Word Embedding & Deep Learning for NLP

Topics covered in this MLS

- Introduction to NLP using deep learning
- Dense Encoding - Word Embedding
  - Word to Vec(CBOW & Skip Gram)
  - Glove-Global Vector
- Recurrent Neural Network(RNN)
- Long Short Term Memory(LSTM)

NLP using Deep Learning

Recently, deep learning methods have obtained very high performance across many different NLP tasks.

The ability of deep learning in the field of NLP is the better achievements by models that may require more data but less linguistic expertise to train and operate

Some of the few large demonstrations of the potential of deep learning were in natural language processing, specifically speech recognition & recently in machine translation.

Potential of Deep Learning for NLP

New NLP Model	Deep Learning technique offers the opportunity of new modelling approaches to challenging natural language problems like sequence-to-sequence prediction.
Drop-in Substitution Models	Deep learning technique can be dropped into existing natural language systems as replacement models that can achieve equivalent or better performance.

Dense Encoding: Word Embedding

Word representation as numbers:

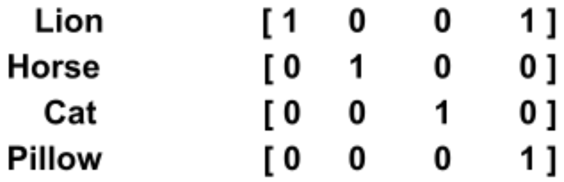
- To represent each word, you will create a zero vector with length equal to the vocabulary, then place a one in the index that corresponds to the word.

	Cat	Mat	On	Sat	The
The	0	0	0	0	1
cat	1	0	0	0	0
sat	0	0	0	1	0

Dense Encoding

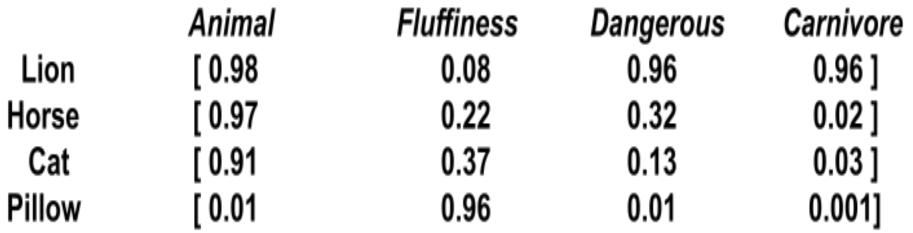
- Moving from sparse(one-hot) to dense vectors

One-hot encoding of words



Lion	[1	0	0	1]
Horse	[0	1	0	0]
Cat	[0	0	1	0]
Pillow	[0	0	0	1]

Dense encoding of words



	Animal	Fluffiness	Dangerous	Carnivore
Lion	[0.98	0.08	0.96	0.96]
Horse	[0.97	0.22	0.32	0.02]
Cat	[0.91	0.37	0.13	0.03]
Pillow	[0.01	0.96	0.01	0.001]

- Each dimension could represent attributes such as geography, gender, POS etc.

Why do we need dense vector ?

- Sparse (one-hot) vectors are high dimensional
- sparse matrix can contain a lot less information than a non-sparse matrix, since a vast majority of its entries must be equal to 0.
- Sparse vectors do not have a neighborhood or directional relationship between words
- Inserting a new word in the vocabulary will lead to catastrophic changes in the input space

Word to Vec

- Word2vec is a neural net that processes text by “vectorizing” words. It takes input as a text corpus and its output is a set of vectors: feature vectors that represent words in that corpus.

- The purpose and usefulness of Word2vec is to group the vectors of similar words together in vector space.

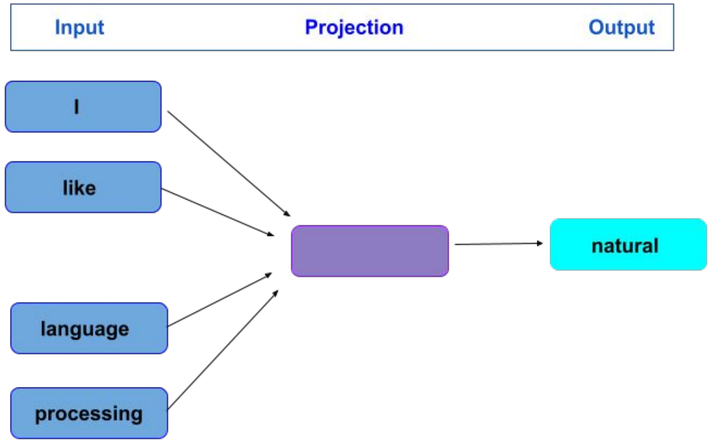
- In Word2Vec, there are two different architectures: Continuous bag of words(CBOW) and Skip Gram.

- We’ll be going through them in details.

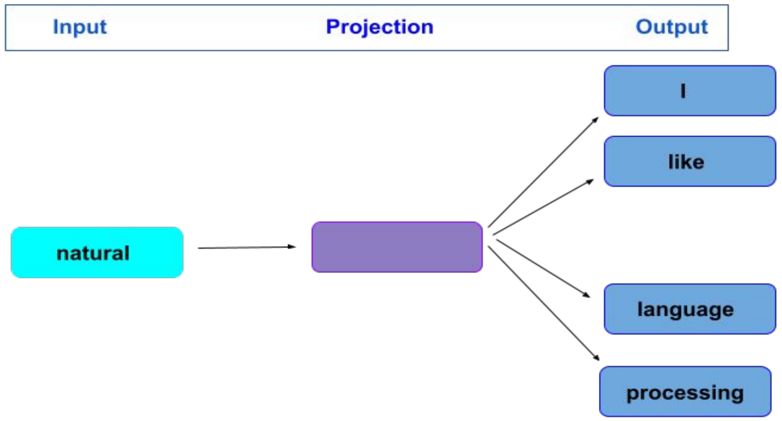
CBOW & Skip-Gram

Example: I like natural language processing

- In a CBOW model,we predict the center word given the surrounding context words(e.g. location ± 2)
- In a skip-gram model, we try to predict the context words (e.g. location ± 2) given the center word.

