Natural Language Processing using RNNs and LSTMs

Objectives

- What is Sequential data?
- Using traditional ML for Sequential Data
- Type of analysis possible on sequential data using recurrence
- Recurrent Neural Networks - Architecture

What is sequential data?

- One-dimensional discrete index
  - Example: time instances, character position
- Each data point can be a scalar, vector, or a symbol from an alphabet
- Number of data points in a series can be *variable*

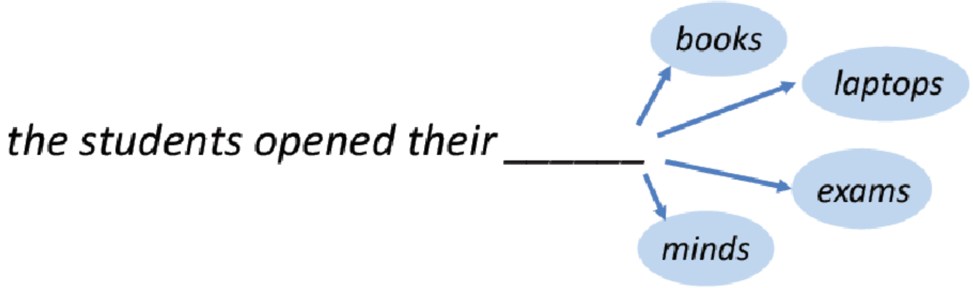
$$\dots \quad \boldsymbol{x}_{n-4} \quad \boldsymbol{x}_{n-3} \quad \boldsymbol{x}_{n-2} \quad \boldsymbol{x}_{n-1} \quad \boldsymbol{x}_{n} \quad \dots$$

Examples of sequential data

- Speech
- Text (NLP)
- Music
- Protein and DNA sequences
- Stock prices and other time series

Language Modeling

- **Language Modeling** is the task of predicting what word comes next.



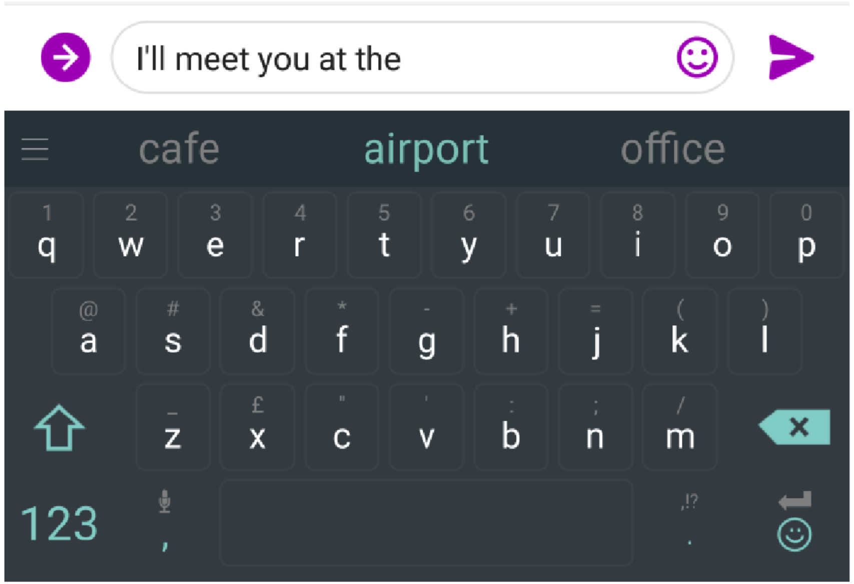
- More formally: given a sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$, compute the probability distribution of the next word $\boldsymbol{x}^{(t+1)}$:

$$P(\boldsymbol{x}^{(t+1)} = \boldsymbol{w}_j \mid \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

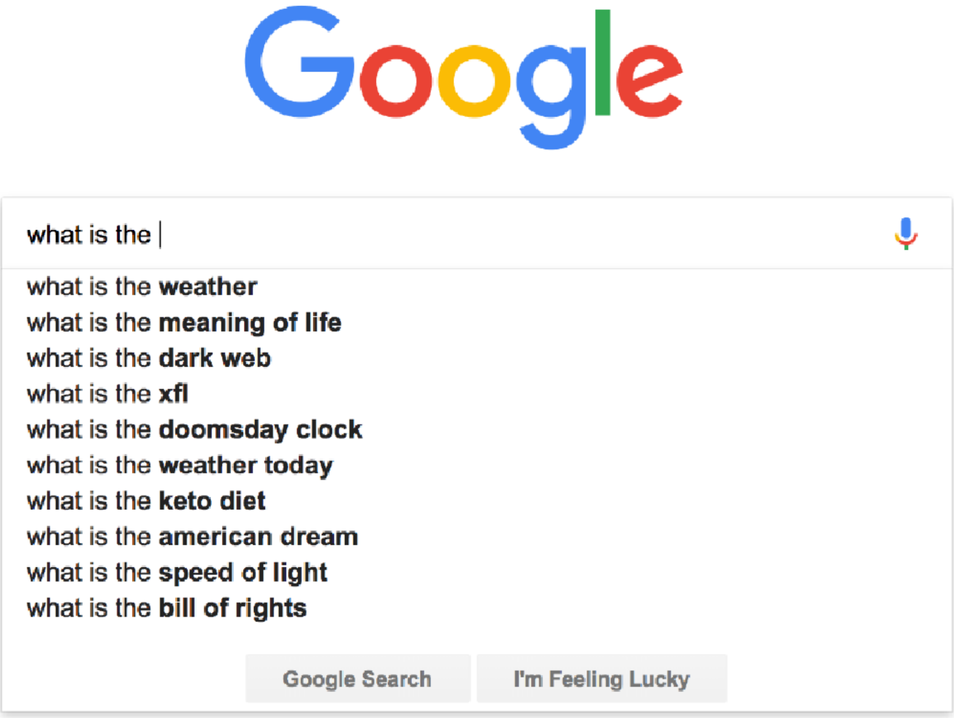
where $\boldsymbol{w}_j$ is a word in the vocabulary $V = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{|V|}\}$

- A system that does this is called a **Language Model**.

You use Language Models every day



You use Language Models every day



N-gram Language Models

the students opened their ______

- **Question**: How to learn a Language Model?
- **Answer** (pre- Deep Learning): learn a *n*-gram Language Model!

- Definition: A *n*-gram is a chunk of *n* consecutive words.
  - unigrams: “the”, “students”, “opened”, ”their”
  - bigrams: “the students”, “students opened”, “opened their”
  - trigrams: “the students opened”, “students opened their”
  - 4-grams: “the students opened their”

- Idea: Collect statistics about how frequent different n-grams are, and use these to predict next word.

N-gram Language Models

- First we make a simplifying assumption: $\boldsymbol{x}^{(t+1)}$ depends only on the preceding *(n-1)* words

$$P(\boldsymbol{x}^{(t+1)}|\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(1)}) = P(\boldsymbol{x}^{(t+1)}|\overbrace{\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)}}^{n\text{-1 words}})$$

(assumption)

prob of a n-gram

prob of a (n-1)-gram

$$= \frac{\boxed{P(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}}{\boxed{P(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}}$$

(definition of conditional prob)

- **Question:** How do we get these *n*-gram and (*n*-1)-gram probabilities?
- **Answer:** By counting them in some large corpus of text!

$$\approx \frac{\text{count}(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{\text{count}(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}$$

(statistical approximation)

N-gram Language Models

Suppose we are learning a 4-gram Language Model.

~~as the proctor started the clock, the~~ *students opened their* ______

discard

condition on this

$$P(\boldsymbol{w}_j|\text{students opened their}) = \frac{\text{count(students opened their } \boldsymbol{w}_j)}{\text{count(students opened their)}}$$

In the corpus:

- "students opened their" occurred 1000 times
- "students opened their books" occurred 400 times
  - → P(books | students opened their) = 0.4
- "students opened their exams" occurred 100 times
  - → P(exams | students opened their) = 0.1

Should we have discarded the "proctor" context?

Problems with N-gram Language Models

Sparsity Problem 1

Problem: What if *"students opened their $\boldsymbol{w}_j$"* never occurred in data? Then $\boldsymbol{w}_j$ has probability 0!

(Partial) Solution: Add small δ to count for every $\boldsymbol{w}_j \in V$. This is called *smoothing*.

$$P(\boldsymbol{w}_j|\text{students opened their}) = \frac{\boxed{\text{count}(\text{students opened their } \boldsymbol{w}_j)}}{\boxed{\text{count}(\text{students opened their})}}$$

Sparsity Problem 2

Problem: What if *"students opened their"* never occurred in data? Then we can't calculate probability for *any* $\boldsymbol{w}_j$!

(Partial) Solution: Just condition on *"opened their"* instead. This is called *backoff*.

Note: Increasing *n* makes sparsity problems *worse*. Typically we can't have *n* bigger than 5.

Problems with N-gram Language Models

Storage: Need to store count for all possible *n*-grams. So model size is O(exp(*n*)).

$$P(\boldsymbol{w}_j|\text{students opened their}) = \frac{\boxed{\text{count}(\text{students opened their } \boldsymbol{w}_j)}}{\text{count}(\text{students opened their})}$$

Increasing *n* makes model size huge!

How to build a Neural Language Model?

- Recall the Language Modeling task:
  - Input: sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$
  - Output: prob dist of the next word $P(\boldsymbol{x}^{(t+1)} = \boldsymbol{w}_j \mid \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$

- How about a window-based neural model?

A Fixed Window Neural Model

~~as the proctor started the clock~~ *the students opened their* ______

discard

fixed window

A Fixed Window Neural Model

output distribution

$$\hat{\boldsymbol{y}} = \mathrm{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

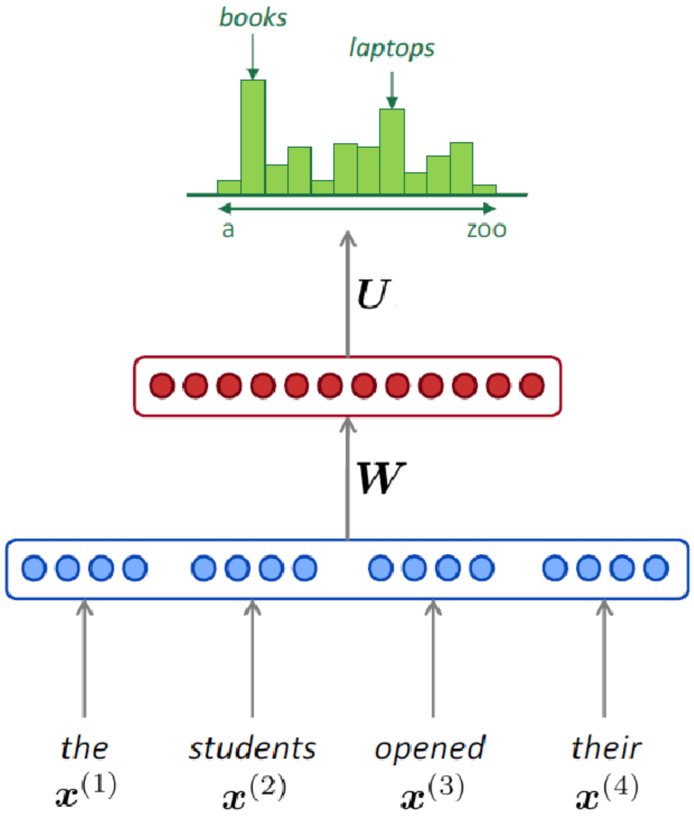
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

$$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

words / one-hot vectors

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$



A Fixed Window Neural Model

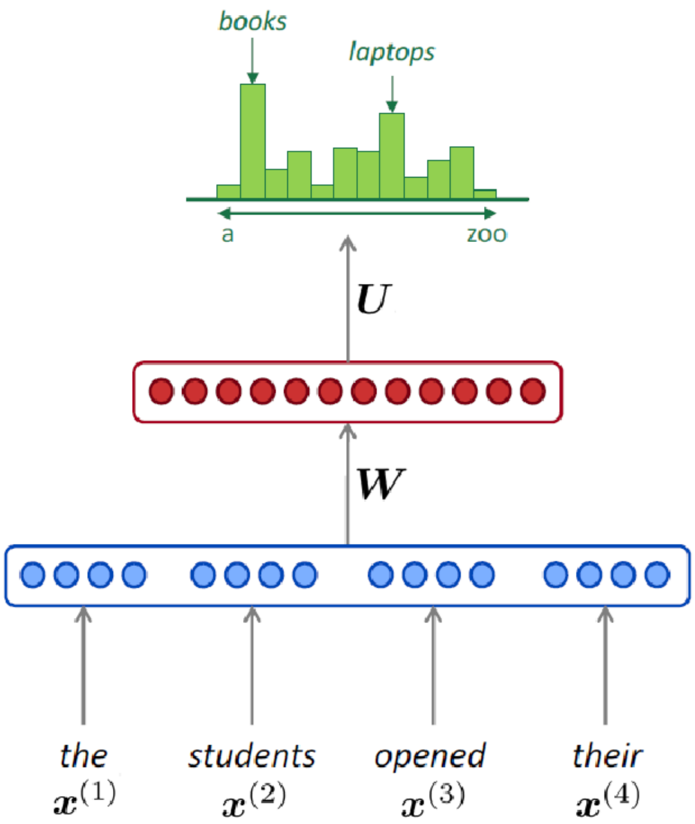
Improvements over n-gram LM:

- No sparsity problem
- Model size is $O(n)$ not $O(\exp(n))$

Remaining **problems**:

- Fixed window is too small
- Enlarging window enlarges $\boldsymbol{W}$
- Window can never be large enough!
- Each $\boldsymbol{x}^{(i)}$ uses different rows of $\boldsymbol{W}$. We don't share weights across the window.

