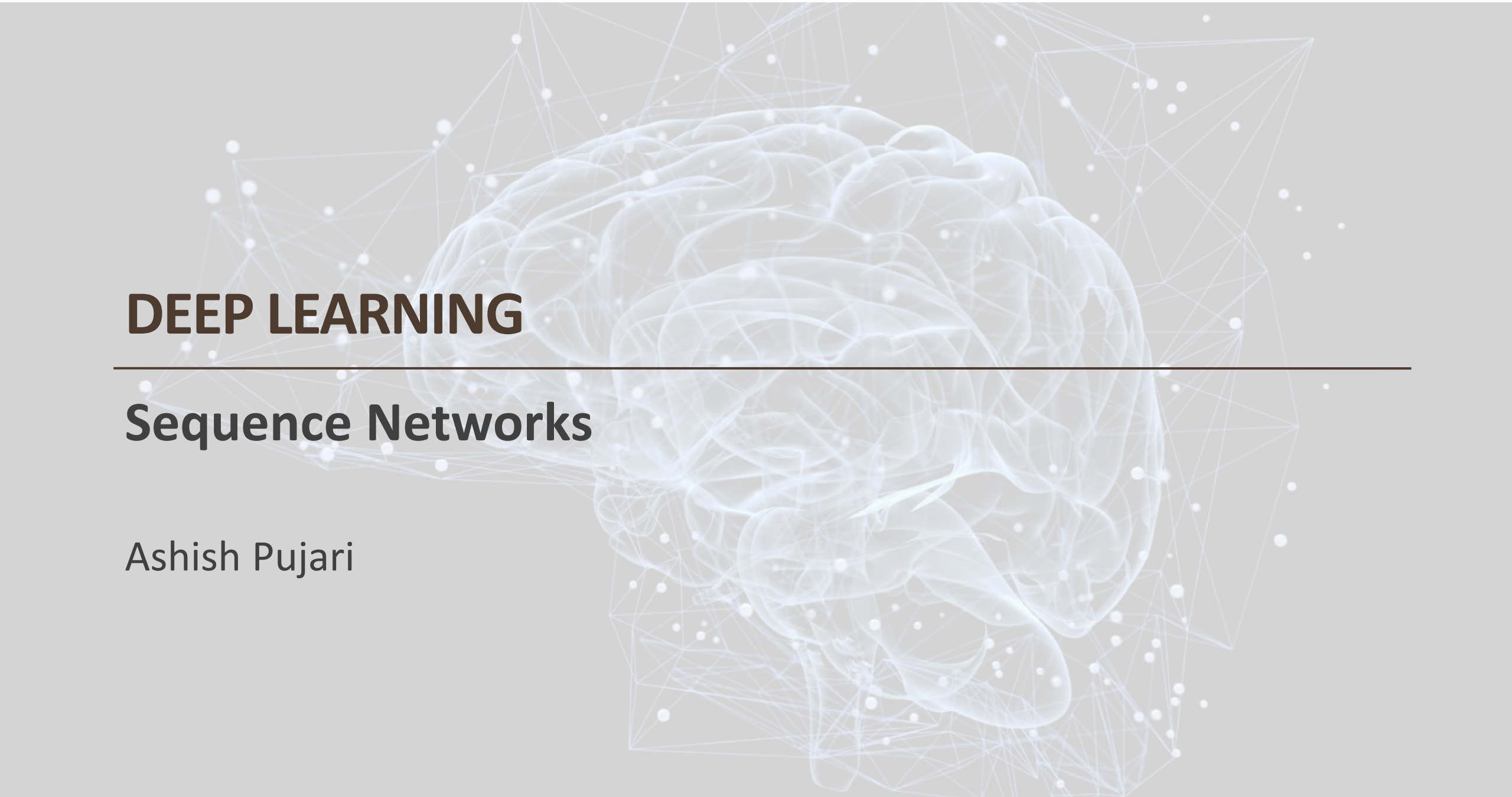


DEEP LEARNING

Sequence Networks

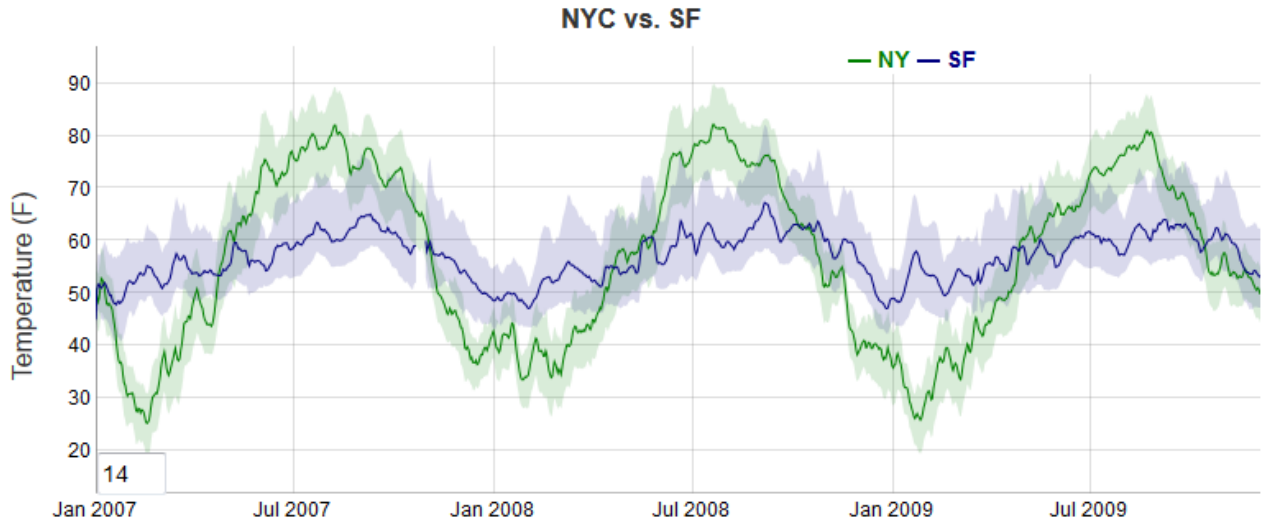
Ashish Pujari

Lecture Outline

1. Sequential Data
2. Recurrent Neural Networks (RNN)
3. RNN Architectures
4. Attention

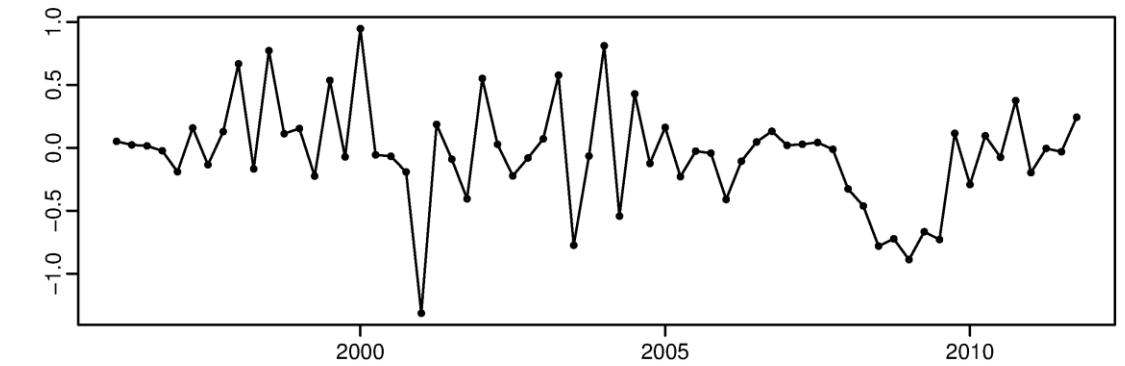
Sequential Data

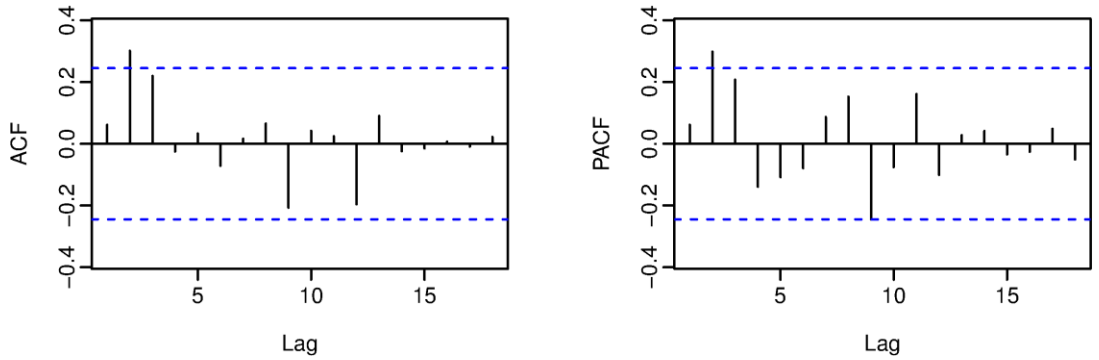
- Sequential data is not IID; the current state is informed by past states
- Length of the data varies from sample to sample
- E.g., Time series - Stocks, Weather, Natural Language, etc.



Traditional Sequential Models: Limitations

- Require a series of hyperparameters which must be calculated based on data (ARIMA, ETS, GARCH, etc.)
- Suffer from dimensionality issues and outliers
- Probabilistic models, sliding windows have other limitations





ACF shows serial correlation in data that changes over time

Natural Language Processing (NLP)

- Part-of-speech (POS) tagging
- Lemmatization & Stemming
- Named Entity Recognition (NER)
- Text Classification
- Sentiment Analysis
- Character/Word Prediction
- Language Translation
- Summarization
- Question Answering (Q&A)
- Information Retrieval
- Natural Language Generation (NLG)

NLP Challenges

- Variable length sequences
- Maintain sequence order
- Keep track of long-term dependencies
- Share parameters across a sequence

Traditional NLP Models - Limitations

- Treat words as discrete atomic symbols (tokens)
- Representing words as unique, discrete ids leads to data sparsity
- Arbitrary encodings provide no semantic information

- unigram (1-gram):

a	swimmer	likes	swimming	thus	he	swims

- bigram (2-gram):

a swimmer	swimmer likes	likes swimming	swimming thus	...

- trigram (3-gram):

a swimmer likes	swimmer likes swimming	likes swimming thus	...

