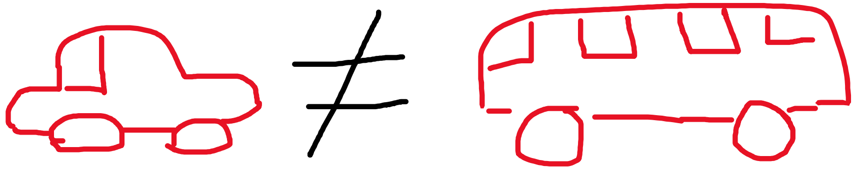
Advanced Features

Ch.4 Text Analytics with Python. Dipanjan Sarkar. 2019. Apress
An implementation guide to word2vec. Derek Chia. 2018
R Machine Learning Projects. Sunil Chinnamgari. 2018. Packt

Different Approaches

Bag of Words

Words are atomic units



- One-hot encoding for each word:

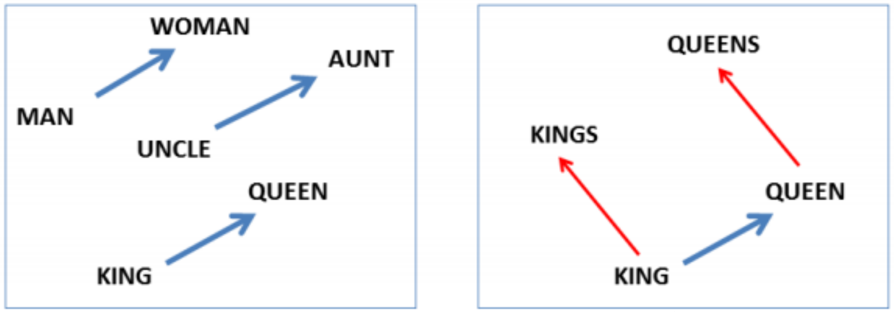
car = [1,0]   bus = [0,1]

$\begin{bmatrix}1\\0\end{bmatrix}$ $\begin{bmatrix}0\\1\end{bmatrix}$

- N Dimensions = N words
- No projection to other dimensions
- Unable to capture Context or Meaning:
  car:bus [transportation]
- sparse
- Frequency-Based method

Word Vectors

Multidimensional continuous word space



(Mikolov et al. 2013)

Distributed representations: dependence of one word on the other words

Word2Vec Model (Tomas Mikolov et al., 2013, Google): distributed, and continuous dense vector representations of words