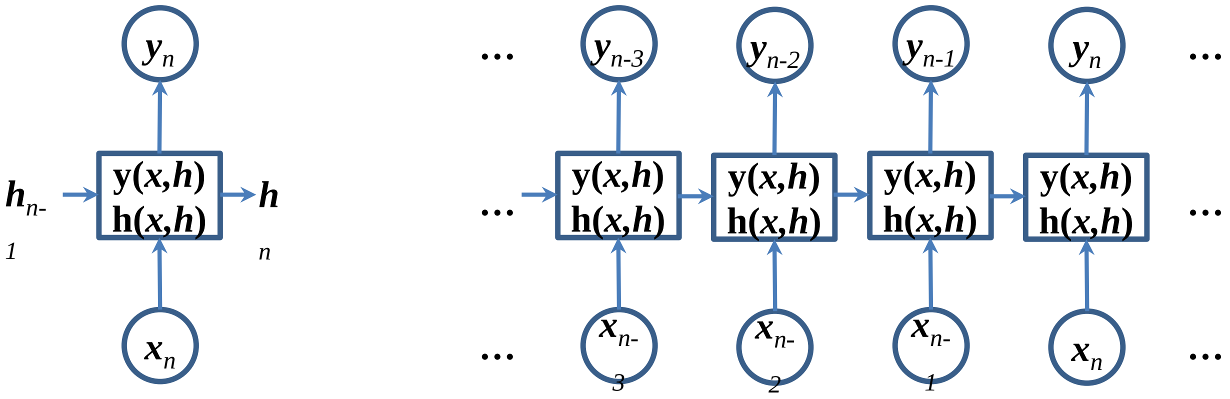
We need a neural architecture that can process *any length input*



$\boldsymbol{x}^{(1)}$ $\boldsymbol{x}^{(2)}$ $\boldsymbol{x}^{(3)}$ $\boldsymbol{x}^{(4)}$

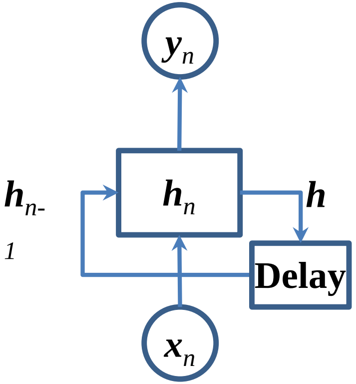
Introducing memory (recurrence or state) in neural networks

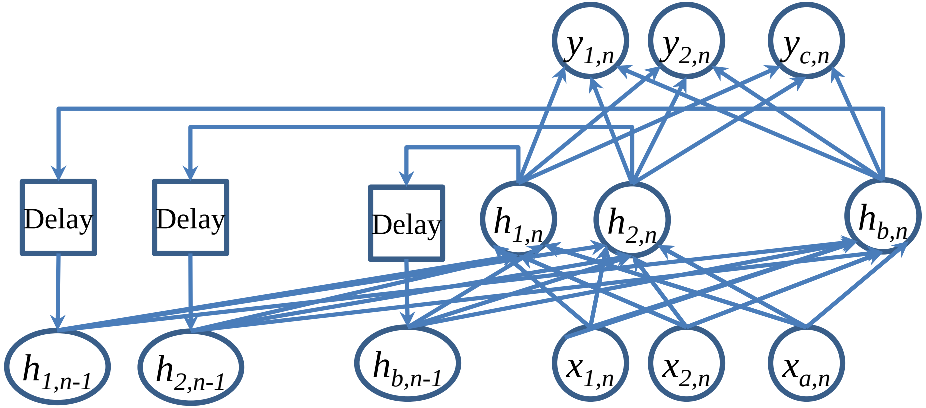
- A memory state is computed in addition to an output, which is sent to the next time instance



Another view of recurrence

- In the most basic form, memory state are simply the hidden neurons



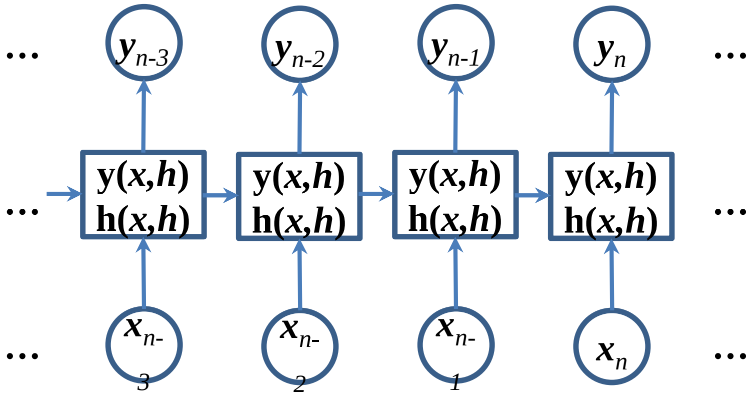


Types of analysis possible on sequential data using “recurrence”

- One to one
- One to many
- Many to one
- Many to many

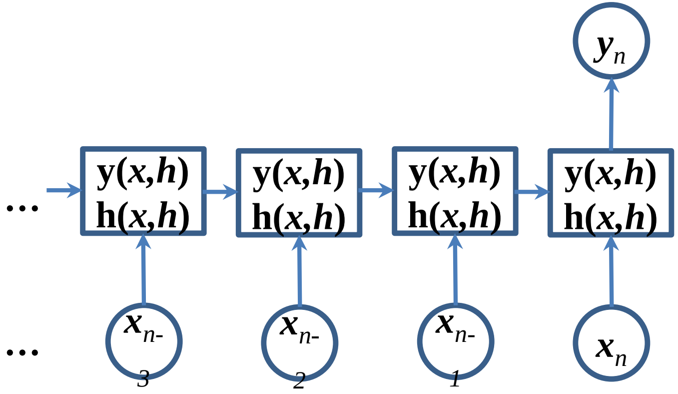
Examples: One to one

- POS tagging in NLP
- Stock trade: {Buy, NoAction, Sell}



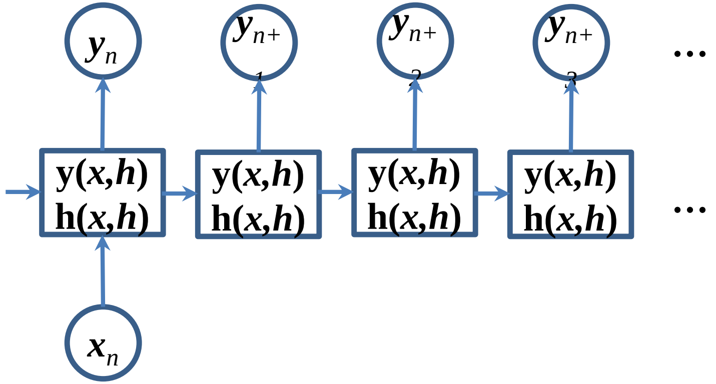
Examples: Many to one

- Sentiment analysis in NLP



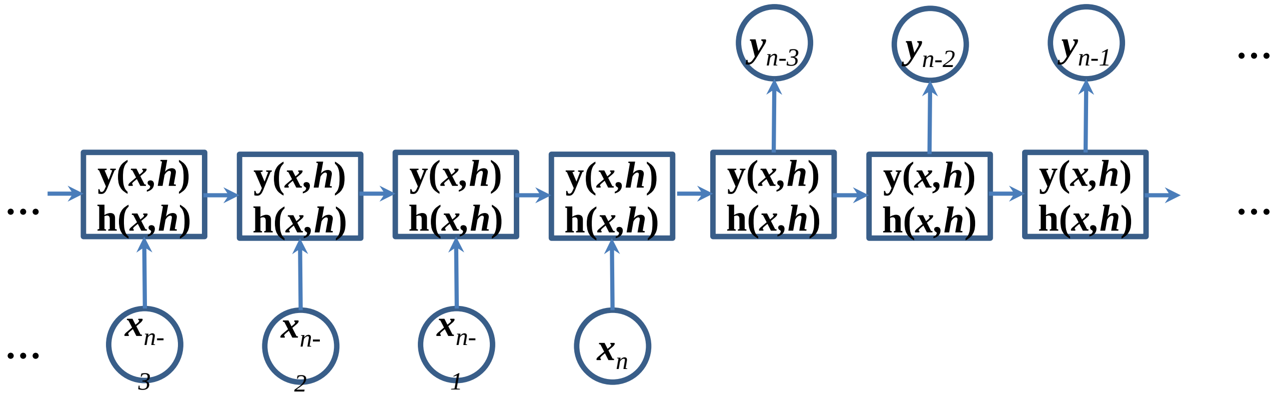
Examples: One to many

- Generate caption based on an image
- Generate text given topic



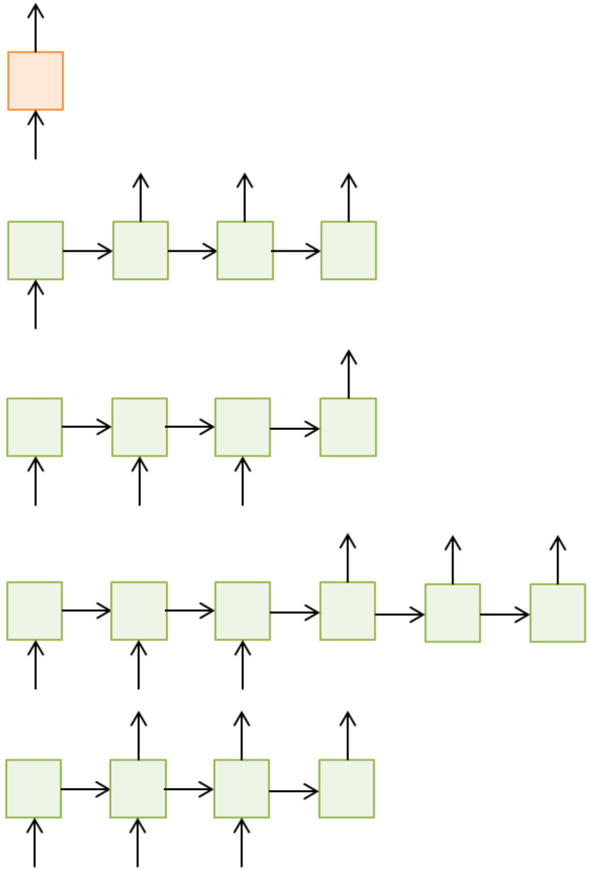
Examples: Many to many

- Language translation



Input – Output Scenarios

greatlearning
Learning for Life



Single - Single — Feed-forward Network

Single - Multiple — Image Captioning

Multiple - Single — Sentiment Classification

Multiple - Multiple — Translation

Image Captioning

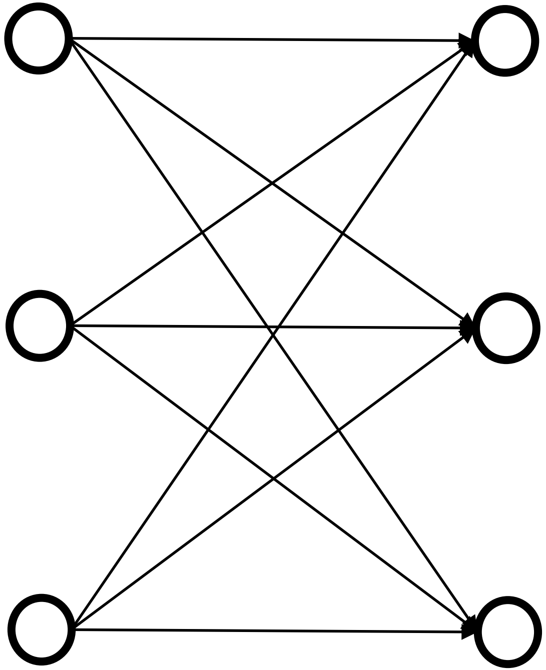
Recurrent Neural Networks

What’s for Dinner…?