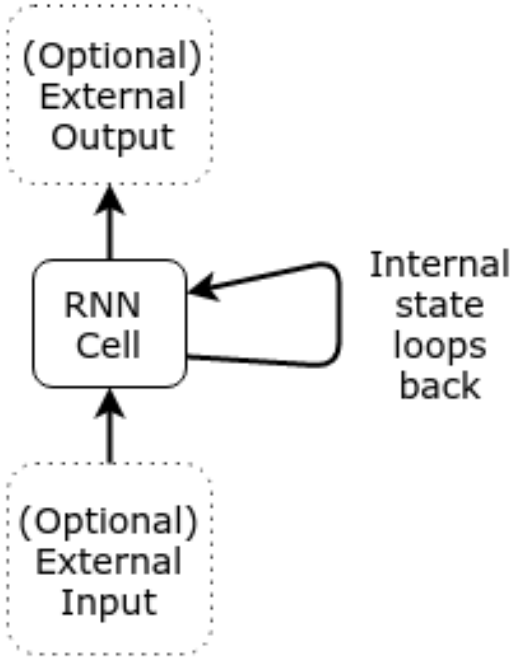
	and	beautiful	blue	is	king	love	old	queen	sky	the	this
0	1	1	1	1	0	0	0	0	1	1	0
1	1	1	0	2	1	0	1	1	0	2	0
2	0	1	1	0	0	1	0	0	1	0	1
3	1	1	0	0	1	0	1	1	0	2	0

RECURRENT NEURAL NETWORKS

Introduction

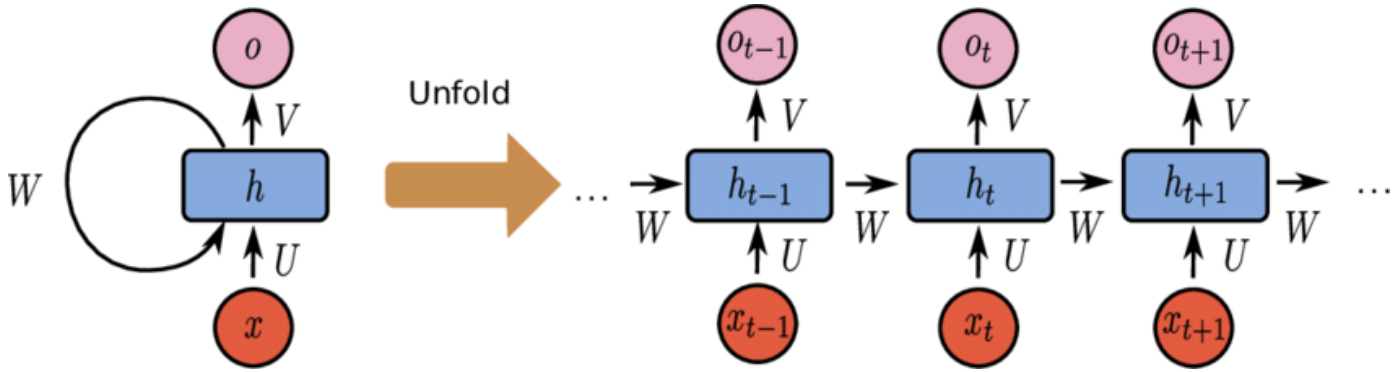
Recurrent Neural Networks (RNN)

- Neural networks with loops that allow the modeling of temporal or spatial sequences.
- Takes information from the previous time step and adds it to the input of the current time step.



RNN: Design

$$h_t = \phi(Ux_t + Wh_{t-1})$$

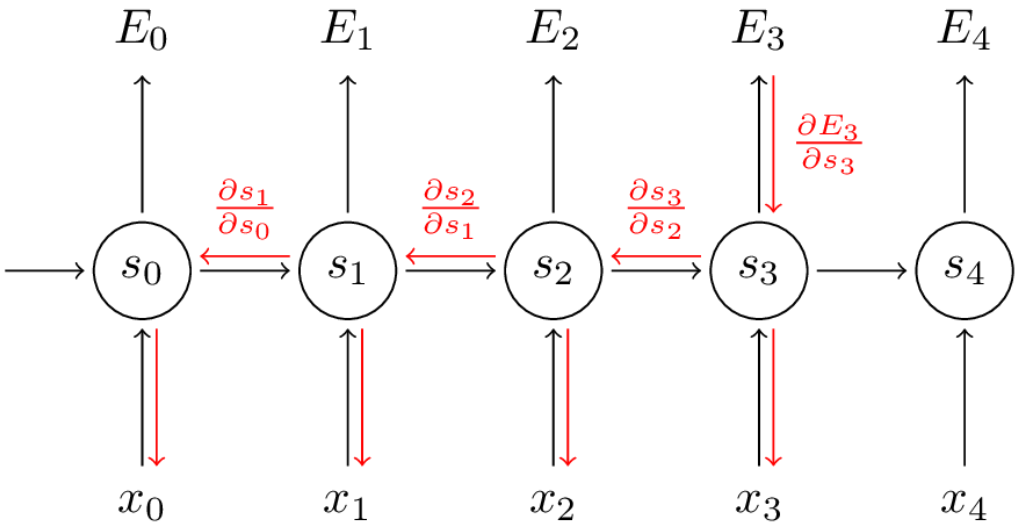


x_t	Input at time t
h_t	Hidden State at time t
h_{t-1}	Hidden State at time t-1
O	Output at time t
U	Weight Matrix for Input to Hidden layer at time t
W	Weight Matrix for Hidden layer at t-1 to Hidden layer at t
V	Weight Matrix for Hidden layer to the output
Ø	Activation Function (Sigmoid or Tanh)

RNN: Training

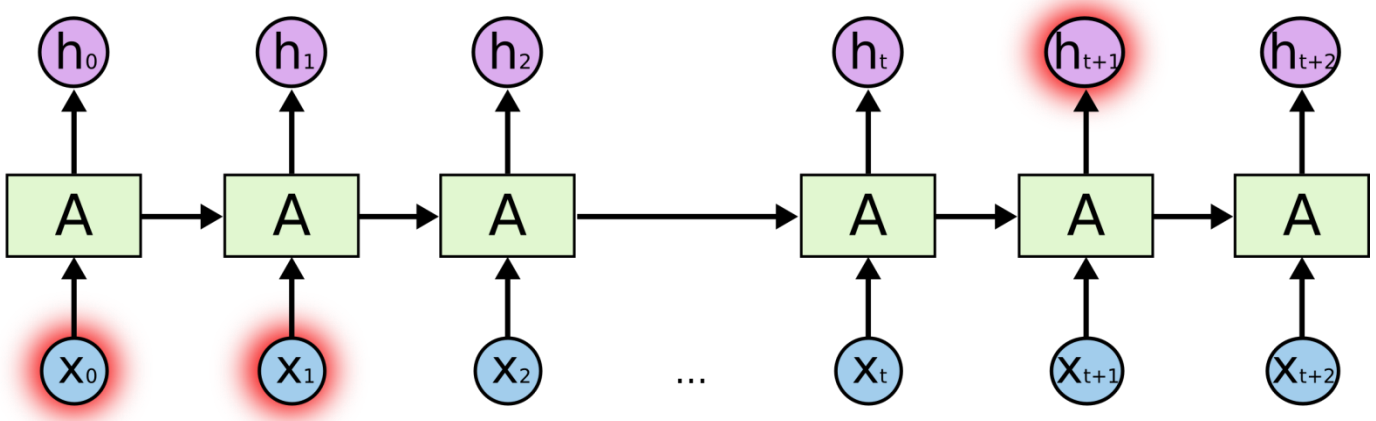
- Parameters are shared across RNN cells, so we need to calculate the gradient with respect to each time step separately and then add them up
- Backpropagation Through Time (BPTT)
  1. Present a sequence of timesteps of input and output pairs to the network.
  2. Unroll the network then calculate and accumulate errors across each timestep.
  3. Roll-up the network and update weights.
  4. Repeat.

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W}$$



Long-term Dependencies

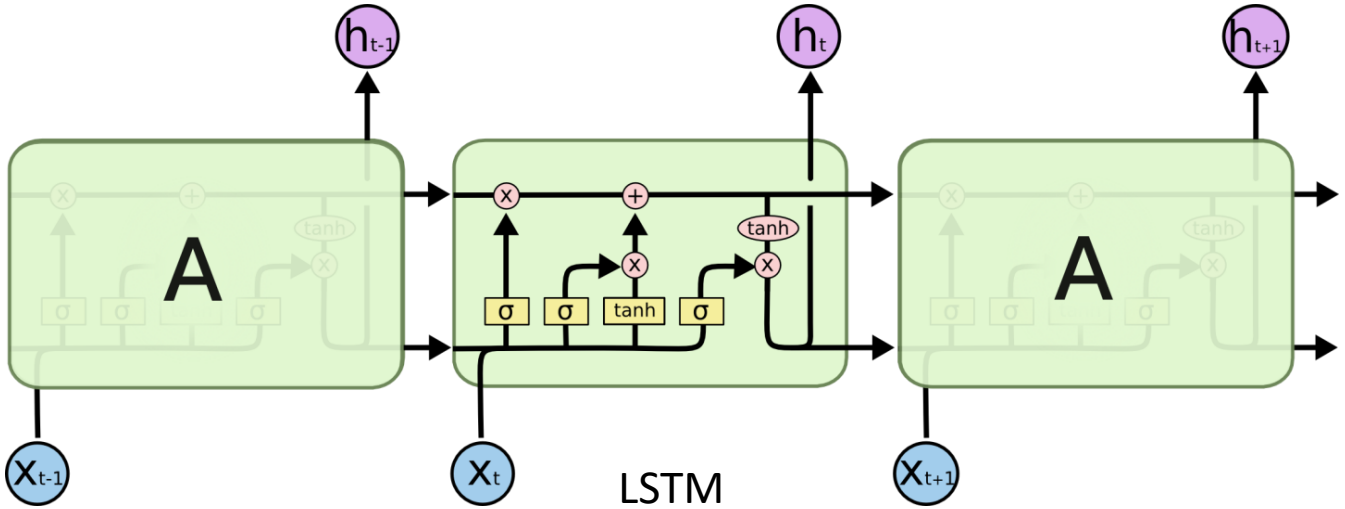
- The challenge of long-term dependencies
  - "The *clouds* are in the *sky*"
  - "I had a great time in *Mexico*… I learned a few words in *Spanish*."
- Unstable for more time-steps
  - Vanishing gradients and exploding gradients
  - Learning not feasible - faded or large signals



http://colah.github.io

LSTM (Long Short-Term Memory)

- RNN capable of learning long-term dependencies:
  - Selectively chooses what it remembers
  - Selectively decides to forget
  - Selects how much of it's memory it should output

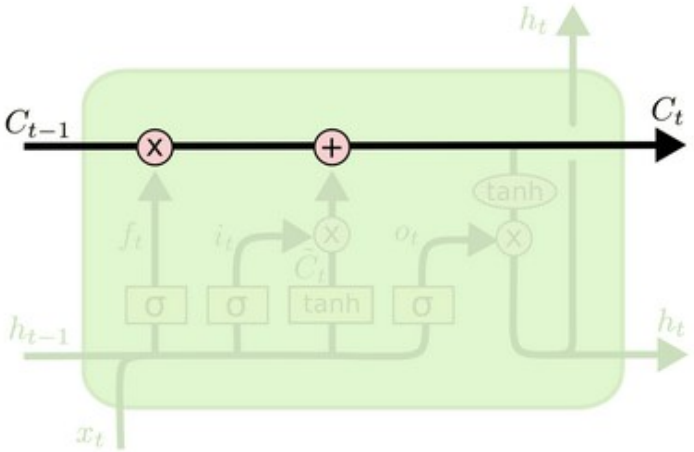


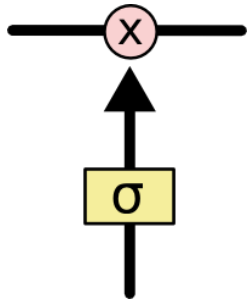
LSTM

http://colah.github.io

LSTM: Cell State

- Horizontal line running through the top of the diagram is the cell state
- LSTM has the ability to remove or add information to the cell state, carefully regulated by structures called gates





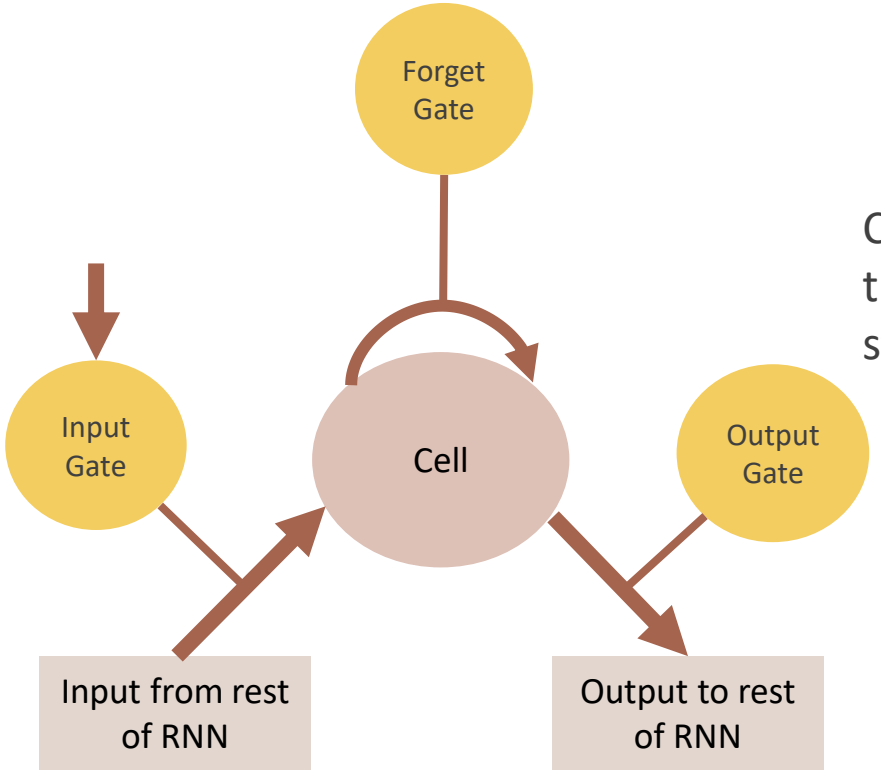
Gate

http://colah.github.io

LSTM: Gates

Allows the model to learn when to clear (or partially clear) the contents of the cell state.