Distributional hypothesis: the context for each word is in its nearby words

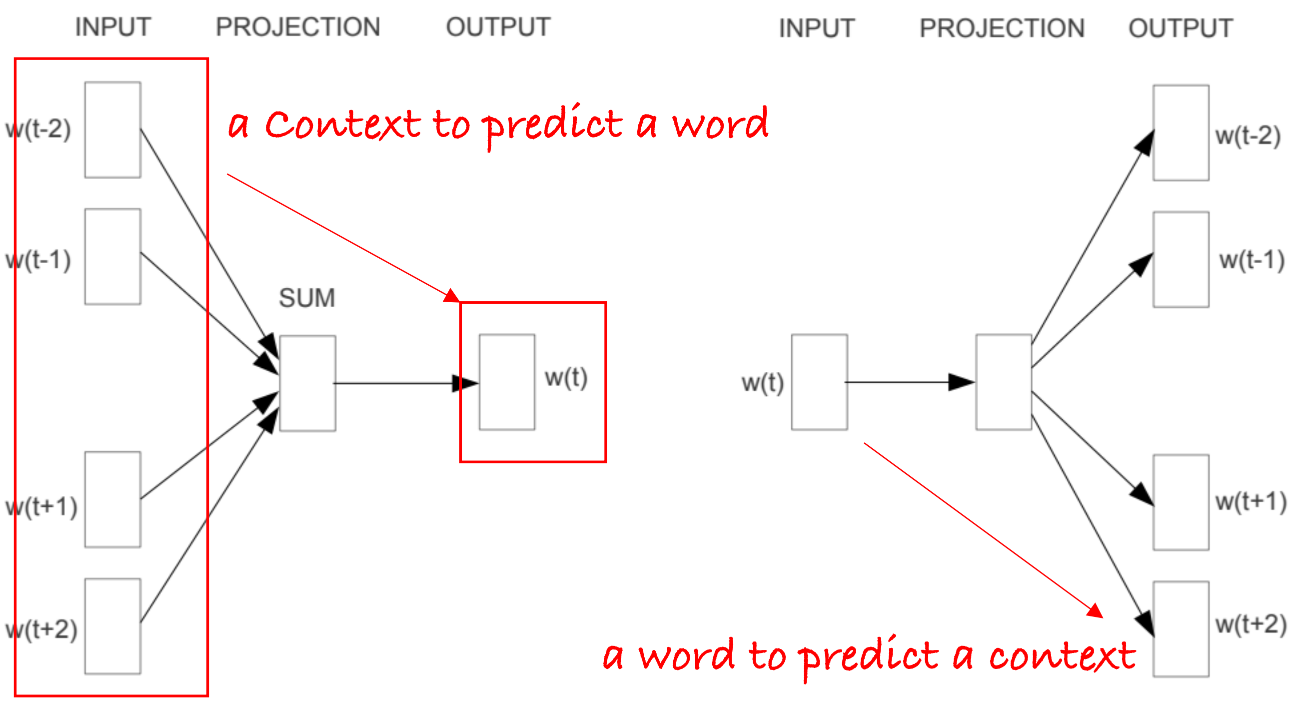
Two Models

Goal: Generate vector representations of words that carry semantic meanings for further NLP tasks.

The transformation from words to vectors is also known as word embedding.

Learns the embedding by predicting the current word based on its context

is faster and has slightly better accuracy for more frequent words.



CBOW
Continuous Bag-of-Words

Skip-gram

Learns the embedding by by predicting the surrounding words given a current word

works well with small amount of data and represents well rare words or phrases

(Sunil Chinnamgari, 2019)

(context_window, target_word)

CBOW

w(t) – the current target word (center word)
w(t-1), w(t+1) – the source content words (surrounding words)

Example

natural language processing and machine learning is fun

w(t) target _ word

w(t-1) w(t+1) context _ window

([natural, processing], language)

#											#
#1	natural	language	processing	and	machine	learning	is	fun	and	exciting	#1
	Xκ	Y(c=1)	Y(c=2)								
#2	natural	language	processing	and	machine	learning	is	fun	and	exciting	#2
	Y(c=1)	Xκ	Y(c=2)	Y(c=3)							
#3	natural	language	processing	and	machine	learning	is	fun	and	exciting	#3
	Y(c=1)	Y(c=2)	Xκ	Y(c=3)	Y(c=4)						
#4	natural	language	processing	and	machine	learning	is	fun	and	exciting	#4
		Y(c=1)	Y(c=2)	Xκ	Y(c=3)	Y(c=4)					
#5	natural	language	processing	and	machine	learning	is	fun	and	exciting	#5
			Y(c=1)	Y(c=2)	Xκ	Y(c=3)	Y(c=4)				

#	Token	#1			#2				#3				
0	natural	1	0	0	0	1	0	0	0	1	0	0	0
1	language	0	1	0	1	0	0	0	0	0	1	0	0
2	processing	0	0	1	0	0	1	0	1	0	0	0	0
3	and	0	0	0	0	0	0	1	0	0	0	1	0
4	machine	0	0	0	0	0	0	0	0	0	0	0	1
5	learning	0	0	0	0	0	0	0	0	0	0	0	0
6	is	0	0	0	0	0	0	0	0	0	0	0	0
7	fun	0	0	0	0	0	0	0	0	0	0	0	0
8	exciting	0	0	0	0	0	0	0	0	0	0	0	0
		Xκ	Y(c=1)	Y(c=2)	Xκ	Y(c=1)	Y(c=2)	Y(c=3)	Xκ	Y(c=1)	Y(c=2)	Y(c=3)	Y(c=4)

