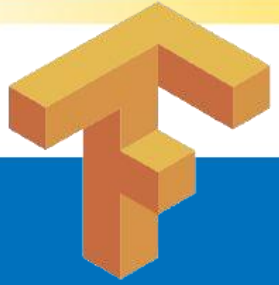


Word Embedding

Keras for Deep Learning Research

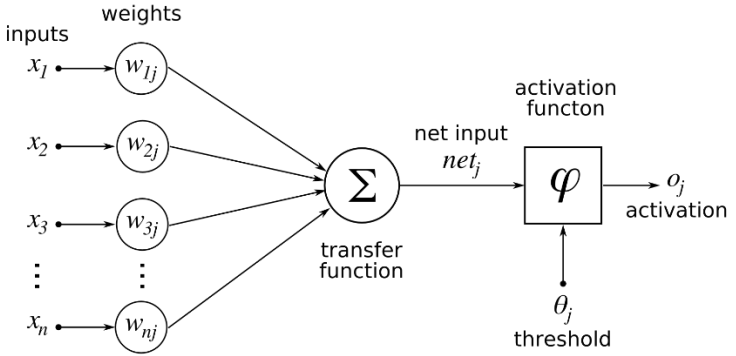
UMKC

Feedback is greatly appreciated!

- Let's quickly review what we've covered so far!

Deep Learning

- Single Neuron
- We now understand how to perform a calculation in a neuron
  - $w \cdot x + b = z$
  - $a = \sigma(z)$



Source: https://www.udemy.com/complete-guide-to-tensorflow-for-deep-learning-with-python/learn/v4/overview

UMKC

Deep Learning

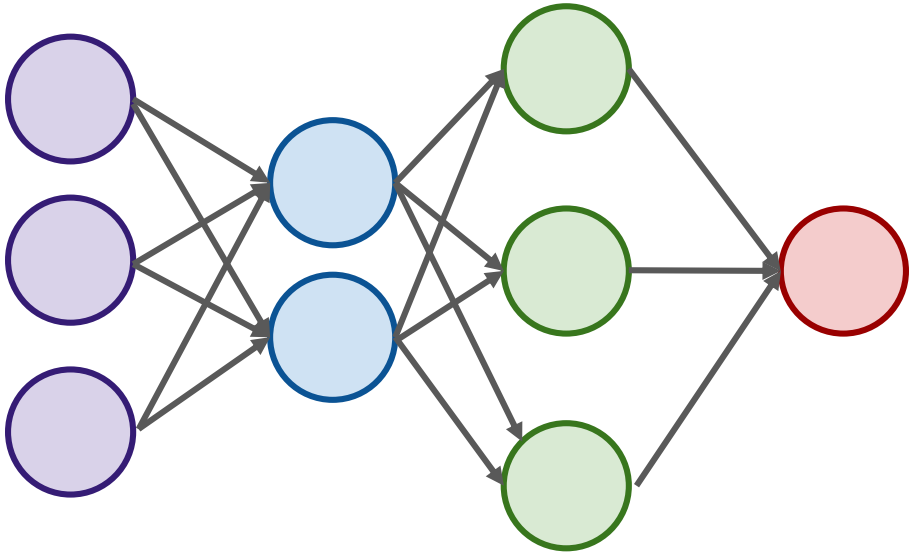
- Activation Functions
  - Threshold Function/Step Function
  - Sigmoid
  - Rectifier
  - Hyperbolic Tangent(tanh)

Source: https://www.udemy.com/complete-guide-to-tensorflow-for-deep-learning-with-python/learn/v4/overview

UMKC

Deep Learning

Connecting Neurons to create a network



Source: https://www.udemy.com/complete-guide-to-tensorflow-for-deep-learning-with-python/learn/v4/overview

UMKC

Deep Learning

- Neural Network
  - Input Layer
  - Hidden Layers
  - Output Layer

- More layers → More Abstraction

Source: https://www.udemy.com/complete-guide-to-tensorflow-for-deep-learning-with-python/learn/v4/overview

UMKC

Deep Learning

- To "learn" we need some measurement of error.
  - We use a Cost/Loss Function
    - Quadratic
    - Cross-Entropy

Source: https://www.udemy.com/complete-guide-to-tensorflow-for-deep-learning-with-python/learn/v4/overview

UMKC

Deep Learning

- Once we have the measurement of error, we need to minimize it by choosing the correct weight and bias values.
- We use gradient descent to find the optimal values.

Source: https://www.udemy.com/complete-guide-to-tensorflow-for-deep-learning-with-python/learn/v4/overview

UMKC

Deep Learning

- We can then backpropagate the gradient descent through multiple layers, from the output layer back to the input layer.

Source: https://www.udemy.com/complete-guide-to-tensorflow-for-deep-learning-with-python/learn/v4/overview

UMKC

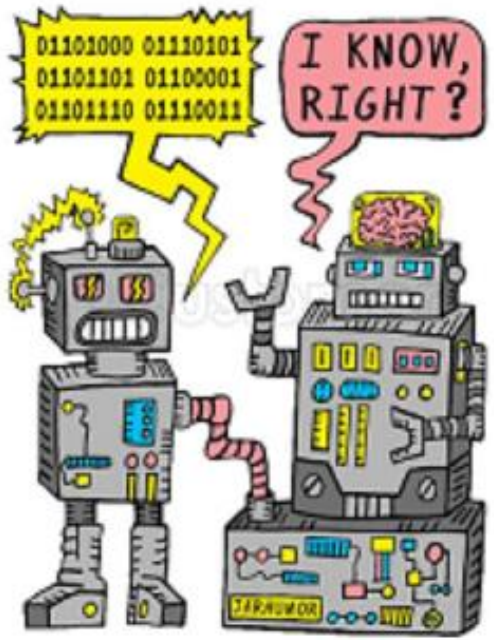
- Word Embedding

UMKC

Why do we use word embeddings in NLP?