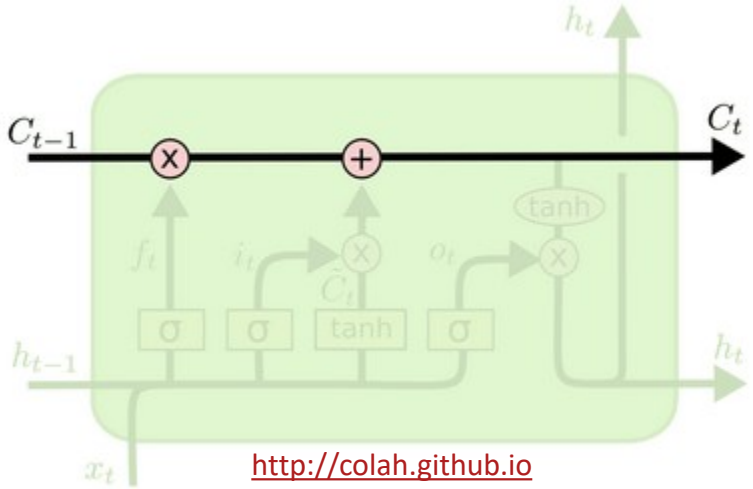
Controls how much new information should be added to cell state



Controls when to use the contents in the cell state for producing h(t)

LSTM: Variables

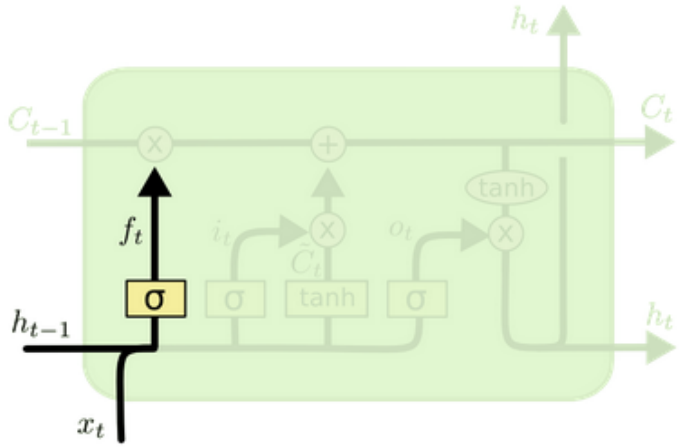
- Internal State (C_t)
  - internal state/center most node feeds into itself with a fixed weight of 1 across time steps
- Input (x_t)
  - network input at time step t
- Output (h_t)
  - network output at time step t
- Input gate (i_t)
  - when to let activation pass into memory cell
- Forget gate (f_t)
  - when to flush internal state
- Output gate(o_t)
  - when to let activation pass out of memory cell
  - controls when to use the contents in the cell state for producing h(t)



http://colah.github.io

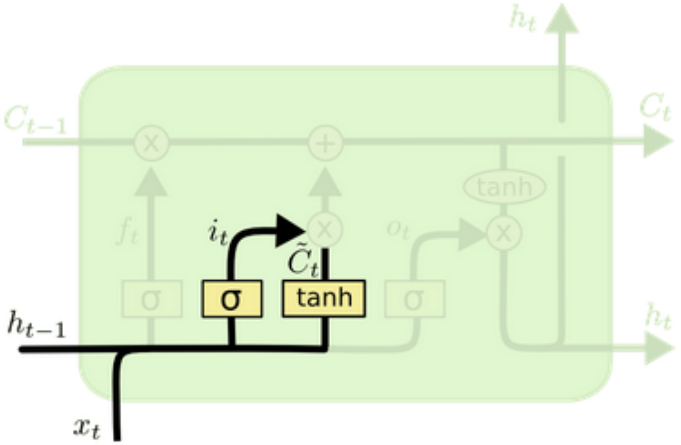
LSTM: Steps 1,2

1. **Forget gate**



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

2. **Decide State Update**



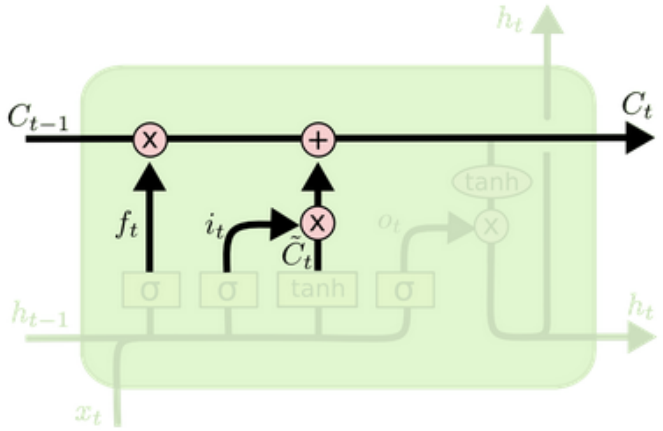
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

http://colah.github.io

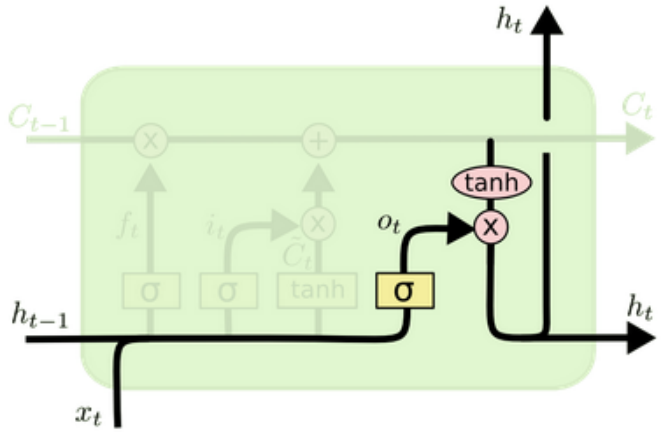
LSTM: Steps 3,4

3. Update



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

4. Decide Output



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

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

http://colah.github.io

LSTM: Parameters

- Input vectors of size m
- Output vectors of size n
- 2 matrices:
  - Different matrices U and W for each of the (3) gates.
  - U has dimensions $n \times m$
  - W has dimensions $n \times n$
  - Another set of these matrices for updating the cell state S
  - Total # parameters = $4(nm + n^2)$
  - Total # parameters with bias = $4(nm + n^2 + n)$

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

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

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$

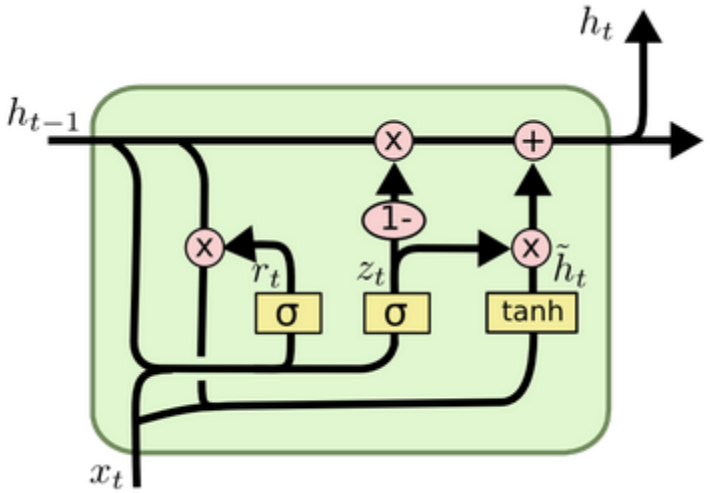
$$g_t = tanh(W_c x_t + U_c h_{t-1} + b_c)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ g_t$$

$$h_t = o_t \circ \sigma(c_t)$$

Gated Recurrent Unit (GRU)

- Simpler than the standard LSTM
- GRU combines the forget and input gates into a single “update gate”
- It also merges the cell state and hidden state, and makes some other changes



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

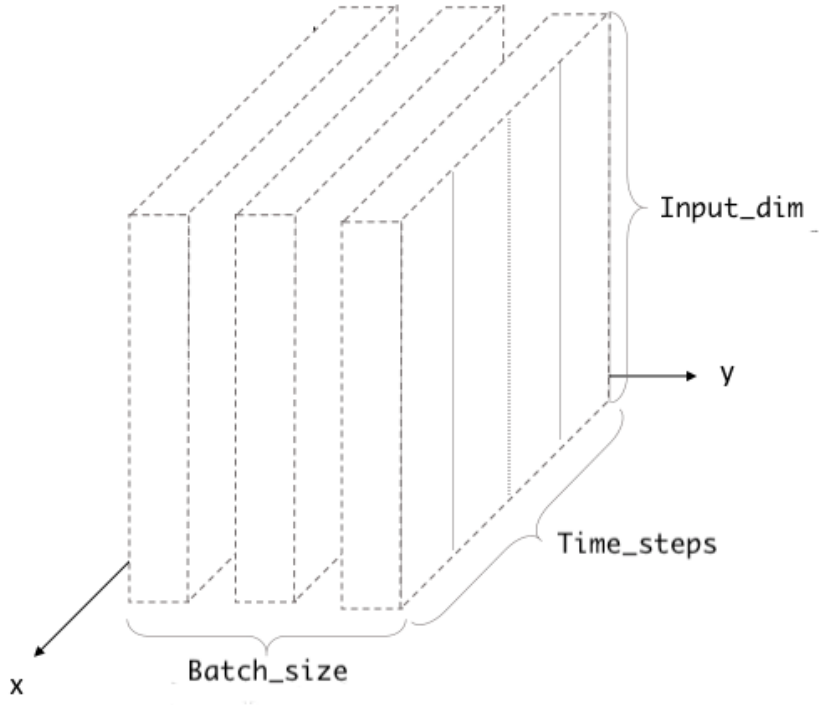
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

