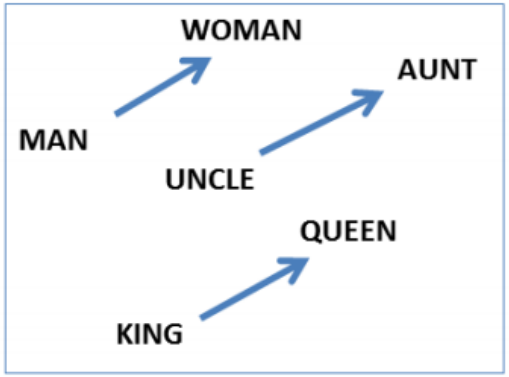
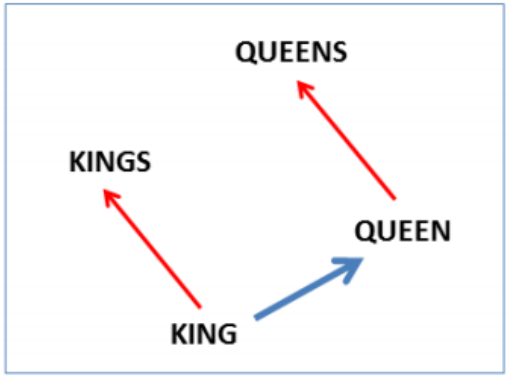
Part 1: Word2Vec

Word Embeddings as a Vector

- “Apple” and “tree”: just unicodes to a computer
- Word embedding: mapping a word to numerical values
- Similarity measure between words, inference using addition or subtraction





Word Embedding Methods

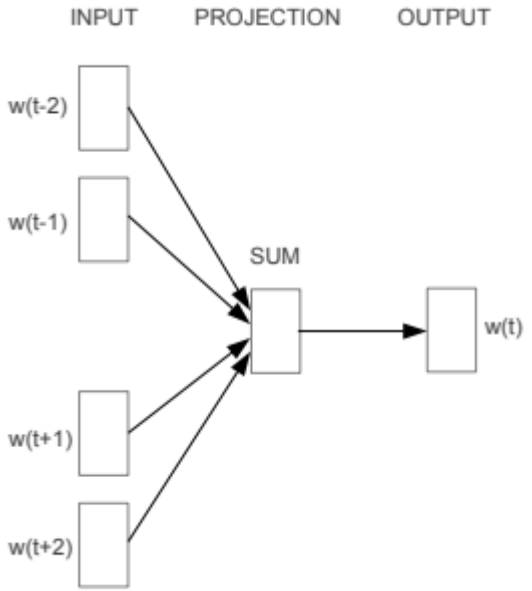
- NNLM, RNNLM
- Word2Vec, GloVE
- ELMo, BERT

Word2Vec

- CBOW (Continuous Bag-of-Words)
- Skip-gram

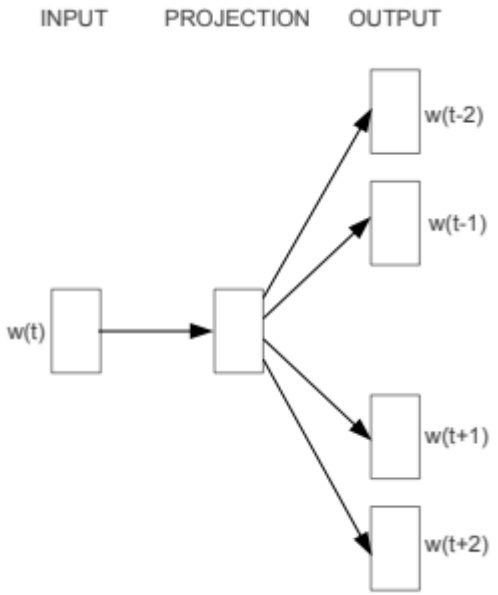
Word2Vec: CBOW

- Today, I went to the _____ and bought some milk and eggs.
- Store, shop, market, ...