- Natural language processing (NLP) is a sub-field of machine learning (ML)

- Dealing with text data is problematic, since our computers, scripts and machine learning models can’t read and understand text in any human sense



UMKC

- So, how should textual input be sent to our models?
- Computers can handle numerical input well, so let's rephrase the question to:

- ***How can we best numerically represent textual input?***

Bag-of-words model

One-Hot encoding

- The vector representation for our first vocabulary word “aardvark” will be [1, 0, 0, 0, …, 0], which is a “1” in the first position followed by 9,999 zeroes.
- This process is called **one-hot vector encoding**.

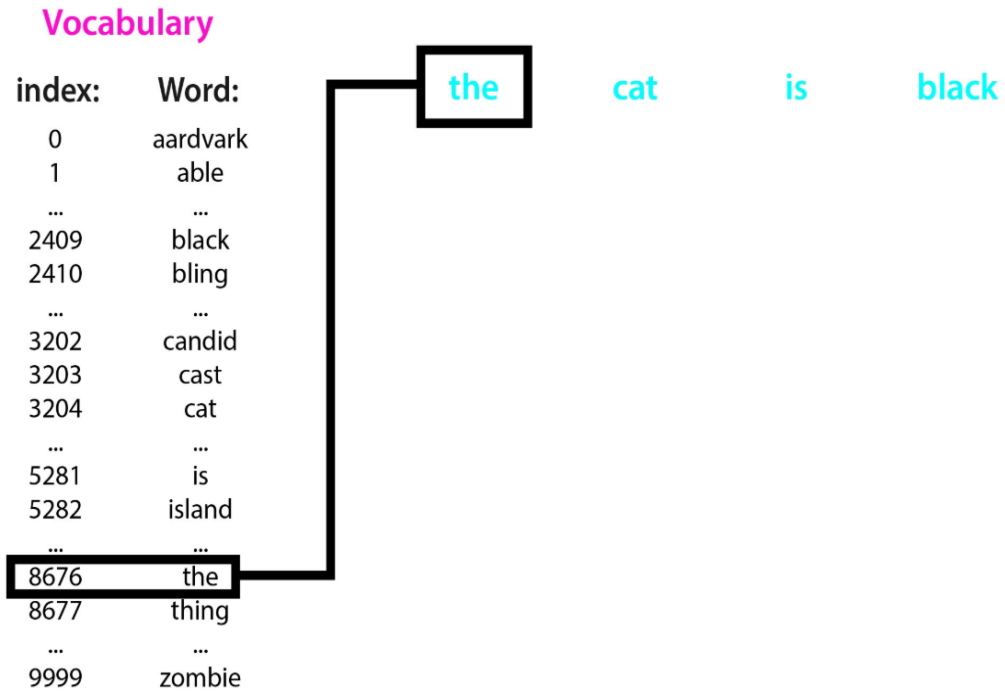
UMKC

- Now, say our NLP project is building a translation model and we want to translate the English input sentence “the cat is black” into another language.

- We first need to represent each word with a one-hot encoding

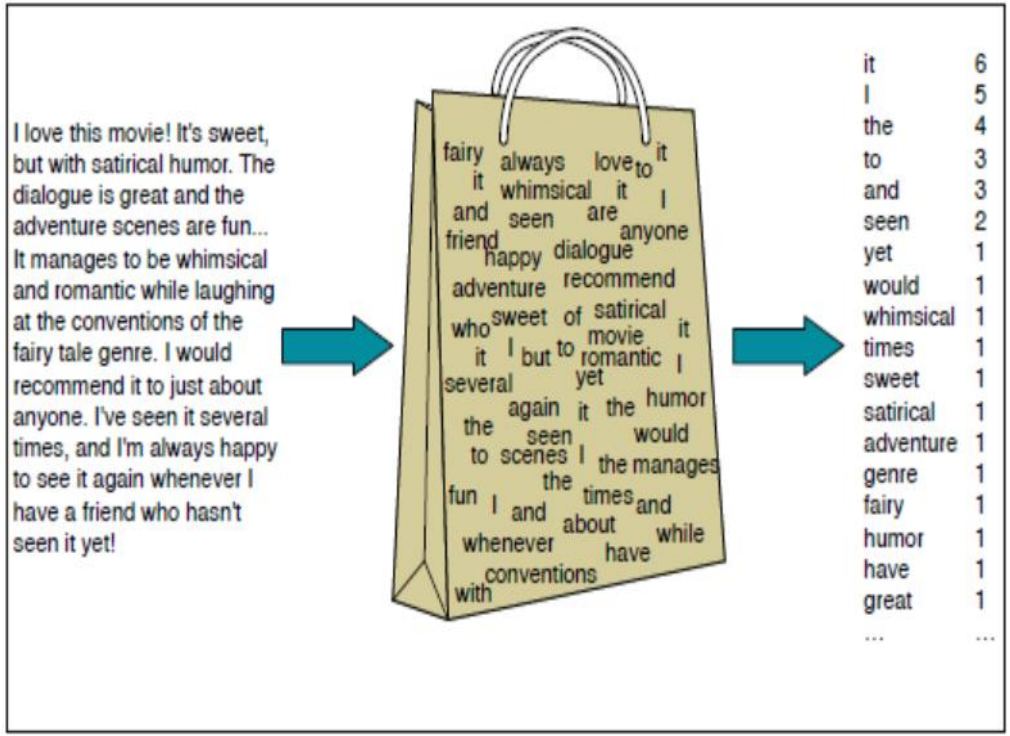
- We would first look up the index of the first word, “the”, and find that its index in our 10,000-long vocabulary list is 8676.

