Sequence to Sequence Models, Attention and Transformers

CS60010: Deep Learning

Abir Das

IIT Kharagpur

Mar 23, 24 and 26, 2022

Agenda

- § Understand basic neural language model, structured prediction and conditional language models
- § Using attention to handle information bottleneck problem
- § Self attention and transformer models

Resources

- CS W182 course by Sergey Levine at UC Berkeley. [Link] [Lecture 11, 12]
- "AI Coffee Break with Letitia" youtube channel [Link]

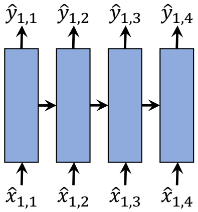
A Basic Neural Language Model

- A language model is a model that assigns probabilites to *sequences* representing texts.
- A language model often is used to generate texts.

Source: CS W182 course, Sergey Levine, UC Berkeley

A Basic Neural Language Model

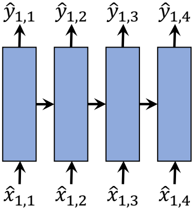
- A language model is a model that assigns probabilites to *sequences* representing texts.
- A language model often is used to generate texts.
- Many language models can be represented as the following general architecture:



Source: CS W182 course, Sergey Levine, UC Berkeley

A Basic Neural Language Model

- A language model is a model that assigns probabilites to *sequences* representing texts.
- A language model often is used to generate texts.
- Many language models can be represented as the following general architecture:



- Why does it need multiple outputs and multiple outputs?
- Most problems that require multiple outputs have strong *dependencies* between these outputs.
- This is sometimes referred to as *structured prediction*.

Source: CS W182 course, Sergey Levine, UC Berkeley

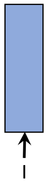
A Basic Neural Language Model

- Lets say we have a text generation model which generates texts given an initial prompt.
- Let the world of the language model consists of the following three sentences.
  - I think therefore I am
  - I like machine learning
  - I am not just a neural network

Source: CS W182 course, Sergey Levine, UC Berkeley

A Basic Neural Language Model

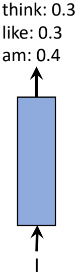
- Lets say we have a text generation model which generates texts given an initial prompt.
- Let the world of the language model consists of the following three sentences.
  - I think therefore I am
  - I like machine learning
  - I am not just a neural network



Source: CS W182 course, Sergey Levine, UC Berkeley

A Basic Neural Language Model

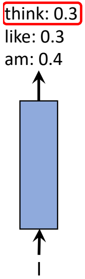
- Lets say we have a text generation model which generates texts given an initial prompt.
- Let the world of the language model consists of the following three sentences.
  - I think therefore I am
  - I like machine learning
  - I am not just a neural network



Source: CS W182 course, Sergey Levine, UC Berkeley

A Basic Neural Language Model

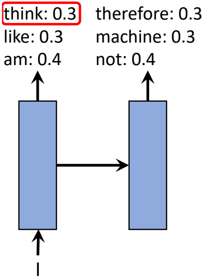
- Lets say we have a text generation model which generates texts given an initial prompt.
- Let the world of the language model consists of the following three sentences.
  - I think therefore I am
  - I like machine learning
  - I am not just a neural network



Source: CS W182 course, Sergey Levine, UC Berkeley

A Basic Neural Language Model

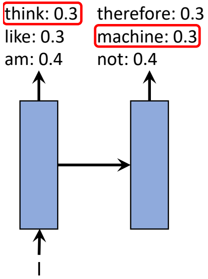
- Lets say we have a text generation model which generates texts given an initial prompt.
- Let the world of the language model consists of the following three sentences.
  - I think therefore I am
  - I like machine learning
  - I am not just a neural network



Source: CS W182 course, Sergey Levine, UC Berkeley

A Basic Neural Language Model

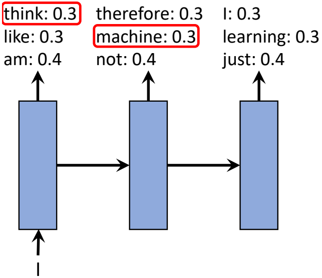
- Lets say we have a text generation model which generates texts given an initial prompt.
- Let the world of the language model consists of the following three sentences.
  - I think therefore I am
  - I like machine learning
  - I am not just a neural network



Source: CS W182 course, Sergey Levine, UC Berkeley

A Basic Neural Language Model

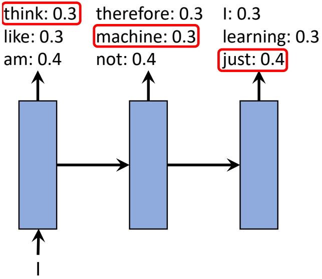
- Lets say we have a text generation model which generates texts given an initial prompt.
- Let the world of the language model consists of the following three sentences.
  - I think therefore I am
  - I like machine learning
  - I am not just a neural network



Source: CS W182 course, Sergey Levine, UC Berkeley

A Basic Neural Language Model

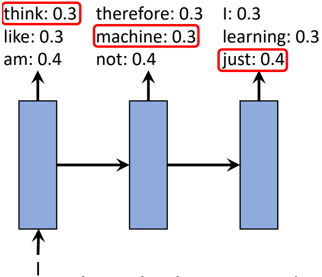
- Lets say we have a text generation model which generates texts given an initial prompt.
- Let the world of the language model consists of the following three sentences.
  - I think therefore I am
  - I like machine learning
  - I am not just a neural network



Source: CS W182 course, Sergey Levine, UC Berkeley

A Basic Neural Language Model

- Lets say we have a text generation model which generates texts given an initial prompt.
- Let the world of the language model consists of the following three sentences.
  - I think therefore I am
  - I like machine learning
  - I am not just a neural network

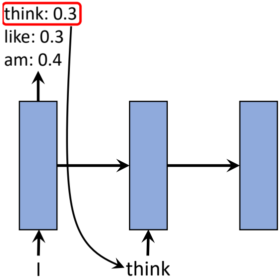


- Output is nonsensical even though the network did great job individually.

Source: CS W182 course, Sergey Levine, UC Berkeley

A Basic Neural Language Model

- Fix: feed sampled output as input to next timestep.

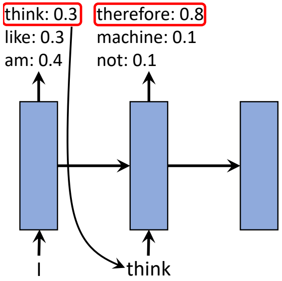


- Now the network knows, it is predicting the third word of a sentence where the first two words are 'I think'

Source: CS W182 course, Sergey Levine, UC Berkeley

A Basic Neural Language Model

- Fix: feed sampled output as input to next timestep.

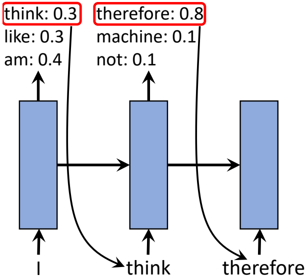


- Now the network knows, it is predicting the third word of a sentence where the first two words are 'I think'

Source: CS W182 course, Sergey Levine, UC Berkeley

A Basic Neural Language Model

- Fix: feed sampled output as input to next timestep.

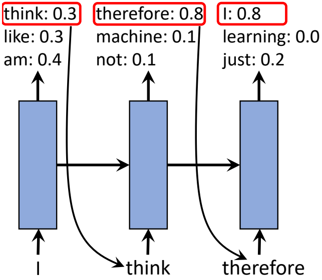


- Now the network knows, it is predicting the third word of a sentence where the first two words are 'I think'

Source: CS W182 course, Sergey Levine, UC Berkeley

A Basic Neural Language Model

- Fix: feed sampled output as input to next timestep.

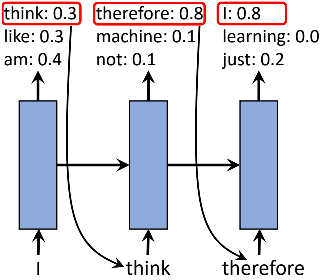


- Now the network knows, it is predicting the third word of a sentence where the first two words are 'I think'

Source: CS W182 course, Sergey Levine, UC Berkeley

A Basic Neural Language Model

- Fix: feed sampled output as input to next timestep.

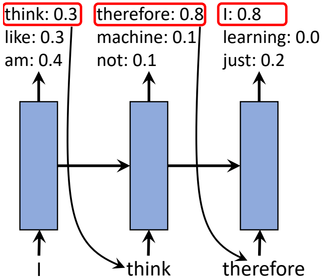


- Now the network knows, it is predicting the third word of a sentence where the first two words are 'I think'
- Key idea: Past outputs should influence future outputs.
- Also known as autoregressive models.

Source: CS W182 course, Sergey Levine, UC Berkeley

A Basic Neural Language Model

- Fix: feed sampled output as input to next timestep.



- Now the network knows, it is predicting the third word of a sentence where the first two words are 'I think'
- Key idea: Past outputs should influence future outputs.
- Also known as autoregressive models.
- During training: input is the sequence and output is the same sequence offset by 1.

Source: CS W182 course, Sergey Levine, UC Berkeley

A Basic Neural Language Model

§ How are the training sequences represented.

- I think therefore I am
- I like machine learning
- I am not just a neural network

Source: CS W182 course, Sergey Levine, UC Berkeley

A Basic Neural Language Model

- How are the training sequences represented.
  - I think therefore I am
  - I like machine learning
  - I am not just a neural network
- Simplest: tokenize the sentence (each word is a token) and use onehot vector representation.

$$x_{1,i} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

- More complex: word embeddings (we'll cover this later)

Source: CS W182 course, Sergey Levine, UC Berkeley

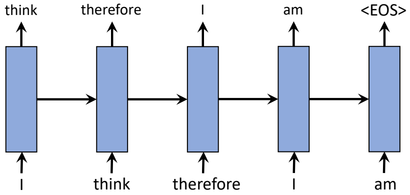
A Few Details

§ How does the model know it has to stop generating words?

Source: CS W182 course, Sergey Levine, UC Berkeley

A Few Details

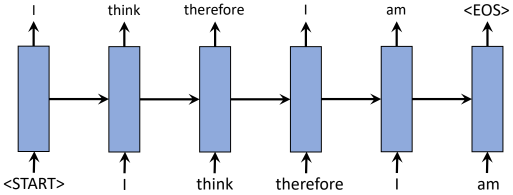
- How does the model know it has to stop generating words?
- During training, add a special token ⟨EOS⟩ at the end of the sequence.
- During testing when it produces an ⟨EOS⟩ token, we know the sentence is complete.



Source: CS W182 course, Sergey Levine, UC Berkeley

A Few Details

- § How does the model know it has to stop generating words?
- § During training, add a special token ⟨EOS⟩ at the end of the sequence.
- § During testing when it produces an ⟨EOS⟩ token, we know the sentence is complete.



- § Similarly a special ⟨START⟩ token is introduced to kick of the start of a sentence.

Source: CS W182 course, Sergey Levine, UC Berkeley

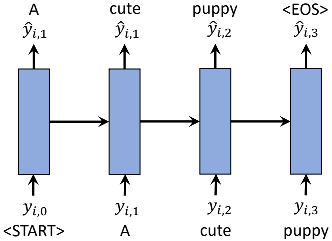
Conditional Language Models

- In conditional language models, text is generated conditioned on some input.
- For example, image captioning conditions text generation on image.

Source: CS W182 course, Sergey Levine, UC Berkeley

Conditional Language Models

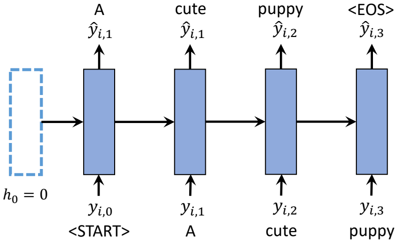
- In conditional language models, text is generated conditioned on some input.
- For example, image captioning conditions text generation on image.



Source: CS W182 course, Sergey Levine, UC Berkeley

Conditional Language Models

- In conditional language models, text is generated conditioned on some input.
- For example, image captioning conditions text generation on image.

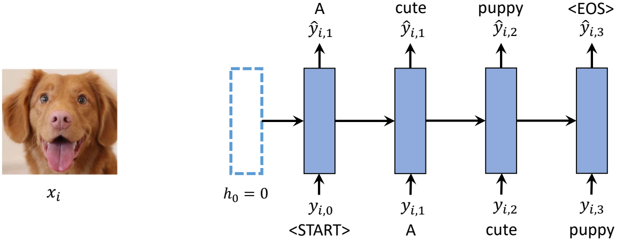


- Previously, initial hidden state of RNN was 0.

Source: CS W182 course, Sergey Levine, UC Berkeley

Conditional Language Models

- In conditional language models, text is generated conditioned on some input.
- For example, image captioning conditions text generation on image.

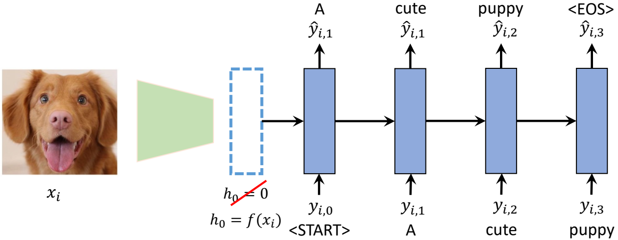


- Previously, initial hidden state of RNN was 0.

Source: CS W182 course, Sergey Levine, UC Berkeley

Conditional Language Models

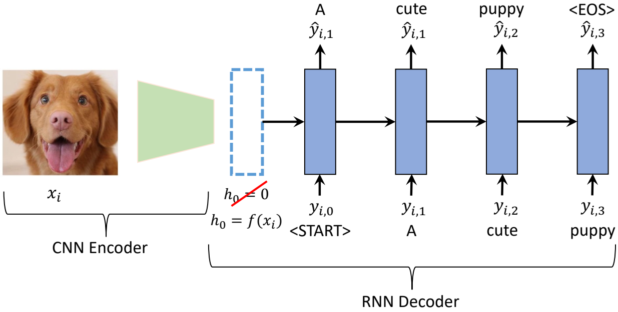
- In conditional language models, text is generated conditioned on some input.
- For example, image captioning conditions text generation on image.



- Previously, initial hidden state of RNN was 0.
- Now, we set the intial state of the RNN as an encoded representation from the image, obtained by passing it thorugh a convnet.
- Both RNN and ConvNet are trained end-to-end.

Source: CS W182 course, Sergey Levine, UC Berkeley

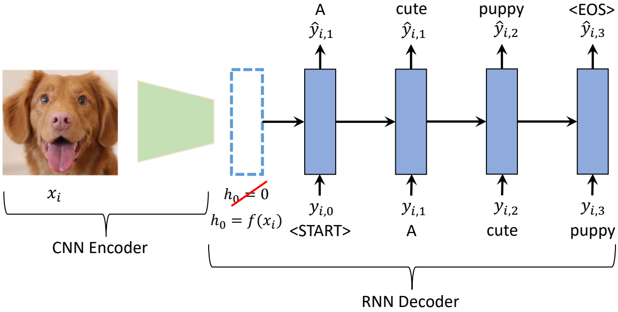
Conditional Language Models



- *CNN Encoder* 'summarizes' what is going on in the image and *RNN Decoder* expresses the content of the image in words.

Source: CS W182 course, Sergey Levine, UC Berkeley

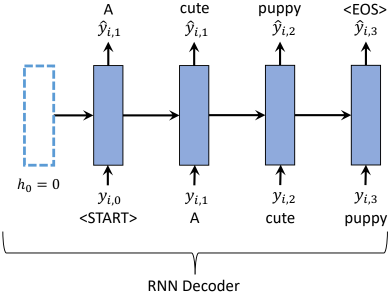
Conditional Language Models



- *CNN Encoder* 'summarizes' what is going on in the image and *RNN Decoder* expresses the content of the image in words.
- Training data: Paired image-text data.

Source: CS W182 course, Sergey Levine, UC Berkeley

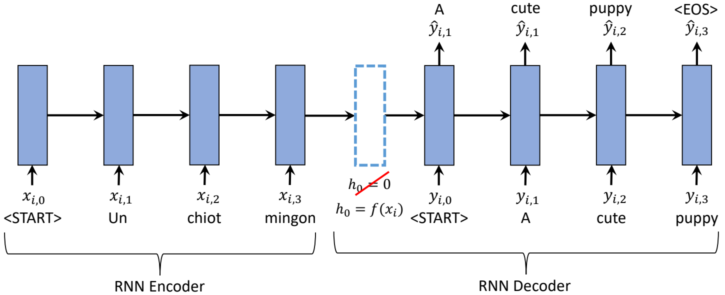
What if we condition on another sequence?



§ The RNN Decoder can also be conditioned on another sequence, say text from another language.

Source: CS W182 course, Sergey Levine, UC Berkeley

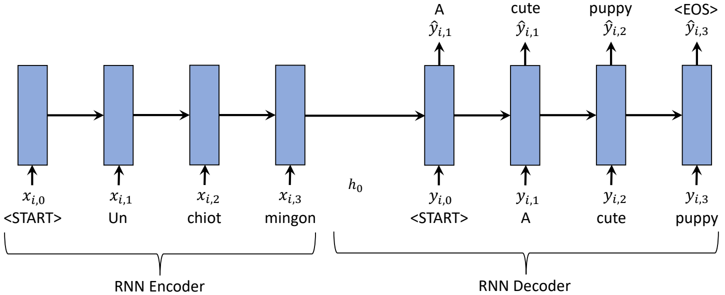
What if we condition on another sequence?



- The RNN Decoder can also be conditioned on another sequence, say text from another language.
- The first RNN reads in French, produces h_0 and the second RNN takes h_0 and produces English text.
- The encoder is also RNN based.

Source: CS W182 course, Sergey Levine, UC Berkeley

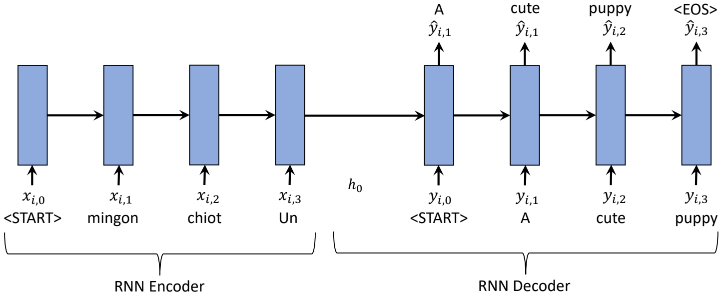
What if we condition on another sequence?



- The RNN Decoder can also be conditioned on another sequence, say text from another language.
- The first RNN reads in French, produces h_0 and the second RNN takes h_0 and produces English text.
- The encoder is also RNN based.
- h_0 is only 'virtual'. $\langle$EOS$\rangle$ token in French doubles as $\langle$START$\rangle$ token in English.

