Introduction to Natural Language Processing - NLP

By **Prudvi**

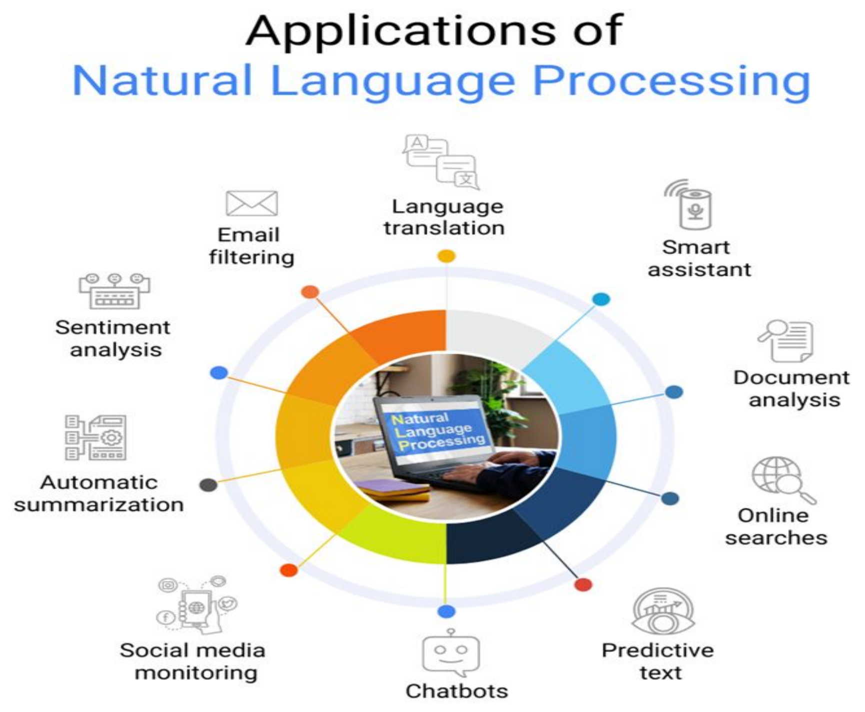
What is NLP

Natural language processing (NLP) is a field of computer science and a subfield of artificial intelligence that aims to make computers understand human language - in the way that it is written, spoken, and organized.

NLP can be divided into two overlapping subfields:
Natural language understanding **(NLU):** focuses on semantic analysis or determining the intended meaning of text.
Natural language generation **(NLG):** focuses on text generation by a machine.

Applications in Real-World Scenarios:

- **Search Engines**: Google uses NLP to rank pages and provide relevant answers.
- **Chatbots**: Customer service bots that handle queries in e-commerce.
- **Translation**: Tools like Google Translate.
- **Sentiment Analysis**: Analyzing social media posts to gauge public sentiment.
- **Summarization**: task of shortening text to highlight the most relevant information.



Structure of Text data

Structure of Text Data:

- Text data is hierarchical:
  - **Corpus**: A collection of documents.
  - **Documents**: A collection of sentences or paragraphs.
  - **Sentences**: A collection of words/tokens.
  - **Words/Tokens**: Words are the basic units of language, representing distinct entities or concepts.
  - **Tokens**: When words are split into smaller units (e.g., by spaces or special characters), they are called tokens.

Word vs. Token:
Words are semantic units. Tokens are syntactic splits.

Subword Tokenization: Break words into subwords to handle unknown words.

Example: "unhappiness" → ["un", "happi", "ness"].

Data Pre-processing

Before a model processes text for a specific task, the text often needs to be preprocessed to improve model performance or to turn words and characters into a format the model can understand.

Various techniques may be used in this data preprocessing:

Tokenization:

- **Definition**: Breaking text into smaller units (words or sentences).

Example:

- Text: "Natural Language Processing is fascinating."
- Tokenized: ["Natural", "Language", "Processing", "is", "fascinating"].

Stopwords:

- **Definition**: Common words like "is," "and," "the" that do not add much value.
- **Why Remove**: Reduces noise in the data for better model performance.

Example:

- Input: "The weather is sunny and pleasant."
- Output after removing stopwords: "weather sunny pleasant."

Stemming and Lemmatization:

- **Stemming**: Stemming is an informal process of converting words to their base forms using heuristic rules.

Example: For example, “university,” “universities,” and “university’s” might all be mapped to the base *univers*.

- limitation : “universe” may also be mapped to *univers*, even though universe and university don’t have a close semantic relationship.

Lemmatization:

Lemmatization is a more formal way to find roots by analyzing a word’s morphology using vocabulary from a dictionary.
Example: "better" → "good," "running" → "run."

Text Normalization:

- Lowercasing, removing punctuation, and correcting misspellings.

Example: "COVID-19 Cases INCREASED!!!" → "covid cases increased.“

There should be other preprocessing steps to be used, like removing URLs, HTML, punctuations, extra spaces etc.

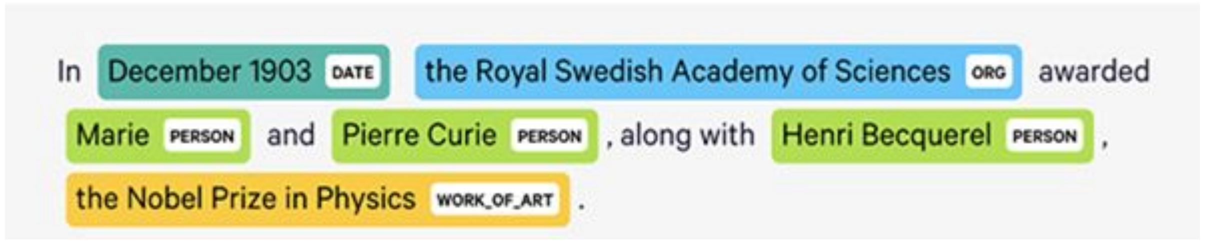
POS tagging & NER

POS Tagging (Part-of-Speech Tagging)

- **POS tagging** involves assigning a part-of-speech (like noun, verb, adjective, etc.) to each word in a sentence. It focuses on categorizing words into grammatical classes based on their role within the sentence.
- **Example**:
  - Sentence: *The dog chased the cat.*
  - POS tags: *The (DT - Determiner), dog (NN - Noun), chased (VBD - Verb), the (DT - Determiner), cat (NN - Noun).*

Named Entity Recognition (NER)

- Named Entity Recognition (NER) identifies and classifies entities like names of people, organizations, and locations within text. NER is crucial for extracting structured information from unstructured text, enabling applications such as information retrieval, question answering, and content recommendation.

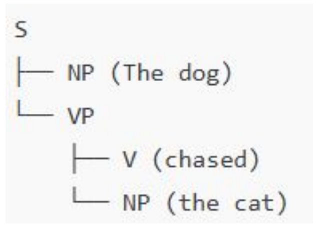

In December 1903 DATE the Royal Swedish Academy of Sciences ORG awarded Marie PERSON and Pierre Curie PERSON , along with Henri Becquerel PERSON , the Nobel Prize in Physics WORK_OF_ART .

Parsing Techniques in NLP

Parsing techniques analyze the grammatical structure of sentences to understand their syntax and relationships between words.

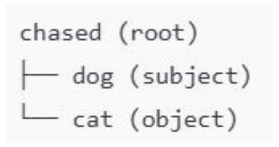
- **Syntactic parsing** (which includes **dependency parsing** and **constituency parsing**) goes beyond POS tagging by analyzing the **structure** of a sentence and the relationships between its words. It identifies how words are related to each other to form meaningful units, such as phrases (NP, VP, etc.), and ultimately creates a syntactic tree.
- **Example**:
  - **Sentence:** ***The dog chased the cat.***
- Constituency Tree: The sentence can be broken down into **noun phrase (NP)** and **verb phrase (VP)**, showing how components like "The dog" (NP) and "chased the cat" (VP) are related.
- Dependency Graph: In a dependency structure, **chased** would be the root, and there would be dependencies like "dog" → "chased" (subject), "cat" → "chased" (object), etc.
- Syntactic parsing provides the grammatical **relationships** between words and how they combine to form a valid sentence structure.