- We could then represent the word “the” using a length 10,000 vector, where every entry is a 0 aside from the entry at position 8676, which is a 1.
- We do this index look-up for every word in the input sentence, and create a vector to represent each input word.
- These one-hot vectors are a quick and easy way to represent words as vectors of real-valued numbers.



Count based model

- Bag-of-words language models rely on the term frequency TF, defined as the number of times that a word occurs in a given text or document



TFIDF Based Model

- We can represent each document by tfidf score

The problems with sparse encodings

- One-hot vectors are high-dimensional and sparse
- Feature vector grows with the vocabulary size
- Semantic and syntactic information are lost
- The computational issue.

Word Embedding

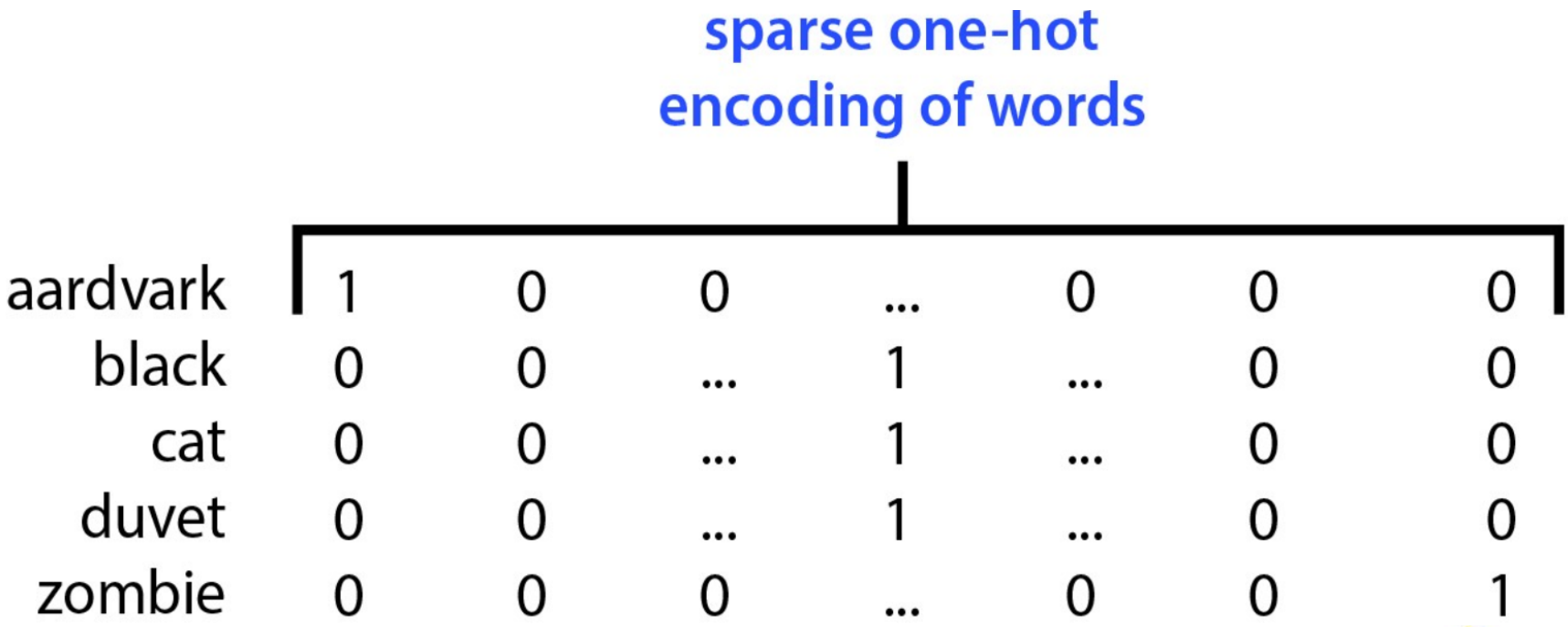
- Why do we need any complex methods to represent words at all?
- What is the simplest way to represent words numerically, and why isn't this sufficient?
- What exactly do we mean by embeddings "mapping words to a high-dimensional semantic space"?
- How can word embeddings get visualised and intuitively understood?

UMKC

Idea behind Word Embedding

- The core idea is that words that are used in similar contexts will be given similar representations.
- That is, words that are used in similar ways will be placed close together within the high-dimensional semantic space–
- These points will cluster together, and their distance to each other will be low.

Example on one hot encoding



UMKC

Equivalent Word embedding

	animal	fluffiness	dangerous	spooky
aardvark	0.97	0.03	0.15	0.04
black	0.07	0.01	0.20	0.95
cat	0.98	0.98	0.45	0.35
duvet	0.01	0.84	0.12	0.02
zombie	0.74	0.05	0.98	0.93

UMKC

Types of Word Embeddings

Word embeddings can be broadly classified into two categories

- Frequency based Embedding
  - Count Vector
  - TF-IDF Vector
  - Co-Occurrence Vector
- Prediction based Embedding
  - word2vec
    - CBOW (Continuous Bag of words)
    - Skip-Gram Model
  - GloVe: Global Vectors for Word Representation

