

Deep Learning – Recurrent Neural Networks (RNN)

By,

Divesh R. Kubal

Data Scientist, eClerx Services

Center of Excellence – Machine Learning



Table of Contents

1. Time Series
2. Sequence Problems
3. Word Embeddings
 - A. CBOW
 - B. Skip-gram
4. Recurrent Neural Nets
5. Long Short Term Memory

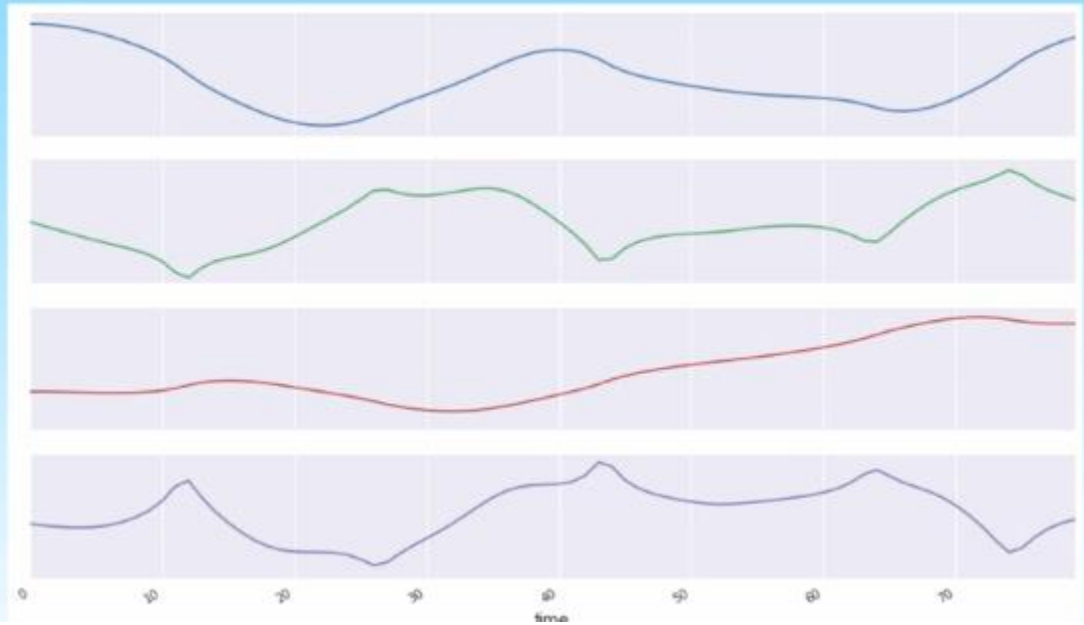


Github Link

- <https://github.com/DiveshRKubal/GreyAtom-Deep-Learning/tree/master/GreyAtom-Deep-Learning/RNN>

Time Series

- Vector sequence



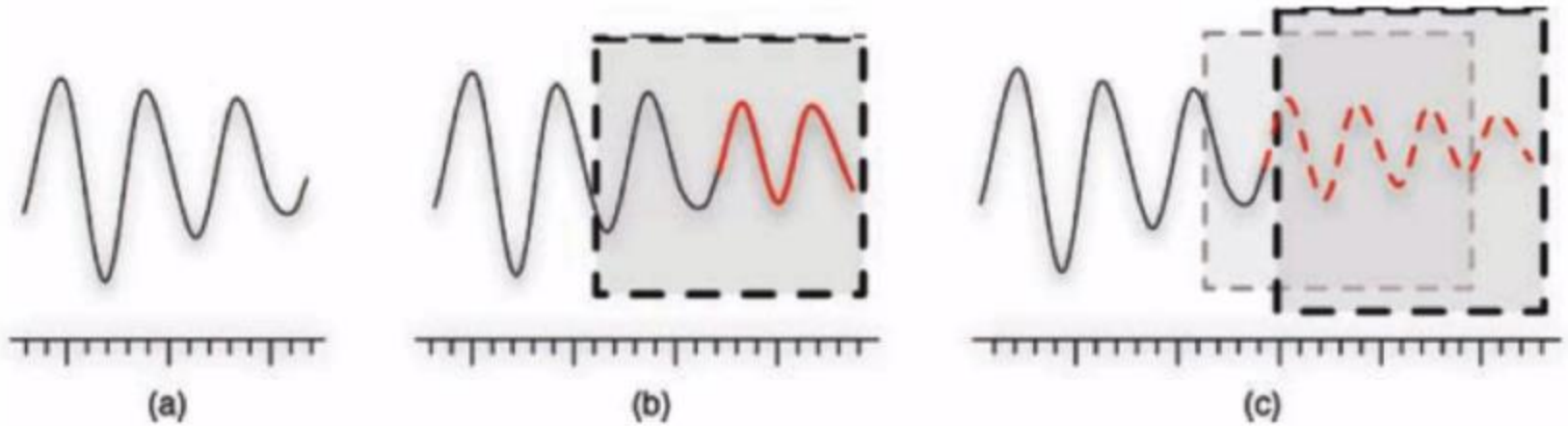


ML for Time Series

- Prediction of future (regression, forecasting)
- Pattern recognition & segmentation (classification, clustering, anomaly detection)
- Compression, noise reduction (preprocessing)