http://colah.github.io

LSTM in Keras

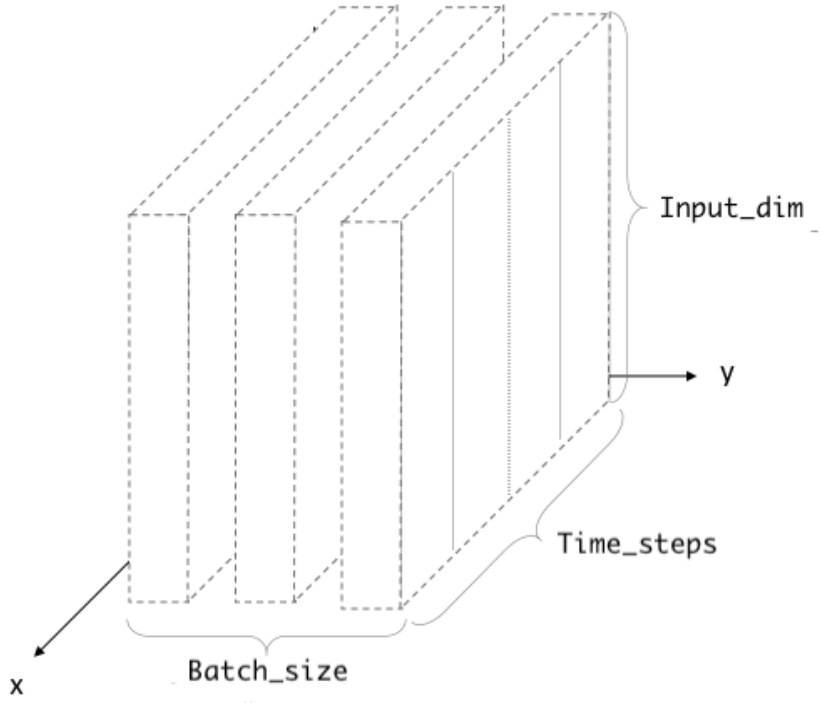
- Input Parameters
  - Take in a three-dimensional array as an input.
- Statefulness
  - If layers are set to 'stateful', then the states computed for the samples in one batch will be reused as initial states for the samples in the next batch.
  - This assumes a one-to-one mapping between samples in different successive batches.
- Parameters
  - $m: size\ of\ input\ ,\ n: size\ of\ output$
  - $parameters\ =\ 4(nm + n^2 + n)$
    - 4 since there are different weight and bias variables for the 3 gates (read / write / forget) and 4th for the cell state.



LSTM in Keras

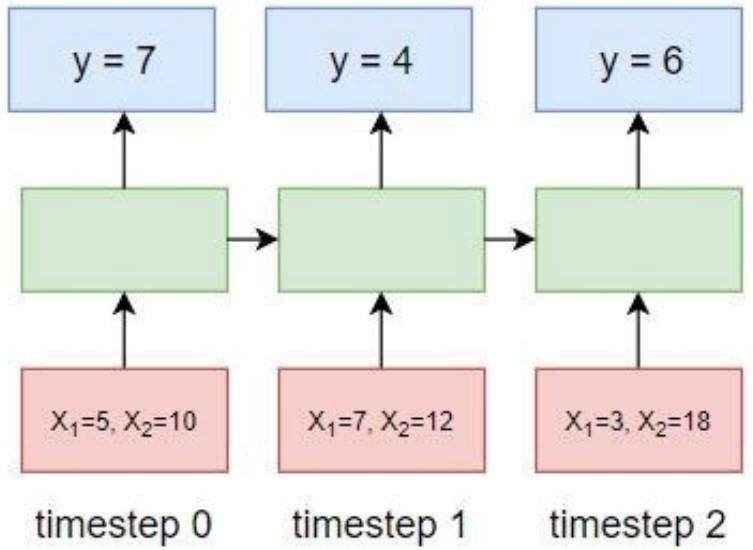
```
model = Sequential()
model.add ( LSTM(units=1, batch_input_shape=(None, 5,1), return_sequences=False))
```

- Units
  - The number of output units in the LSTM
- Batch Input Shape
  - First dimension : **batch size**
  - Second dimension : number of **time-steps**
  - Third dimension : number of **units** in one input sequence.
- Return Sequences
  - Whether to return the output at each **time step** instead of the final time step



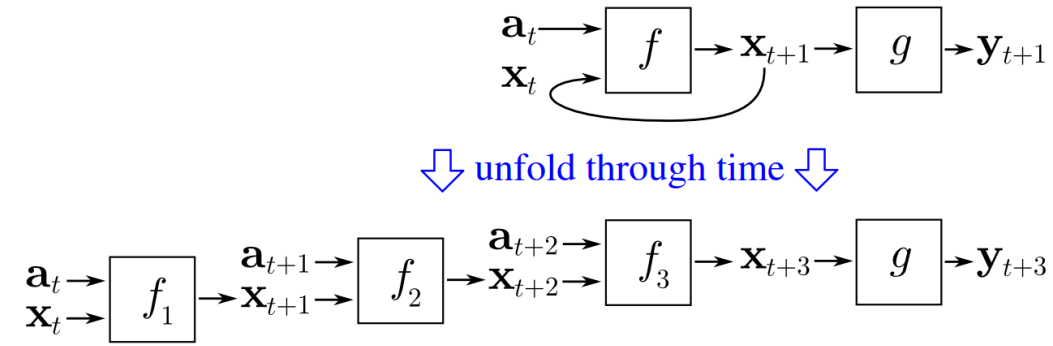
LSTM: Multivariate Timeseries

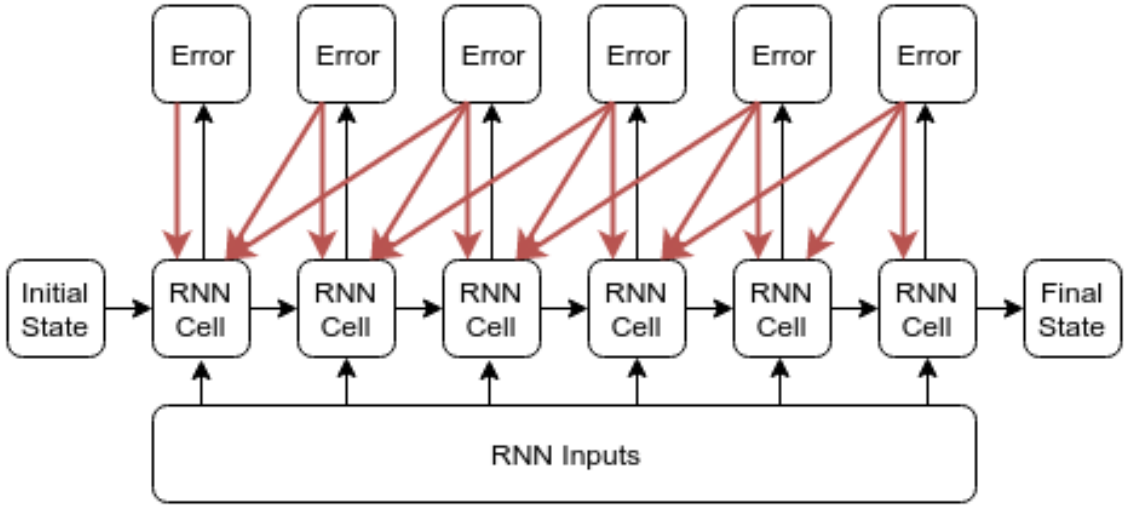
	features		target
timesteps	X_1	X_2	y
timestep 0	5	10	7
timestep 1	7	12	4
timestep 2	3	18	6
timestep 3	8	16	9
timestep 4	2	11	2
timestep 5	9	21	1



Truncated Back Propagation Through Time (TBPTT)

- Issues with BPTT
  - Expensive to backpropagate the error through many steps
  - Vanishing gradients
- Solution
  - Naive method that splits a long sequence M into smaller sequences each of length m and treats each split as a separate training case.
  - Works well in practice, but it is blind to temporal dependencies that span more than m timesteps.





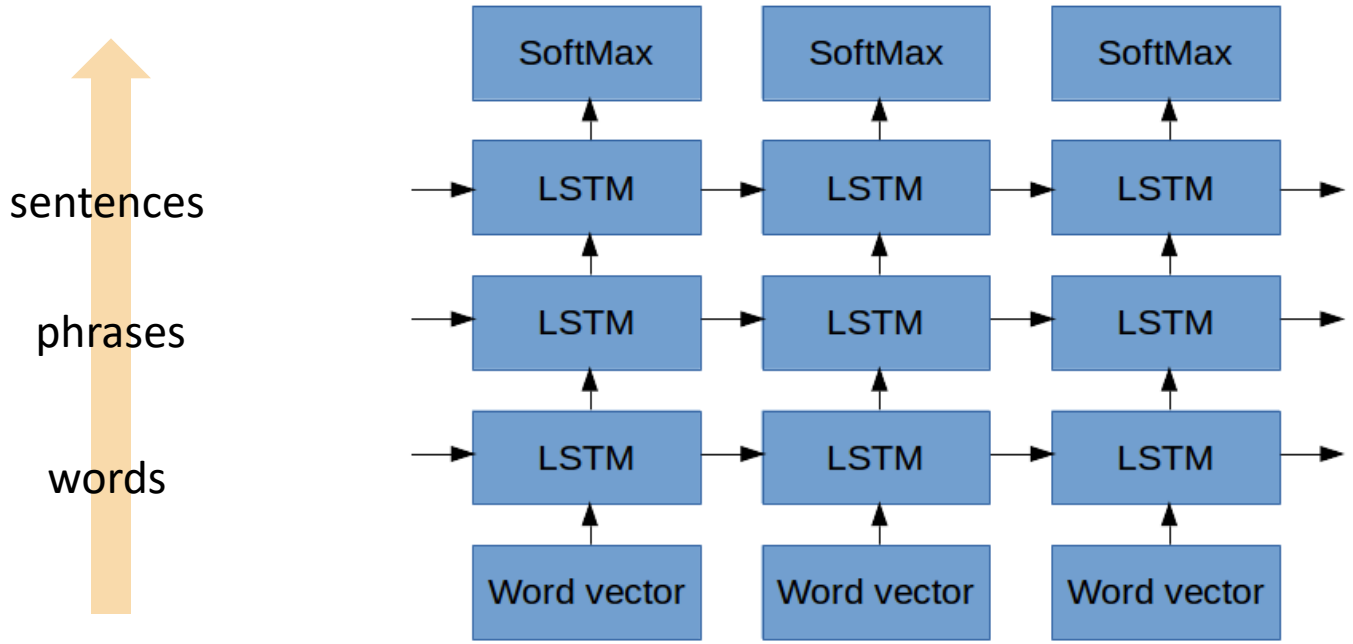
https://r2rt.com/styles-of-truncated-backpropagation.html