Day of the week

Month of the year

Late Meeting

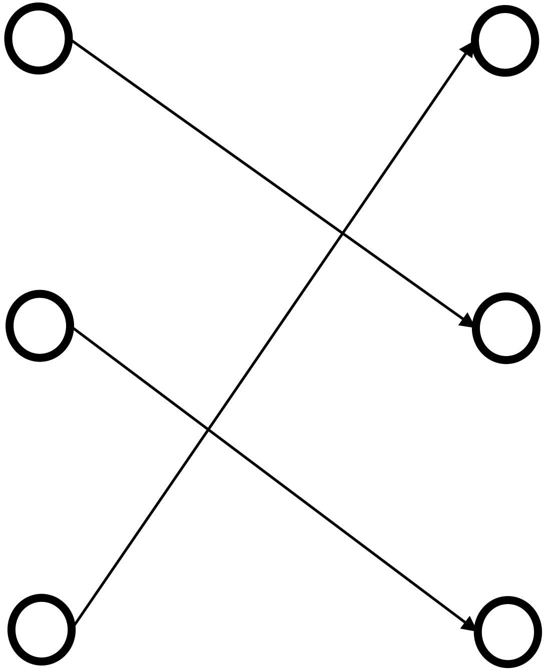


What’s for Dinner…?

Idly
Yesterday

Roti
Yesterday

Rice
Yesterday



What’s for Dinner…?

Predicted idly for yesterday

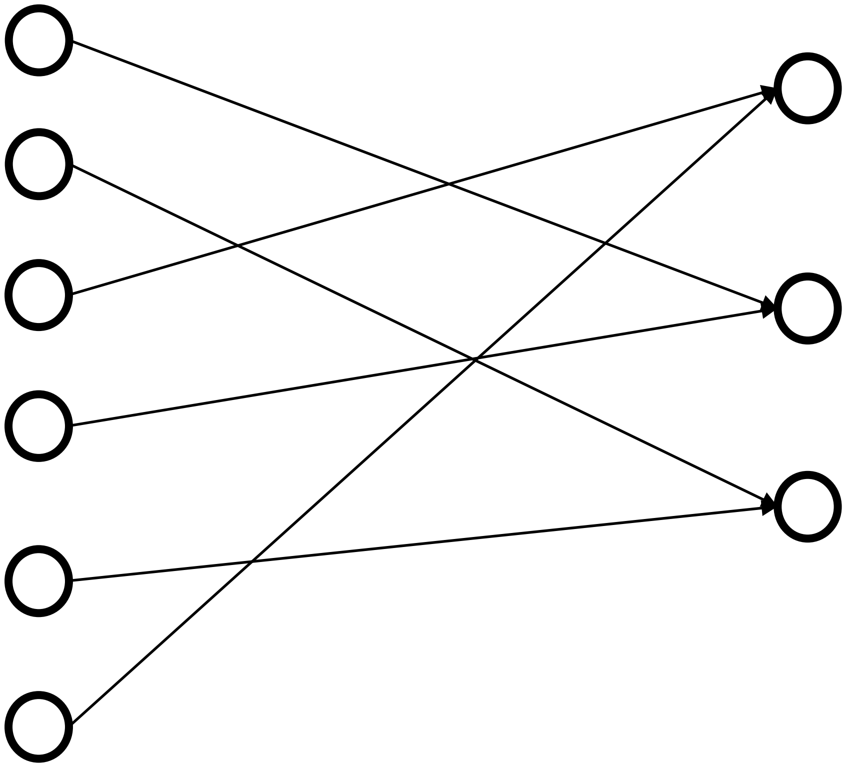
Predicted roti for yesterday

Predicted rice for yesterday

Idly Yesterday

Roti Yesterday

Rice Yesterday



What's for Dinner…?

Predicted idly for yesterday

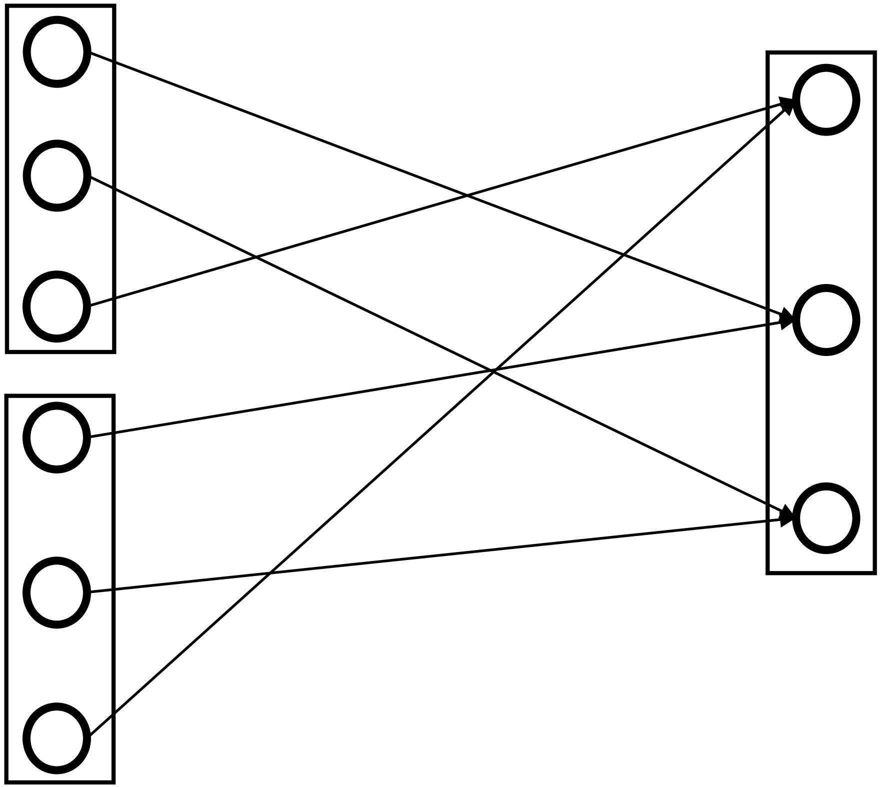
Predicted roti for yesterday

Predicted rice for yesterday

Idly Yesterday

Roti Yesterday

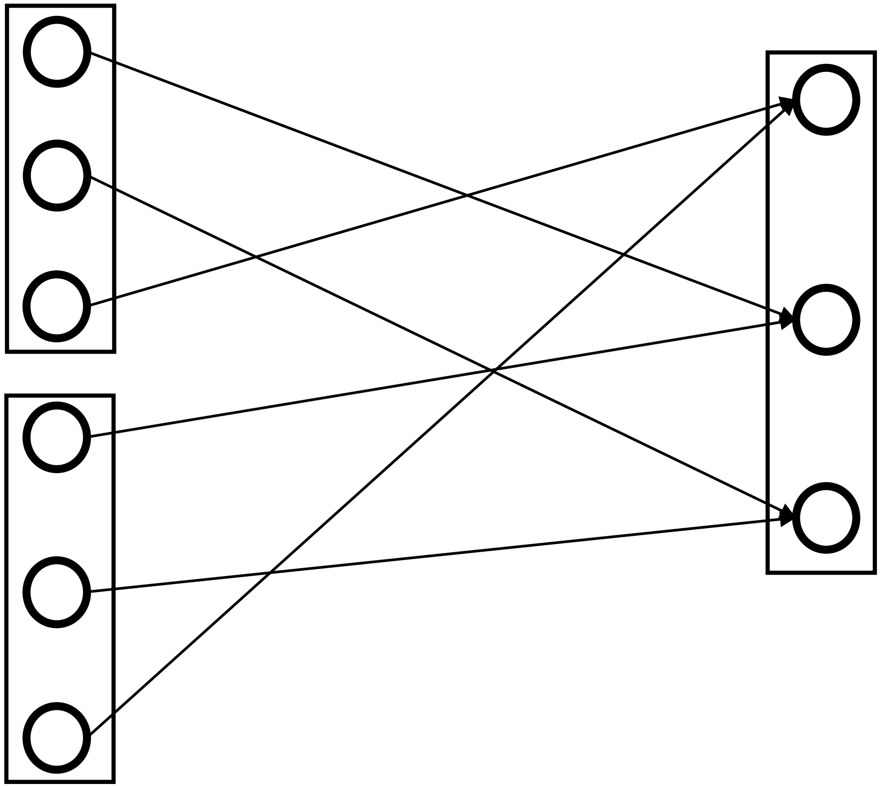
Rice Yesterday



What’s for Dinner…?

