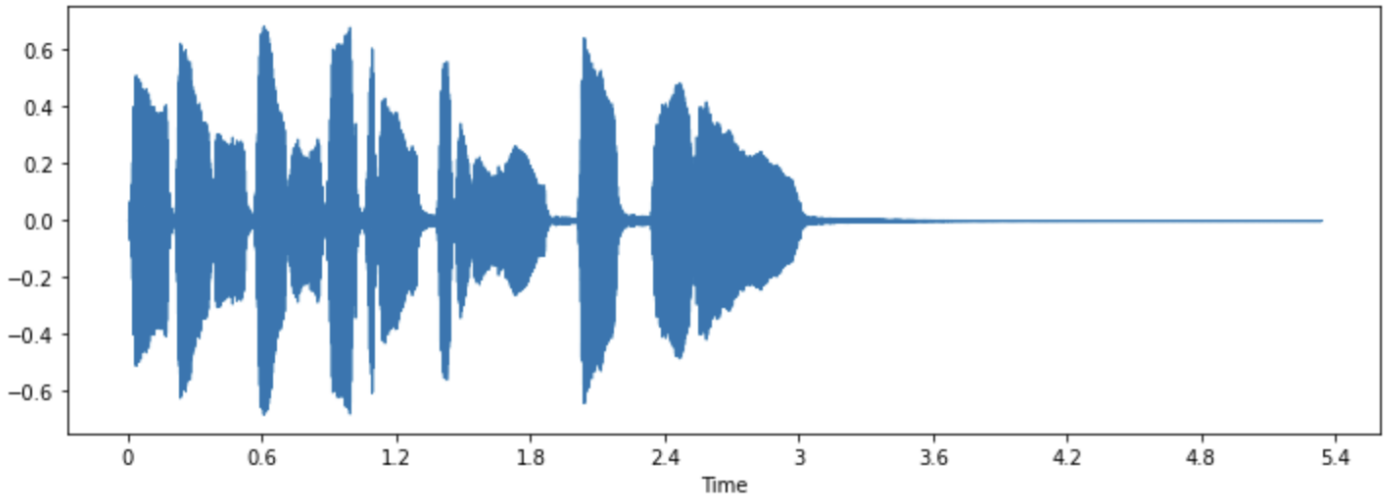
Temporal Networks

RNN, LSTM, Transformers

Temporal data

- The data can be spatial like an image, or treated as temporal like text, or speech.
- The speech varies at each second and its features can’t be preserved with CNN



Temporal feature extraction

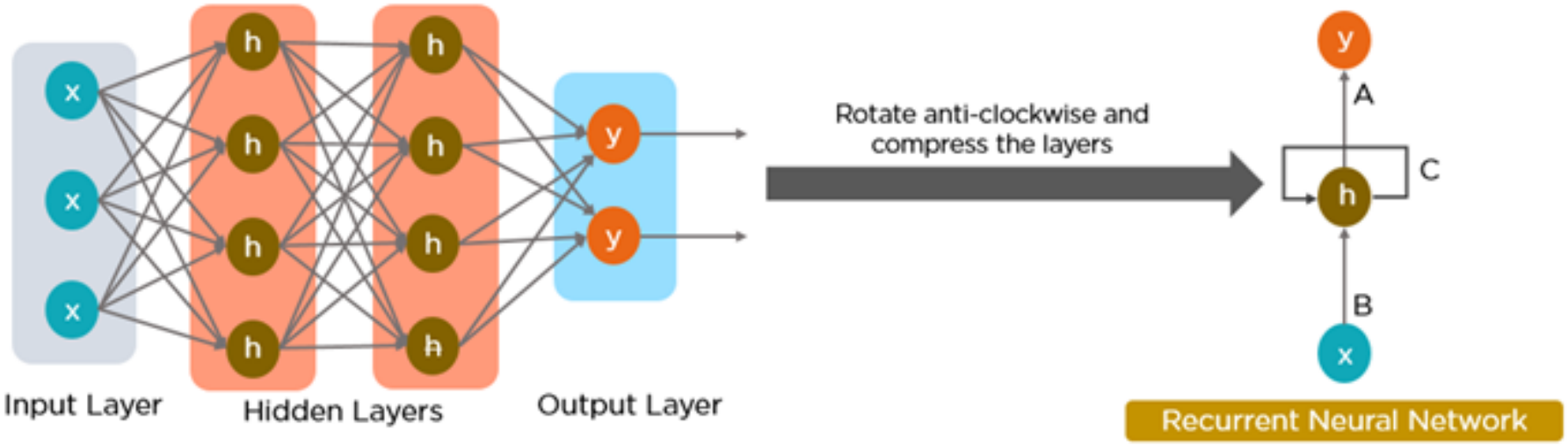
- Models that doesn’t have spatial kernel and can be used to model the long short term dependencies on temporal data like speech , text is needed!

RNN: An Intuition

- RNN stands for recurrent neural network
- They are used for sequence data
- a deep learning model that is trained to process and convert a sequential data input into a specific sequential data output

RNN

- RNNs, which are formed from feedforward networks, are similar to human brains in their behavior



RNN

- All of the inputs and outputs in standard neural networks are independent of one another, however in some circumstances, such as when predicting the next word of a phrase, the prior words are necessary, and so the previous words must be remembered. As a result, RNN was created, which used a Hidden Layer to overcome the problem.

RNN

Recurrent neural networks (RNNs) set have unique capabilities:

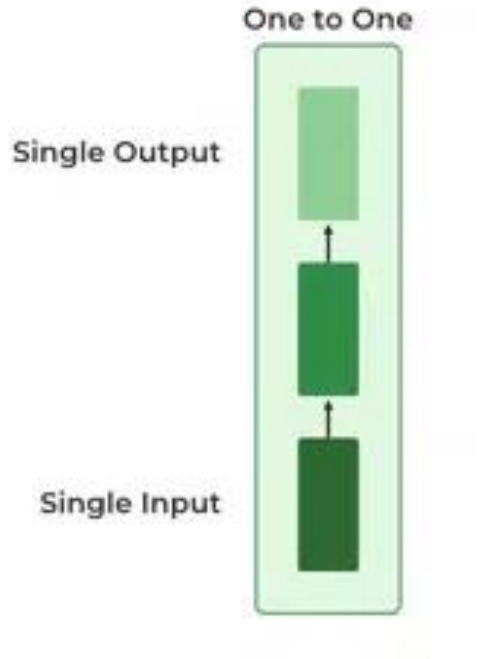
- **Internal Memory:** This is the key feature of RNNs. It allows them to remember past inputs and use that context when processing new information.
- **Sequential Data Processing:** Because of their memory, RNNs are exceptional at handling sequential data where the order of elements matters. This makes them ideal for tasks like speech recognition, machine translation, natural language processing(nlp) and text generation.
- **Contextual Understanding:** RNNs can analyze the current input in relation to what they've "seen" before. This contextual understanding is crucial for tasks where meaning depends on prior information.
- **Dynamic Processing:** RNNs can continuously update their internal memory as they process new data. This allows them to adapt to changing patterns within a sequence.

Key Points of RNN

- **Recurrent Connection:** A key distinction of RNNs is the recurrent connection within the hidden layer. This connection allows the network to pass the hidden state information (the network's memory) to the next time step.
- Input Layer
- Output Layer
- Hidden Layer

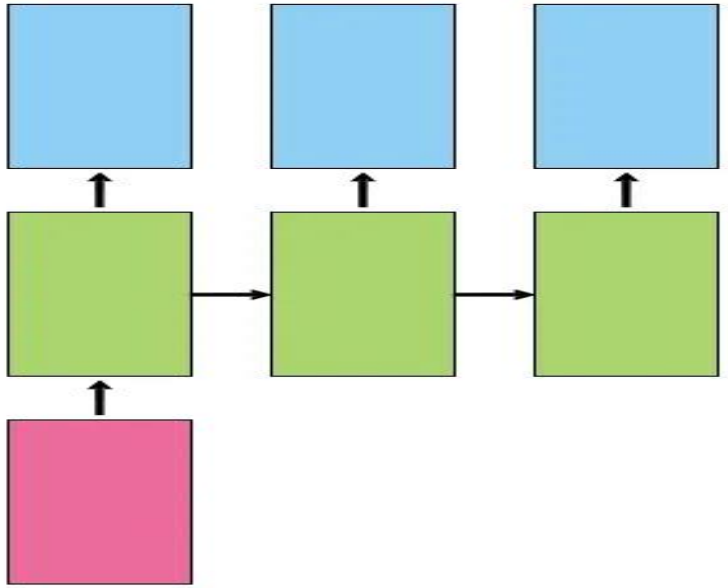
Sequence prediction

- One-to-one: This type of RNN behaves the same as any simple Neural network it is also known as Vanilla Neural Network. In this Neural network, there is only one input and one output.



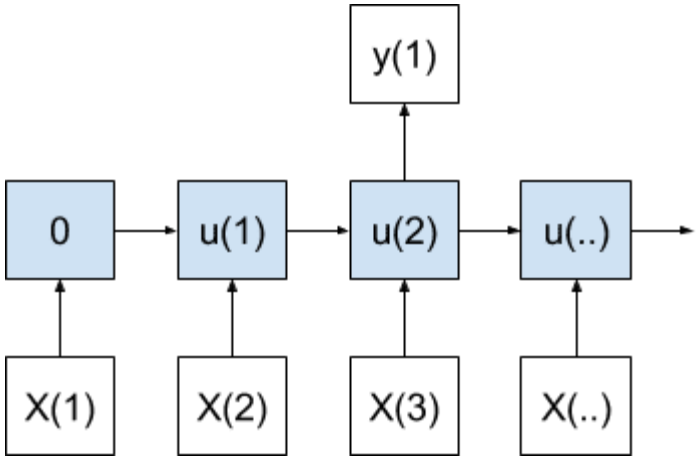
Sequence prediction

- One-to-many: The internal state is accumulated as each value in the output sequence is produced.



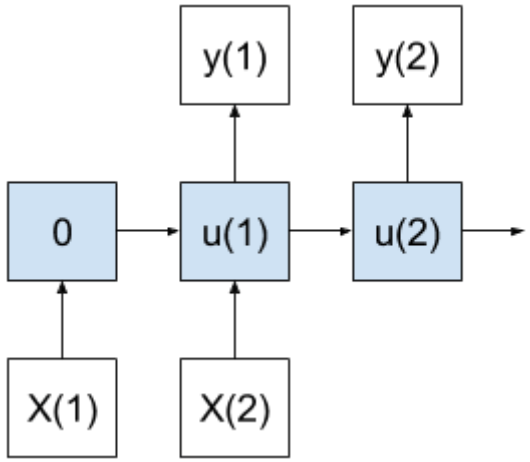
Sequence prediction

- Many to one: The internal state is accumulated with each input value before a final output value is produced.