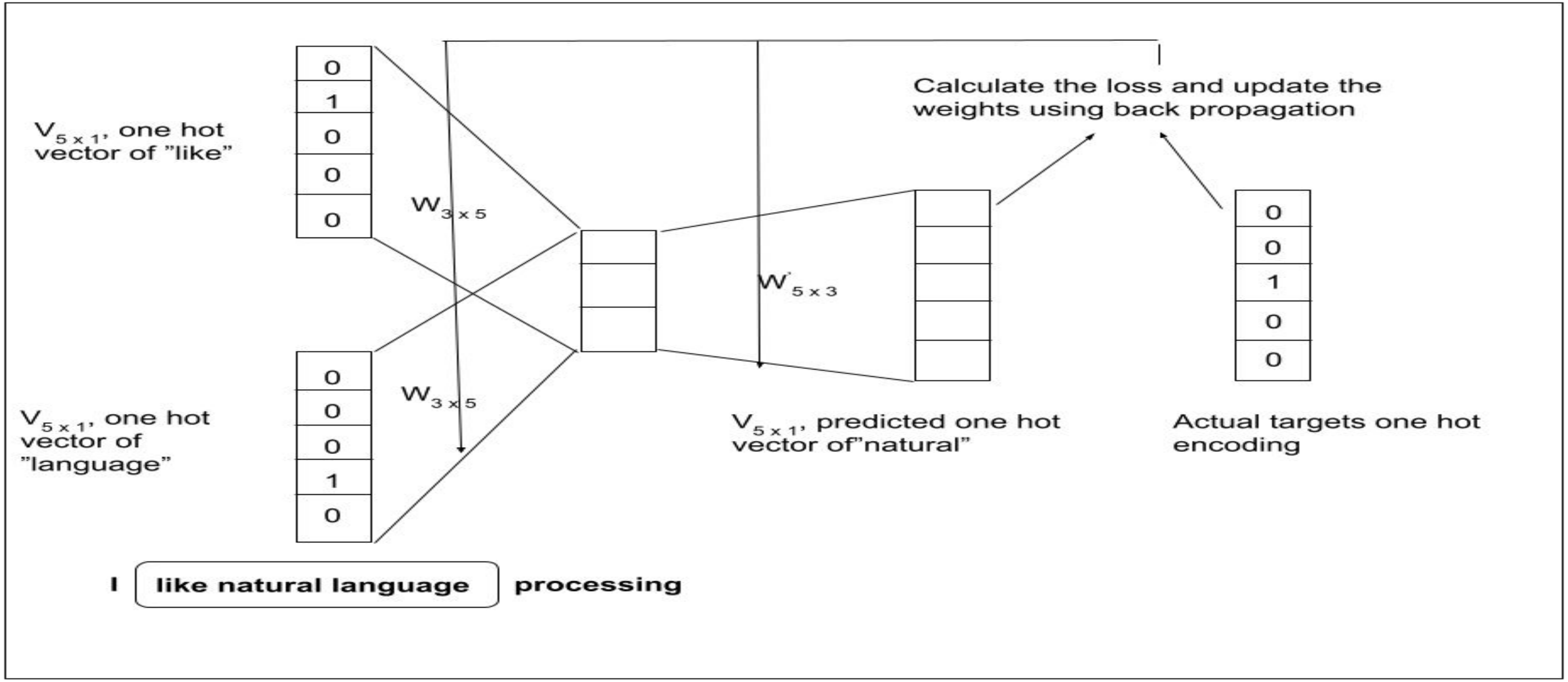


CBOW



Skip gram

Visualization of Continuous bag of words(CBOW)

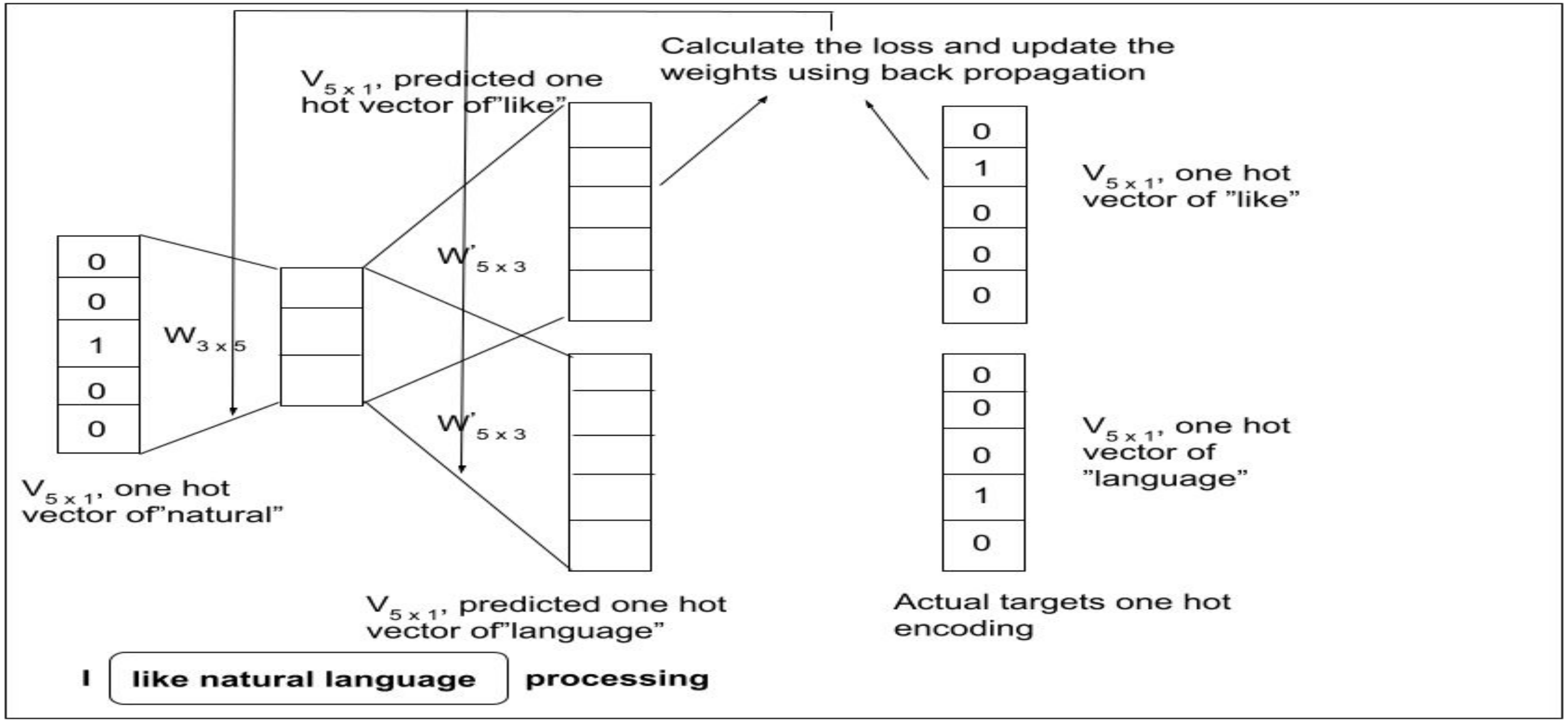

$V_{5 \times 1}$, one hot vector of "like"
0
1
0
0
0
$W_{3 \times 5}$
$V_{5 \times 1}$, one hot vector of "language"
0
0
0
1
0
$W_{3 \times 5}$
$W'_{5 \times 3}$
Calculate the loss and update the weights using back propagation
0
0
1
0
0
$V_{5 \times 1}$, predicted one hot vector of"natural"
Actual targets one hot encoding
I
like natural language
processing

CBOW likelihood function

Given a text sequence of length T , where the word at time step t is denoted as w(t) . For context window size m , the **likelihood function of the continuous bag of words model** is the probability of generating all center words given their context words:

$$\prod_{t=1}^{T} P(w^{(t)} \mid w^{(t-m)}, \ldots, w^{(t-1)}, w^{(t+1)}, \ldots, w^{(t+m)})$$

Visualization of Skip-Gram


Calculate the loss and update the weights using back propagation
$V_{5 \times 1}$, predicted one hot vector of"like"
0
0
1
0
0
$W_{3 \times 5}$
$W'_{5 \times 3}$
$W'_{5 \times 3}$
$V_{5 \times 1}$, one hot vector of"natural"
$V_{5 \times 1}$, predicted one hot vector of"language"
0
1
0
0
0
$V_{5 \times 1}$, one hot vector of "like"
0
0
0
1
0
$V_{5 \times 1}$, one hot vector of "language"
Actual targets one hot encoding
I
like natural language
processing