Syntactic parsing



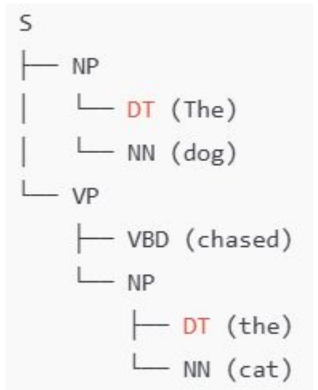
```
S
├── NP (The dog)
└── VP
    ├── V (chased)
    └── NP (the cat)
```

dependency parsing



```
chased (root)
├── dog (subject)
└── cat (object)
```

constituency parsing



```
S
├── NP
│   └── DT (The)
│   └── NN (dog)
└── VP
    ├── VBD (chased)
    └── NP
        ├── DT (the)
        └── NN (cat)
```

Parsing Techniques in NLP

Feature	Dependency Parsing	Syntactic Parsing
Output	Word-to-word relationships (flat)	Hierarchical structure (phrases)
Focus	Grammatical relationships	Full grammatical structure
Representation	Dependency tree	Constituency tree
Applications	Information extraction, QA, semantics	Grammar checking, translation, TTS
Efficiency	Faster, simpler	Slower, more complex
Best for	Semantic tasks	Structural tasks

Data Tools and Libraries

- **NLTK**: Ideal for beginners; supports tokenization, stemming, etc.
- **SpaCy**: This open-source library is designed for efficient NLP preprocessing and has a wide range of features, including tokenization, part-of-speech tagging, dependency parsing, and named entity recognition.
- **Gensim:** This library provides tools for preprocessing text data, including tokenization, stopword removal, and lemmatization. It also has a wide range of algorithms for topic modelling and document similarity analysis. Implements algorithms like Word2Vec for word embeddings and doc2vec for document embeddings.
- **Pattern**: This library is a web mining module for Python that provides tools for NLP tasks, including tokenization, part-of-speech tagging, and spelling correction.
- **TextBlob**: This library provides a simple interface for common NLP tasks, such as tokenization, part-of-speech tagging, and sentiment analysis. It is built on top of the NLTK library.
- **Stanford CoreNLP**: This suite of NLP tools from Stanford University includes a wide range of capabilities, including tokenization, part-of-speech tagging, named entity recognition, and parsing. It is available as a standalone Java application or as a Python wrapper.
- **sklearn:** Although a general-purpose machine learning library, Scikit-learn offers robust tools for text analysis and feature extraction.
- **Hugging Face Transformers**: Access to pre-trained models like BERT, GPT.

Feature extraction or text represention

Most conventional machine-learning techniques work on the features – generally numbers that describe a document in relation to the corpus that contains it. Below are the feature extraction techniques or methods.

- One hot encoding
- Bag-of-Words,
- TF-IDF
- Word Embeddings
  - Word2Vec,
  - GLoVE
  - Learning the features during the training process of a neural network. RNN,LSTM,BERT,GPT etc.

One-Hot Encoding

In one hot encoding scheme, each word in the vocabulary is represented as a unique vector.

1. Dimensionality of the vector is equal to the size of the vocabulary.
2. All elements set to 0, except for the word

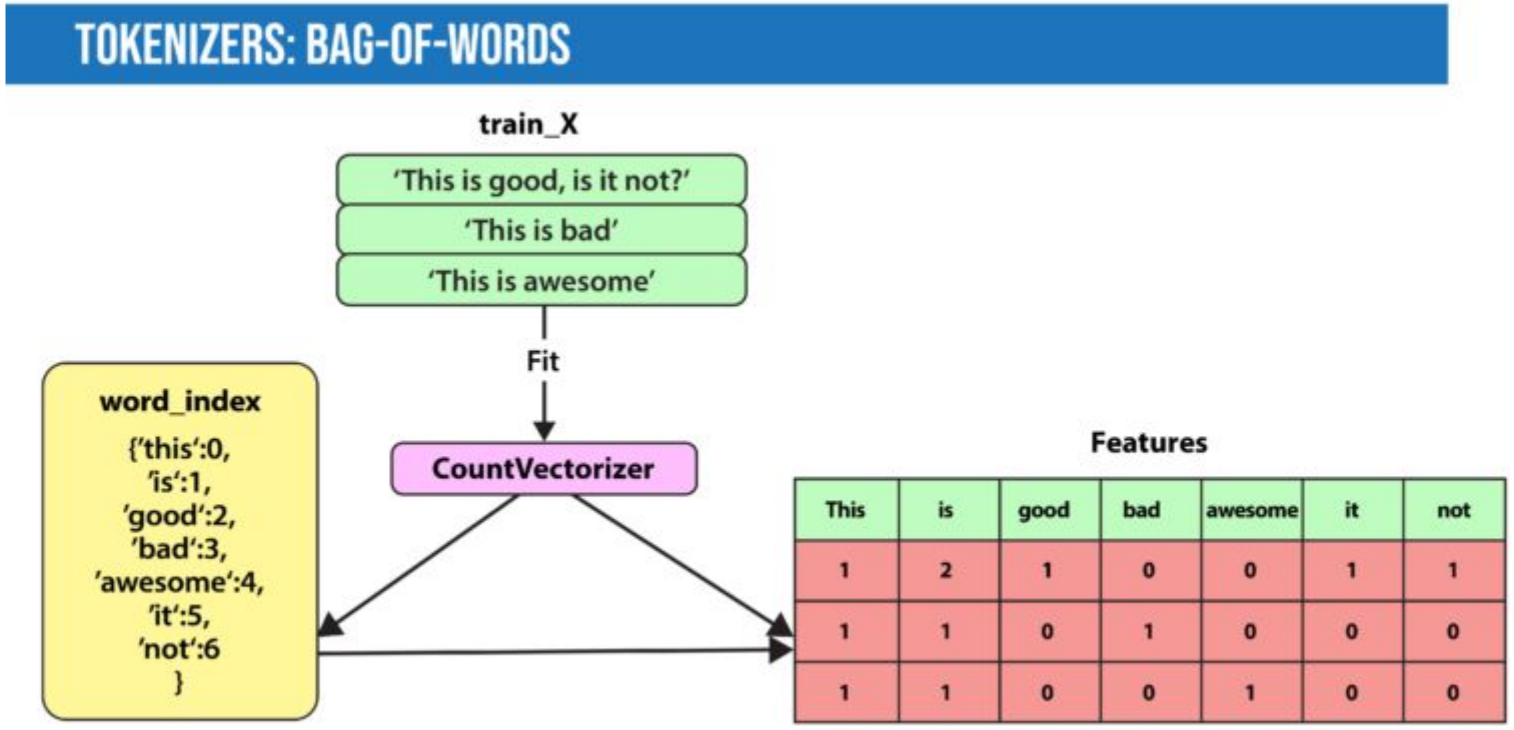
example_texts = "cat in the hat ; dog on the mat ; bird in the tree”

Vocabulary: {'mat', 'the', 'bird', 'hat', 'on', 'in', 'cat', 'tree', 'dog'}

Sentence	mat	the	bird	hat	on	in	cat	tree	dog
"cat in the hat"	0	1	0	1	0	1	1	0	0
"dog on the mat"	1	1	0	0	1	0	0	0	1
"bird in the tree"	0	1	1	0	0	1	0	1	0

Bag of words (BoW)

The Bag-of-Words model represents text as a collection (or "bag") of words, without considering word order or grammar. It creates a vocabulary of all unique words in the corpus and counts the occurrence of each word in a document.



