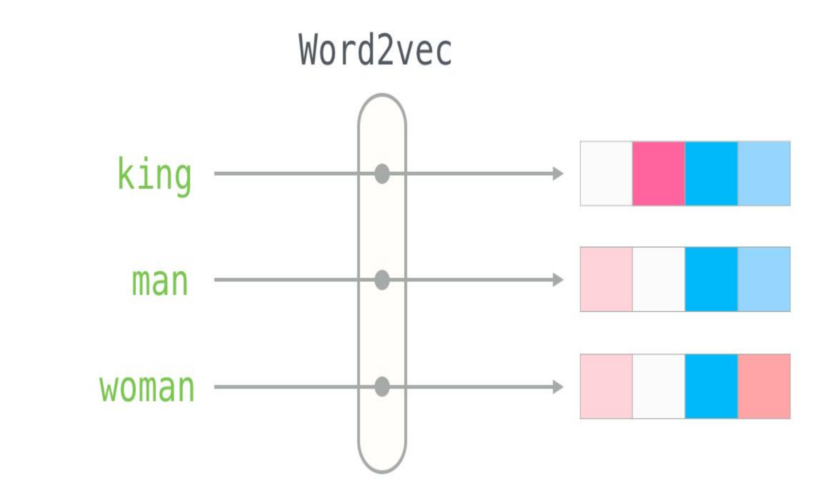
NATURAL LANGUAGE PROCESSING

LECTURE AGENDA

- **Word2Vec**
- **Text Classification**

4

WORD2VEC



WHAT'S WORD2VEC?

Technique/Algorithm for natural language processing published in 2013.

Represents each distinct **word** with a particular list of numbers called a **vector**.

The vectors are chosen carefully such that a simple mathematical function (the cosine similarity between the vectors) indicates the level of semantic similarity between the words represented by those vectors.

Uses a two-layer **neural network** model to **learn word associations** from a large corpus of text.

The input to the NN is the text corpus and the output is vector of features.

WHAT'S WORD2VEC?

Once trained, such a model can

1. **Detect synonymous words.**
2. **Suggest additional words for a partial sentence.**

It uses **word embeddings** in training the model.

Its main purpose is to **compute the weight of each word in the sentence which indicates its rank and thus can detect the missing word**.

It can detect the relatedness of words such as **man-boy= woman-girl.**

It can also detect the singulars and the plurals which helps in natural language analysis and modeling.

When trained in a large amount of data, it can detect synonyms of words.

WORD2VEC MODEL ARCHITECTURES

Two types of Word2Vec :

1. **CBOW "Continuous Bag Of Words":** the model predicts the current word from a window of surrounding context words.

Example: I went to the restaurant to order …….. . (**Food**)

I went to ……… to order food. (the **restaurant**)

2. **Skip-Grams:** the model uses the current word to predict the surrounding window of context words. The skip- gram architecture weighs nearby context words more heavily than more distant context words. It requires **text generation**.

Example: We ……………?

We **shall…………**

We shall **not ……….**

We shall not **accept ………**

We shall not accept **this deal.**

CBOW is faster while skip-gram does a better job for infrequent words.

WORD2VEC: CBOW

It is a hybrid technique using BOW and N-Gram techniques.

BOW is the representation of each word.