Likelihood function of skip gram

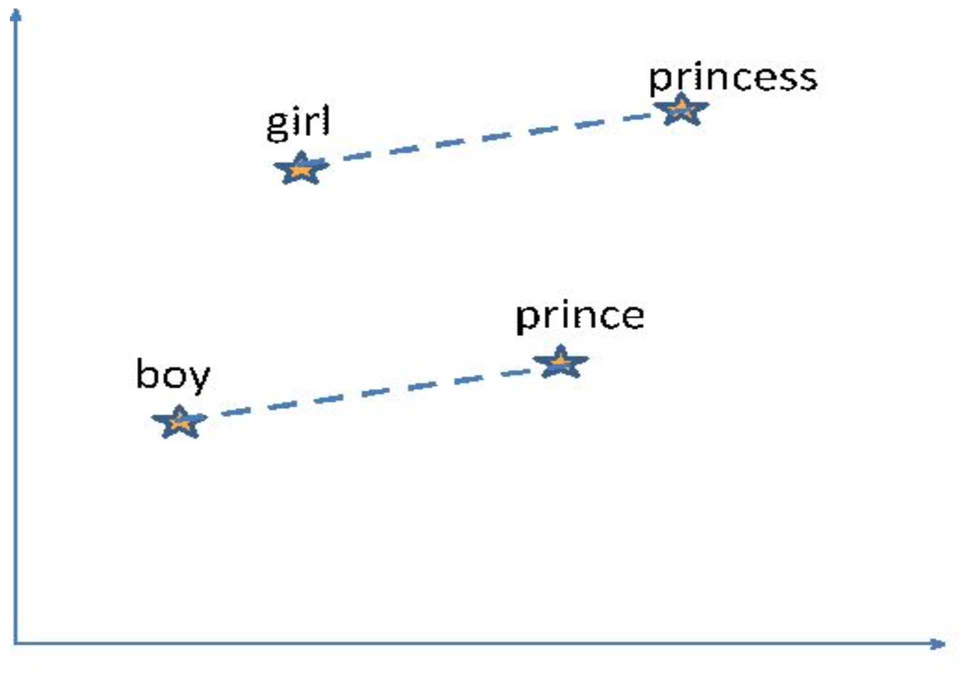
Given a text sequence of length T, where the word at time step t is denoted as w(t). Assume that context words are independently generated given any center word. For context window size m, **the likelihood function of the skip-gram model** is the probability of generating all context words given any center word:

$$\prod_{t=1}^{T} \prod_{-m \leq j \leq m,\ j \neq 0} P(w^{(t+j)} \mid w^{(t)})$$

where any time step that is less than 1 or greater than T can be omitted.

Analogs with new vectors

- The new vectors can directly be used to find analogs
- E.g. Vprince – Vboy + Vgirl = Vprincess



Word2Vec example results

Type of Relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
Currency	USA	Dollar	Iran	Rial
City in state	Chicago	Illinois	Stockton	California
Adjective to Adverb	apparent	apparently	rapid	rapidly
Nationality Adjective	Switzerland	Swiss	Cambodia	Cambodian
Superlative	Easy	Easiest	Lucky	Luckiest

Co-Occurrence Matrix

Raw counts within a certain window

	Cat	Is	chasing	the	house
cat	0	2	0	3	0
is	2	0	2	0	0
chasing	0	2	0	3	4
the	3	0	3	0	5
mouse	0	0	4	5	0

Counts converted to probabilities

	Cat	Is	chasing	the	house
cat	0.00	0.40	0.00	0.60	0.00
is	0.50	0.00	0.50	0.00	0.00
chasing	0.00	0.22	0.00	0.33	0.44
the	0.27	0.00	0.27	0.00	0.45
mouse	0.00	0.00	0.44	0.56	0.00

Co-occurrence

- Consider two similar words (cat, kitty) with similar context
- Their co-occurrence vectors will be similar
- The co-occurrence matrix will be low rank
- We can represent co-occurrence matrix using SVD $C = U \sum V$
- SVD is an expensive operation for a large vocabulary

Glove: Global vector

GloVe captures word-word co-occurrences in the entire corpus better

Word2vec leverages co-occurance within local context (neighbouring words).where as Glove model is based on leveraging global word to word co-occurance counts leveraging the entire corpus.

- Let Xij be the co-occurrence probability of words indexed with i and j
- Let Xi be $\sum_j$ Xij
- And, let Pij = P(j|i) = Xij / Xi
- What GloVe models is $F((wi - wj)^T wk) = Pik / Pjk$

Glove: Global vector

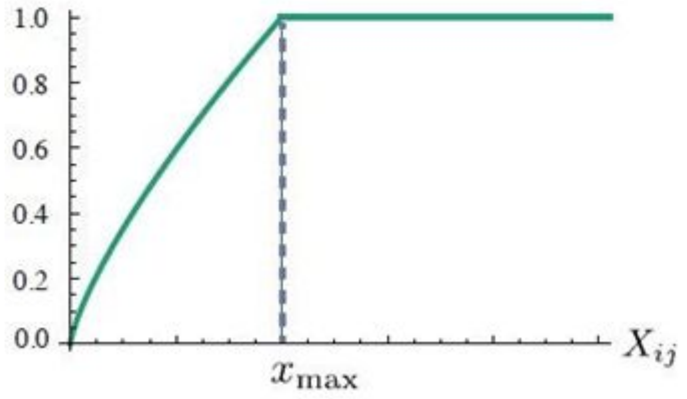
GloVe captures word-word co-occurrences in the entire corpus better

- Let Xij be the co-occurrence probability of words indexed with i and j
- Let Xi be $\sum_j$ Xij
- And, let Pij = P(j|i) = Xij / Xi
- What GloVe models is $F((wi - wj)^T wk) = Pik / Pjk$

Glove Explanation

Cost function:

$$J = \sum_{i,j} f(X_{ij})(w_i{}^T \tilde{w}_j - \log X_{ij})^2$$



- For words i,j co occurrence probability is $\mathbf{x}_{i,j}$

  & a weighing function f

- Suppresses rare occurrence
- Prevent frequent occurrences from taking over.