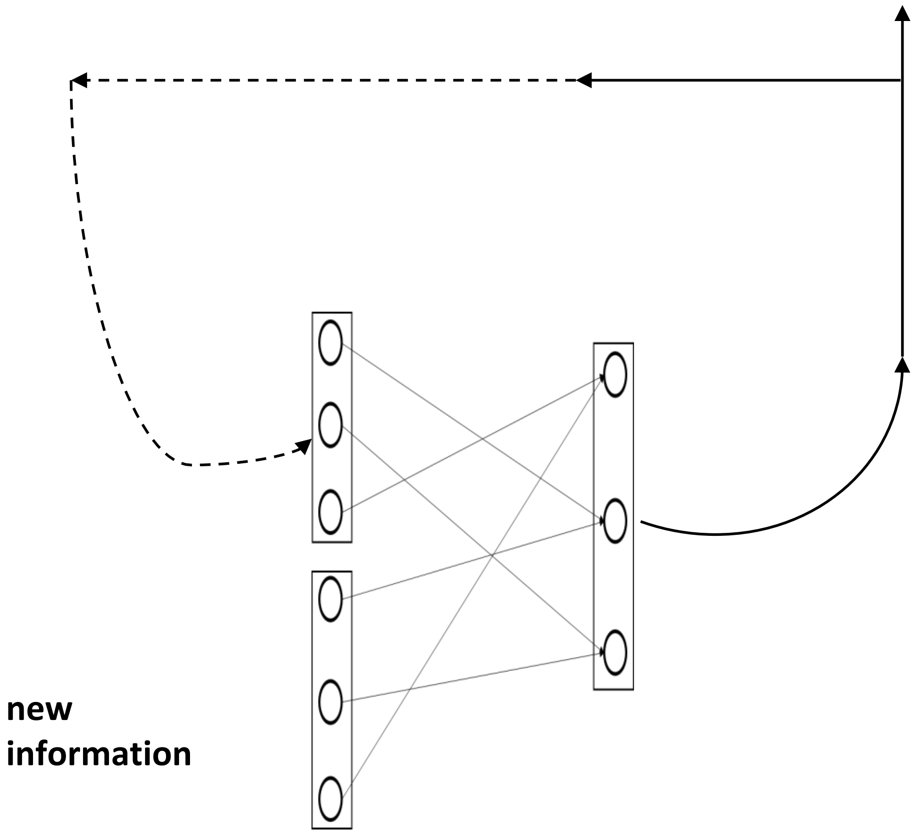
Predictions for
Yesterday

Dinner
Yesterday

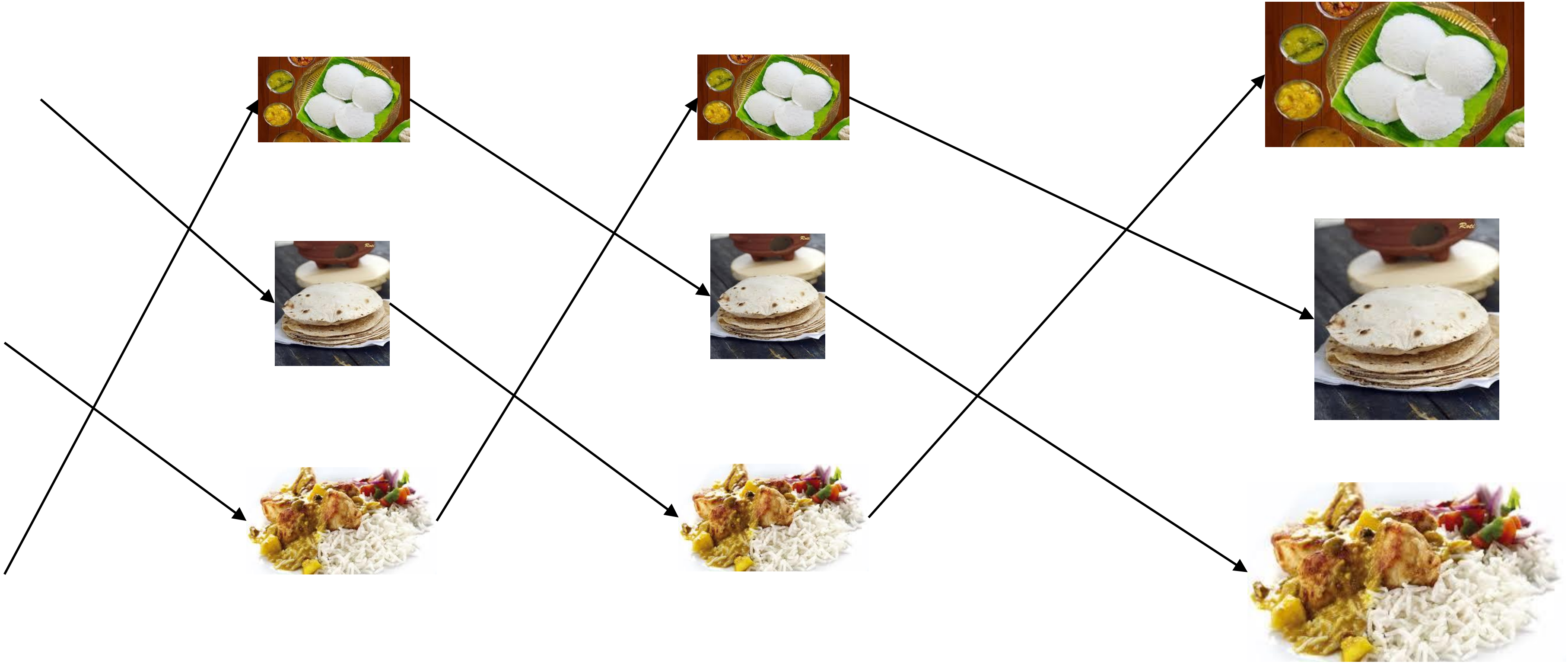


Predictions for
Today

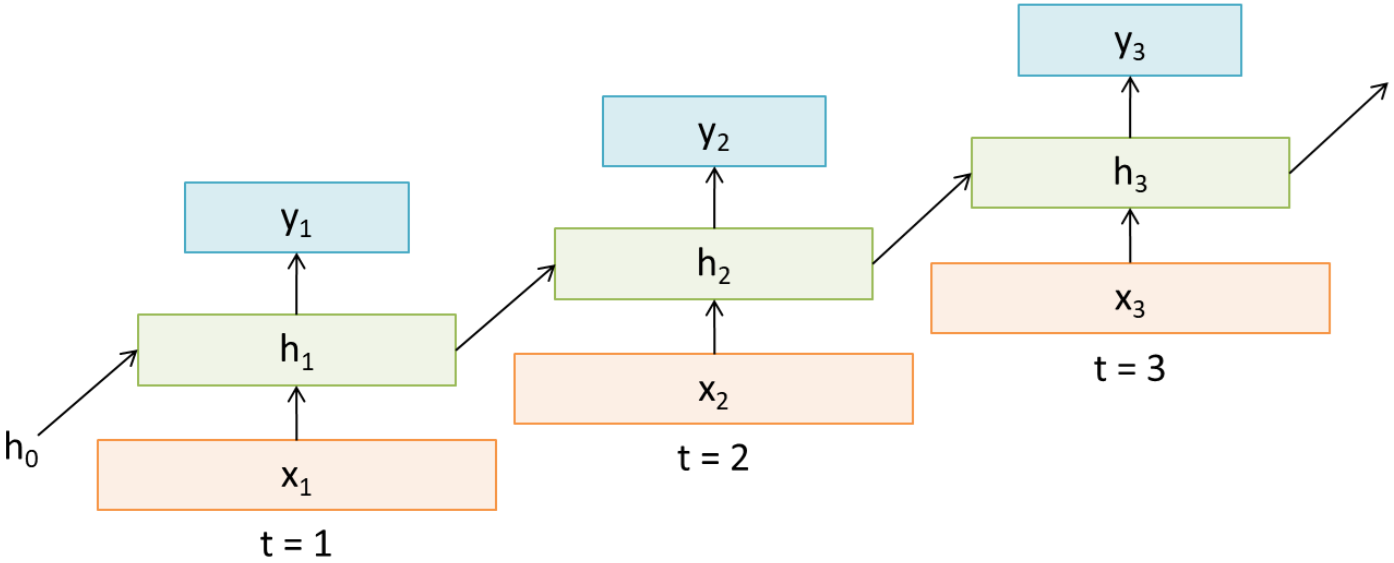
What's for Dinner…?



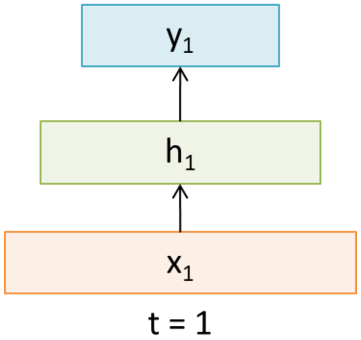
Unrolled predictions…



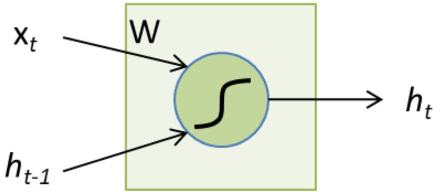
Sample RNN



Sample Feed Forward Network

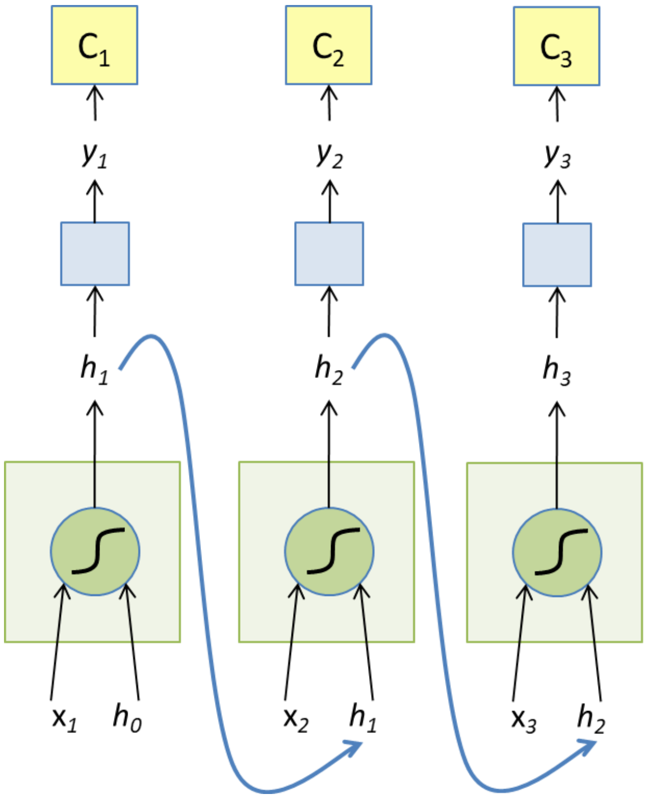


Vanilla RNN Cell



$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

Vanilla RNN Forward

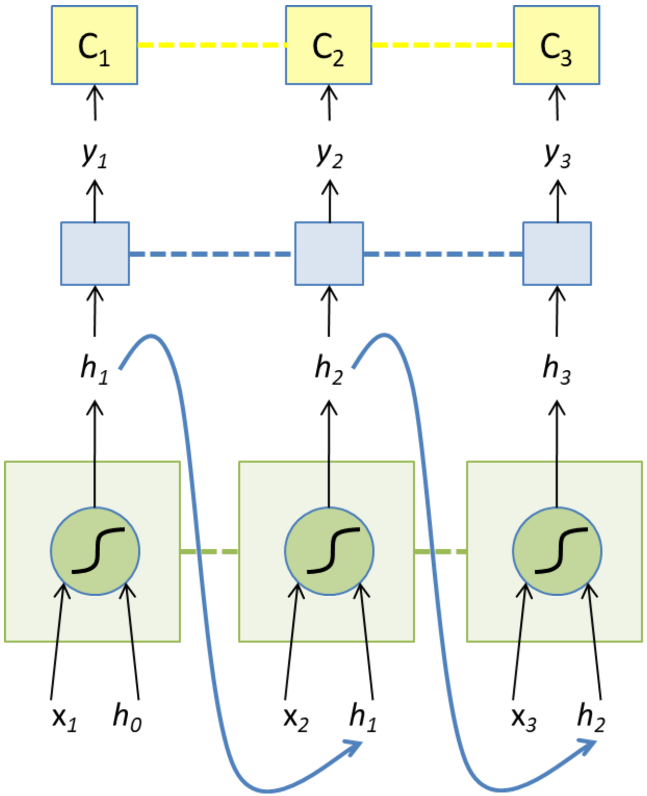


$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$y_t = \mathrm{F}(h_t)$$

$$C_t = \mathrm{Loss}(y_t, \mathrm{GT}_t)$$

Vanilla RNN Forward



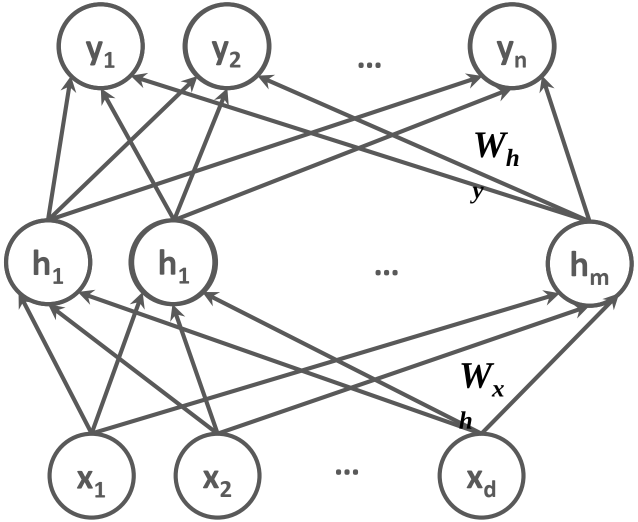
$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