Source: https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/

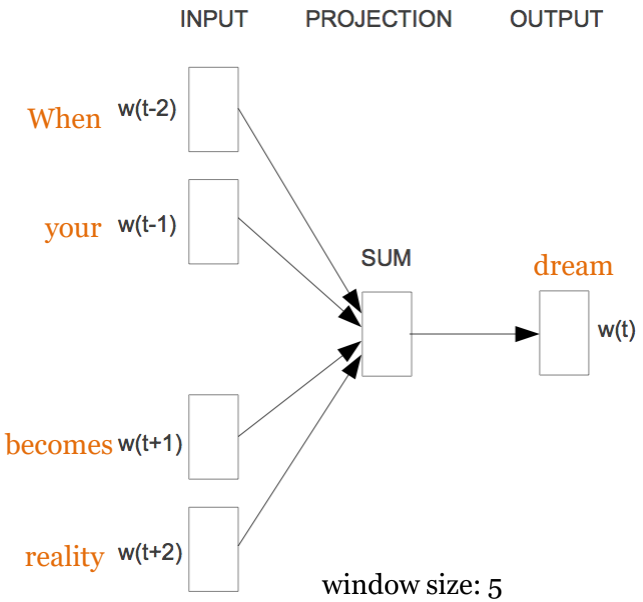
UMKC

Two different architecture for word2vec

- **CBOW:**
  - ✓ The *input* to the model is the *preceding and following words of the current word*
  - ✓ The *output* of the neural network will be ***Wi***
- **Skip-gram:**
  - ✓ The *input* to the model is ***wi*** and
  - ✓ The *output* is the *words around it*

UMKC

Word2vec - CBOW (Continuous Bag of words)



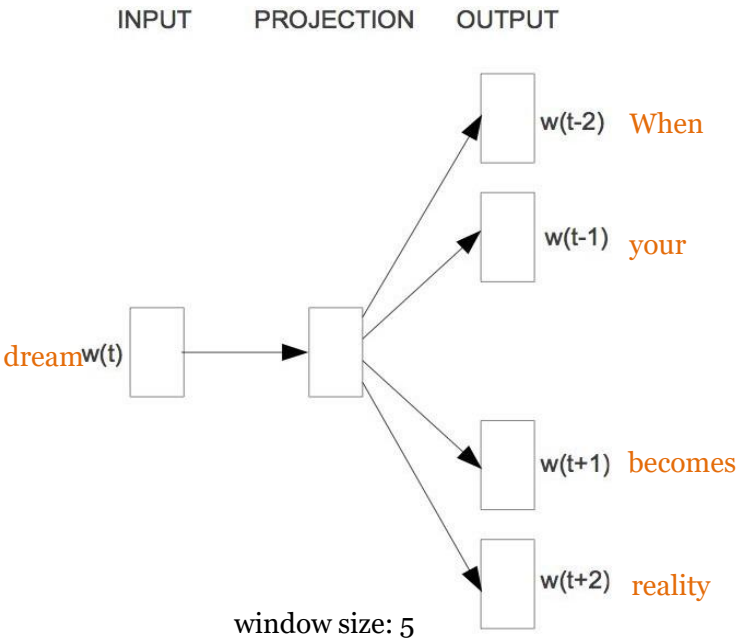
The way CBOW work is that it tends to predict the probability of a word given a context.

Let us start with an example (*Tweet*):
"When your dream becomes reality"

One approach is to treat ("When", "your", "becomes", "reality“) as a context and from these words, be able to predict or generate the center word "dream".

Source: https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/

Word2vec - Skip-Gram Model



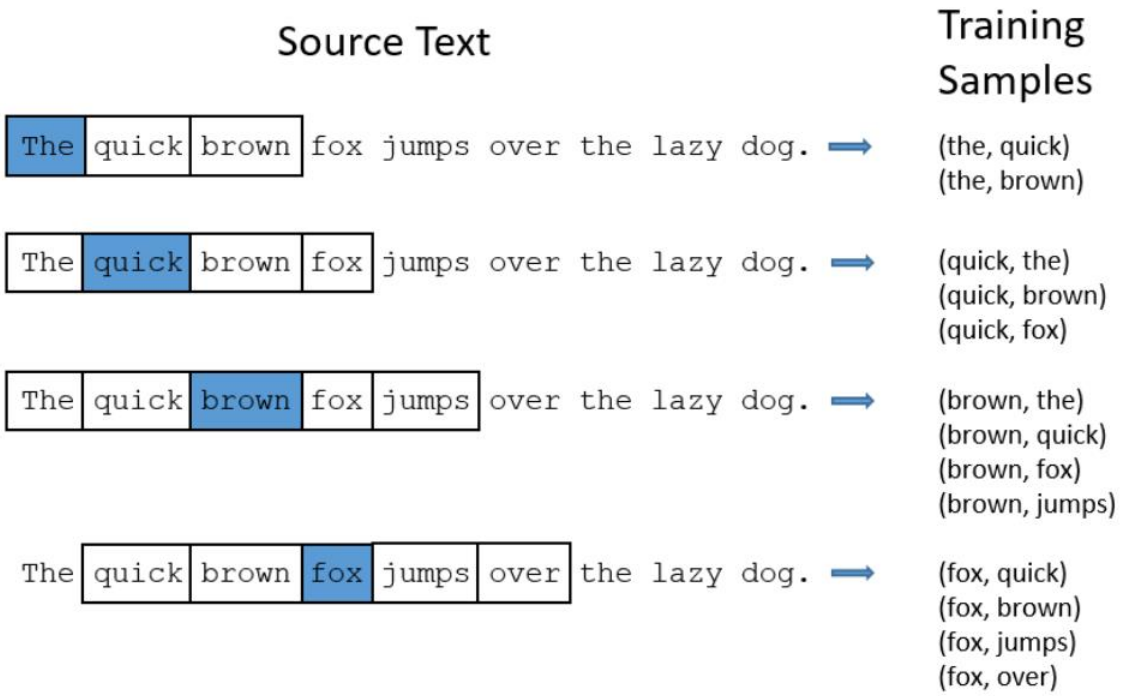
The aim of skip-gram is to predict the context given a word.

Let us check with same example (*Tweet*):
"When your dream becomes reality"

The approach is to create a model such that given the center word "dream", the model will be able to predict or generate the surrounding words ("When", "your", "becomes", "reality"). Here we call the word "dream" the context.