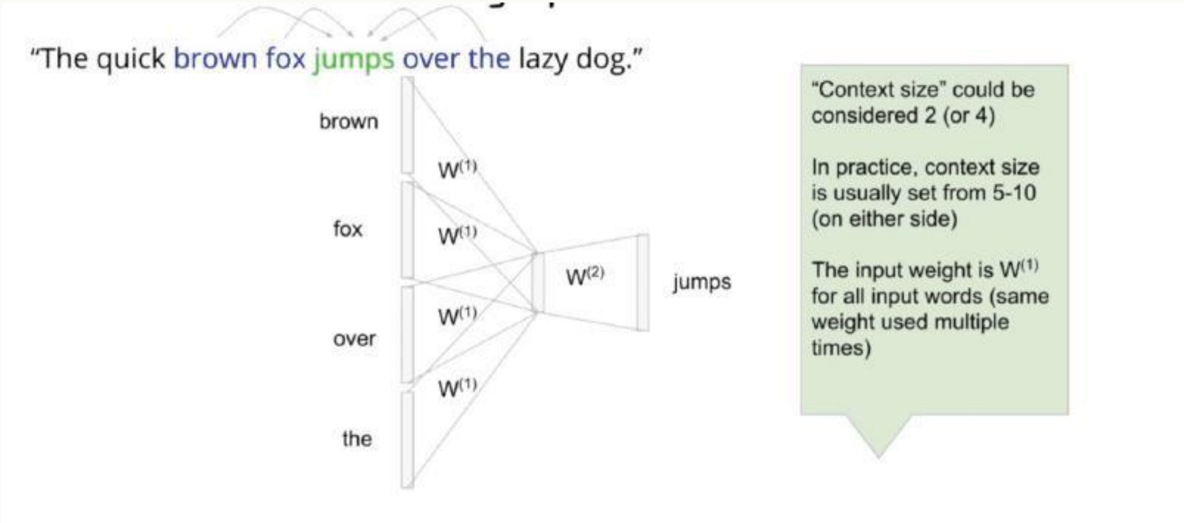
The **bag-of-words model** is a simplifying representation used in natural language processing and information retrieval (IR), where the text (such as a sentence or a document) is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity.

The bag-of-words model is commonly used in methods of **document classification** where the (frequency of) occurrence of each word is used as a feature for training a classifier.

The Bag-of-words model is one example of a Vector space model.

WORD2VEC: CBOW

- Continuous Bag of Words is a hybrid model using BOW and N-Grams.
- **BOW**: depends on the words found in the context and represent them as 0s or 1s or percentage of the word existence.
- **NGram:** a contiguous sequence of *n* items from a given sample of text or speech. The items can be phonemes, syllables, letters, words or base pairs according to the application. The *n*-grams typically are collected from a text or speech corpus. Using Latin numerical prefixes, an *n*-gram of size 1 is referred to as a "unigram"; size 2 is a "bigram" (or, less commonly, a "digram"); size 3 is a "trigram". English cardinal numbers are sometimes used, e.g., "four-gram", "five-gram", and so on.
- **CBOW**: using more than one existing word to detect a specific word (most commonly the last one in the sentence), via NN utilizing the embedding matrix for the input words.



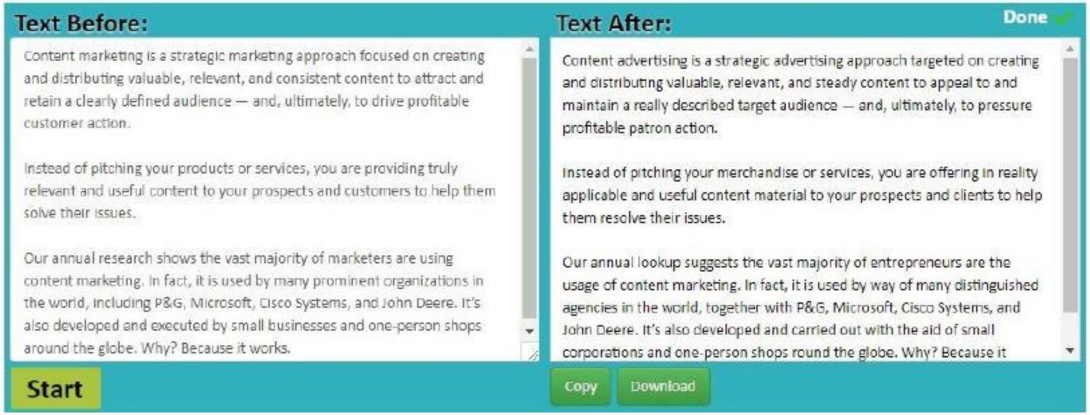
Context size could be 2 or more (normally between 5 and 10) in both ways.