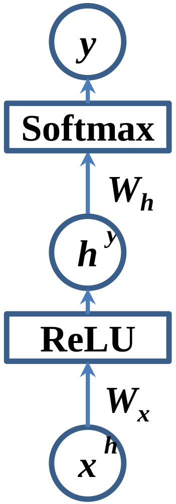
$$y_t = \mathrm{F}(h_t)$$

$$C_t = \mathrm{Loss}(y_t, \mathrm{GT}_t)$$

------ indicates shared weights

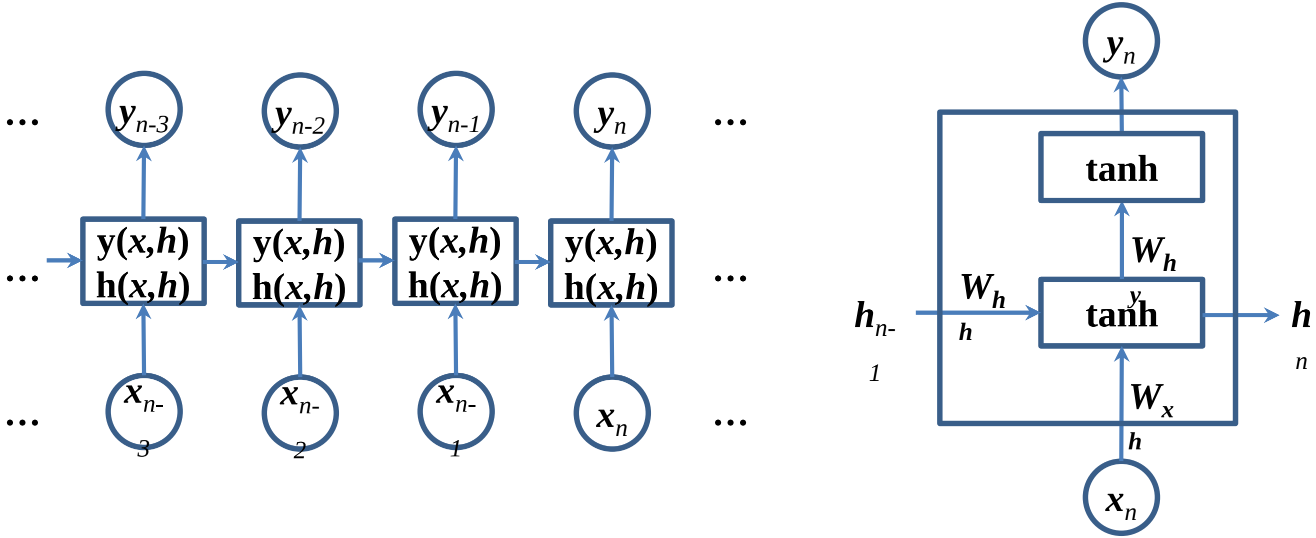
Revising feedforward neural networks





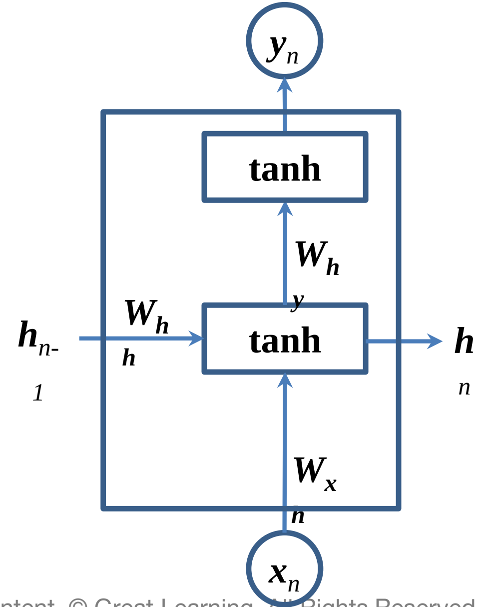
Recurrent neural networks

- Vanilla RNNs used the hidden layer activation as a state

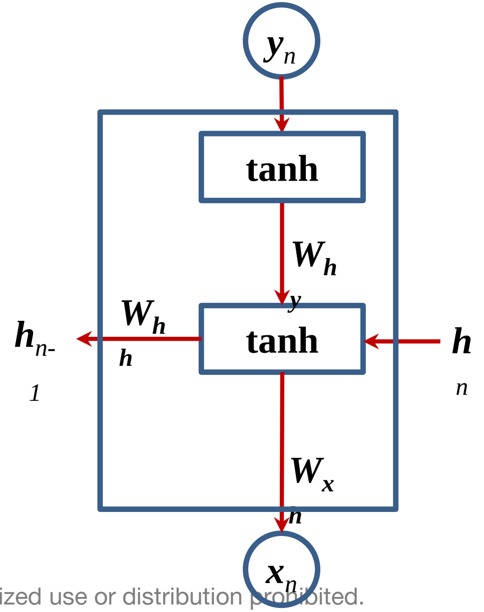


Backpropagation through time (BPTT)

- Just like how forward propagation uses previous state …

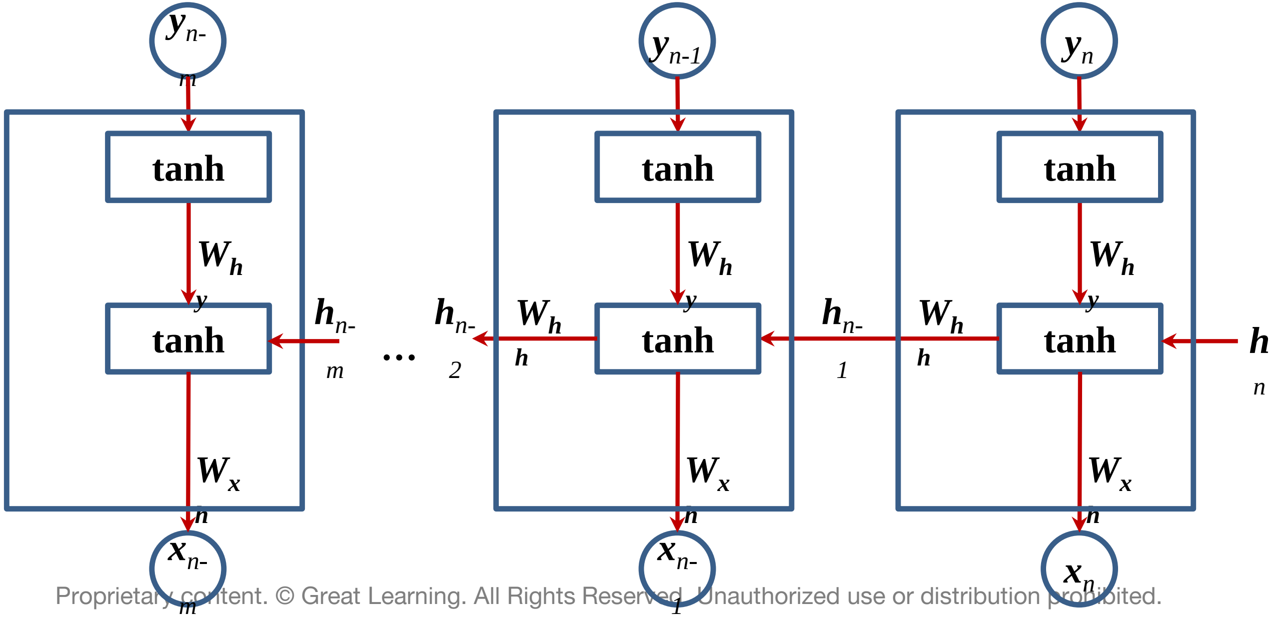


- Backpropagation uses derivative from future output



Use of a window length

- We need to put a limit on how long will the gradient travel back in time



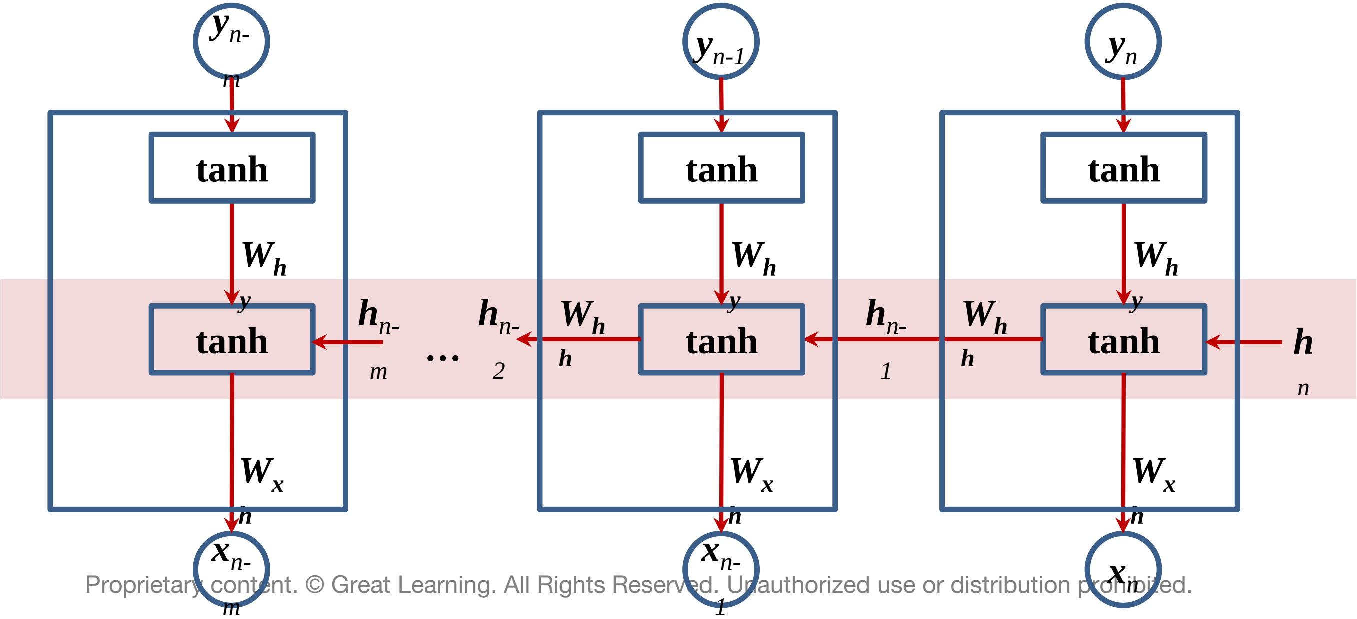
Mathematical expression for BPTT

- Forward: $\boldsymbol{y}_n = \boldsymbol{g}\left(\boldsymbol{W}_{hy}\boldsymbol{h}_n\right)$
$$= \boldsymbol{g}\left(\boldsymbol{W}_{hy}\boldsymbol{f}(\boldsymbol{W}_{hh}\boldsymbol{h}_{n-1} + \boldsymbol{W}_{xh}\boldsymbol{x}_n)\right)$$

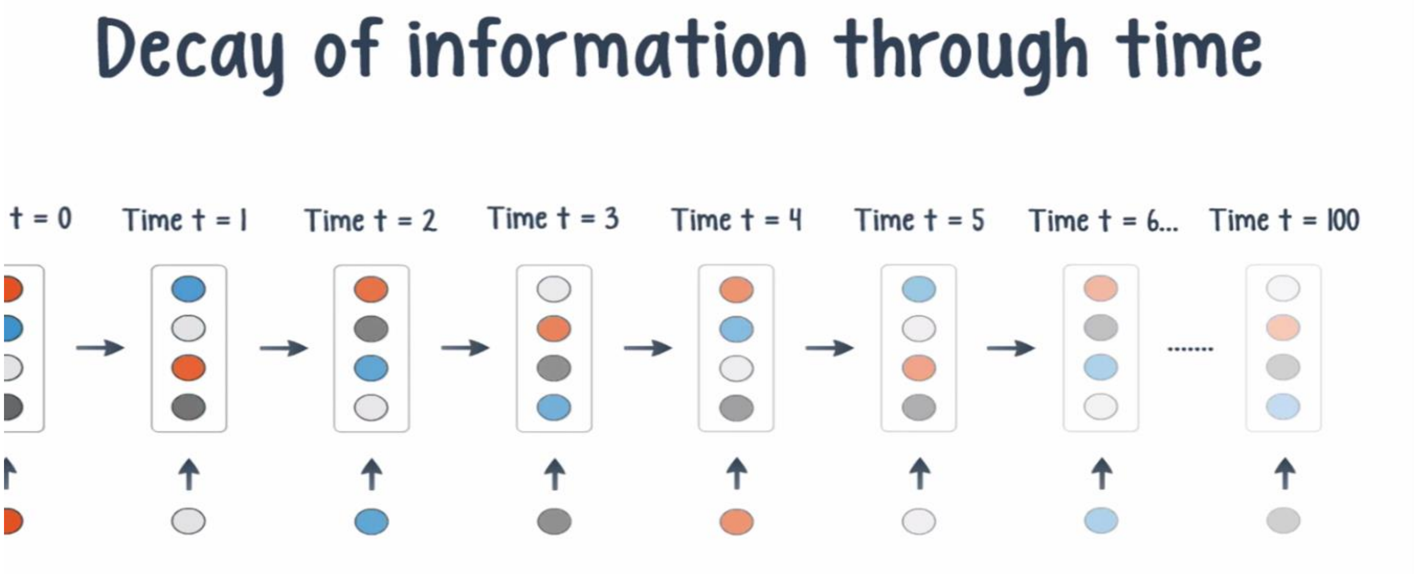
- Backward example:
$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{W}_{hh}} = \boldsymbol{g}'\boldsymbol{W}_{hy}\boldsymbol{h}_n{}' = \boldsymbol{g}'\boldsymbol{W}_{hy}\boldsymbol{f}'(\boldsymbol{h}_{n-1} + \boldsymbol{W}_{hh}\boldsymbol{h}_{n-1}{}')$$