Source: CS W182 course, Sergey Levine, UC Berkeley

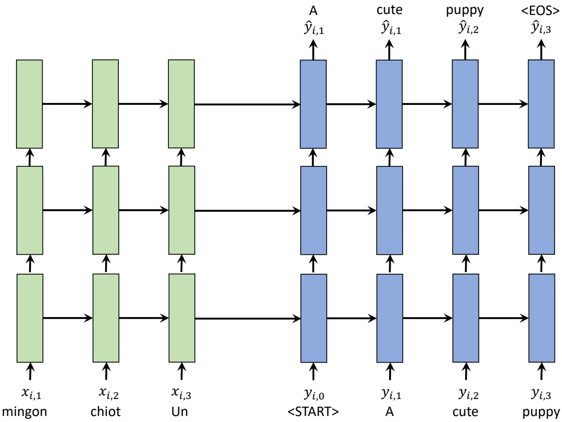
A Few Details



- Sometimes, the encoder RNN reads the source language sentence in reverse.
- Typically two separate RNNs (with different weights) are used.
- Both the RNNS are trained end-to-end on paired data (*e.g.*, pairs of French and English sentences)

Source: CS W182 course, Sergey Levine, UC Berkeley

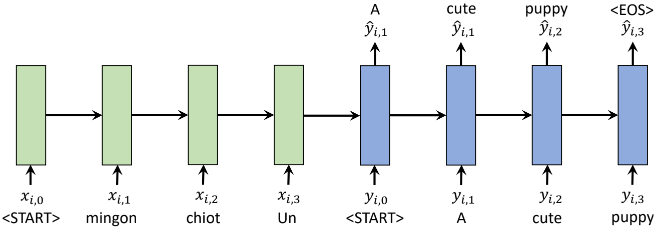
A Few Details



- RNNs can be stacked.
- Each RNN layer can use LSTM cells (or GRU)

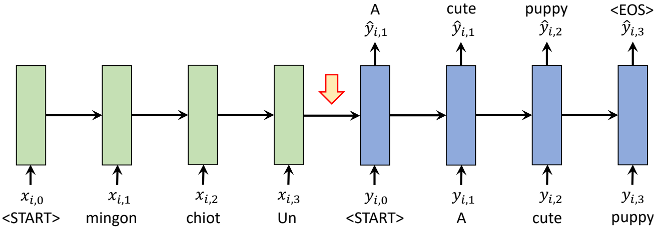
Source: CS W182 course, Sergey Levine, UC Berkeley

The Bottleneck Problem



Source: CS W182 course, Sergey Levine, UC Berkeley

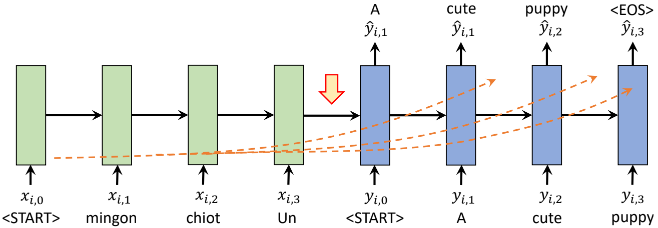
The Bottleneck Problem



- All the information about the source sequence is contained only in the activation at the begining of the decoding.
- Entire decoding is based on this initial information only.

Source: CS W182 course, Sergey Levine, UC Berkeley

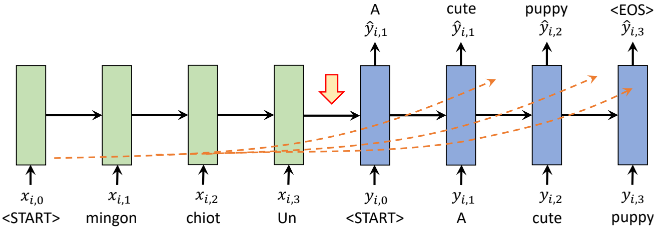
The Bottleneck Problem



- All the information about the source sequence is contained only in the activation at the begining of the decoding.
- Entire decoding is based on this initial information only.
- For long and complex sequences, it will help if the decoder can 'peek' into the input sequence time and again during decoding.

Source: CS W182 course, Sergey Levine, UC Berkeley

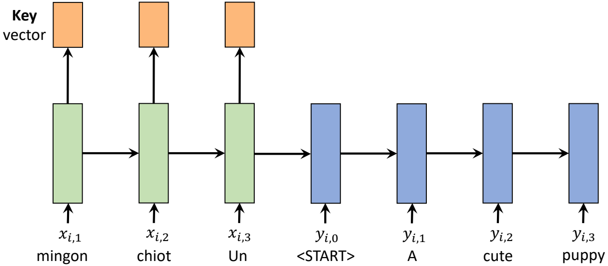
The Bottleneck Problem



- All the information about the source sequence is contained only in the activation at the begining of the decoding.
- Entire decoding is based on this initial information only.
- For long and complex sequences, it will help if the decoder can 'peek' into the input sequence time and again during decoding.
- How can we do this?

Source: CS W182 course, Sergey Levine, UC Berkeley

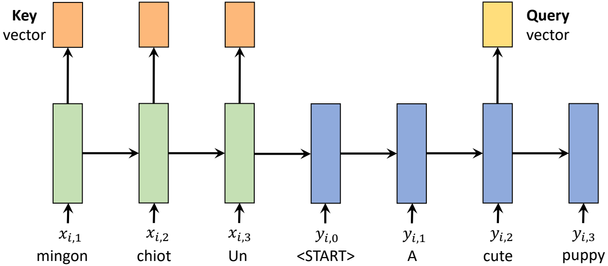
Can we 'peek' at the Input



- key vector represents what type of information is present at that step
- It is *learned* from the hidden state via some function (*e.g.*, lin+ReLU)

Source: CS W182 course, Sergey Levine, UC Berkeley

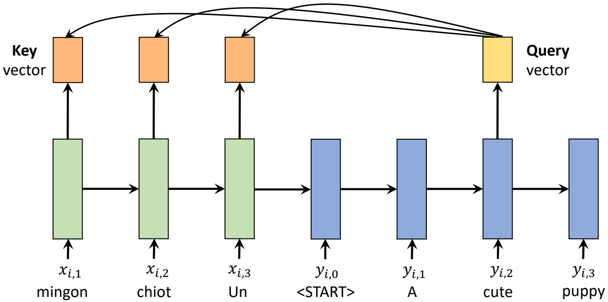
Can we 'peek' at the Input



- key vector represents what type of information is present at that step
- It is *learned* from the hidden state via some function (*e.g.*, lin+ReLU)
- query vector represents what we are looking for at that step

Source: CS W182 course, Sergey Levine, UC Berkeley

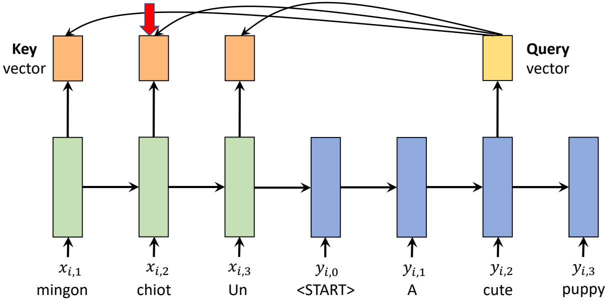
Can we 'peek' at the Input



- key vector represents what type of information is present at that step
- It is *learned* from the hidden state via some function (*e.g.*, lin+ReLU)
- query vector represents what we are looking for at that step
- A query is compared with each key to find the closest one
- This tells, which timestep in the input is the most relevent to this timestep in the decoding process

Source: CS W182 course, Sergey Levine, UC Berkeley

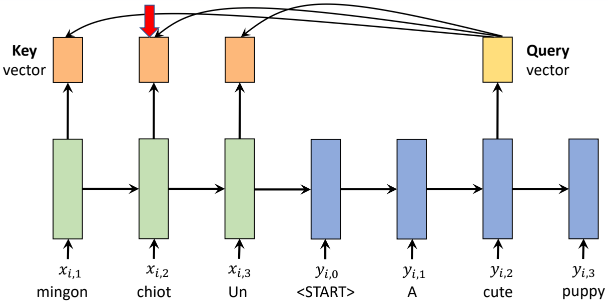
Can we 'peek' at the Input



- key vector represents what type of information is present at that step
- It is *learned* from the hidden state via some function (*e.g.*, lin+ReLU)
- query vector represents what we are looking for at that step
- A query is compared with each key to find the closest one
- This tells, which timestep in the input is the most relevent to this timestep in the decoding process
- The corresponding hidden state is sent to the decoder

Source: CS W182 course, Sergey Levine, UC Berkeley

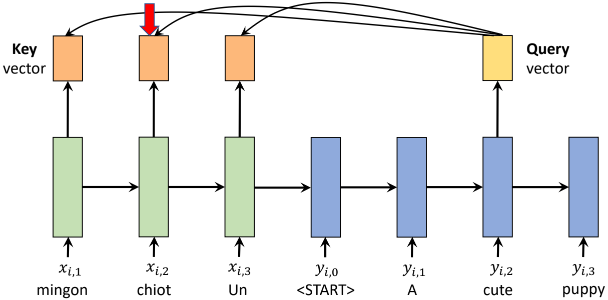
Can we 'peek' at the Input



- **(crude) intuition**: key might encode "the subject of the sentence", and query might ask for "the subject of the sentence"

Source: CS W182 course, Sergey Levine, UC Berkeley

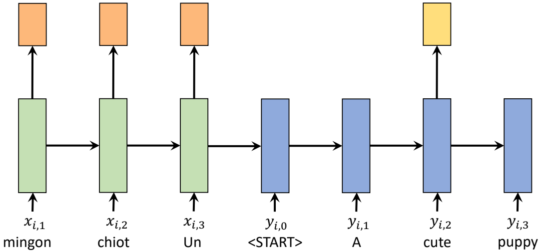
Can we ‘peek’ at the Input



- **(crude) intuition**: key might encode “the subject of the sentence”, and query might ask for “the subject of the sentence”
- What keys and queries mean is **learned** as a part of the training process – we do not have to select it manually!

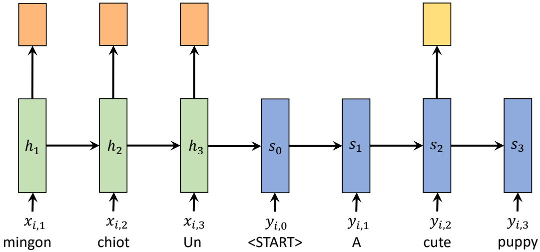
Source: CS W182 course, Sergey Levine, UC Berkeley

Attention



Source: CS W182 course, Sergey Levine, UC Berkeley

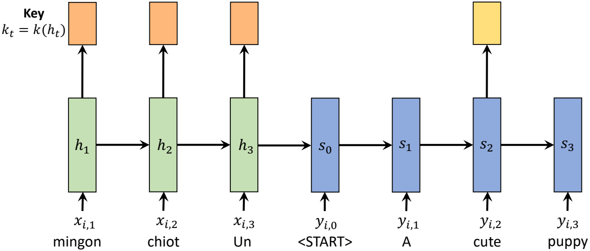
Attention



§ Letter h and s denote hidden states of encoder and decoder respectively

Source: CS W182 course, Sergey Levine, UC Berkeley

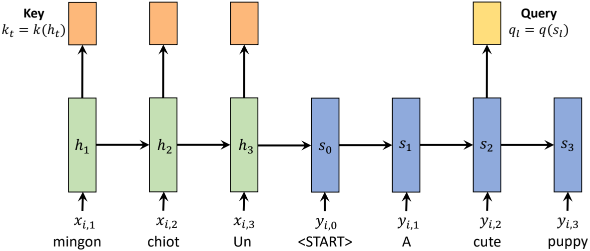
Attention



- Letter h and s denote hidden states of encoder and decoder respectively
- Key k_t at each timestep t is some learnable function of h_t, *e.g.*, $k_t = \sigma(W_k h_t + b_k)$

Source: CS W182 course, Sergey Levine, UC Berkeley

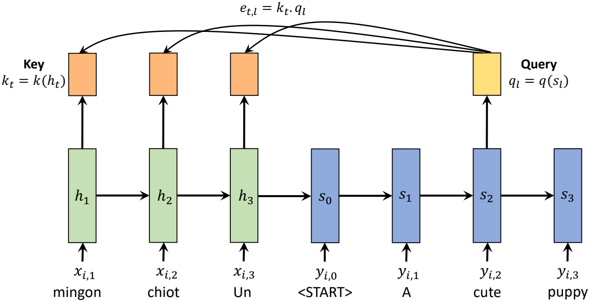
Attention



- Letter h and s denote hidden states of encoder and decoder respectively
- Key k_t at each timestep t is some learnable function of h_t, *e.g.*, $k_t = \sigma(W_k h_t + b_k)$
- Similarly query q_l is some learnable function of decoder state s_l

Source: CS W182 course, Sergey Levine, UC Berkeley

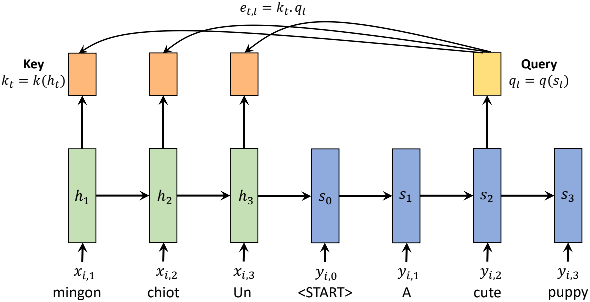
Attention



- Letter h and s denote hidden states of encoder and decoder respectively
- Key k_t at each timestep t is some learnable function of h_t, *e.g.*, $k_t = \sigma(W_k h_t + b_k)$
- Similarly query q_l is some learnable function of decoder state s_l
- Attention $e_{t,l}$ measures the similarity between the key and the query and is given by the dot product between them
- Intuitively, we want to pull out the hidden state h_t for the timestep t at which $e_{t,l}$ is largest

Source: CS W182 course, Sergey Levine, UC Berkeley

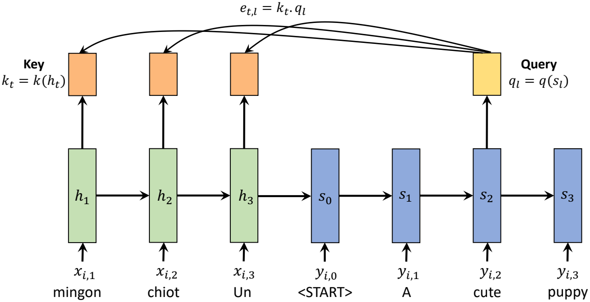
Attention



§ Intuitively, send h_t for $\arg\max_t e_{t,l}$ to decoder at step l

Source: CS W182 course, Sergey Levine, UC Berkeley

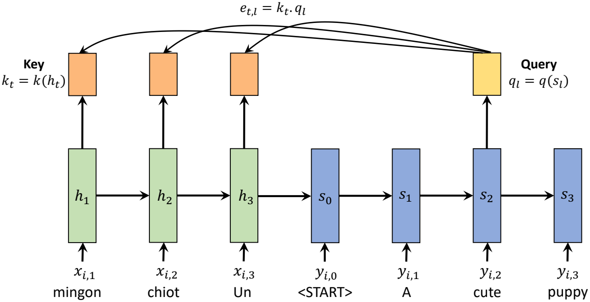
Attention



- Intuitively, send h_t for $\arg\max_t e_{t,l}$ to decoder at step l
- '$\arg\max$' is not differentiable, we will not be able to train the network.
- We will use softmax: $\alpha_{.,l} = \text{softmax}(e_{.,l})$, where $\alpha_{t,l} = \dfrac{\exp(e_{t,l})}{\sum_{t'} \exp(e_{t',l})}$

Source: CS W182 course, Sergey Levine, UC Berkeley

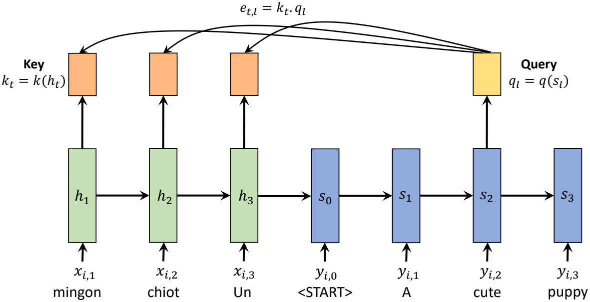
Attention



- Intuitively, send h_t for $\arg\max_t e_{t,l}$ to decoder at step l
- '$\arg\max$' is not differentiable, we will not be able to train the network.
- We will use softmax: $\alpha_{.,l} = \text{softmax}(e_{.,l})$, where $\alpha_{t,l} = \dfrac{\exp(e_{t,l})}{\sum_{t'} \exp(e_{t',l})}$
- Send $a_l = \sum_t \alpha_{t,l} h_t$. $\alpha_{t,l}$s are small numbers except for the max $e_{t,l}$

Source: CS W182 course, Sergey Levine, UC Berkeley

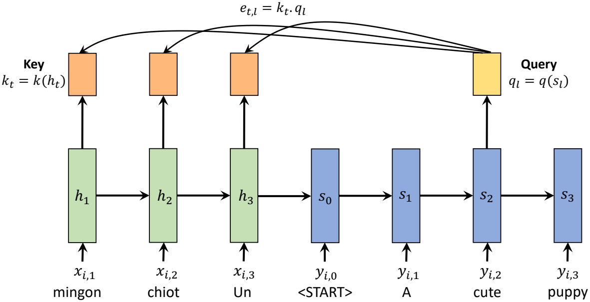
Attention



§ Send $a_l = \sum_t \alpha_{t,l} h_t$. What does 'sending' mean?

Source: CS W182 course, Sergey Levine, UC Berkeley

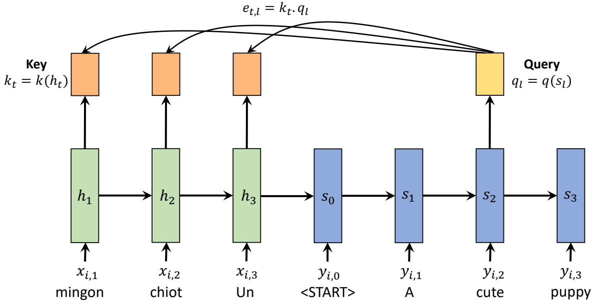
Attention



- Send $a_l = \sum_t \alpha_{t,l} h_t$. What does 'sending' mean?
  - $\hat{y}_l = f(s_l, a_l)$

Source: CS W182 course, Sergey Levine, UC Berkeley

Attention

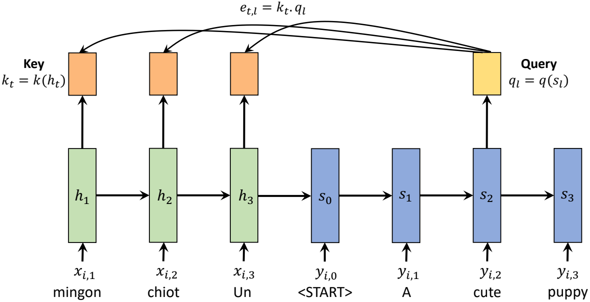


§ Send $a_l = \sum_t \alpha_{t,l} h_t$. What does 'sending' mean?