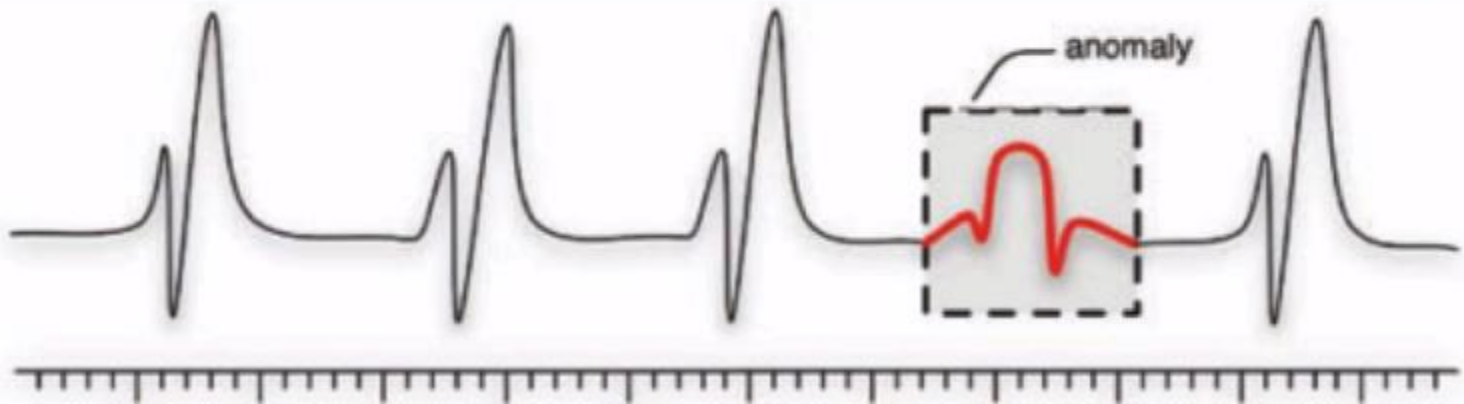
Prediction

Use past to predict the future



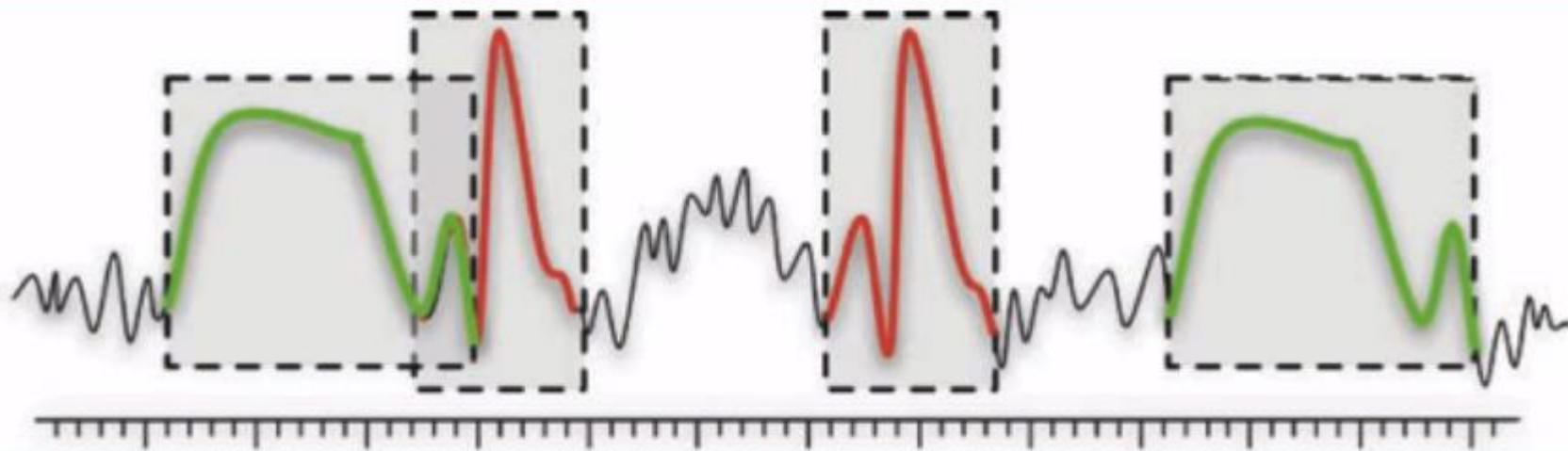
Prediction

Detect deviation from standard behavior



Prediction

Find recurring patterns



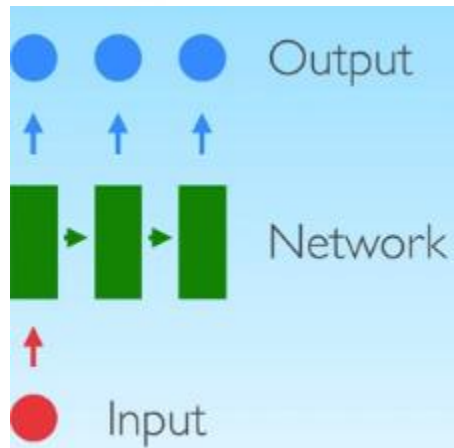
Sequence Problems

One to One



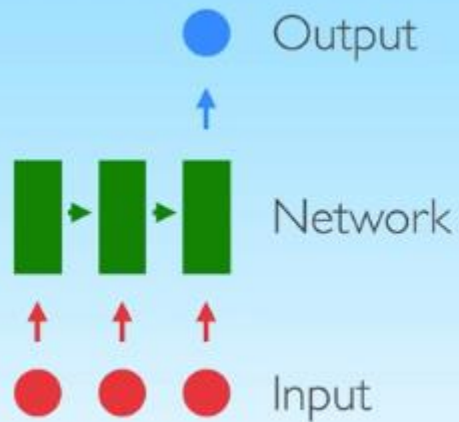
- Point-wise Forecasting
- Classification (fixed input/output size)