WORD2VEC: CBOW

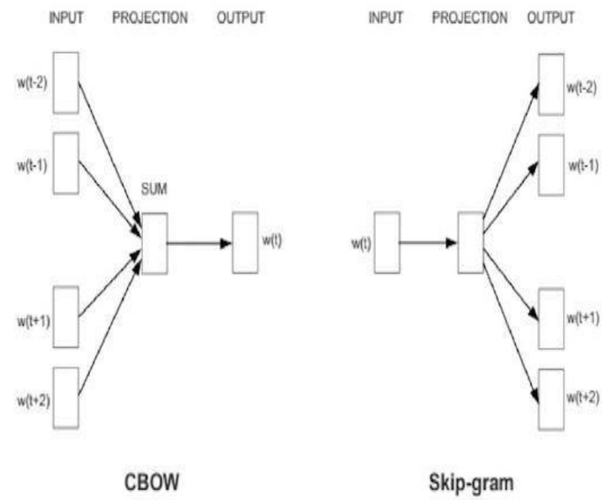
WORD2VEC: CBOW- APPLICATIONS

Used in **Article Spinning** which means to rephrase the article with new words/synonyms keeping the same meaning but without being considered as a plagiarized version.

This is performed by first removing some words from the original article (randomly or with a certain criteria to be automated, i.e. remove every 6th word…etc), and then let the CBOW model predict the missing words (if the same word was predicted, re-iterate to get different