- $\hat{y}_l = f(s_l, a_l)$
- Give a_l to next RNN layer if stacked RNN is used

Source: CS W182 course, Sergey Levine, UC Berkeley

Attention

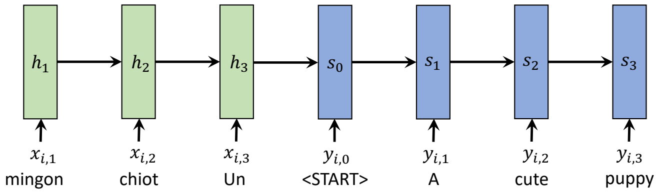


§ Send $a_l = \sum_t \alpha_{t,l} h_t$. What does 'sending' mean?

- $\hat{y}_l = f(s_l, a_l)$
- Give a_l to next RNN layer if stacked RNN is used
- Append a_l to the next decoder step $\bar{s}_l = \begin{bmatrix} s_{l-1} \\ a_{l-1} \\ x_l \end{bmatrix}$

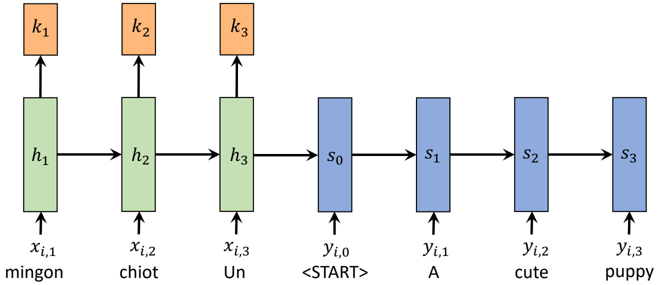
Source: CS W182 course, Sergey Levine, UC Berkeley

Attention Walkthrough (Example)



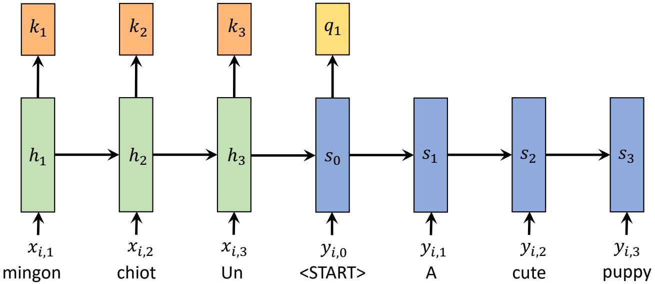
Source: CS W182 course, Sergey Levine, UC Berkeley

Attention Walkthrough (Example)



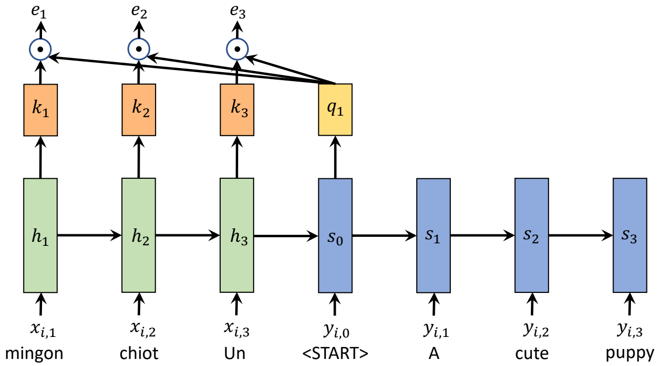
Source: CS W182 course, Sergey Levine, UC Berkeley

Attention Walkthrough (Example)



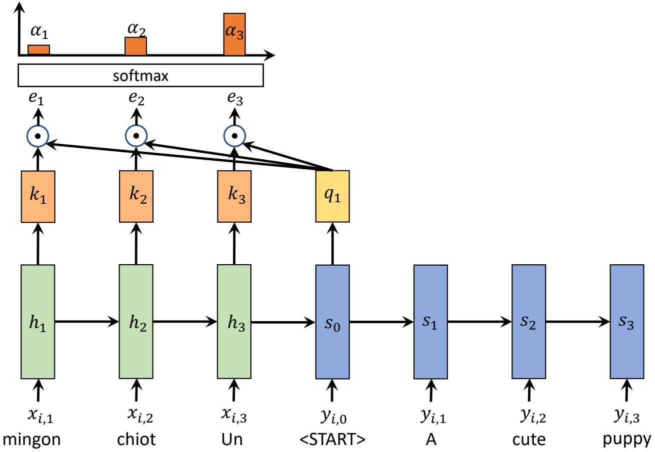
Source: CS W182 course, Sergey Levine, UC Berkeley

Attention Walkthrough (Example)



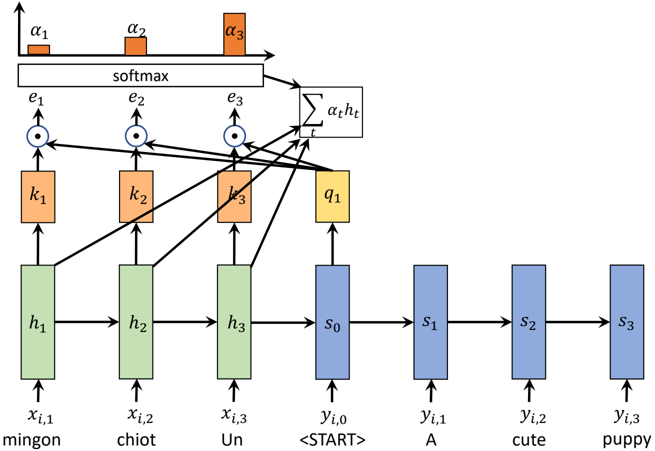
Source: CS W182 course, Sergey Levine, UC Berkeley

Attention Walkthrough (Example)



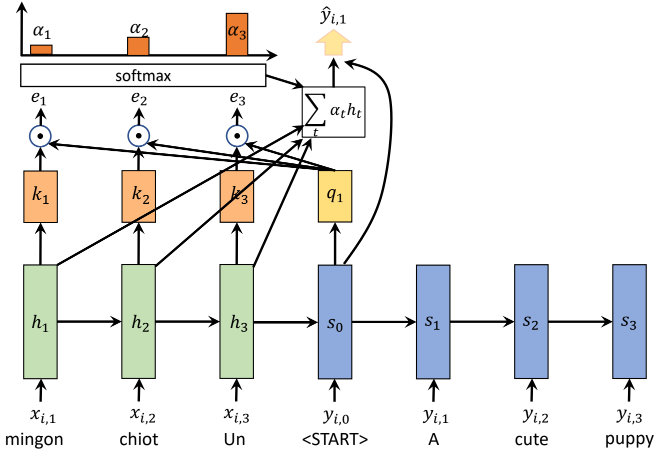
Source: CS W182 course, Sergey Levine, UC Berkeley

Attention Walkthrough (Example)



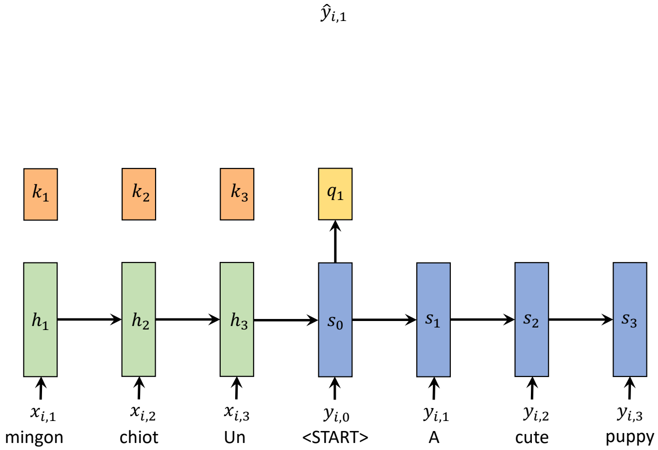
Source: CS W182 course, Sergey Levine, UC Berkeley

Attention Walkthrough (Example)



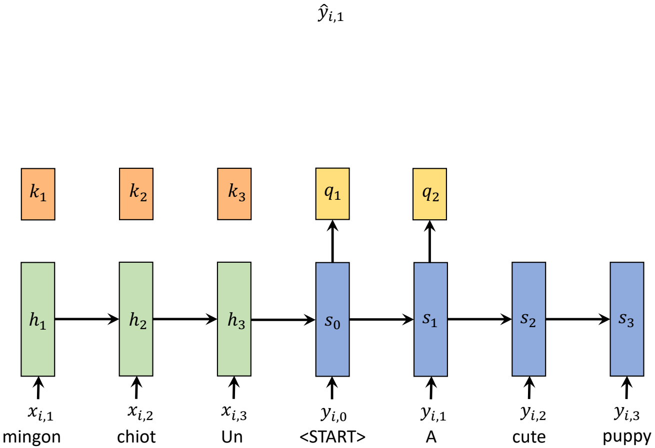
Source: CS W182 course, Sergey Levine, UC Berkeley

Attention Walkthrough (Example)



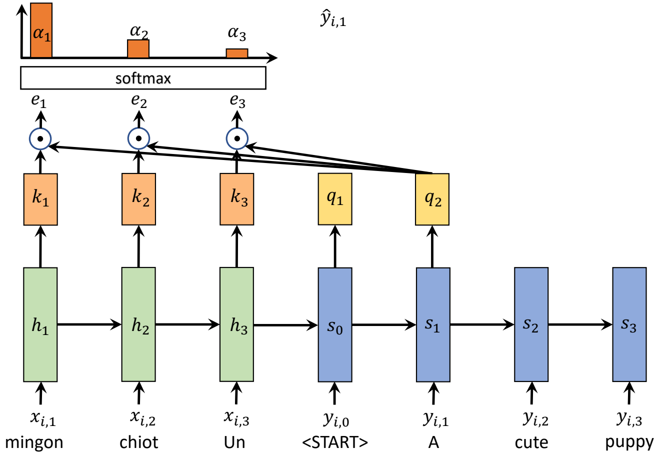
Source: CS W182 course, Sergey Levine, UC Berkeley

Attention Walkthrough (Example)



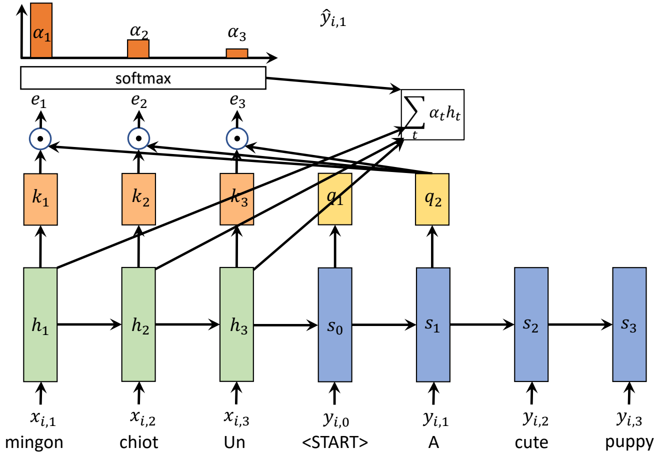
Source: CS W182 course, Sergey Levine, UC Berkeley

Attention Walkthrough (Example)



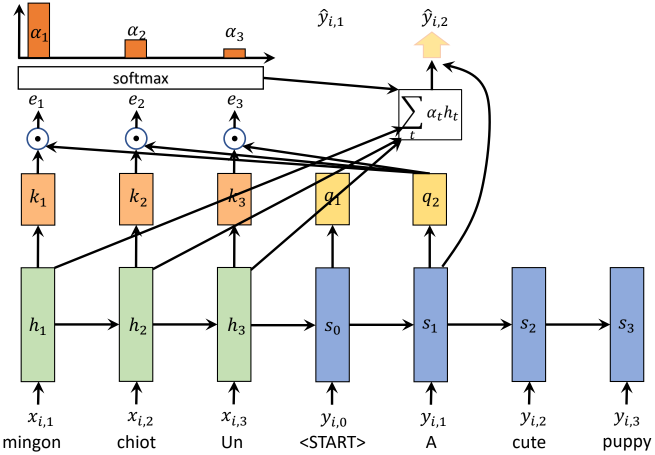
Source: CS W182 course, Sergey Levine, UC Berkeley

Attention Walkthrough (Example)



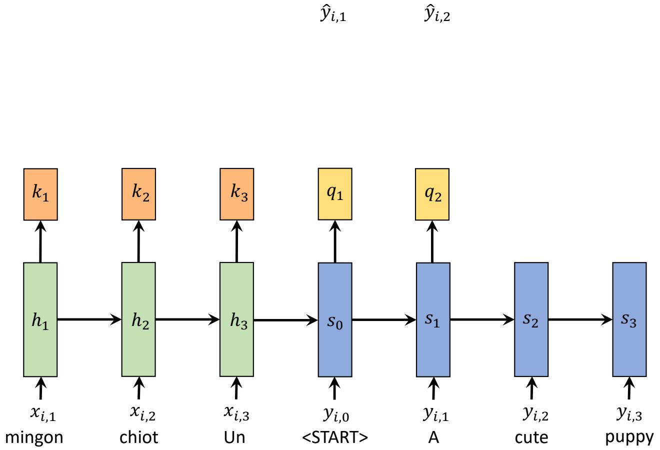
Source: CS W182 course, Sergey Levine, UC Berkeley

Attention Walkthrough (Example)



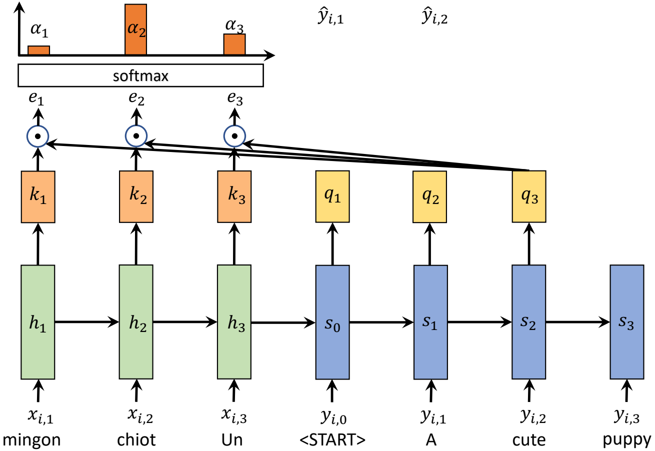
Source: CS W182 course, Sergey Levine, UC Berkeley

Attention Walkthrough (Example)



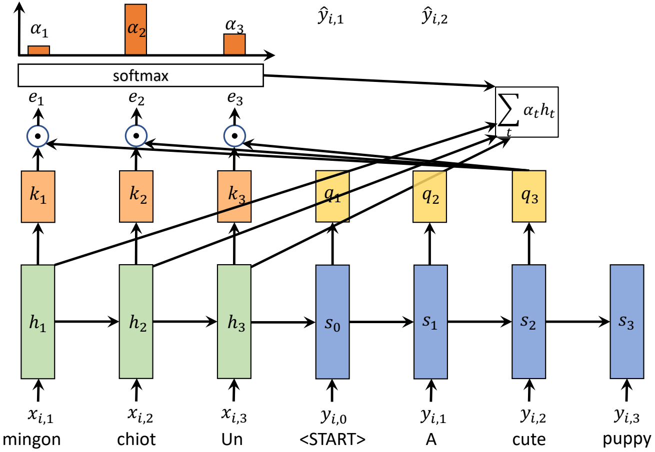
Source: CS W182 course, Sergey Levine, UC Berkeley

Attention Walkthrough (Example)



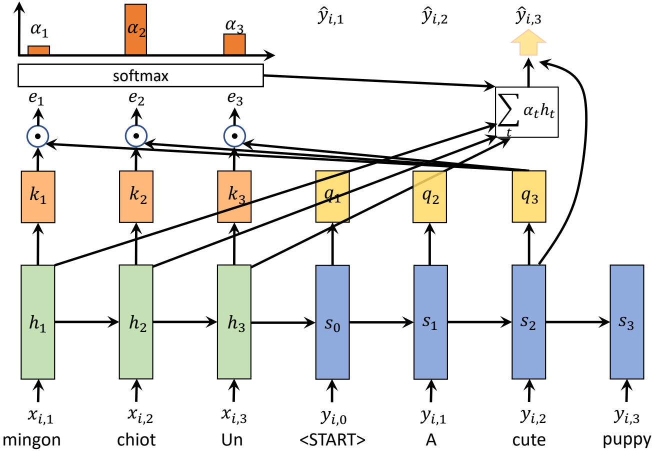
Source: CS W182 course, Sergey Levine, UC Berkeley

Attention Walkthrough (Example)



Source: CS W182 course, Sergey Levine, UC Berkeley

Attention Walkthrough (Example)



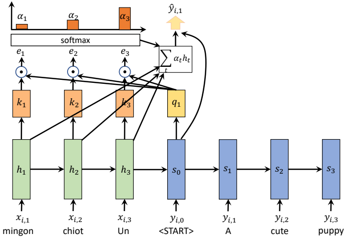
Source: CS W182 course, Sergey Levine, UC Berkeley

Attention Equations

- Encoder side:
  $k_t = k(h_t)$
- Decoder side:
  $q_l = q(s_l)$
- $e_{t,l} = k_t \cdot q_l$

$$\alpha_{t,l} = \frac{\exp(e_{t,l})}{\sum_{t'} \exp(e_{t',l})}$$

$$a_l = \sum_t \alpha_{t,l} h_t$$



Source: CS W182 course, Sergey Levine, UC Berkeley

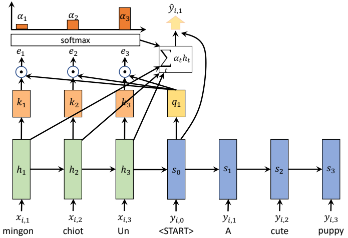
Attention Equations

- Encoder side:
  $k_t = k(h_t)$
- Decoder side:
  $q_l = q(s_l)$
- $e_{t,l} = k_t \cdot q_l$

$$\alpha_{t,l} = \frac{\exp(e_{t,l})}{\sum_{t'} \exp(e_{t',l})}$$

$$a_l = \sum_t \alpha_{t,l} h_t$$

- Can be used in different ways:
- Concatenate to hidden state
  $\begin{bmatrix} s_{l-1} \\ a_{l-1} \\ x_l \end{bmatrix}$
- Use for readout:
  $\hat{y}_l = f(s_l, a_l)$
- Concatenate as input to next RNN layer.



