



DEEP LEARNING

Sequence Networks

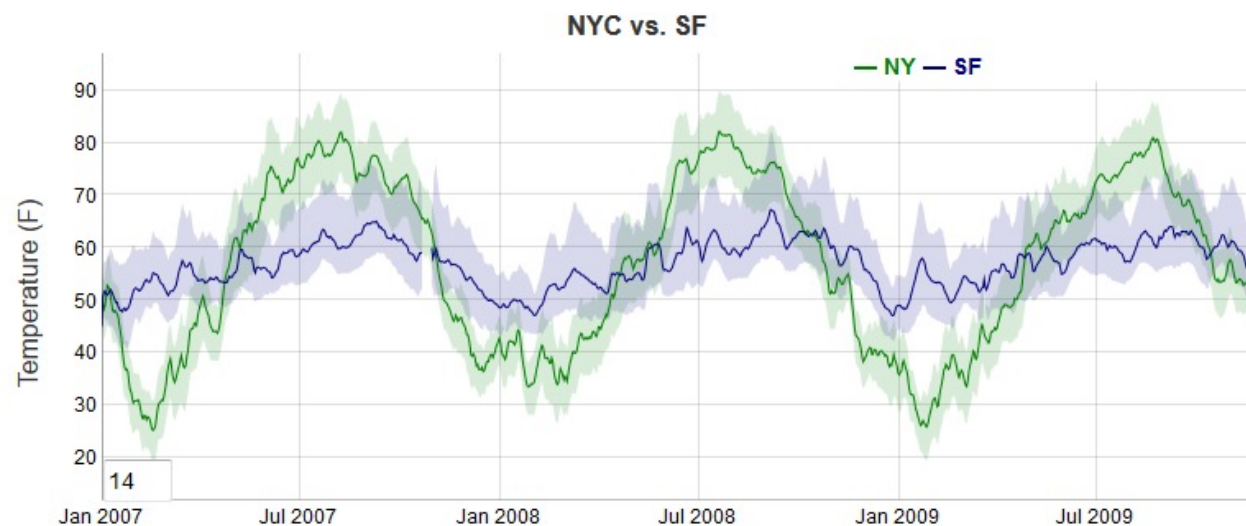
Ashish Pujari

Lecture Outline

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

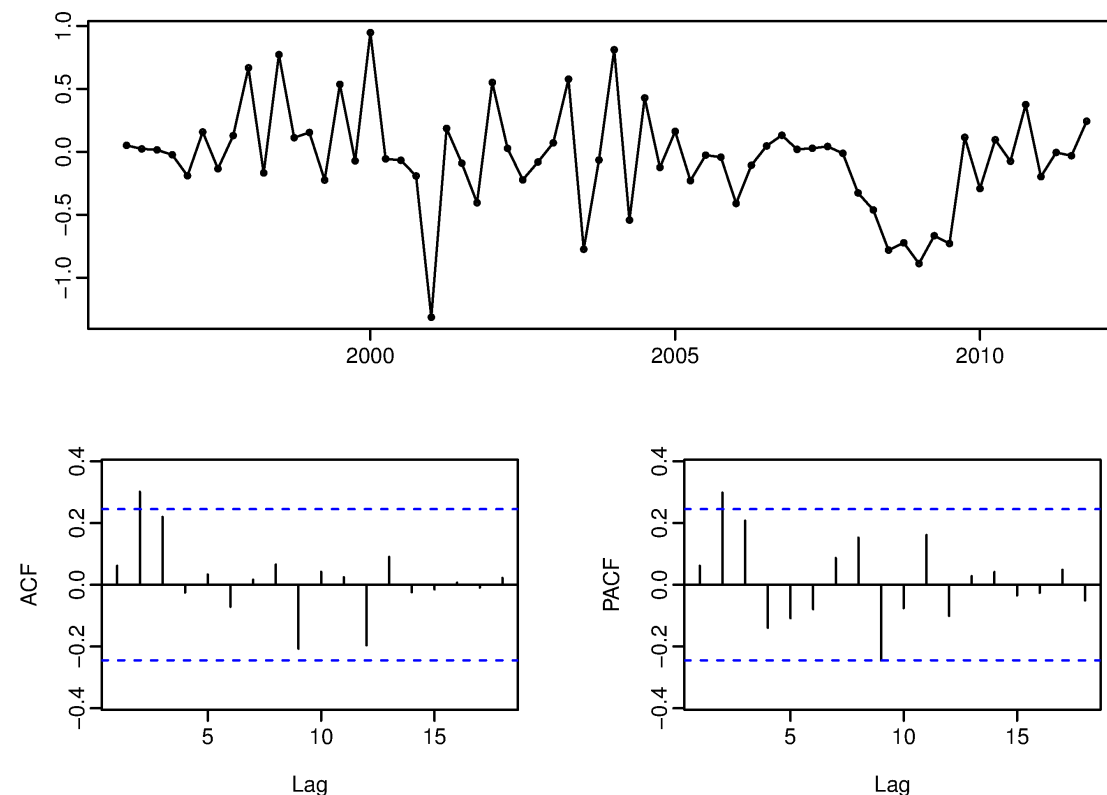
Sequential Data

- 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 |
| 2 | 0 | | 1 | 1 | 0 | 0 | 1 | 0 | | 0 | 1 | 0 |
| 3 | 1 | | 1 | 0 | 0 | 1 | 0 | 1 | | 1 | 0 | 2 |

Distributional Similarity Hypothesis

- Words that occur in the same contexts tend to have similar semantic meanings
- For a given word w and another word w' appearing in contexts around it, let us denote by $P(w'|w)$ the probability of w appearing close to w'

$$\frac{P(w'|man)}{P(w'|woman)} \approx \frac{P(w'|king)}{P(w'|queen)}$$

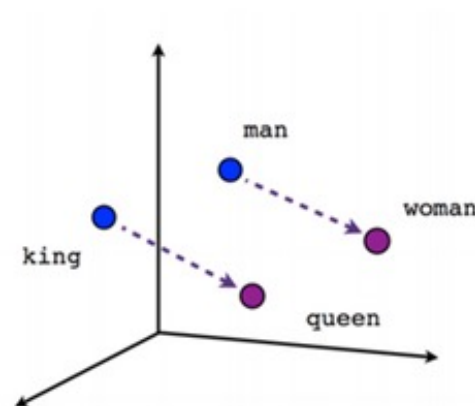
You shall know a word by the company it keeps – J.R Firth 1957

Word Embedding

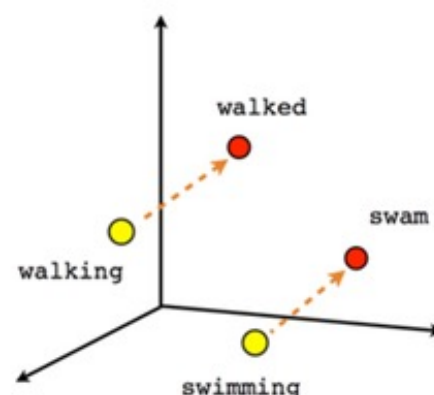
- Real number vector representation of a word
- Words with similar meaning will have vector representations that are close together in the embedding space

$$W(\text{"cat"}) = (0.2, -0.4, 0.7, \dots)$$

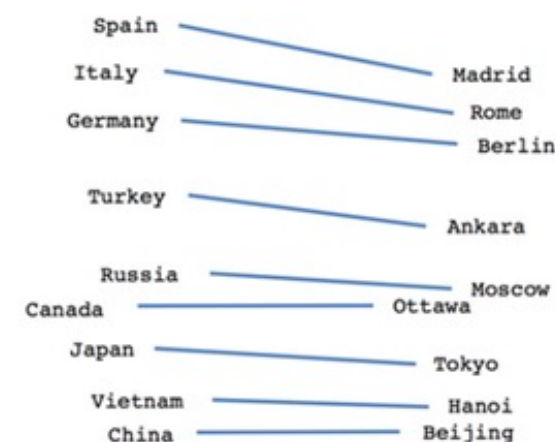
$$W(\text{"mat"}) = (0.0, 0.6, -0.1, \dots)$$



