Deep Learning – Recurrent Neural Networks (RNN)

By,
Divesh R. Kubal
Data Scientist, eClerx Services
Center of Excellence – Machine Learning

Table of Contents

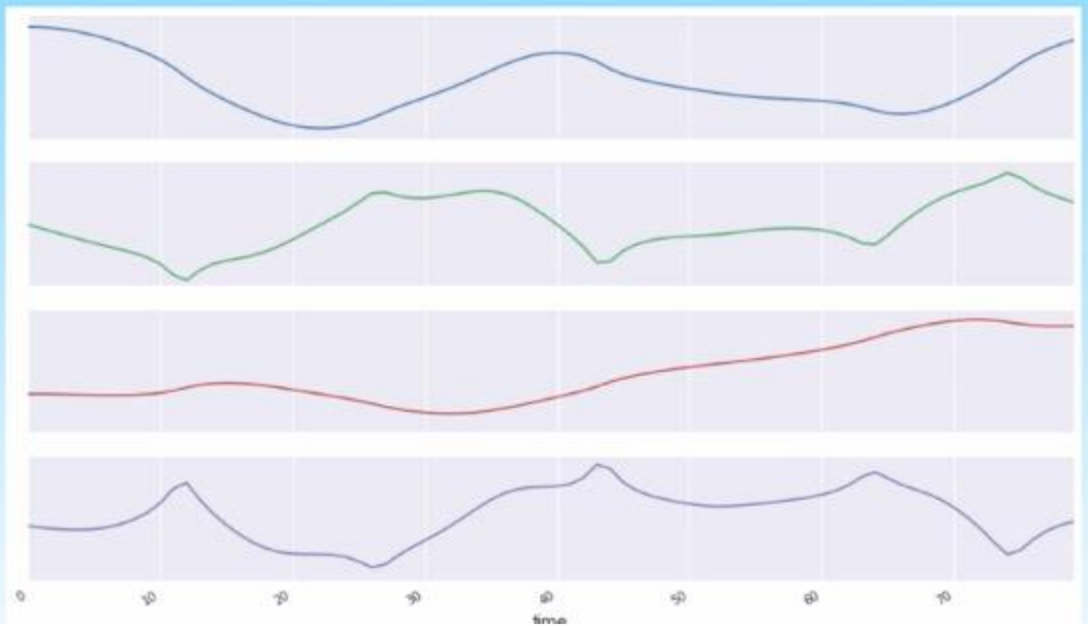
1. Time Series
2. Sequence Problems
3. Word Embeddings
   A. CBOW
   B. Skip-gram
4. Recurrent Neural Nets
5. Long Short Term Memory