Source: https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/

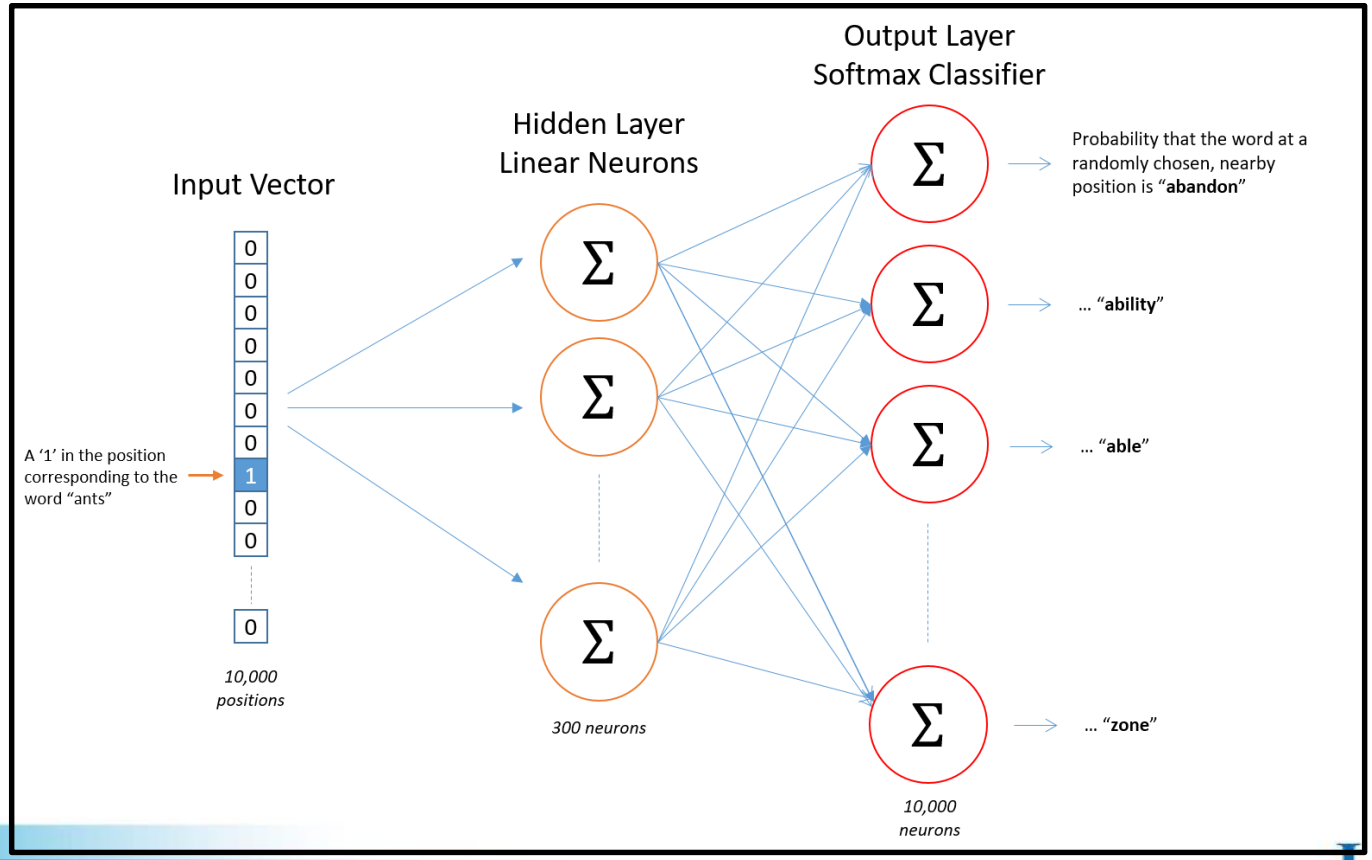
Preparing training sample



Word2Vec Skip-gram model

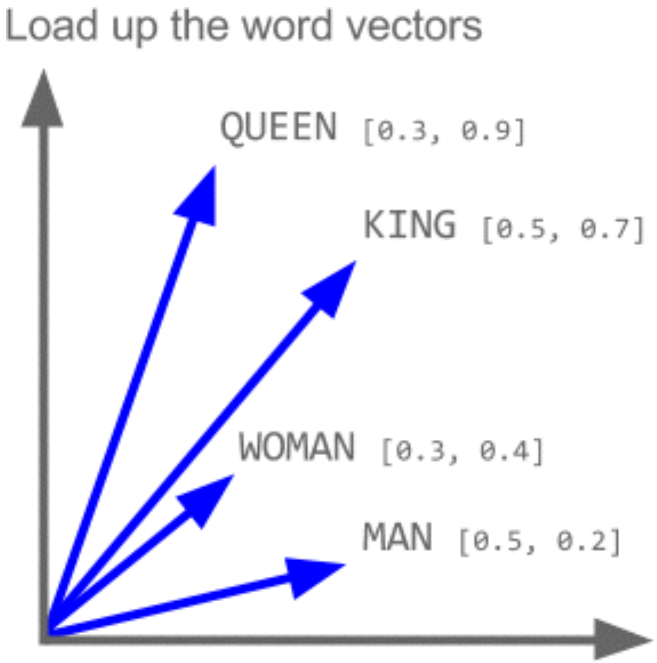
UMKC

word2vec. skip-gram model

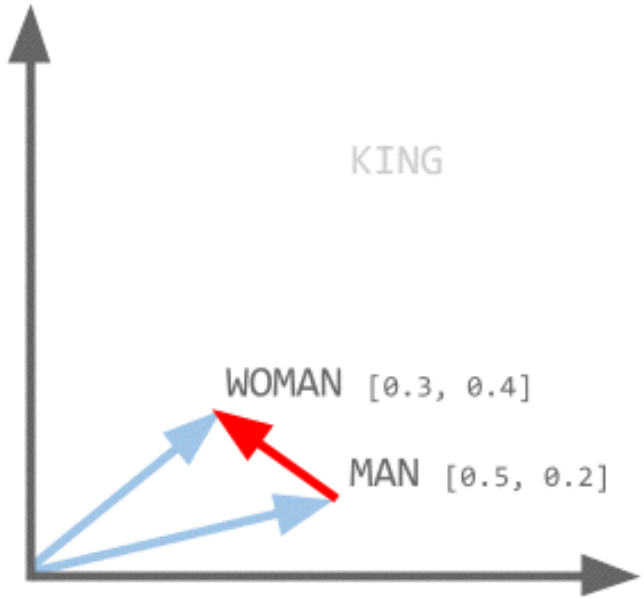


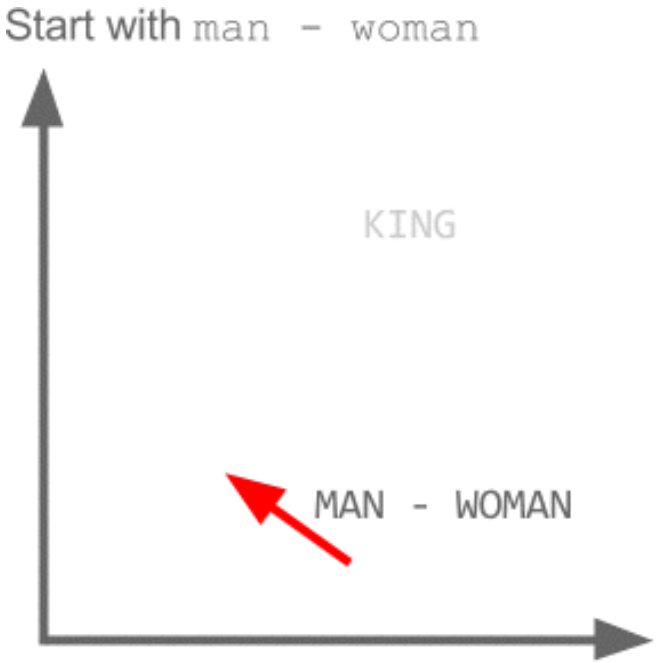
Properties of word2vec

- .