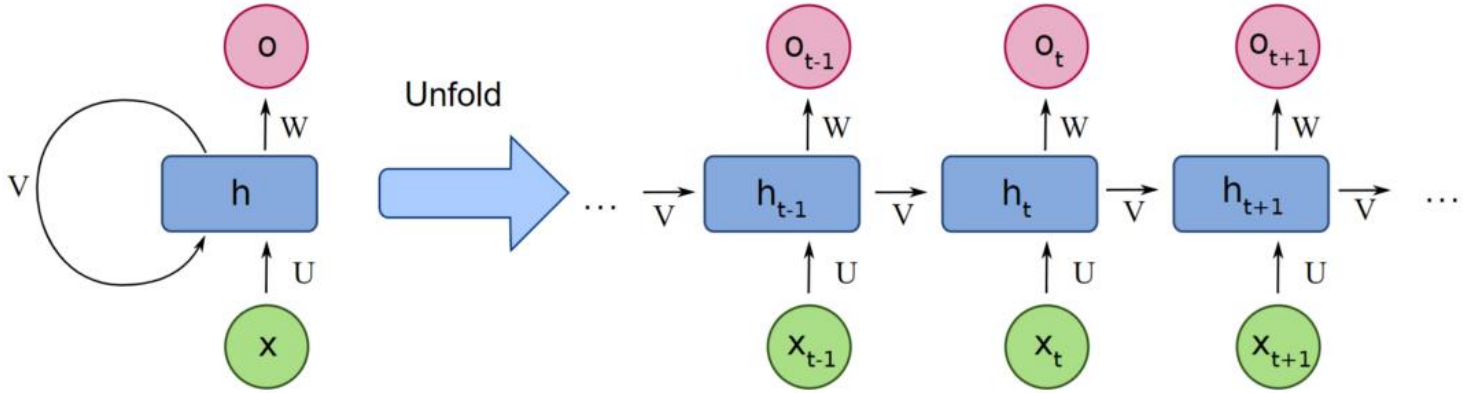
Sequence prediction

- Many to many: As with the many-to-one case, state is accumulated until the first output is created, but in this case multiple time steps are output.



RNN

- RNNs are a type of neural network that has hidden states and allows past outputs to be used as inputs



- The Recurrent Neural Network consists of multiple fixed activation function units, one for each time step.
- Each unit has an internal state which is called the hidden state of the unit.
- This hidden state signifies the past knowledge that the network currently holds at a given time step.
- This hidden state is updated at every time step to signify the change in the knowledge of the network about the past.

RNN

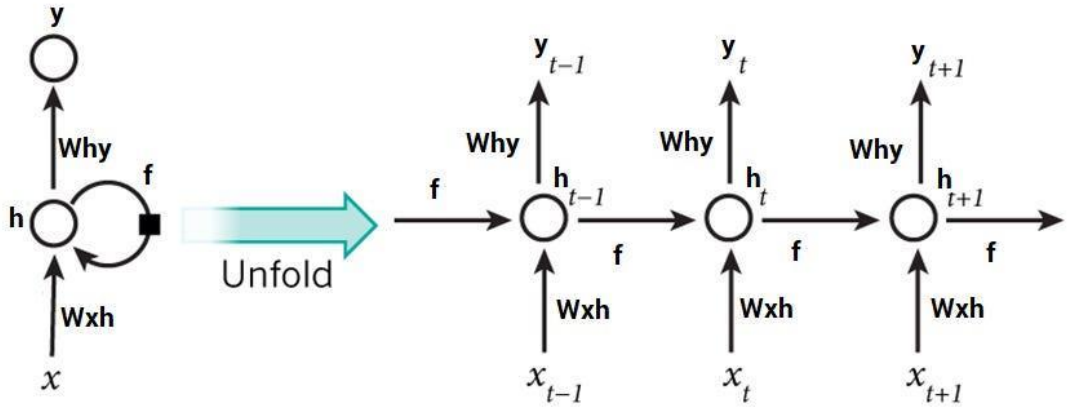
- The input layer **x** receives and processes the neural network's input before passing it on to the middle layer.
- Multiple hidden layers can be found in the middle layer **h**, each with its own activation functions, weights, and biases.

Steps

1. A single time step of the input is supplied to the network i.e. xt is supplied to the network
2. We then calculate its current state using a combination of the current input and the previous state i.e. we calculate ht
3. The current ht becomes ht-1 for the next time step
4. We can go as many time steps as the problem demands and combine the information from all the previous states
5. Once all the time steps are completed the final current state is used to calculate the output yt
6. The output is then compared to the actual output and the error is generated
7. The error is then backpropagated to the network to update the weights(we shall go into the details of backpropagation in further sections) and the network is trained

Forward pass

- current state $\mathbf{h_t = f(h_{t-1}, x_t)}$
- Here, f is the activation function
- So, $h_t = \tanh(W_{hh} * h_{t-1}, W_{xh} * x_t)$



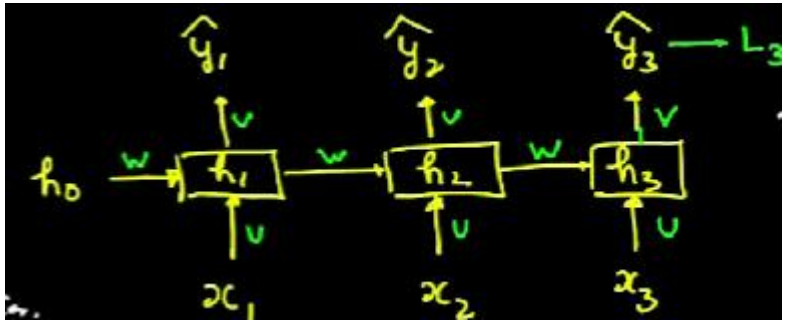
- output state $\mathbf{o_t = W_{h\gamma} * h_t}$
- Here, o_t = output state; $W_{h\gamma}$ = weight at output layer; h_t = current state

Backward pass

In forward pass,

- $h_t=\tanh(W_{hh}*h_{t-1} + W_{xh} * x_t)$ and here $W_{hh} = W$; $W_{xh} = U$
- so $h_t=\tanh(W*h_{t-1} + U* x_t)$
- $y`_t = \text{ReLU}(W_{yh}*h_t)$ and $W_{yh} = V$
- so $y`_t$ = linear Activation function$(V*h_t)$

for backward propagation through time we need to find **∂L/∂W, ∂L/∂V and ∂L/∂U.**



Backward pass

- $y_3 = ReLU(V*h_3)$;
- $y_t = ReLU(V*h_t)$;
- $L_3 = 1/2(y_3 - y`_3)^2$
- **To find $\partial L_3/\partial V$,**
  - $\partial L_3/\partial V = \partial L_3/\partial y`_3 * \partial y`_3/\partial V$
  - $\partial L_3/\partial V = -(y_3 - y`_3) * h_3$

Backward pass

- To find $\partial L_3/\partial W$
  - $\partial L_3/\partial W = \partial L_3/\partial y`_3 * \partial y`_3/\partial h_3 * \partial h_3/\partial W$
  - As seen $\partial L_3/\partial y`_3 = -(y_3 - y`_3)$ and $\partial y`_3/\partial h_3 = V$
  - $\partial h_3/\partial W \Rightarrow h_3 = g(W*h_2 + U*x_3) \Rightarrow h_3 = g*z_3$; here g = Activation function
  - Passing it through different hidden layers,
    - $\partial h_3/\partial W = g` * z_3 * [h_2 + W * \partial h_2/\partial W]$
    - $\partial h_2/\partial W = g` * z_2 * [h_1 + W * \partial h_1/\partial W]$
    - $\partial h_1/\partial W = g` * z_1 * [h_0 + W * \partial h_0/\partial W]$
  - $\partial h_3/\partial W = g` * z_3 * [h_2 + W * g` * z_2 * [h_1 + W * g` * z_1 * [h_0 + W * \partial h_0/\partial W]]]$

Backward pass

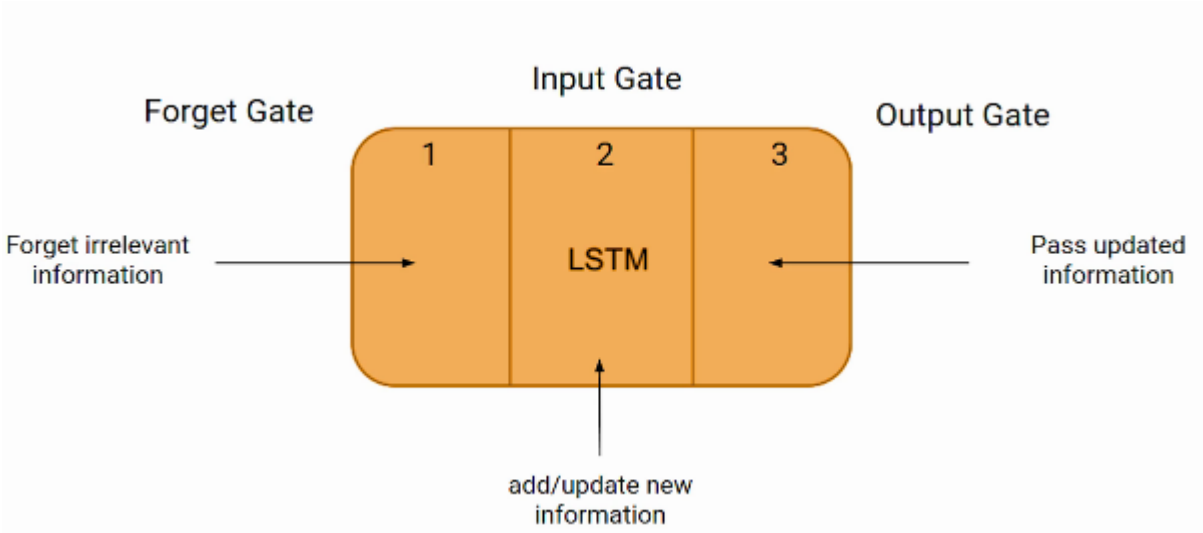
- To find $\partial L_3/\partial U$
- $\partial L_3/\partial U = \partial L_3/\partial y`_3 * \partial y`_3/\partial h_3 * \partial h_3/\partial U$
  - $\partial h_3/\partial U \Rightarrow h_3 = g(W*h_2 + U*x_3) \Rightarrow h_3 = g * z_3$; here g = Activation function
  - $\partial h_3/\partial U = \partial g(z_3)/\partial z_3 * \partial z_3/\partial U \Rightarrow g`*z_3 * [x_3 + u* \partial x_3/\partial U + \partial(Wh_2)/\partial U]$
  - here $u* \partial x_3/\partial U = 0$
  - $\partial(Wh_2)/\partial U = W * \partial h_2/\partial U + \partial W/\partial U * h_2$ and here $\partial W/\partial U * h_2 = 0$
  - $\partial h_2/\partial U = W* \partial g * z_2 / \partial U \Rightarrow \partial h_2/\partial U = W*g`* z_2 * \partial z_2 / \partial U$ and $\partial z_2 / \partial U$ which is not zero since $z_2 = W*h_1 + U*x_2$ and continue it until z_1.