Recurrent neural network(RNN)

Why RNN

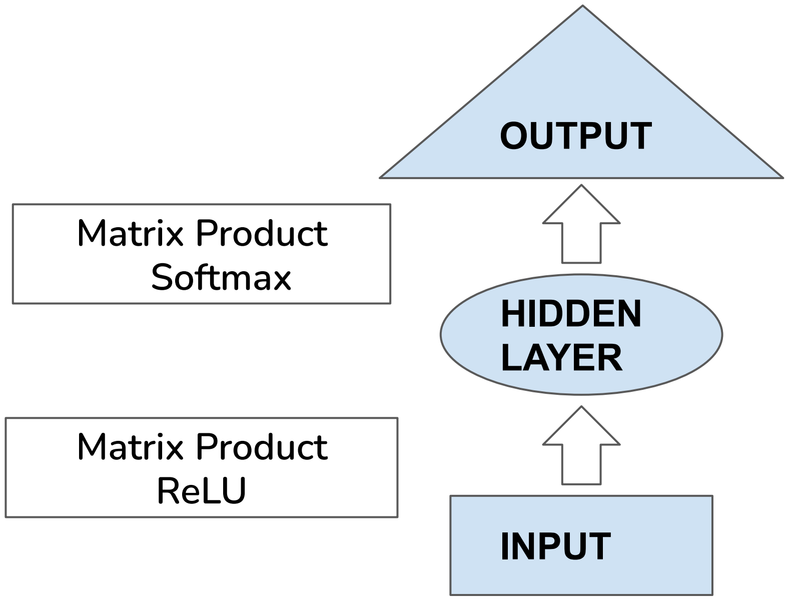
- There are multiple cases/scenario wherein the sequence of information/data determines the event itself.
- If we are trying to use such data for any reasonable output, we need a network which has access to some prior knowledge about the data to thoroughly understand and dealing with sequences in the information. **Recurrent Neural Networks(RNN) thus come into play.**
- **For example,** imagine we want to classify what kind of event is happening at every point in a movie. It's unclear how a traditional neural network could use its reasoning about previous events in the film to inform later ones. Recurrent neural networks address this issue.
- **They are networks with loops in them, allowing information to persist.**

Examples of Sequential data

- Speech
- Text
- Music
- Protein & DNA Sequences
- Stock Prices and other Times series data

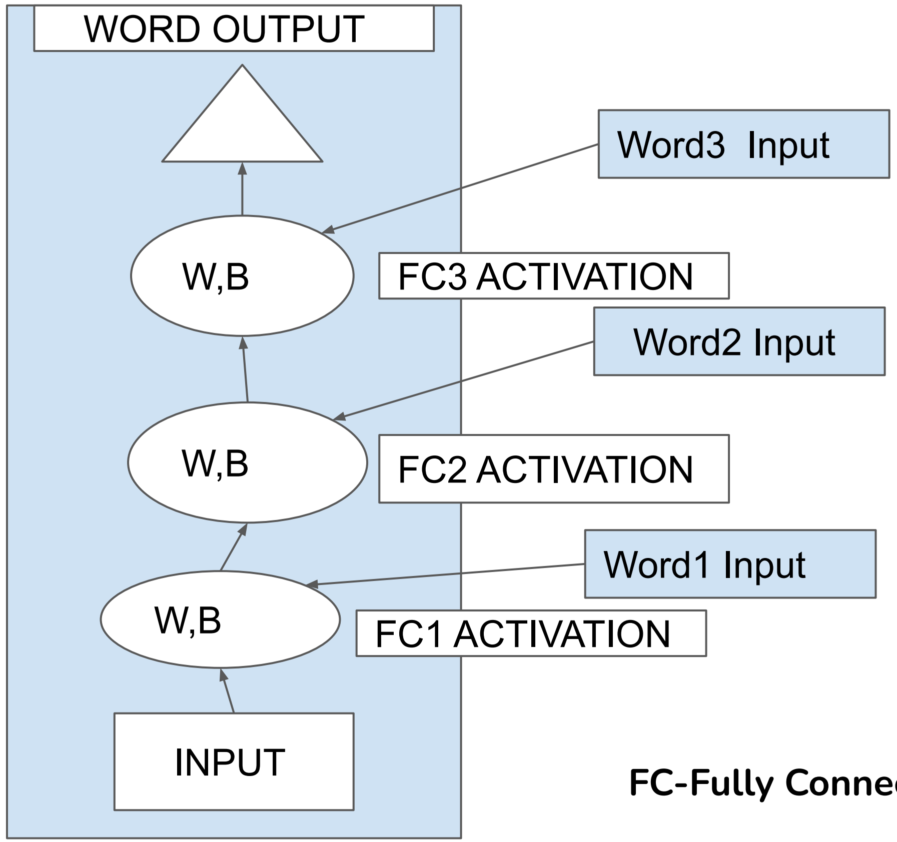
What is a Recurrent Neural Network(RNN)?

MLP
Multi-layer Perceptron where we have an input, hidden and output layer.



- The input layer receives the input,
- the hidden layer activations are applied
- we finally receive the output.

Architecture of an RNN

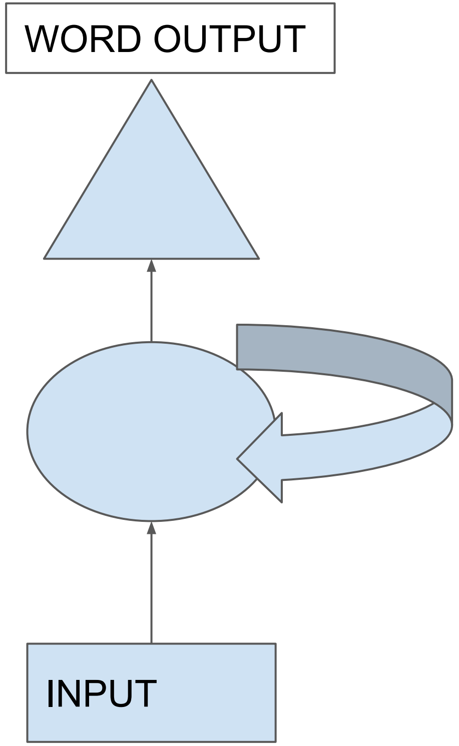


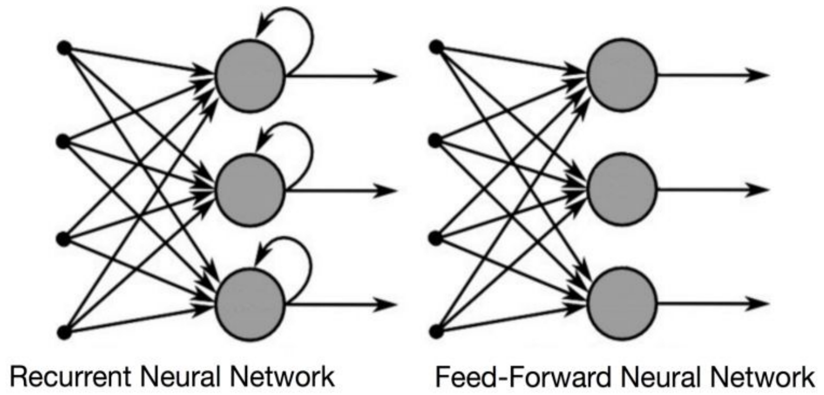
In previous slides,the weights and bias of the hidden layers are different and hence each of those layers behaves independently and cannot be combined together

To combine those hidden layers all together, we'll have the same weights and bias for those hidden layers.