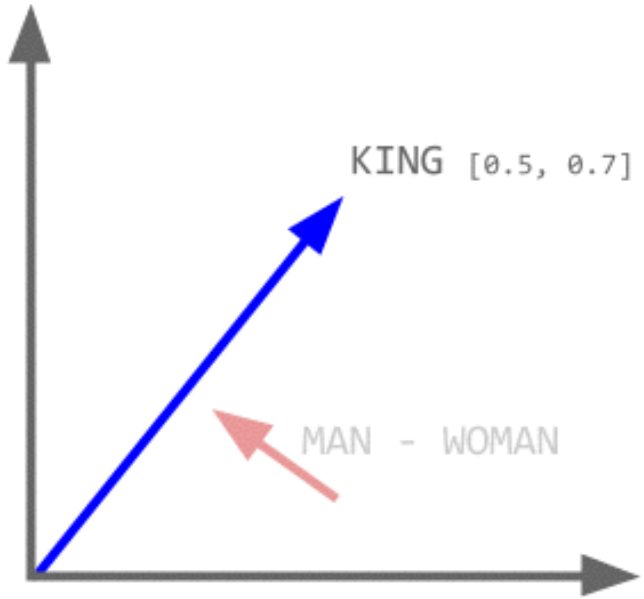


Start with `man - woman`

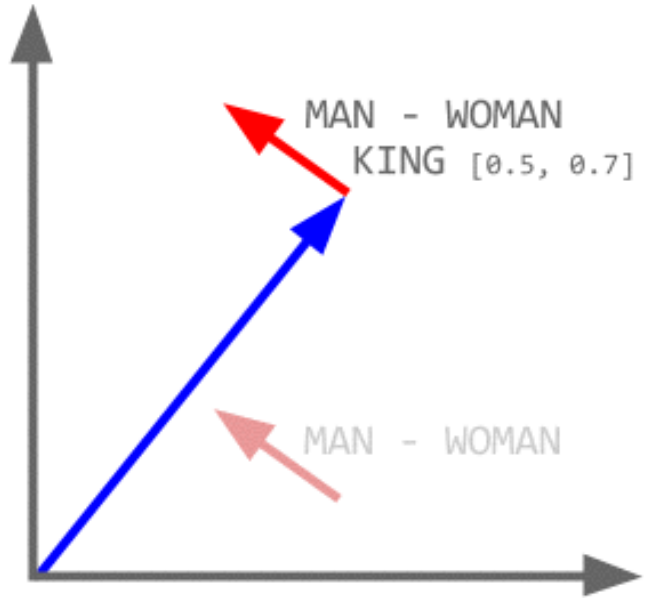


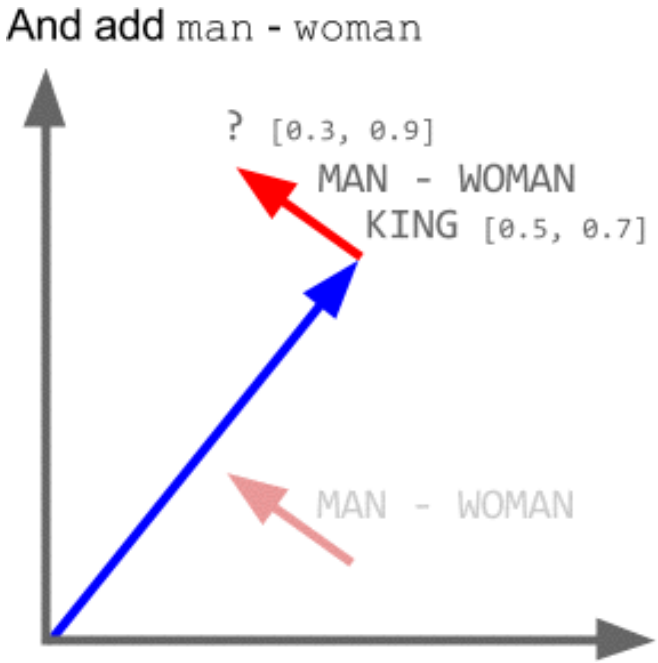

Start with man - woman
KING
MAN - WOMAN

Then take `king`

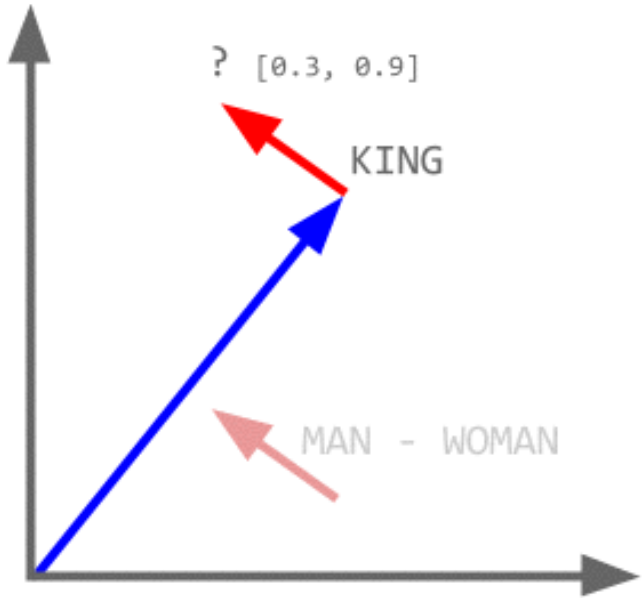


And add `man - woman`

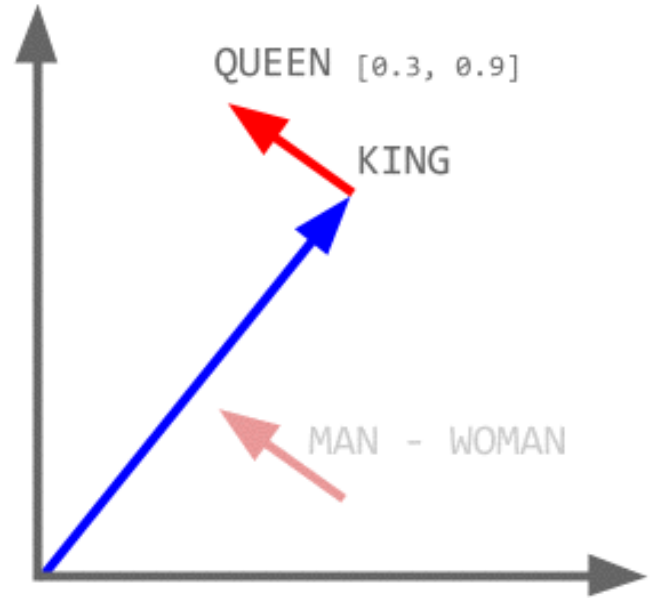