LSTM

- RNNs have short term dependencies. Further, they are more prone to the vanishing gradient problem
- LSTM stands for long short term memory and is more elegant solution

LSTM

- LSTM consists of three parts

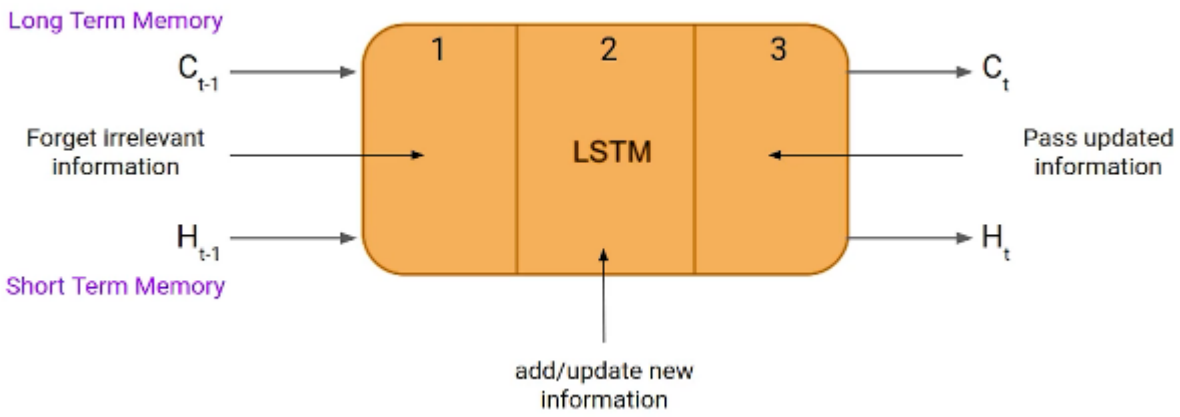


LSTM

- The first part chooses whether the information coming from the previous timestamp is to be remembered or is irrelevant and can be forgotten.
- In the second part, the cell tries to learn new information from the input to this cell.
- Third part, the cell passes the updated information from the current timestamp to the next timestamp.

LSTM

- Just like a simple RNN, an LSTM also has a hidden state where H(t-1) represents the hidden state of the previous timestamp and Ht is the hidden state of the current timestamp. In addition to that, LSTM also has a cell state represented by C(t-1) and C(t) for the previous and current timestamps, respectively.

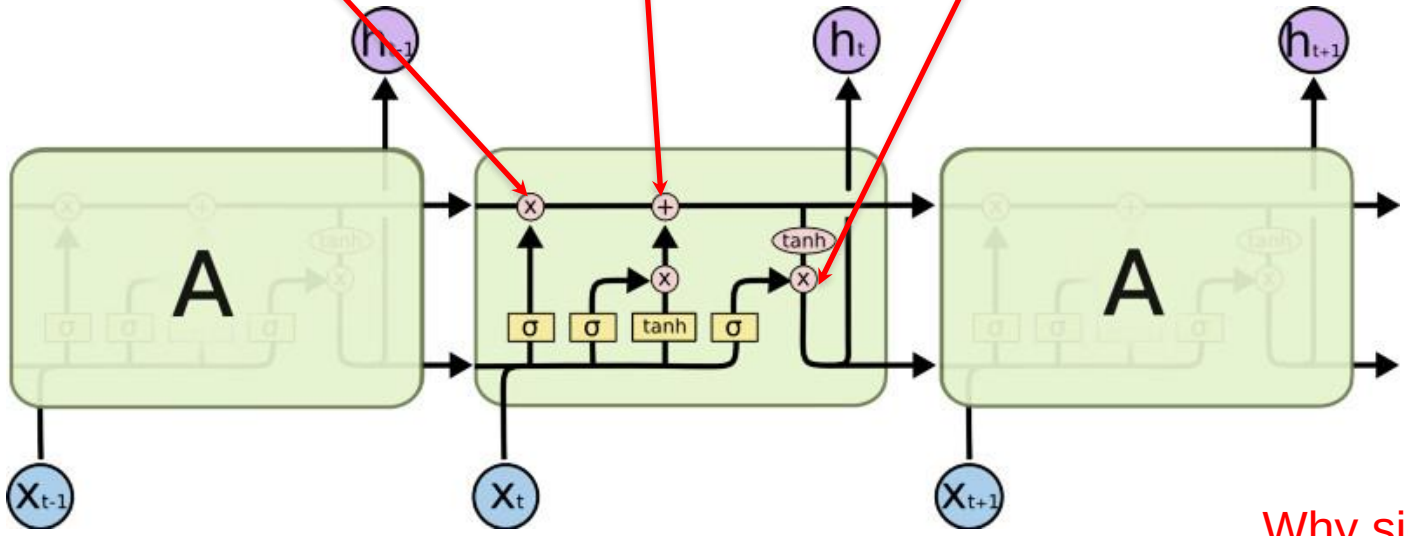


LSTM

Forget gate: This sigmoid gate determines how much information goes thru

Input gate: This decides what info
Is to add to the cell state

Output gate
Controls what goes into output

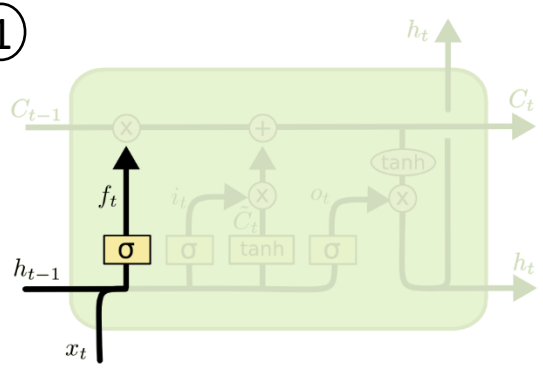


The core idea is this cell state C_t, it is changed slowly, with only minor linear interactions. It is very easy for information to flow along it unchanged.

Why sigmoid or tanh:
Sigmoid: 0,1 gating as switch.
Vanishing gradient problem in LSTM is handled already.
ReLU replaces tanh ok?

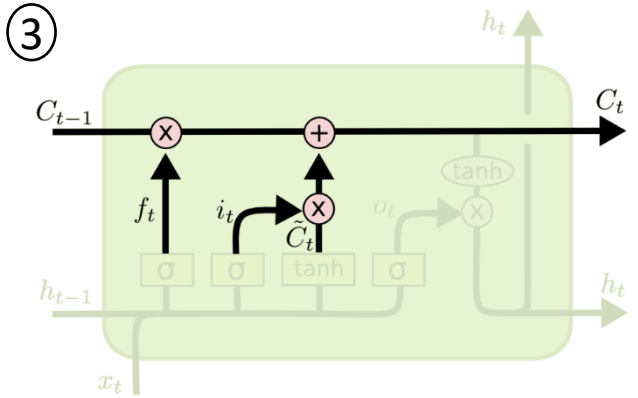
LSTM: Forward pass

①



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

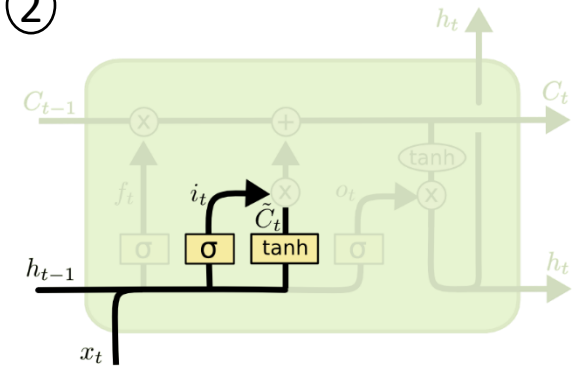
③



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Updating the cell state

②

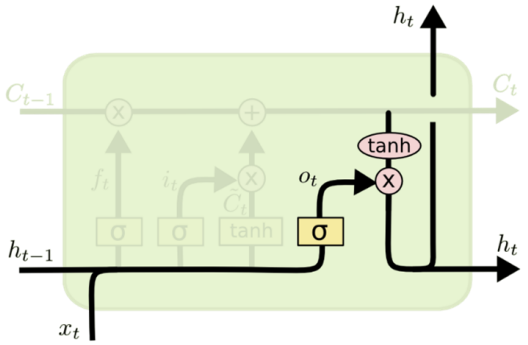


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

i_t decides what component is to be updated.
C'_t provides change contents

④

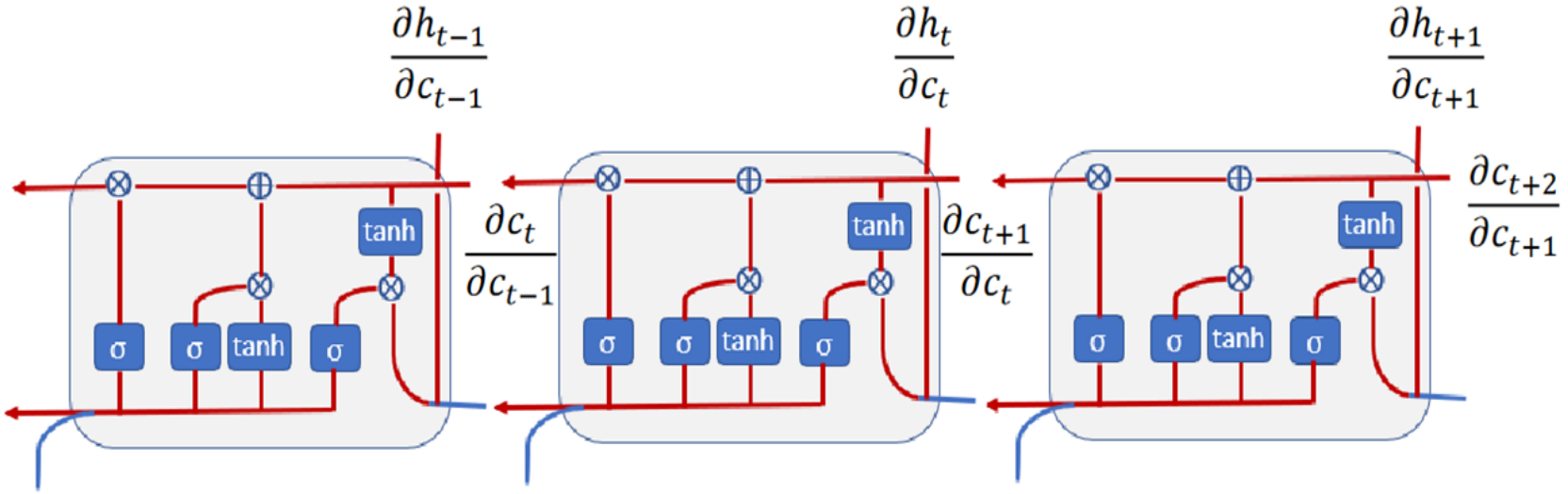


$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Decide what part of the cell state to output