Male-Female



Verb tense



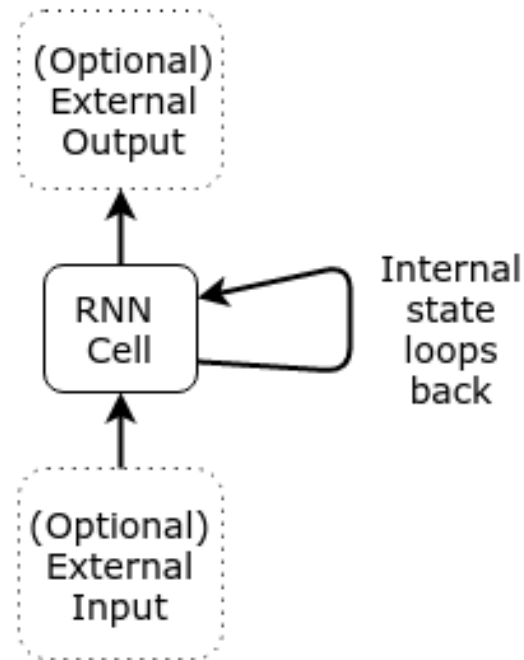
Country-Capital

RECURRENT NEURAL NETWORKS

Introduction

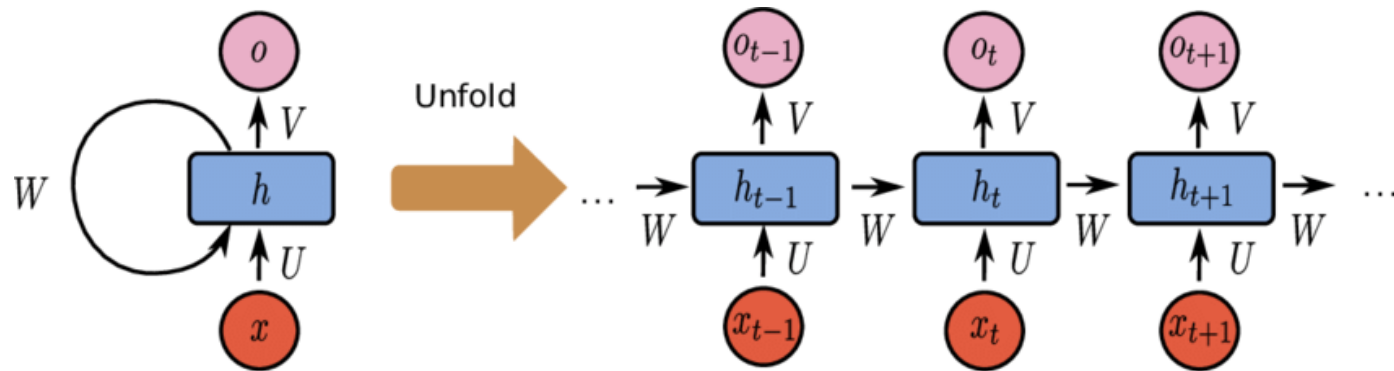
Recurrent Neural Networks (RNN)

- 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

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

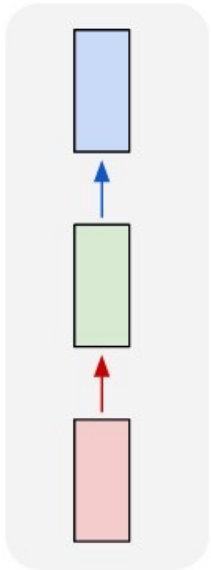


x_t Input at time t
 h_t Hidden State at time t
 h_{t-1} Hidden State at time $t-1$
 o Output at time t

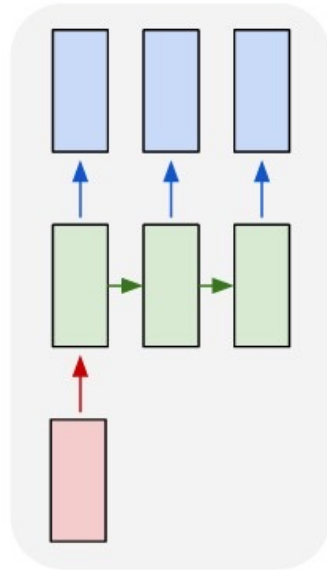
U Weight Matrix for Input to Hidden layer at time t
 W Weight Matrix for Hidden layer at $t-1$ to Hidden layer at t
 V Weight Matrix for Hidden layer to the output
 ϕ Activation Function (Sigmoid or Tanh)