One to Many



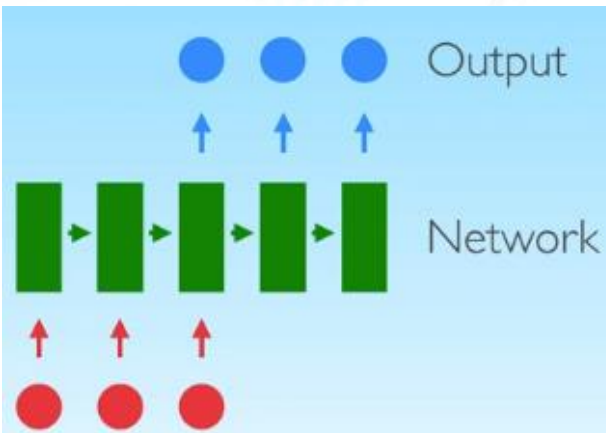
- Sequence output from single input
- e.g. image captioning

Many to One



- Sequence input, single output
- e.g. sentiment analysis from text

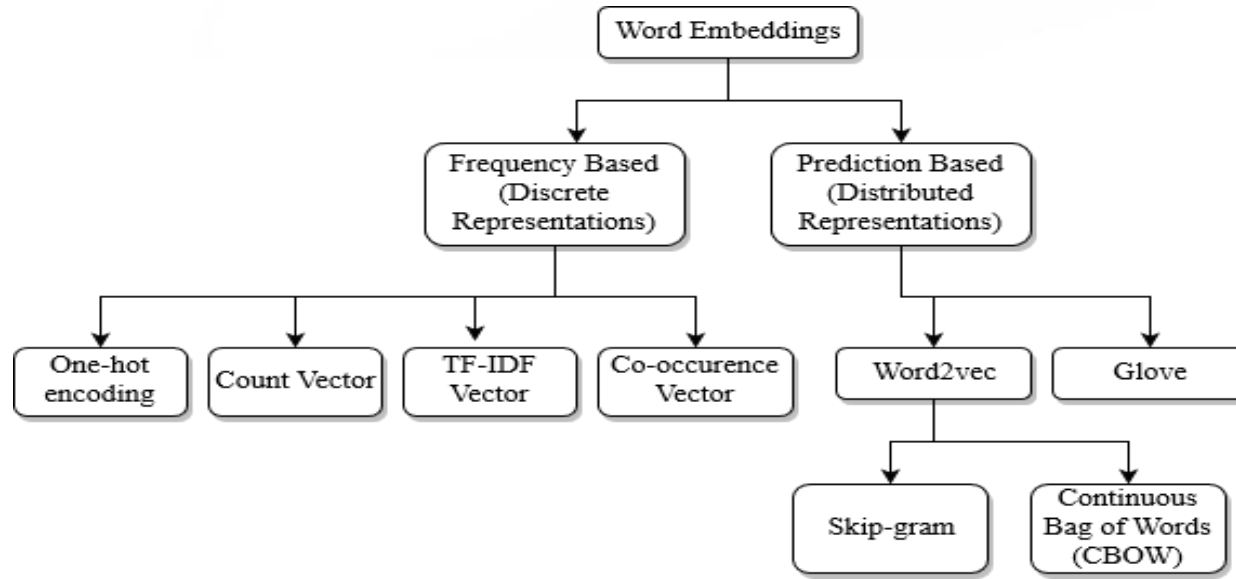
Many to Many



- Sequence input, sequence output
- e.g. text translation

Word Embeddings

- Word Embeddings are the texts converted into numbers and there may be different numerical representations of the same text.



- Predict the probability of a word given a context.
- A context may be a single word or a group of words.
- C = “Hey, this is sample corpus using only one context word.” and we have defined a context window of 1

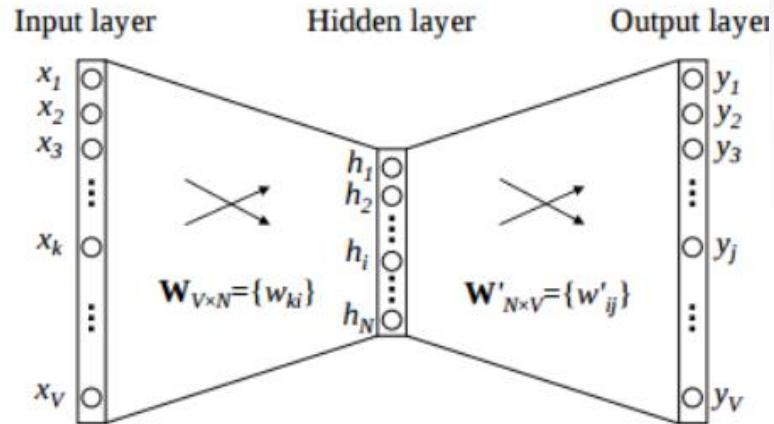


Sample one-hot encoded Matrix

[illegible]

- This matrix shown in the image is sent into a shallow neural network with three layers:
 - an input layer,
 - a hidden layer and,
 - an output layer.
- The output layer is a *softmax* layer which is used to sum the probabilities obtained in the output layer to 1.