This	is	good	bad	awesome	it	not
1	2	1	0	0	1	1
1	1	0	1	0	0	0
1	1	0	0	1	0	0

Bag-of-Words (through the CountVectorizer method) encodes the total number of times a document uses each word in the associated corpus.

TF-IDF

Term Frequency-Inverse Document Frequency (TF-IDF) is an improvement over Bag-of-Words that accounts for the importance of words by considering their frequency in a document relative to the entire corpus. Words that are common across many documents (like "the", "is") get less weight, while rare but significant words get higher weight.

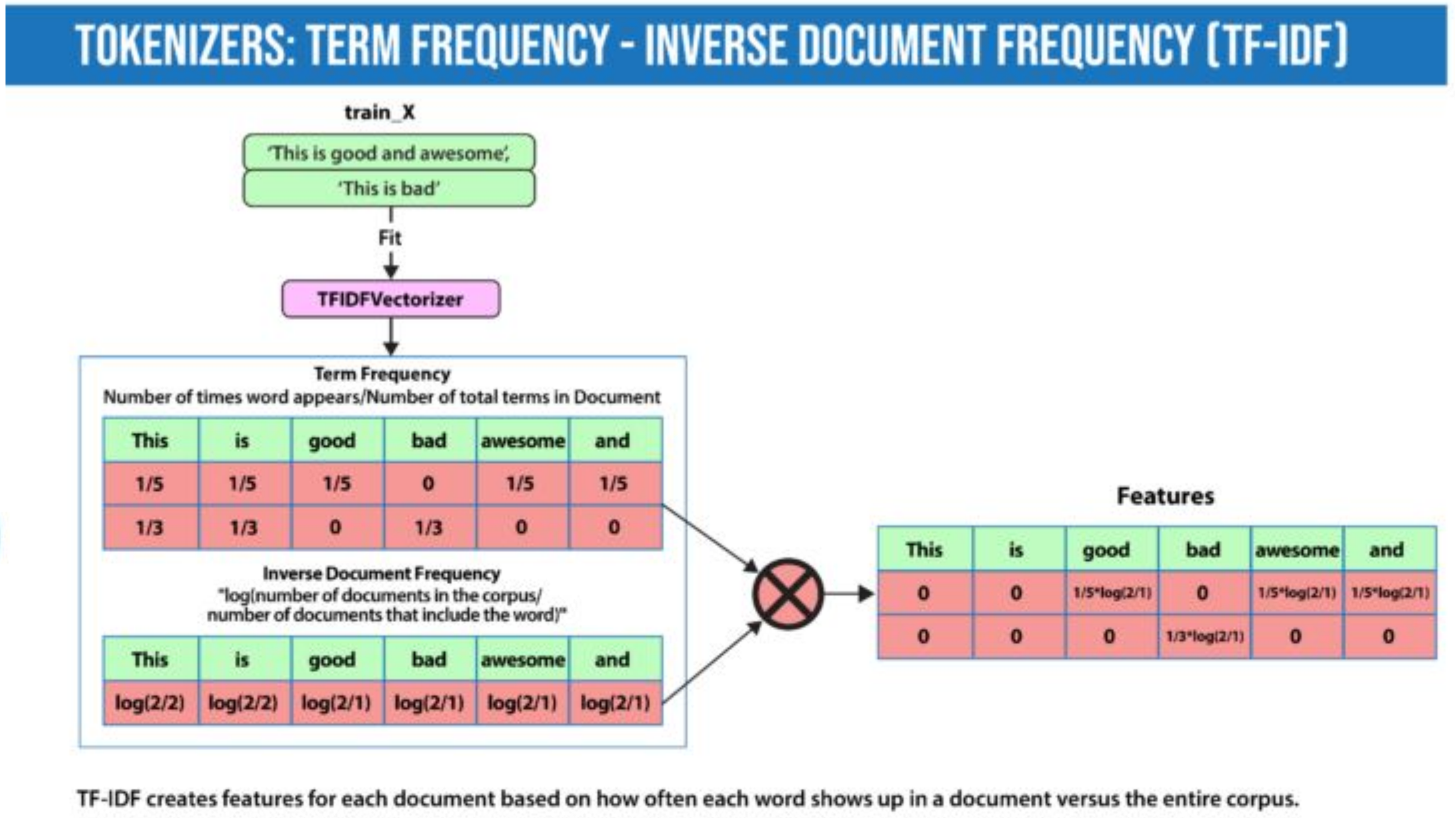
- **Term Frequency:** How important is the word in the document?

$$\mathrm{TF}(\text{word}) = \frac{\text{Number of times word appears in the document}}{\text{Total words in the document}}$$

- **Inverse Document Frequency:** How important is the term in the whole corpus?

$$\mathrm{IDF}(\text{word}) = \log\left(\frac{\text{Total number of documents}}{\text{Number of documents containing the word}}\right)$$

TF-IDF(word)=TF(word)×IDF(word)


TOKENIZERS: TERM FREQUENCY - INVERSE DOCUMENT FREQUENCY (TF-IDF)
train_X
'This is good and awesome',
'This is bad'
Fit
TFIDFVectorizer

Term Frequency
Number of times word appears/Number of total terms in Document

This	is	good	bad	awesome	and
1/5	1/5	1/5	0	1/5	1/5
1/3	1/3	0	1/3	0	0

Inverse Document Frequency
"log(number of documents in the corpus/ number of documents that include the word)"

This	is	good	bad	awesome	and
log(2/2)	log(2/2)	log(2/1)	log(2/1)	log(2/1)	log(2/1)

Features

This	is	good	bad	awesome	and
0	0	1/5*log(2/1)	0	1/5*log(2/1)	1/5*log(2/1)
0	0	0	1/3*log(2/1)	0	0

TF-IDF creates features for each document based on how often each word shows up in a document versus the entire corpus.

Word Embeddings – Word2Vec

Word embedding's represent words as continuous vector spaces, which capture semantic relationships between words.