RNN ARCHITECTURES

Stacked, Bidirectional RNNs

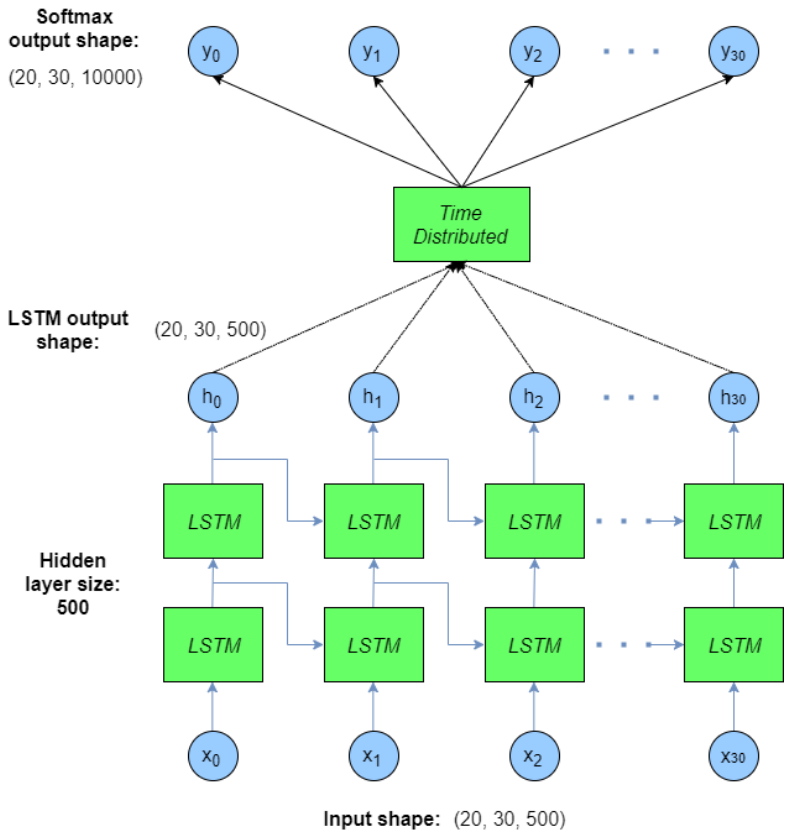
Stacked LSTM

- LSTMs learn more abstract concepts by stacking them on top of each other



Time-Distributed Layer

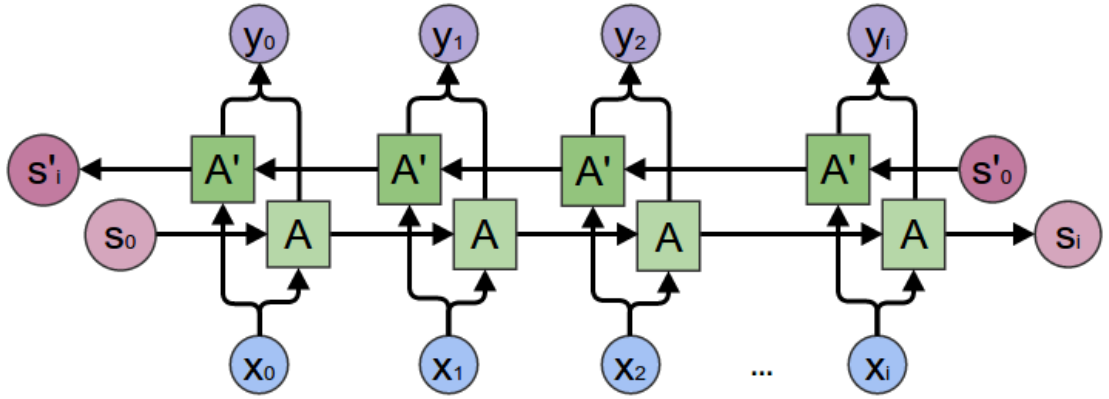
- A wrapper that allows to apply a layer to every temporal slice of an input.

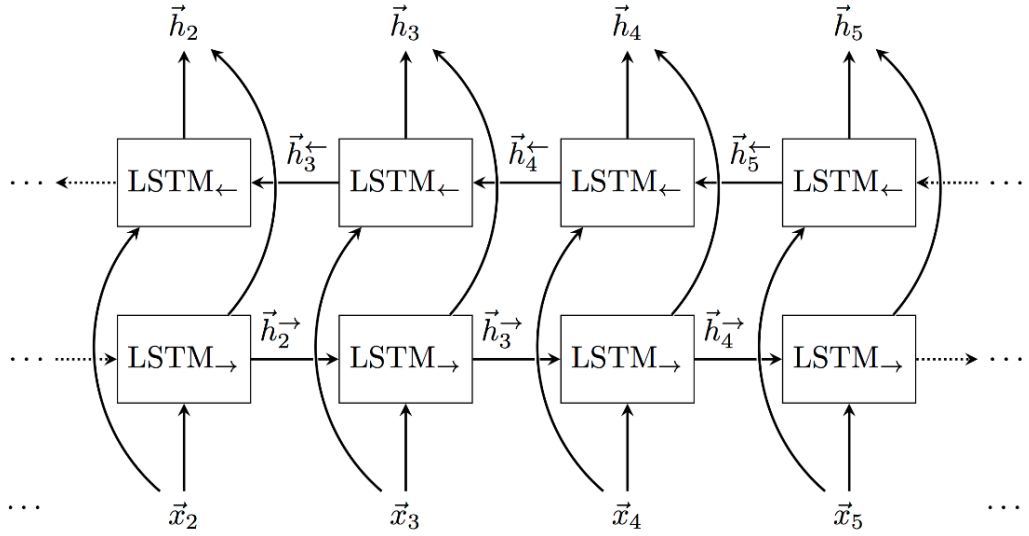


(batch size, number of time steps, hidden size)

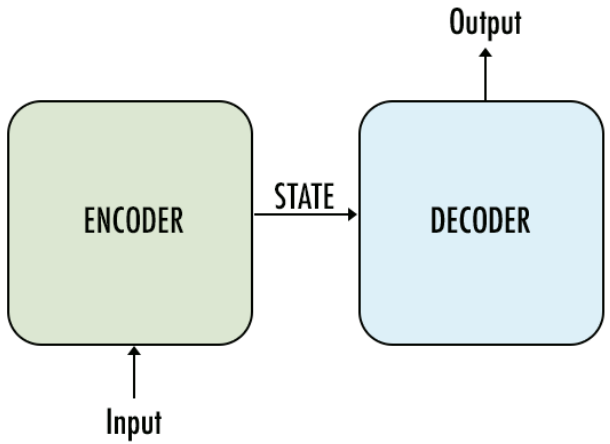
Bidirectional RNN

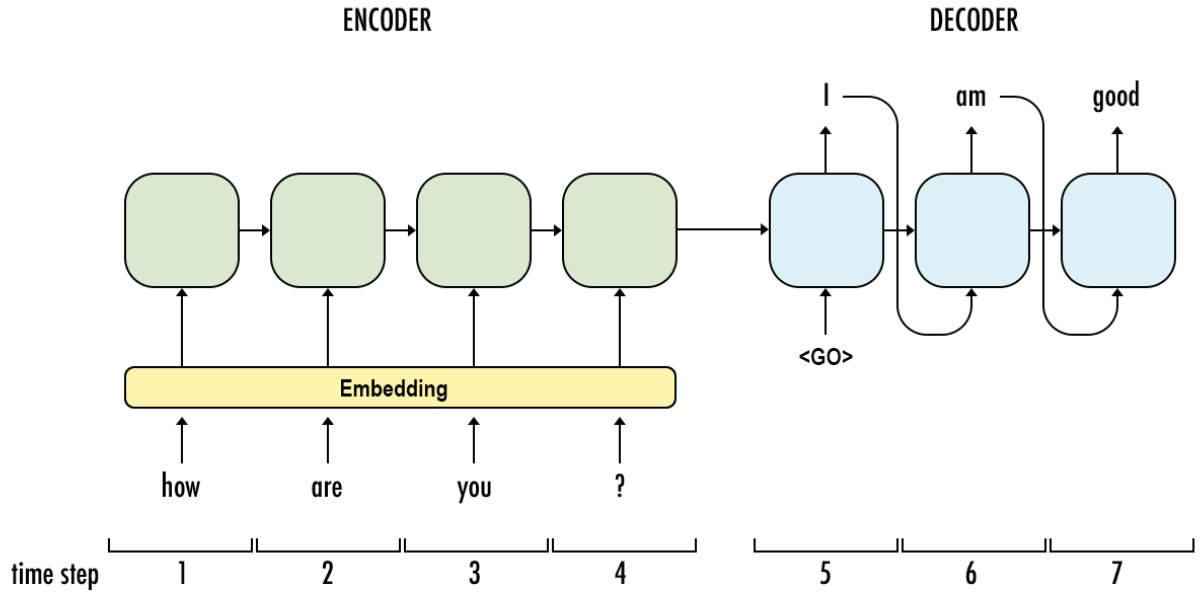
- Allows the networks to have both backward and forward information about the sequence at every time step (to exploit both directions of temporal dependence)