FC-Fully Connected

Architecture of an RNN

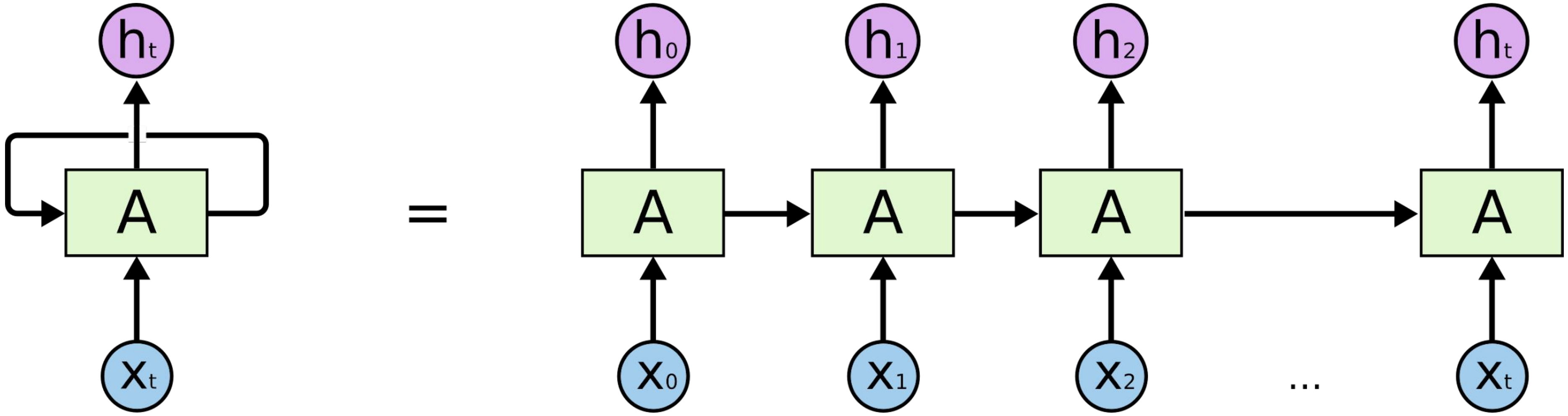




- All these hidden layers can be rolled in together in a single recurrent layer, as the weight and bias of all the hidden layers are the same.
- A recurrent neuron stores the state of a previous input and combines with the current input thereby preserving some association of the current input with the previous input.

https://builtin.com/data-science/recurrent-neural-networks-and-lstm

Architecture of an RNN



An unrolled Recurrent Neural Network.

This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists.

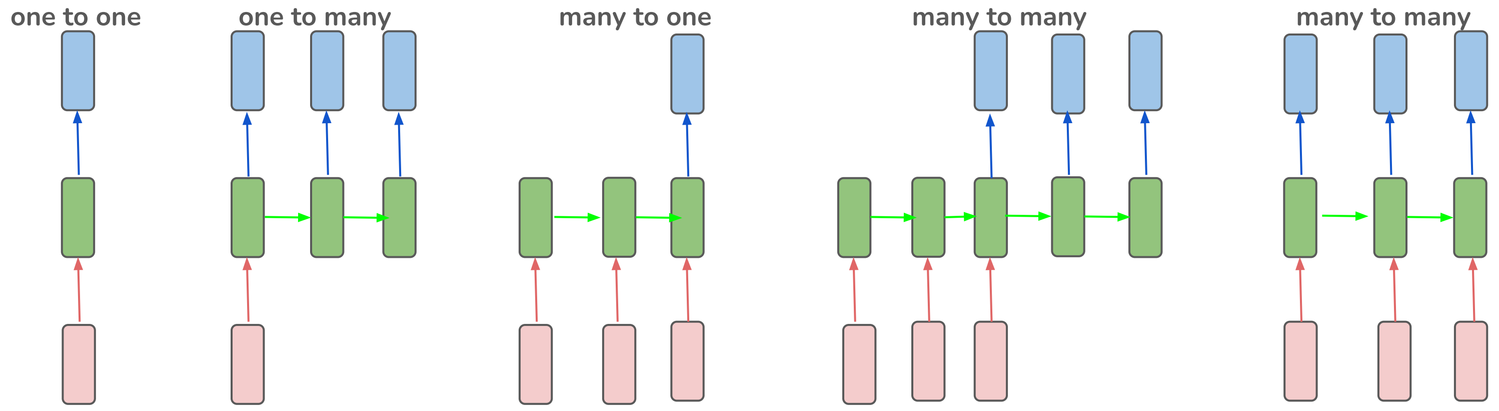
RNN in detail

- Let’s predict the fifth word of “This game is not good enough”.
- The word “This” has nothing preceding it, let’s take the letter “game”.
- At the time the word “game” is supplied to the network, a recurrence formula is applied to the word “game” and the preceding state which is the word “This”.
- These are known as numerous time steps of the input.
- At time t, the input is “game”, at time t-1, the input was “This”& the recurrence formula is applied to game and This both and we get a new state.
- The Formula for the current state as : $h_t = f\left(h_{t-1}, X_t\right)$
- Here, Ht is the new state, ht-1 is the preceding state while xt is the current input.

Types of analysis on sequential data using “recurrence”

Types of analysis possible on sequential data using “recurrence”.

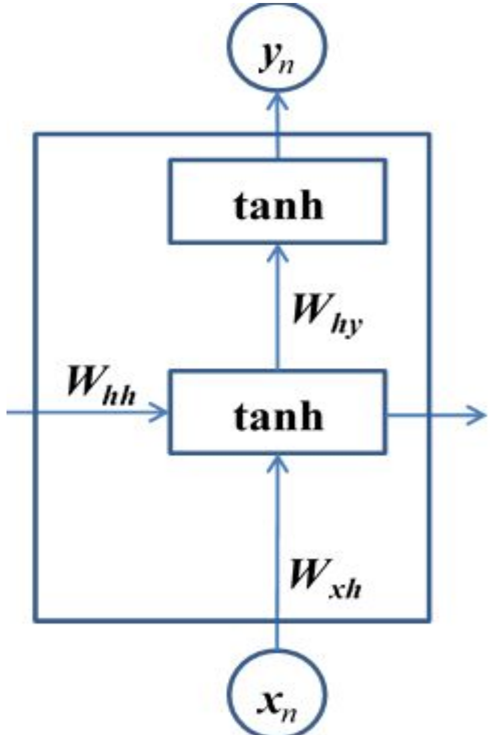
- One to one - Examples : POS tagging in NLP, Stock trade: {Buy, NoAction, Sell}
- One to many - Examples : Sentiment analysis in NLP
- Many to one - Examples : Generate caption based on an image, Generate text given topic
- Many to many - Examples : Language translation,

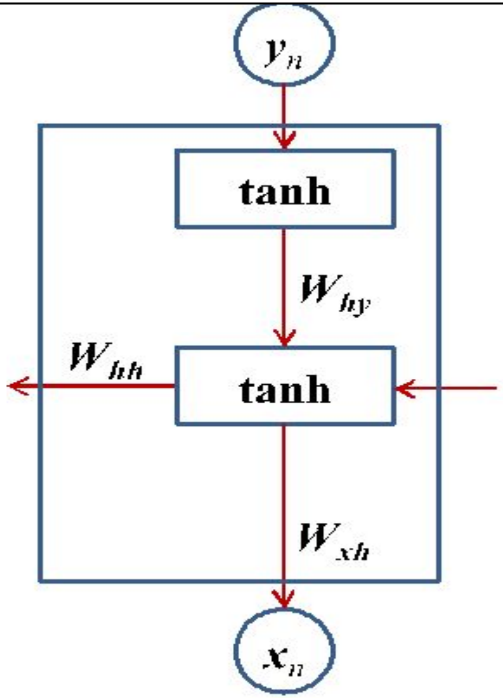


Backpropagation through time (BPTT)