RNN Input Output

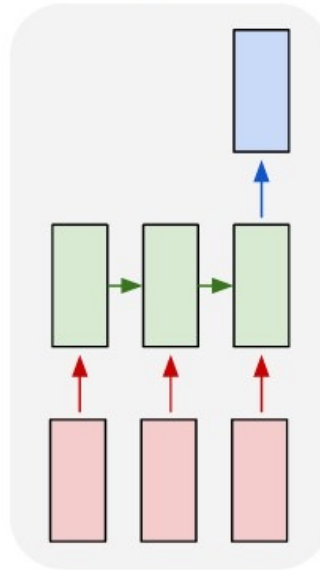
one to one



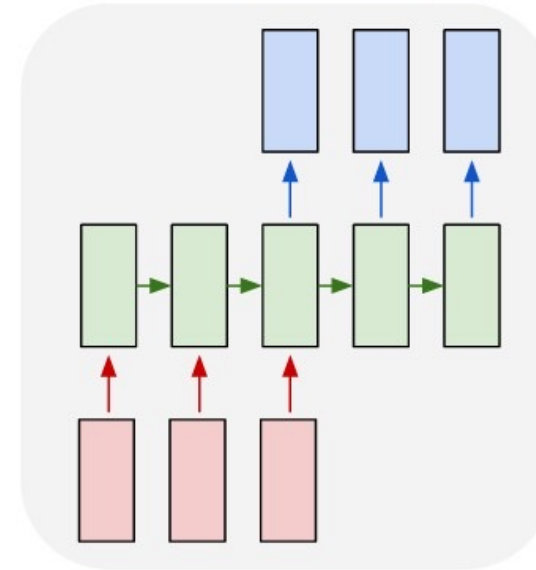
one to many



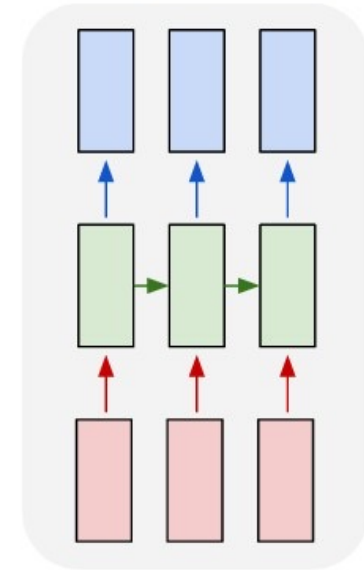
many to one



many to many



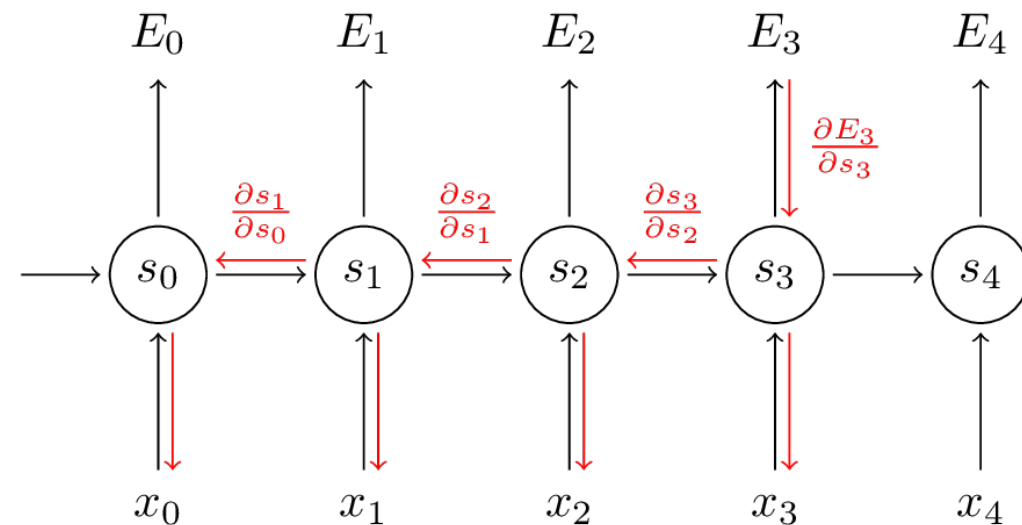
many to many



RNN: Training

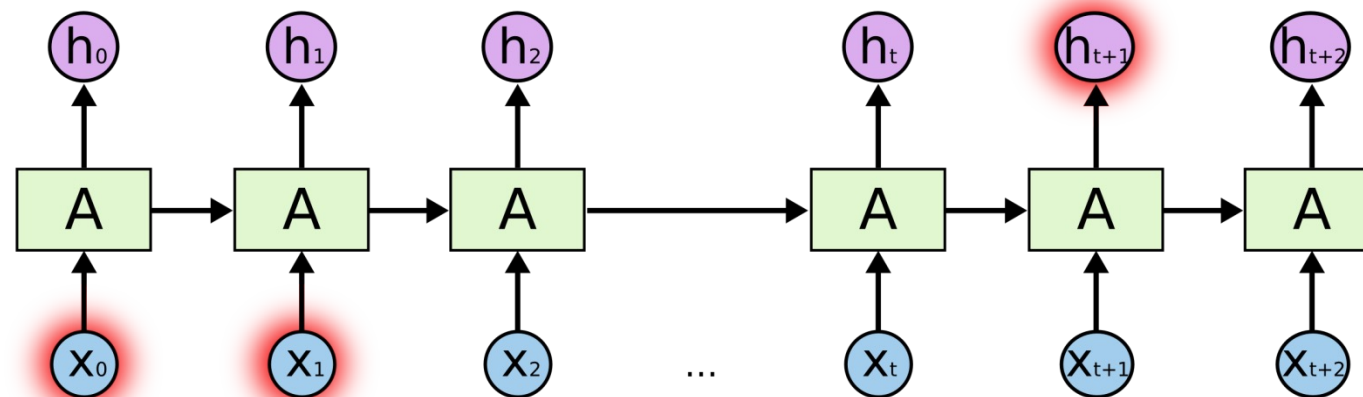
- Parameters are shared across RNN cells, so we need to calculate the gradient with respect to each time step separately and then add them up
- Backpropagation Through Time (BPTT)
 - Present a sequence of timesteps of input and output pairs to the network.
 - Unroll the network then calculate and accumulate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

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



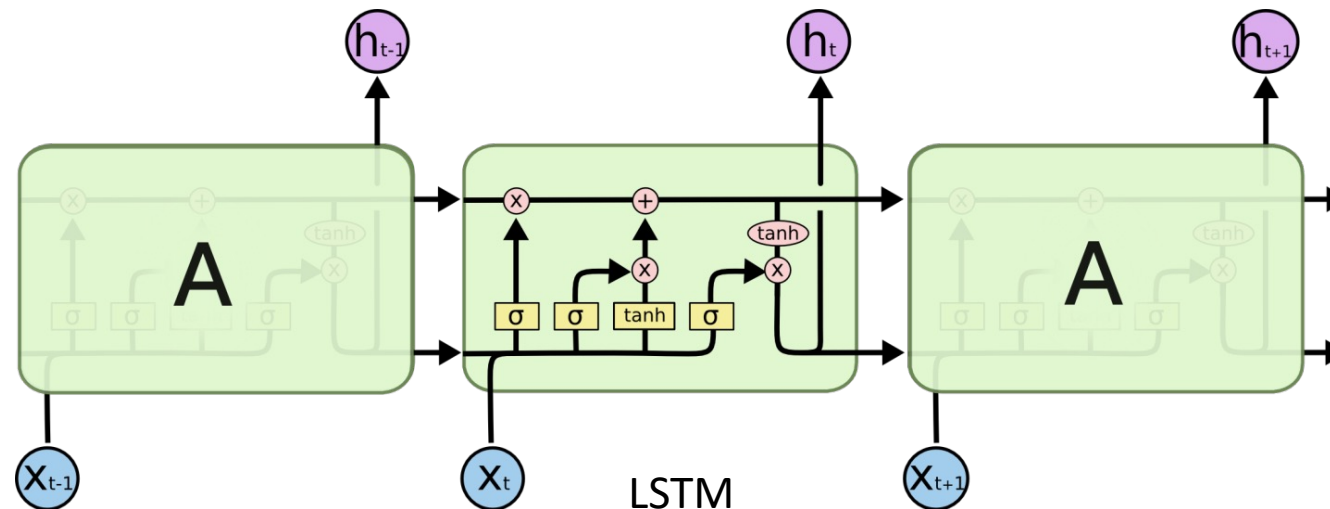
Long-term Dependencies

- 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



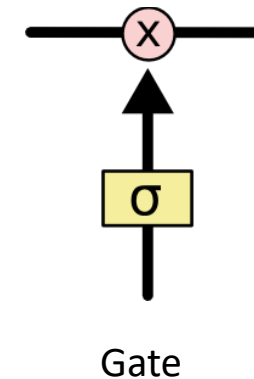
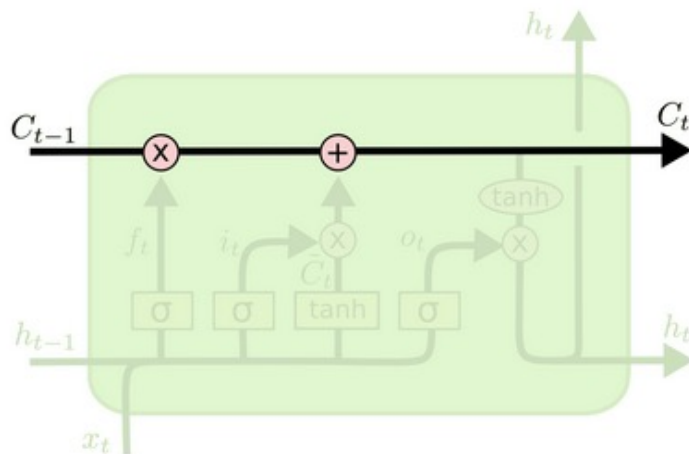
LSTM (Long Short-Term Memory)