Vanishing and exploding gradient

- Gradient gets repeatedly multiplied by W_{hh}
- This can lead to vanishing or exploding gradient depending on the norm of W_{hh}



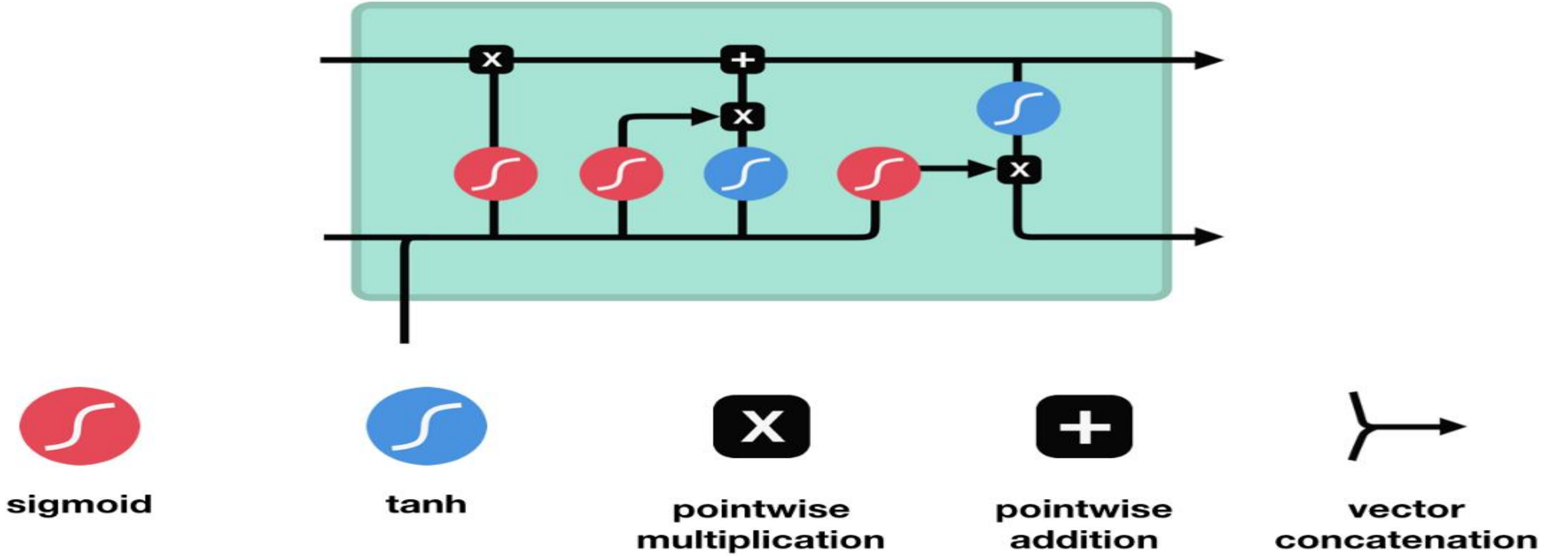
Why LSTMs?


Decay of information through time
t = 0
Time t = 1
Time t = 2
Time t = 3
Time t = 4
Time t = 5
Time t = 6...
Time t = 100

Intro to LSTMs

An LSTM has a similar control flow as a recurrent neural network. It processes data passing on information as it propagates forward. The differences are the operations within the LSTM's cells.

LSTM Architecture

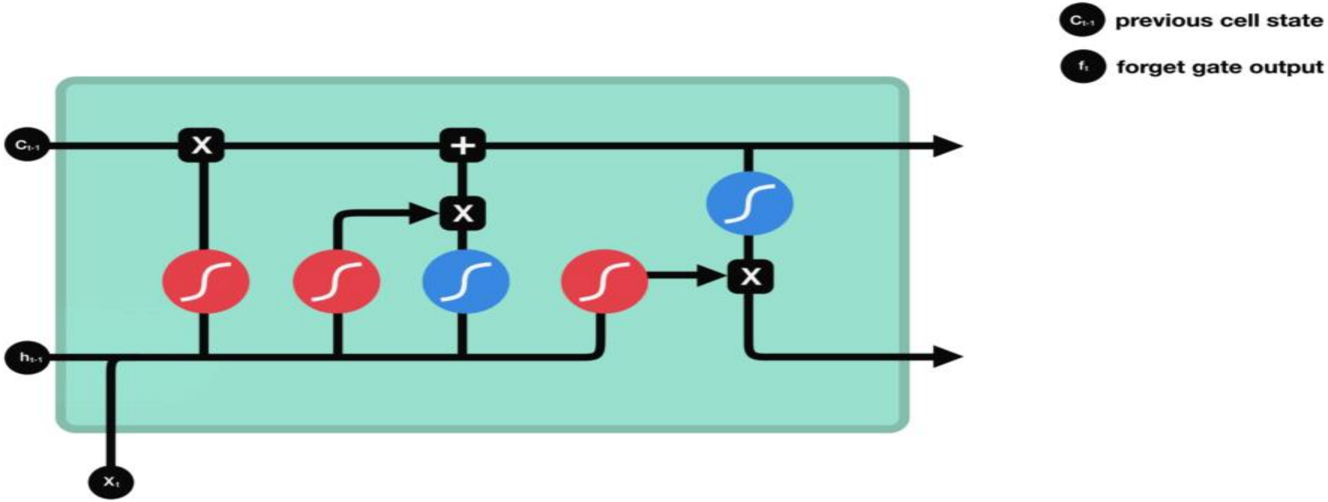


Core Concepts of LSTMs

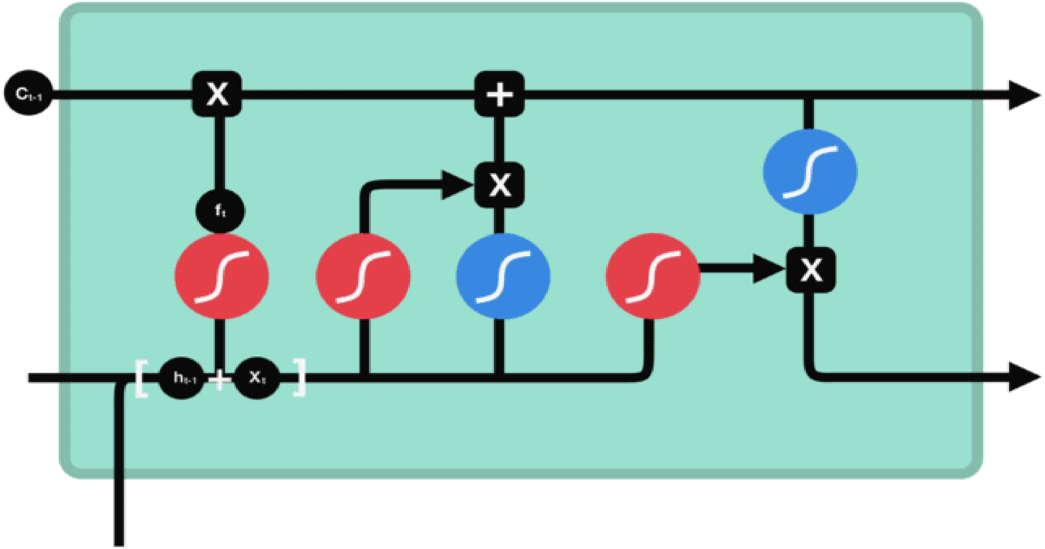
- Sigmoid
- Forget Gate
- Input Gate
- Cell State
- Output Gate

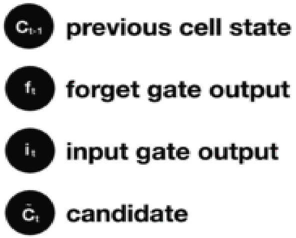
Forget Gate

This gate decides what information should be thrown away or kept. Information from the previous hidden state and information from the current input is passed through the sigmoid function. Values come out between 0 and 1. The closer to 0 means to forget, and the closer to 1 means to keep.



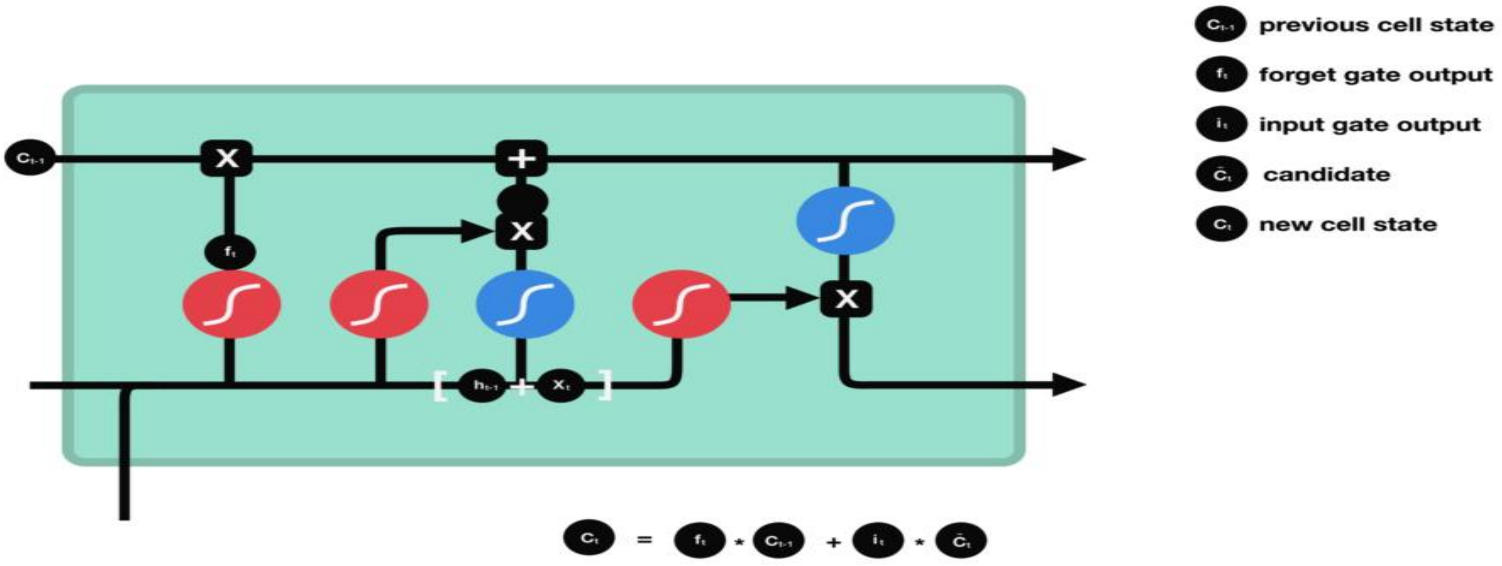
Input Gate


C_{t-1}
f_t
h_{t-1}
x_t
X
+
X
X


C_{t-1} previous cell state
f_t forget gate output
i_t input gate output
$\tilde{C}_t$ candidate