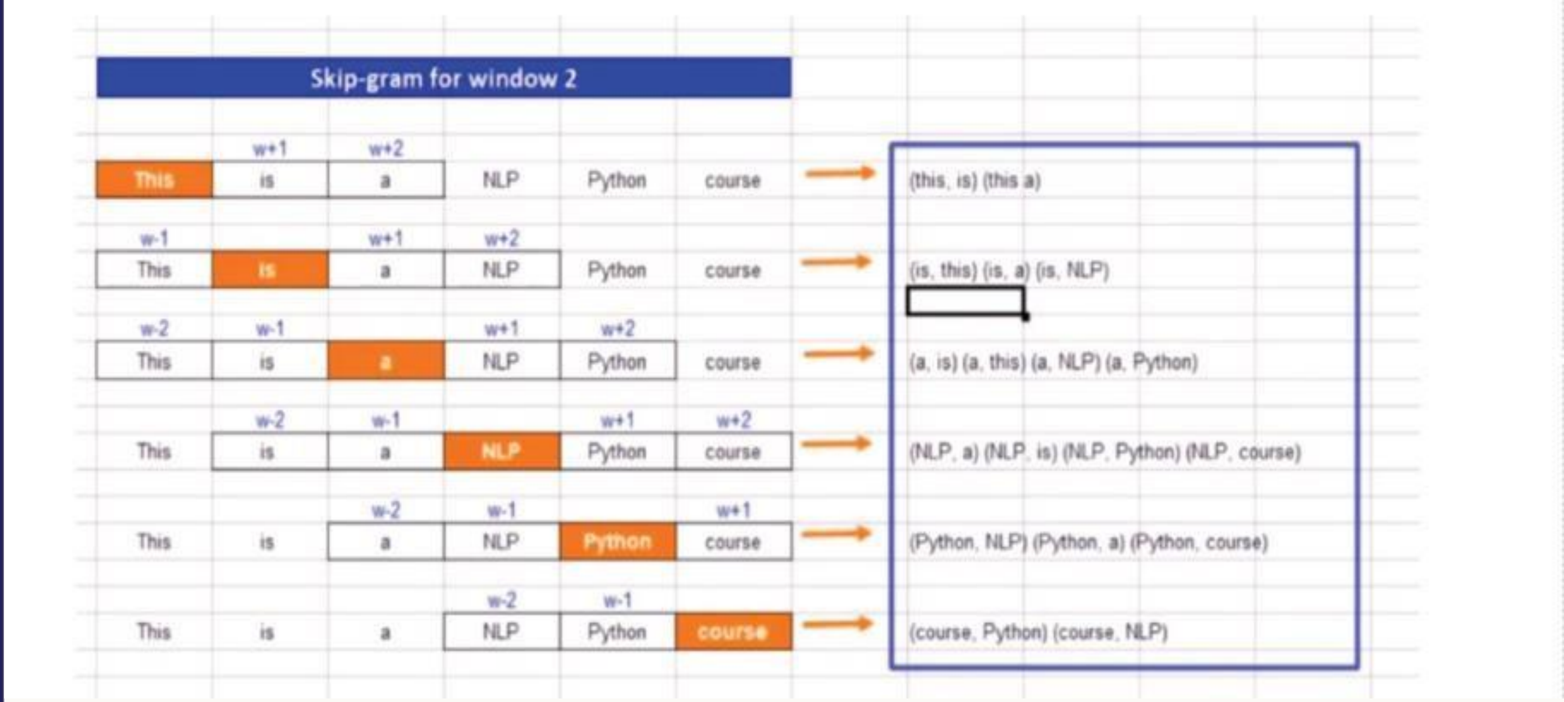


WORD2VEC: SKIP-GRAM



CBOW Vs.Skip-Gram

WORD2VEC: SKIP-GRAM TRAINING



Predicts a sentence given a word

WORD2VEC: SKIP-GRAM

It is to be noted that we should specify the maximum size of word relatedness.

- For example, if the maximum =4, then the window size is two-words surrounding the specified word.

Then train the NN to be able to predict the upcoming word if we just give it 1 word as input.

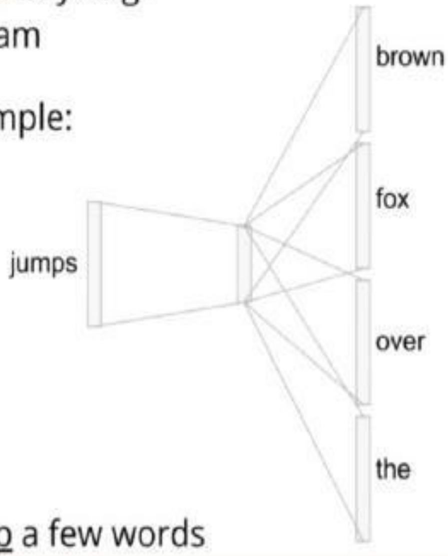
We input to the network the embedding matrix of each word and its one hot encoder. Then calculate the multiplication of the two matrices.

The embedding ng random values and thro built-in library like SpaCy.

$$[0\ 0\ 0\ 1\ 0] \times \begin{vmatrix} 0.5 & 0.8 & 1.3 \\ 2.1 & 1.2 & 0.2 \\ 0.4 & 0.7 & 1.1 \\ \mathbf{2.8} & \mathbf{1.4} & \mathbf{0.9} \\ 0.8 & 1.2 & 0.4 \end{vmatrix} = [\mathbf{2.8}\ \mathbf{1.4}\ \mathbf{0.9}]$$