Derek Chia, 2018. An implementation guide to Word2Vec

- Word2Vec family of models is unsupervised
- No additional labels or information needed

Implementing CBOW Model

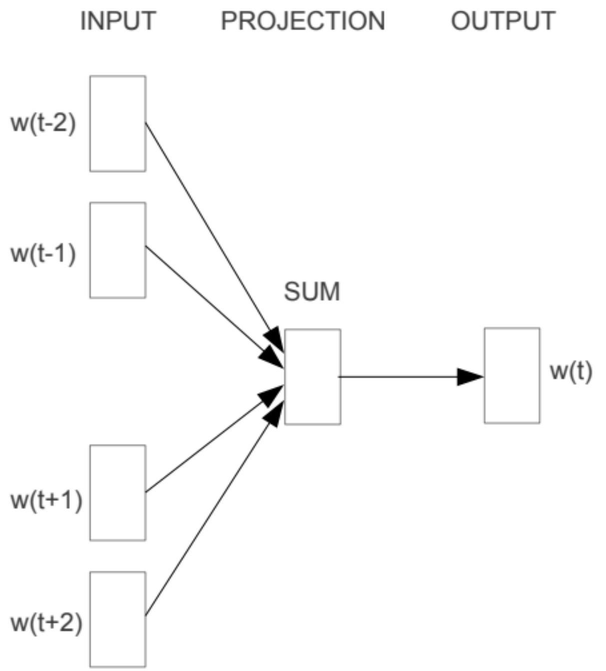
```
from keras.preprocessing import text
from keras.utils import np_utils
from keras.preprocessing import sequence
```

Keras

- Deep learning framework for Python
- Enables fast experimentation
- Runs on top of other frameworks
- Written by François Chollet (2015)

Workflow

- Preprocess corpus
- Build the corpus vocabulary
- Build a CBOW (context, target) generator
- Build the CBOW model architecture
- Train the model
- Get word embeddings



Data Preparation

Step 1. Preprocessing Corpus

alice_norm = norm(alice)

```
['another moment went alice never considering world get',
 'rabbit hole went straight like tunnel way dipped suddenly suddenly alice m
 'either well deep fell slowly plenty time went look wonder going happen nex
```

Step 2. Corpus Vocabulary

fit_on_texts: word -> index dictionary

word_index: dictionary mapping words (str) to their rank/index (int)

```
tokenizer = text.Tokenizer()
tokenizer.fit_on_texts(alice_norm)
word2id = tokenizer.word_index
```

```
from nltk.tokenize import sent_tokenize
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
from nltk.corpus import gutenberg
alice = gutenberg.raw(fileids='carroll-alice.txt')
```

```
def norm(text):
   norm_text = []
   tokenizer = RegexpTokenizer('[a-zA-Z]+')
   tokens_sentences = [tokenizer.tokenize(t) for t in
sent_tokenize(text)]
   stop_words = stopwords.words('english')
   for s in tokens_sentences:
      w_norm = []
      for w in s:
         if not w.lower() in stop_words:
            w_norm.append(w.lower())
      norm_text.append(' '.join(w_norm))
   return(norm_text)
```

```
from keras.preprocessing import text
```

(Dipanjar Sarkar, 2019, Ch4)

Vocabulary

Step 2. Corpus Vocabulary (cont)

PAD: Padding to a fixed length (e.g. sentence)

```
word2id['PAD'] = 0
id2word = {v:k for k, v in word2id.items()}
wids = [[word2id[w] for w in
text.text_to_word_sequence(doc)] for doc in
alice_norm]
```

- a vocabulary of unique words in the corpus
- a map of a word to its unique identifier and vice versa

```
vocab_size = len(word2id)
```

word2id

```
[('said', 1),
 ('alice', 2),
 ('little', 3),
 ('one', 4),
 ('know', 5),
 ('like', 6),
 ('would', 7),
 ('went', 8),
 ('could', 9),
 ('queen', 10)]
```

id2word

```
[(1, 'said'),
 (2, 'alice'),
 (3, 'little'),
 (4, 'one'),
 (5, 'know'),
 (6, 'like'),
 (7, 'would'),
 (8, 'went'),
 (9, 'could'),
 (10, 'queen')]
```