Github Link

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

Time Series

- Vector sequence

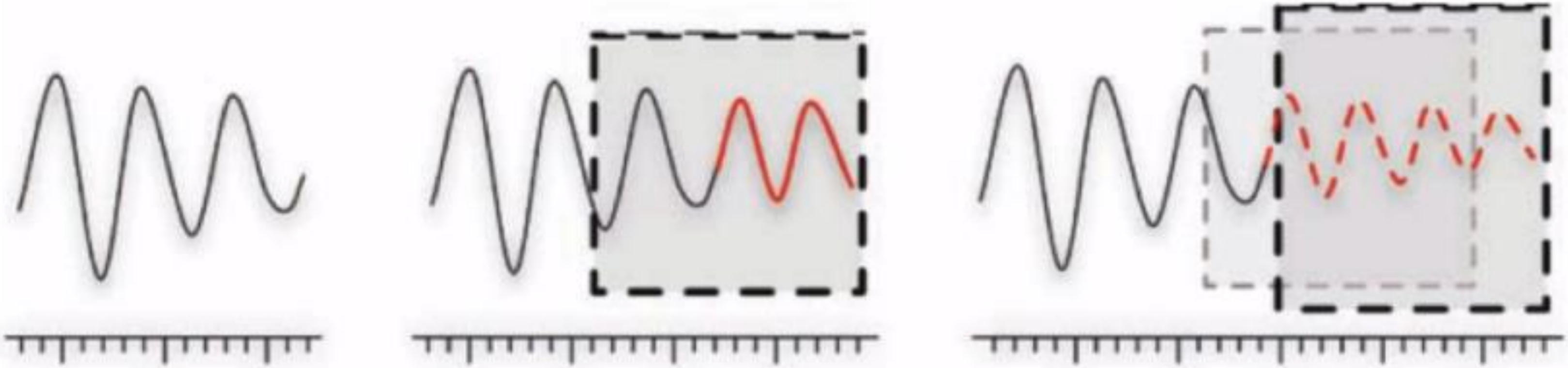


ML for Time Series

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

Prediction

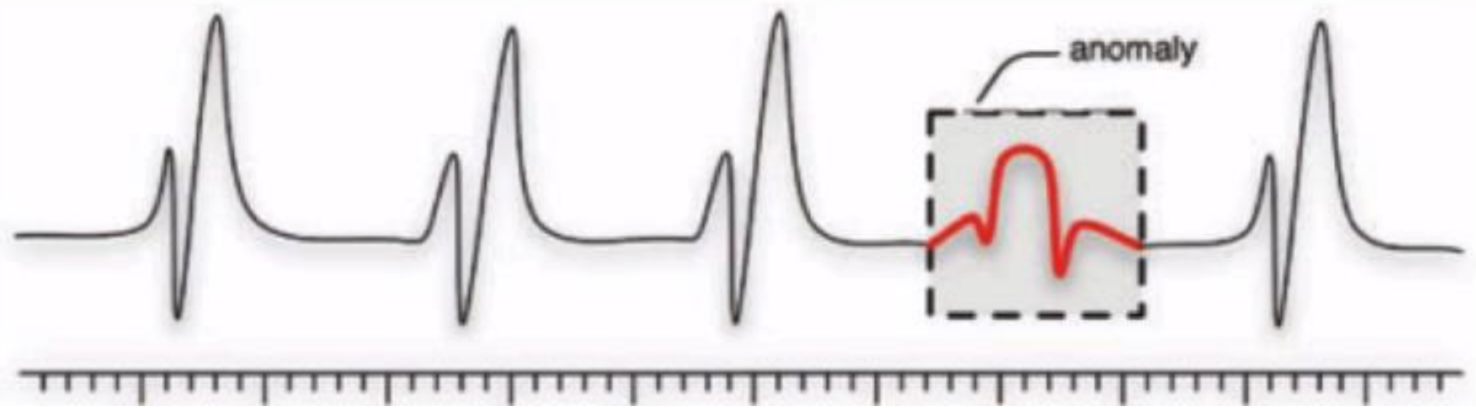
Use past to predict the future



(a) (b) (c)

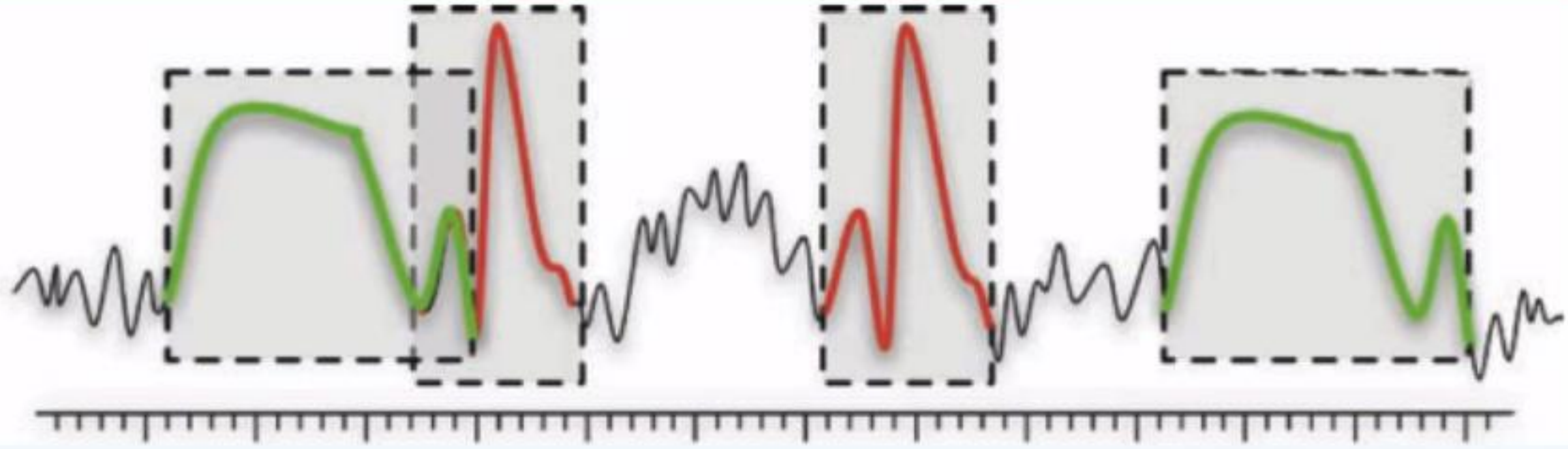
Prediction

Detect deviation from standard behavior



Prediction

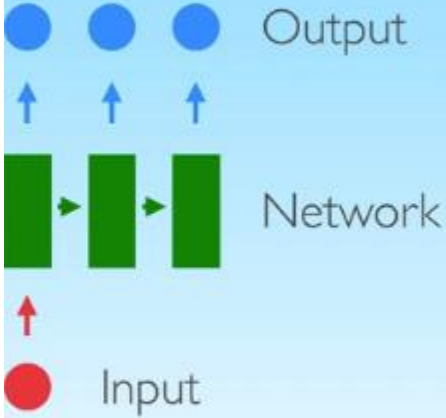
Find recurring patterns



Sequence Problems
One to One

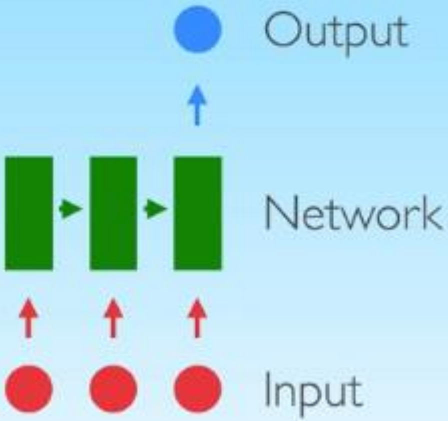
- Point-wise Forecasting
- Classification (fixed input/output size)