WORD2VEC: SKIP-GRAM

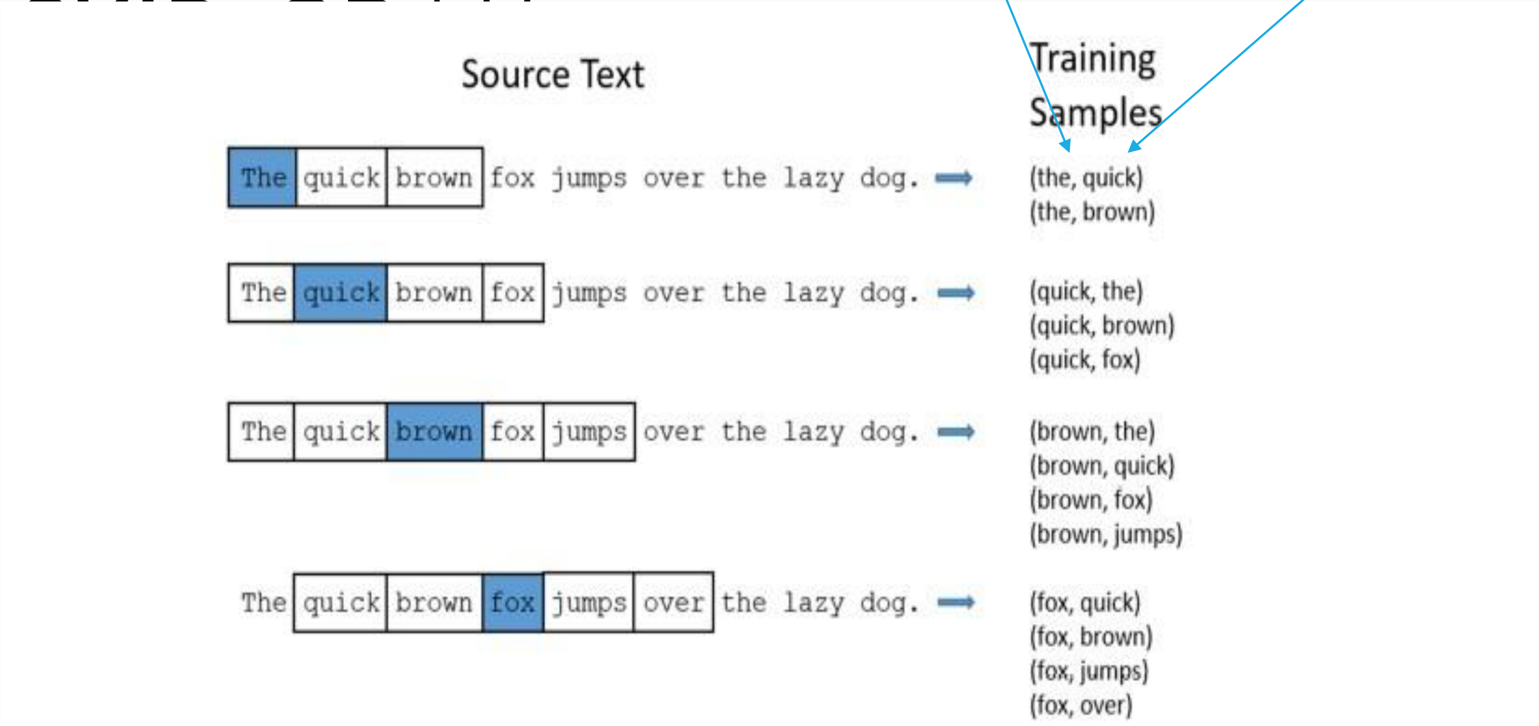
- "The quick brown fox jumps over the lazy dog."
- Helpful to think of it in terms of bigram
- Bigram model gives us 1 training sample:
  jumps → over
- Skipgram gives us 3 additional training samples:
  jumps → brown
  jumps → fox
  jumps → the
- Skipgram: *like* bigram, except we skip a few words



WORD2VEC:

featur
e

outpu
t



SOME PRACTICAL INSIGHTS

CBOW

We will build it from scratch, no libraries used.

Text domain (Generic, Economics….etc) and writing style are specified according to the domain of the trained input text.

Re.sub(…): remove everything but alphanumeric

Original Notebook

https://www.kaggle.com/alincijov/continuous-bag-of-words-cbow-numpy-for-beginners

Imports

In [1]:
```
import re
import numpy as np
import matplotlib.pyplot as plt
```

Data

In [2]:
```
sentences = """We are about to study the idea of a computational process.
Computational processes are abstract beings that inhabit computers.
As they evolve, processes manipulate other abstract things called data.
The evolution of a process is directed by a pattern of rules
called a program. People create programs to direct processes. In effect,
we conjure the spirits of the computer with our spells."""
```

Clean Data

In [4]:
```
# remove special characters
sentences = re.sub('[^A-Za-z0-9]+', ' ', sentences)
sentences
```

Out[4]:
```
'We are about to study the idea of a computational process Computational processes are abstract beings that inhabit computers A
s they evolve processes manipulate other abstract things called data The evolution of a process is directed by a pattern of rul
es called a program People create programs to direct processes In effect we conjure the spirits of the computer with our spells
'
```

CBOW

```
Out[4]: 'We are about to study the idea of a computational process Computational processes are abstract beings that inhabit computers A
        s they evolve processes manipulate other abstract things called data The evolution of a process is directed by a pattern of rul
        es called a program People create programs to direct processes In effect we conjure the spirits of the computer with our spells
        '
```

```
In [5]: # remove 1 letter words
        sentences = re.sub(r'(?:^| )\w(?:$| )', ' ', sentences).strip()
        sentences
```

```
Out[5]: 'We are about to study the idea of computational process computational processes are abstract beings that inhabit computers As
        they evolve processes manipulate other abstract things called data The evolution of process is directed by pattern of rules cal
        led program People create programs to direct processes In effect we conjure the spirits of the computer with our spells'
```

```
In [7]: # lower all characters
        sentences = sentences.lower()
        sentences
```

```
Out[7]: 'we are about to study the idea of computational process computational processes are abstract beings that inhabit computers as
        they evolve processes manipulate other abstract things called data the evolution of process is directed by pattern of rules cal
        led program people create programs to direct processes in effect we conjure the spirits of the computer with our spells'
```