Just like how forward propagation uses previous state

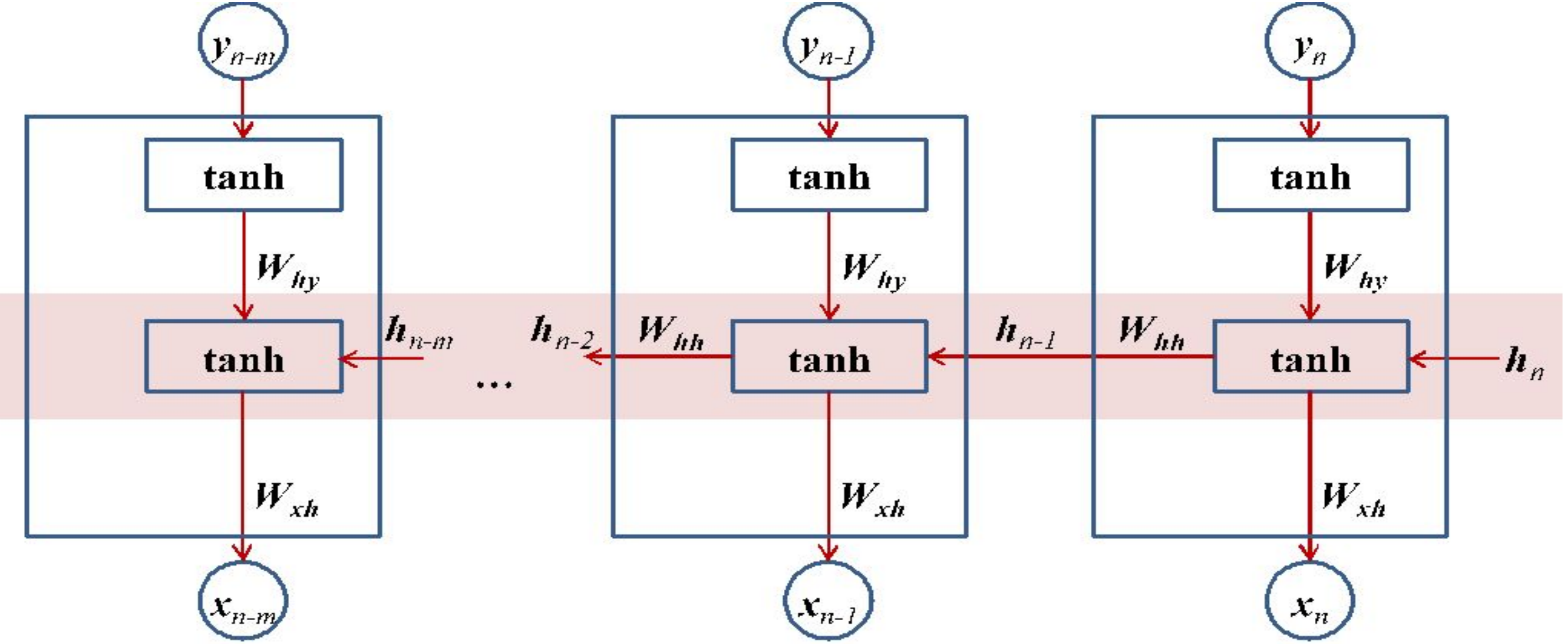
Backpropagation uses derivative from future output





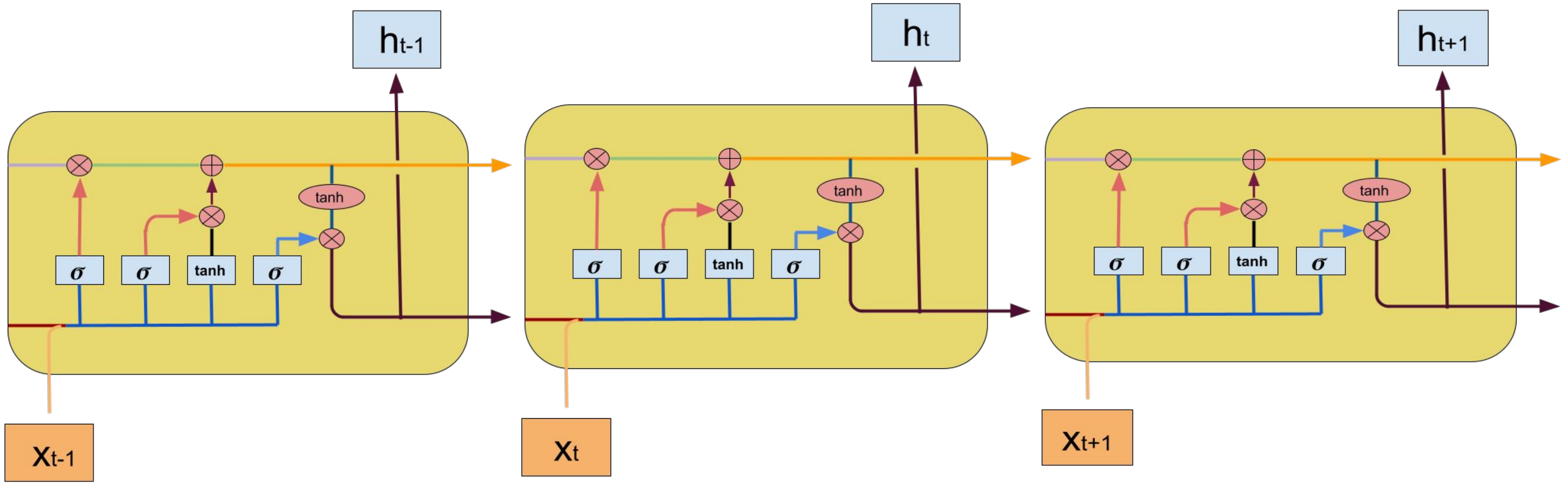
Vanishing and exploding gradient

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



LSTM (Long Short Term Memory)