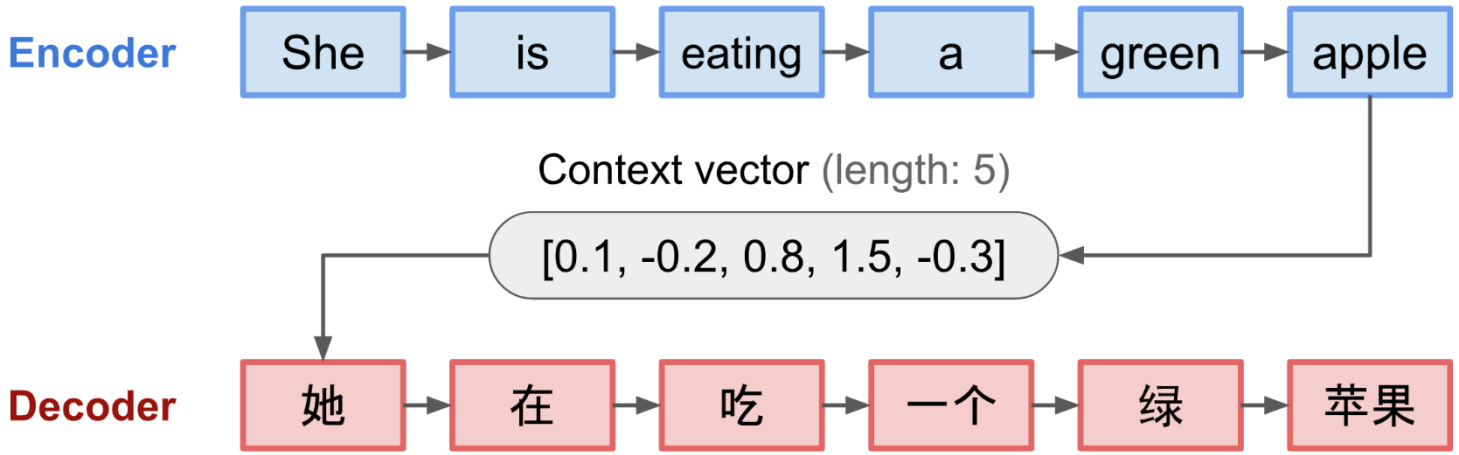
Seq2Seq Models





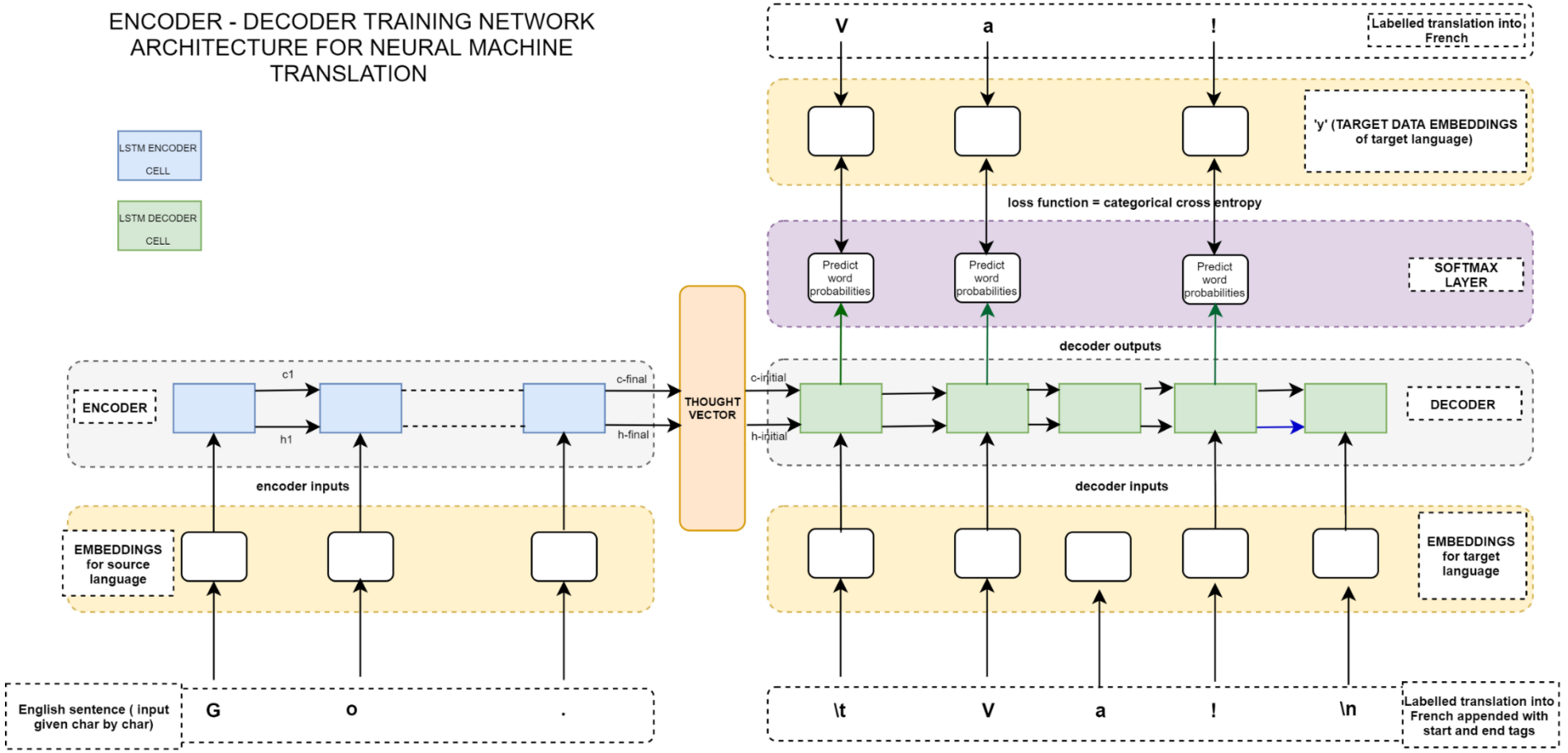
https://github.com/tensorflow/nmt

Seq2Seq: Neural Machine Translation (NMT)



https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

Seq2Seq: Neural Machine Translation (NMT)



NMT systems map the meaning of a sentence into a **fixed-length vector** representation and then generate a translation based on that vector

Seq2Seq Pros/Cons

Pros

- NMT systems based on Seq2Seq generalize to new sentences better than many other approaches (n-gram counts)
- Lengths of input and output sequences can be different - no explicit one on one relation between the input and output sequences.
- NTM systems are much easier to build and train, and do not require feature engineering

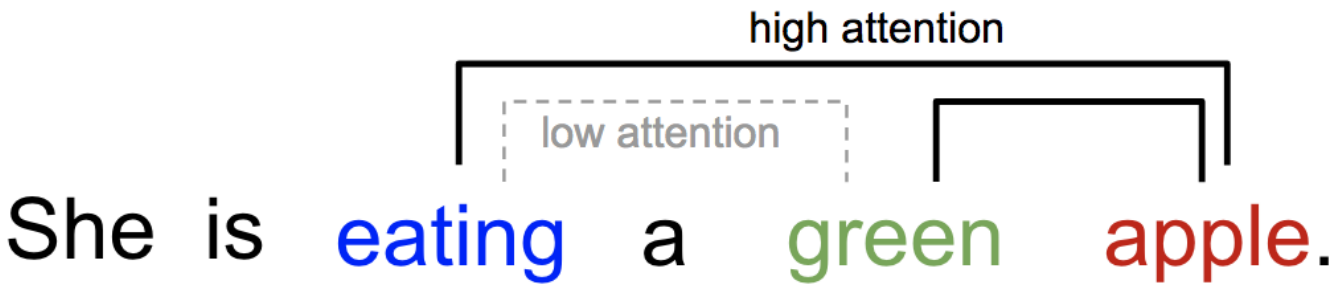
Cons

- Network must compress all the necessary information of a source sentence into a fixed-length vector; poor performance with long sentences
- No way to emphasize inputs (e.g., words, parts of a picture) that are more important to the prediction.

ATTENTION

Attention Mechanism

- Refer back to the input sequence rather than encode all information into a fixed-length vector.
- Improves translation of longer sentences, image caption, speech recognition etc.

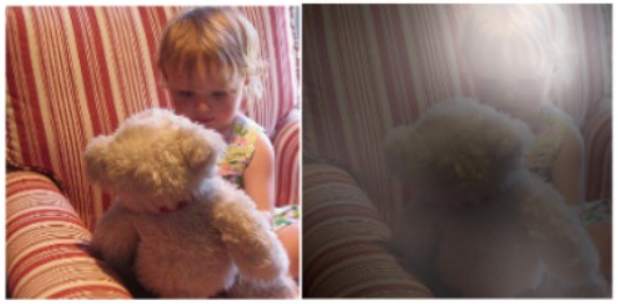


Visual Attention

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.



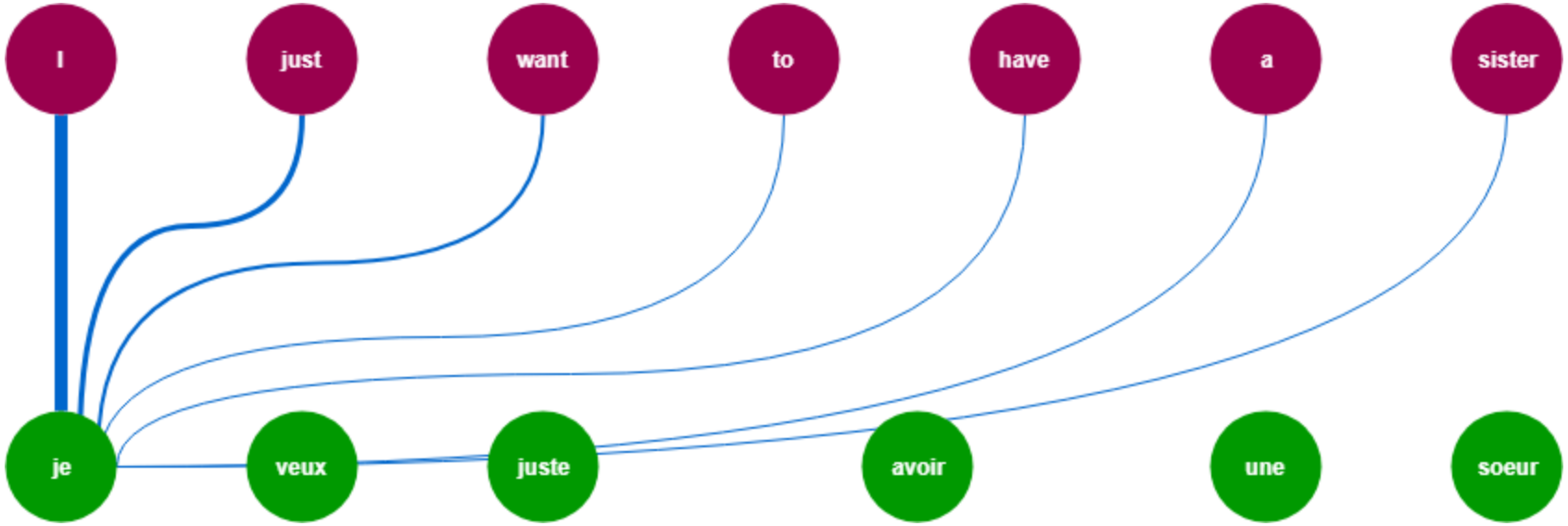
A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

Human visual attention allows us to focus on a certain region with “high resolution”

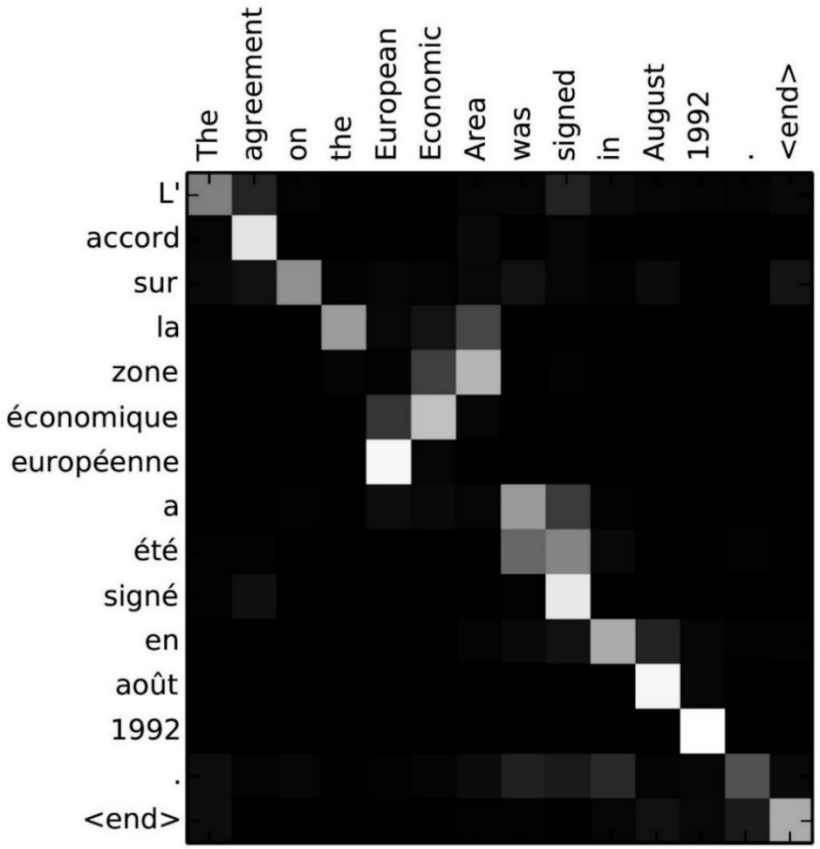
Language Translation



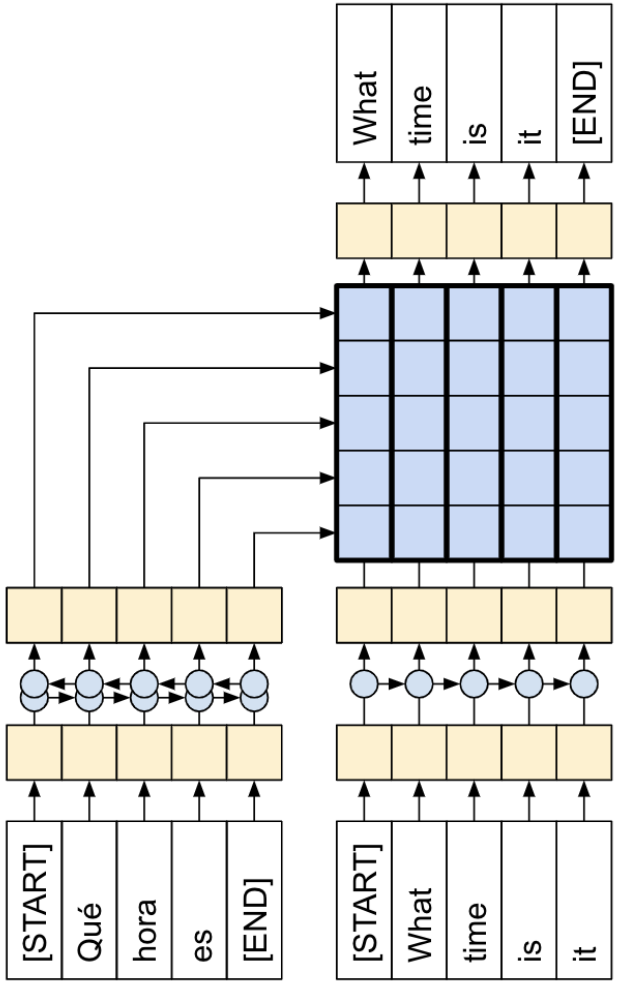
The transparency of the blue link represents how much the decoder gives attention to a coded word.