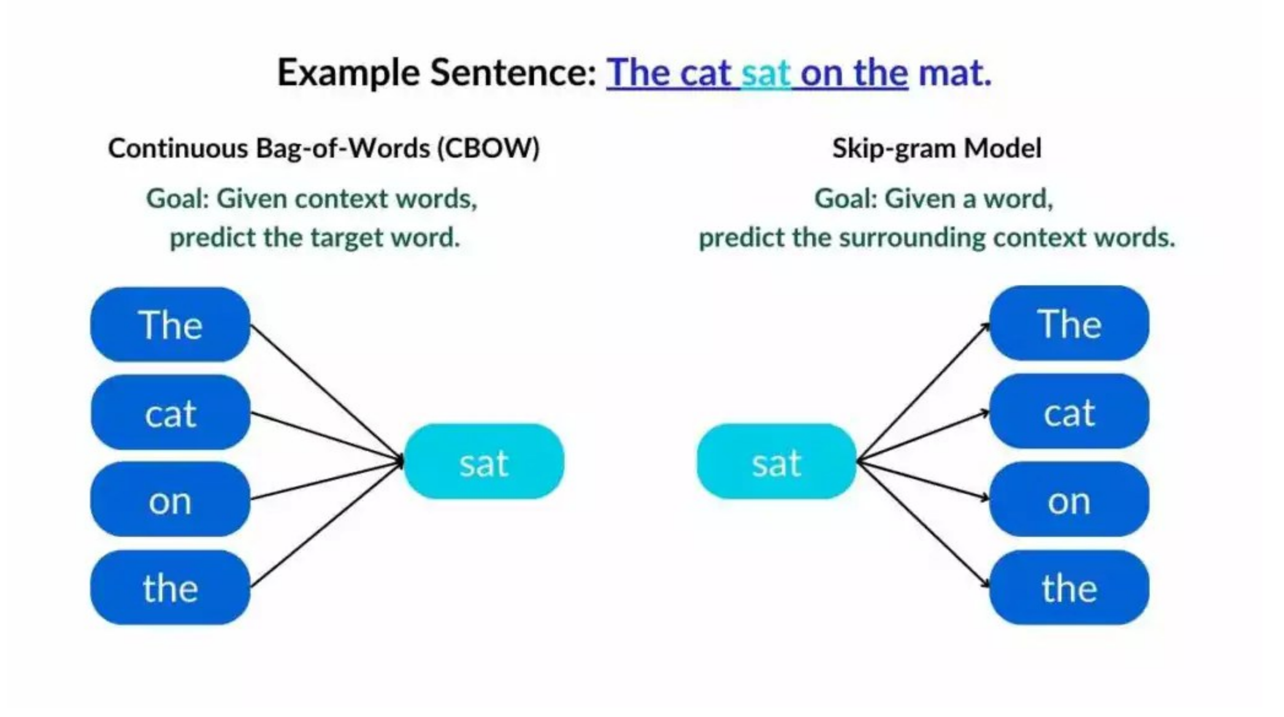
Ex: Wor2Vec,Glove other DNN models

Word2Vec: It is a neural approach for generating word embeddings.

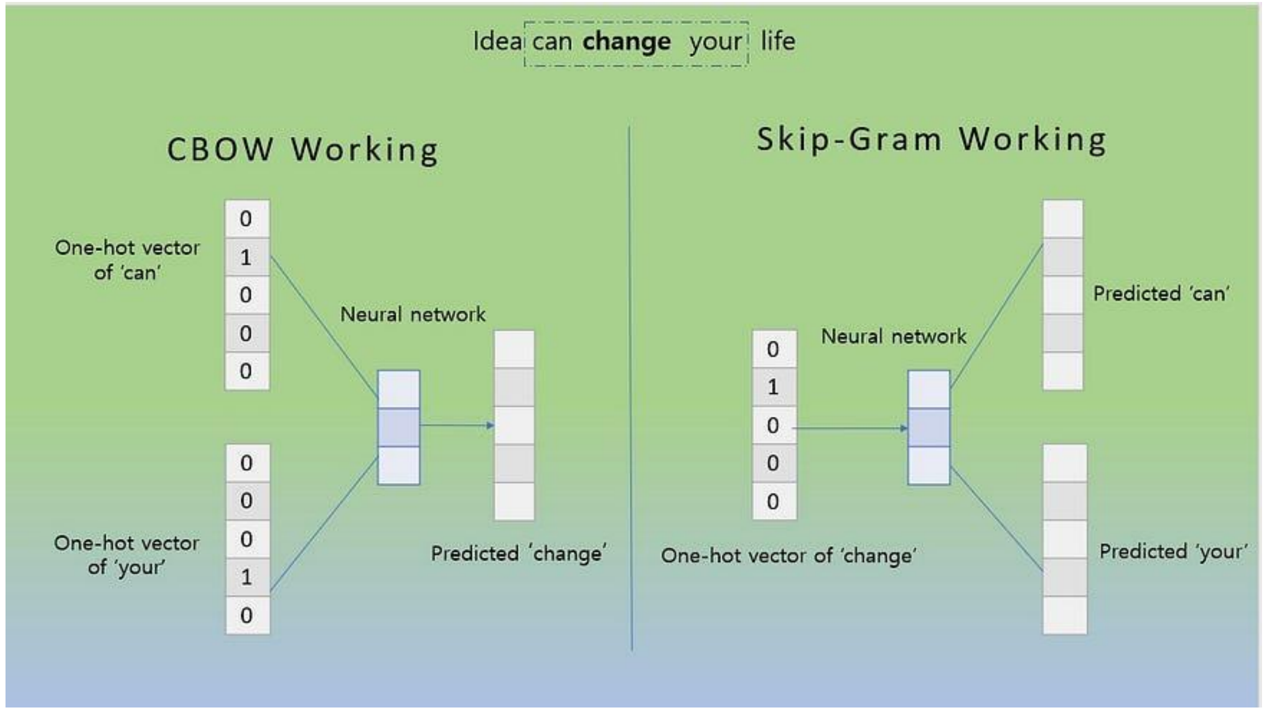
Two primary models within Word2Vec are
- Skip-gram
- Continuous Bag of Words (CBOW).

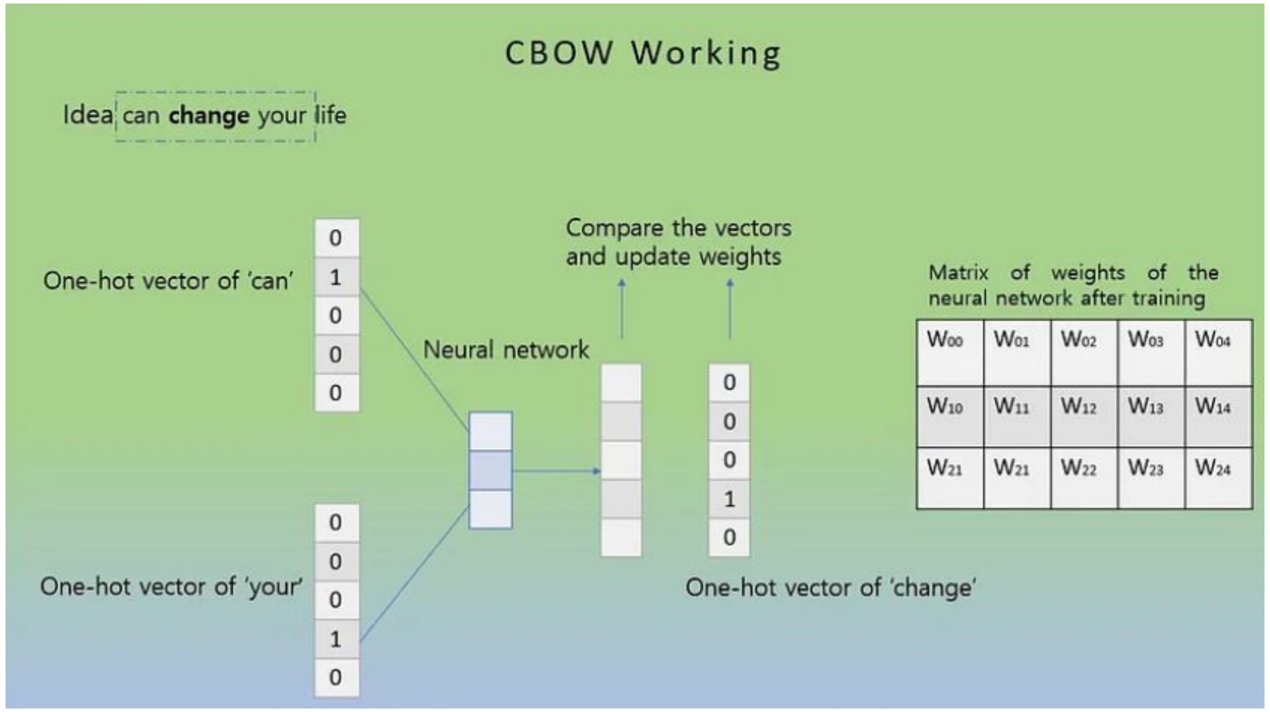
- **Skip-gram:** In the Skip-gram model, the algorithm predicts the context words given a target word. This helps capture a word's semantic meaning by considering its surrounding context.

- **Continuous Bag of Words (CBOW):** Contrastingly, CBOW predicts the target word based on its context. It is adequate for representing the syntactic structure of words.


