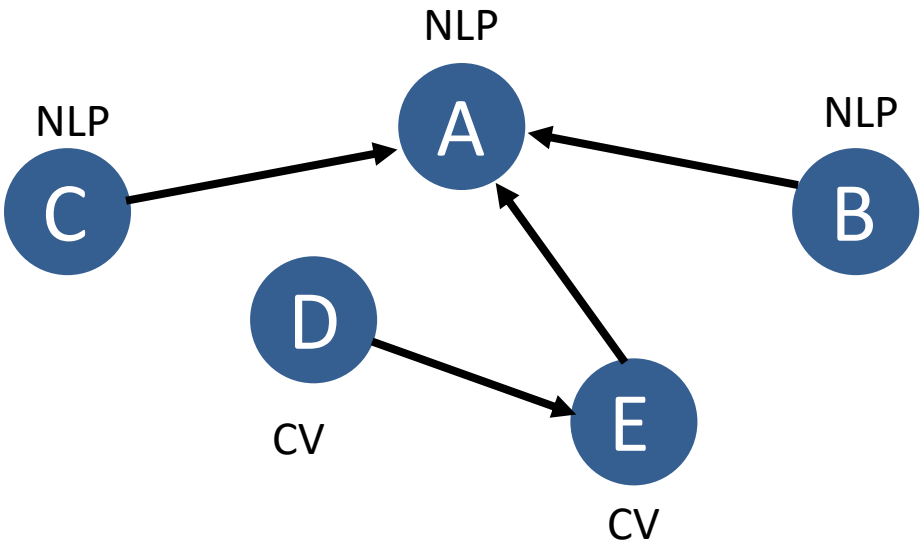
- Also, various combinatorial optimization problems, e.g. travelling salesman problem

All of these tasks can be performed using Graph Neural Networks

Graph Neural Nets (through an example)

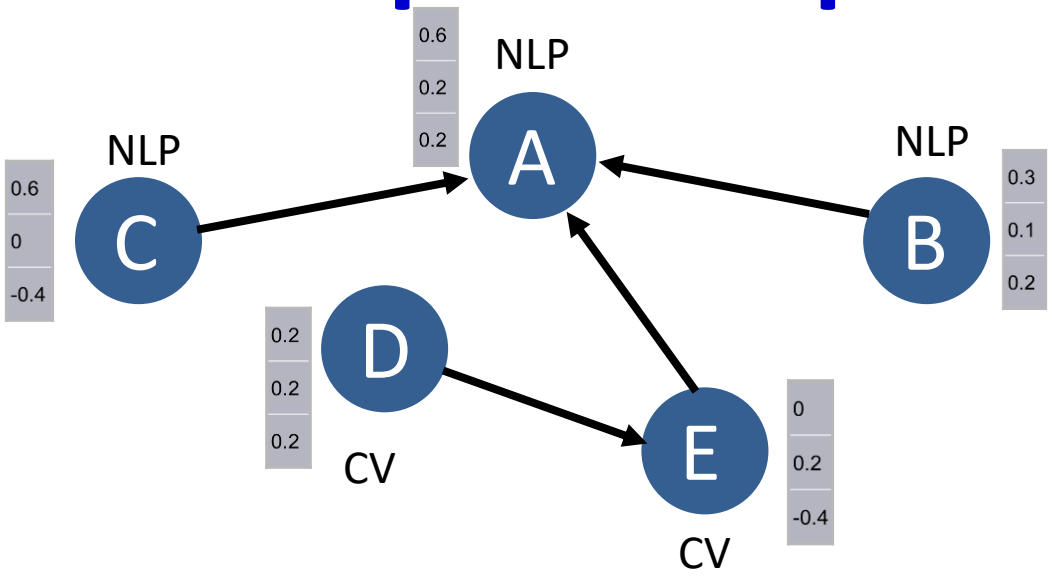
- Given a citation network, classify a paper topic into either Natural Language Processing (NLP) or Computer Vision (CV) paper.
  - Based on its content and its citations
- Problem of node classification (NLP or CV)