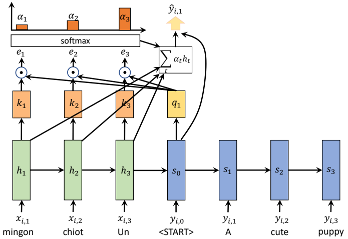
Source: CS W182 course, Sergey Levine, UC Berkeley

Attention Variants

- Simple key-query choice: k and q are identity functions: $k_t = h_t,\ q_l = s_l$
- $e_{t,l} = k_t \cdot q_l$

$$\alpha_{t,l} = \frac{\exp(e_{t,l})}{\sum_{t'} \exp(e_{t',l})}$$

$$a_l = \sum_t \alpha_{t,l} h_t$$



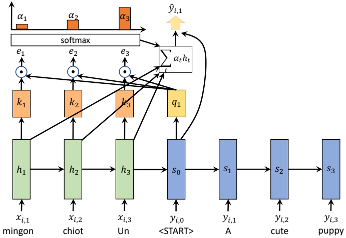
Source: CS W182 course, Sergey Levine, UC Berkeley

Attention Variants

- Linear multiplicative attention:
  $k_t = W_k h_t, \;\; q_l = W_q s_l$
- $e_{t,l} = h_t^T W_k^T W_q s_l = h_t^T W_e s_l$

  $$\alpha_{t,l} = \frac{\exp(e_{t,l})}{\sum_{t'} \exp(e_{t',l})}$$

  $$a_l = \sum_t \alpha_{t,l} h_t$$



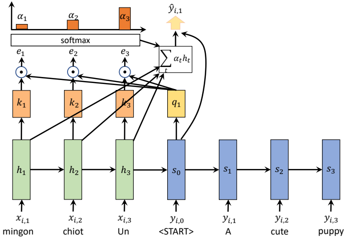
Source: CS W182 course, Sergey Levine, UC Berkeley

Attention Variants

- Learned value encoding
- Encoder side: $k_t = k(h_t)$
- Decoder side: $q_l = q(s_l)$
- $e_{t,l} = k_t \cdot q_l$

  $$\alpha_{t,l} = \frac{\exp(e_{t,l})}{\sum_{t'} \exp(e_{t',l})}$$

  $$a_l = \sum_t \alpha_{t,l} v(h_t)$$

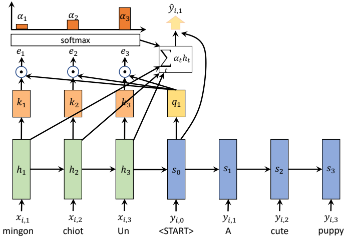


- $v(.)$ is some learned function and known as the 'value'.
- The interpretation is that now you don't just compute 'key', rather you compute a 'key-value' pair of the input hidden states. During decoding, key-query provides the timestep with largest similarity between key and query.
- The attention (ideally) collects the value of that timestep from the input. In 'softmaxed' version, a weighted combination of the input values are taken.

Source: CS W182 course, Sergey Levine, UC Berkeley

Attention Summary

- Every encoder step t produces a key k_t
- Every decoder step l produces a query q_l
- Decoder gets "sent" encoder activation h_t corresponding to the largest value of $k_t \cdot q_l$
- Actually gets $a_l = \sum_t \alpha_{t,l} h_t$

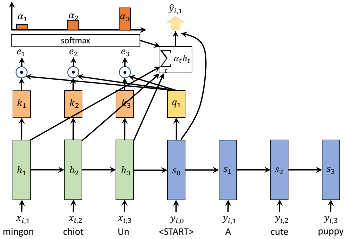


Source: CS W182 course, Sergey Levine, UC Berkeley

Attention Summary

- Every encoder step t produces a key k_t
- Every decoder step l produces a query q_l
- Decoder gets "sent" encoder activation h_t corresponding to the largest value of $k_t \cdot q_l$
- Actually gets $a_l = \sum_t \alpha_{t,l} h_t$

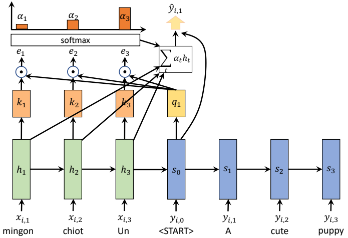
- Why is this good?



Source: CS W182 course, Sergey Levine, UC Berkeley

Attention Summary

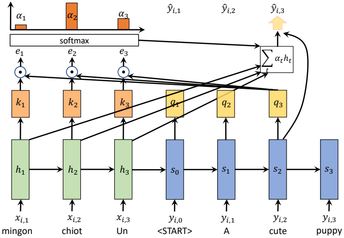
- Every encoder step t produces a key k_t
- Every decoder step l produces a query q_l
- Decoder gets "sent" encoder activation h_t corresponding to the largest value of $k_t \cdot q_l$
- Actually gets $a_l = \sum_t \alpha_{t,l} h_t$



- Why is this good?
- Attention is very powerful, because now **all** decoder steps are connected to **all** encoder steps!
- Bottleneck is much less important
- Gradients are much better behaved

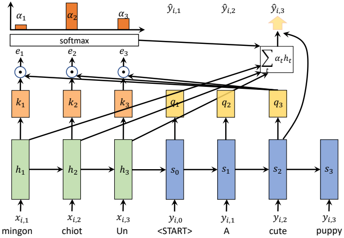
Source: CS W182 course, Sergey Levine, UC Berkeley

Attention



Source: CS W182 course, Sergey Levine, UC Berkeley

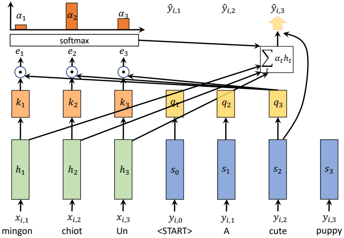
Attention



- If we have **attention**, do we even need recurrent connections?

Source: CS W182 course, Sergey Levine, UC Berkeley

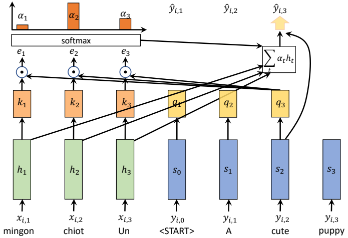
Attention



- If we have **attention**, do we even need recurrent connections?
- Can we transform RNN into a **purely attention-based** model?

Source: CS W182 course, Sergey Levine, UC Berkeley

Attention



- If we have **attention**, do we even need recurrent connections?
- Can we transform RNN into a **purely attention-based** model?
- This has a few issues we must overcome:
  - Now, step $l = 2$ can't acess s_0 or s_2
  - The encoder has no temporal dependences at all.

Source: CS W182 course, Sergey Levine, UC Berkeley

Self-Attention

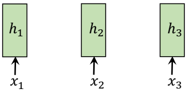
- Basic self attention: without making distinction between encoder and decoder

x_1 x_2 x_3

Source: CS W182 course, Sergey Levine, UC Berkeley

Self-Attention

- Basic self attention: without making distinction between encoder and decoder
- Input from each time-step is encoded
  e.g., $h_t = \sigma(Wx_t + b)$
- This is not a recurrent model, but still weight sharing

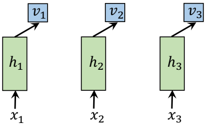


Source: CS W182 course, Sergey Levine, UC Berkeley

Self-Attention

- Basic self attention: without making distinction between encoder and decoder
- Input from each time-step is encoded
  e.g., $h_t = \sigma(Wx_t + b)$
- This is not a recurrent model, but still weight sharing

- Value $v_t = v(h_t)$ for each timestep. Before just had $v(h_t) = h_t$, now, *e.g.*, $v(h_t) = W_v h_t$

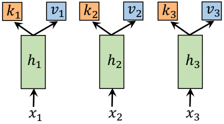


Source: CS W182 course, Sergey Levine, UC Berkeley

Self-Attention

- Basic self attention: without making distinction between encoder and decoder
- Input from each time-step is encoded
  e.g., $h_t = \sigma(Wx_t + b)$
- This is not a recurrent model, but still weight sharing

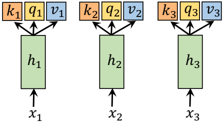
- Value $v_t = v(h_t)$ for each timestep. Before just had $v(h_t) = h_t$, now, *e.g.*, $v(h_t) = W_v h_t$
- Every timestep also outputs key $k_t = k(h_t)$, *e.g.*, $k(h_t) = W_k h_t$



Source: CS W182 course, Sergey Levine, UC Berkeley

Self-Attention

- Basic self attention: without making distinction between encoder and decoder
- Input from each time-step is encoded
  e.g., $h_t = \sigma(Wx_t + b)$
- This is not a recurrent model, but still weight sharing

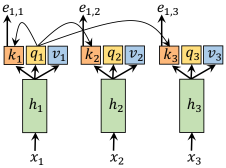


- Value $v_t = v(h_t)$ for each timestep. Before just had $v(h_t) = h_t$, now, *e.g.*, $v(h_t) = W_v h_t$
- Every timestep also outputs key $k_t = k(h_t)$, *e.g.*, $k(h_t) = W_k h_t$
- Every timestep also outputs query $q_t = q(h_t)$, *e.g.*, $q(h_t) = W_q h_t$

Source: CS W182 course, Sergey Levine, UC Berkeley

Self-Attention

- Basic self attention: without making distinction between encoder and decoder
- Input from each time-step is encoded
  e.g., $h_t = \sigma(Wx_t + b)$
- This is not a recurrent model, but still weight sharing

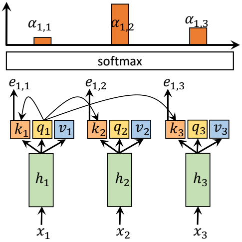


- Value $v_t = v(h_t)$ for each timestep. Before just had $v(h_t) = h_t$, now, *e.g.*, $v(h_t) = W_v h_t$
- Every timestep also outputs key $k_t = k(h_t)$, *e.g.*, $k(h_t) = W_k h_t$
- Every timestep also outputs query $q_t = q(h_t)$, *e.g.*, $q(h_t) = W_q h_t$
- Get similarity between every key and every query including both coming from the same timestep.
  $e_{l,t} = q_l \cdot k_t$

Source: CS W182 course, Sergey Levine, UC Berkeley

Self-Attention

- Basic self attention: without making distinction between encoder and decoder
- Input from each time-step is encoded
  e.g., $h_t = \sigma(Wx_t + b)$
- This is not a recurrent model, but still weight sharing

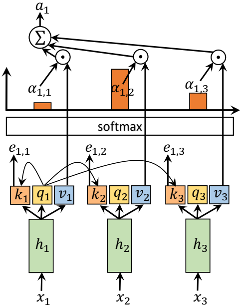


- Value $v_t = v(h_t)$ for each timestep. Before just had $v(h_t) = h_t$, now, *e.g.*, $v(h_t) = W_v h_t$
- Every timestep also outputs key $k_t = k(h_t)$, *e.g.*, $k(h_t) = W_k h_t$
- Every timestep also outputs query $q_t = q(h_t)$, *e.g.*, $q(h_t) = W_q h_t$
- Get similarity between every key and every query including both coming from the same timestep.
  $e_{l,t} = q_l \cdot k_t$
- Compute attention scores: $\alpha_{l,t} = \dfrac{\exp(e_{l,t})}{\sum_{t'} \exp(e_{l,t'})}$

Source: CS W182 course, Sergey Levine, UC Berkeley

Self-Attention

- Basic self attention: without making distinction between encoder and decoder
- Input from each time-step is encoded
  e.g., $h_t = \sigma(Wx_t + b)$
- This is not a recurrent model, but still weight sharing

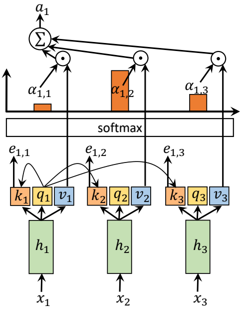


- Value $v_t = v(h_t)$ for each timestep. Before just had $v(h_t) = h_t$, now, *e.g.*, $v(h_t) = W_v h_t$
- Every timestep also outputs key $k_t = k(h_t)$, *e.g.*, $k(h_t) = W_k h_t$
- Every timestep also outputs query $q_t = q(h_t)$, *e.g.*, $q(h_t) = W_q h_t$
- Get similarity between every key and every query including both coming from the same timestep.
  $e_{l,t} = q_l \cdot k_t$
- Compute attention scores: $\alpha_{l,t} = \dfrac{\exp(e_{l,t})}{\sum_{t'} \exp(e_{l,t'})}$
- Compute attention at timestep l: $a_l = \sum_t \alpha_{l,t} v_t$

Source: CS W182 course, Sergey Levine, UC Berkeley

Self-Attention

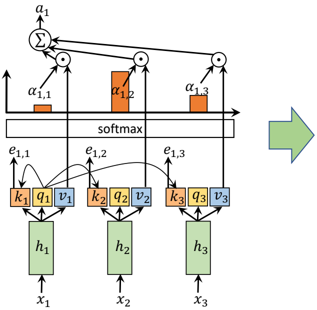
- At every timestep, self attention takes input and produces an output
- This can be regarded as a layer



Source: CS W182 course, Sergey Levine, UC Berkeley

Self-Attention

- At every timestep, self attention takes input and produces an output
- This can be regarded as a layer

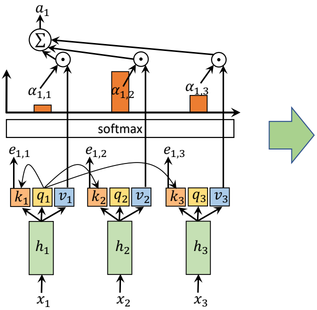


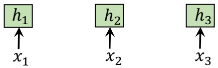
- We can build an entire network by stacking such layers

Source: CS W182 course, Sergey Levine, UC Berkeley

Self-Attention

- At every timestep, self attention takes input and produces an output
- This can be regarded as a layer



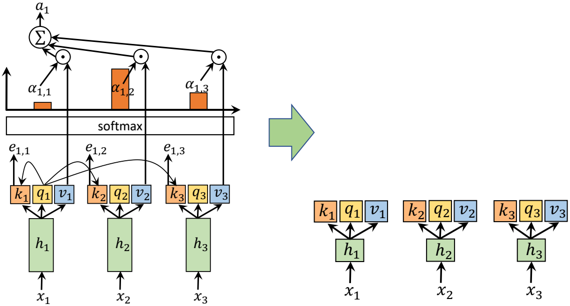


- We can build an entire network by stacking such layers

Source: CS W182 course, Sergey Levine, UC Berkeley

Self-Attention

- At every timestep, self attention takes input and produces an output
- This can be regarded as a layer



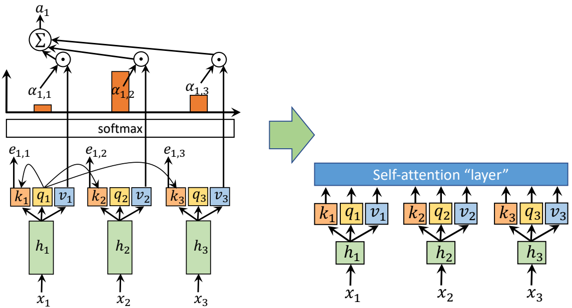
- We can build an entire network by stacking such layers

Source: CS W182 course, Sergey Levine, UC Berkeley

Self-Attention

§ At every timestep, self attention takes input and produces an output

§ This can be regarded as a layer

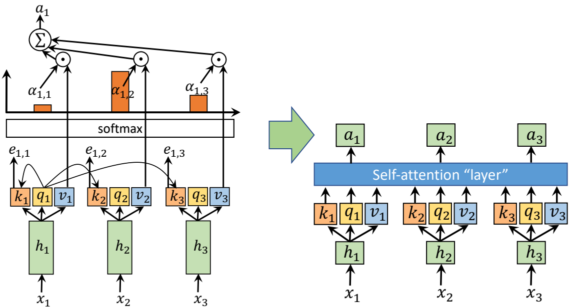


§ We can build an entire network by stacking such layers

Source: CS W182 course, Sergey Levine, UC Berkeley

Self-Attention

- At every timestep, self attention takes input and produces an output
- This can be regarded as a layer

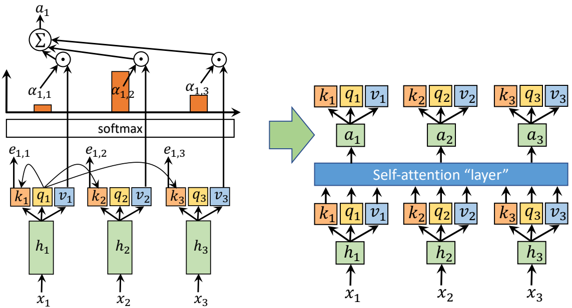


- We can build an entire network by stacking such layers

Source: CS W182 course, Sergey Levine, UC Berkeley

Self-Attention

- At every timestep, self attention takes input and produces an output
- This can be regarded as a layer

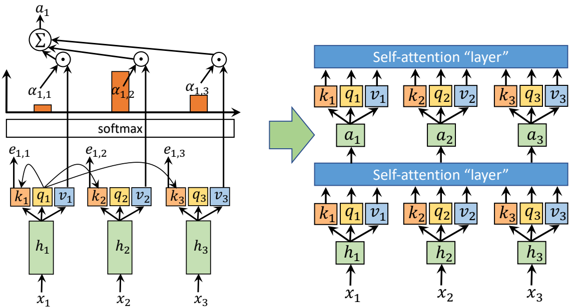


- We can build an entire network by stacking such layers

Source: CS W182 course, Sergey Levine, UC Berkeley

Self-Attention

- At every timestep, self attention takes input and produces an output
- This can be regarded as a layer

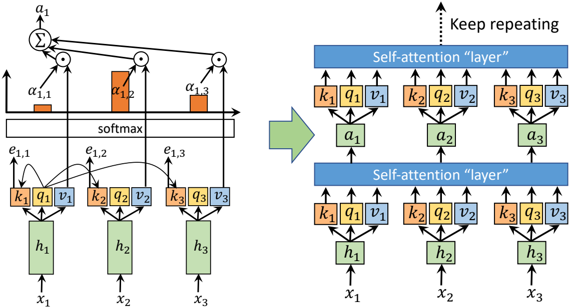


- We can build an entire network by stacking such layers

Source: CS W182 course, Sergey Levine, UC Berkeley

Self-Attention

- At every timestep, self attention takes input and produces an output
- This can be regarded as a layer



- We can build an entire network by stacking such layers
- This basic idea of getting another sequence from an input sequence is used to get another class of sequence-to-sequence models known as 'Transformers'

Source: CS W182 course, Sergey Levine, UC Berkeley

From Self-Attention to Transformers

§ But to make this actually work, we need to develop a few additional components to address some fundamental limitations

Source: CS W182 course, Sergey Levine, UC Berkeley

From Self-Attention to Transformers

- But to make this actually work, we need to develop a few additional components to address some fundamental limitations
- Positional encoding
  - Addresses lack of sequence information

Source: CS W182 course, Sergey Levine, UC Berkeley

From Self-Attention to Transformers

- But to make this actually work, we need to develop a few additional components to address some fundamental limitations
- Positional encoding
  - Addresses lack of sequence information
- Multiheaded attention
  - allows querying multiple positions at each layer

Source: CS W182 course, Sergey Levine, UC Berkeley

From Self-Attention to Transformers

- But to make this actually work, we need to develop a few additional components to address some fundamental limitations
- Positional encoding
  - Addresses lack of sequence information
- Multiheaded attention
  - allows querying multiple positions at each layer
- Adding nonlinearities
  - So far, each successive layer is *linear* in the previous one
    $a_l = \sum_t \alpha_{l,t} v_t$ where, $v_t = W_v h_t$

Source: CS W182 course, Sergey Levine, UC Berkeley

From Self-Attention to Transformers

- But to make this actually work, we need to develop a few additional components to address some fundamental limitations
- Positional encoding
  - Addresses lack of sequence information
- Multiheaded attention
  - allows querying multiple positions at each layer
- Adding nonlinearities
  - So far, each successive layer is *linear* in the previous one
    $a_l = \sum_t \alpha_{l,t} v_t$ where, $v_t = W_v h_t$
- Masked decoding
  - How to prevent attention lookups into the future?