Diagrammatic representation of the CBOW model



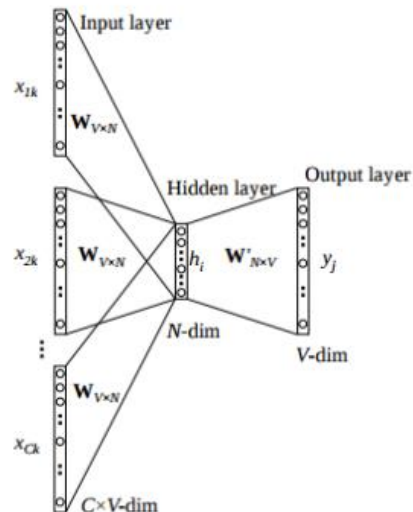
Context											Input-Hidden Weight				Hidden Activation			
C1	this	0	1	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8
											5	6	7	8				
											9	10	11	12				
											13	14	15	16				
											17	18	19	20				
											21	22	23	24				
											25	26	27	28				
											29	30	31	32				
											33	34	35	36				
											37	38	39	40				



Working of CBOW

- The input layer and the target, both are one-hot encoded of size $[1 \times V]$. Here $V=10$ in the above example.
- There are two sets of weights. One is between the input and the hidden layer and second between hidden and output layer.
Input-Hidden layer matrix size $= [V \times N]$, hidden-Output layer matrix size $= [N \times V]$: Where N is the number of dimensions we choose to represent our word in. It is arbitrary and a hyper-parameter for a Neural Network. Also, N is the number of neurons in the hidden layer. Here, $N=4$.
- There is no activation function between any layers. (More specifically, I am referring to linear activation)
- The input is multiplied by the input-hidden weights and called hidden activation. It is simply the corresponding row in the input-hidden matrix copied.
- The hidden input gets multiplied by hidden-output weights and output is calculated.
- Error between output and target is calculated and propagated back to re-adjust the weights.
- The weight between the hidden layer and the output layer is taken as the word vector representation of the word.

Multiple Context Words as I/P



Context									
C1	this	0	1	0	0	0	0	0	0
C2	corpus	0	0	0	0	1	0	0	0
C3	context	0	0	0	0	0	0	0	1

Input-Hidden Weight			
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16
17	18	19	20
21	22	23	24
25	26	27	28
29	30	31	32
33	34	35	36
37	38	39	40

Hidden Activation			
5	6	7	8
17	18	19	20
33	34	35	36
Average hidden Activation			
18.33333333	19.33333333	20.33333333	21.33333333



Advantages of CBOW

- Being probabilistic in nature, it is supposed to perform superior to deterministic methods (generally).
- It is low on memory. It does not need to have huge RAM requirements like that of co-occurrence matrix where it needs to store three huge matrices.



Disadvantages of CBOW

- CBOW takes the average of the context of a word (as seen above in calculation of hidden activation). For example, Apple can be both a fruit and a company but CBOW takes an average of both the contexts and places it in between a cluster for fruits and companies.
- Training a CBOW from scratch can take forever if not properly optimized.

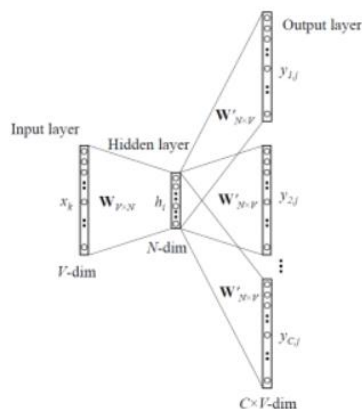


Skip – Gram model

- Aim of skip-gram is to predict the context given a word.
- C="Hey, this is sample corpus using only one context word."

Input	Output(Context1)	Output(Context2)
Hey	this	<padding>
this	Hey	is
is	this	sample
sample	is	corpus
corpus	sample	corpus
using	corpus	only
only	using	one
one	only	context
context	one	word
word	context	<padding>

Skip-gram architecture



Hidden Activation

This

0.12	0.10	0.14	0.15	0.16	0.17	0.18	0.19	0.2	0.21
0.22	0.23	0.24	0.25	0.26	0.27	0.28	0.29	0.3	0.31
0.32	0.33	0.34	0.35	0.36	0.37	0.38	0.39	0.4	0.41
0.42	0.43	0.44	0.45	0.46	0.47	0.48	0.49	0.5	0.51

Hidden output weight matrix

output

7.52	7.79	6.94	6.3	6.26	6.82	5.88	6.34	9.6	9.86
7.52	7.79	6.94	6.3	6.26	6.82	5.88	6.34	9.6	9.86

softmax probabilities

0.024	0.03009744	0.04007152	0.05197034	0.0674015	0.07401555	0.10337114	0.14793512	0.19069461	0.24731156
0.024	0.03009744	0.04007152	0.05197034	0.0674015	0.07401555	0.10337114	0.14793512	0.19069461	0.24731156

Target

1	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0

Error

-0.55	0.03009744	0.04007152	0.05197034	0.0674015	0.07401555	0.10337114	0.14793512	0.19069461	0.24731156
0.024	0.03009744	-0.0099282	0.05197034	0.0674015	0.07401555	0.10337114	0.14793512	0.19069461	0.24731156

Squared error

-0.02	0.0170407	-0.0000614	0.00204019	0.0040321	0.0040321	0.0040321	0.0040321	0.0040321	0.0040321
-------	-----------	------------	------------	-----------	-----------	-----------	-----------	-----------	-----------



Working of Skip-gram

In the above example, C is the number of context words=2, $V=10$, $N=4$

- The row in red is the hidden activation corresponding to the input one-hot encoded vector. It is basically the corresponding row of input-hidden matrix copied.
- The yellow matrix is the weight between the hidden layer and the output layer.
- The blue matrix is obtained by the matrix multiplication of hidden activation and the hidden output weights. There will be two rows calculated for two target(context) words.
- Each row of the blue matrix is converted into its *softmax* probabilities individually as shown in the green box.
- The grey matrix contains the one hot encoded vectors of the two context words(target).
- Error is calculated by subtracting the first row of the grey matrix(target) from the first row of the green matrix(output) element-wise. This is repeated for the next row. Therefore, for n target context words, we will have n error vectors.
- Element-wise sum is taken over all the error vectors to obtain a final error vector.
- This error vector is propagated back to update the weights



Advantages of Skip-Gram Model

- Skip-gram model can capture two semantics for a single word. i.e it will have two vector representations of Apple. One for the company and other for the fruit.
- Skip-gram with negative sub-sampling outperforms every other method generally.