Step 0: Setup



- Graph representing a citation network with labels

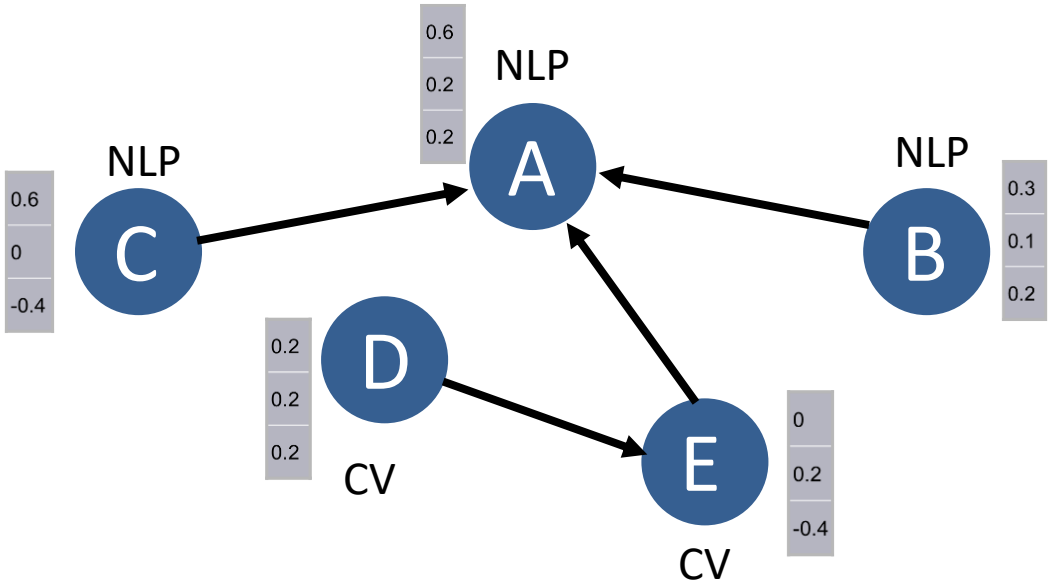
Step 0: Setup



- Graph representing a citation network with labels
- Each node has a feature vector
  - E.g. word count vectors

- **Objective:** Learn to compute an *embedding* for each node in the graph from the node features and graph structure

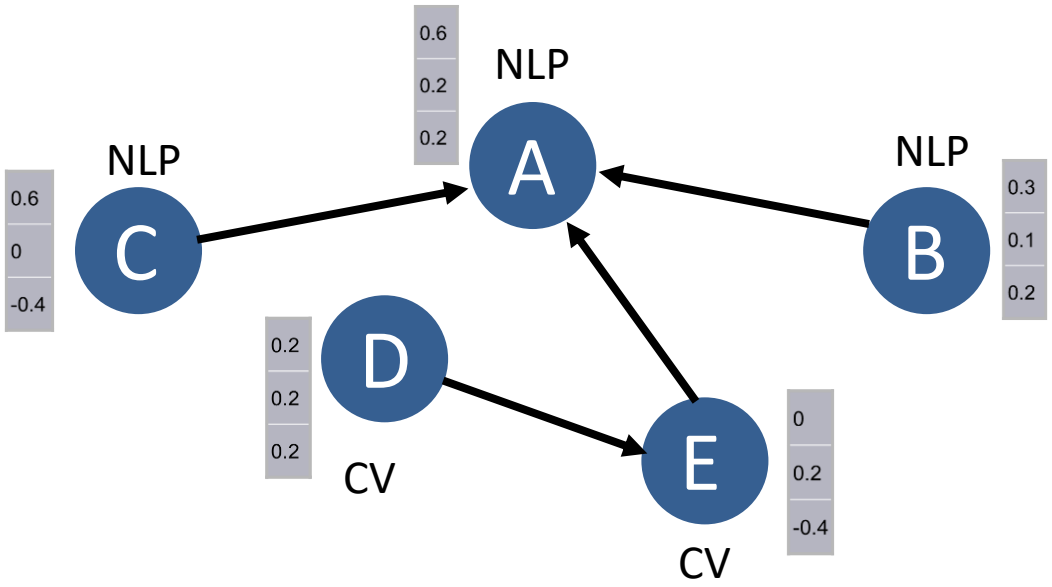
Step 0: Setup



- Graph : $G = (V, E)$
- Node features: $X \in R^{d \times |V|}$

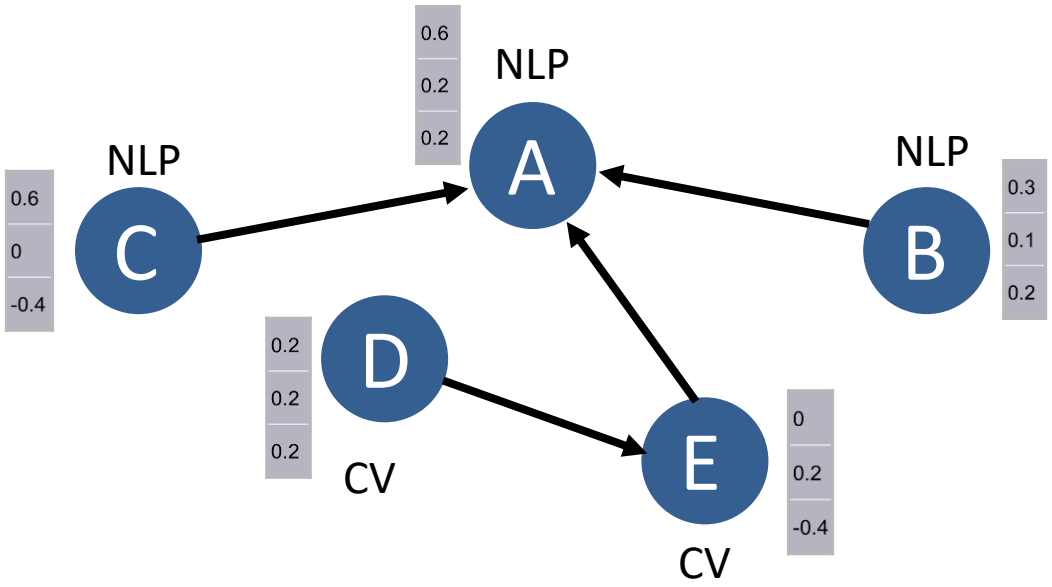
- **To estimate** : embeddings z_u for all nodes $u \in V$

Step 0: Setup



- We will use a multi-layer network which also computes intermediate values:
- Intermediate terms
  - kth-layer node activation $h_u^k \; \forall \; u \in V$
  - kth-layer edge activation $h_{u,v}^k \; \forall \; u, v \in V$

