NLP

Word2vec

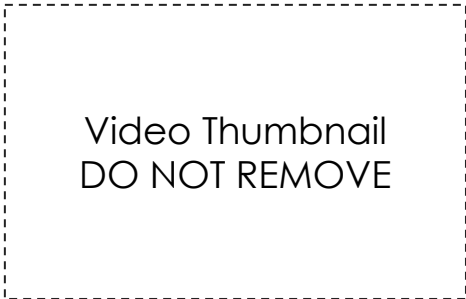
Natural Language Processing

Agenda

- Introduction to Word2vec
- CBOW and Skip-gram
- Limitations of Word2vec

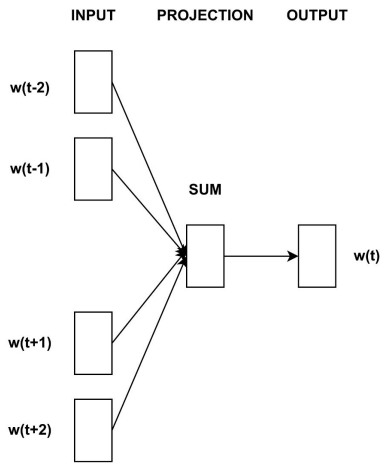
Introduction to Word2vec

- Word2vec is a **two-layer neural network-based method for efficiently creating word embeddings**.
- It was developed in **2013 by Tomas Mikolov et al**. at **Google** as a response to make neural-network-based embedding training more efficient, and it has since become **the de facto standard for building pre-trained word embeddings**.
- Word2vec **takes a text corpus as input** and **returns a set of vectors known as feature vectors** that represent the words in that corpus.
- Note that **Word2vec is by itself not a deep neural network**, but it converts text to a numerical format that ML models can understand.

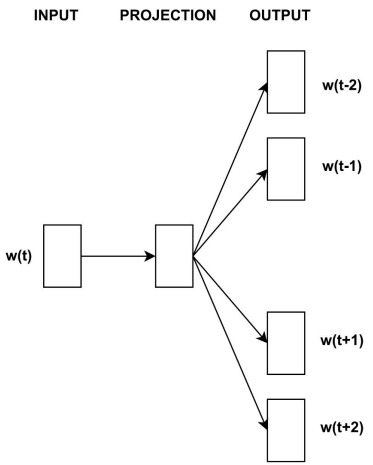


Introduction to Word2vec

- Word2vec has **two models which are used to train words against their neighboring words** in the corpus, namely:
  - **Continuous Bag of Words (CBOW)** - context to predict the target word
  - **Skip Gram Model** - word to predict a target context



CBOW



Skip-gram

Video Thumbnail
DO NOT REMOVE

Continuous Bag of Words (CBOW)

The sliding window concept

Content Sequence:

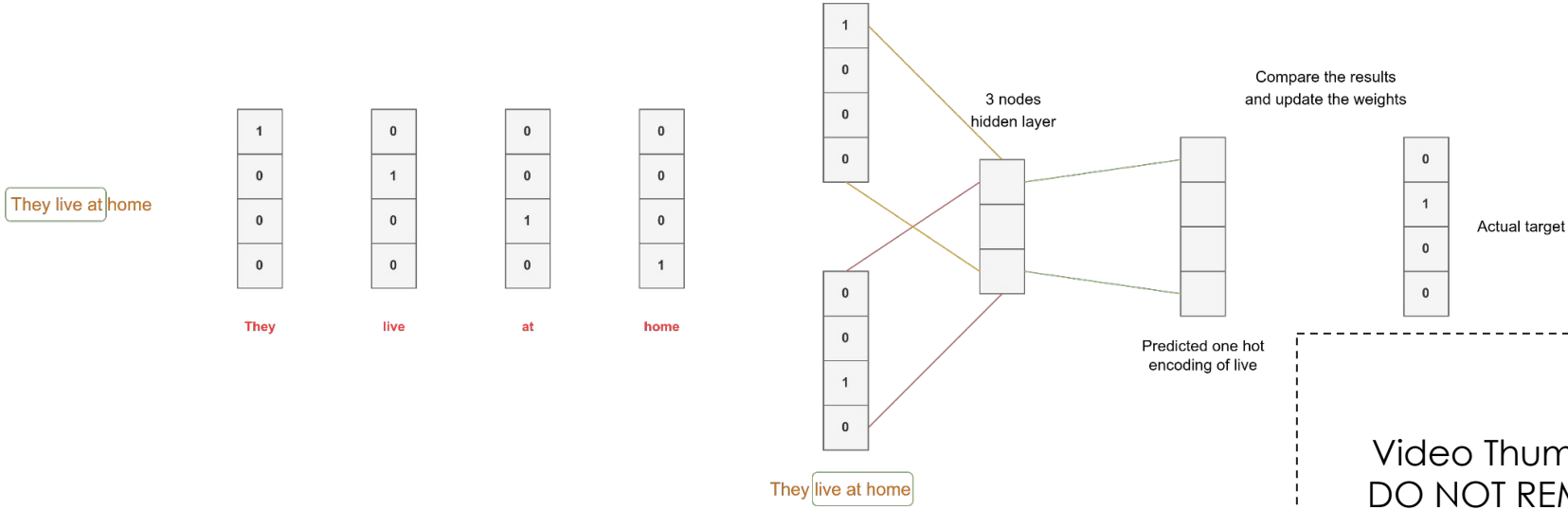
Window size = 1	(in, rises) (in, the)
Window size = 2	(rises, in) (Sun, in) (the, in) (east, in)