Example Sentence: The cat sat on the mat.
Continuous Bag-of-Words (CBOW)
Goal: Given context words, predict the target word.
The
cat
on
the
sat
Skip-gram Model
Goal: Given a word, predict the surrounding context words.
sat
The
cat
on
the

Word Embeddings – Word2Vec




CBOW Working
Idea can change your life
One-hot vector of 'can'
One-hot vector of 'your'
Neural network
Compare the vectors and update weights
One-hot vector of 'change'
Matrix of weights of the neural network after training

W_{00}	W_{01}	W_{02}	W_{03}	W_{04}
W_{10}	W_{11}	W_{12}	W_{13}	W_{14}
W_{21}	W_{21}	W_{22}	W_{23}	W_{24}

Word Embeddings - GloVe

GloVe (Global Vectors for Word Representation): GloVe is another influential word embedding technique focusing on global word co-occurrence statistics. It constructs an embedding matrix by analyzing the frequency of word co-occurrence in a large corpus.

GloVe is similar to Word2Vec as it also learns word embeddings, but it does so by using matrix factorization techniques rather than neural learning. The GLoVE model builds a matrix based on the global word-to-word co-occurrence counts.

The words which occur next to each other get a value of 1, if they are one word apart then 1/2, if two words apart then 1/3 and so on. Let us take an example to understand how the matrix is created. We have a small corpus:

Corpus:
It is a nice evening.
Good Evening!
Is it a nice evening?

Word	it	is	a	nice	evening	good
it	0	1	0.5	0.333	0.25	0
is	1	0	1	0.5	0.333	0
a	1/2	1	0	1	0.5	0
nice	0.333	0.5	1	0	1	0
evening	0.25	0.333	0.5	1	0	1
good	0	0	0	0	1	0

Why Use Weighted Co-Occurrence?
Words closer to the target word in context often have a stronger relationship, so they are weighted more heavily.
This helps GloVe learn more accurate embeddings by prioritizing meaningful relationships.

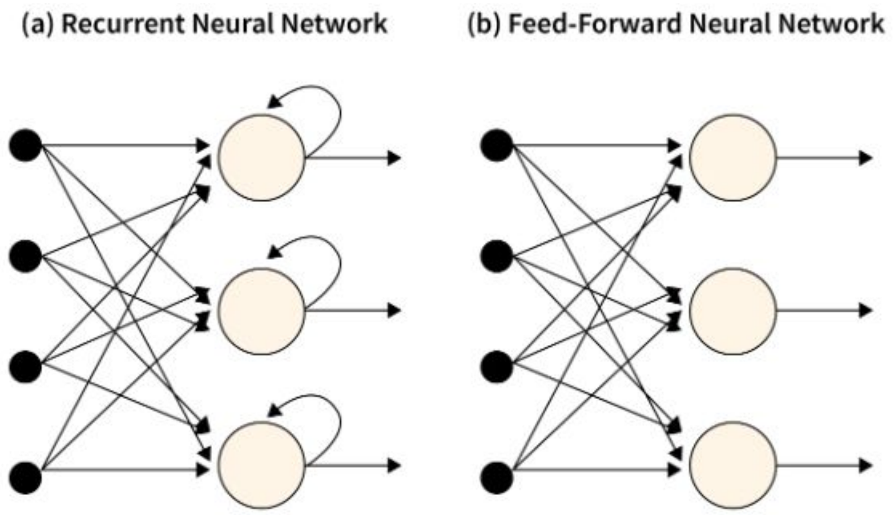
Recurrent NN (RNN)

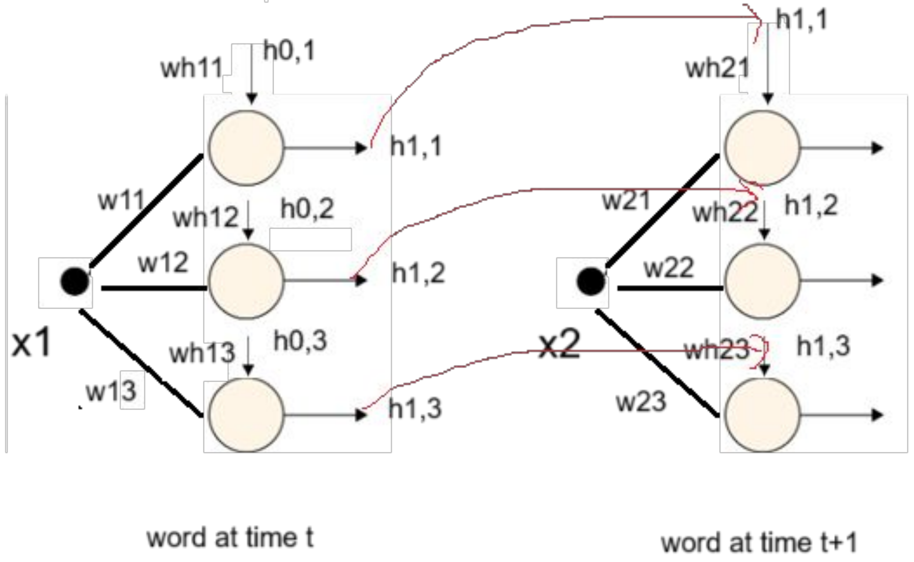
A **recurrent neural network(RNN)** in a deep learning model contains a feedback loop that predicts the output using the previous hidden state along the input.

Standard Feedforward Neural Networks are only suitable for independent data points. To include the dependencies between these data points, we must change the neural network if the data are organized in a sequence where each data point depends on the one before.

The idea of "memory" in RNNs enables them to store the states or details of earlier inputs to produce the subsequent output in the sequence. h(t) acts as the network's "memory".

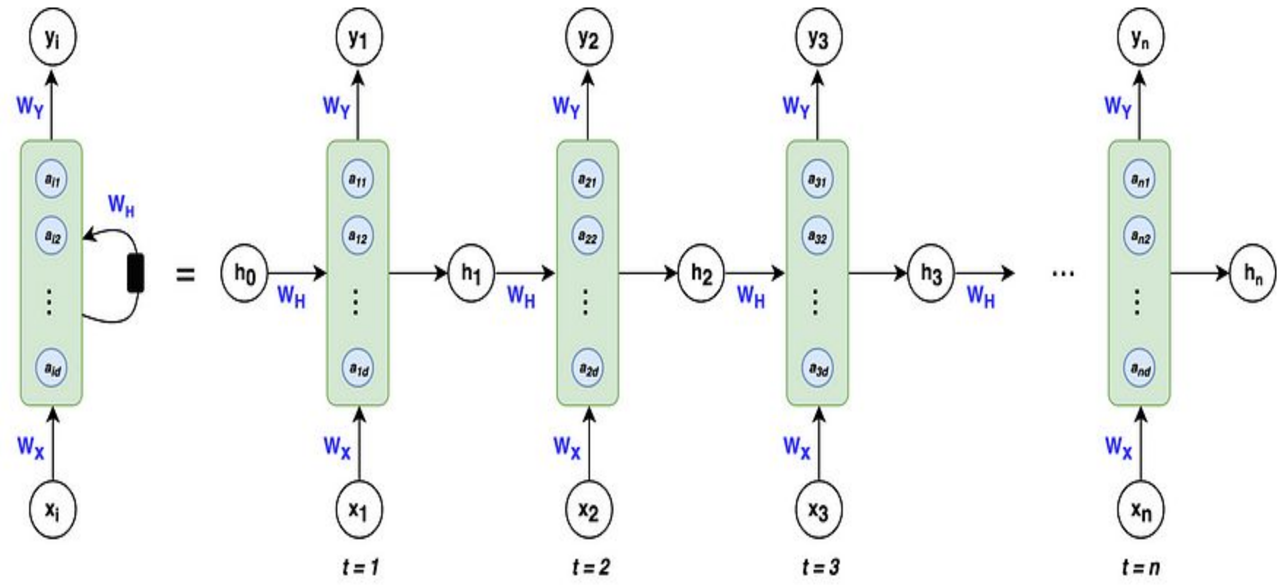
Backward Propagation Through Time (BPTT)
The Backpropagation Through Time (BPTT) technique applies the Backpropagation training method to the recurrent neural network in a deep learning model trained on sequence data, such as time series. Since RNN neural network processes sequence one step at a time, gradients flow backward across time steps during this backpropagation process.