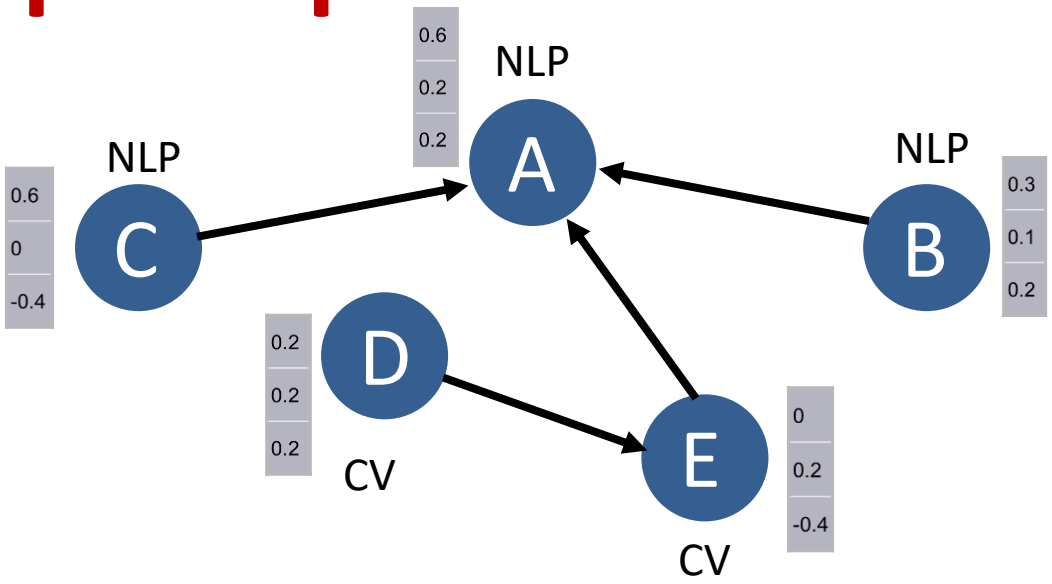
Step 0: Setup



- We will use a multi-layer network which also computes intermediate valu
- Intermediate terr
  - Kth-layer node activation $h_u^k \; \forall \; u \in V$
  - Kth-layer edge activation $h_{u,v}^k \; \forall \; u, v \in V$

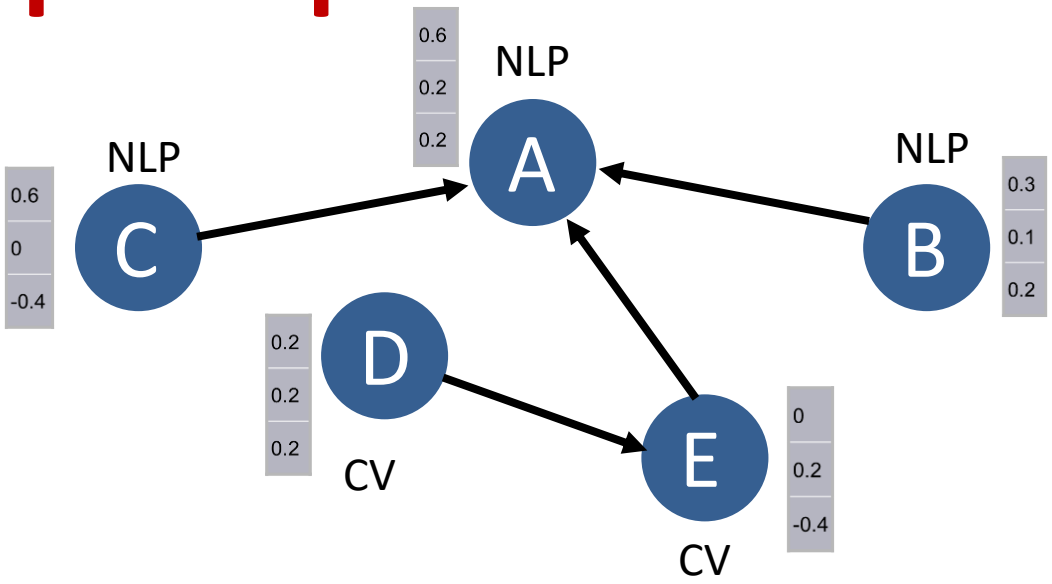
For node classification, only need node embeddings (but discussion generalizes to link-embedding models)

Step 1: Update node vectors



- Next step: Update all the node vectors using "context" information from all their neighbors
- **Aggregate** information from neighboring nodes to compute an incoming "message" m_u^k
- **Update** the activation h_u^k at each node by combining h_u^k with the message m_u^k to obtain the updated vector h_u^{k+1}