- 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: Cell State

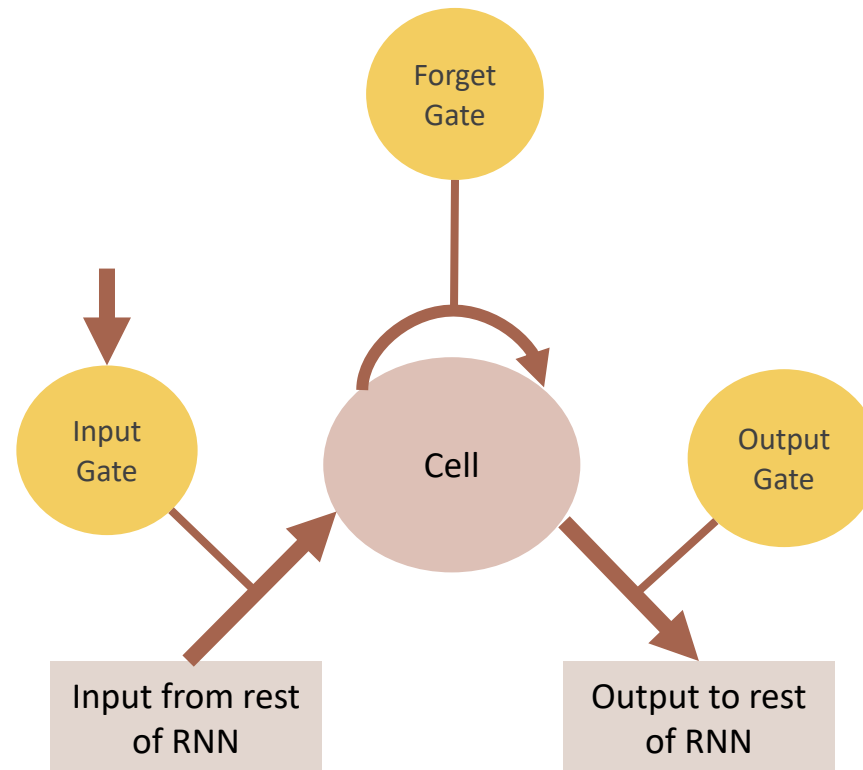
- Horizontal line running through the top of the diagram is the cell state
- LSTM has the ability to remove or add information to the cell state, carefully regulated by structures called gates



LSTM: Gates

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

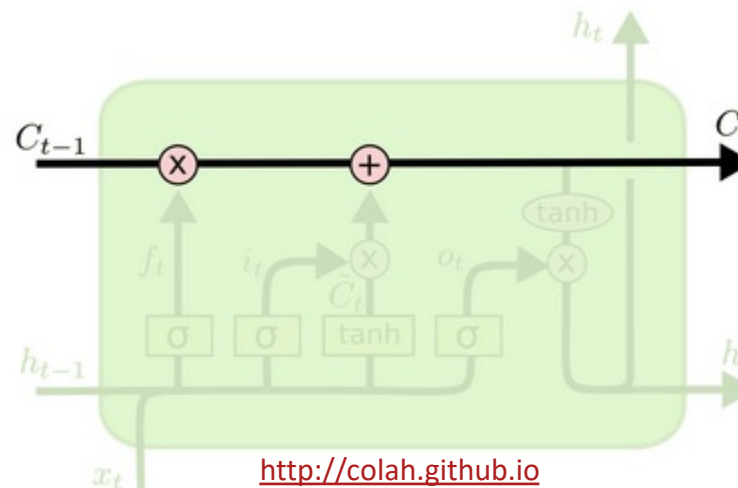
Controls how much new information should be added to cell state