Source: CS W182 course, Sergey Levine, UC Berkeley

Positional Encoding: What is the Order

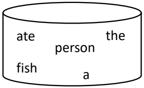
§ What we see:

- the person ate a fish.

Source: CS W182 course, Sergey Levine, UC Berkeley

Positional Encoding: What is the Order

- What we see:
  - the person ate a fish.
- What naive self-attention sees:



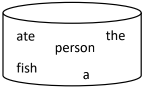
Source: CS W182 course, Sergey Levine, UC Berkeley

Positional Encoding: What is the Order

§ What we see:

- the person ate a fish.

§ What naive self-attention sees:



§ Most alternative orderings are nonsense, but some change meaning

- the fish ate a person
- the ate person a fish
- person fish the a ate

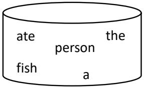
Source: CS W182 course, Sergey Levine, UC Berkeley

Positional Encoding: What is the Order

§ What we see:

- the person ate a fish.

§ What naive self-attention sees:



§ Most alternative orderings are nonsense, but some change meaning

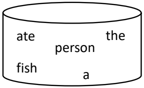
- the fish ate a person
- the ate person a fish
- person fish the a ate

§ In general: the position of words in a sentence carries information!

Source: CS W182 course, Sergey Levine, UC Berkeley

Positional Encoding: What is the Order

- What we see:
  - the person ate a fish.
- What naive self-attention sees:

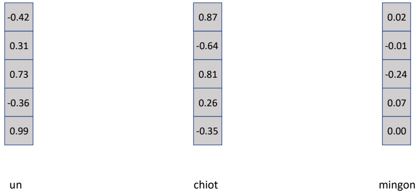


- Most alternative orderings are nonsense, but some change meaning
  - the fish ate a person
  - the ate person a fish
  - person fish the a ate
- In general: the position of words in a sentence carries information!
- Idea: add some information at the beginning that indicates where it is in the sequence!
  - $h_t = f(x_t, t)$

Source: CS W182 course, Sergey Levine, UC Berkeley

Positional Encoding

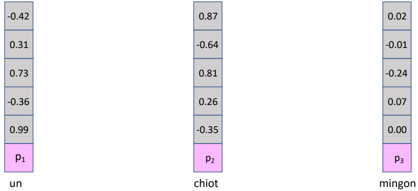
- There are two main ways to provide the model with this information.
  - Concatenating position embedding with word embedding



Positional Encoding

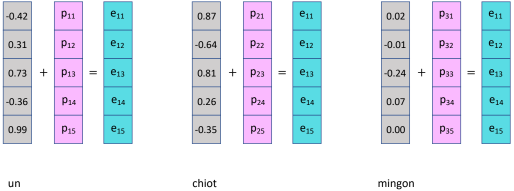
§ There are two main ways to provide the model with this information.

- Concatenating position embedding with word embedding



Positional Encoding

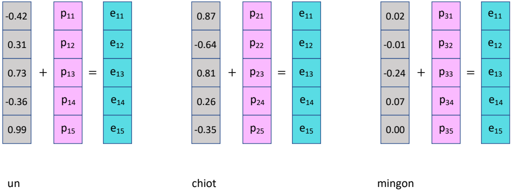
§ There are two main ways to provide the model with this information.
- ▶ Concatenating position embedding with word embedding
- ▶ Adding position embedding with word embedding



Positional Encoding

§ There are two main ways to provide the model with this information.

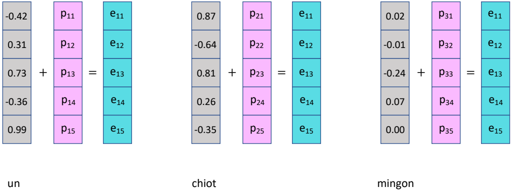
- ▶ Concatenating position embedding with word embedding
- ▶ Adding position embedding with word embedding



§ There isn't a clear winner, but both has advantages or disadvantages.

Positional Encoding

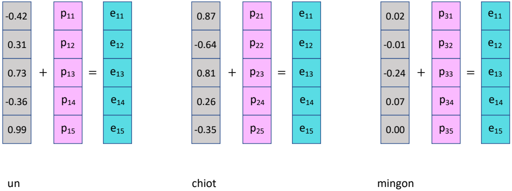
- There are two main ways to provide the model with this information.
  - Concatenating position embedding with word embedding
  - Adding position embedding with word embedding



- There isn't a clear winner, but both has advantages or disadvantages.
- Just concatenating word position is simple and avoids messing up the semantic relationship between different words.

Positional Encoding

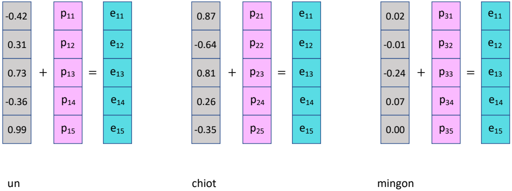
- There are two main ways to provide the model with this information.
  - Concatenating position embedding with word embedding
  - Adding position embedding with word embedding



- There isn't a clear winner, but both has advantages or disadvantages.
- Just concatenating word position is simple and avoids messing up the semantic relationship between different words.
- But its comes with additional memory and parameters.
- Down-the-line similarity computation between key and query may contain extra unrelevant terms

Positional Encoding

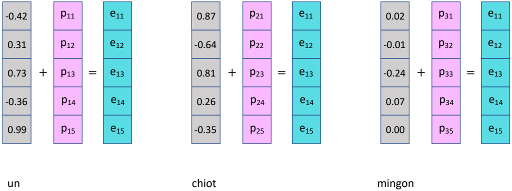
- There are two main ways to provide the model with this information.
  - Concatenating position embedding with word embedding
  - Adding position embedding with word embedding



- There isn't a clear winner, but both has advantages or disadvantages.
- Just concatenating word position is simple and avoids messing up the semantic relationship between different words.
- But its comes with additional memory and parameters.
- Down-the-line similarity computation between key and query may contain extra unrelevant terms
- Addition is another option and tends to work well

Positional Encoding

- There are two main ways to provide the model with this information.
  - Concatenating position embedding with word embedding
  - Adding position embedding with word embedding



- There isn't a clear winner, but both has advantages or disadvantages.
- Just concatenating word position is simple and avoids messing up the semantic relationship between different words.
- But its comes with additional memory and parameters.
- Down-the-line similarity computation between key and query may contain extra unrelevant terms
- Addition is another option and tends to work well
- More information: Link1 Link1

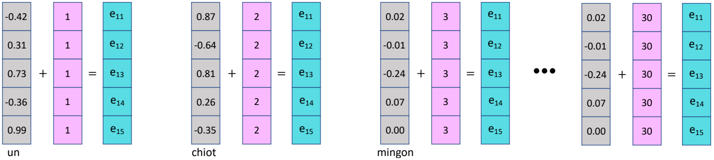
Positional Encoding - Addition

§ Adding the word positions as all dimensions of position embedding

$$\underbrace{\begin{bmatrix}-0.42\\0.31\\0.73\\-0.36\\0.99\end{bmatrix}}_{\text{un}} + \begin{bmatrix}1\\1\\1\\1\\1\end{bmatrix} = \begin{bmatrix}e_{11}\\e_{12}\\e_{13}\\e_{14}\\e_{15}\end{bmatrix} \qquad \underbrace{\begin{bmatrix}0.87\\-0.64\\0.81\\0.26\\-0.35\end{bmatrix}}_{\text{chiot}} + \begin{bmatrix}2\\2\\2\\2\\2\end{bmatrix} = \begin{bmatrix}e_{11}\\e_{12}\\e_{13}\\e_{14}\\e_{15}\end{bmatrix} \qquad \underbrace{\begin{bmatrix}0.02\\-0.01\\-0.24\\0.07\\0.00\end{bmatrix}}_{\text{mingon}} + \begin{bmatrix}3\\3\\3\\3\\3\end{bmatrix} = \begin{bmatrix}e_{11}\\e_{12}\\e_{13}\\e_{14}\\e_{15}\end{bmatrix}$$

Positional Encoding - Addition

- Adding the word positions as all dimensions of position embedding



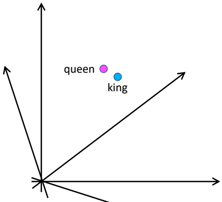
- However, adding word positions can significantly distort semantic positions of the words, especially for words in long sentences.

Positional Encoding - Addition

§ Adding the word positions as all dimensions of position embedding

$$\underset{\text{un}}{\begin{bmatrix}-0.42\\0.31\\0.73\\-0.36\\0.99\end{bmatrix}} + \begin{bmatrix}1\\1\\1\\1\\1\end{bmatrix} = \begin{bmatrix}e_{11}\\e_{12}\\e_{13}\\e_{14}\\e_{15}\end{bmatrix} \quad \underset{\text{chiot}}{\begin{bmatrix}0.87\\-0.64\\0.81\\0.26\\-0.35\end{bmatrix}} + \begin{bmatrix}2\\2\\2\\2\\2\end{bmatrix} = \begin{bmatrix}e_{11}\\e_{12}\\e_{13}\\e_{14}\\e_{15}\end{bmatrix} \quad \underset{\text{mingon}}{\begin{bmatrix}0.02\\-0.01\\-0.24\\0.07\\0.00\end{bmatrix}} + \begin{bmatrix}3\\3\\3\\3\\3\end{bmatrix} = \begin{bmatrix}e_{11}\\e_{12}\\e_{13}\\e_{14}\\e_{15}\end{bmatrix} \quad \cdots \quad \begin{bmatrix}0.02\\-0.01\\-0.24\\0.07\\0.00\end{bmatrix} + \begin{bmatrix}30\\30\\30\\30\\30\end{bmatrix} = \begin{bmatrix}e_{11}\\e_{12}\\e_{13}\\e_{14}\\e_{15}\end{bmatrix}$$

§ However, adding word positions can significantly distort semantic positions of the words, especially for words in long sentences.

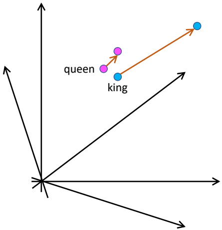


Positional Encoding - Addition

§ Adding the word positions as all dimensions of position embedding

$$\underset{\text{un}}{\begin{bmatrix}-0.42\\0.31\\0.73\\-0.36\\0.99\end{bmatrix}} + \begin{bmatrix}1\\1\\1\\1\\1\end{bmatrix} = \begin{bmatrix}e_{11}\\e_{12}\\e_{13}\\e_{14}\\e_{15}\end{bmatrix} \quad \underset{\text{chiot}}{\begin{bmatrix}0.87\\-0.64\\0.81\\0.26\\-0.35\end{bmatrix}} + \begin{bmatrix}2\\2\\2\\2\\2\end{bmatrix} = \begin{bmatrix}e_{11}\\e_{12}\\e_{13}\\e_{14}\\e_{15}\end{bmatrix} \quad \underset{\text{mingon}}{\begin{bmatrix}0.02\\-0.01\\-0.24\\0.07\\0.00\end{bmatrix}} + \begin{bmatrix}3\\3\\3\\3\\3\end{bmatrix} = \begin{bmatrix}e_{11}\\e_{12}\\e_{13}\\e_{14}\\e_{15}\end{bmatrix} \quad \cdots \quad \begin{bmatrix}0.02\\-0.01\\-0.24\\0.07\\0.00\end{bmatrix} + \begin{bmatrix}30\\30\\30\\30\\30\end{bmatrix} = \begin{bmatrix}e_{11}\\e_{12}\\e_{13}\\e_{14}\\e_{15}\end{bmatrix}$$

§ However, adding word positions can significantly distort semantic positions of the words, especially for words in long sentences.



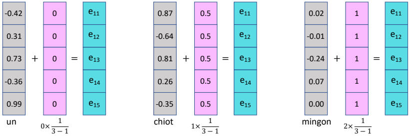
Positional Encoding - Addition

- How about adding fractions only. The added values will never surpass 1

$$\underbrace{\begin{bmatrix}-0.42\\0.31\\0.73\\-0.36\\0.99\end{bmatrix}}_{\text{un}} + \underbrace{\begin{bmatrix}0\\0\\0\\0\\0\end{bmatrix}}_{0\times\frac{1}{3-1}} = \begin{bmatrix}e_{11}\\e_{12}\\e_{13}\\e_{14}\\e_{15}\end{bmatrix} \qquad \underbrace{\begin{bmatrix}0.87\\-0.64\\0.81\\0.26\\-0.35\end{bmatrix}}_{\text{chiot}} + \underbrace{\begin{bmatrix}0.5\\0.5\\0.5\\0.5\\0.5\end{bmatrix}}_{1\times\frac{1}{3-1}} = \begin{bmatrix}e_{11}\\e_{12}\\e_{13}\\e_{14}\\e_{15}\end{bmatrix} \qquad \underbrace{\begin{bmatrix}0.02\\-0.01\\-0.24\\0.07\\0.00\end{bmatrix}}_{\text{mingon}} + \underbrace{\begin{bmatrix}1\\1\\1\\1\\1\end{bmatrix}}_{2\times\frac{1}{3-1}} = \begin{bmatrix}e_{11}\\e_{12}\\e_{13}\\e_{14}\\e_{15}\end{bmatrix}$$

Positional Encoding - Addition

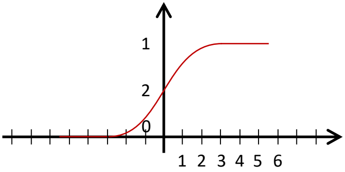
- How about adding fractions only. The added values will never surpass 1



- Not great, as the same position will have different encoding value

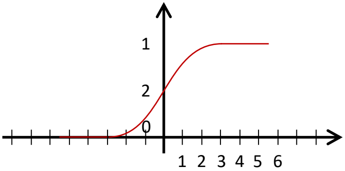
Positional Encoding - Addition

- How about adding fractions only. The added values will never surpass 1
- Not great, as the same position will have different encoding value
- Then what about using a bounded function that extends till infinity?



Positional Encoding - Addition

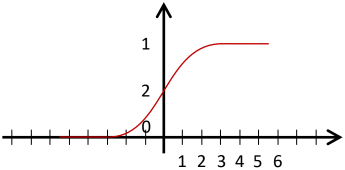
- How about adding fractions only. The added values will never surpass 1
- Not great, as the same position will have different encoding value
- Then what about using a bounded function that extends till infinity?



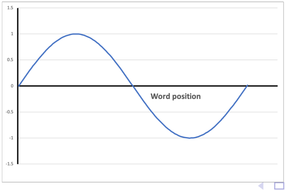
- The problem is that the encoding very quickly saturates and for all higher position values, they are same

Positional Encoding - Addition

- How about adding fractions only. The added values will never surpass 1
- Not great, as the same position will have different encoding value
- Then what about using a bounded function that extends till infinity?

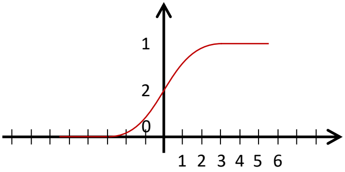


- The problem is that the encoding very quickly saturates and for all higher position values, they are same
- Then what about a bounded periodic function extending till infinity?



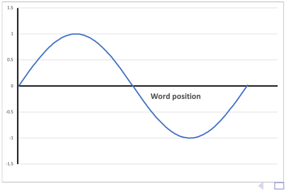
Positional Encoding - Addition

- How about adding fractions only. The added values will never surpass 1
- Not great, as the same position will have different encoding value
- Then what about using a bounded function that extends till infinity?



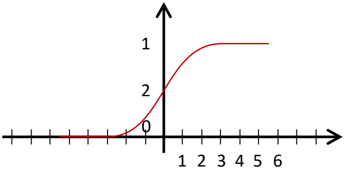
- The problem is that the encoding very quickly saturates and for all higher position values, they are same
- Then what about a bounded periodic function extending till infinity?

What is a basic problem?



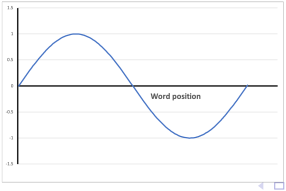
Positional Encoding - Addition

- How about adding fractions only. The added values will never surpass 1
- Not great, as the same position will have different encoding value
- Then what about using a bounded function that extends till infinity?



- The problem is that the encoding very quickly saturates and for all higher position values, they are same
- Then what about a bounded periodic function extending till infinity?

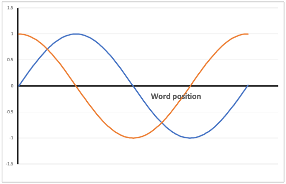
What is a basic problem?



Different positions may have same encoding

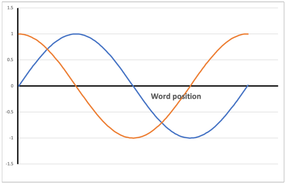
Positional Encoding - Multiple Sinusoids

- Fix: use a cosine also (with same frequency)



Positional Encoding - Multiple Sinusoids

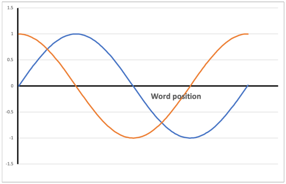
- Fix: use a cosine also (with same frequency)



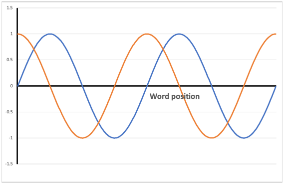
- If two components of the encoding comes from two sinusoids, the problem is largely avoided.

Positional Encoding - Multiple Sinusoids

- Fix: use a cosine also (with same frequency)

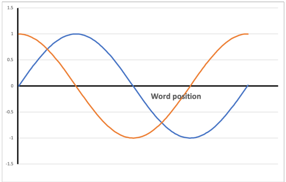


- If two components of the encoding comes from two sinusoids, the problem is largely avoided.
- But not fully. Because the same pattern repeats.

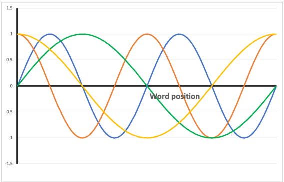


Positional Encoding - Multiple Sinusoids

- Fix: use a cosine also (with same frequency)

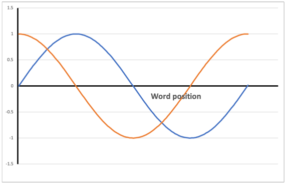


- If two components of the encoding comes from two sinusoids, the problem is largely avoided.
- But not fully. Because the same pattern repeats.
- Add more sinusoids with different frequencies.