Step 1: Update node vectors

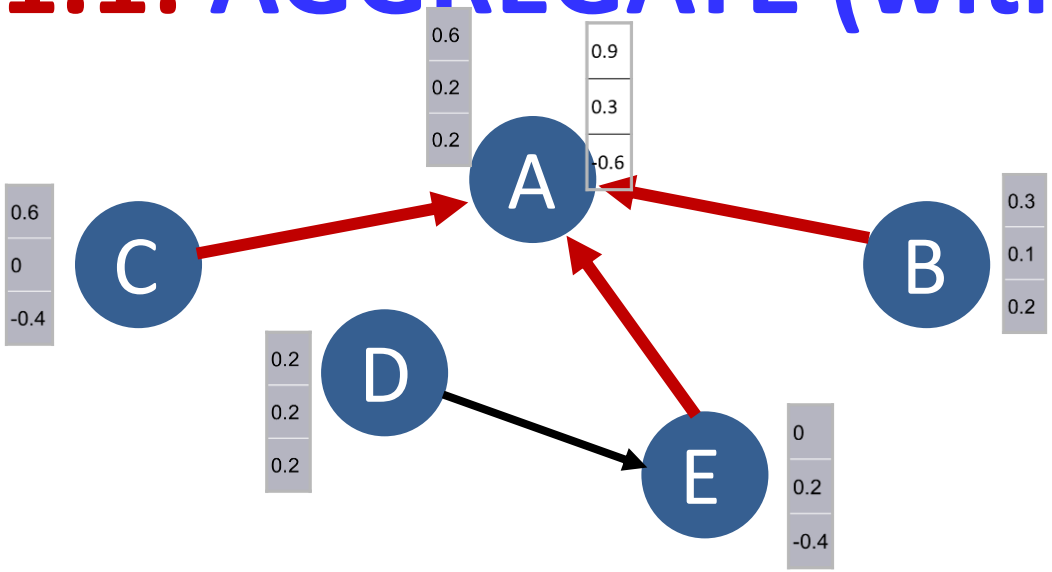


$$m_u^k = \text{AGGREGATE}^k\left(\{h_v^k, \forall v \in Neighbor(u)\}\right)$$

$$h_u^{k+1} = \text{UPDATE}^k\left(h_u^k, m_u^k\right)$$

- AGGREGATE is an order-invariant operation, such as sum or max
- UPDATE is typically a regular MLP layer (linear layer plus activation)

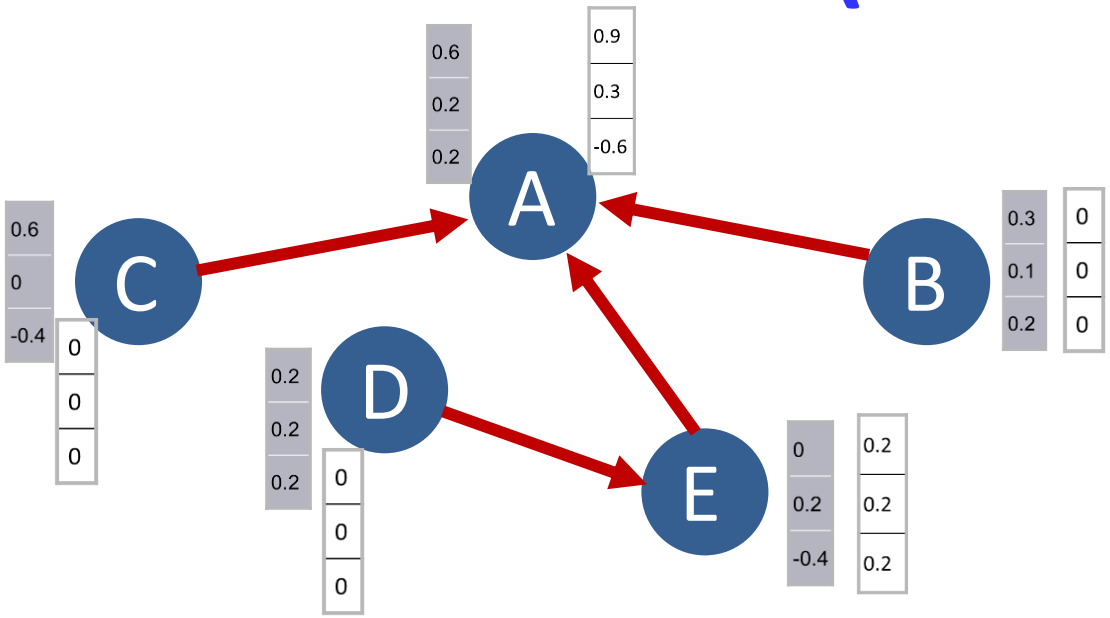
Step 1.1: AGGREGATE (with sum)