One to Many



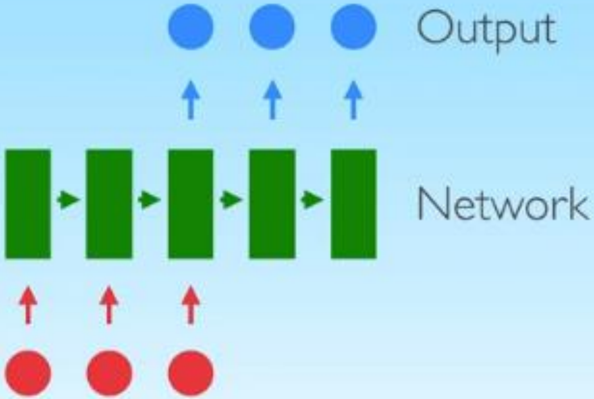
- Sequence output from single input
- e.g. image captioning

Many to One



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

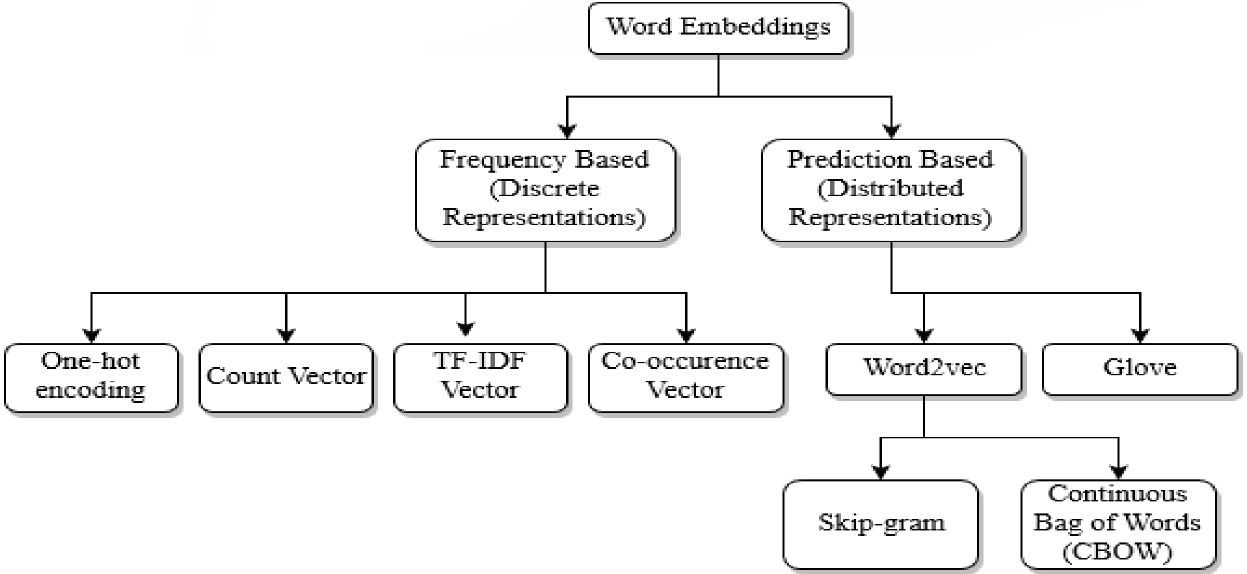
Many to Many



- 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.



CBOW

- 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

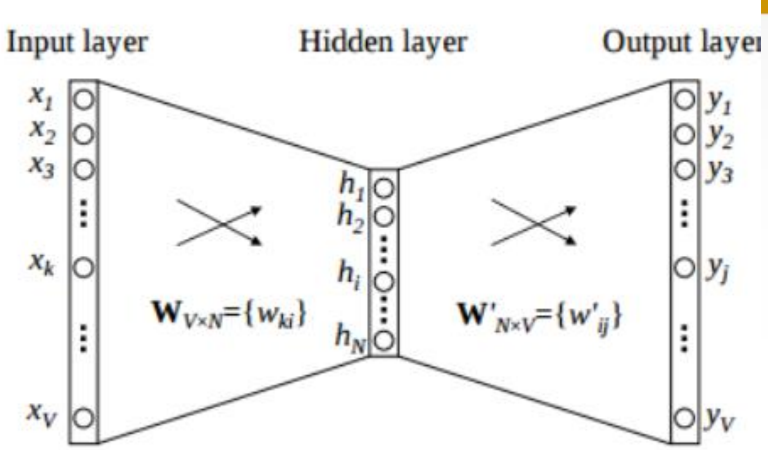
Sample one-hot encoded Matrix

Input	Output			Hey	This	is	sample	corpus	using	only	one	context	word
Hey	this		Datapoint 1	1	0	0	0	0	0	0	0	0	0
this	hey		Datapoint 2	0	1	0	0	0	0	0	0	0	0
is	this		Datapoint 3	0	0	1	0	0	0	0	0	0	0
is	sample		Datapoint 4	0	0	1	0	0	0	0	0	0	0
sample	is		Datapoint 5	0	0	0	1	0	0	0	0	0	0
sample	corpus		Datapoint 6	0	0	0	1	0	0	0	0	0	0
corpus	sample		Datapoint 7	0	0	0	0	1	0	0	0	0	0
corpus	using		Datapoint 8	0	0	0	0	1	0	0	0	0	0
using	corpus		Datapoint 9	0	0	0	0	0	1	0	0	0	0
using	only		Datapoint 10	0	0	0	0	0	1	0	0	0	0
only	using		Datapoint 11	0	0	0	0	0	0	1	0	0	0
only	one		Datapoint 12	0	0	0	0	0	0	1	0	0	0
one	only		Datapoint 13	0	0	0	0	0	0	0	1	0	0
one	context		Datapoint 14	0	0	0	0	0	0	0	1	0	0
context	one		Datapoint 15	0	0	0	0	0	0	0	0	1	0
context	word		Datapoint 16	0	0	0	0	0	0	0	0	1	0
word	context		Datapoint 17	0	0	0	0	0	0	0	0	0	1