- We observe that as this window slides against the text, we can generate a dataset that we use to train a model.
- As you can see – the input is a list of words, and output is the nearest word.
- This style of data preparation fuels the **Continuous Bag of Words – CBOW** model in Word2vec

Video Thumbnail
DO NOT REMOVE

Continuous Bag of Words (CBOW)

- **CBOW** is trained to **predict a single word from a fixed window size of context words**.
- In CBOW, to predict the target word, the **sum of the background vectors is used**.
- The CBOW model **calculates the average of the context vectors**.



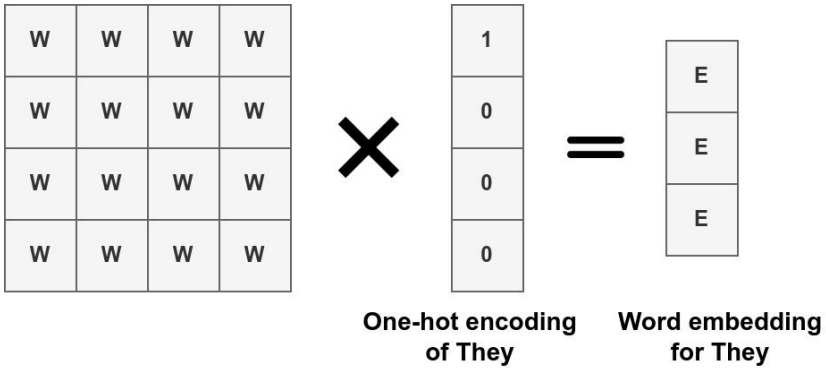
Video Thumbnail
DO NOT REMOVE

Continuous Bag of Words (CBOW)

- Each word is **first transformed into a one-hot encoding form**. Also, **only particular words that are in a window will be taken into consideration** rather than all the words in the sentence.

  For example, for a window size of 3, we only consider 3 words in a sentence. **The middle word is to be predicted, and the 2 words surrounding it provide as context for the Neural Network.** The window is then slid and the process is repeated again.

- Finally, after repeatedly **training the network by sliding the window**, we have weights, which we use to **compute the embeddings**, as shown below -



Video Thumbnail
DO NOT REMOVE

Skip-gram

- Now let's go back to our original statement. When we slide on the text, we can generate Context & Target pairs in which **the input is a word and the output is the context words**.

Words highlighted in Green are targets:	**(Context Seq, Target with window size 2)**
The Sun rises in the east	(The, ([Sun, rises, in, the])
The Sun rises in the east	(Sun, [the, rises, in, the])
The Sun rises in the east	(rises, [the, Sun, in, the])
The Sun rises in the east	(in, [Sun, rises, the, east])
The Sun rises in the east	(the, [rises, in, east])

Skip-gram

- As can be realized, the input is a word, and output will be a list of nearest words.
- This style of data preparation fuels the Skip-gram model in Word2vec. Skip-gram tries to predict the nearest words, given a word.

Words highlighted in Green are targets:

The	Sun	rises	in	the	east
The	Sun	rises	in	the	east
The	Sun	rises	in	the	east
The	Sun	rises	in	the	east
The	Sun	rises	in	the	east

(Context Seq, Target with window size 2)

(The, ([Sun, rises, in, the])

(Sun, [the, rises, in, the])

(rises, [the, Sun, in, the])