RNN

Recurrent Neural Networks (RNNs) are a type of neural network designed to handle sequential data.
Sentences are also sequential, “I love dogs” has a different meaning than “Dogs I love.”



$$a_t = \boxed{W_H h_{t-1} + W_X X_t}$$ **Hidden Nodes**

Activation Function

Output From Hidden State $\longrightarrow$ $$h_t = \underbrace{\boxed{tanh(\boxed{a_t})}}_{\textbf{Hidden State}}$$

Prediction at time t $\longrightarrow$ $$y_t = softmax(W_Y h_t)$$

$$a_1 = \begin{pmatrix} W_{H,11} & W_{H12} & W_{H,13} \\ W_{H,21} & W_{H,22} & W_{H,23} \\ W_{H,31} & W_{H,32} & W_{H,33} \end{pmatrix} \begin{pmatrix} h_{0,1} \\ h_{0,2} \\ h_{0,3} \end{pmatrix} + \begin{pmatrix} W_{X,11} & W_{X,12} & W_{X,13} & W_{X,14} \\ W_{X,21} & W_{X,22} & W_{X,23} & W_{X,24} \\ W_{X,31} & W_{X,32} & W_{X,33} & W_{X,34} \end{pmatrix} \begin{pmatrix} x_{1,1} \\ x_{1,2} \\ x_{1,3} \\ x_{1,4} \end{pmatrix} = \begin{pmatrix} a_{1,1} \\ a_{1,2} \\ a_{1,3} \end{pmatrix}$$

$$h_1 = tanh(\begin{pmatrix} a_{1,1} \\ a_{1,2} \\ a_{1,3} \end{pmatrix}) = \begin{pmatrix} h_{1,1} \\ h_{1,2} \\ h_{1,3} \end{pmatrix}$$

$$y_1 = softmax(\begin{pmatrix} W_{Y,11} & W_{Y,12} & W_{Y,13} \\ W_{Y,21} & W_{Y,22} & W_{Y,23} \\ W_{Y,31} & W_{Y,32} & W_{Y,33} \\ W_{Y,41} & W_{Y,42} & W_{Y,43} \end{pmatrix} \begin{pmatrix} h_{1,1} \\ h_{1,2} \\ h_{1,3} \end{pmatrix}) = \begin{pmatrix} y_{1,1} \\ y_{1,2} \\ y_{1,3} \\ y_{1,4} \end{pmatrix}$$

Word2vec Vs RNN

Feature	Word2Vec	RNN
Input Type	Individual words (independent of order).	Sequential data (order matters).
Output	Static word embeddings.	Context-sensitive outputs.
Context Handling	Context-free (word meaning doesn't change).	Context-sensitive (dynamic meaning).
Use Case	Pre-trained embeddings for downstream models.	Tasks requiring sequence understanding (e.g., translation, sentiment analysis).
Memory of Past Words	None.	Maintains memory through hidden state.
Post-Training / inference Use	Only the embeddings (lookup table) are used, **not** the model.	The trained RNN model is used to process sequences.

RNN

Advantages and Disadvantages of Recurrent Neural Networks

Advantages of the Recurrent Neural Networks are:

- The processing of sequential data.
- Ability to remember and preserve primary outcomes.
- When calculating new results, consider the most recent and previous results.
- The model size does not change as the input size does. Over time, it distributes weights to other components.

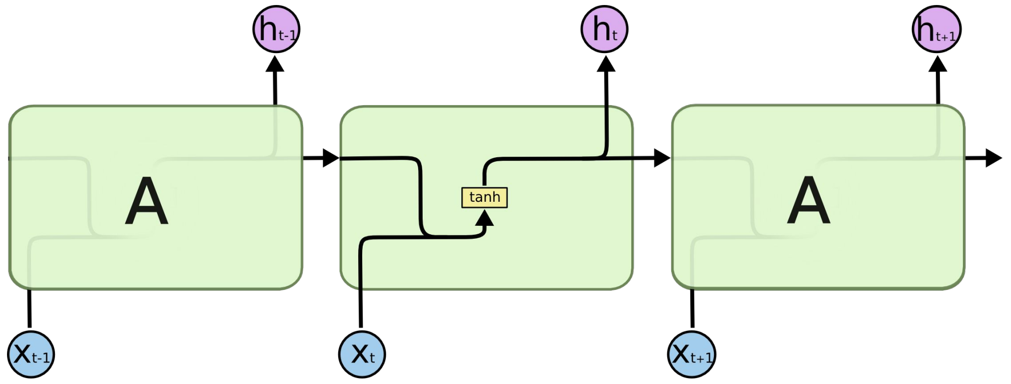
Some of the **disadvantages** of RNN are:

- Problems with vanishing and gradient explosion.
- The challenge of training an RNN is quite challenging.
- Using Tanh or Relu as an activation feature prevents it from processing long sequences.

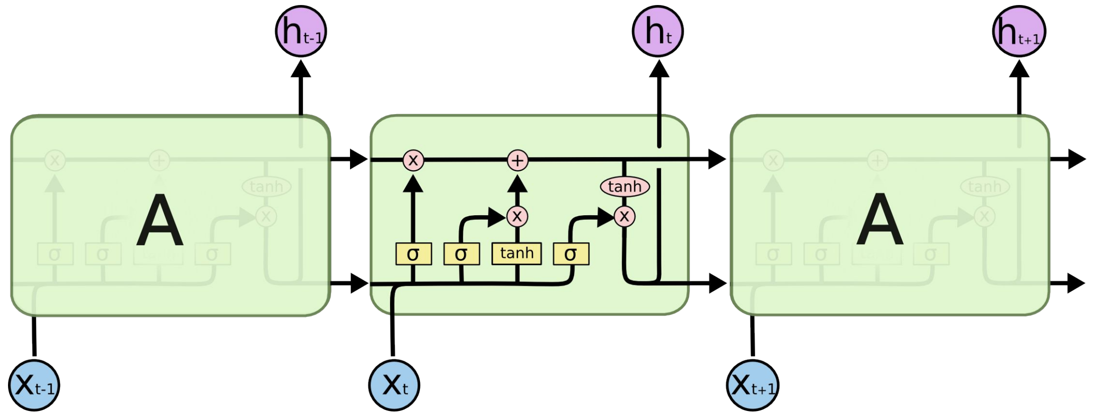
LSTM

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.