$$m_u^k = \sum_{v \in Neighbor(u)} h_v^k$$

$$m_A^k = h_B^k + h_C^k + h_E^k$$

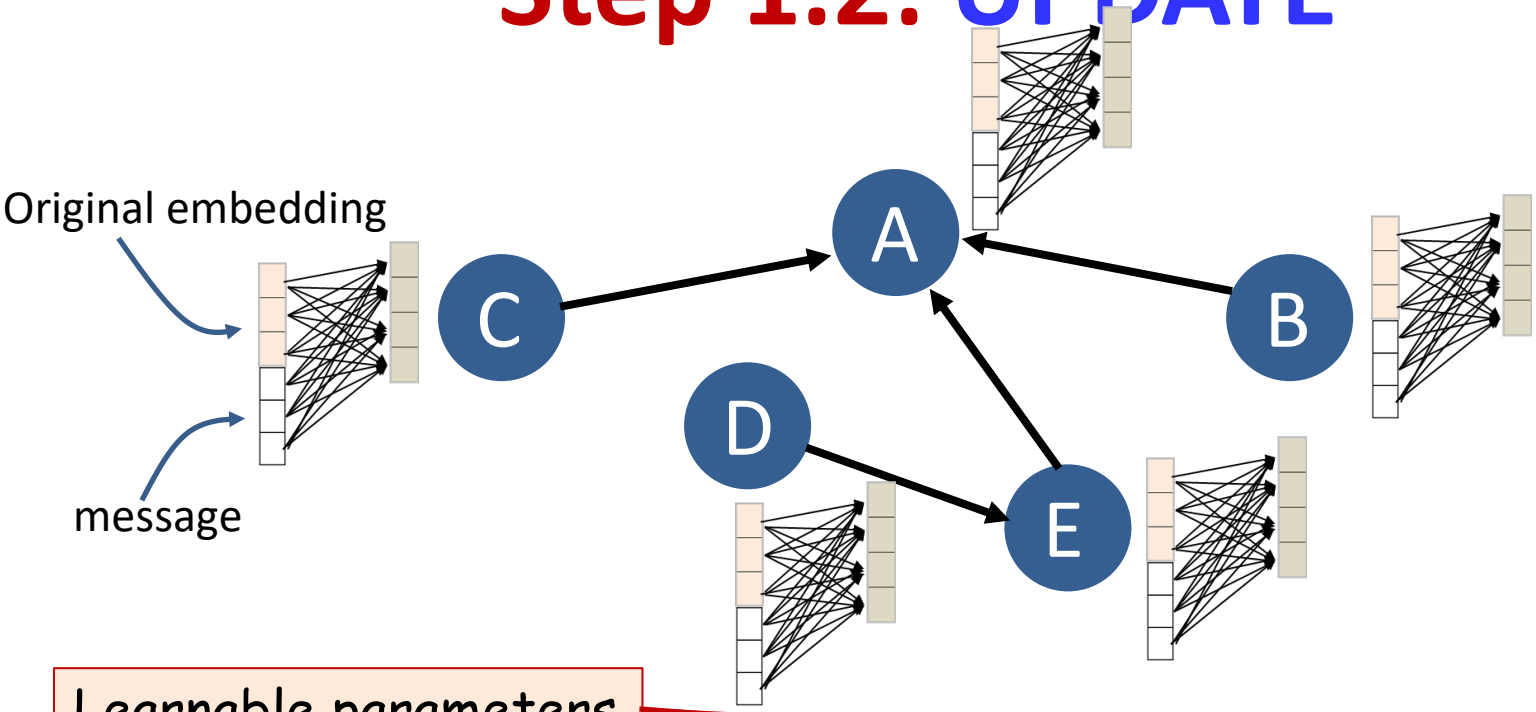
Step 1.1: AGGREGATE (with sum)



$$m_u^k = \sum_{v \in Neighbor(u)} h_v^k$$

- Compute messages for all nodes
  - Some nodes may have zero message in a directed graph

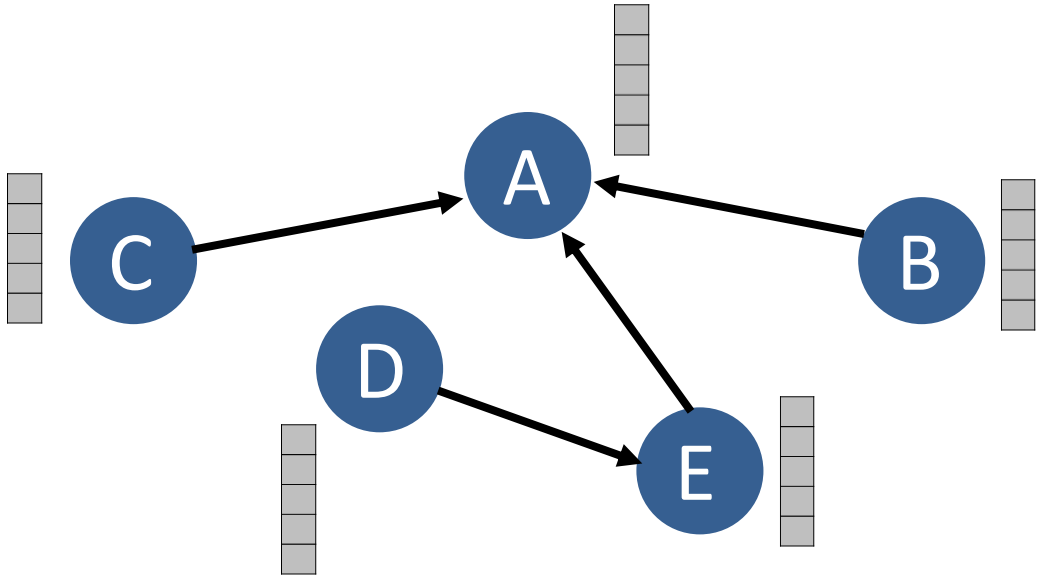
Step 1.2: UPDATE



$$h_u^{k+1} = \sigma\left(W_{self}^k h_u^k + W_{msg}^k m_u^k + b^k\right)$$

- Updates may change the size of the embeddings
- Typical activation functions are tanh and ReLU