(Dipanjar Sarkar, 2019, Ch4)

Vocabulary

vocab_size = len(word2id)
window_size = 2

Step 2. Corpus Vocabulary (cont)

We need to generate Context_window, target_word pairs

```
def generate_context_word_pairs(corpus, window_size, vocab_size):
    context_length = window_size*2
    for words in corpus:
        sentence_length = len(words)
        for index, word in enumerate(words):
            context_words = []
            label_word   = []
            start = index - window_size
            end = index + window_size + 1
            context_words.append([words[i]
                        for i in range(start, end)
                        if 0 <= i < sentence_length
                        and i != index])
            label_word.append(word)
            x = sequence.pad_sequences(context_words, maxlen=context_length)
            y = np_utils.to_categorical(label_word, vocab_size)
            yield (x, y)
```

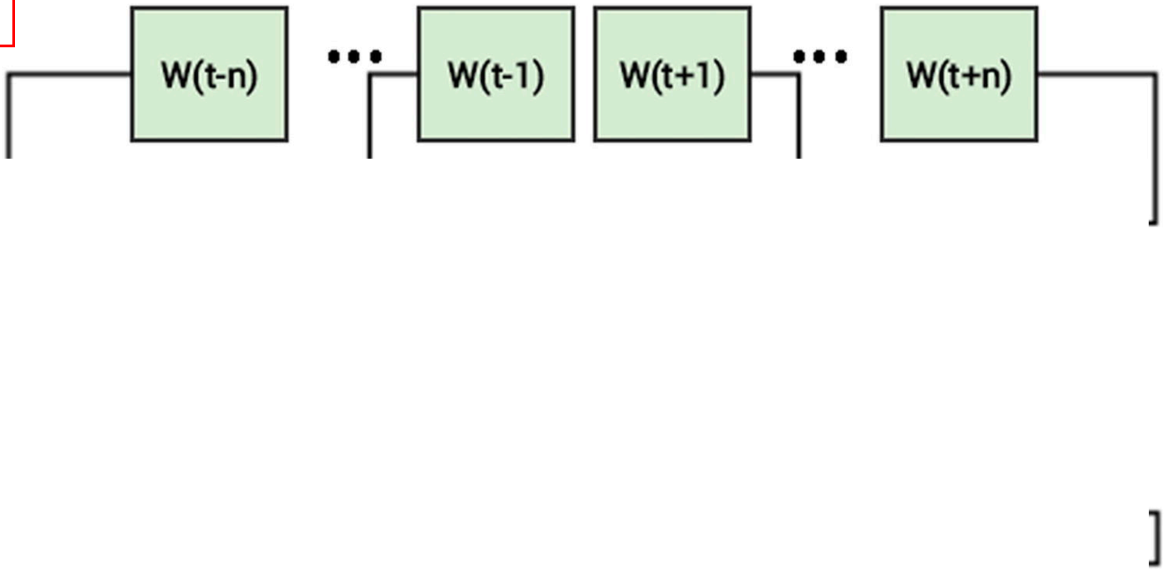
```
Context (X): ['alice', 'adventures', 'lewis', 'carroll'] -> Target (Y): wonderland
Context (X): ['adventures', 'wonderland', 'carroll', 'chapter'] -> Target (Y): lewis
Context (X): ['rabbit', 'hole', 'beginning', 'get'] -> Target (Y): alice
Context (X): ['hole', 'alice', 'get', 'tired'] -> Target (Y): beginning
Context (X): ['alice', 'beginning', 'tired', 'sitting'] -> Target (Y): get
Context (X): ['beginning', 'get', 'sitting', 'sister'] -> Target (Y): tired
Context (X): ['get', 'tired', 'sister', 'bank'] -> Target (Y): sitting
Context (X): ['tired', 'sitting', 'bank', 'nothing'] -> Target (Y): sister
Context (X): ['sitting', 'sister', 'nothing', 'twice'] -> Target (Y): bank
Context (X): ['sister', 'bank', 'twice', 'peeped'] -> Target (Y): nothing
Context (X): ['bank', 'nothing', 'peeped', 'book'] -> Target (Y): twice
```