Word2Vec: Skip-gram

- The opposite of CBOW
- ____ ____ ____ ____ store ____ ____ ____ ____



Word2Vec (code)

- python3
- numpy
- matplotlib
- gensim
- sklearn

Word2Vec (code)

```
# download the corpus
# text8: English Wikipedia data (10^8 bytes)
corpus = api.load('text8')
```

```
# train a model from the corpus
# default: CBOW, sg=1: skipgram
model = Word2Vec(corpus)
#model = Word2Vec(corpus, sg=1)
```

Word2Vec (code)

```
# print the embedding of 'car'
model.wv['car']
```

```
# print the 10 most similar words of 'car' based on cosine similarity
output = model.wv.most_similar('car')
output = np.array(output)

print(output)
```

```
[['driver' '0.7981117963790894']
 ['taxi' '0.7222708463668823']
 ['cars' '0.7116928696632385']
 ['motorcycle' '0.7056505680084229']
 ['truck' '0.6872323751449585']
```

Word2Vec (code)

```
# print the similairty between two words
print('car', '- driver:', model.wv.similarity('car', 'driver'))
print('car', '- water:', model.wv.similarity('car', 'water'))
print('fish', '- water:', model.wv.similarity('fish', 'water'))
```

```
car - driver: 0.7981118
car - water: 0.106778875
fish - water: 0.5285856
```

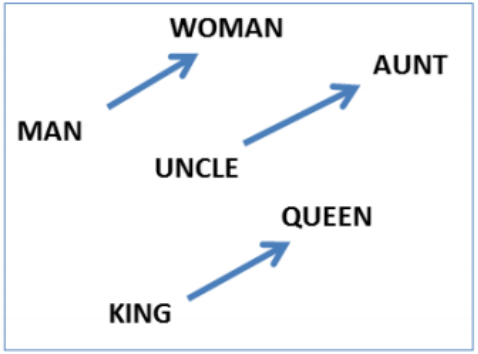
```
# save and reload the model
model.save('newmodel')
model = Word2Vec.load('newmodel')
```

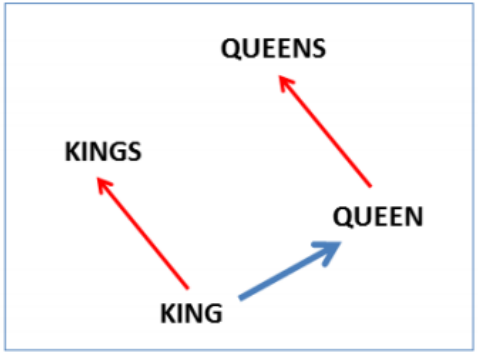
Word2Vec (code)

```
vector = model.wv['woman'] - model.wv['man'] + model.wv['king']
output = model.wv.similar_by_vector(vector)
output = np.array(output)

print(output)
```

```
[['king' '0.849290132522583']
 ['queen' '0.7148767113685608']
 ['prince' '0.6808396577835083']
 ['empress' '0.6308130621910095']
 ['son' '0.6299114227294922']
 ['throne' '0.6296331286430359']
 ['emperor' '0.621495246887207']
 ['princess' '0.6193820238113403']
 ['aragon' '0.6184202432632446']
 ['kings' '0.6152184009552002']]
```





Word2Vec (code)

```
# train words whose frequency > 3000
model = Word2Vec(corpus, min_count=3000, workers=4)
```

```
# keys: a list of trained words
keys = list(model.wv.vocab.keys())
# X: a list of word vectors
X = model.wv[keys]

# apply tsne to project X into 2d space
tsne = TSNE(n_components=2)
X_tsne = tsne.fit_transform(X)
```

```
# plot on the 2d space
plt.figure(figsize=(16, 16))
for i in range(len(X)):
  plt.scatter(X_tsne[i, 0], X_tsne[i, 1])
  plt.annotate(keys[i], xy=(X_tsne[i, 0], X_tsne[i, 1]))
plt.show()
```

Reference

- paper: https://arxiv.org/pdf/1301.3781.pdf
- paper: https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf
- library: https://radimrehurek.com/gensim/models/word2vec.html
- blog: https://shuuki4.wordpress.com/2016/01/27/word2vec-%EA%B4%80%EB%A0%A8-%EC%9D%B4%EB%A1%A0-%EC%A0%95%EB%A6%AC/