Word Embeddings use case scenarios

1. Finding the degree of similarity between two words.

```
model.similarity('woman','man')
```

```
0.73723527
```

2. Finding odd one out.

```
model.doesnt_match('breakfast cereal dinner lunch'.split())
```

```
'cereal'
```

3. Amazing things like woman+king-man =queen

```
model.most_similar(positive=['woman','king'],negative=['man'],topn=1)
```

```
queen: 0.508
```

4. Probability of a text under the model

```
model.score(['The fox jumped over the lazy dog'.split()])
```

```
0.21
```

Word Embedding Approaches Comparison

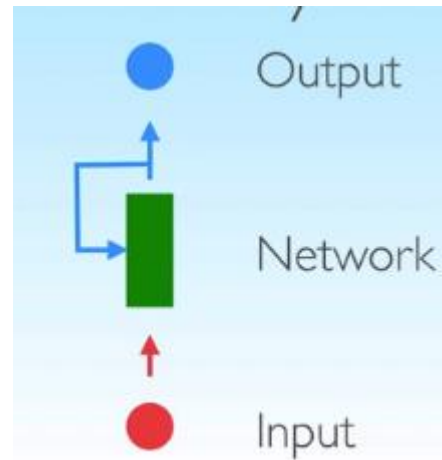
Type		Advantages	Limitations
Frequency Based (Discrete Word Representations)	One-Hot Encoding	Easy to compute. Low computational time to generate.	Faces curse of dimensionality as the number of vocabulary increases. Does not capture context and semantics
	Count Vector	Simple to implement. Takes a document in consideration.	Does not consider full corpus at a time. Fails to compute semantics and context.
	TF-IDF vector	Takes full corpus in consideration. Easy computation of similarity between document pair.	Based on CBOW, hence does not capture context. Also fails to capture semantic meaning.
	Co-occurrence vector	Semantic relationship is preserved. Compute once, use later, which makes it faster. Uses Singular Value Decomposition (SVD) so vectors produced are more accurate	Requirement of huge memory to store co-occurrence matrix.
Prediction Based (Distributed Word Representations)	Word2vec CBOW Model	Better than deterministic method as it probabilistic in nature. Less memory (RAM) requirements.	Based on taking average of context words. Training from scratch takes a long time if not optimized.
	Word2vec Skip-Gram Model	Computes two different semantics for same word occurring in different context	Training on separate local context windows instead of at global level. Hence it poorly utilize the statistics of the corpus

Word Embedding Approaches Comparison

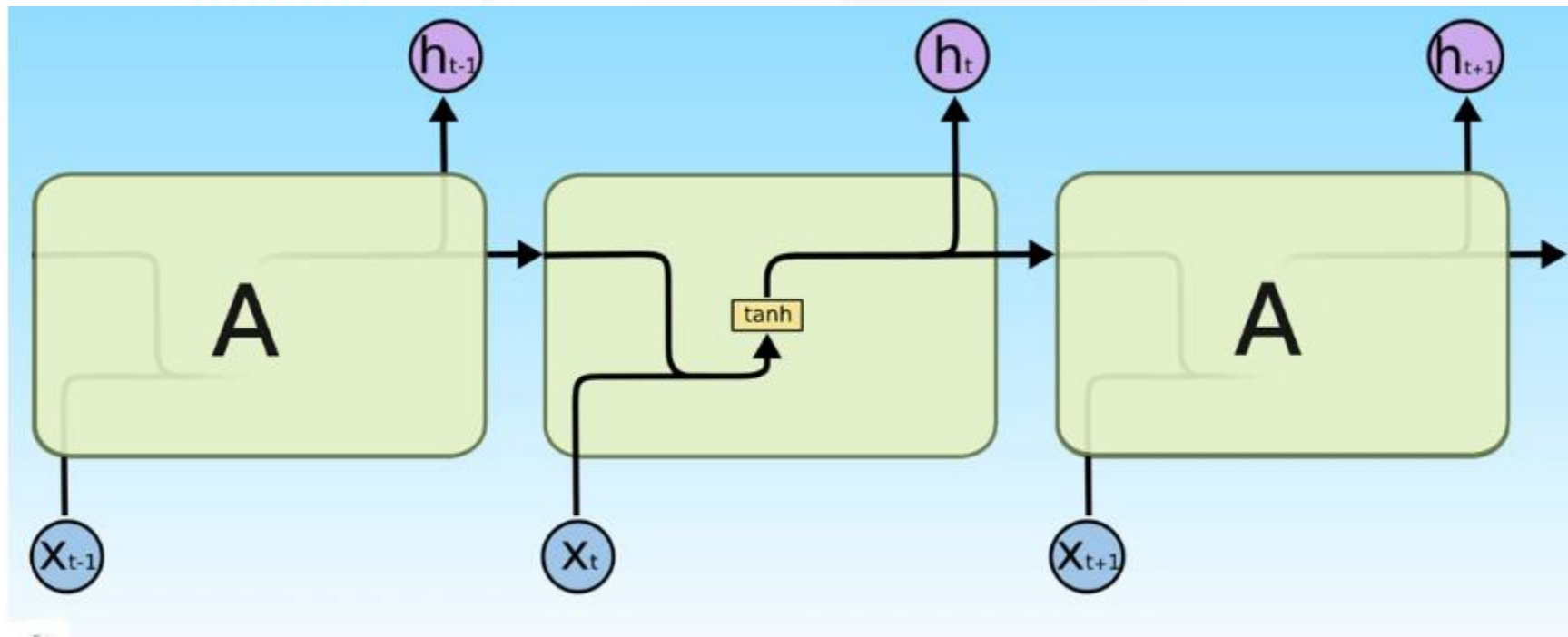
Type of Approach	Techniques	Type	Capture Semantics	Capture Context	Memory Requirement	Computational Time
Discrete Word Representations	One-Hot Encoding	Frequency based	No	No	Low	Low
	Count Vector	Frequency based	No	No	Low	Low
	TF_IDF	Frequency based	No	No	Low	Low to Moderate
	Co-occurrence Vector	Frequency based	No	Yes	High	Low to Moderate
Distributed Word Representations	Word2vec CBOW model	Probability Based	Yes	Yes	Low to Moderate	Moderate to High
	Word2vec Skip-Gram model	Probability Based	Yes	Yes	Low to Moderate	Moderate to High