Step 1: Estimating embeddings



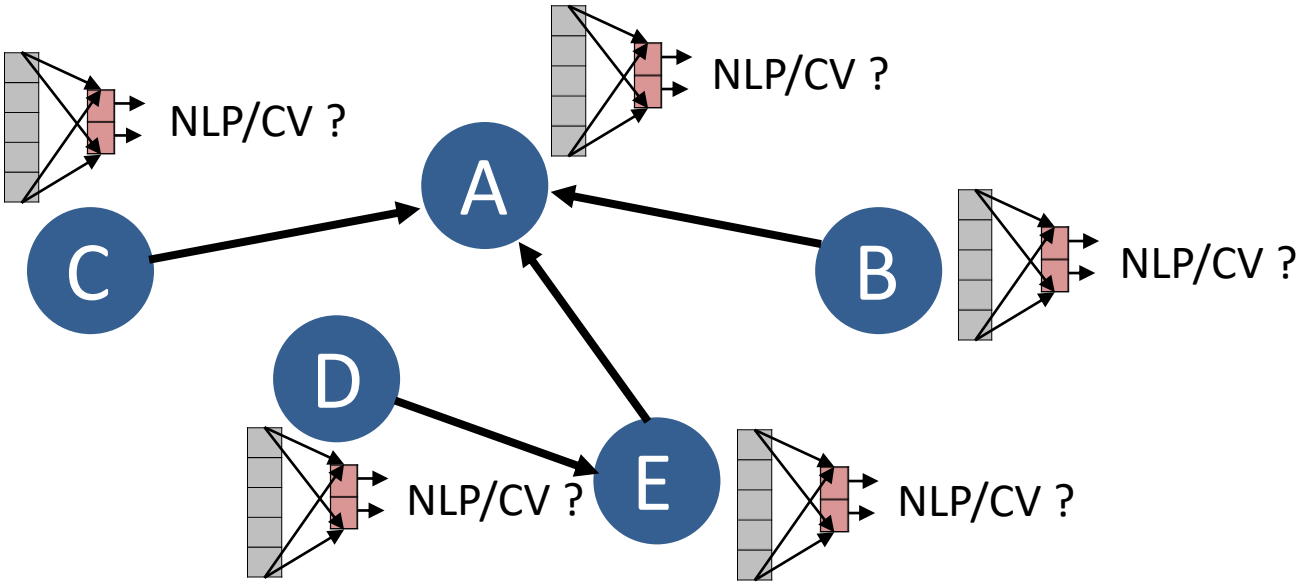
$$m_u^k = \text{AGGREGATE}^k\left(\{h_v^k, \forall v \in Neighbor(u)\}\right)$$

$$h_u^{k+1} = \text{UPDATE}^k\left(h_u^k, m_u^k\right)$$

- After K layers of aggregate/update steps, we obtain our final embedding

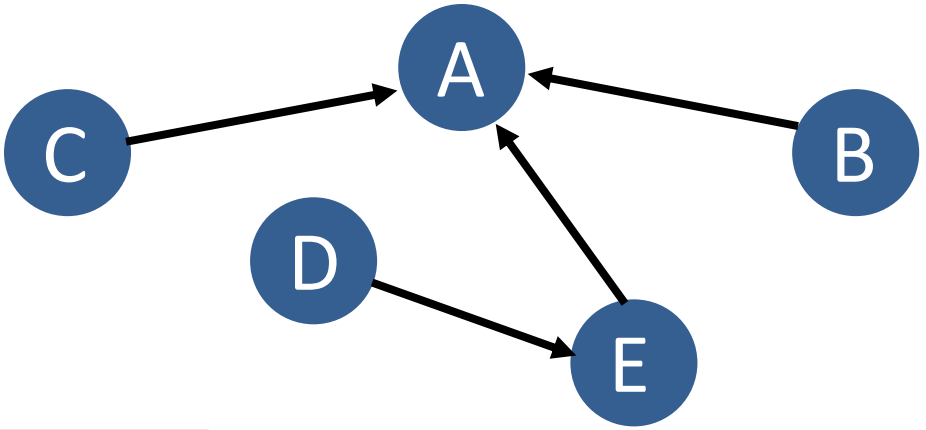
$$z_u = m_u^K$$

Step 2: Classification



- We can add a final classification layer to the embedding for the final classification