The repeating module in a standard RNN

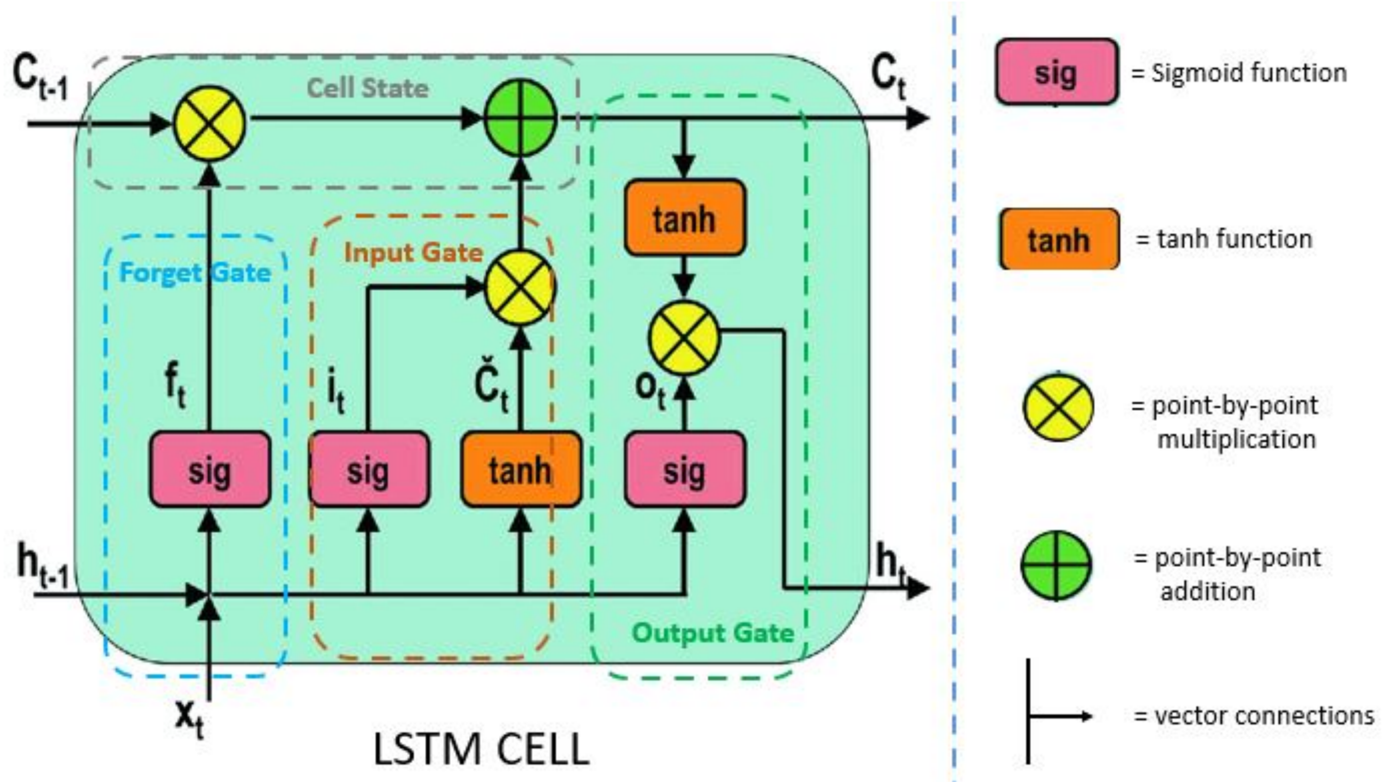


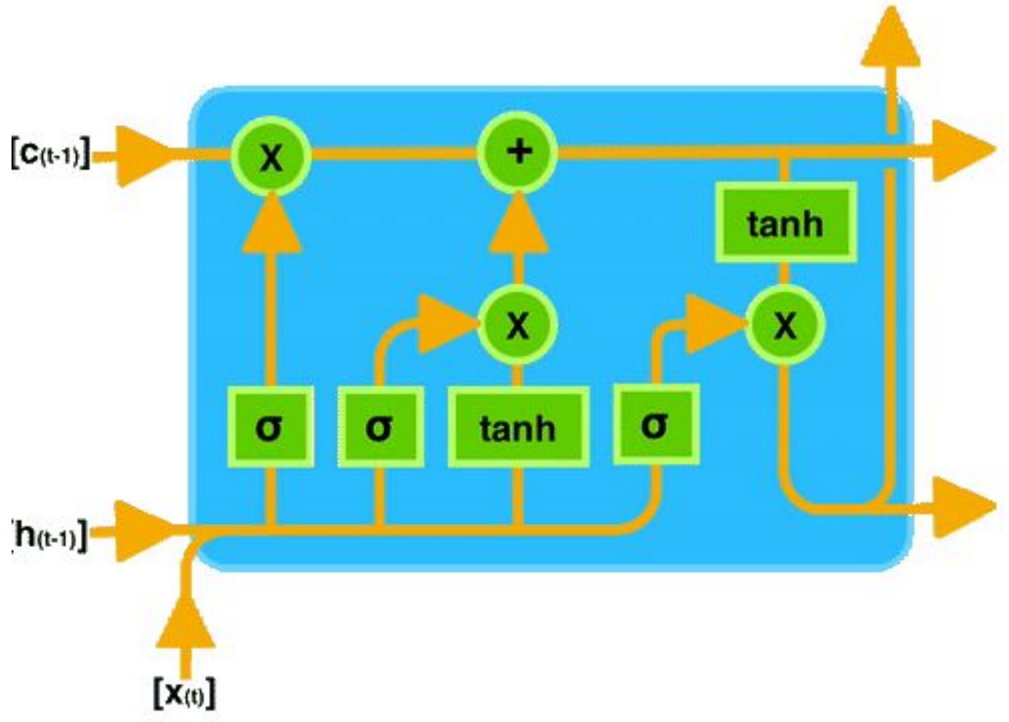
The repeating module in an LSTM

LSTM - Gates

LSTM contains three different gates in an LSTM cell

1. Forget Gate - determines which relevant information from the prior steps is needed.
2. Input Gate - input gate decides what relevant information can be added from the current step
3. Output Gate - output gates finalize the next hidden state

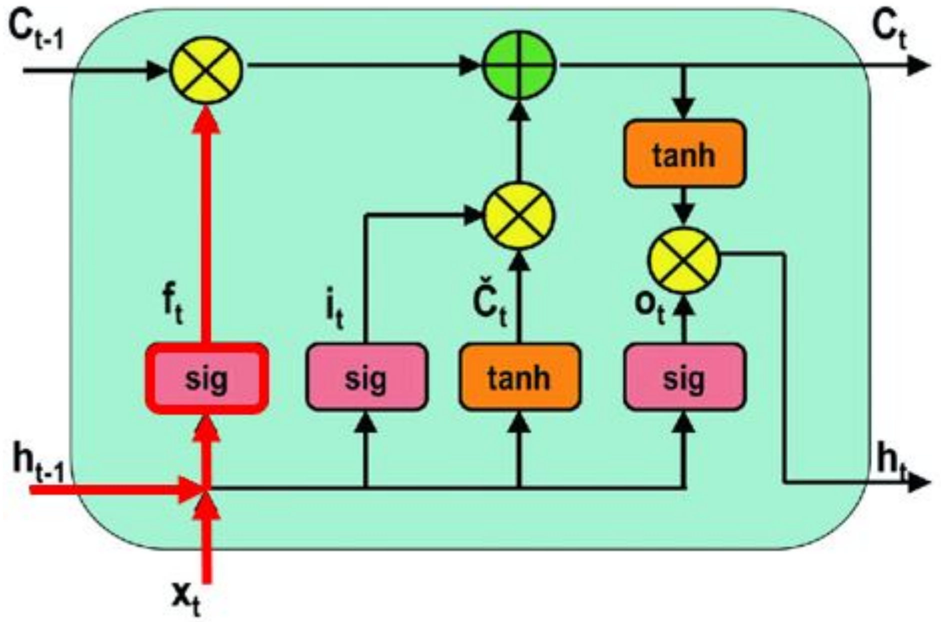




LSTM – Forget Gate

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 h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}.
- A 1 represents "completely keep this" while a 0 represents "completely get rid of this."