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

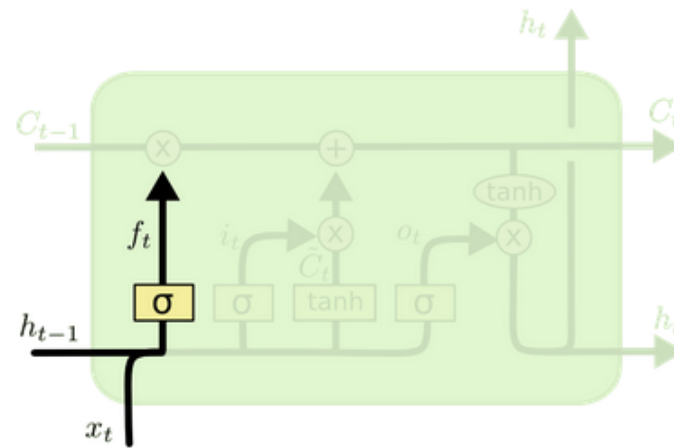
LSTM: Variables

- 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 $h(t)$



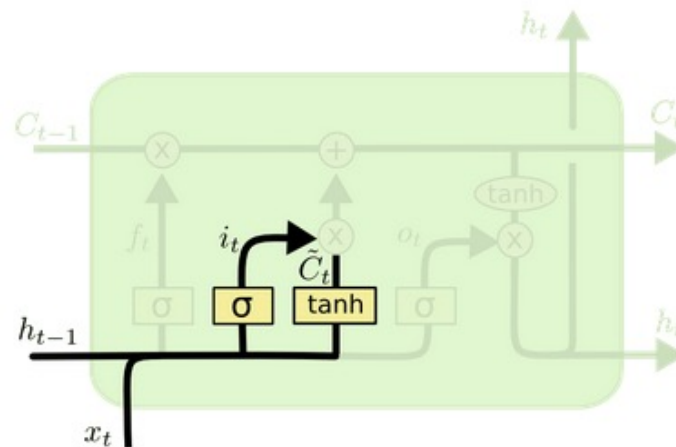
LSTM: Steps 1,2

1. Forget gate



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

2. Decide State Update

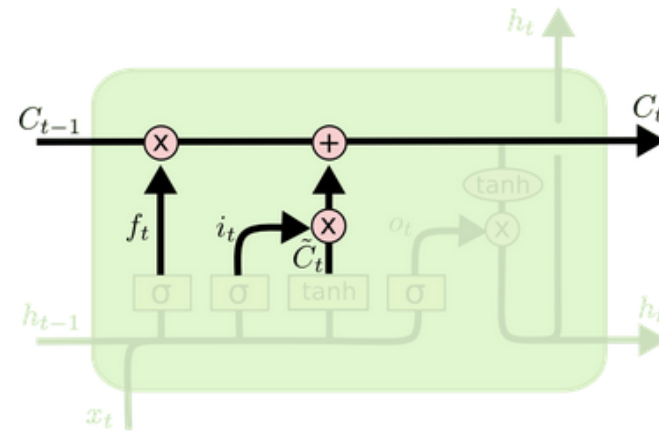


$$i_t = \sigma (W_i \cdot [h_{t-1}, x_t] + b_i)$$

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

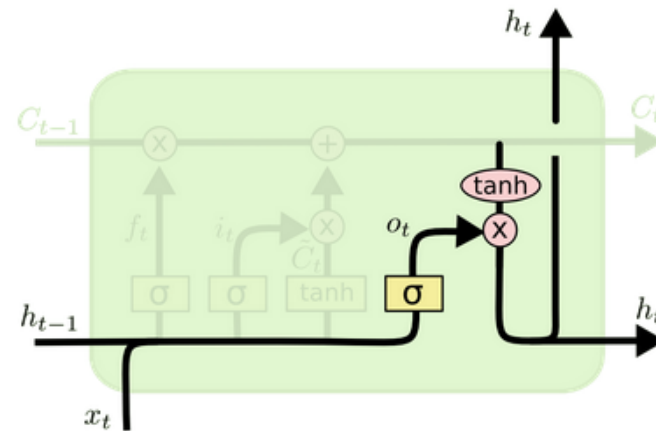
LSTM: Steps 3,4

3. Update



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

4. Decide Output



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

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

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 = $4(nm + n^2 + n)$

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

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

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

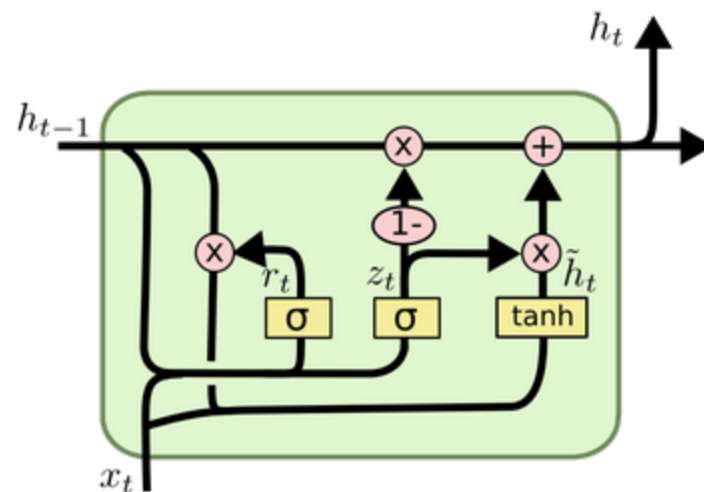
$$g_t = \tanh(W_c x_t + U_c h_{t-1} + b_c)$$

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

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

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



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

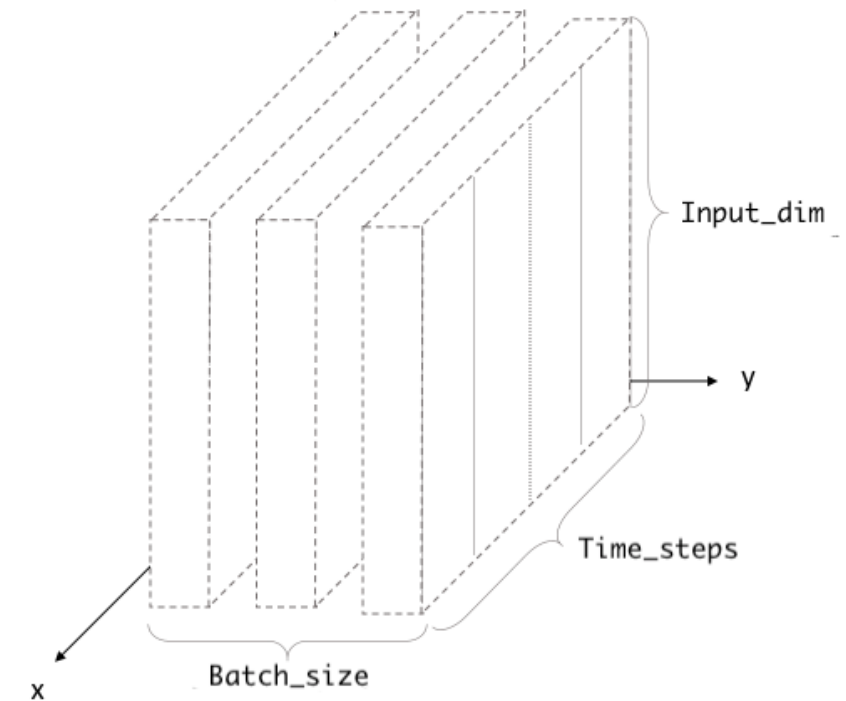
$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

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

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

LSTM in Keras

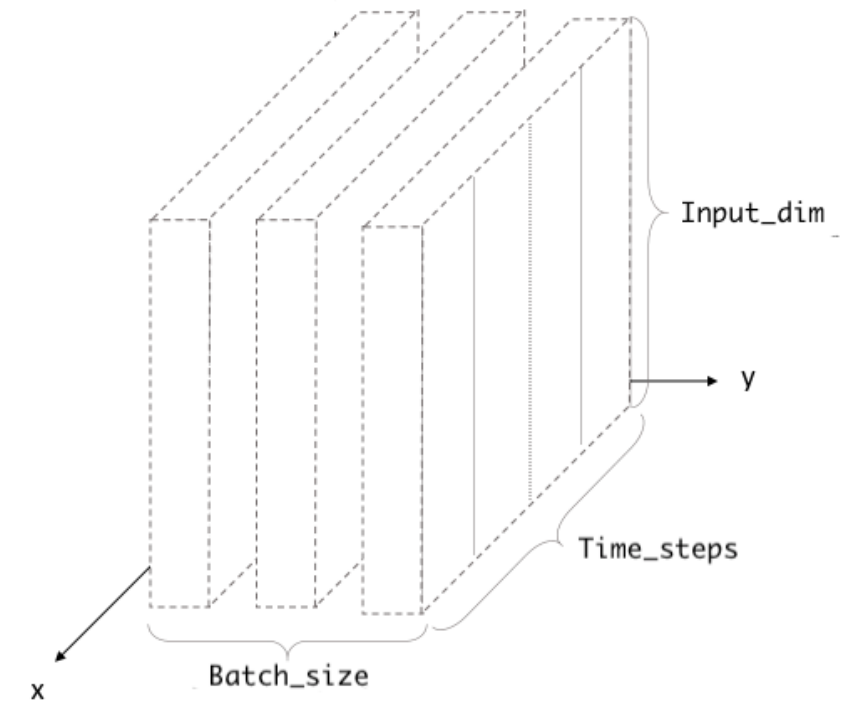
- Input Parameters
 - Take in a three-dimensional array as an input.
- Statefulness
 - If layers are set to 'stateful', then the states computed for the samples in one batch will be reused as initial states for the samples in the next batch.
 - This assumes a one-to-one mapping between samples in different 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

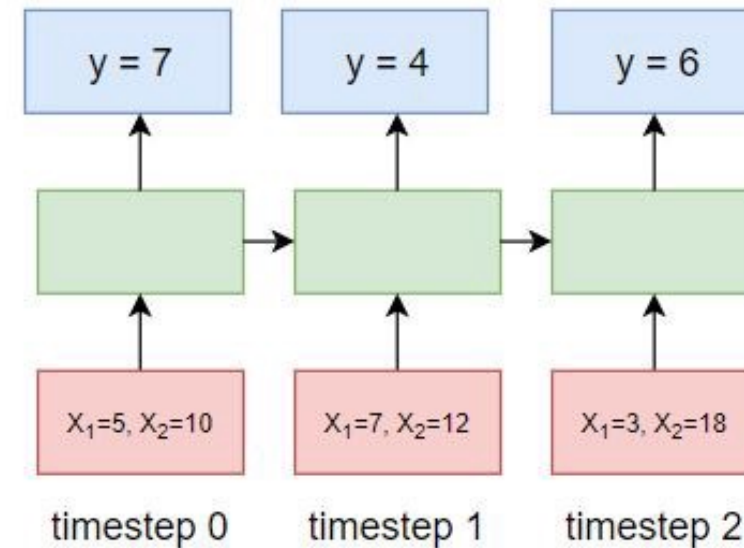
```
model = Sequential()  
model.add ( LSTM(units=1, batch_input_shape=(None, 5,1), return_sequences=False) )
```