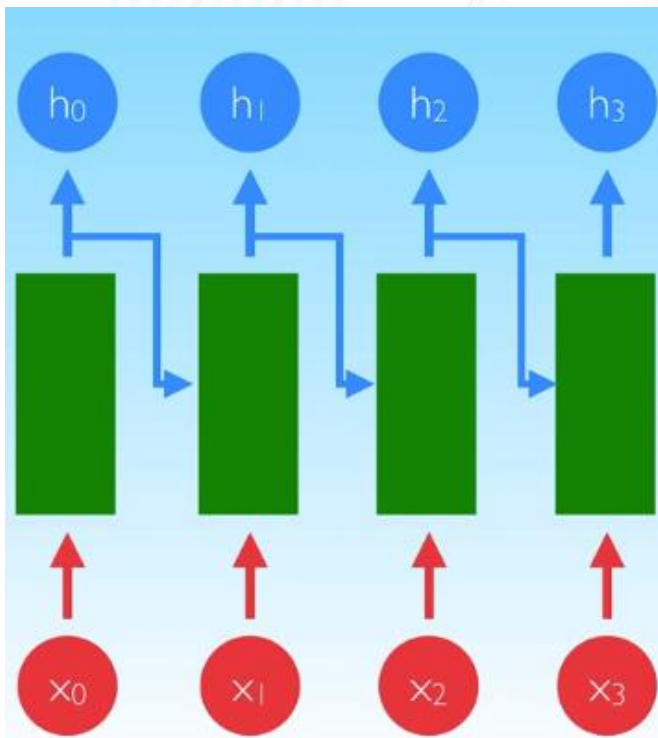
- Connections between units form a directed cycle
- Networks with internal state