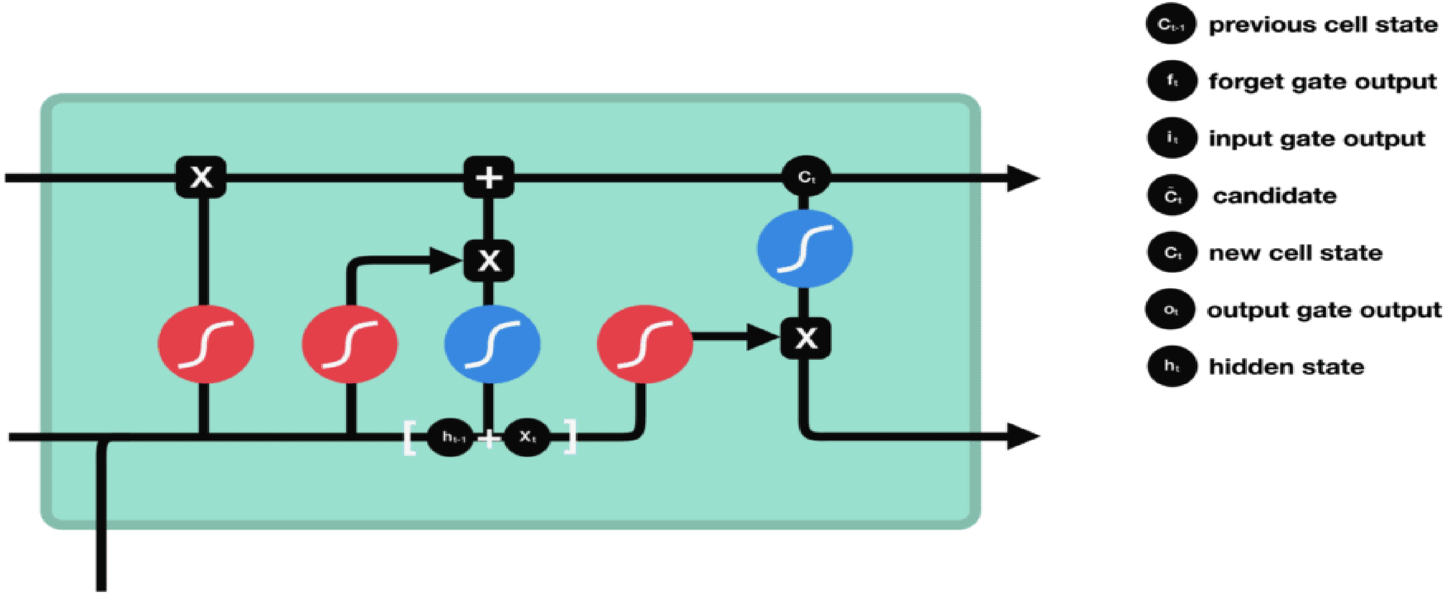
Cell State

Now we should have enough information to calculate the cell state. First, the cell state gets pointwise multiplied by the forget vector. This has a possibility of dropping values in the cell state if it gets multiplied by values near 0. Then we take the output from the input gate and do a pointwise addition which updates the cell state to new values that the neural network finds relevant. That gives us our new cell state.



Output Gate

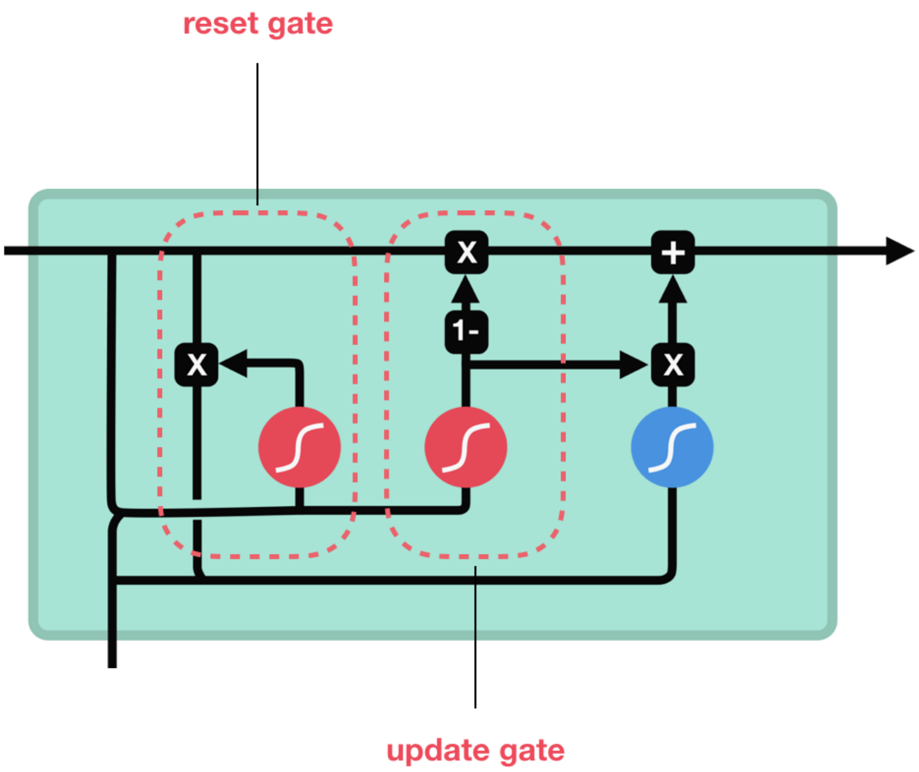
The output gate decides what the next hidden state should be. Remember that the hidden state contains information on previous inputs. The hidden state is also used for predictions.



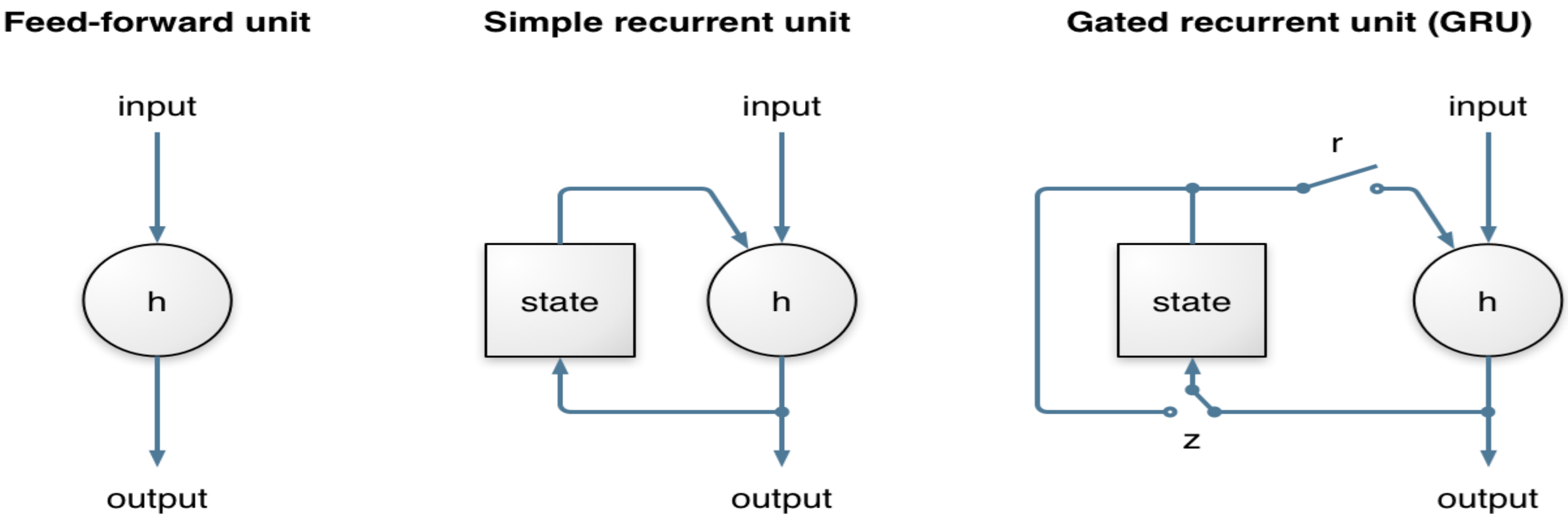
In Summary

To review, the **Forget gate** decides what is relevant to keep from prior steps. The **input gate** decides what information is relevant to add from the current step. The **output gate** determines what the next hidden state should be.

GRU

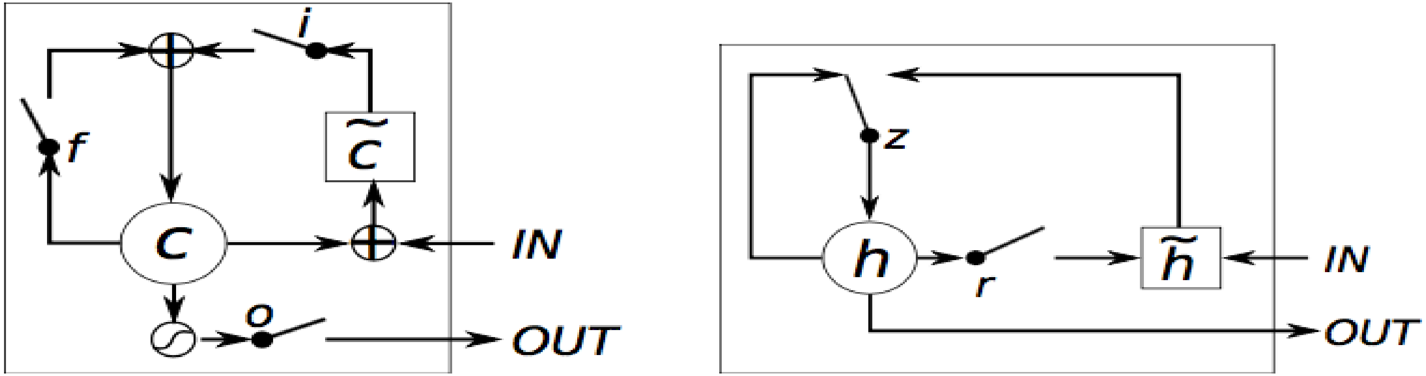


More on GRUs



reference -https://people.xiph.org/~jm/demo/rnnoise/

GRUs and LSTMs



(a) Long Short-Term Memory

(b) Gated Recurrent Unit

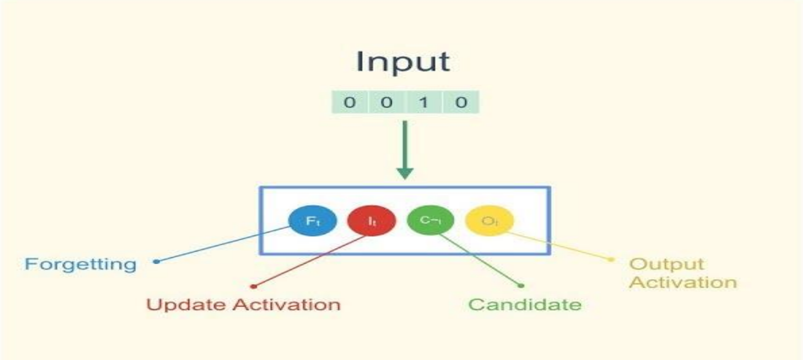
Figure 1: Illustration of (a) LSTM and (b) gated recurrent units. (a) i, f and o are the input, forget and output gates, respectively. c and $\tilde{c}$ denote the memory cell and the new memory cell content. (b) r and z are the reset and update gates, and h and $\tilde{h}$ are the activation and the candidate activation.