LSTM: Backward Pass



Total error E can be defined as

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}$$

$$\frac{\partial E_t}{\partial W} = \frac{\partial E}{\partial h_t}\frac{\partial h_t}{\partial c_t}\frac{\partial c_t}{\partial c_{t-1}}\cdots\frac{\partial c_2}{\partial c_1}\frac{\partial c_1}{\partial W} = \frac{\partial E}{\partial h_t}\frac{\partial h_t}{\partial c_t}\left(\prod_{t=2}^{T}\frac{\partial c_t}{\partial c_{t-1}}\right)\frac{\partial c_1}{\partial W}$$

LSTM: Backward Pass

$$c_t = c_{t-1} \otimes f_t \oplus i_t \otimes \tilde{c}_t$$

$$\frac{\partial c_t}{\partial c_{t-1}} = \frac{\partial}{\partial c_{t-1}}(c_{t-1} \otimes f_t \oplus i_t \otimes \tilde{c}_t)$$

$$= \frac{\partial}{\partial c_{t-1}}[c_{t-1} \otimes f_t] \oplus [i_t \otimes \tilde{c}_t]$$

$$= \frac{\partial c_{t-1}}{\partial c_{t-1}} f_t + \frac{\partial f_t}{\partial c_{t-1}} c_{t-1} + \frac{\partial \tilde{c}_t}{\partial c_{t-1}} i_t + \frac{\partial i_t}{\partial c_{t-1}} \tilde{c}_t$$

$$= f_t + \frac{\partial f_t}{\partial c_{t-1}} c_{t-1} + \frac{\partial \tilde{c}_t}{\partial c_{t-1}} i_t + \frac{\partial i_t}{\partial c_{t-1}} \tilde{c}_t$$

$p \qquad q \qquad\qquad r \qquad\qquad s$

$$f_t = \sigma(W_f \cdot x_t + U_f \cdot h_{t-1} + b_i)$$

$$i_t = \sigma(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i)$$

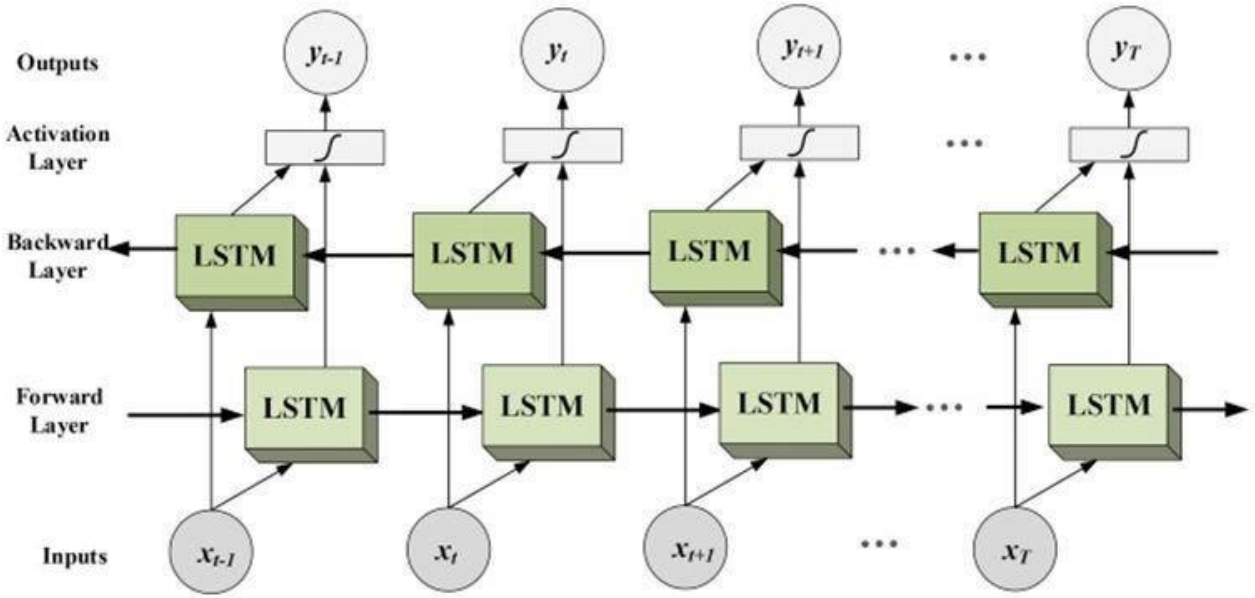
$$\tilde{c}_t = tanh(W_c \cdot x_t + U_c \cdot h_{t-1} + b_c)$$

$$h_t = tanh(c_t) \otimes o_t$$

$$\frac{\partial E_t}{\partial W} = \frac{\partial E}{\partial h_t}\frac{\partial h_t}{\partial c_t}\left(\prod_{t=2}^{T}(p + q + r + s)\right)\frac{\partial c_1}{\partial W}$$

Bi-directional LSTM

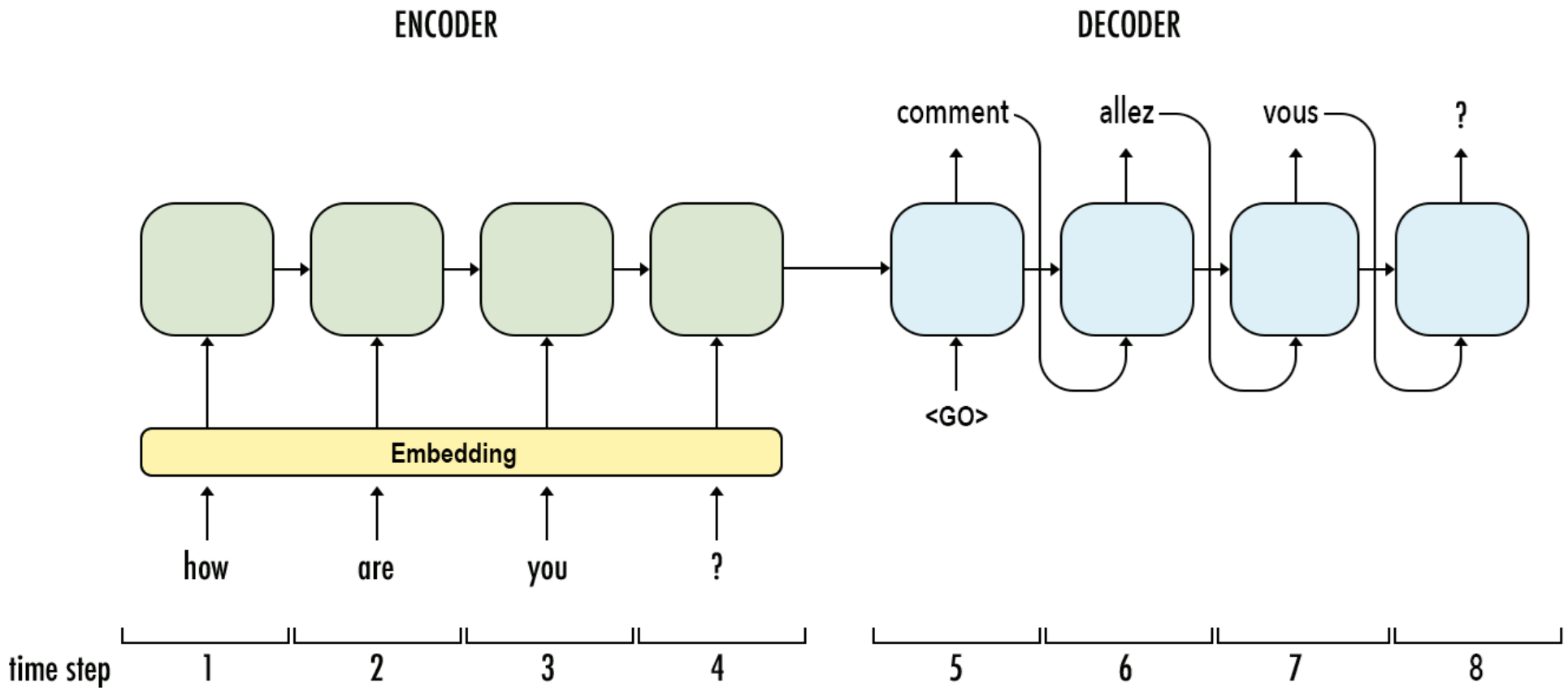
- The bidirectional LSTM comprises two LSTM layers, one processing the input sequence in the forward direction and the other in the backward direction. This allows the network to access information from past and future time steps simultaneously.



Drawbacks of LSTM

- Problem of vanishing gradient persists
- Slower than RNN
- Distance between layers is linear
- Does not exhibit parallelism

Application: Sequence modelling

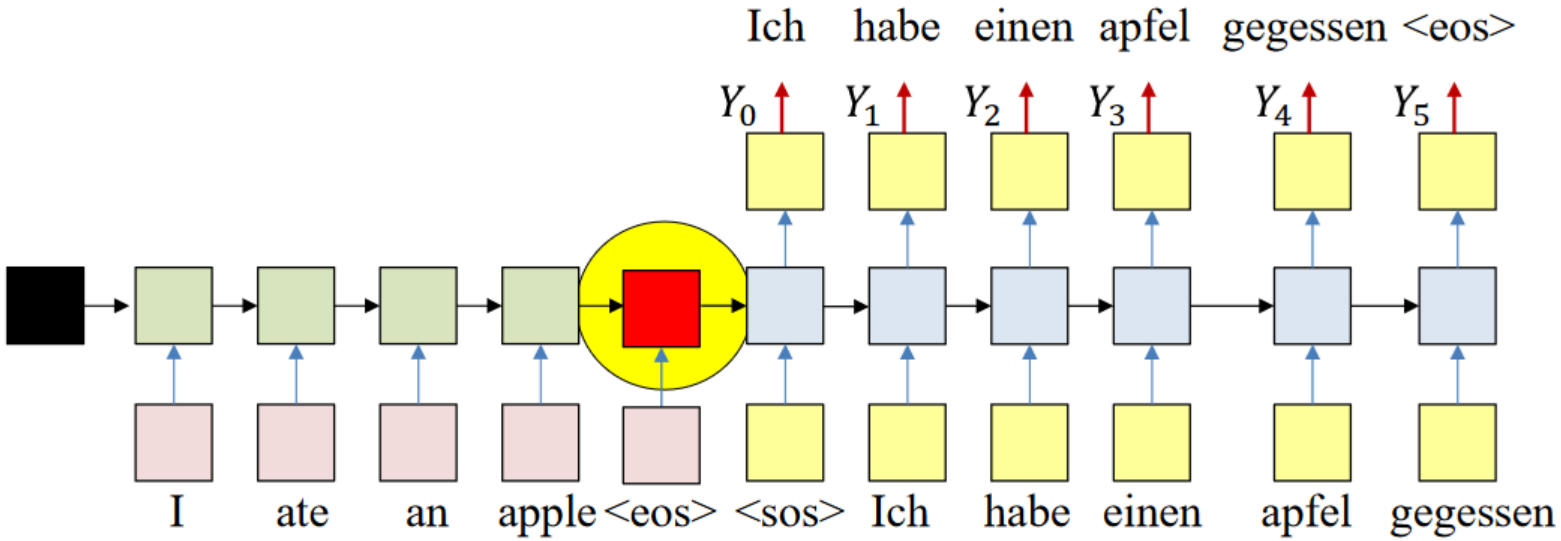


Sequence modelling

- Sequence goes in, sequence comes out
- No notion of “time synchrony” between input and output
  - May even not even maintain order of symbol