CBOW

- 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

C1	this	0	1	0	0	0	0	0	0	0	0

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

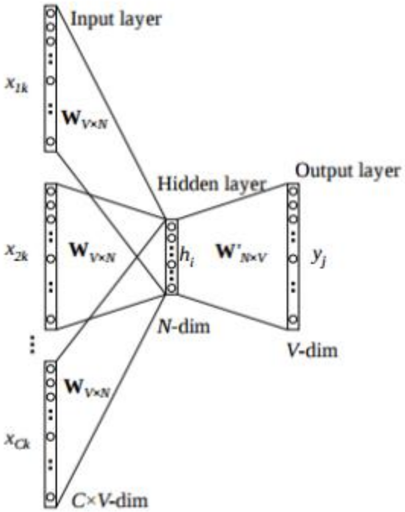
Hidden Activation

5	6	7	8

Working of CBOW

- The input layer and the target, both are one- hot encoded of size [1 X 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 X N] , hidden-Output layer matrix size =[N X V] : Where N is the number of dimensions we choose to represent our word in. It is arbitary and a hyper-parameter for a Neural Network. Also, N is the number of neurons in the hidden layer. Here, N=4.
- There is a 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	0	0
C2	corpus	0	0	0	0	1	0	0	0	0	0
C3	context	0	0	0	0	0	0	0	0	1	0

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 is 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

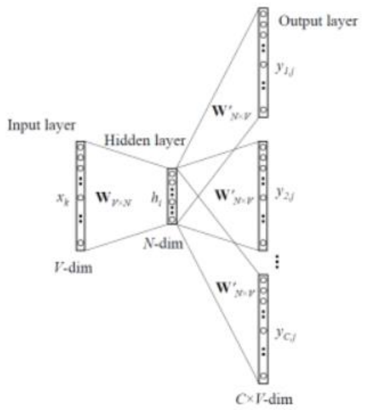
- CBOW takes the average of the context of a word (as seen above in calculation of hidden activation). For example, Apple can be both a fruit and a company but CBOW takes an average of both the contexts and places it in between a cluster for fruits and companies.
- Training a CBOW from scratch can take 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	5	6	7	8

hidden-output weights matrix

0.12	0.13	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

output

7.52	7.79	0.04	0.3	0.56	0.82	0.88	0.34	0.6	0.86
7.52	7.79	0.04	0.3	0.56	0.82	0.88	0.34	0.6	0.86

softmax probabilities

0.024	0.03009744	0.04007182	0.05197034	0.0674019	0.08741555	0.11337186	0.14703530	0.19069461	0.24731158
0.024	0.03009744	0.04007182	0.05197034	0.0674019	0.08741555	0.11337186	0.14703530	0.19069461	0.24731158

Target

Hey	1	0	0	0	0	0	0	0	0	0
is	0	0	1	0	0	0	0	0	0	0

Error

-0.98	0.03009744	0.04007182	0.05197034	0.0674019	0.08741555	0.11337186	0.14703530	0.19069461	0.24731158
0.024	0.03009744	-0.9599282	0.05197034	0.0674019	0.08741555	0.11337186	0.14703530	0.19069461	0.24731158

Sum of error

-0.95	0.06179487	-0.9199564	0.10394068	0.1348038	0.17483111	0.22674373	0.29407076	0.38138922	0.49462316

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

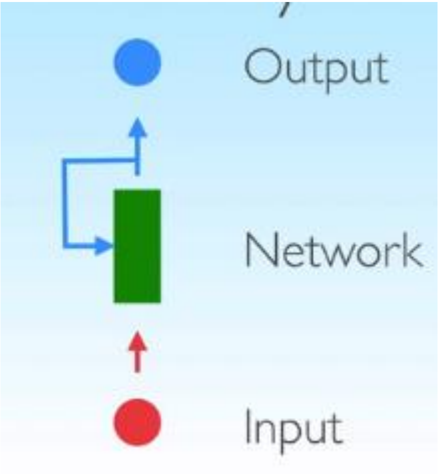
Type		Advantages	Limitations
Frequency Based (Discrete Word Representations)	One-Hot Encoding	Easy to compute. Low computational time to generate.	Faces curse of dimensionality as the number of vocabulary increases. Does not capture context and semantics
	Count Vector	Simple to implement. Takes a document 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