Model Parameters

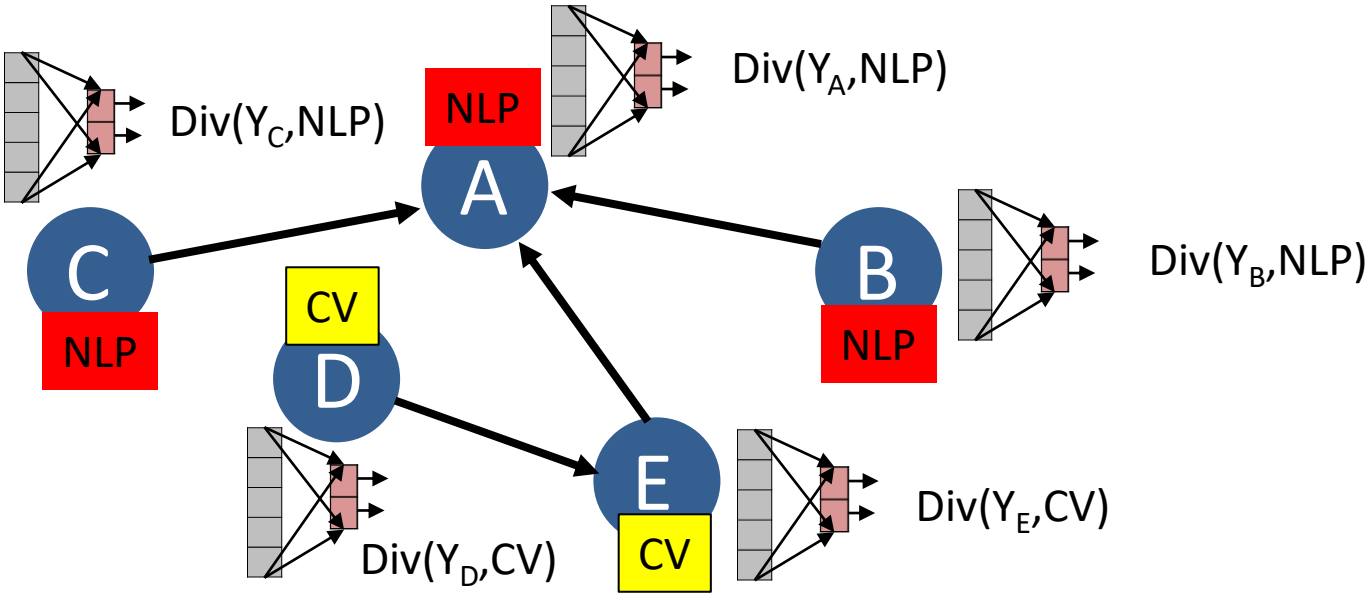


Learnable parameters

$$h_u^{k+1} = \sigma\left(W_{self}^k h_u^k + W_{msg}^k m_u^k + b^k\right)$$

- The learnable parameters of the network are the self and message weights and bias, for all the K layers

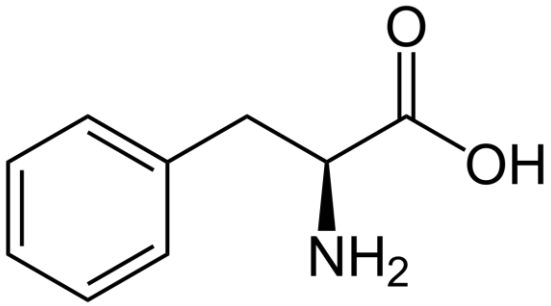
Training and Inference

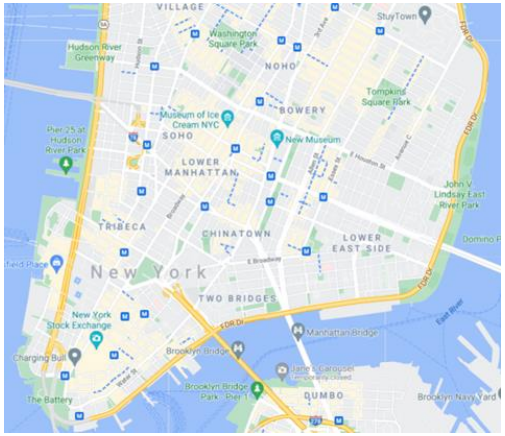


- The divergence with respect to the ground truth labels of nodes (on training data) can minimized via backpropagation, to learn the parameters