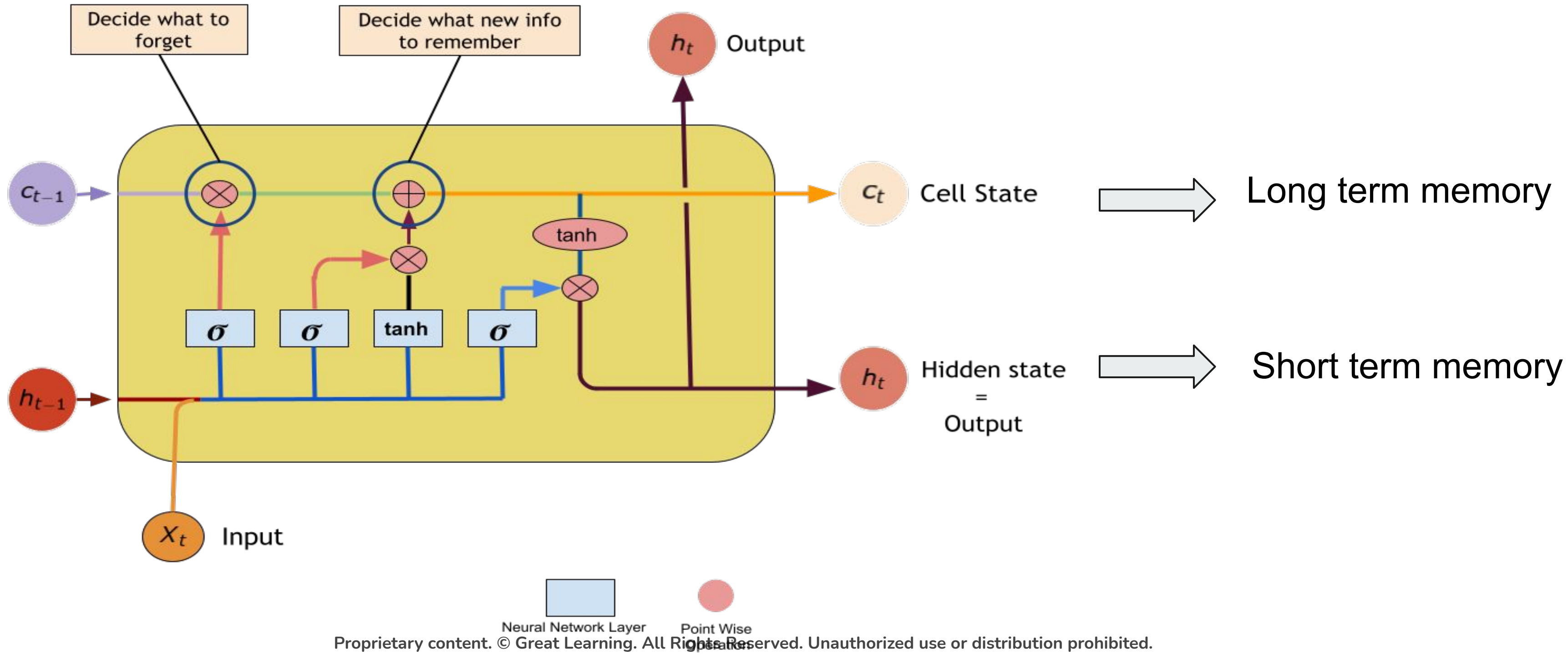
LSTM was designed to overcome the problems of simple Recurrent Network (RNN) by allowing the network to store data in a sort of memory that it can access at a later times.



LSTM is a special type of Recurrent Neural Network (RNN) that can learn long term patterns.

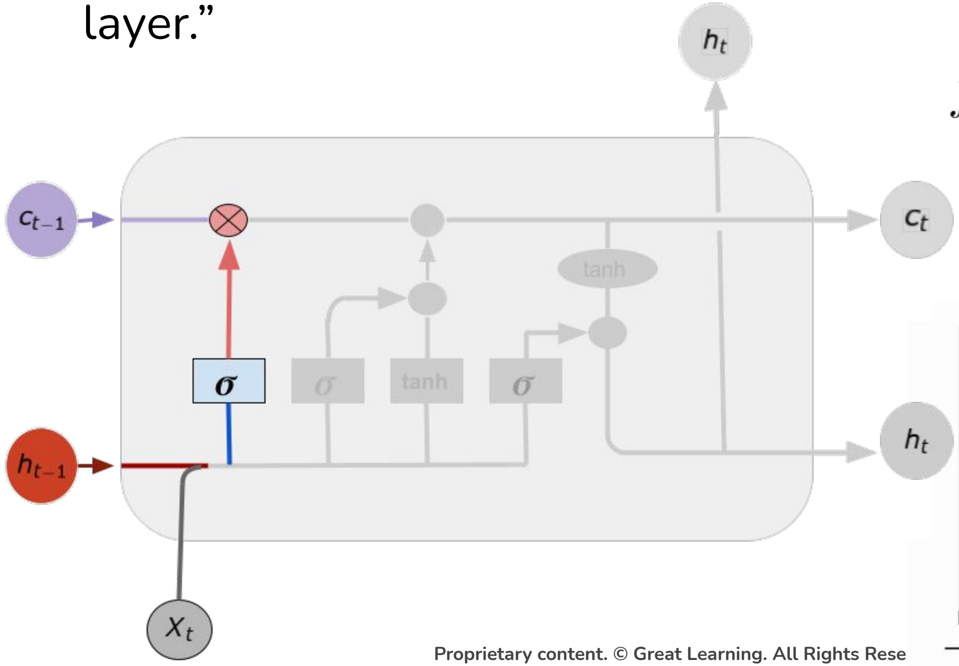
LSTM (Long Short Term Memory)

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram. The cell state is updated twice with few computations that resulting stabilize gradients.



LSTM - Forget Gate

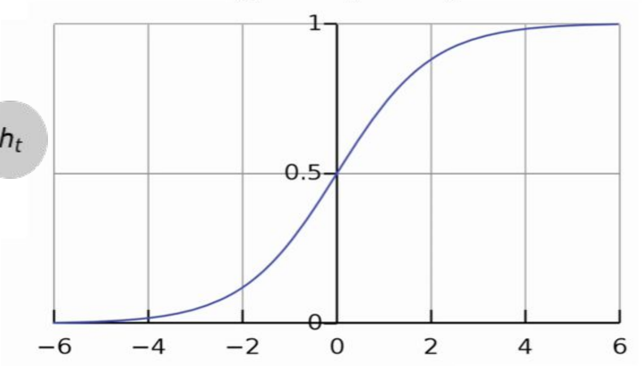
- In LSTM there are Forget Gate, Input Gate and Output Gate that we will go through it one by one.
- The first step is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "Forget Gate layer."



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

0 to forget, 1 to remember

sigmoid [0 to 1]

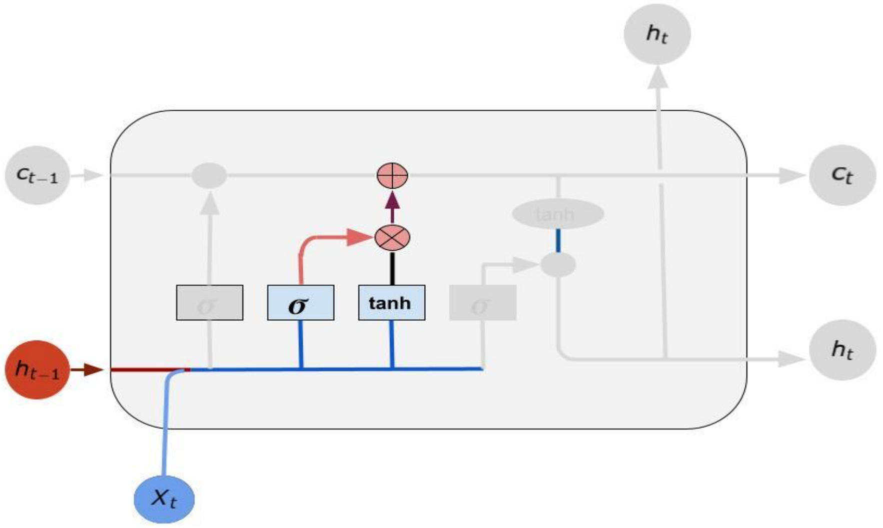


LSTM -Input Gate

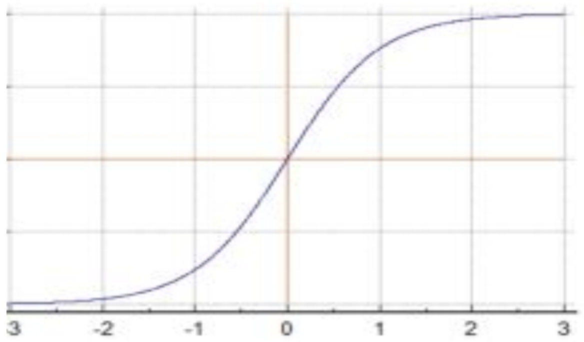
- The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "Input Gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values that could be added to the state.