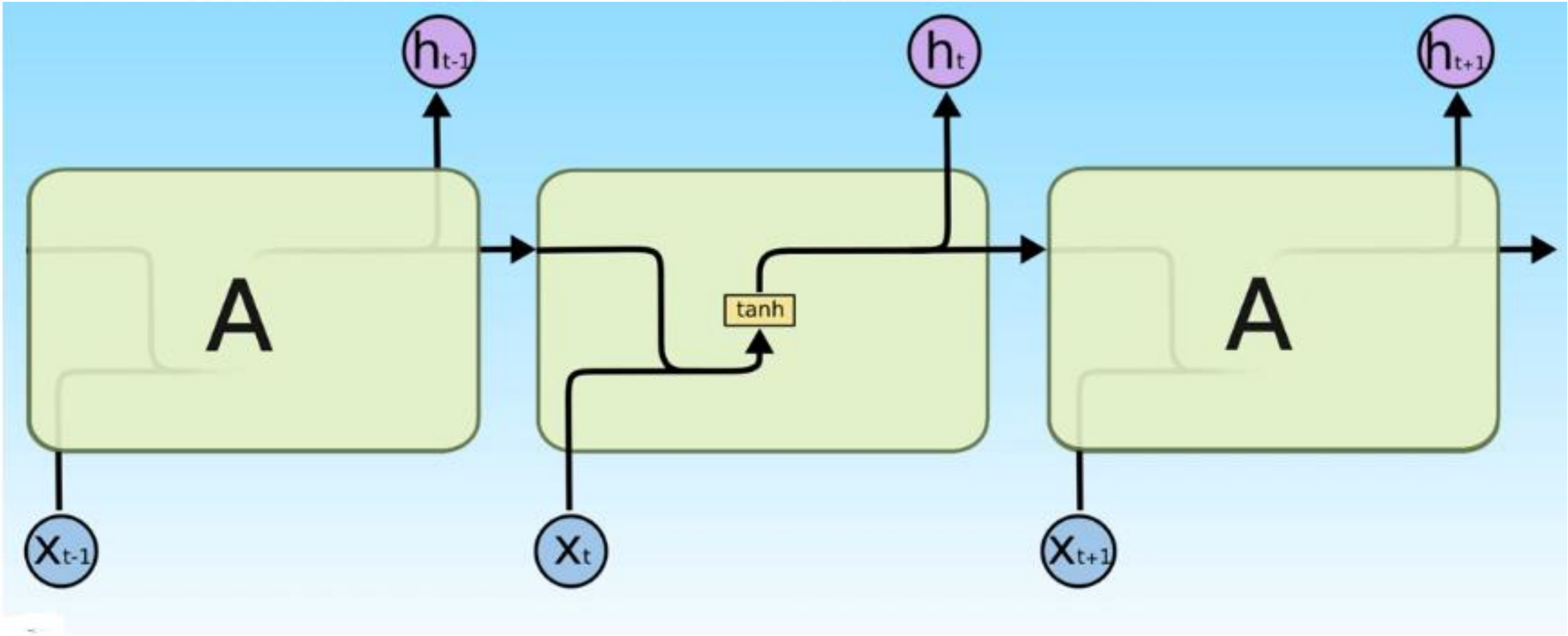
Type of Approach	Techniques	Type	Capture Semantics	Capture Context	Memory Requirement	Computational Time
Discrete Word Representations	One-Hot Encoding	Frequency based	No	No	Low	Low
	Count Vector	Frequency based	No	No	Low	Low
	TF_IDF	Frequency based	No	No	Low	Low to Moderate
	Co-occurence Vector	Frequency 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

RNN

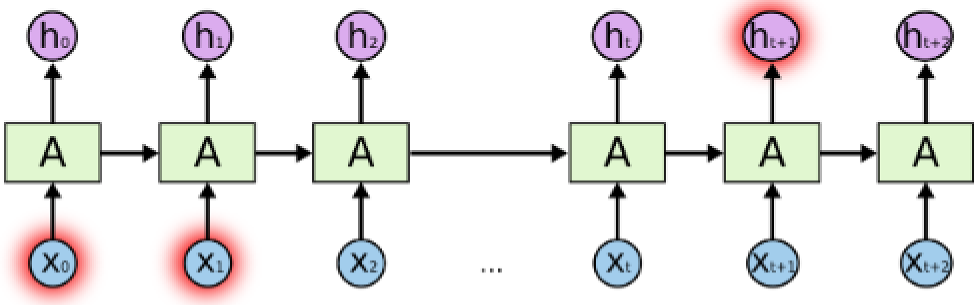
- Connections between units form a directed cycle
- Networks with internal state



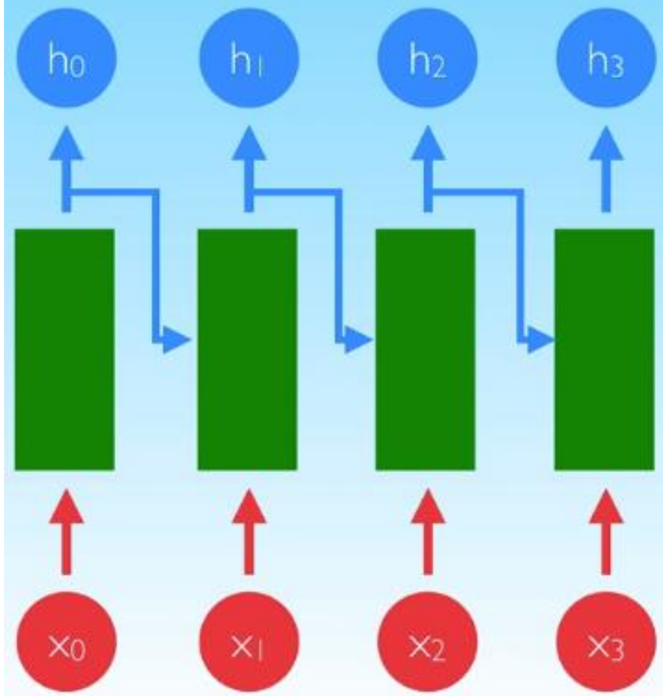
Vanilla RNN



- If we are trying to predict the last word in “the clouds are in the *sky*,” RNN work better.
- “I grew up in France… I speak fluent *French*.”. Here RNN won’t work.