(Dipanjar Sarkar, 2019, Ch4)

CBOW Model

vocab_size = len(word2id)
embed_size = 100
window_size = 2

Step 3. CBOW



cbow = Sequential()

Embedding	input:	(None, 4)
	output:	(None, 4, 100)

cbow.add(Embedding(input_dim=**vocab_size**,
output_dim=**embed_size**,
input_length=**window_size***2))

Lambda	input:	(None, 4, 100)
	output:	(None, 100)

cbow.add(Lambda(lambda x: K.**mean**(x, axis=1), output_shape=

Dense	input:	(None, 100)
	output:	(None, 2424)

cbow.add(Dense(vocab_size, activation="softmax"))

cbow.compile(loss='categorical_crossentropy', optimizer=

(Dipanjar Sarkar, 2019, Ch4)

Step 4. Train Model

Epoch: One iteration over the entire dataset

Batch: 1 Epoch is divided into several smaller batches

Loss: Cost or error function is used to evaluate a candidate solution (i.e. a set of weights)

Softmax – calculates the output probabilities

2000 examples
500 batches
4 iterations
=
1 epoch

(Ian Goodfellow et al., 2016. Deep Learning)

```
for epoch in range(1, 3): # 2 epoch for demo. Use more epoch
    loss = 0.
    i = 0
    for x, y in generate_context_word_pairs(corpus=wids, window_size=window_size, vocab_size=vocab_size):
        i += 1
        loss += cbow.train_on_batch(x, y)
        if i % 100000 == 0:
            print('Processed {} (context, word) pairs'.format(i))
    print('Epoch:', epoch, '\tLoss:', loss)
    print()
```

```
Epoch: 1        Loss: 87117.61428439617

Epoch: 2        Loss: 91431.85377651453
```

After training, similar words should have similar weights based on the embedding layer (Dipanjar Sarkar, 2019, Ch4)

Step 5. Get Word Embeddings

cbow.get_weights()[0]

Weights for the first layer (N dimensions * vocabulary size)

	0	1	2	3
alice	-0.219162	0.135351	0.264799	-0.057425
little	0.140651	-0.026189	-0.121107	0.045276
one	0.105631	-0.024040	0.141738	0.112736
know	0.116042	-0.027436	-0.132211	0.145596
like	-0.056407	0.123225	0.046632	0.059488

```
from sklearn.metrics.pairwise import euclidean_distances
distance_matrix = euclidean_distances(weights)
```

Let's find similar terms for Alice, Queen, and rabbit

```
similar_words = {search_term: [id2word[idx] for idx in distance_matrix[word2id[search_term]-1].argsort()[1:6]+1]
                 for search_term in ['alice', 'queen', 'rabbit']}
```