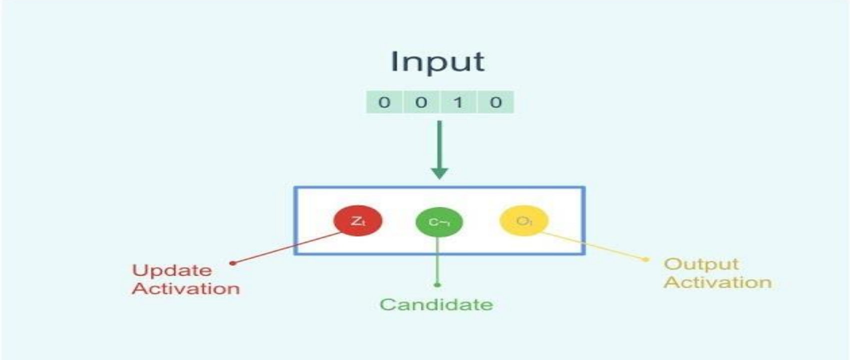
reference -https://medium.com/paper-club/grus-vs-lstms-e9d8e2484848

More Flexible - LSTM

Less Parameters - GRU

LSTMs and GRUs





references -https://www.youtube.com/watch?v=4F69m3krMHw

Understanding LSTMs

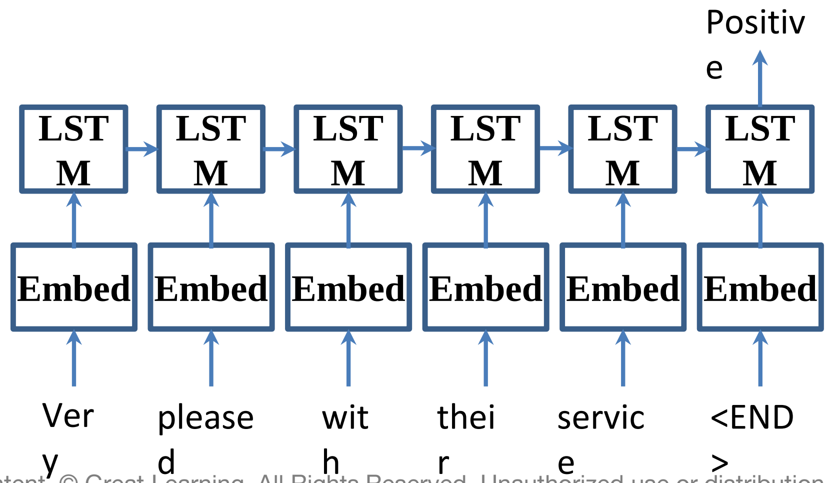
Refer - http://colah.github.io/posts/2015-08-Understanding-LSTMs/

https://medium.com/datathings/the-magic-of-lstm-neural-networks-6775e8b540cd

Some Applications of LSTM in NLP

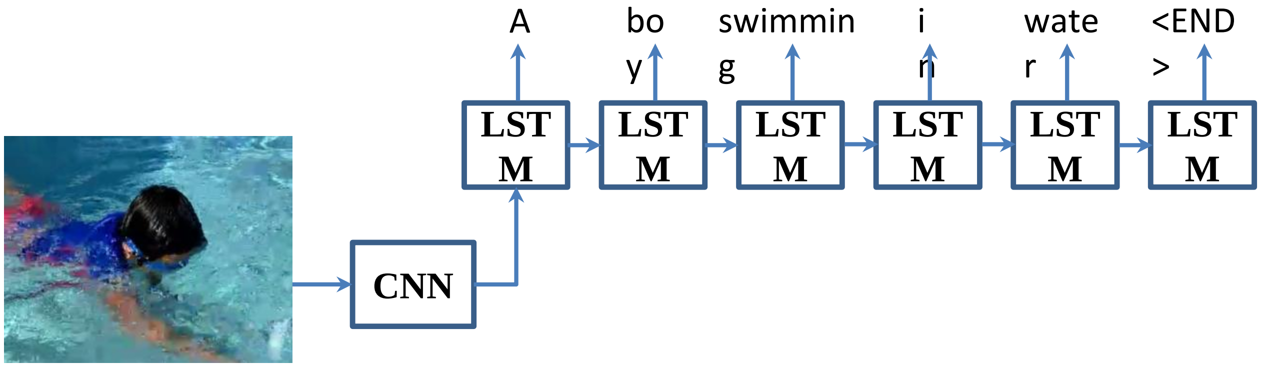
Sentiment analysis

- Very common for customer review or new article analysis
- Output before the end can be discarded (not used for backpropagation)
- This is a many-to-one task



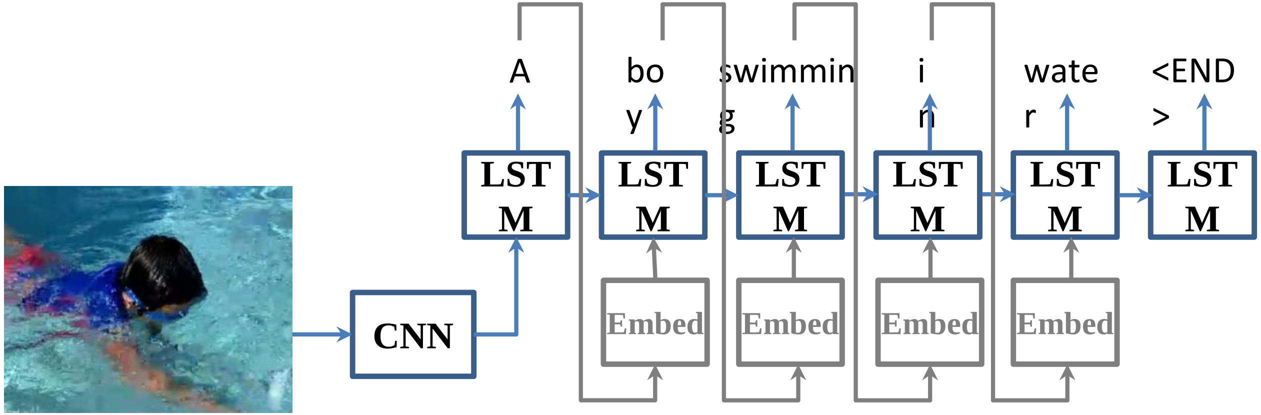
Sentence generation

- Very common for image captioning
- Input is given only in the beginning
- This is a one-to-many task



Sentence generation

- Very common for image captioning
- Input is given only in the beginning
- This is a one-to-many task



Time Series Prediction with LSTMs

What is time series data?

A sequence of vectors (or scalars) which depend on time t. In this lecture we will deal exclusively with scalars:

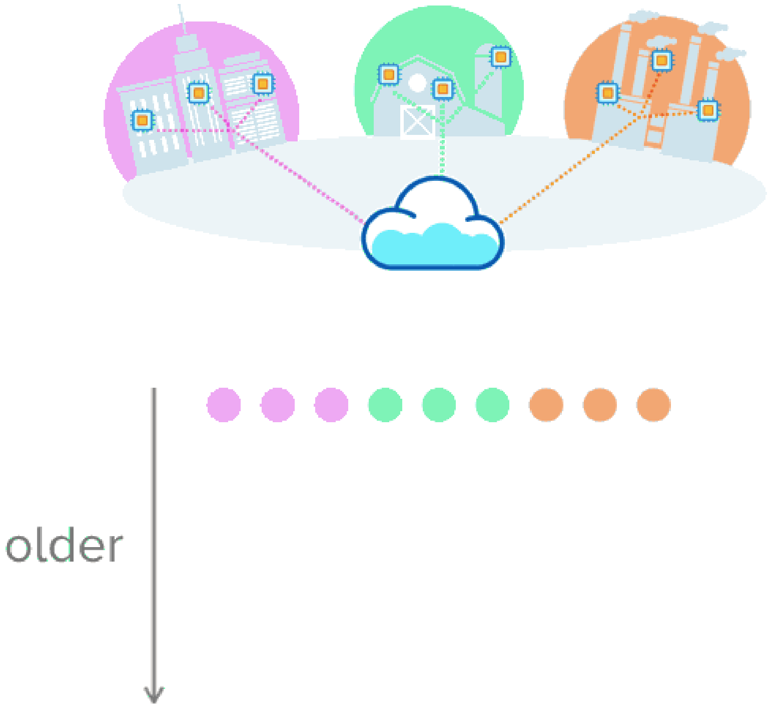
$$\{\ x(t_0),\ x(t_1),\ \cdots\ x(t_{i-1}),\ x(t_i),\ x(t_{i+1}),\ \cdots\ \}$$

It's the output of some process P that we are interested in:

$$\boxed{P} \longrightarrow x(t)$$

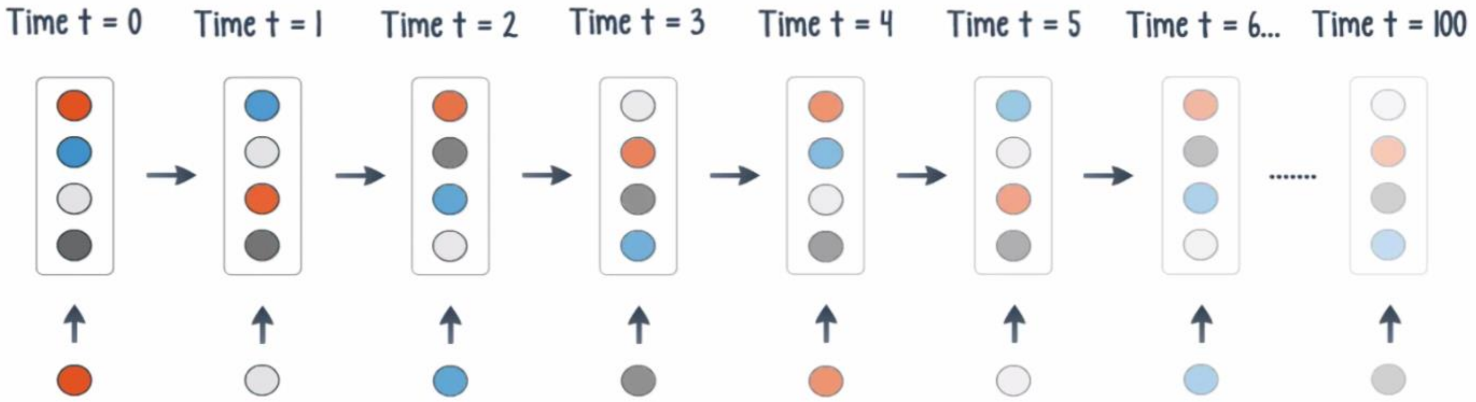
Example of Time series data

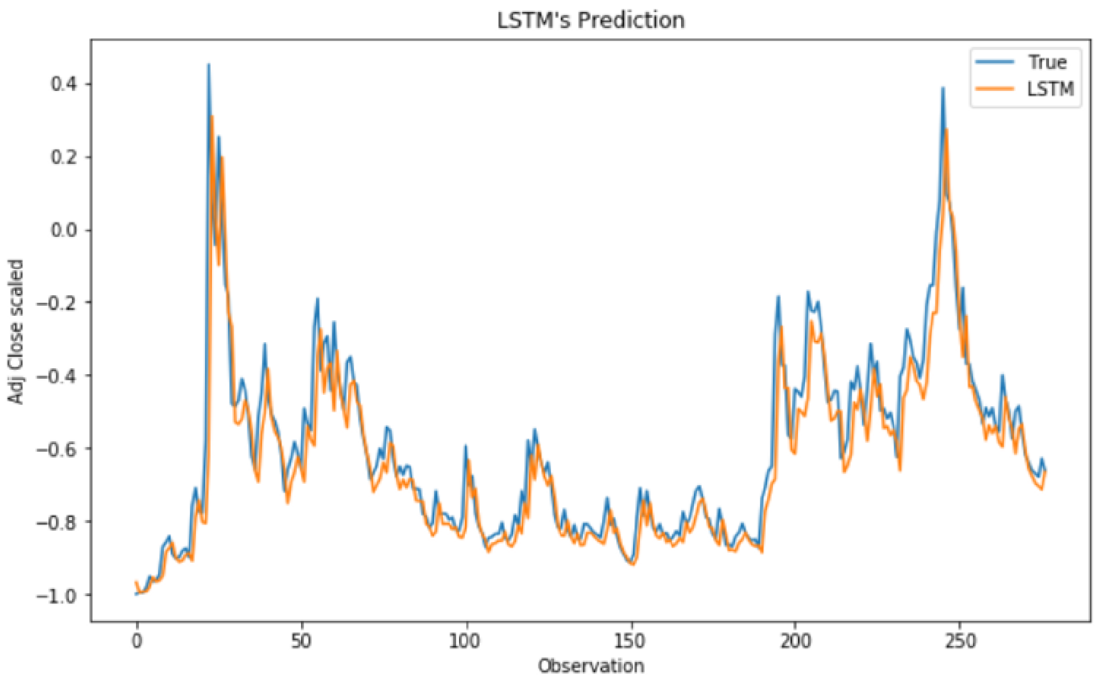
- the prices of stocks and shares taken at regular intervals of time
- the temperature reading taken at your house at hourly intervals
- the number of cases of a disease in town taken at daily intervals
- No of births in a community
- Electricity demand for a city etc



Here’s a basic illustration. Imagine sensors collecting data from three settings: a city, farm, and factory. In this example, each of these sources periodically sends new readings, creating a series of measurements collected over time.

Decay of information through time




LSTM's Prediction
True
LSTM
Adj Close scaled
Observation
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
0
50
100
150
200
250

In particular, deep learning techniques hold great promise for time series analysis. As time series become more dense and begin to overlap, machine learning offers a way to separate the signal from the noise. Deep learning holds potential because it is often the best fit for the seemingly random nature of financial time series.

Let’s go the python notebooks!