Vanilla RNN



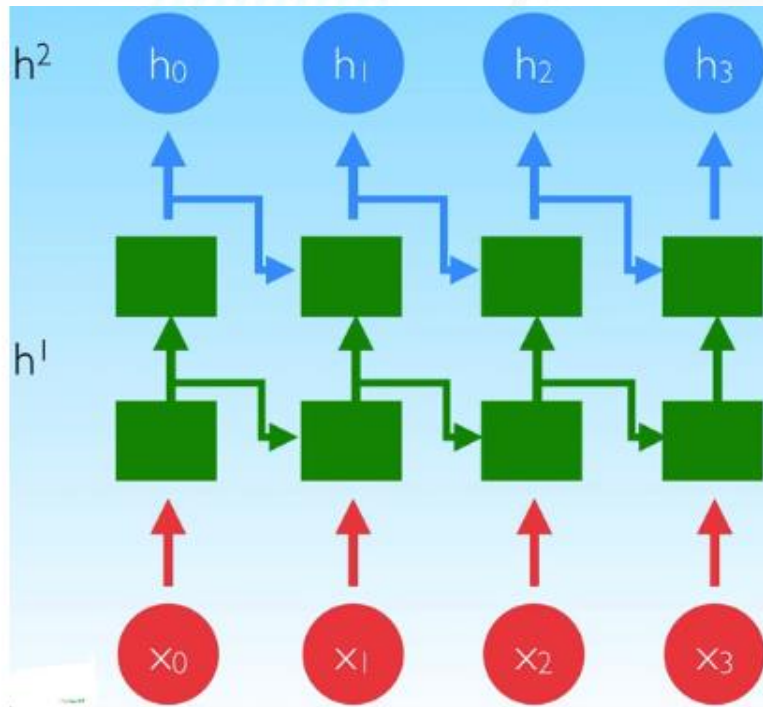
Unroll Time



$$h_t = \tanh(w h_{t-1} + u x_t)$$

- w, u do not depend on t
- same weights at all times

Deep RNN



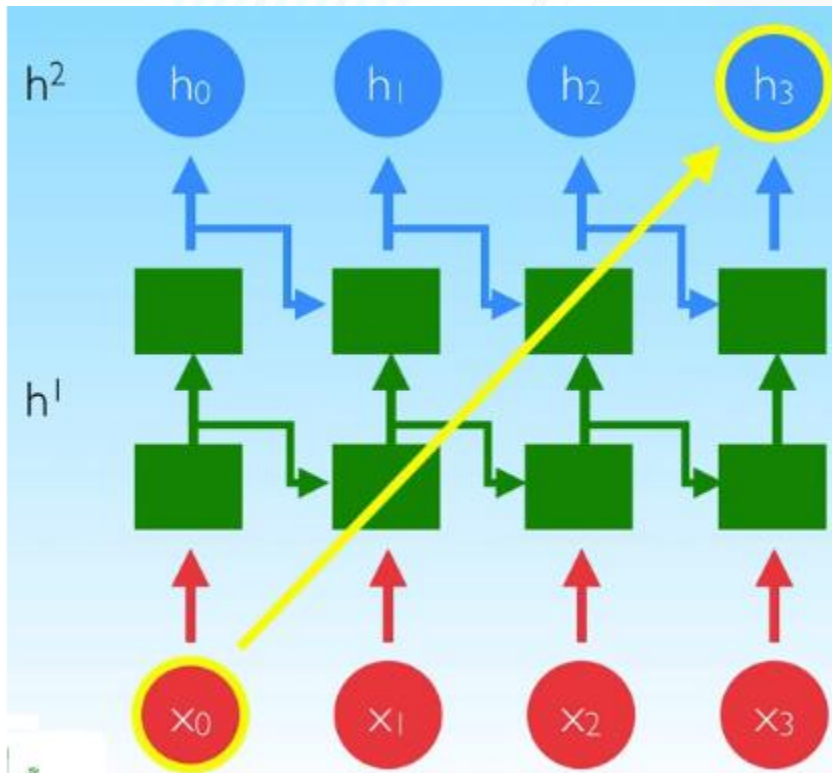
$$h_t^2 = \tanh(w^2 h_{t-1}^2 + u^2 h_t^1)$$