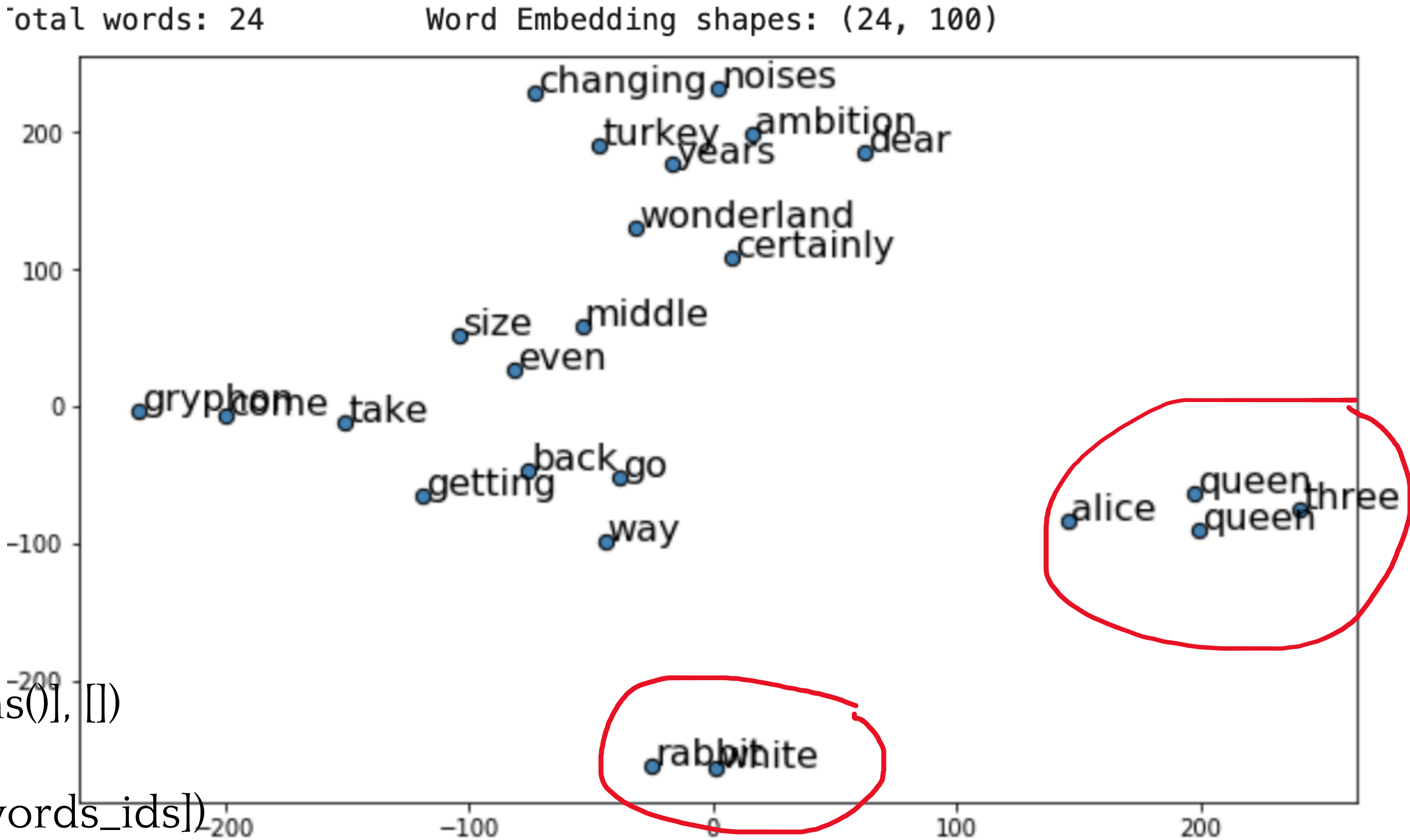
```
similar_words
```

```
{'alice': ['way', 'queen', 'getting', 'back', 'take'],
 'queen': ['gryphon', 'go', 'white', 'dear', 'come'],
 'rabbit': ['even', 'three', 'size', 'middle', 'certainly']}
```

(Dipanjar Sarkar, 2019, Ch4)

Visualizing Distance

from sklearn.manifold import TSNE
import pylab as plt




```
words = sum([[k] + v for k, v in similar_words.items()], [])
words_ids = [word2id[w] for w in words]
word_vectors = np.array([weights[idx] for idx in words_ids])
print('Total words:', len(words), '\tWord Embedding shapes:', word_vectors.shape)
tsne = TSNE(n_components=2, random_state=0, n_iter=10000, perplexity=3)
np.set_printoptions(suppress=True)
T = tsne.fit_transform(word_vectors)
labels = words
plt.figure(figsize=(10, 6))
plt.scatter(T[:, 0], T[:, 1], c="steelblue", edgecolors="k", s= 40)
for label, x, y in zip(labels, T[:, 0], T[:, 1]):
   plt.annotate(label, xy=(x+1, y+1), xytext=(0, 0), textcoords='offset points',  fontsize = 16)
```

(Dipanjar Sarkar, 2019, Ch4)