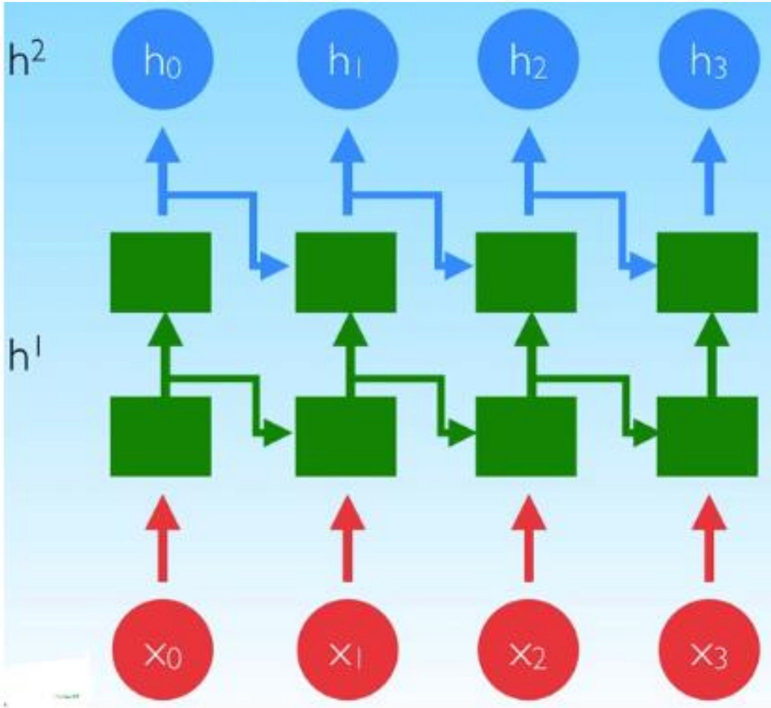
Unroll Time



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

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

Deep RNN



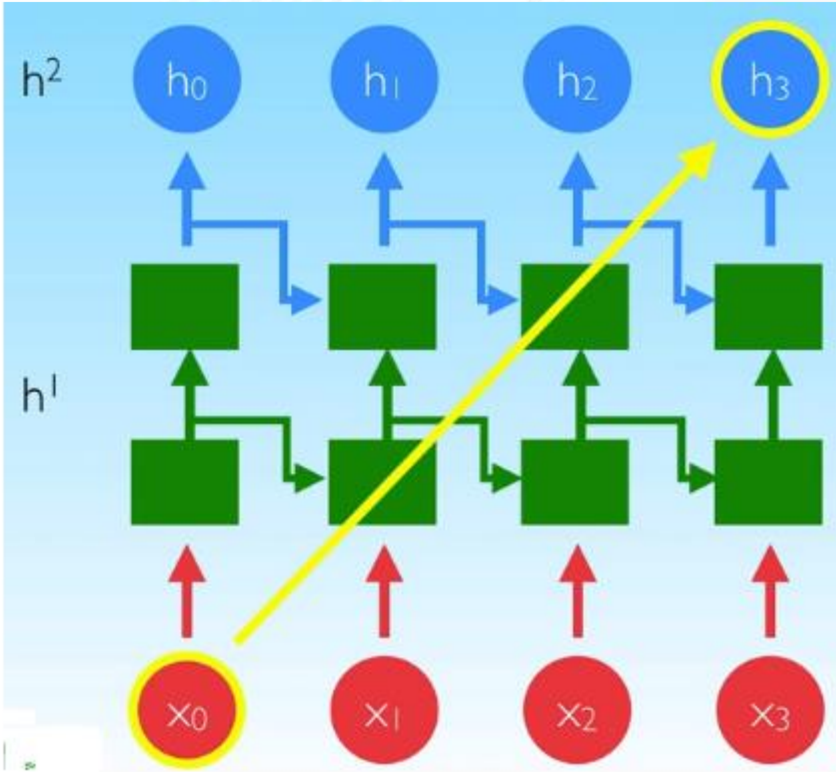
$$h_t^2 = \tanh(w^2 h_{t-1}^2 + u^2 h_t^1)$$