And add man - woman
? [0.3, 0.9]
MAN - WOMAN
KING [0.5, 0.7]
MAN - WOMAN

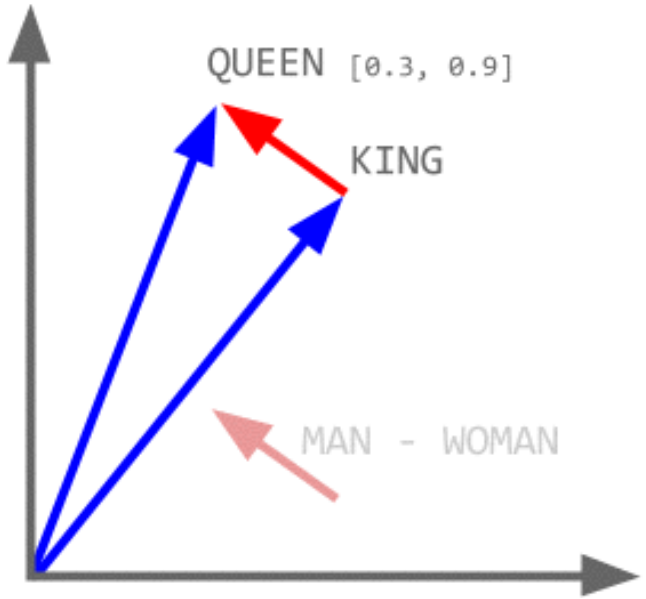
Find nearest word to result



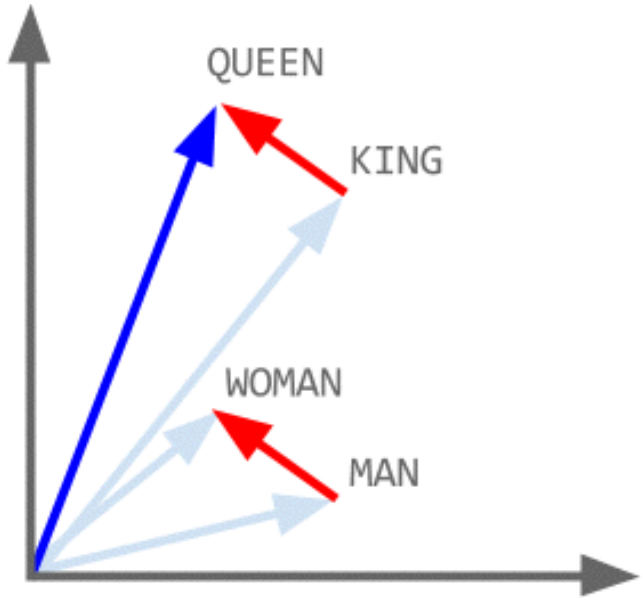
`queen` is closest to resulting vector



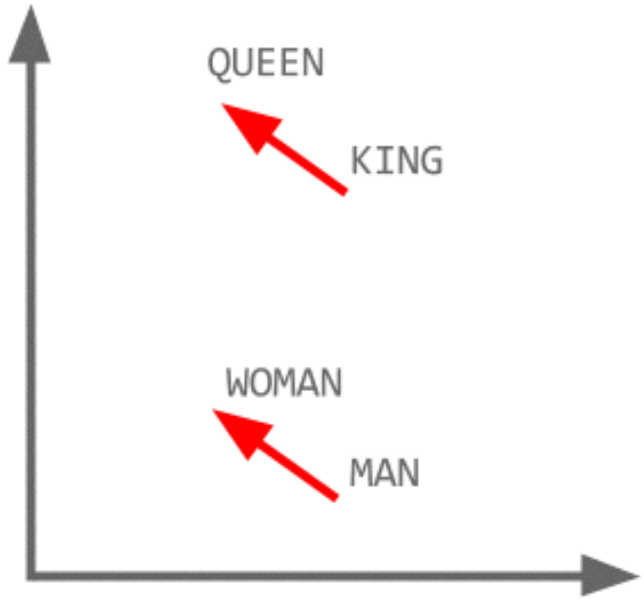