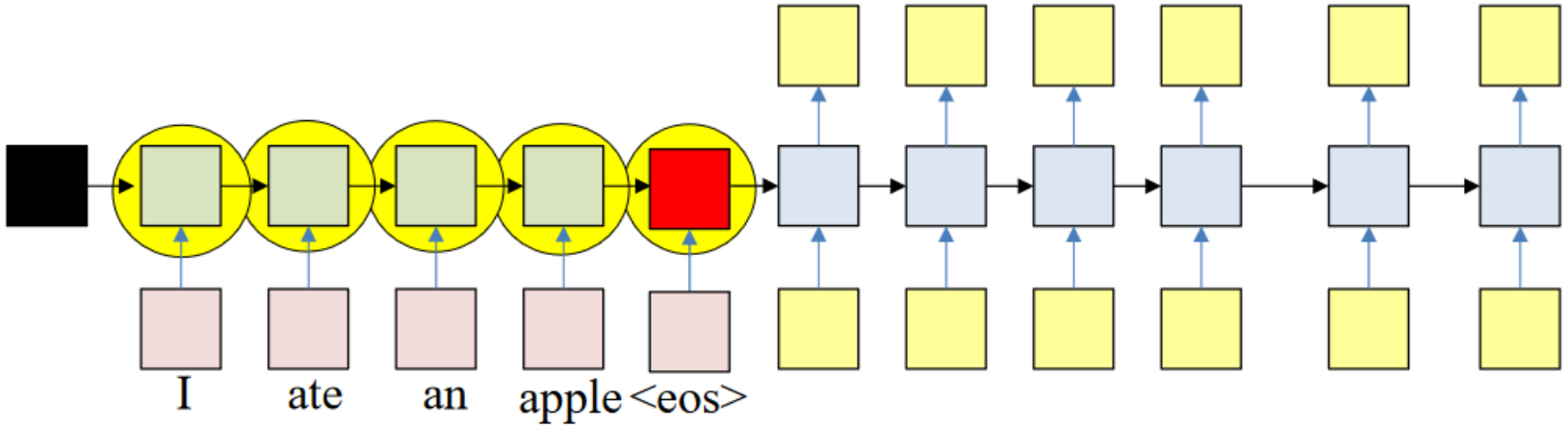
Problem with RNN/LSTM

- Dependency on the last node of encoder that contains the entire input embedding
- If input too large it fails



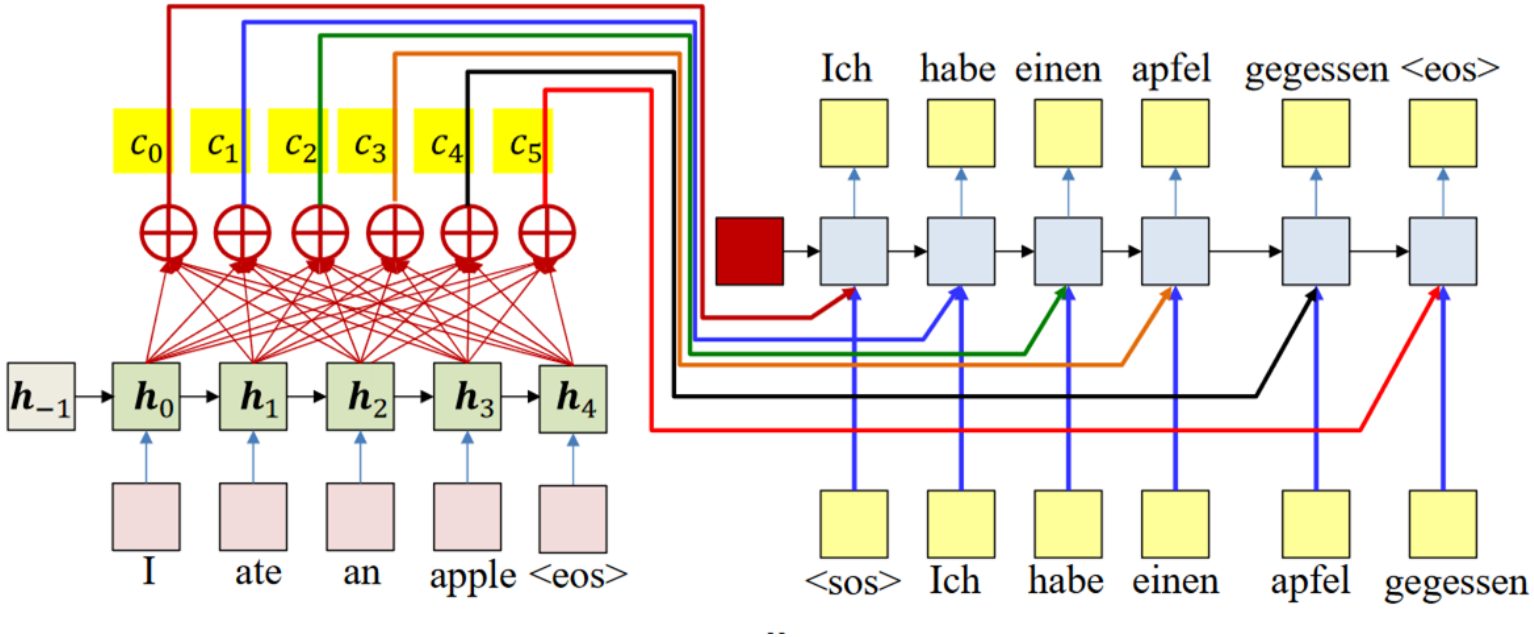
Sequence modelling

- All the hidden layers contain some information
- Each output uses some information from input hidden layers



Sequence modelling

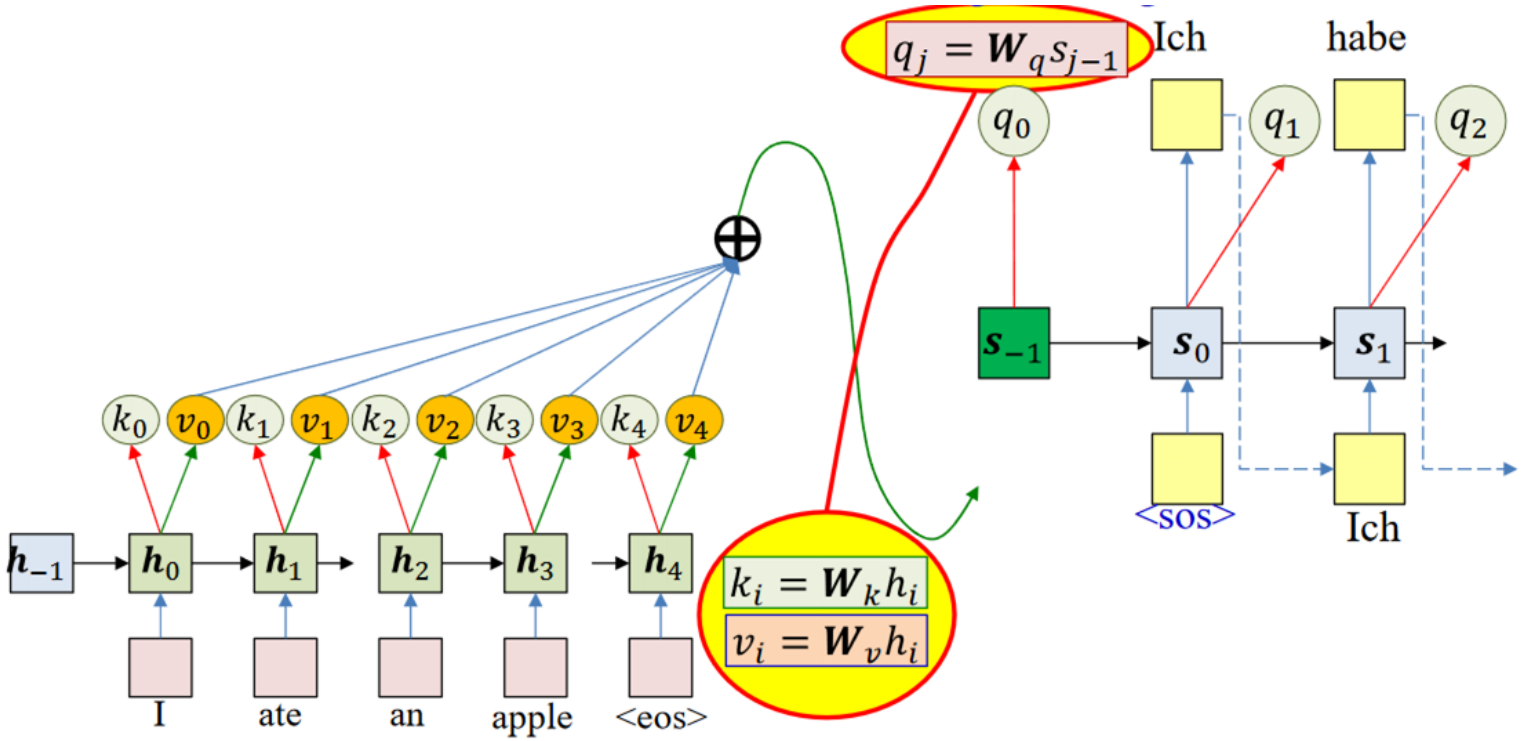
- A mechanism is required that can be used to focus on right word from input
- when predicting the word “apfel”, the weight for “apple” must be high while the rest must be low