Forget Gate Operation

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

$t = timestep$

$f_t = forget\ gate\ at\ t$

$x_t = input$

$h_{t-1} = Previous\ hidden\ state$

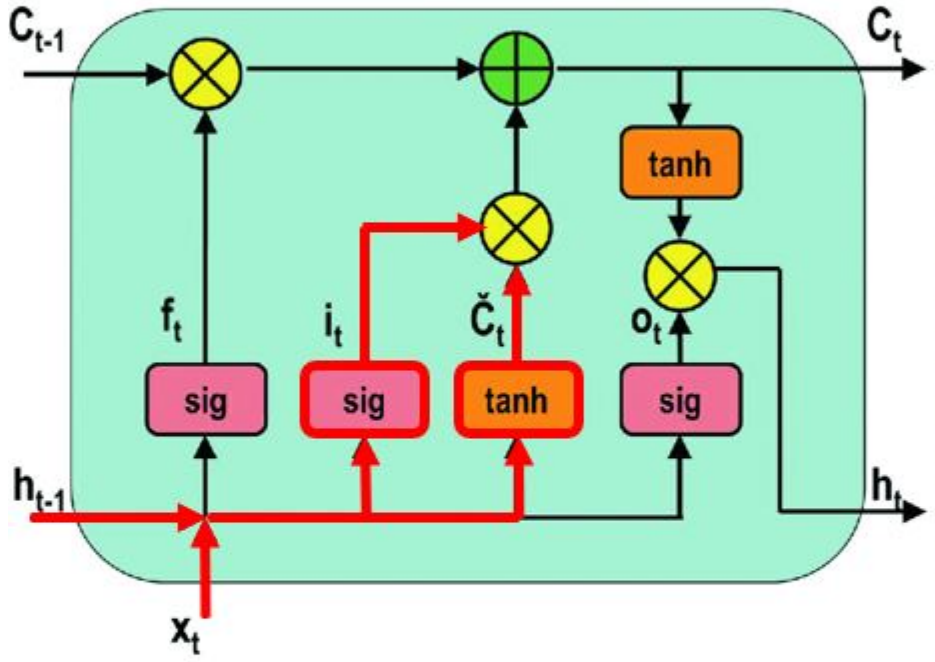
$W_f = Weight\ matrix\ between\ forget\ gate\ and\ input\ gate$

$b_t = connection\ bias\ at\ t$

LSTM – Input Gate

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, C~t, that could be added to the state.
- In the next step, we'll combine these two to create an update to the state.



Input Gate Operation

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

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

$t = timestep$

$i_t = input\ gate\ at\ t$

$W_i = Weight\ matrix\ of\ sigmoid\ operator\ between\ input\ gate\ and\ output\ gate$

$b_t = \ bias\ vector\ at\ t$

$C{\sim}_t = value\ genrated\ by\ tanh$

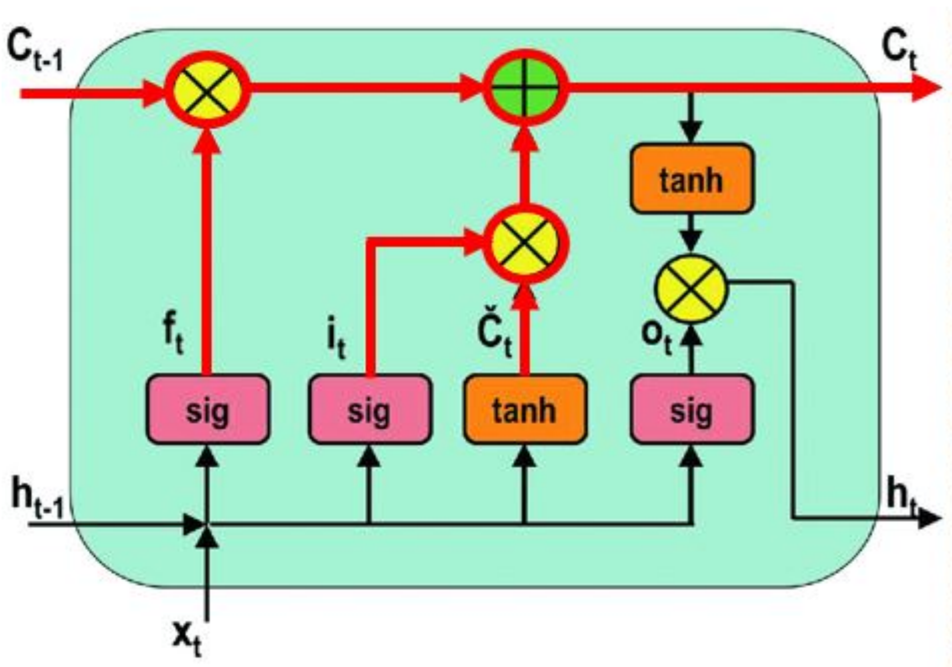
$W_c = \ Weight\ matrix\ of\ tanh\ operator\ between\ cell\ state\ information\ and\ network\ output$

$b_c = \ bias\ vector\ at\ t, w.r.t\ W_c$

LSTM – Cell state

Cell state:

- The next step is to decide and store the information from the new state in the cell state.
- The previous cell state C(t-1) gets multiplied with forget vector f(t). If the outcome is 0, then values will get dropped in the cell state.
- Next, the network takes the output value of the input vector i(t) and performs point-by-point addition, which updates the cell state giving the network a *new cell state C(t*)



Cell State Operation

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

$t = timestep$

$C_t = cell\ state\ information$

$f_t = forget\ gate\ at\ t$

$i_t = input\ gate\ at\ t$

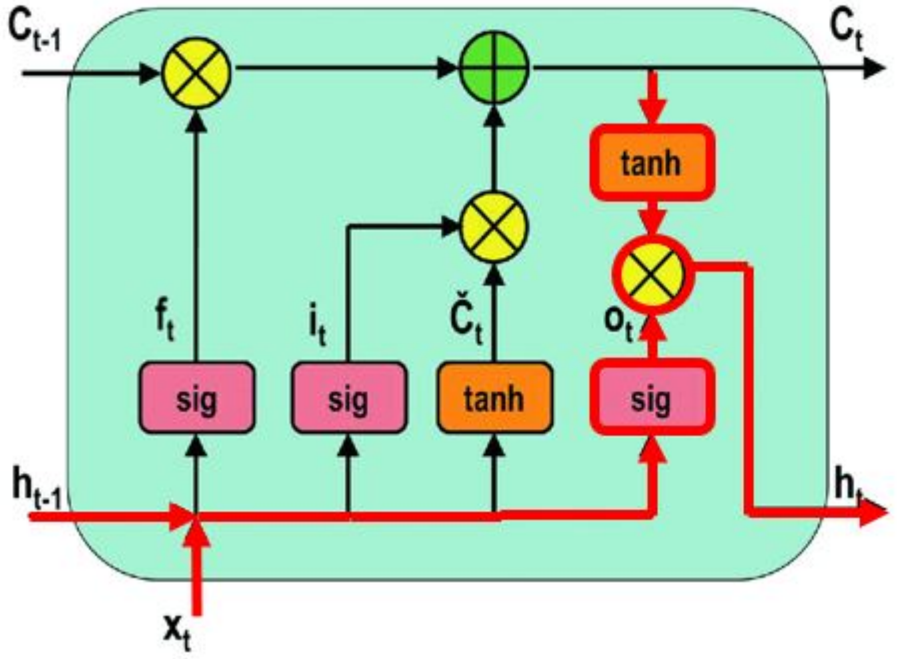
$C_{t-1} = Previous\ timestemp$

$C\sim_t = value\ genrated\ by\ tanh$

LSTM – Output Gate

Output Gate:

- 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.



Output Gate Operation

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

$t = timestep$

$O_t = output\ gate\ at\ t$

$W_o = Weight\ matrix\ of\ output\ gate$

$b_o = bias\ vector, w.r.t\ W_o$

$h_t = LSTM\ output$

Finally, the new cell state and new hidden state are carried over to the next time step.

Thank you