- 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

| timesteps | features | | target |
|------------|----------|-------|--------|
| | 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 |



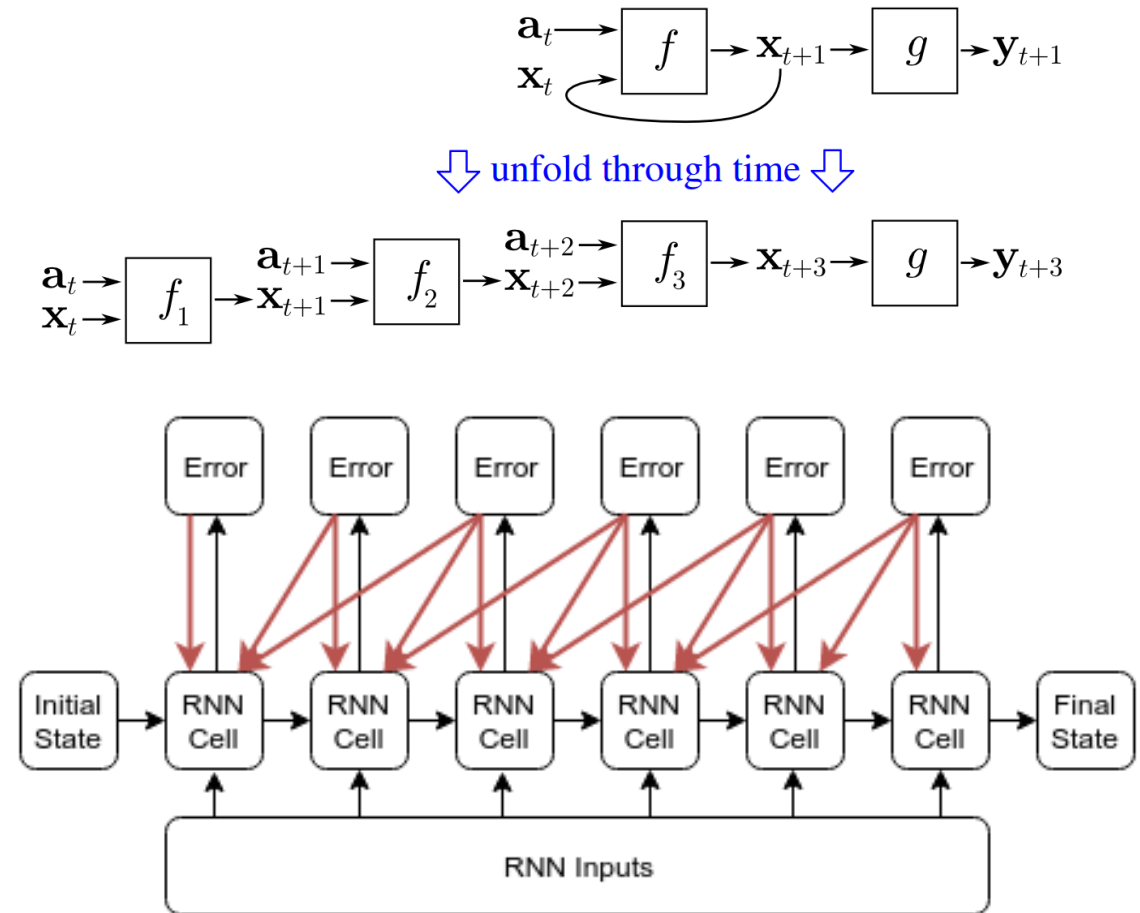
Truncated Back Propagation Through Time (TBPTT)

- 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.