Positional Encoding - Multiple Sinusoids

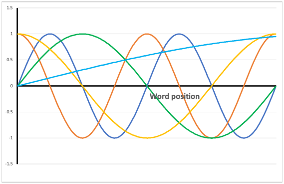
§ Fix: use a cosine also (with same frequency)



§ If two components of the encoding comes from two sinusoids, the problem is largely avoided.

§ But not fully. Because the same pattern repeats.

§ Add more sinusoids with different frequencies.



Positional Encoding - sine-cosine Embedding

$$\mathbf{p_t} = \begin{bmatrix} p_t^{(1)} \\ p_t^{(2)} \\ p_t^{(3)} \\ p_t^{(4)} \\ \cdots \\ p_t^{(d-1)} \\ p_t^{(d)} \end{bmatrix} = \begin{bmatrix} \sin(\omega_1 t) \\ \cos(\omega_1 t) \\ \sin(\omega_2 t) \\ \cos(\omega_2 t) \\ \cdots \\ \sin(\omega_{d/2} t) \\ \cos(\omega_{d/2} t) \end{bmatrix}$$

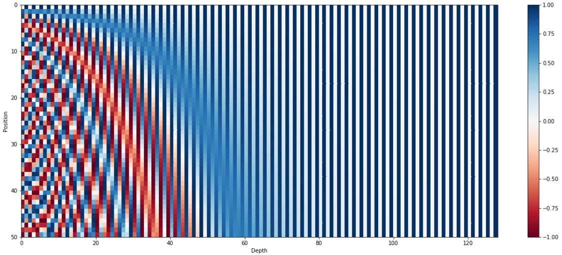
$$p_t^{(i)} = \begin{cases} \sin(\omega_k t), & \text{if } i = 2k-1 \\ \cos(\omega_k t), & \text{if } i = 2k \end{cases}$$

$$\omega_k = \frac{1}{10000^{2k/d}}$$

k varies from 0 to $\dfrac{d}{2}$

Positional Encoding - sine-cosine Embedding

$$\mathbf{p_t} = \begin{bmatrix} p_t^{(1)} \\ p_t^{(2)} \\ p_t^{(3)} \\ p_t^{(4)} \\ \dots \\ p_t^{(d-1)} \\ p_t^{(d)} \end{bmatrix} = \begin{bmatrix} \sin(\omega_1 t) \\ \cos(\omega_1 t) \\ \sin(\omega_2 t) \\ \cos(\omega_2 t) \\ \dots \\ \sin(\omega_{d/2} t) \\ \cos(\omega_{d/2} t) \end{bmatrix}$$



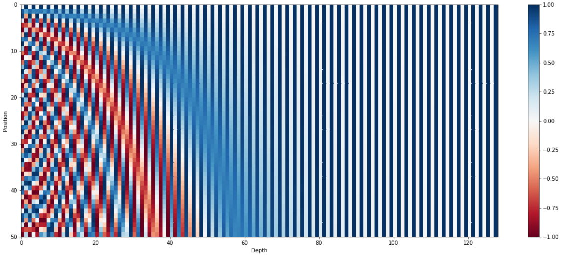
$$p_t^{(i)} = \begin{cases} \sin(\omega_k t), & \text{if } i = 2k-1 \\ \cos(\omega_k t), & \text{if } i = 2k \end{cases}$$

$$\omega_k = \frac{1}{10000^{2k/d}}$$

k varies from 0 to $\dfrac{d}{2}$

Positional Encoding - sine-cosine Embedding

$$\mathbf{p_t} = \begin{bmatrix} p_t^{(1)} \\ p_t^{(2)} \\ p_t^{(3)} \\ p_t^{(4)} \\ \dots \\ p_t^{(d-1)} \\ p_t^{(d)} \end{bmatrix} = \begin{bmatrix} \sin(\omega_1 t) \\ \cos(\omega_1 t) \\ \sin(\omega_2 t) \\ \cos(\omega_2 t) \\ \dots \\ \sin(\omega_{d/2} t) \\ \cos(\omega_{d/2} t) \end{bmatrix}$$



$$p_t^{(i)} = \begin{cases} \sin(\omega_k t), & \text{if } i = 2k-1 \\ \cos(\omega_k t), & \text{if } i = 2k \end{cases}$$

$$\omega_k = \frac{1}{10000^{2k/d}}$$

k varies from 0 to $\dfrac{d}{2}$

§ Learnable positional encoding is sometimes also used

From Self-Attention to Transformers

- But to make this actually work, we need to develop a few additional components to address some fundamental limitations
- Positional encoding
  - Addresses lack of sequence information
- Multiheaded attention
  - allows querying multiple positions at each layer
- Adding nonlinearities
  - So far, each successive layer is *linear* in the previous one $a_l = \sum_t \alpha_{l,t} v_t$ where, $v_t = W_v h_t$
- Masked decoding
  - How to prevent attention lookups into the future?

Source: CS W182 course, Sergey Levine, UC Berkeley

Multi-Head Attention

§ Since we are relying entirely on attention now, we might want to 'query to' different timestep.

Multi-Head Attention

- Since we are relying entirely on attention now, we might want to 'query to' different timestep.
- A sentence like "The animal didn't cross the street because it was too tired', we would want to know
  - If "animal" refers to "it"
  - If it is the "animal" who didn't "cross"

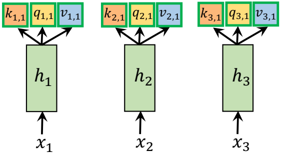
Multi-Head Attention

- Idea: have multiple keys, queries, and values for every time step!

Source: CS W182 course, Sergey Levine, UC Berkeley

Multi-Head Attention

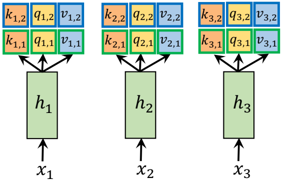
§ Idea: have multiple keys, queries, and values for every time step!



Source: CS W182 course, Sergey Levine, UC Berkeley

Multi-Head Attention

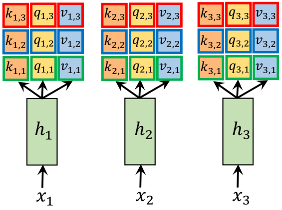
§ Idea: have multiple keys, queries, and values for every time step!



Source: CS W182 course, Sergey Levine, UC Berkeley

Multi-Head Attention

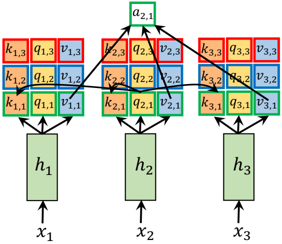
- Idea: have multiple keys, queries, and values for every time step!



Source: CS W182 course, Sergey Levine, UC Berkeley

Multi-Head Attention

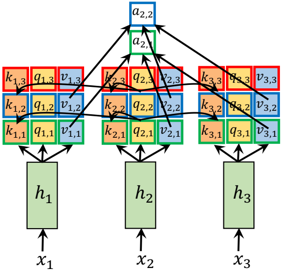
§ Idea: have multiple keys, queries, and values for every time step!



Source: CS W182 course, Sergey Levine, UC Berkeley

Multi-Head Attention

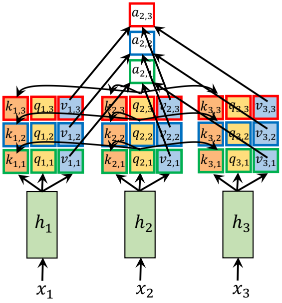
§ Idea: have multiple keys, queries, and values for every time step!



Source: CS W182 course, Sergey Levine, UC Berkeley

Multi-Head Attention

- Idea: have multiple keys, queries, and values for every time step!

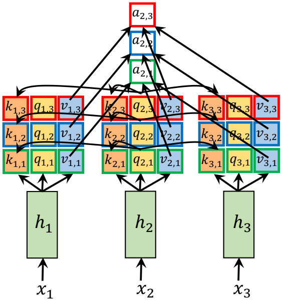


- Compute weights *independently* for each head

Source: CS W182 course, Sergey Levine, UC Berkeley

Multi-Head Attention

- Idea: have multiple keys, queries, and values for every time step!

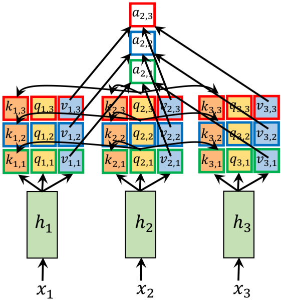


- Compute weights *independently* for each head
- $e_{l,t,i} = q_{l,i} \cdot k_{t,i}$

Source: CS W182 course, Sergey Levine, UC Berkeley

Multi-Head Attention

- Idea: have multiple keys, queries, and values for every time step!

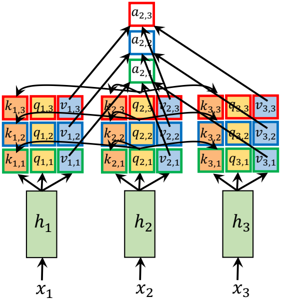


- Compute weights *independently* for each head
- $e_{l,t,i} = q_{l,i} \cdot k_{t,i}$
- $\alpha_{l,t,i} = \dfrac{\exp(e_{l,t,i})}{\sum_{t'} \exp(e_{l,t',i})}$
- $a_{l,i} = \sum_t \alpha_{l,t,i} v_{t,i}$

Source: CS W182 course, Sergey Levine, UC Berkeley

Multi-Head Attention

- Idea: have multiple keys, queries, and values for every time step!



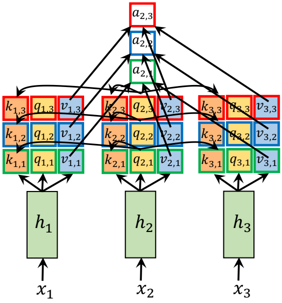
- Compute weights *independently* for each head
- $e_{l,t,i} = q_{l,i} \cdot k_{t,i}$
- $\alpha_{l,t,i} = \dfrac{\exp(e_{l,t,i})}{\sum_{t'} \exp(e_{l,t',i})}$
- $a_{l,i} = \sum_t \alpha_{l,t,i} v_{t,i}$
- Full attention vector is formed by concatenation

$$a_2 = \begin{bmatrix} a_{2,1} \\ a_{2,2} \\ a_{2,3} \end{bmatrix}$$

Source: CS W182 course, Sergey Levine, UC Berkeley

Multi-Head Attention

- Idea: have multiple keys, queries, and values for every time step!



- Compute weights *independently* for each head
- $e_{l,t,i} = q_{l,i} \cdot k_{t,i}$
- $\alpha_{l,t,i} = \dfrac{\exp(e_{l,t,i})}{\sum_{t'} \exp(e_{l,t',i})}$
- $a_{l,i} = \sum_t \alpha_{l,t,i} v_{t,i}$
- Full attention vector is formed by concatenation

$$a_2 = \begin{bmatrix} a_{2,1} \\ a_{2,2} \\ a_{2,3} \end{bmatrix}$$

- Around 8 heads per layer tend to work well.

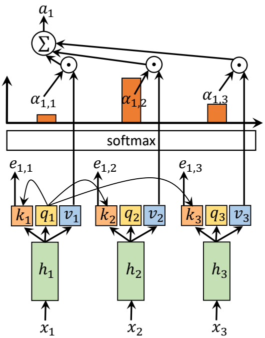
Source: CS W182 course, Sergey Levine, UC Berkeley

From Self-Attention to Transformers

- But to make this actually work, we need to develop a few additional components to address some fundamental limitations
- Positional encoding
  - Addresses lack of sequence information
- Multiheaded attention
  - allows querying multiple positions at each layer
- Adding nonlinearities
  - So far, each successive layer is *linear* in the previous one
    $a_l = \sum_t \alpha_{l,t} v_t$ where, $v_t = W_v h_t$
- Masked decoding
  - How to prevent attention lookups into the future?

Source: CS W182 course, Sergey Levine, UC Berkeley

Self-Attention is Linear



- $k(h_t) = W_k h_t,\ q(h_t) = W_q h_t,\ v(h_t) = W_v h_t$

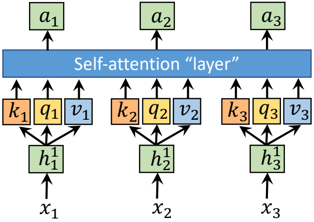
  $e_{l,t} = q_l \cdot k_t$

  $\alpha_{l,t} = \dfrac{\exp(e_{l,t})}{\sum_{t'} \exp(e_{l,t'})}$

  $a_l = \sum_t \alpha_{l,t} v_t = \sum_t \alpha_{l,t} W_v h_t = W_v \sum_t \alpha_{l,t} h_t$
- We make a non-linear choice of the timesteps we need to attend to, but we combine the the linear transformations of those timesteps
- Every self-attention "layer" is a linear transformation of the previous layer (with nonlinear weights)
- In many situations, This is not very expressive

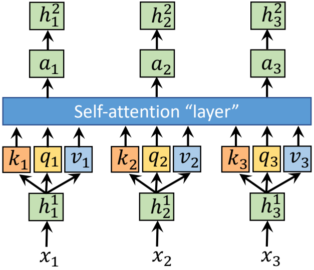
Source: CS W182 course, Sergey Levine, UC Berkeley

Alternating Self-Attention and Non-Linearity



Source: CS W182 course, Sergey Levine, UC Berkeley

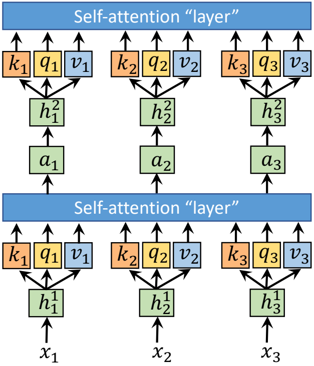
Alternating Self-Attention and Non-Linearity



- Some *learnable* non-linear function *e.g.*, $h_t^l = \sigma(W^l a_t + b^l)$
- It is just a neural network at every position after self-attention layer

Source: CS W182 course, Sergey Levine, UC Berkeley

Alternating Self-Attention and Non-Linearity



- Some *learnable* non-linear function *e.g.*, $h_t^l = \sigma(W^l a_t + b^l)$
- It is just a neural network at every position after self-attention layer
- Referred to as "position-wise feedforward network"

Source: CS W182 course, Sergey Levine, UC Berkeley

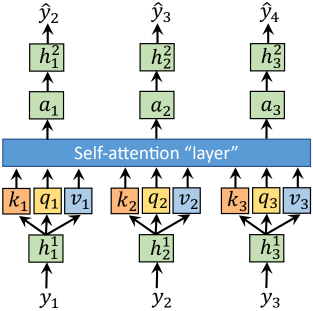
From Self-Attention to Transformers

- § But to make this actually work, we need to develop a few additional components to address some fundamental limitations
- § Positional encoding
  - ▶ Addresses lack of sequence information
- § Multiheaded attention
  - ▶ allows querying multiple positions at each layer
- § Adding nonlinearities
  - ▶ So far, each successive layer is *linear* in the previous one
    $a_l = \sum_t \alpha_{l,t} v_t$ where, $v_t = W_v h_t$
- § Masked decoding
  - ▶ How to prevent attention lookups into the future?

Source: CS W182 course, Sergey Levine, UC Berkeley

Self-Attention Can See the Future

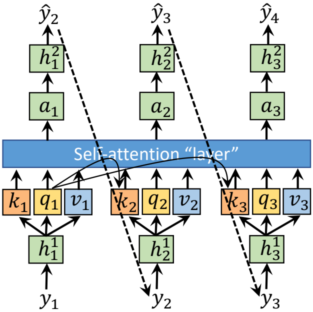
- Nothing is preventing a crude self-attention 'language model' to look into the future
- (In reality, we have many alternating self-attention layers and position-wise feedforward networks, not just one)



Source: CS W182 course, Sergey Levine, UC Berkeley

Self-Attention Can See the Future

- Nothing is preventing a crude self-attention 'language model' to look into the future
- (In reality, we have many alternating self-attention layers and position-wise feedforward networks, not just one)

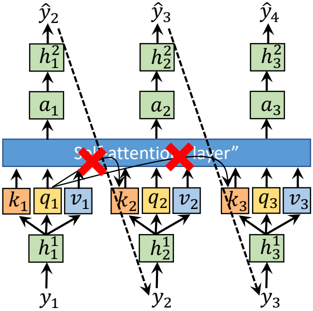


- Self-attention at first timestep needs to peek into activation at timestep two and three. but activation at timestep two depends on output from timestep one and activation at timestep three depends on output from timestep two

Source: CS W182 course, Sergey Levine, UC Berkeley

Self-Attention Can See the Future

- Nothing is preventing a crude self-attention 'language model' to look into the future
- (In reality, we have many alternating self-attention layers and position-wise feedforward networks, not just one)



- Self-attention at first timestep needs to peek into activation at timestep two and three. but activation at timestep two depends on output from timestep one and activation at timestep three depends on output from timestep two
- Easy solution of this circular dependency:

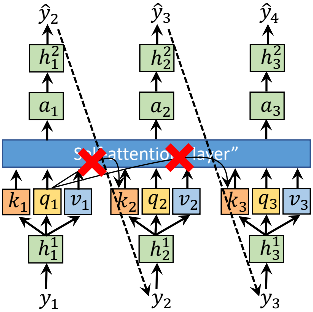
$\cancel{e_{l,t} = q_l \cdot k_t}$

$$e_{l,t} = \begin{cases} q_l \cdot k_t, & \text{if } l \geq t \\ -\infty, & \text{otherwise} \end{cases}$$

Source: CS W182 course, Sergey Levine, UC Berkeley

Self-Attention Can See the Future

- Nothing is preventing a crude self-attention 'language model' to look into the future
- (In reality, we have many alternating self-attention layers and position-wise feedforward networks, not just one)



- Self-attention at first timestep needs to peek into activation at timestep two and three. but activation at timestep two depends on output from timestep one and activation at timestep three depends on output from timestep two
- Easy solution of this circular dependency:

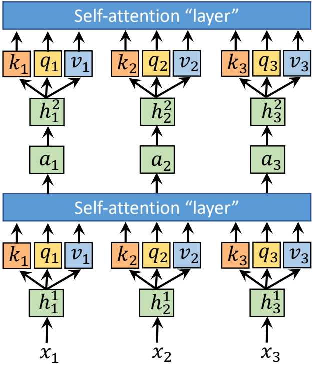
$\cancel{e_{l,t} = q_l \cdot k_t}$

$$e_{l,t} = \begin{cases} q_l \cdot k_t, & \text{if } l \geq t \\ -\infty, & \text{otherwise} \end{cases}$$