Second Layer

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

First Layer

Long Term Dependency Problem



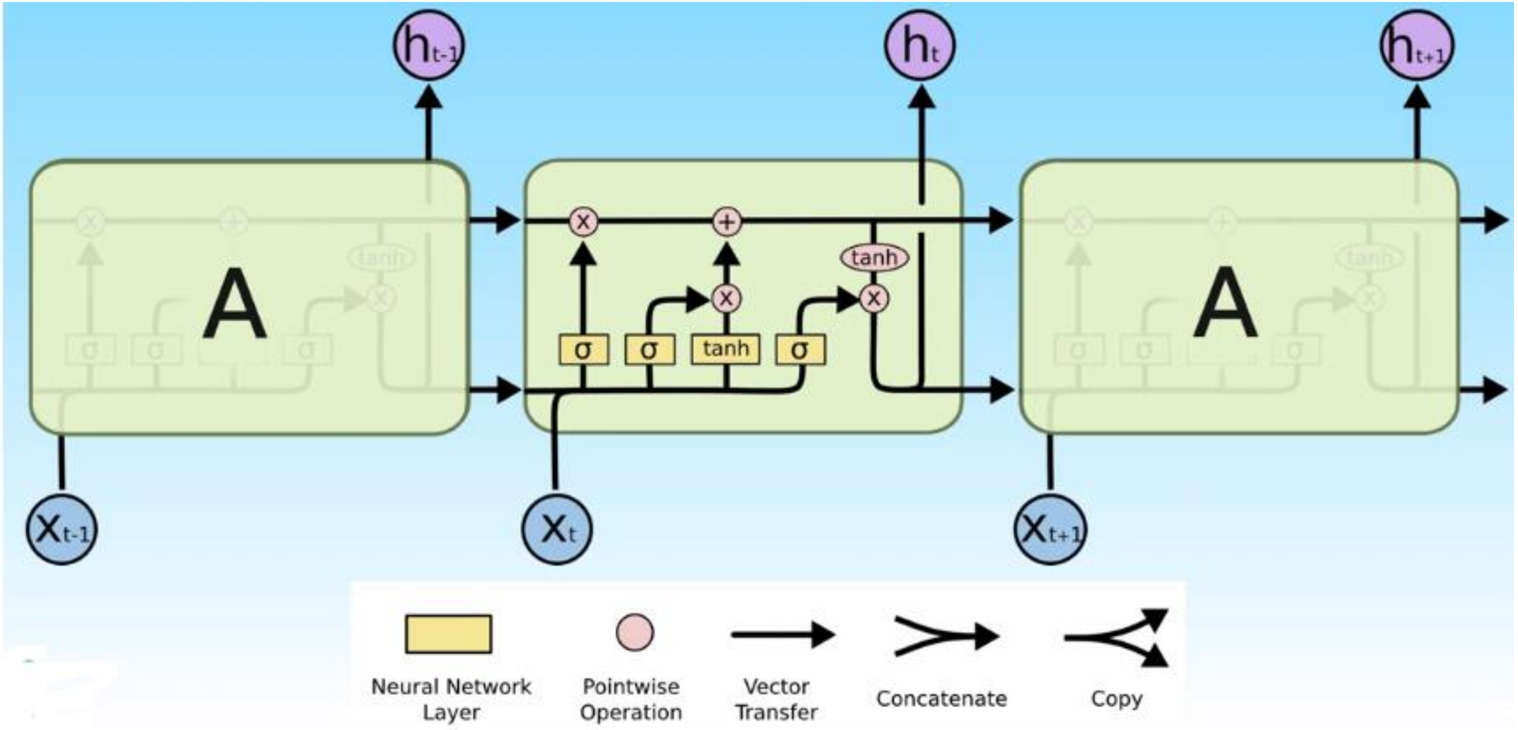
- Short term OK
- Long term problematic

LSTM

- Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies.
- LSTMs are explicitly designed to avoid the long-term dependency problem.

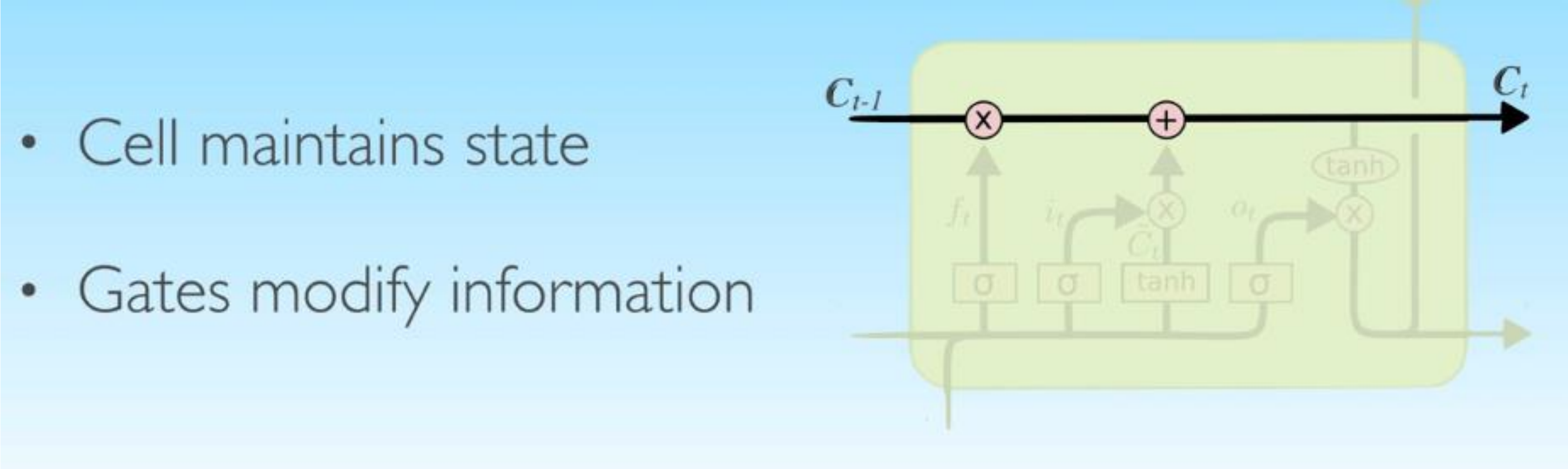
LSTM

Learn to selectively remember and forget



Cell State

- The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.
- The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It’s very easy for information to just flow along it unchanged.