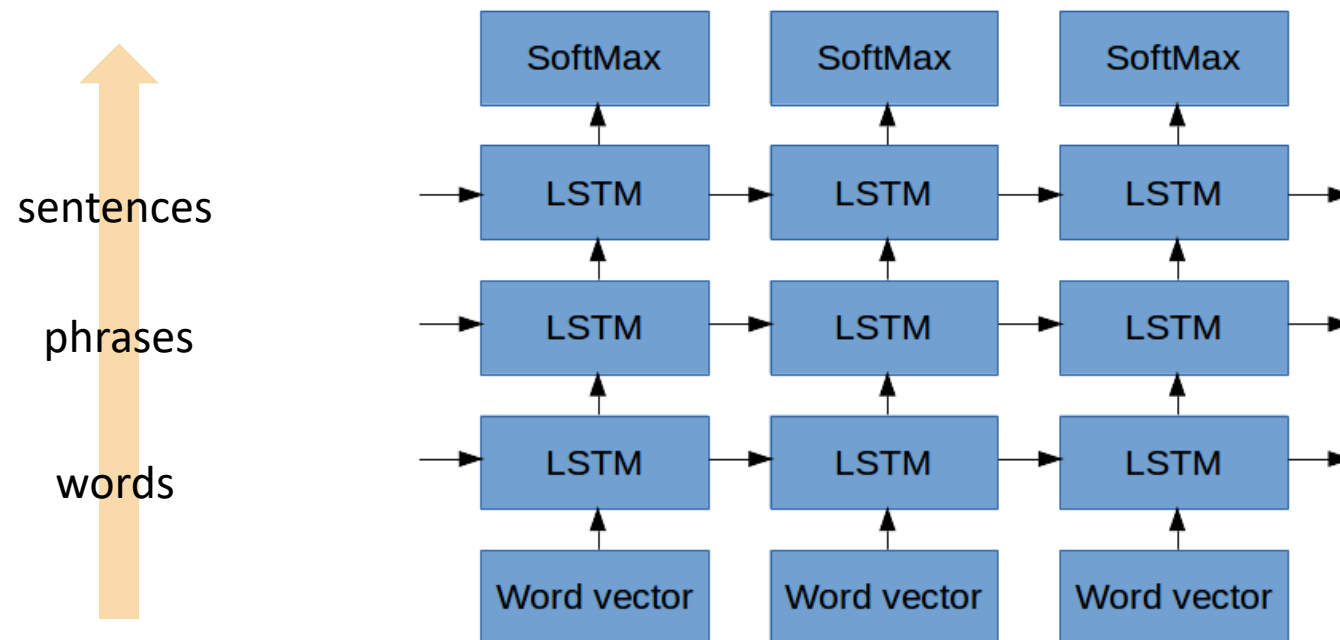


RNN ARCHITECTURES

Stacked, Bidirectional RNNs

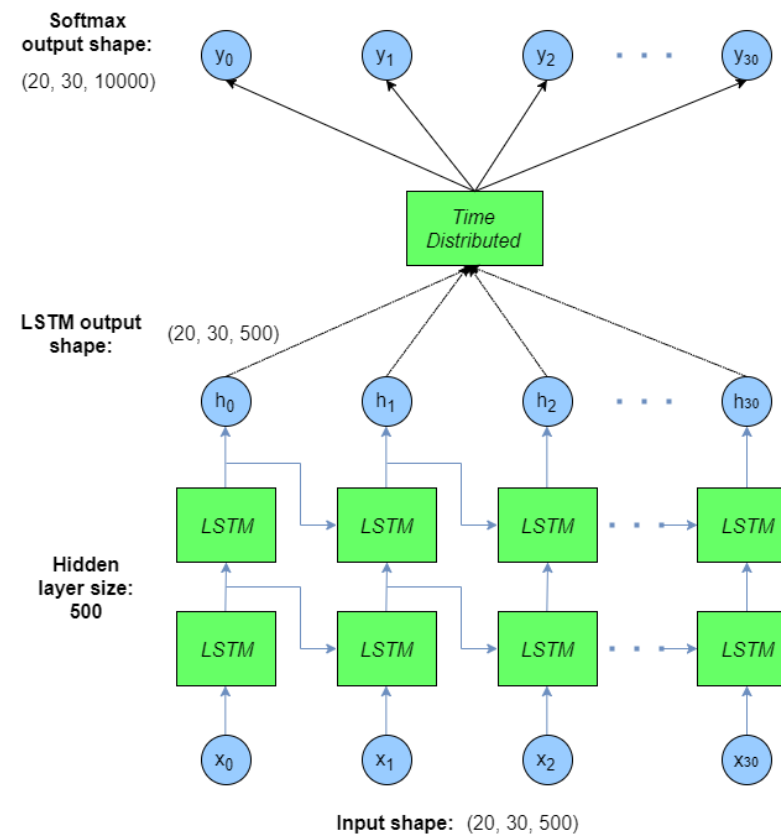
Stacked LSTM

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



Time-Distributed Layer

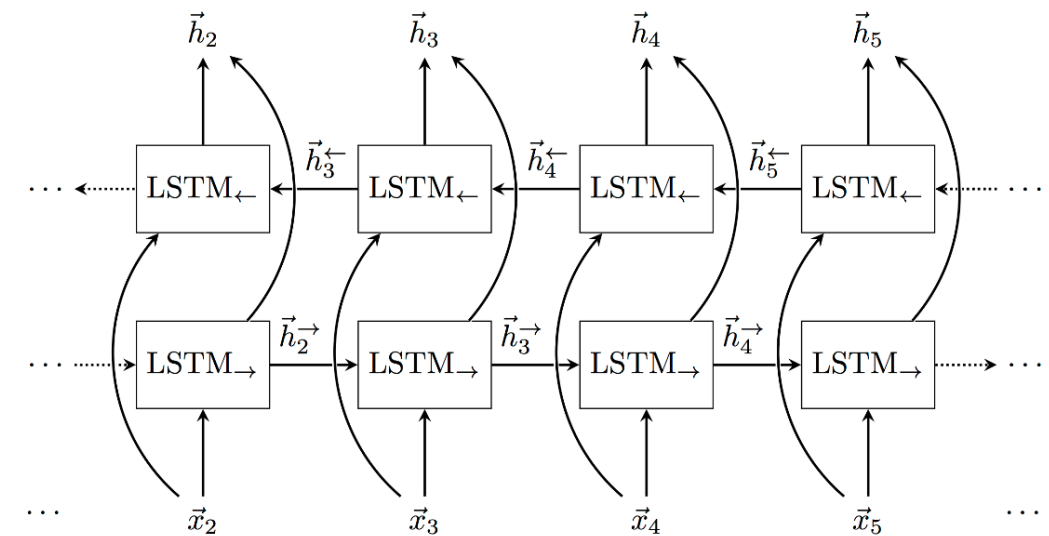
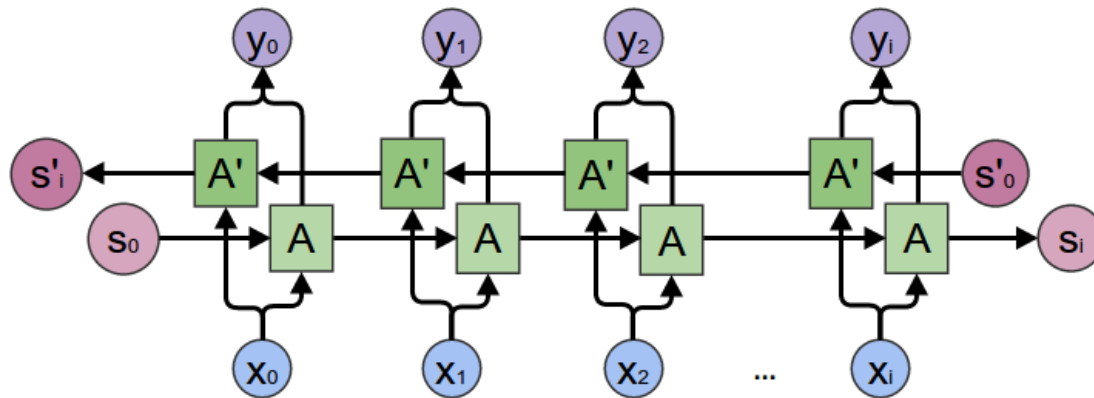
- A wrapper that allows to apply a layer to every temporal slice of an input.



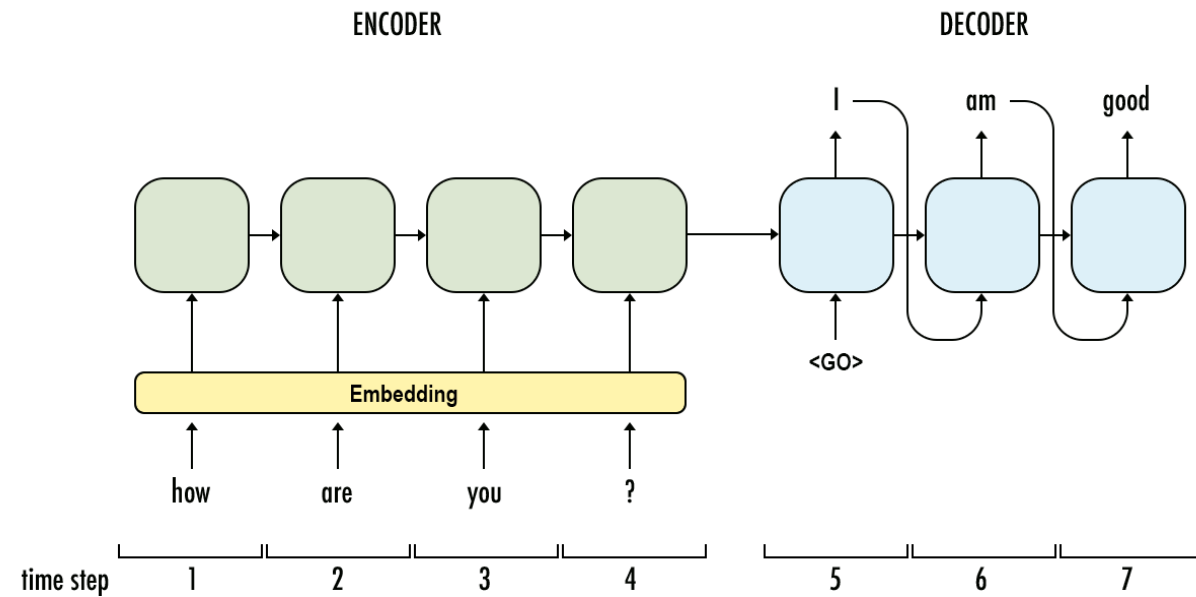
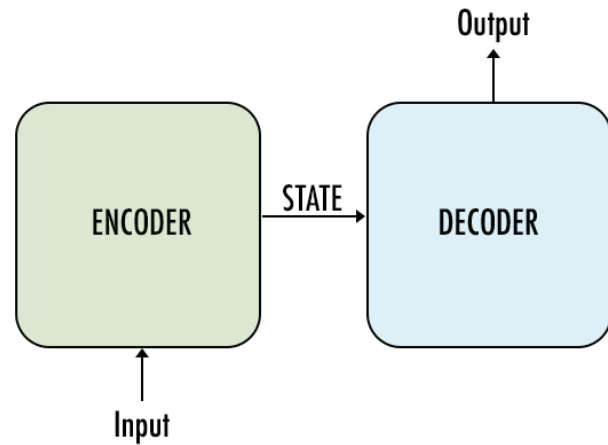
(batch size, number of time steps, hidden size)