- In practice: Just replace $\exp(e_{l,t})$ with 0 if $l < t$ inside the softmax

Source: CS W182 course, Sergey Levine, UC Berkeley

Transformer

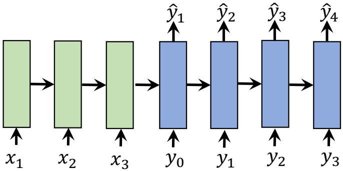


- We will combine the pieces that we learnt to get the classic Transformer model
- There are a number of model designs that use successive self-attention and position-wise nonlinear layers to process sequences
- These are generally called "Transformers" because they transform one sequence into another at each layer
  - See Vaswani *et al.* **Attention Is All You Need**, NeurIPS 2017
- The "classic" transformer (Vaswani et al. 2017) is a sequence to sequence model.
- A number of well-known follow works also use transformers for language modeling (BERT, GPT, *etc.*)

Source: CS W182 course, Sergey Levine, UC Berkeley

The "Classic" Transformer

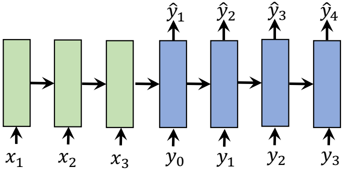
§ As compared to a sequence to sequence RNN model

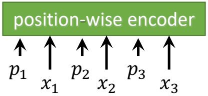


Source: CS W182 course, Sergey Levine, UC Berkeley

The "Classic" Transformer

§ As compared to a sequence to sequence RNN model

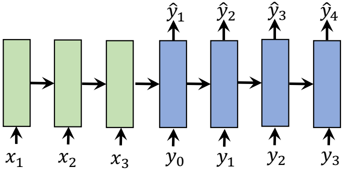


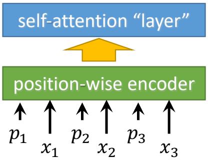


Source: CS W182 course, Sergey Levine, UC Berkeley

The "Classic" Transformer

§ As compared to a sequence to sequence RNN model

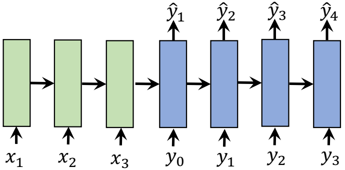


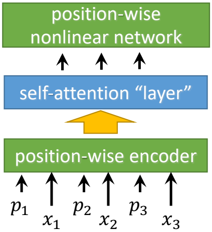


Source: CS W182 course, Sergey Levine, UC Berkeley

The "Classic" Transformer

- As compared to a sequence to sequence RNN model

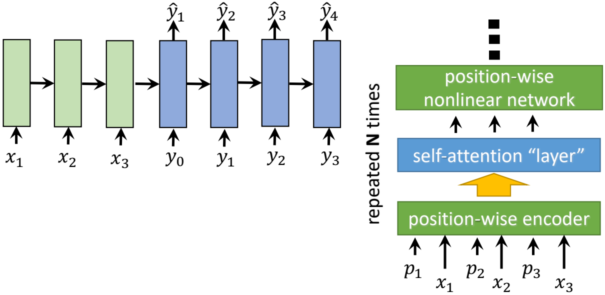




Source: CS W182 course, Sergey Levine, UC Berkeley

The "Classic" Transformer

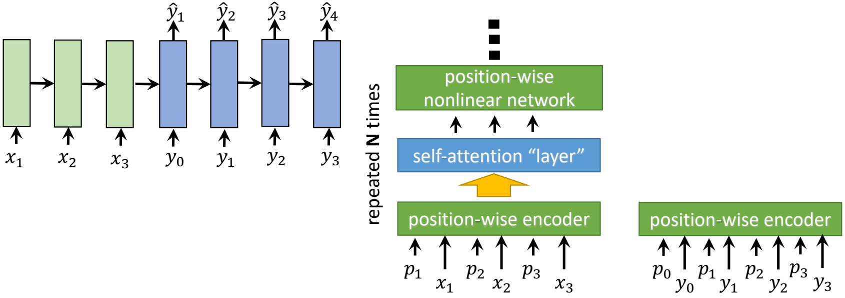
§ As compared to a sequence to sequence RNN model



Source: CS W182 course, Sergey Levine, UC Berkeley

The "Classic" Transformer

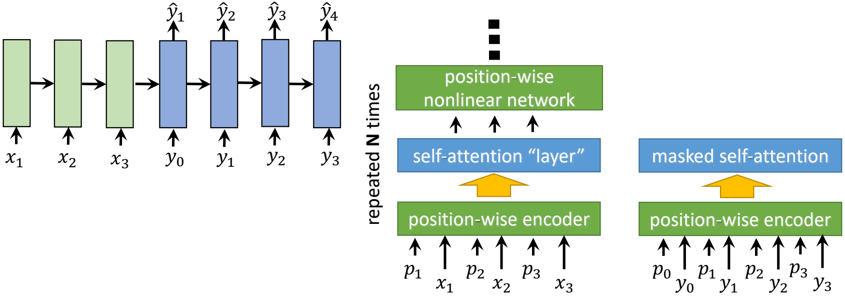
- As compared to a sequence to sequence RNN model



Source: CS W182 course, Sergey Levine, UC Berkeley

The "Classic" Transformer

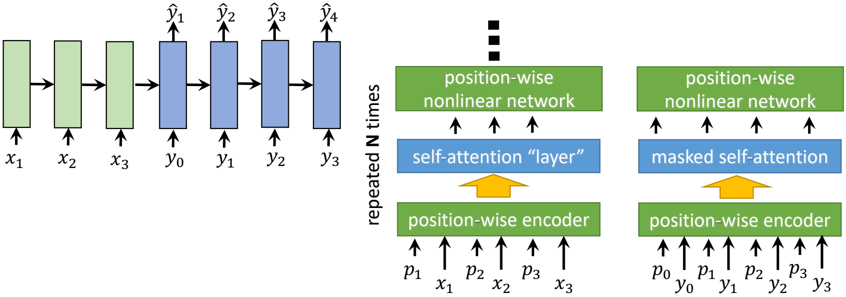
§ As compared to a sequence to sequence RNN model



Source: CS W182 course, Sergey Levine, UC Berkeley

The "Classic" Transformer

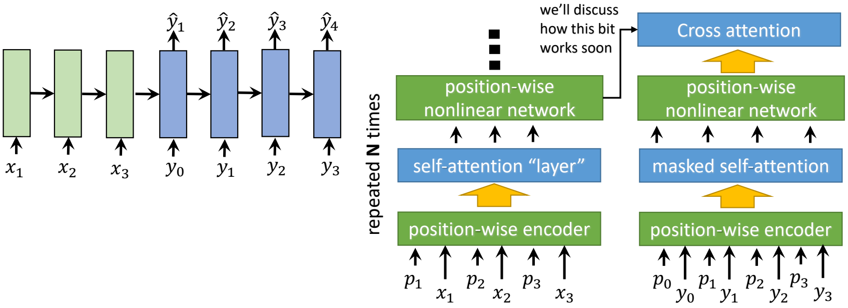
- As compared to a sequence to sequence RNN model



Source: CS W182 course, Sergey Levine, UC Berkeley

The "Classic" Transformer

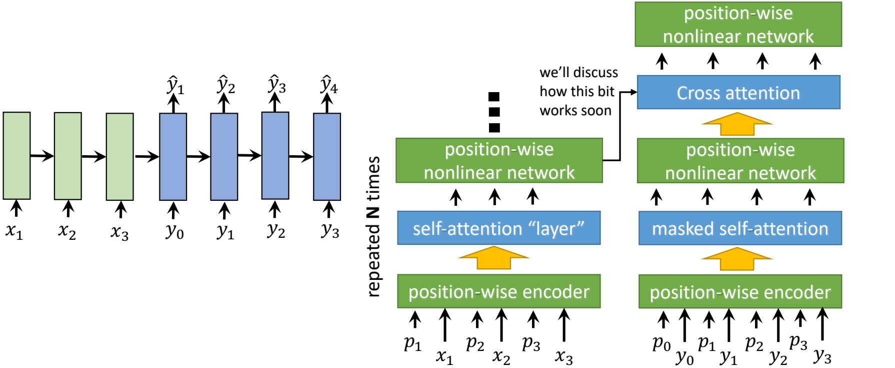
§ As compared to a sequence to sequence RNN model



Source: CS W182 course, Sergey Levine, UC Berkeley

The "Classic" Transformer

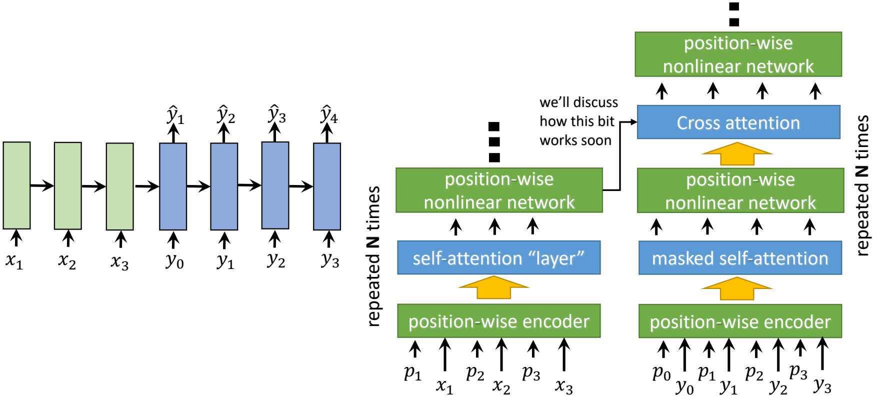
§ As compared to a sequence to sequence RNN model



Source: CS W182 course, Sergey Levine, UC Berkeley

The "Classic" Transformer

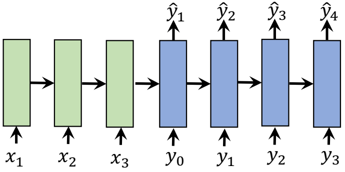
§ As compared to a sequence to sequence RNN model

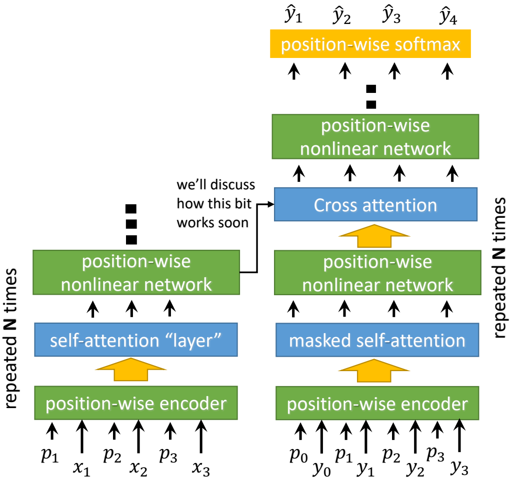


Source: CS W182 course, Sergey Levine, UC Berkeley

The "Classic" Transformer

- As compared to a sequence to sequence RNN model

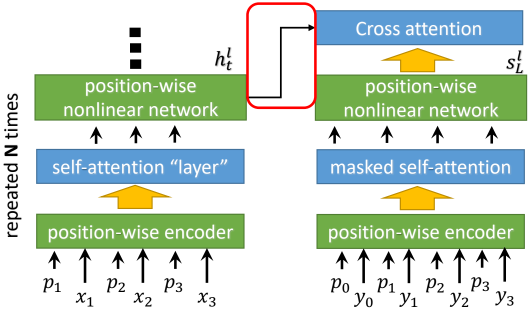




Source: CS W182 course, Sergey Levine, UC Berkeley

Cross-Attention: Combining Encoder and Decoder Values

§ Much like the standard attention



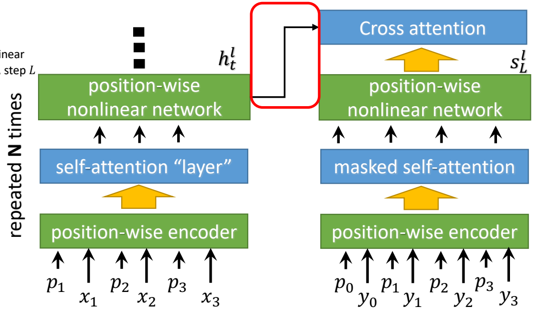
Source: CS W182 course, Sergey Levine, UC Berkeley

Cross-Attention: Combining Encoder and Decoder Values

§ Much like the standard attention

query: $q_L^l = W_q^l s_L^l$

output of position-wise nonlinear network at (decoder) layer l, step L



Source: CS W182 course, Sergey Levine, UC Berkeley

Cross-Attention: Combining Encoder and Decoder Values

§ Much like the standard attention

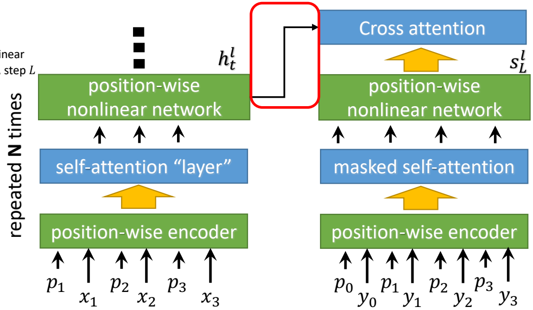
query: $q_L^l = W_q^l s_L^l$

output of position-wise nonlinear network at (decoder) layer l, step L

key: $k_t^l = W_k^l h_t^l$

value: $v_t^l = W_v^l h_t^l$

output of position-wise nonlinear network at (encoder) layer l, step t



Source: CS W182 course, Sergey Levine, UC Berkeley

Cross-Attention: Combining Encoder and Decoder Values

§ Much like the standard attention

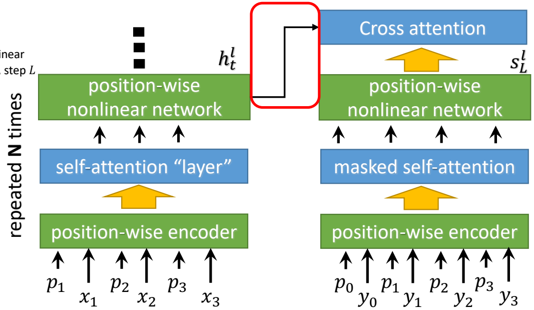
query: $q_L^l = W_q^l s_L^l$ output of position-wise nonlinear network at (decoder) layer l, step L

key: $k_t^l = W_k^l h_t^l$ output of position-wise nonlinear network at (encoder) layer l, step t

value: $v_t^l = W_v^l h_t^l$

$$e_{L,t}^l = q_L^l \cdot k_t^l$$

$$\alpha_{L,t}^l = \frac{\exp(e_{L,t}^l)}{\sum_{t'} \exp(e_{L,t'}^l)}$$



Source: CS W182 course, Sergey Levine, UC Berkeley

Cross-Attention: Combining Encoder and Decoder Values

§ Much like the standard attention

query: $q_L^l = W_q^l s_L^l$ output of position-wise nonlinear network at (decoder) layer l, step L

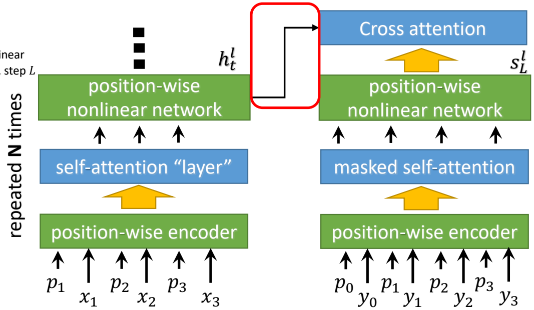
key: $k_t^l = W_k^l h_t^l$ output of position-wise nonlinear network at (encoder) layer l, step t

value: $v_t^l = W_v^l h_t^l$

$$e_{L,t}^l = q_L^l \, . \, k_t^l$$

$$\alpha_{L,t}^l = \frac{\exp(e_{L,t}^l)}{\sum_{t'} \exp(e_{L,t'}^l)}$$

$$c_L^l = \sum_t \alpha_{L,t}^l v_t^l$$ cross attention output



Source: CS W182 course, Sergey Levine, UC Berkeley

Cross-Attention: Combining Encoder and Decoder Values

§ Much like the standard attention

query: $q_L^l = W_q^l s_L^l$ output of position-wise nonlinear network at (decoder) layer l, step L

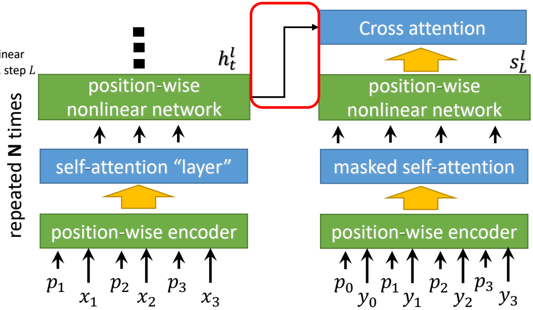
key: $k_t^l = W_k^l h_t^l$ output of position-wise nonlinear network at (encoder) layer l, step t

value: $v_t^l = W_v^l h_t^l$

$$e_{L,t}^l = q_L^l \,.\, k_t^l$$

$$\alpha_{L,t}^l = \frac{\exp(e_{L,t}^l)}{\sum_{t'} \exp(e_{L,t'}^l)}$$

$$c_L^l = \sum_t \alpha_{L,t}^l v_t^l$$ cross attention output



in reality, cross-attention is **also** multi-headed!