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

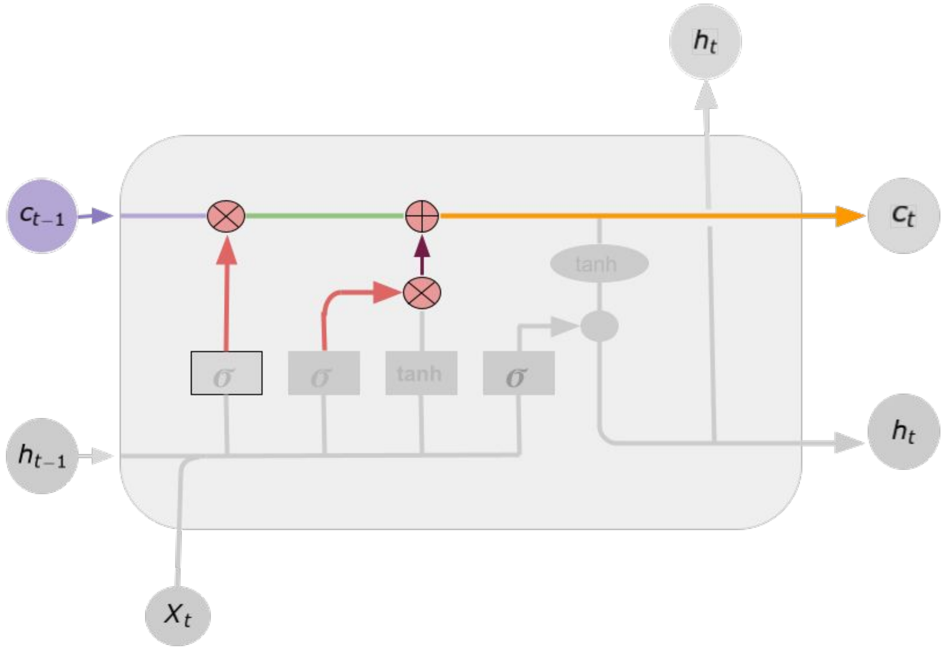
$$\tilde{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$



tanh [-1 to 1]



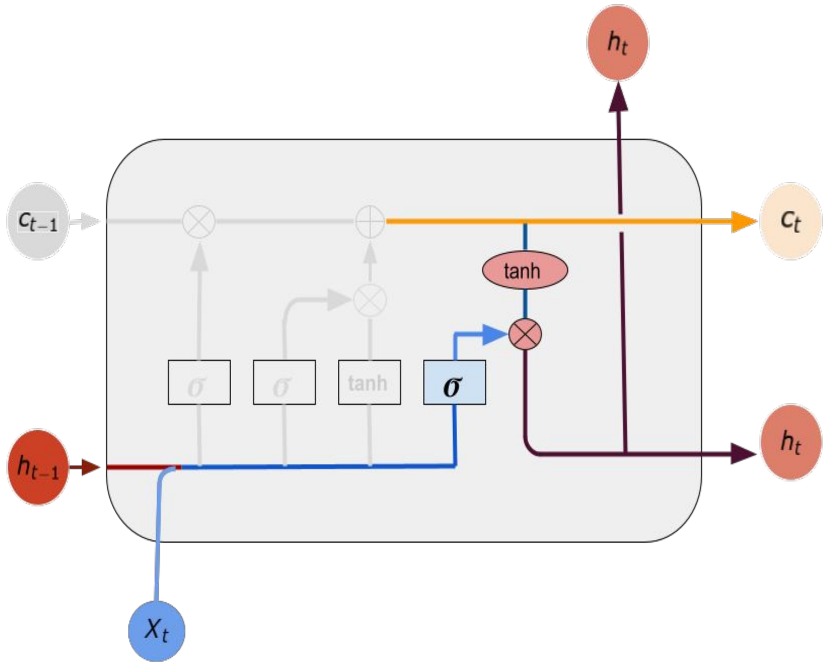
LSTM- Cell State Updates



- In further steps, we'll combine these two(forget & input) to create an update to the cell state.
- Then we update the old cell state into the new cell state.

$$c_t = f_t * c_{t-1} + i_t \tilde{c_t}$$

LSTM - Output Gate

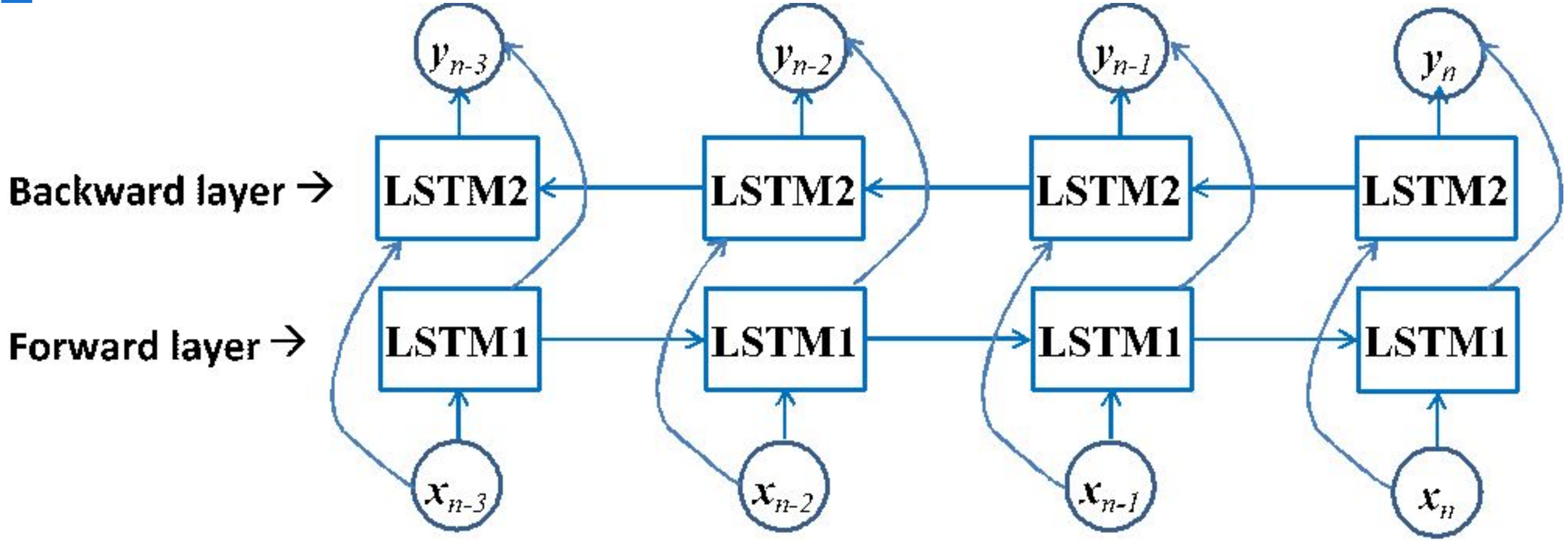


This output will be based on our cell state, but will be a filtered version.

$$o_t = \sigma(W_o [h_{t-1}, x_t] + b_0)$$

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

Bi-directional LSTM



- Many problems require a reverse flow of information as well
- For example, POS tagging may require context from future words

Questions
?

greatlearning
Power Ahead

Happy Learning !