https://machinetalk.org/2019/03/29/neural-machine-translation-with-attention-mechanism/

Language Translation

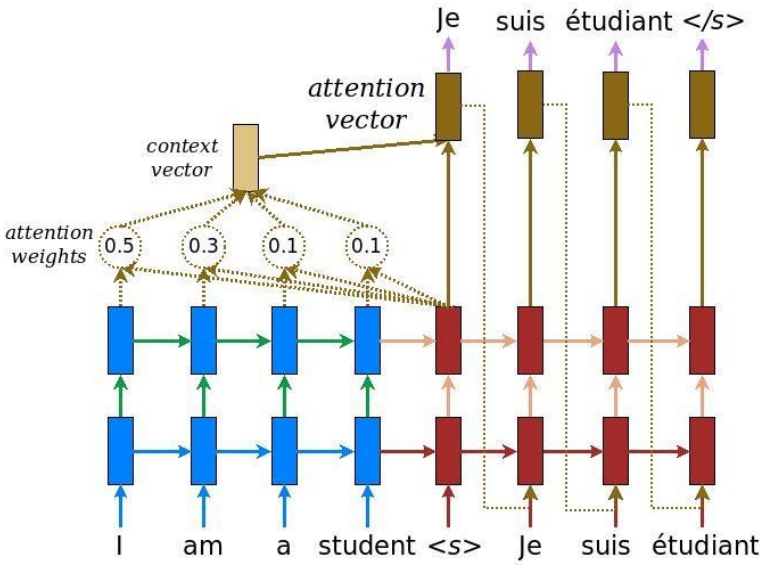


Attention visualization – example of the alignments between source and target sentences



https://www.tensorflow.org/text/tutorials/nmt_with_attention

Attention Computation



$$\alpha_{ts} = \frac{\exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)\right)}{\sum_{s'=1}^{S} \exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_{s'})\right)} \quad \text{[Attention weights]} \quad (1)$$

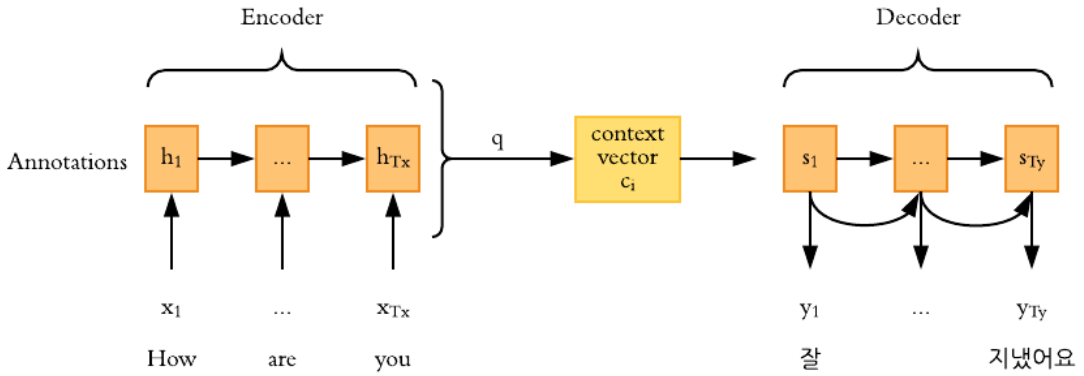
$$\boldsymbol{c}_t = \sum_s \alpha_{ts} \bar{\boldsymbol{h}}_s \quad \text{[Context vector]} \quad (2)$$

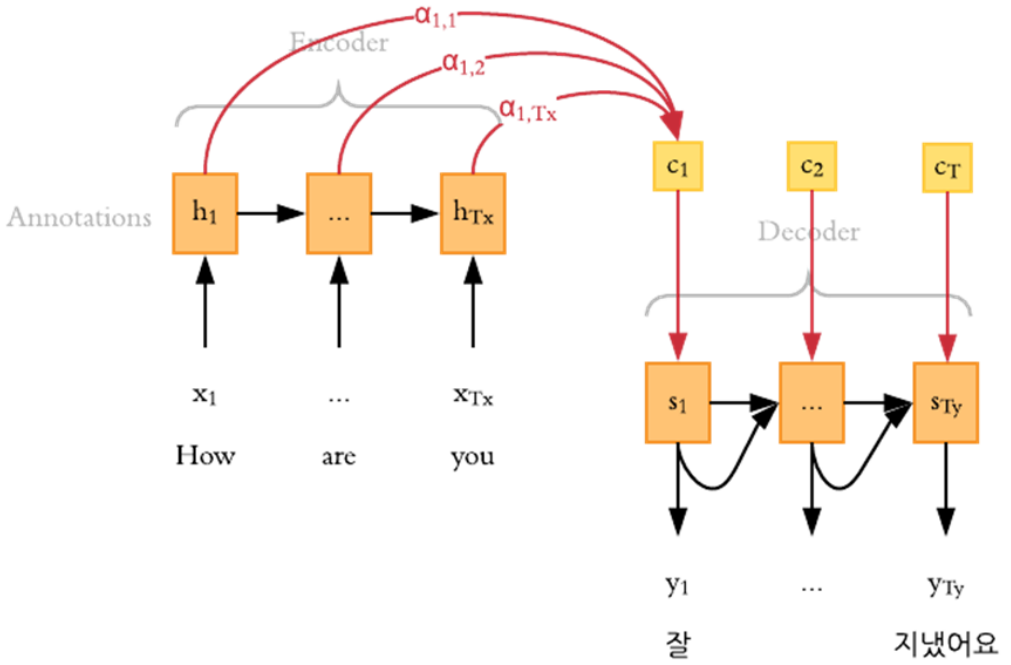
$$\boldsymbol{a}_t = f(\boldsymbol{c}_t, \boldsymbol{h}_t) = \tanh(\boldsymbol{W}_{\boldsymbol{c}}[\boldsymbol{c}_t; \boldsymbol{h}_t]) \quad \text{[Attention vector]} \quad (3)$$

$$\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \boldsymbol{W} \bar{\boldsymbol{h}}_s & \text{[Luong's multiplicative style]} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W}_1 \boldsymbol{h}_t + \boldsymbol{W}_2 \bar{\boldsymbol{h}}_s\right) & \text{[Bahdanau's additive style]} \end{cases} \quad (4)$$

https://www.tensorflow.org/tutorials/text/nmt_with_attention

Language Translation





The context vector has been given the responsibility of encoding all the information in a given source sentence into a vector of few hundred elements.

Attention allows the decoder to peek at computed hidden states (past states of the encoder – computed dynamic memory). The context vector has become a **weighted sum of all the past encoder states**.

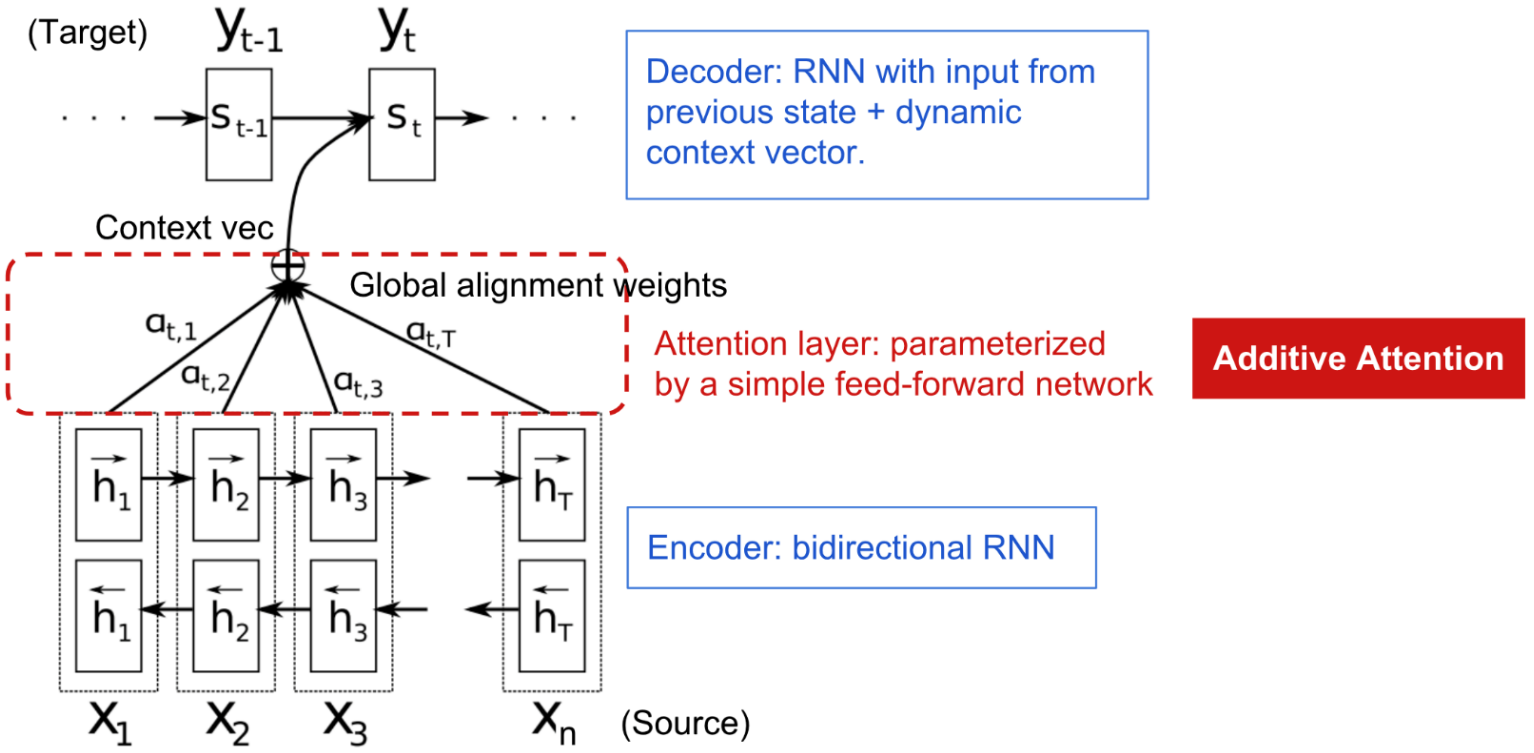
Attention Mechanisms

Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

Attention Mechanisms

Name	Definition	Citation
Self-Attention(&)	Relating different positions of the same input sequence. Theoretically the self-attention can adopt any score functions above, but just replace the target sequence with the same input sequence.	Cheng2016
Global/Soft	Attending to the entire input state space.	Xu2015
Local/Hard	Attending to the part of input state space; i.e. a patch of the input image.	Xu2015; Luong2015

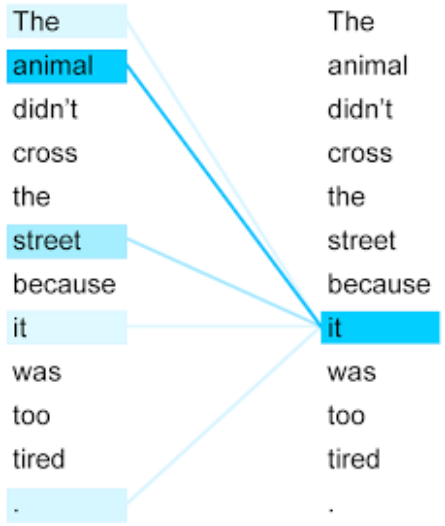
Additive Attention

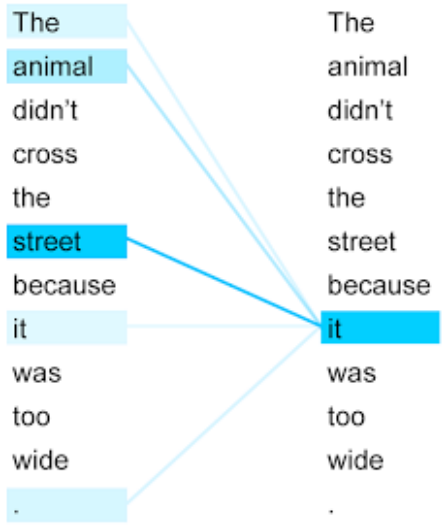


The encoder-decoder model with additive attention mechanism in Bahdanau et al., 2015.