Source: CS W182 course, Sergey Levine, UC Berkeley

One Last Detail: Layer Normalization

- batch normalization is very helpful, but hard to use with sequence models
- Sequences are different lengths, makes normalizing across the batch hard
- Sequences can be very long, so we sometimes have small batches

Source: CS W182 course, Sergey Levine, UC Berkeley

One Last Detail: Layer Normalization

- batch normalization is very helpful, but hard to use with sequence models
- Sequences are different lengths, makes normalizing across the batch hard
- Sequences can be very long, so we sometimes have small batches
- Solution: "Layer normalization" like batch norm, but not across the batch

Source: CS W182 course, Sergey Levine, UC Berkeley

One Last Detail: Layer Normalization

- batch normalization is very helpful, but hard to use with sequence models
- Sequences are different lengths, makes normalizing across the batch hard
- Sequences can be very long, so we sometimes have small batches
- Solution: "Layer normalization" like batch norm, but not across the batch
- Batch norm

  d dimensional vectors for each sample in batch: $a_1, a_2, \cdots, a_B$

$$\mu = \frac{1}{B}\sum_{i=1}^{B} a_i$$

$$\sigma = \sqrt{\frac{1}{B}\sum_{i=1}^{B}(a_i - \mu)^2}$$

$$\bar{a}_i = \frac{a_i - \mu}{\sigma}\gamma + \beta$$

Source: CS W182 course, Sergey Levine, UC Berkeley

One Last Detail: Layer Normalization

- batch normalization is very helpful, but hard to use with sequence models
- Sequences are different lengths, makes normalizing across the batch hard
- Sequences can be very long, so we sometimes have small batches
- Solution: "Layer normalization" like batch norm, but not across the batch
- Batch norm

  d dimensional vectors for each sample in batch: $a_1, a_2, \cdots, a_B$

  $$\mu = \frac{1}{B}\sum_{i=1}^{B} a_i$$

  $$\sigma = \sqrt{\frac{1}{B}\sum_{i=1}^{B}(a_i - \mu)^2}$$

  $$\bar{a}_i = \frac{a_i - \mu}{\sigma}\gamma + \beta$$

- Layer norm

  One d dimensional vector a

  $$\mu = \frac{1}{d}\sum_{j=1}^{d} a_j$$

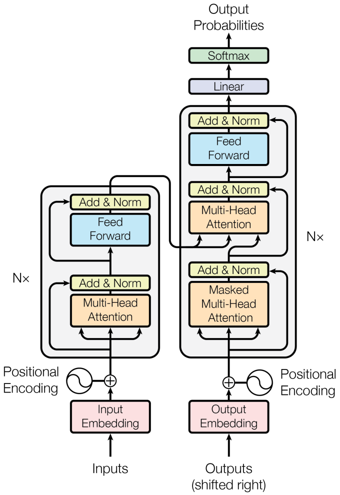
  $$\sigma = \sqrt{\frac{1}{d}\sum_{j=1}^{d}(a_j - \mu)^2}$$

  $$\bar{a} = \frac{a - \mu}{\sigma}\gamma + \beta$$

Source: CS W182 course, Sergey Levine, UC Berkeley

Putting it all together

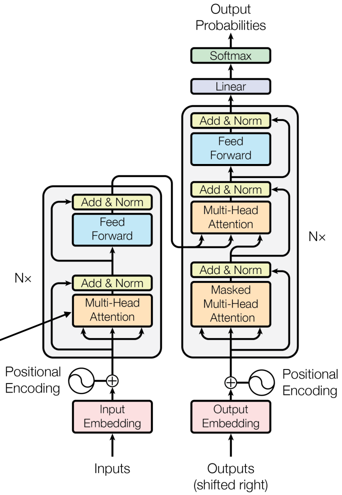
- The Transformer from Vaswani *et al.* ‘Attention Is All You Need’, NeurIPS, 2017



Source: CS W182 course, Sergey Levine, UC Berkeley

Putting it all together

§ The Transformer from Vaswani *et al.* 'Attention Is All You Need', NeurIPS, 2017



input: $\bar{h}_t^{l-1}$
output: a_t^l

concatenates attention from all heads

Source: CS W182 course, Sergey Levine, UC Berkeley

Putting it all together

- The Transformer from Vaswani *et al.* 'Attention Is All You Need', NeurIPS, 2017

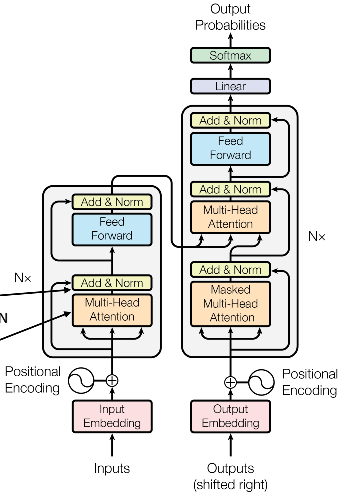
$\bar{a}_t^l = LayerNorm(\bar{h}_t^{l-1} + a_t^l)$

essentially a residual connection with LN

input: $\bar{h}_t^{l-1}$

output: a_t^l

concatenates attention from all heads



Source: CS W182 course, Sergey Levine, UC Berkeley

Putting it all together

- The Transformer from Vaswani *et al.* 'Attention Is All You Need', NeurIPS, 2017

$h_t^l = W_2^l RELU\left(W_1^l \bar{a}_t^l + b_1^l\right) + b_2^l$

2-layer neural net at each position

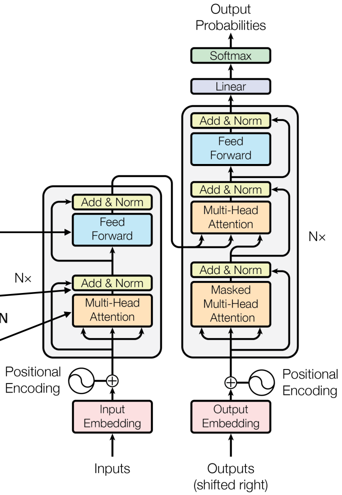
$\bar{a}_t^l = LayerNorm(\bar{h}_t^{l-1} + a_t^l)$

essentially a residual connection with LN

input: $\bar{h}_t^{l-1}$

output: a_t^l

concatenates attention from all heads



Source: CS W182 course, Sergey Levine, UC Berkeley

Putting it all together

§ The Transformer from Vaswani *et al*. ‘Attention Is All You Need’, NeurIPS, 2017

$\bar{h}_t^l = LayerNorm(\bar{a}_t^l + h_t^l)$

$h_t^l = W_2^l RELU\left(W_1^l \bar{a}_t^l + b_1^l\right) + b_2^l$

2-layer neural net at each position

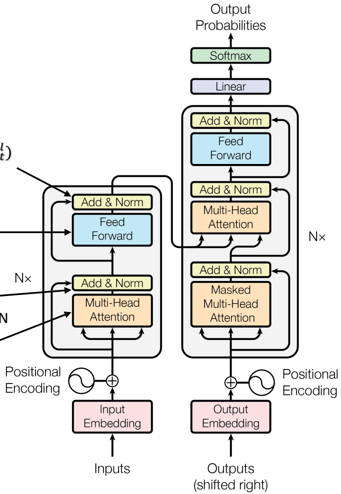
$\bar{a}_t^l = LayerNorm(\bar{h}_t^{l-1} + a_t^l)$

essentially a residual connection with LN

input: $\bar{h}_t^{l-1}$

output: a_t^l

concatenates attention from all heads



Source: CS W182 course, Sergey Levine, UC Berkeley

Putting it all together

§ The Transformer from Vaswani *et al*. ‘Attention Is All You Need’, NeurIPS, 2017

6 layers, each with d = 512

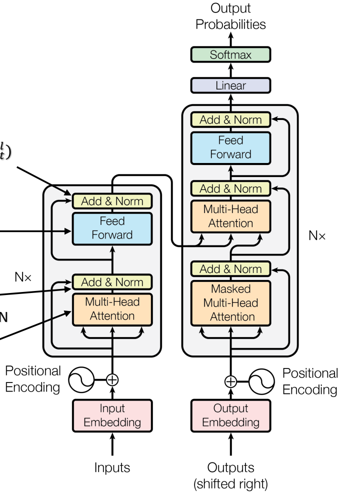
$\bar{h}_t^l = LayerNorm(\bar{a}_t^l + h_t^l)$
passed to next layer $l+1$

$h_t^l = W_2^l RELU\left(W_1^l \bar{a}_t^l + b_1^l\right) + b_2^l$
2-layer neural net at each position

$\bar{a}_t^l = LayerNorm(\bar{h}_t^{l-1} + a_t^l)$
essentially a residual connection with LN

input: $\bar{h}_t^{l-1}$
output: a_t^l

concatenates attention from all heads



Source: CS W182 course, Sergey Levine, UC Berkeley

Putting it all together

§ The Transformer from Vaswani *et al*. 'Attention Is All You Need', NeurIPS, 2017

multi-head attention keys and values
$k^l_{t,1}, \ldots, k^l_{t,m}$ and $v^l_{t,1}, \ldots, v^l_{t,m}$

6 layers, each with d = 512

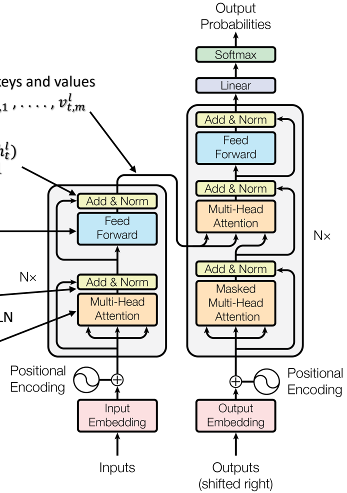
$\bar{h}^l_t = LayerNorm(\bar{a}^l_t + h^l_t)$
passed to next layer $l + 1$

$h^l_t = W^l_2 RELU\left(W^l_1 \bar{a}^l_t + b^l_1\right) + b^l_2$
2-layer neural net at each position

$\bar{a}^l_t = LayerNorm(\bar{h}^{l-1}_t + a^l_t)$
essentially a residual connection with LN

input: $\bar{h}^{l-1}_t$
output: a^l_t

concatenates attention from all heads



Source: CS W182 course, Sergey Levine, UC Berkeley

Putting it all together

§ The Transformer from Vaswani *et al.* 'Attention Is All You Need', NeurIPS, 2017

multi-head attention keys and values
$k^l_{t,1}, \ldots, k^l_{t,m}$ and $v^l_{t,1}, \ldots, v^l_{t,m}$

6 layers, each with d = 512

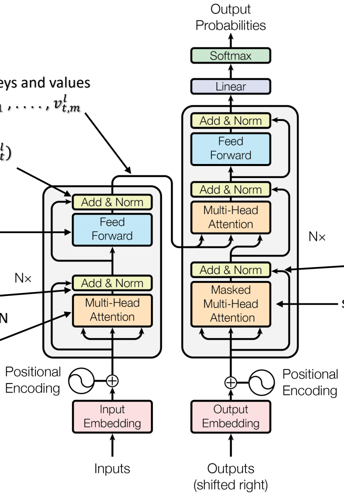
$\bar{h}^l_t = LayerNorm(\bar{a}^l_t + h^l_t)$
passed to next layer $l+1$

$h^l_t = W^l_2 RELU(W^l_1 \bar{a}^l_t + b^l_1) + b^l_2$
2-layer neural net at each position

$\bar{a}^l_t = LayerNorm(\bar{h}^{l-1}_t + a^l_t)$
essentially a residual connection with LN

input: $\bar{h}^{l-1}_t$
output: a^l_t

concatenates attention from all heads



residual connection with LN

same as encoder only **masked**

Source: CS W182 course, Sergey Levine, UC Berkeley

Putting it all together

§ The Transformer from Vaswani *et al.* 'Attention Is All You Need', NeurIPS, 2017

6 layers, each with d = 512

multi-head attention keys and values $k^l_{t,1}, \ldots, k^l_{t,m}$ and $v^l_{t,1}, \ldots, v^l_{t,m}$

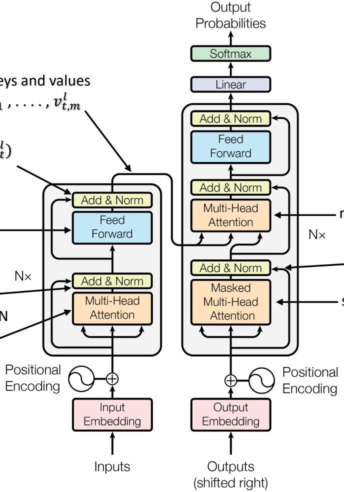
$\bar{h}^l_t = LayerNorm(\bar{a}^l_t + h^l_t)$
passed to next layer $l+1$

$h^l_t = W^l_2 RELU(W^l_1 \bar{a}^l_t + b^l_1) + b^l_2$
2-layer neural net at each position

$\bar{a}^l_t = LayerNorm(\bar{h}^{l-1}_t + a^l_t)$
essentially a residual connection with LN

input: $\bar{h}^{l-1}_t$
output: a^l_t

concatenates attention from all heads



multi-head cross attention

residual connection with LN

same as encoder only **masked**

Source: CS W182 course, Sergey Levine, UC Berkeley

Putting it all together

§ The Transformer from Vaswani *et al*. 'Attention Is All You Need', NeurIPS, 2017

multi-head attention keys and values
$k^l_{t,1},\dots,k^l_{t,m}$ and $v^l_{t,1},\dots,v^l_{t,m}$

6 layers, each with d = 512

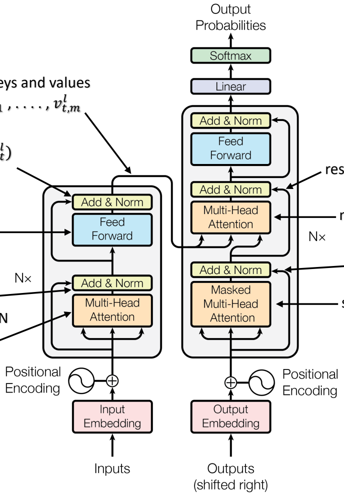
$\bar{h}^l_t = LayerNorm(\bar{a}^l_t + h^l_t)$
passed to next layer $l+1$

$h^l_t = W^l_2 RELU\left(W^l_1 \bar{a}^l_t + b^l_1\right) + b^l_2$
2-layer neural net at each position

$\bar{a}^l_t = LayerNorm(\bar{h}^{l-1}_t + a^l_t)$
essentially a residual connection with LN

input: $\bar{h}^{l-1}_t$
output: a^l_t

concatenates attention from all heads



residual connection with LN

multi-head cross attention

residual connection with LN

same as encoder only **masked**

Source: CS W182 course, Sergey Levine, UC Berkeley

Putting it all together

§ The Transformer from Vaswani *et al.* 'Attention Is All You Need', NeurIPS, 2017

multi-head attention keys and values
$k_{t,1}^l, \ldots, k_{t,m}^l$ and $v_{t,1}^l, \ldots, v_{t,m}^l$

6 layers, each with d = 512

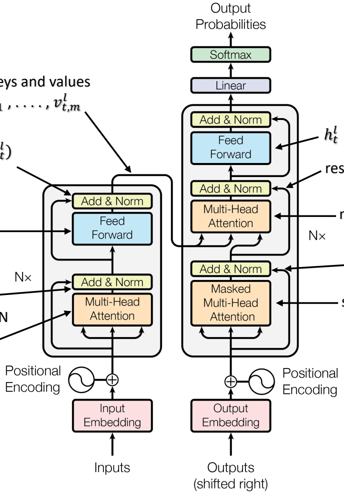
$\bar{h}_t^l = LayerNorm(\bar{a}_t^l + h_t^l)$
passed to next layer $l+1$

$h_t^l = W_2^l RELU(W_1^l \bar{a}_t^l + b_1^l) + b_2^l$
2-layer neural net at each position

$\bar{a}_t^l = LayerNorm(\bar{h}_t^{l-1} + a_t^l)$
essentially a residual connection with LN

input: $\bar{h}_t^{l-1}$
output: a_t^l

concatenates attention from all heads



$h_t^l = W_2^l RELU(W_1^l \bar{a}_t^l + b_1^l) + b_2^l$

residual connection with LN

multi-head cross attention

residual connection with LN

same as encoder only **masked**

Source: CS W182 course, Sergey Levine, UC Berkeley

Putting it all together

§ The Transformer from Vaswani *et al.* 'Attention Is All You Need', NeurIPS, 2017

6 layers, each with d = 512

multi-head attention keys and values
$k^l_{t,1}, \ldots, k^l_{t,m}$ and $v^l_{t,1}, \ldots, v^l_{t,m}$

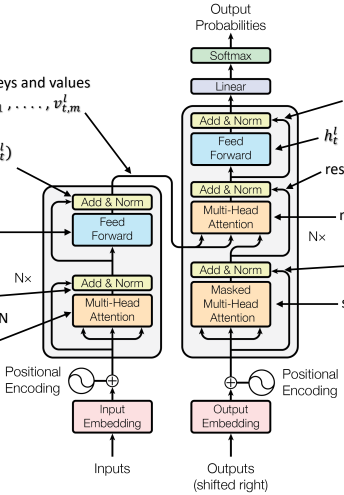
$\bar{h}^l_t = LayerNorm(\bar{a}^l_t + h^l_t)$
passed to next layer $l+1$

$h^l_t = W^l_2 RELU\left(W^l_1 \bar{a}^l_t + b^l_1\right) + b^l_2$
2-layer neural net at each position

$\bar{a}^l_t = LayerNorm(\bar{h}^{l-1}_t + a^l_t)$
essentially a residual connection with LN

input: $\bar{h}^{l-1}_t$
output: a^l_t

concatenates attention from all heads



residual connection with LN

$h^l_t = W^l_2 RELU\left(W^l_1 \bar{a}^l_t + b^l_1\right) + b^l_2$

residual connection with LN

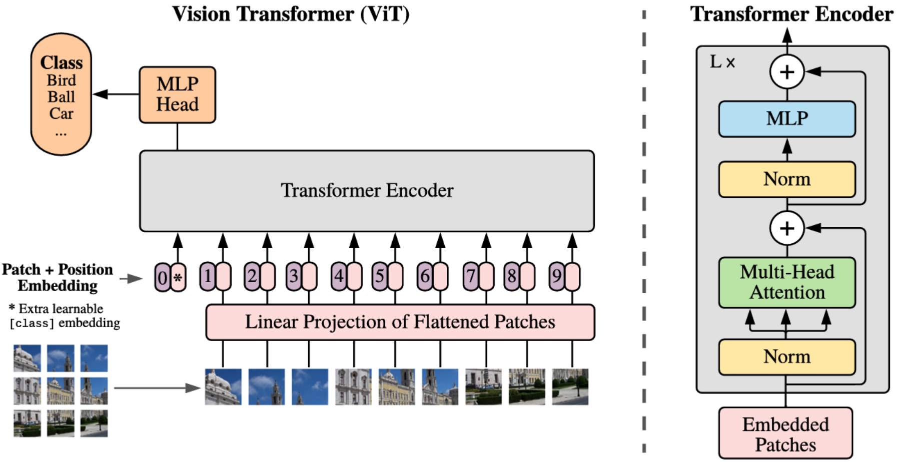
multi-head cross attention

residual connection with LN

same as encoder only **masked**

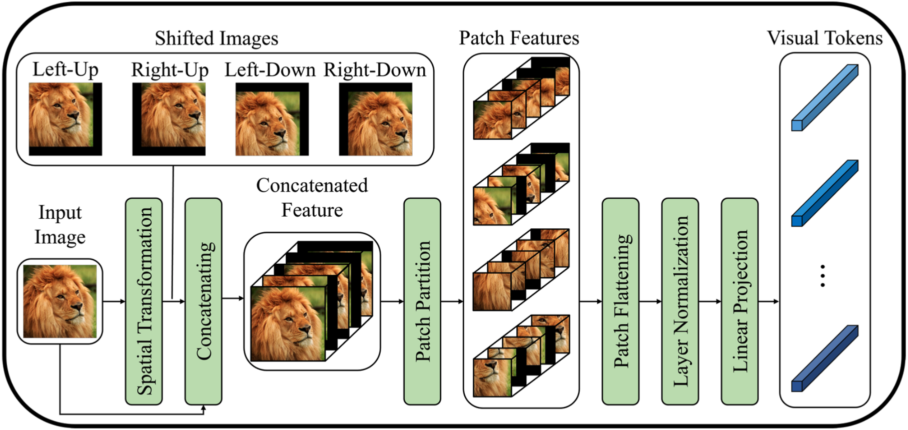
Source: CS W182 course, Sergey Levine, UC Berkeley

Vision Transformer: ViT



Source: An Image is Worth 16 × 16 Words: Transformers for Image Recognition at Scale

ViT for Small-Size Datasets



(a) Shifted Patch Tokenization

Source: Vision Transformer for Small-Size Datasets