Vocabulary

```
In [8]: words = sentences.split()
        words
```

```
Out[8]: ['we',
         'are',
         'about',
         'to',
         'study',
         'the',
         'idea',
         'of',
```

```
In [10]: vocab_size = len(vocab)
         embed_dim = 10
         context_size = 2
```

Dictionaries for vocab

```
In [11]: word_to_ix = {word: i for i, word in enumerate(vocab)}
         word_to_ix

Out[11]: {'beings': 0,
          'they': 1,
          'the': 2,
          'data': 3,
          'that': 4,
          'by': 5,
          'spells': 6,
          'as': 7,
          'inhabit': 8,
          'pattern': 9,
          'study': 10,
          'process': 11,
          'manipulate': 12,
```

CBOW

Context size: we take two- word window (2 before the word and 2 after it)
Embed dim: embedding matrix size

Data bags

```python
# data - [(context), target]


#  """We are about to study the idea of a computational process.
# Computational processes are abstract beings that inhabit computers.
# As they evolve, processes manipulate other abstract things called data.
# The evolution of a process is directed by a pattern of rules
# called a program. People create programs to direct processes. In effect,
# we conjure the spirits of the computer with our spells."""


data = []
for i in range(2, len(words) - 2):
    context = [words[i - 2], words[i - 1], words[i + 1], words[i + 2]]
    target = words[i]
    data.append((context, target))
print(data[:5])
```

```
[(['we', 'are', 'to', 'study'], 'about'), (['are', 'about', 'study', 'the'], 'to'), (['about', 'to', 'the', 'idea'], 'study'),
(['to', 'study', 'idea', 'of'], 'the'), (['study', 'the', 'of', 'computational'], 'idea')]
```

CBOW

- **Context= feature**
- **Context size=2**
- **1st Word Bag:** starting by the word “**about**” and taking the surrounding 2 words prior
  and next to it as its features (we are , to study)
- 2nd Word Bag: the word “**to**” and taking the surrounding 2 words prior and next to it as its features (are about , study the)
- …..and so on.

CBOW

Embeddings

```
In [30]: embeddings =  np.random.random_sample((vocab_size, embed_dim))
         print(embeddings.shape)
         embeddings

(43, 10)

Out[30]: array([[0.08939882, 0.3892089 , 0.4567939 , 0.53094761, 0.74348623,
                  0.34150442, 0.55660483, 0.02072189, 0.13327499, 0.45594249],
                 [0.40784607, 0.07627876, 0.18211992, 0.26745623, 0.76726664,
                  0.37154191, 0.24283864, 0.45072007, 0.97014264, 0.52154393],
                 [0.76033567, 0.84936684, 0.48639942, 0.53791185, 0.75022506,
                  0.20517451, 0.24352414, 0.57309764, 0.17133082, 0.05075539],
                 [0.62764997, 0.41710412, 0.25153431, 0.03136028, 0.75527662,
                  0.22397494, 0.81831366, 0.30164402, 0.10766157, 0.97602104],
                 [0.82987746, 0.29197956, 0.19200094, 0.91949316, 0.41277894,
                  0.81531249, 0.84972957, 0.56546699, 0.60244343, 0.2987964 ],
                 [0.43238712, 0.66300723, 0.27339831, 0.67379816, 0.89164669,
                  0.8395173 , 0.64306478, 0.57261948, 0.84380829, 0.52783897],
                 [0.57172159, 0.22998524, 0.79892047, 0.72848208, 0.91182284,
                  0.0038121 , 0.95686758, 0.30823109, 0.03101914, 0.01781339],
                 [0.46979495, 0.66534856, 0.30779333, 0.66321806, 0.74266468,
                  0.82716778, 0.25796039, 0.81673259, 0.80101095, 0.10745899],
                 [0.09521155, 0.37634719, 0.92529312, 0.62081955, 0.28410062,
                  0.77916037, 0.32243643, 0.07827828, 0.85349066, 0.56071045],
                 [0.49451585, 0.94768413, 0.09402939, 0.4012761 , 0.70149502,
                  0.58324133, 0.71914628, 0.27482303, 0.00647916, 0.13874169],
                 [0.09834639, 0.14981483, 0.72454517, 0.75044612, 0.78671315,
                  0.35935154, 0.18212539, 0.26895909, 0.45237431, 0.19156351],
                 [0.44024797, 0.89947275, 0.50075014, 0.09110534, 0.04977168,
                  0.76241979, 0.03342138, 0.30986895, 0.64687912, 0.96879223],
                 [0.95271849, 0.35837667, 0.65011583, 0.57853376, 0.87372702,
                  0.45309184, 0.85684804, 0.87045669, 0.07431674, 0.3991994 ],
                 [0.23262485, 0.31681525, 0.85842673, 0.4381845 , 0.63433476,
```