Second Layer

$$h_t^1 = \tanh(w^1 h_{t-1}^1 + u^1 x_t)$$

First Layer

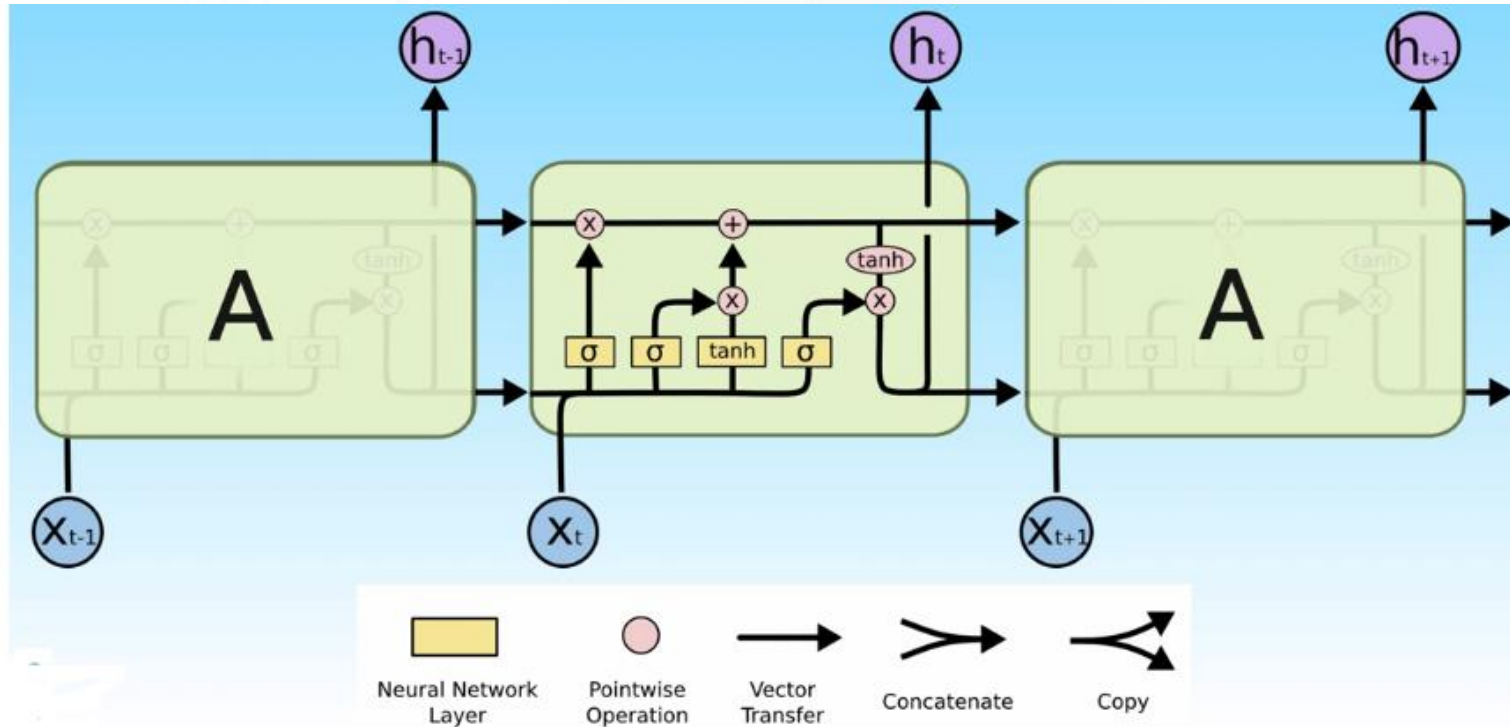
Long Term Dependency Problem



- Short term OK
- Long term problematic

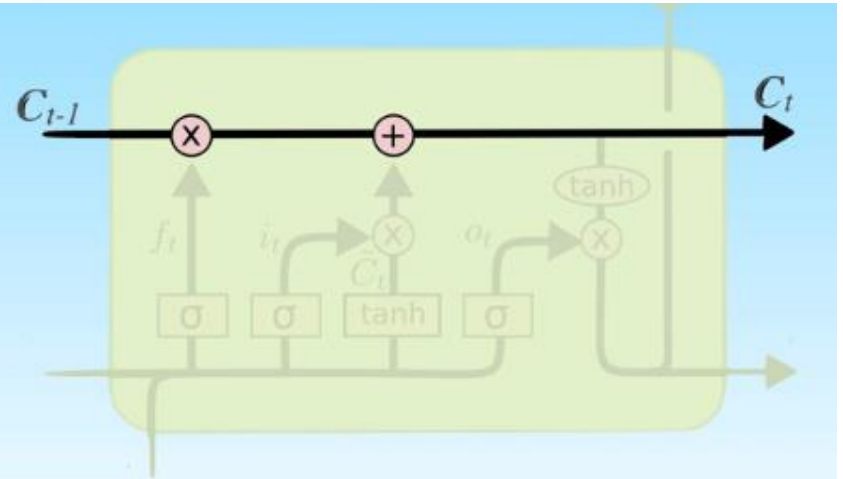
LSTM

Learn to selectively remember and forget

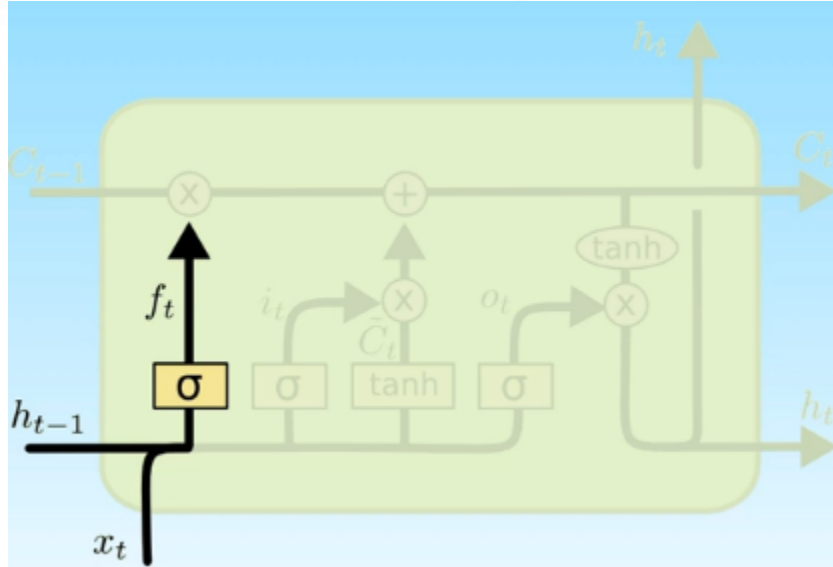


Cell State

- Cell maintains state
- Gates modify information

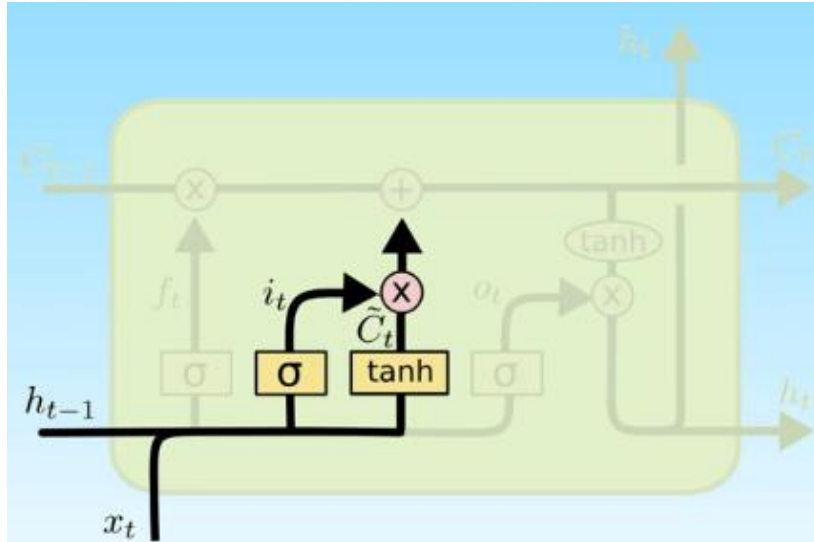


Forget Gate



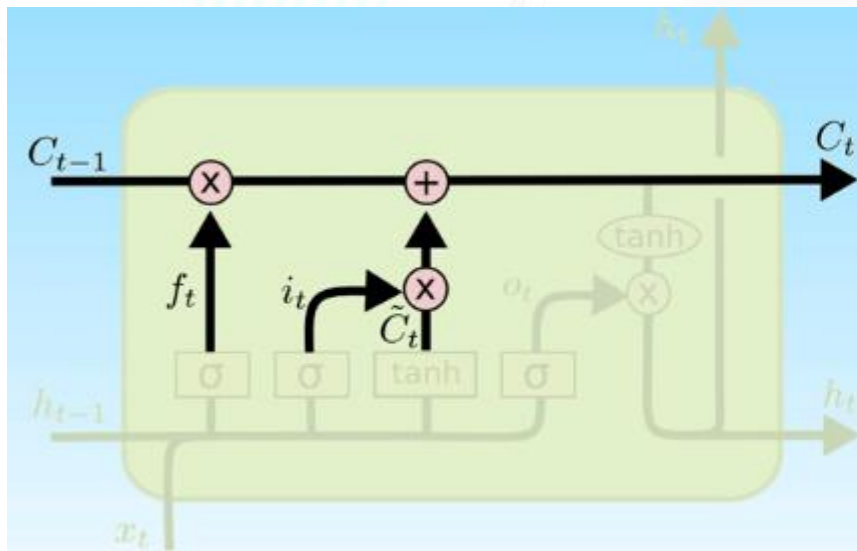
$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

Input Gate



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

State Update



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

- <https://github.com/DiveshRKubal/GreyAtom-Deep-Learning/tree/master/GreyAtom-Deep-Learning/RNN>