Bidirectional RNN

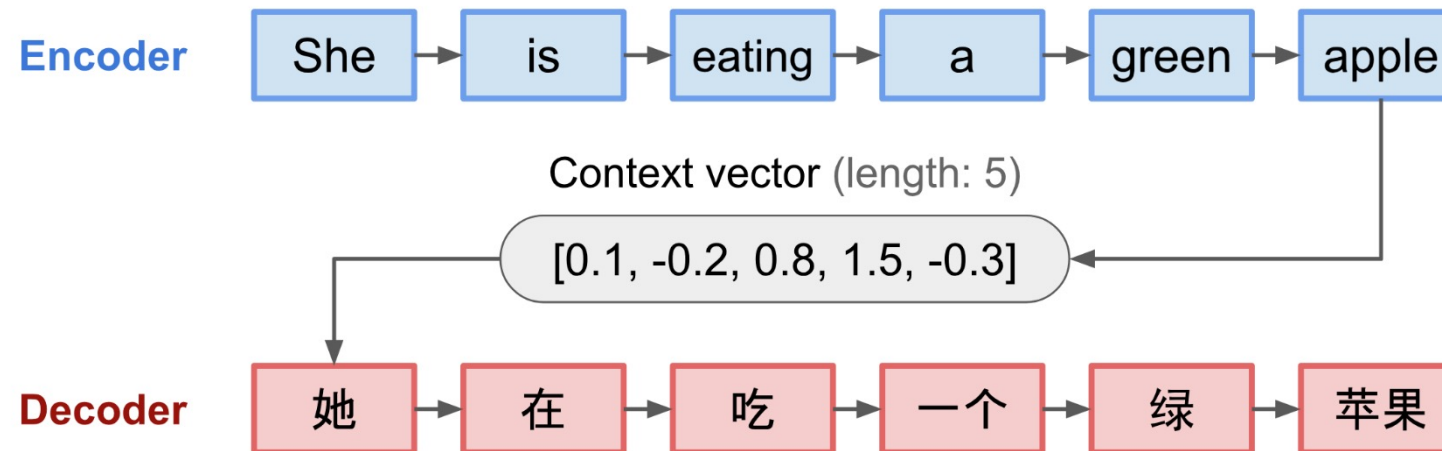
- Allows the networks to have both backward and forward information about the sequence at every time step (to exploit both directions of temporal dependence)



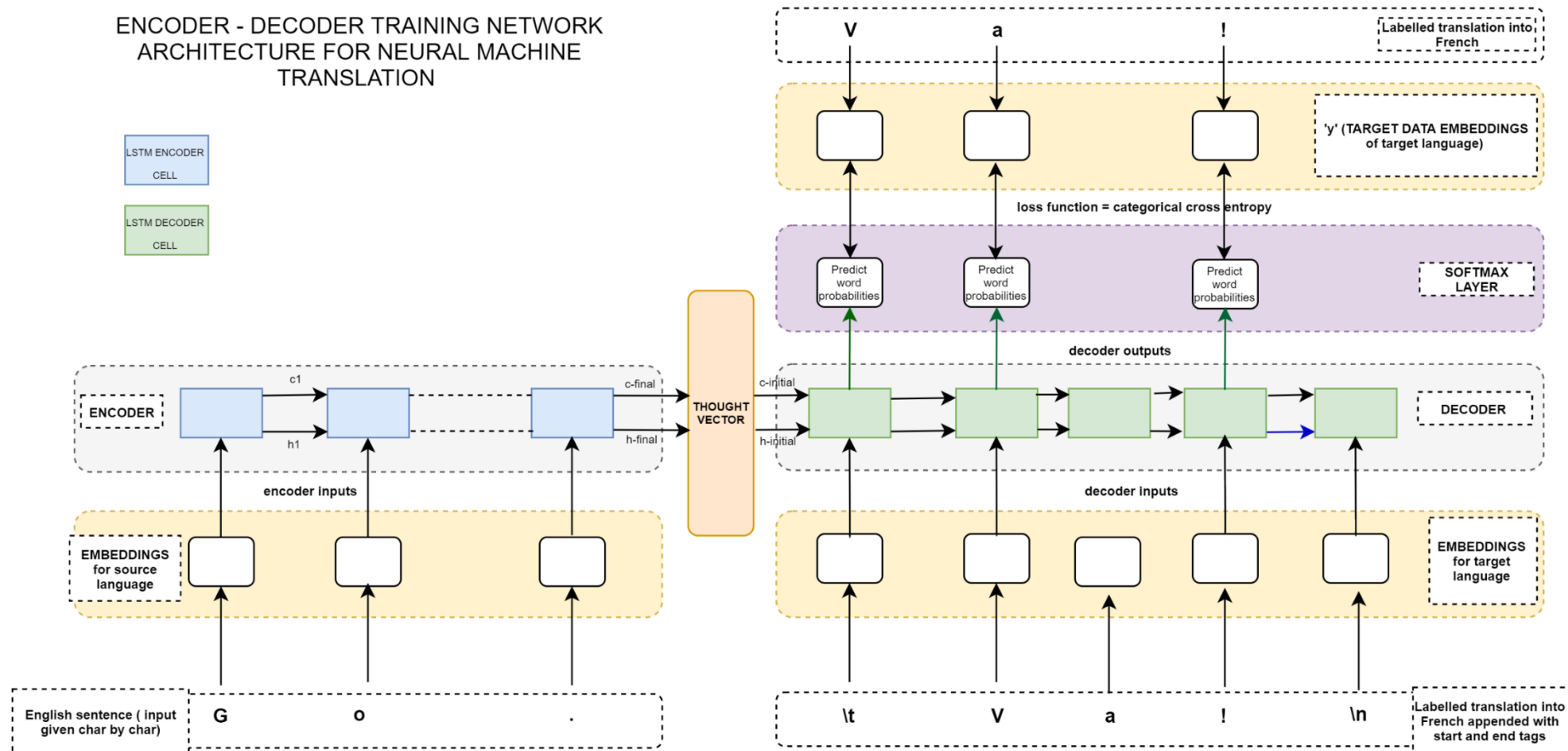
Seq2Seq Models



Seq2Seq: Neural Machine Translation (NMT)



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