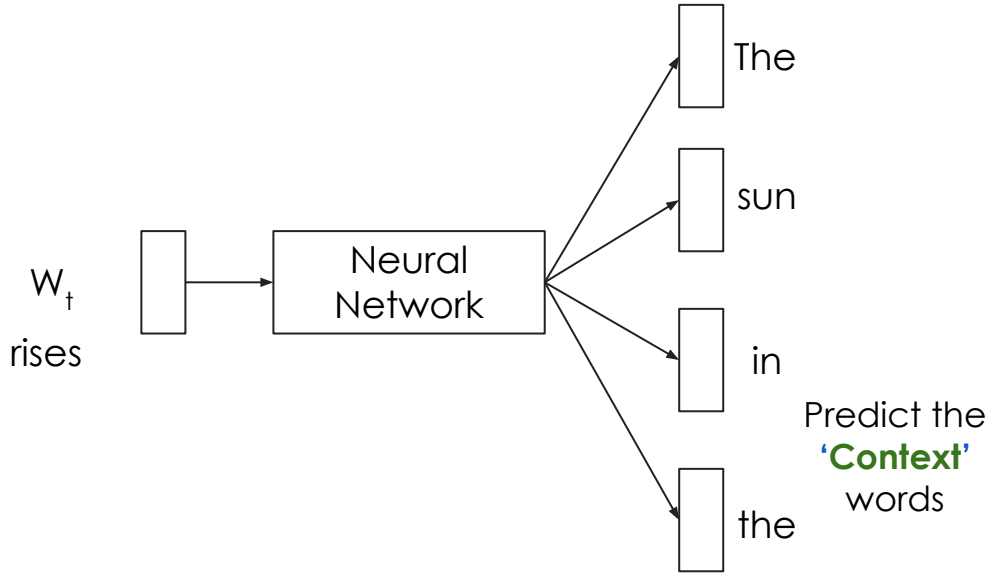
(in, [Sun, rises, the, east])

(the, [rises, in, east])

Skip-gram

- Using Skip-gram, The **target word is fed at the input**, the **hidden layer remains unchanged**, and the **output layer** of the neural network is **replicated multiple times to match the number of context words chosen**.

- Although the **skip-gram model's operation is relatively similar to that of the CBOW**, there are **differences in the design of its neural network** and the manner in which the **weight matrix is produced**.



Video Thumbnail
DO NOT REMOVE

Skip-gram

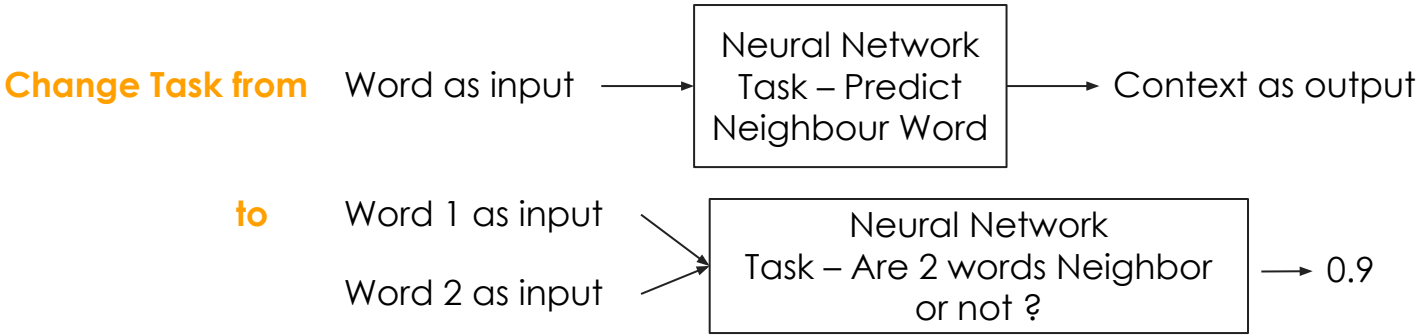
- After obtaining the weight matrix, the steps to get word embeddings is same as CBOW.
- If our vocabulary has a size of 10,000 words, the last output layer will be quite a big vector and we will have to predict many context words. **This is computationally very expensive.** We also know that generally, a word does not have more than a certain number of context words around it.
- Even **rare words or phrases can be accurately represented by skip-gram** when only a small portion of the training data is used.
- **CBOW** is found to train faster than Skip-gram, and **can better represent more frequent words**.