Attention Mechanism

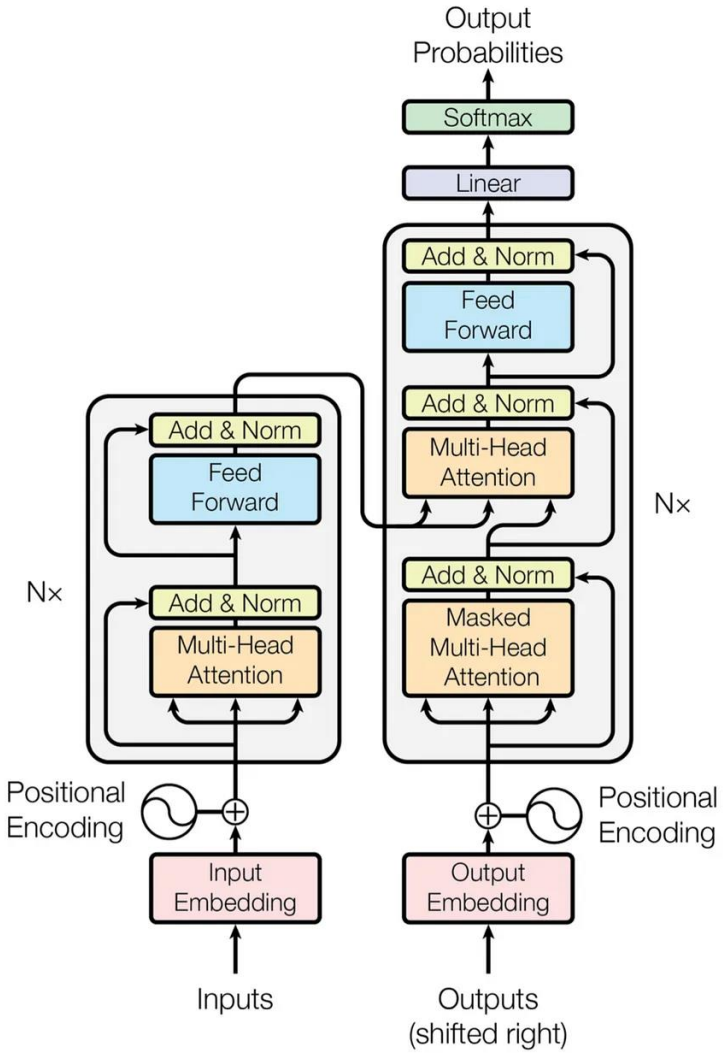
- The attention mechanism focuses on computing the weights dynamically as function of decoder state
- Encoder outputs an explicit “key” and “value” at each input time
  - Key is used to evaluate the importance of the input at that time, for a given output
- Decoder outputs an explicit “query” at each output time
  - Query is used to evaluate which inputs to pay attention to
- The weight is a function of key and query
- The actual context is a weighted sum of value

Attention mechanism



Transformers

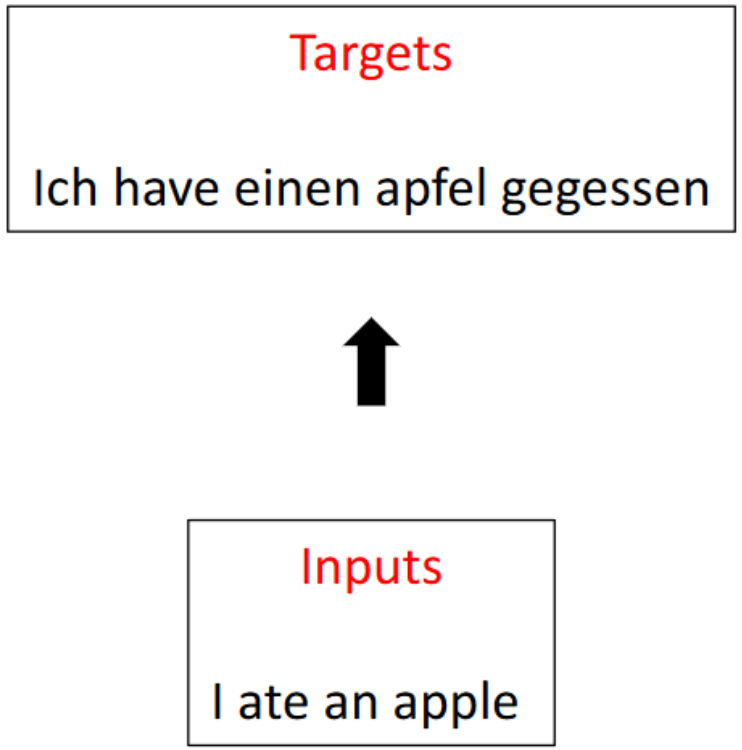
- Transformers were initially designed for **sequence transduction**
- Sequence transduction: converting an input sequence into an output sequence, potentially of different lengths



Key Points

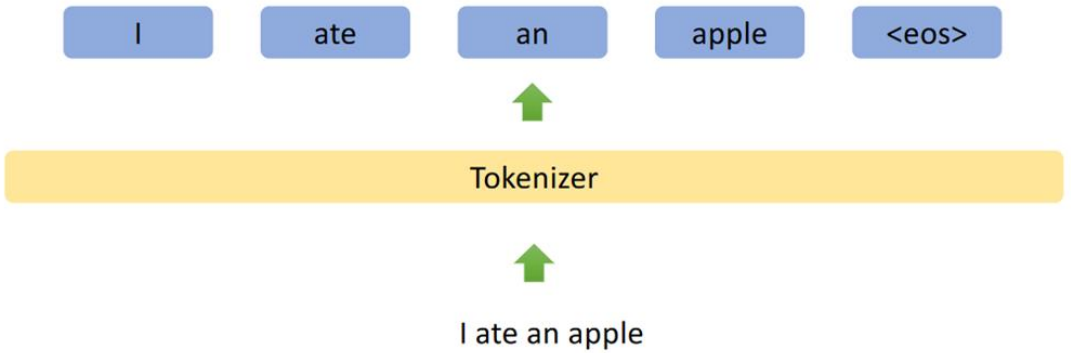
- **Non sequential**: sentences are processed as a whole rather than word by word.
- **Self Attention**: compute similarity scores between words in a sentence.
- **Positional embeddings**: The idea is to use fixed or learned weights which encode information related to a specific position of a token in a sentence.

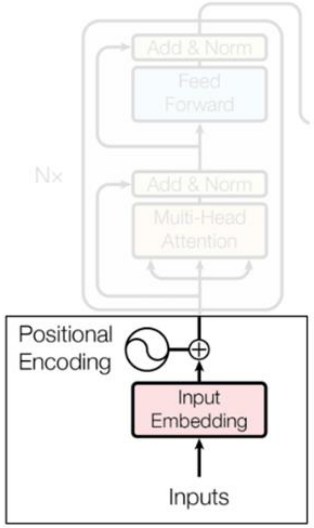
Language Translation



Tokenizer

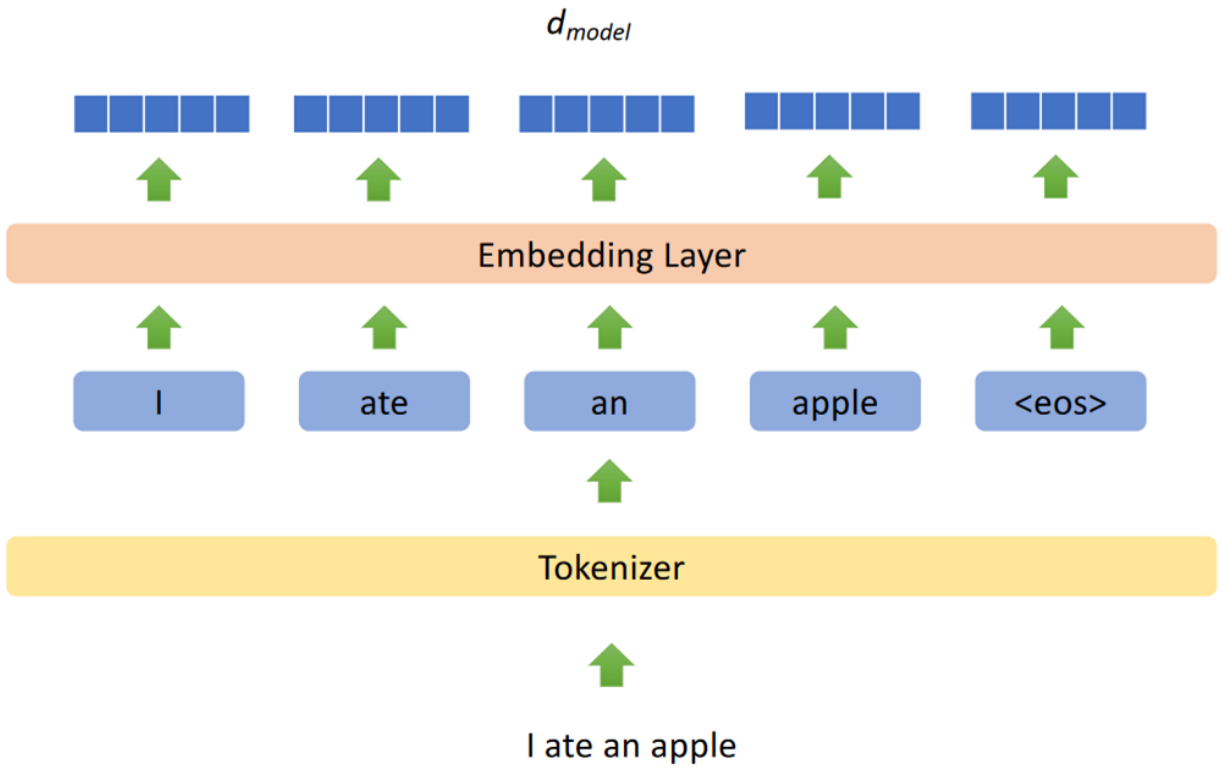
- The model doesn’t directly work on texts and we need to generate embeddings first
- Tokens are generated by considering word, letter, or characters