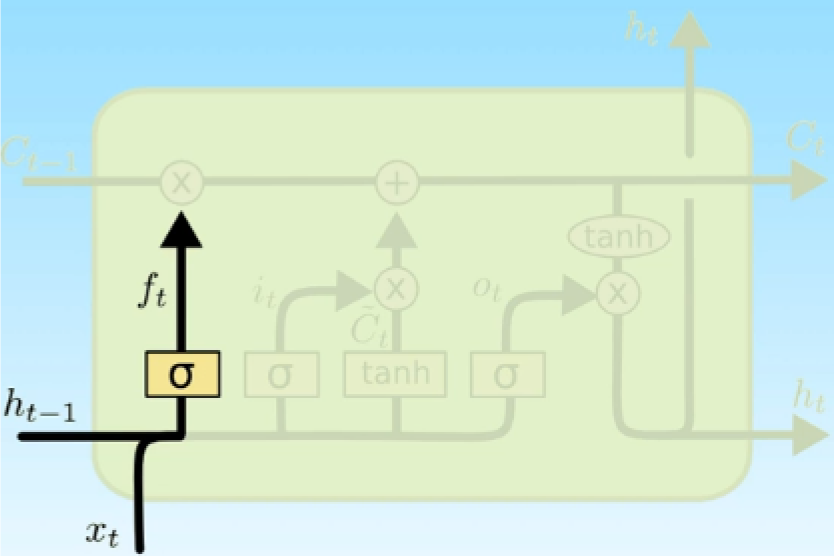


LSTM Gates

- The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.
- Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.

Forget Gate

- The first step in our LSTM is to decide what information we’re going to throw away from the cell state. This decision is made by a sigmoid layer called the “forget gate layer.”
- It looks at ht−1and xt, and outputs a number between 0 and 1 for each number in the cell state Ct−1. A 1represents “completely keep this” while a 0 represents “completely get rid of this.”



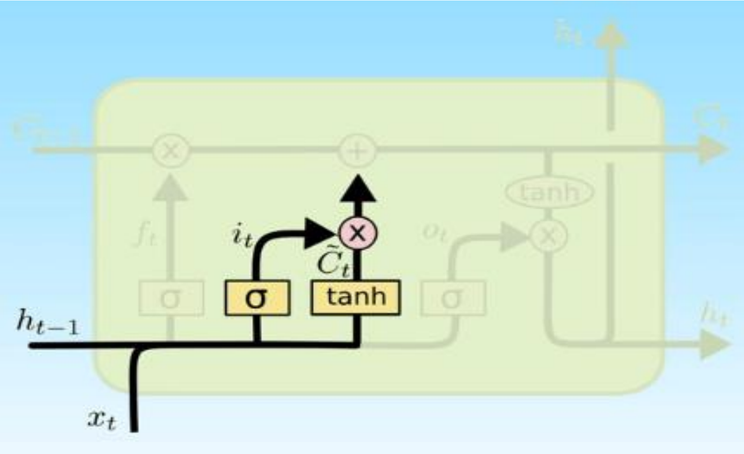
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Forget gate Example

- Let’s go back to our example of a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.

Input Gate

- The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values, Ct, that could be added to the state. In the next step, we'll combine these two to create an update to the state.
- In the example of our language model, we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting.

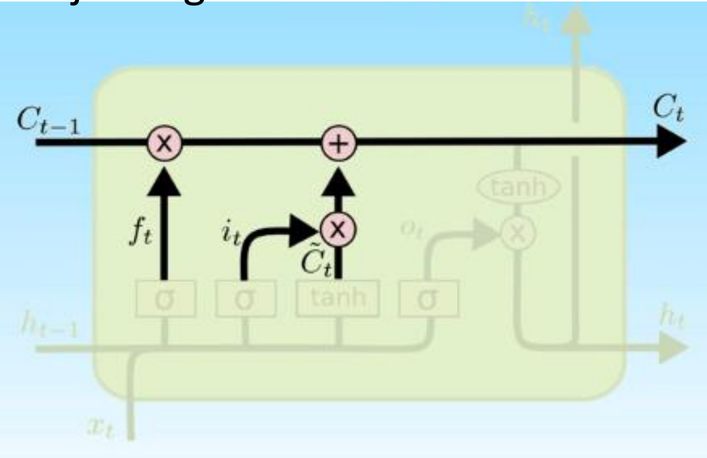


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

State Update

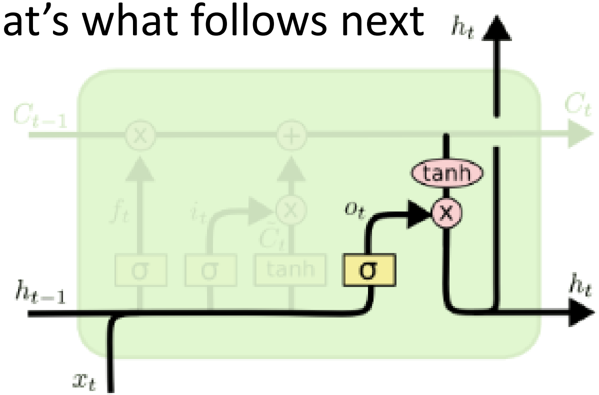
- It's now time to update the old cell state, Ct−1, into the new cell state Ct. The previous steps already decided what to do, we just need to actually do it.
- We multiply the old state by ft, forgetting the things we decided to forget earlier. Then we add it∗Ct. This is the new candidate values, scaled by how much we decided to update each state value.
- In the case of the language model, this is where we'd actually drop the information about the old subject's gender and add the new information, as we decided in the previous steps.



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

Output gate

- Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through tanh (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.
- For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case that's what is coming next. For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that's what follows next



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

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

Github Link

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