Video Thumbnail
DO NOT REMOVE

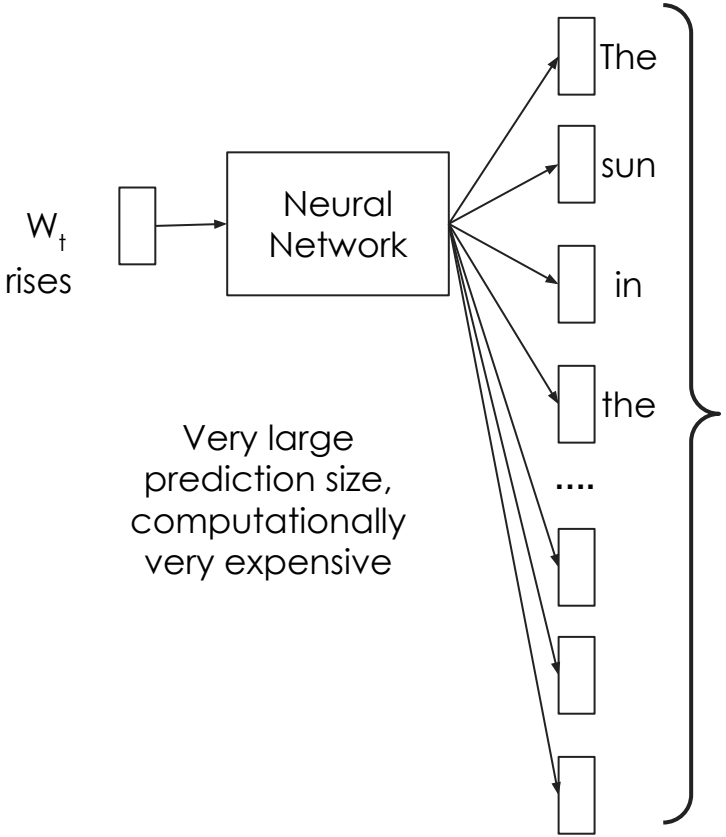
Skip-gram: Negative Sampling

- **Negative sampling** allows us to only modify a small percentage of the weights, rather than all of them for each training sample.

- We do this by slightly modifying our problem. Instead of trying to predict the probability of being a nearby word for all the words in the vocabulary, we try to predict the probability that our training sample words are neighbors or not.



Skip-gram: Negative Sampling



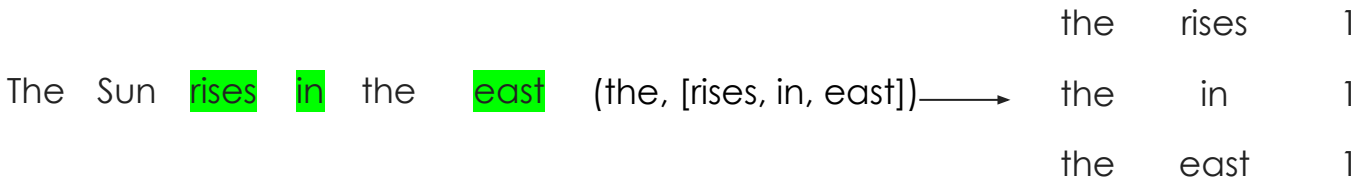
The Sun rises in the east **(the, [rises, in, east])**

input1	input2	Target
the	rises	1
the	in	1
the	east	1

Skip-gram: Negative Sampling

- However if all samples are generated like this, we will only end up having our targets as 1. That is not desirable, as the model will have no learning capabilities

- To address this, we need to introduce negative samples to our dataset – samples of words that are not neighbors. Our model needs to return 0 for those samples.



Skip-gram: Negative Sampling

- We randomly find words from vocabulary which are already not marked as 1. We end up having:

The Sun rises in the east (the, [rises, in, east]) ⟶

the	rises	1
the	in	1
the	east	1
the	wow	0
the	xx	0

- Now, we should be able to train the model faster with a wide understanding of what's near and what's not.

Skip-gram: Negative Sampling

- In terms of training, **Skip-gram with Negative Sampling (SGNS) is slower than CBOW.** However, it **performs effectively with a small amount of the training data** and accurately represents even uncommon words or phrases.
- Training involves **2 hyperparameters**: **window size** and the **number of negative samples.**
- **Smaller window sizes** (2-15) produce embeddings with **high similarity scores,** indicating that the **words are interchangeable.**
- **Larger window sizes** (15-50, or even more) result in embeddings where **similarity is more indicative of the words' relatedness.**

Word2vec: Limitations

- One of the biggest problems with Word2vec is the **inability to handle unknown or out-of-vocabulary (OOV) words**. If your model hasn't encountered a word before, it will have no idea how to interpret it or how to build a vector for it. You are then forced to use a random vector, which is far from ideal.

- **No shared representations at sub-word levels**. If Word2vec encounters a new word which ends with say 'less' – for example 'flawless', there will be no link between less and flawless – even though we as humans know that it's probably an adjective.

- **Word2vec represents every word as an independent vector**, even though many words are morphologically similar.

- **Scaling to new languages requires new embedding matrices** and does not allow for parameter sharing, meaning cross-lingual use of the same model isn't an option.

- **Word2Vec also requires a large corpus to train** – though at this point in time in the industry, large text volumes are abundantly available.

Summary

- We looked at why numerical representations are important for text data in Machine Learning, and revisited how to transform text into numerical data.
- We've also understood Word2vec's ways of doing this via the CBOW and Skip-gram methods, and taken a look at the limitations of the Word2vec approach.

Great Learning

Happy Learning !