- Nodes on novel graphs can now be labelled using these parameters for inference

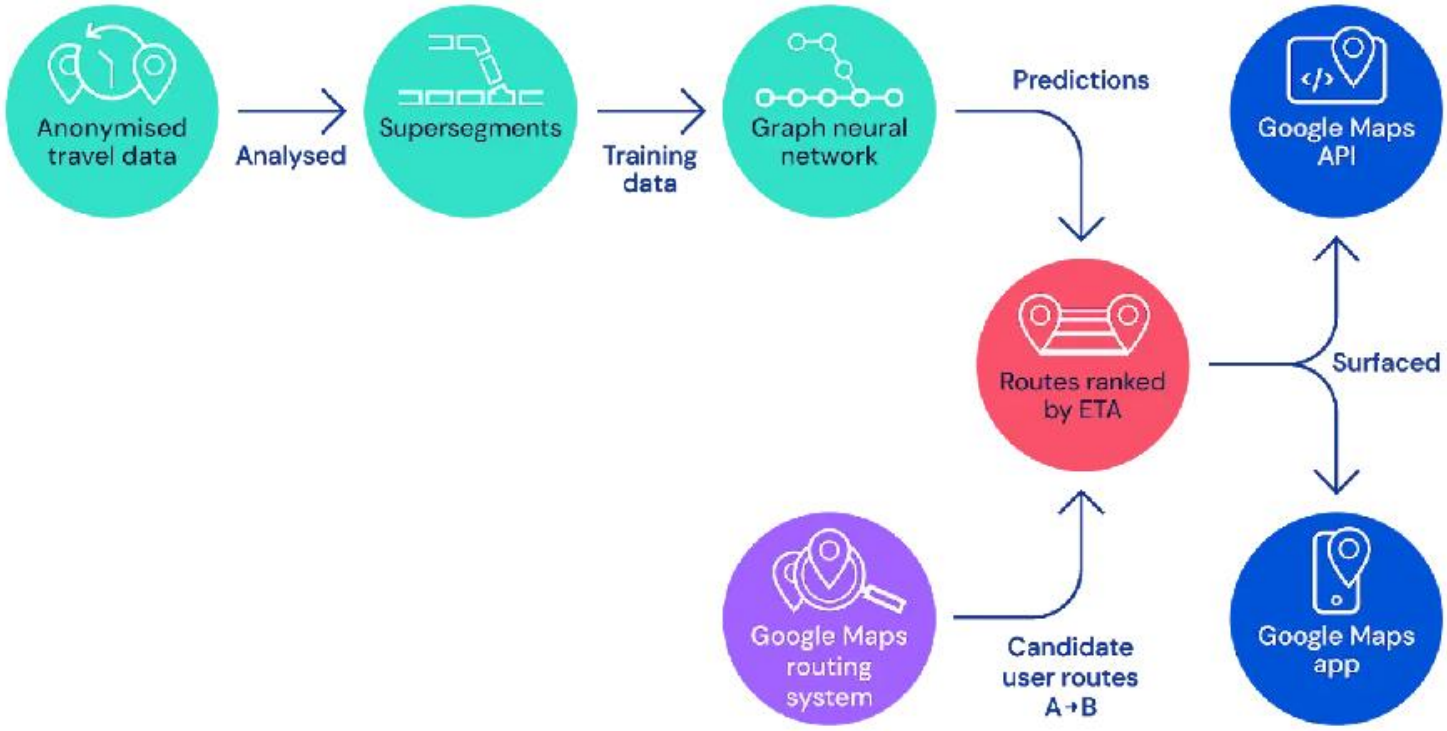
GNN uses





- Molecular properties/materials chemistry/drug design
- Social network analysis
- Maps…

Breakthrough in GNN



The model architecture for determining optimal routes and their travel time.

Image Credit: DeepMind

https://deepmind.com/blog/article/traffic-prediction-with-advanced-graph-neural-networks

Breakthrough in GNN

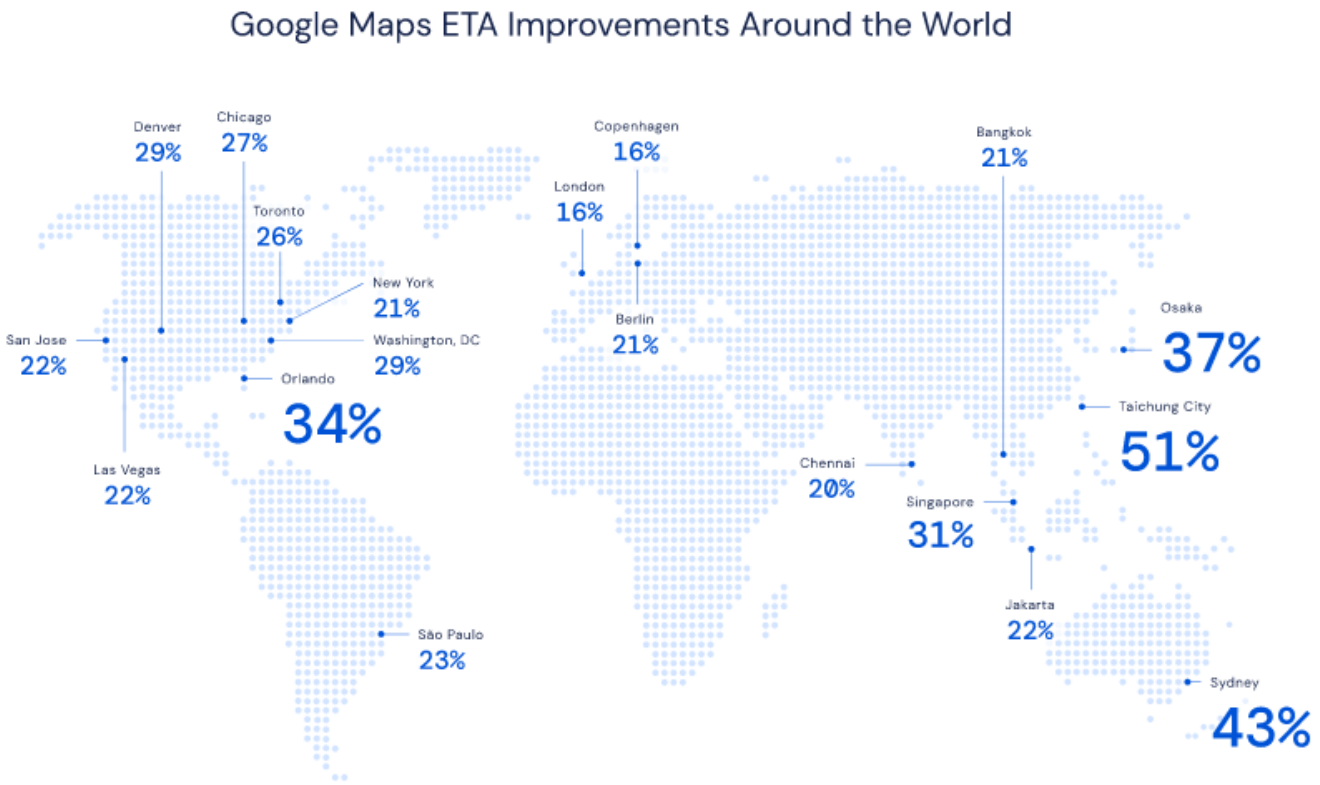


Image Credit: DeepMind

Next up: Representation learning