Linear Model

```
def linear(m, theta):
    w = theta
    return m.dot(w)
```

Log softmax + NLLloss = Cross Entropy

```
def log_softmax(x):
    e_x = np.exp(x - np.max(x))
    return np.log(e_x / e_x.sum())
```

```
def NLLLoss(logs, targets):
    out = logs[range(len(targets)), targets]
    return -out.sum()/len(out)
```

```
def log_softmax_crossentropy_with_logits(logits,target):

    out = np.zeros_like(logits)
    out[np.arange(len(logits)),target] = 1

    softmax = np.exp(logits) / np.exp(logits).sum(axis=-1,keepdims=True)

    return (- out + softmax) / logits.shape[0]
```

CBOW

CBOW

Forward function

```python
def forward(context_idxs, theta):
    m = embeddings[context_idxs].reshape(1, -1)
    n = linear(m, theta)
    o = log_softmax(n)
    return m, n, o
```

Backward function

```python
def backward(preds, theta, target_idxs):
    m, n, o = preds
    dlog = log_softmax_crossentropy_with_logits(n, target_idxs)
    dw = m.T.dot(dlog)
    return dw
```

Optimize function

```python
def optimize(theta, grad, lr=0.03):
    theta -= grad * lr
    return theta
```

Training function

```
theta = np.random.uniform(-1, 1, (2 * context_size * embed_dim, vocab_size))
```

```
epoch_losses = {}

for epoch in range(80):

    losses =  []

    for context, target in data:
        context_idxs = np.array([word_to_ix[w] for w in context])
        preds = forward(context_idxs, theta)

        target_idxs = np.array([word_to_ix[target]])
        loss = NLLLoss(preds[-1], target_idxs)

        losses.append(loss)

        grad = backward(preds, theta, target_idxs)
        theta = optimize(theta, grad, lr=0.03)


    epoch_losses[epoch] = losses
```

CBOW

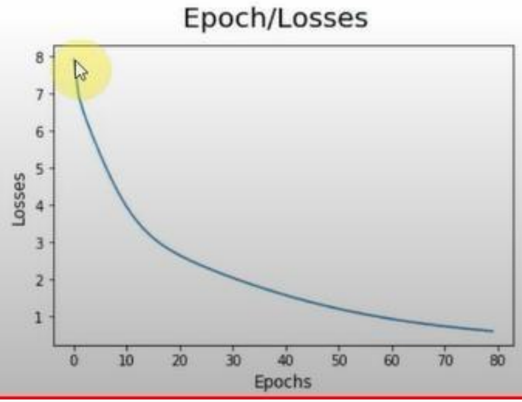
CBOW

Plot loss/epoch

```
ix = np.arange(0,80)

fig = plt.figure()
fig.suptitle('Epoch/Losses', fontsize=20)
plt.plot(ix,[epoch_losses[i][0] for i in ix])
plt.xlabel('Epochs', fontsize=12)
plt.ylabel('Losses', fontsize=12)
```

```
Text(0, 0.5, 'Losses')
```



Predict function

```
In [26]: def predict(words):
             context_idxs = np.array([word_to_ix[w] for w in words])
             preds = forward(context_idxs, theta)
             word = ix_to_word[np.argmax(preds[-1])]

             return word
```

```
In [27]: # (['we', 'are', 'to', 'study'], 'about')
         predict(['we', 'are', 'to', 'study'])

Out[27]: 'about'
```

CBOW