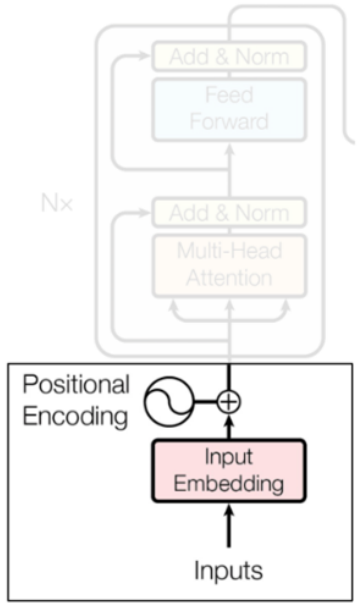




Generate Input Emebeddings

Generate embeddings



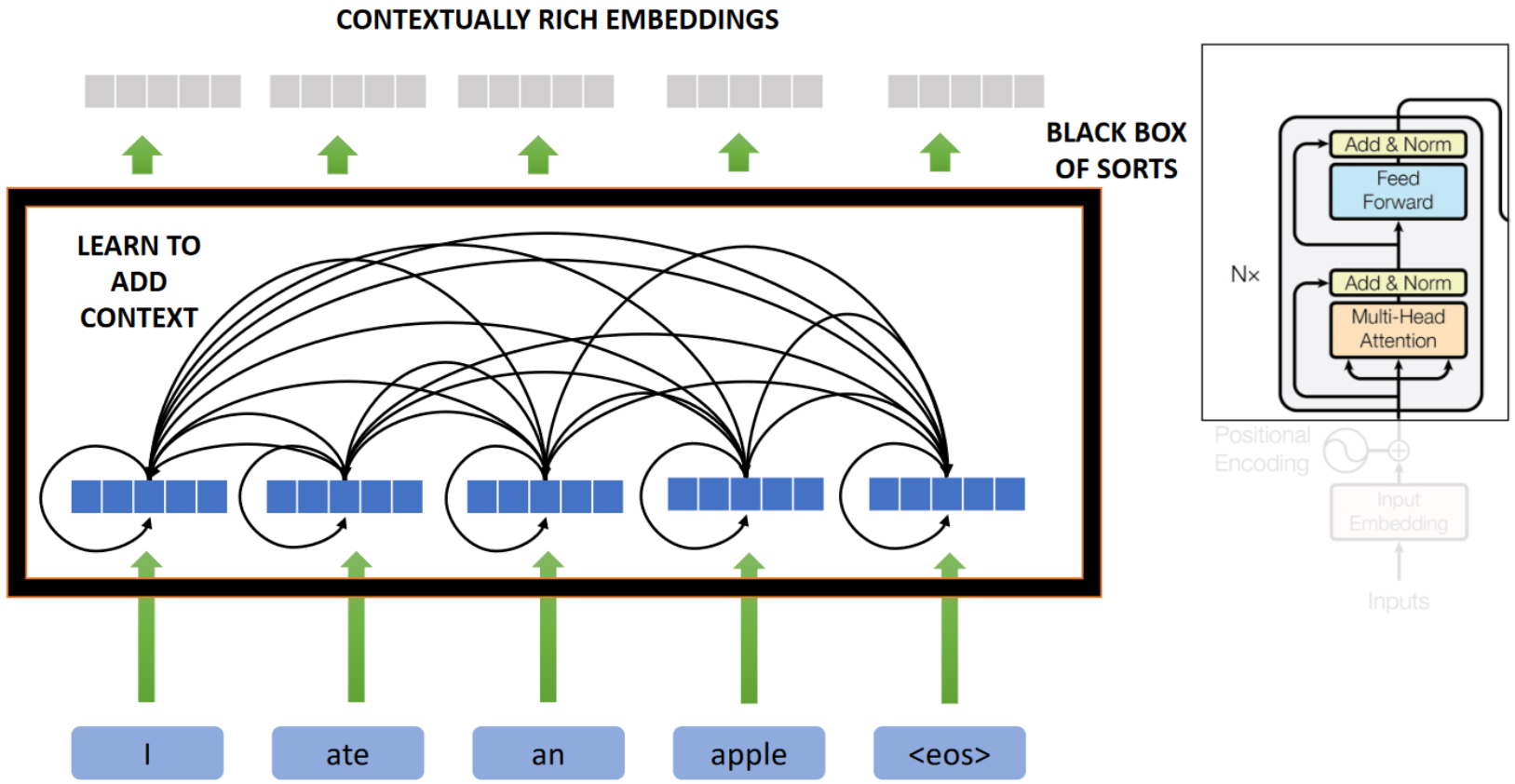


Generate Input Emebeddings

Generate embeddings

- Prepare dictionary from sentences. (As a pre-processing step)
  - S1: Apple is a fruit
  - S2: I ate an apple
  - S3: It was delicious
- Lets do word embedding so
- Input = {“Apple”, “is”, “a”, “fruit”, “I”, “ate”, “an”, “It”, “was”, “delicious”}
- Dictionary ={1,2,3,4,5,6,7,8,9,10}
- Embedding of S2 = {5,6,7,1}

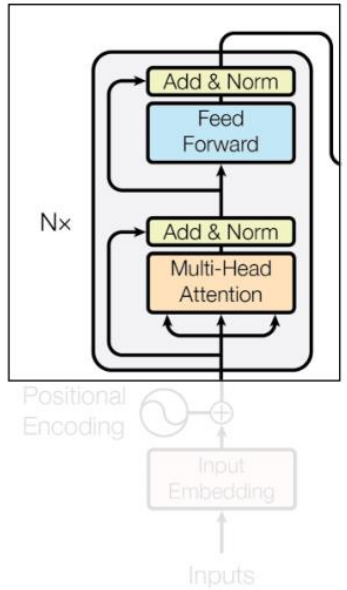
Adding context

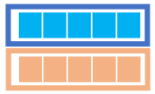


Attention

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

- Query
- Key
- Value



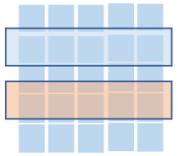


Query

Q

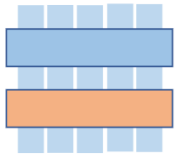
Key Value Store

QK^T



Key

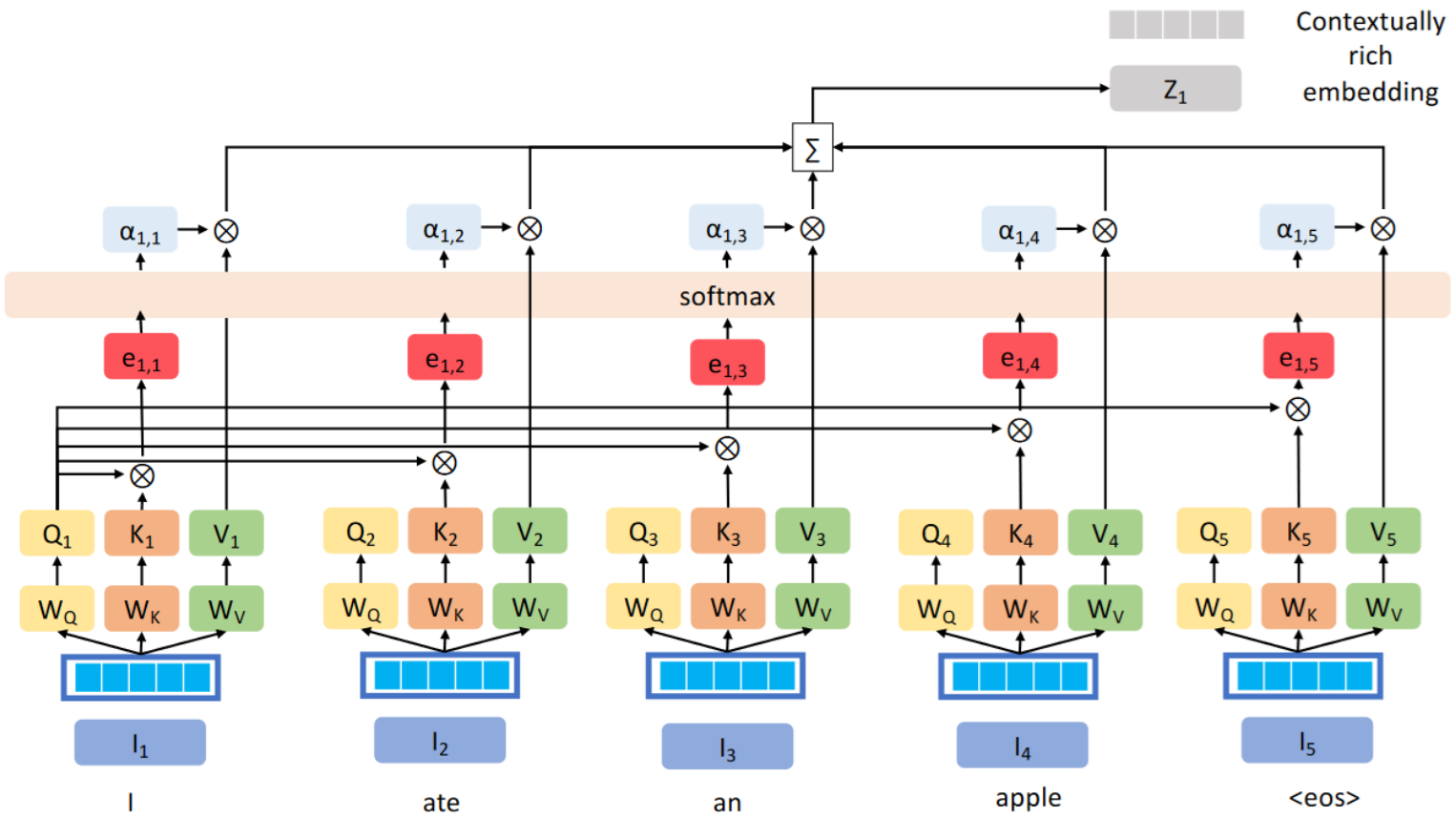
$\text{softmax}(\frac{QK^T}{\sqrt{d}})$



Value

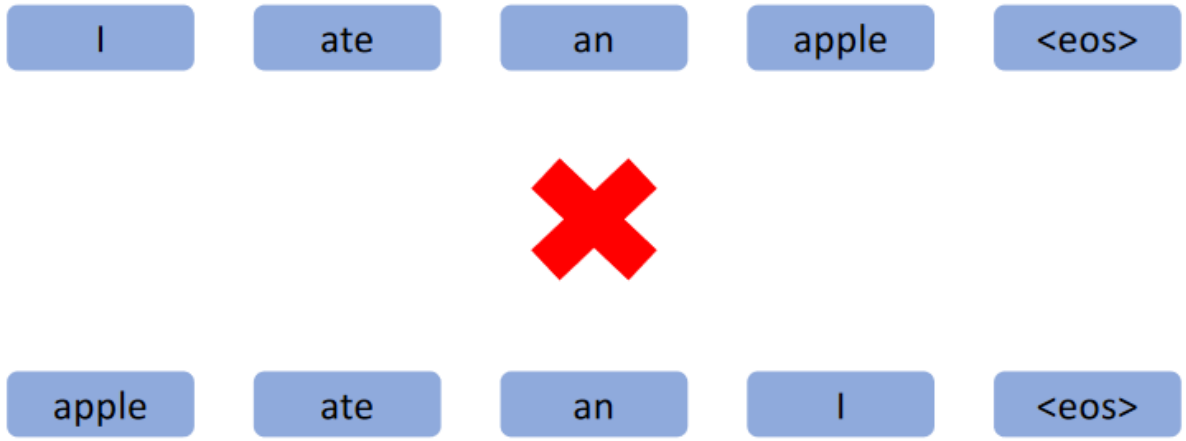
$\text{softmax}(\frac{QK^T}{\sqrt{d}})V$