Self Attention

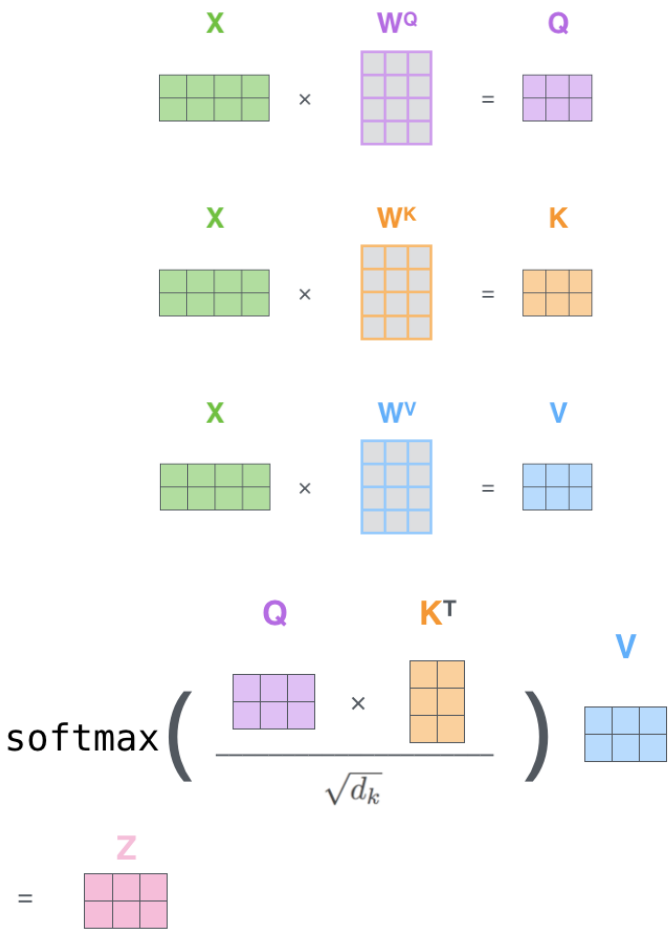
*"The animal didn't cross the street because **it** was too tired"*

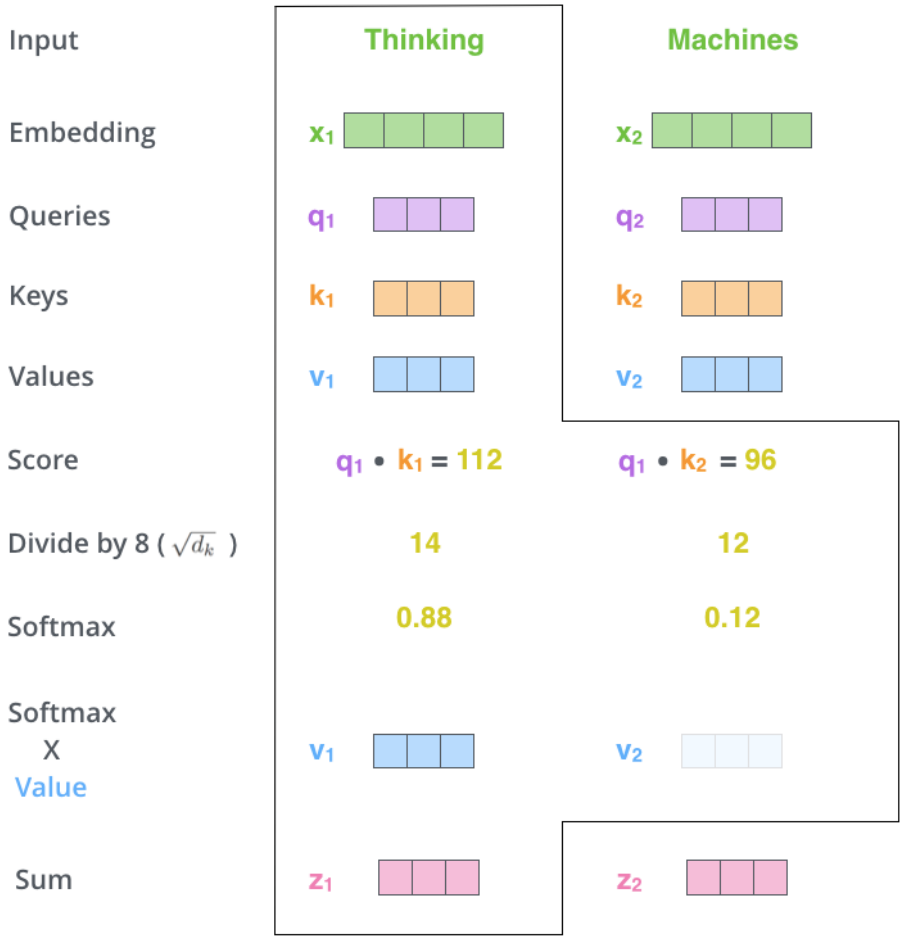




http://jalammar.github.io/illustrated-transformer/

Self Attention Computation

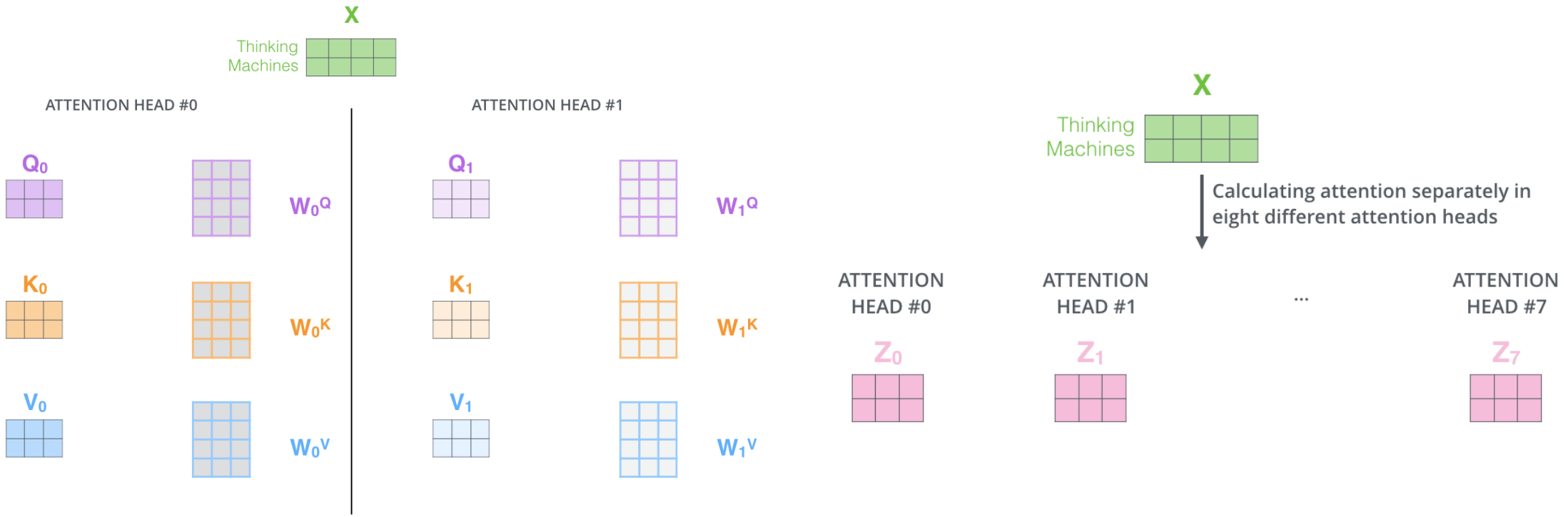




http://jalammar.github.io/illustrated-transformer/

Multiheaded Attention

- Expands the model's ability to focus on different positions by giving the attention layer multiple "representation subspaces"
- Produces multiple sets of Query/Key/Value weight matrices



http://jalammar.github.io/illustrated-transformer/

Transformer

- Transformer is a sequence model that forgoes the recurrent structure of RNN's for a fully attention-based approach
- Significantly more parallelization compared to LSTMs and reached SOTA within a few hours of training on GPUs

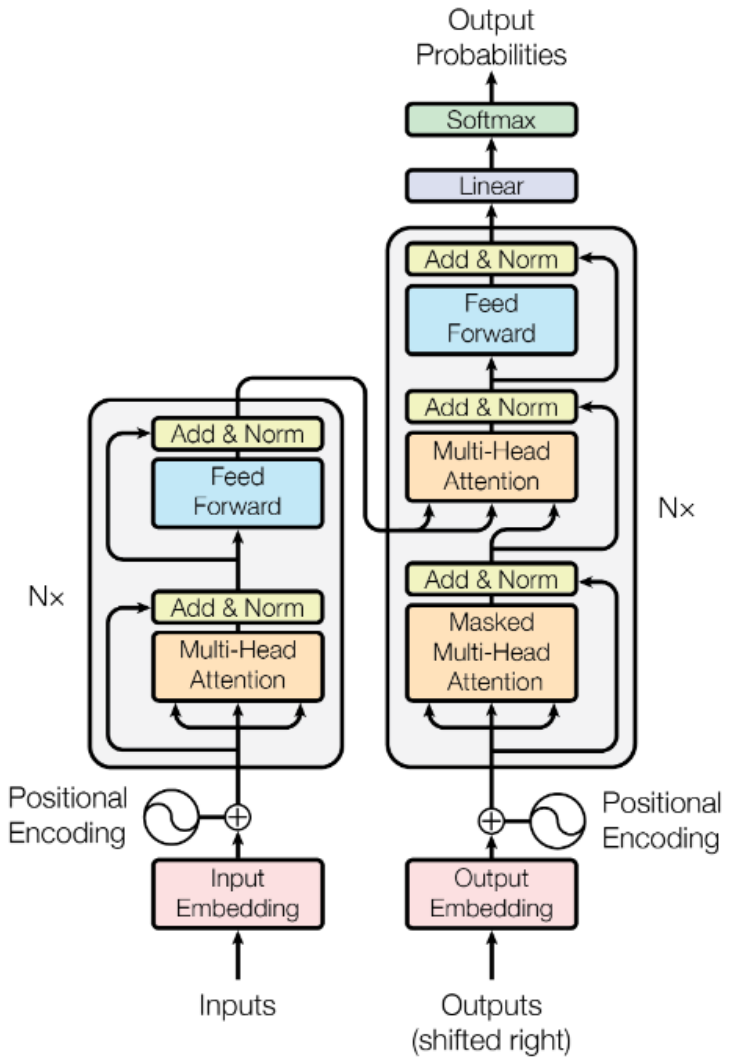


Figure 1: The Transformer - model architecture.

https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

Exercises

1. Keras RNN API
2. Time series - LSTM, GRU
3. Real world time series
4. Language models