Attention

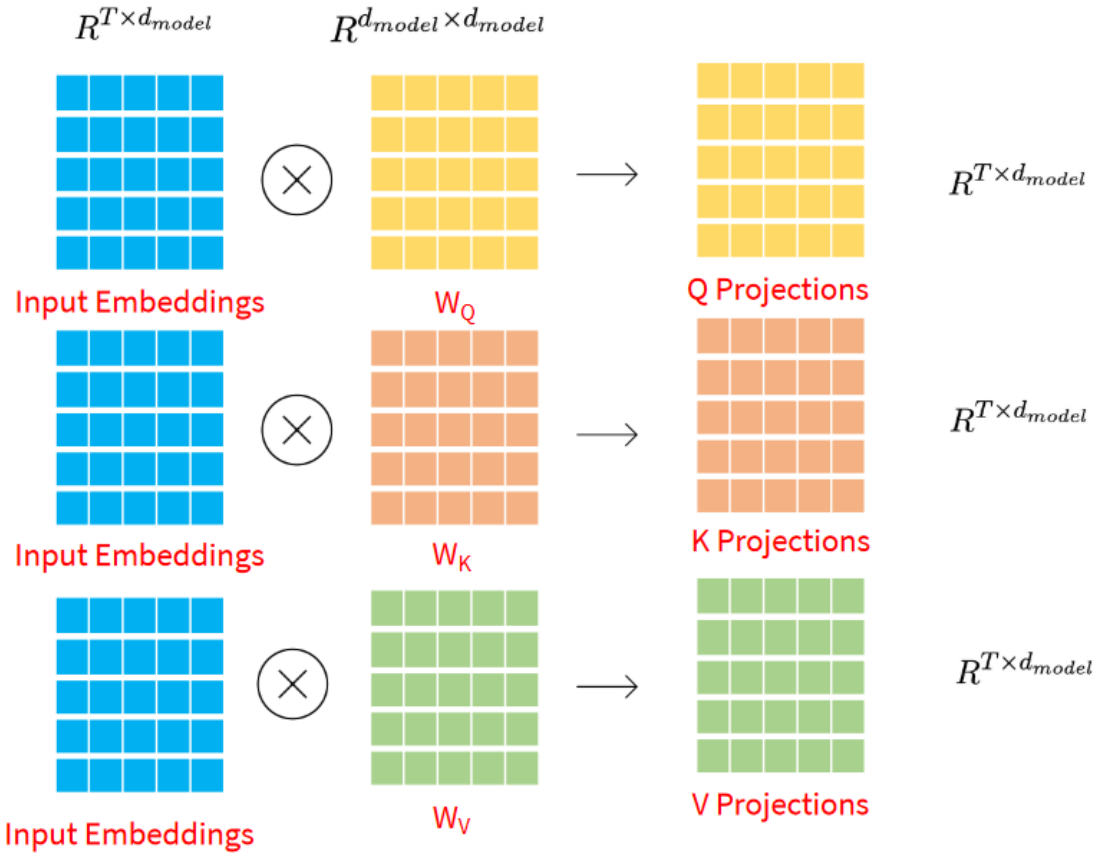

Contextually rich embedding
Z_1
Σ
$\alpha_{1,1}$
$\alpha_{1,2}$
$\alpha_{1,3}$
$\alpha_{1,4}$
$\alpha_{1,5}$
softmax
$e_{1,1}$
$e_{1,2}$
$e_{1,3}$
$e_{1,4}$
$e_{1,5}$
Q_1 K_1 V_1
Q_2 K_2 V_2
Q_3 K_3 V_3
Q_4 K_4 V_4
Q_5 K_5 V_5
W_Q W_K W_V
I_1 I_2 I_3 I_4 I_5
I
ate
an
apple
<eos>

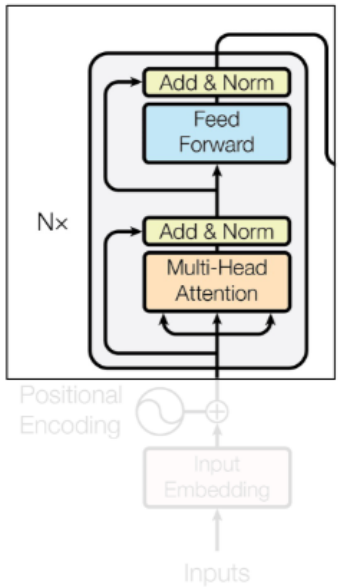
Positional embedding

- Positional embedding is necessary
- The meaning changes based on position of word
- Most commonly used : sine, cosine positional embeddings



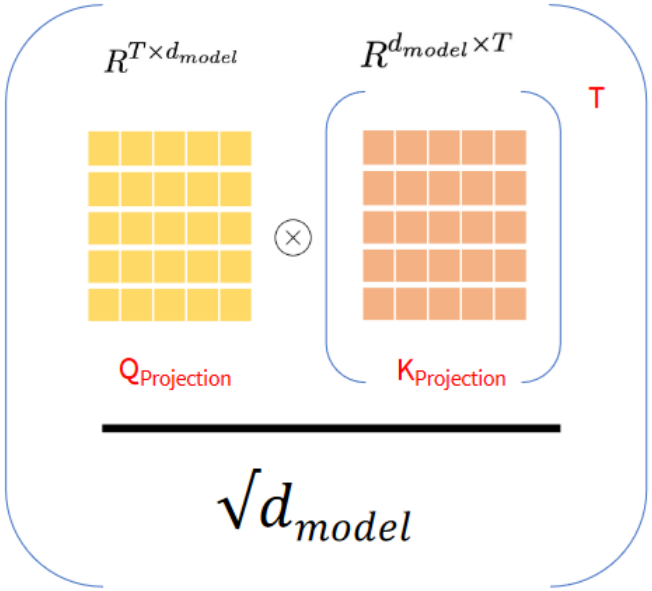
Self Attention



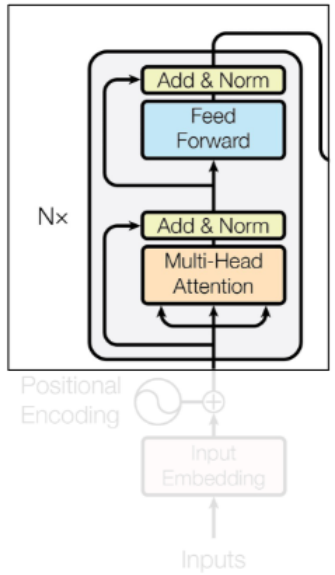


Self Attention

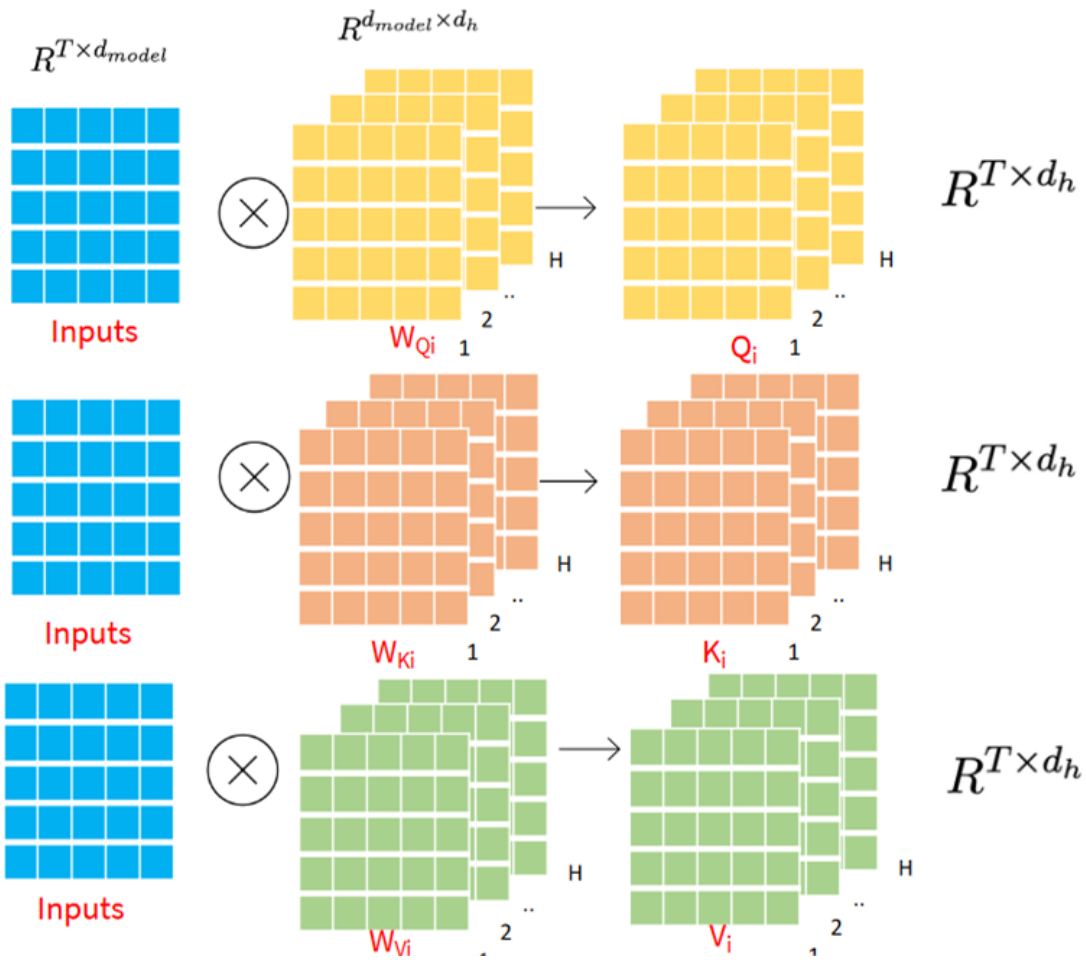
softmax

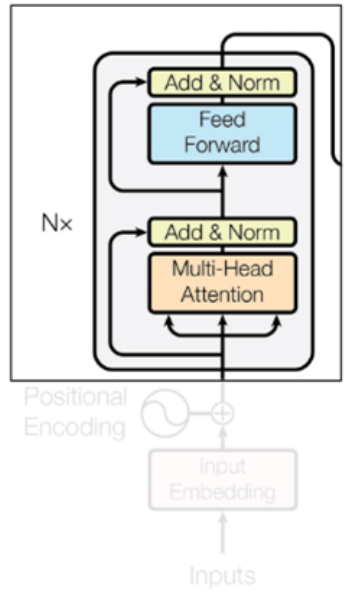


$R^{T \times T}$



Multi-head Attention



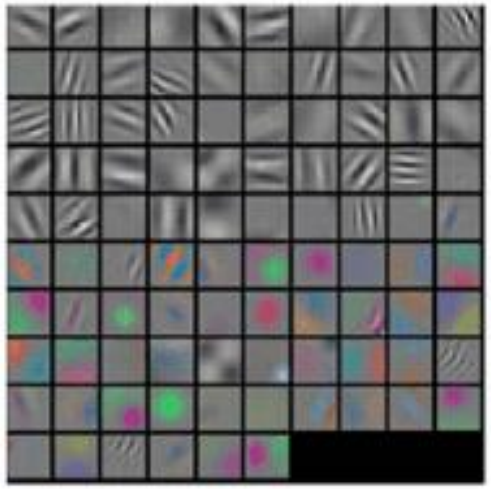


Vision Transformer

CNN vs ViT

well-trained networks often show nice and **smooth** filters.

Alexnet 1st conv filters



ViT 1st linear embedding filters