`queen` is closest to resulting vector



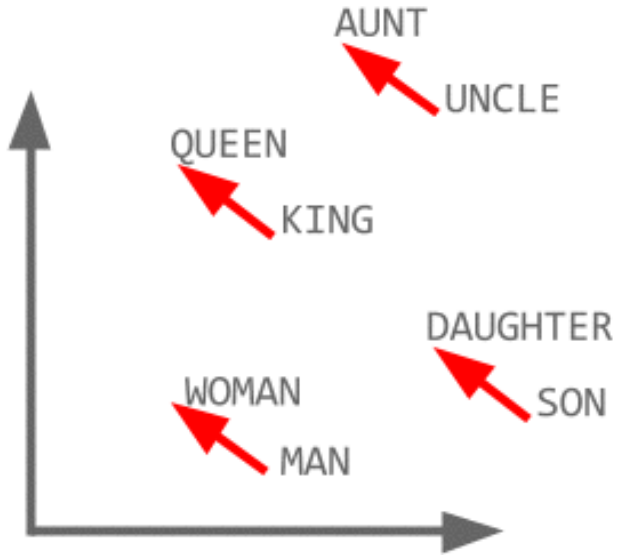
So `king + man - woman = queen!`



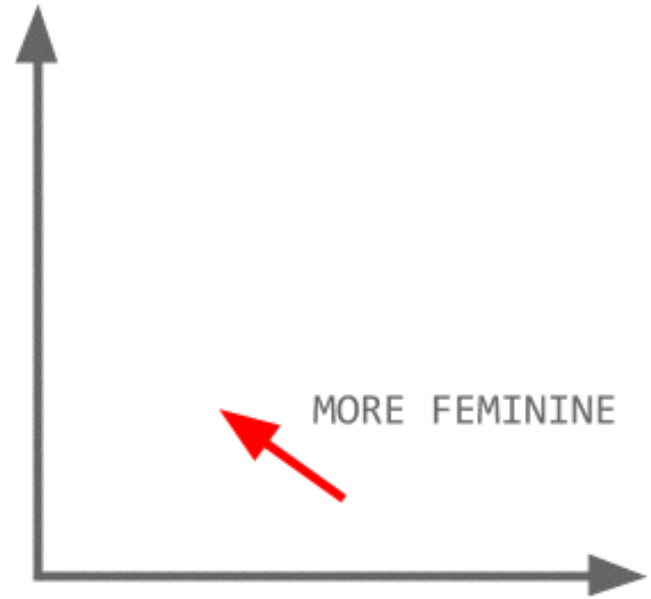
The red direction encodes gender



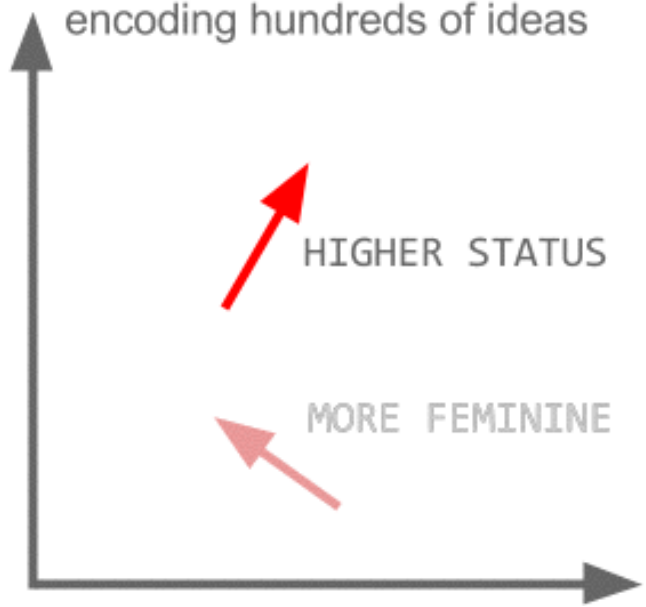
Which is consistent across all words



This direction always means gender



We have hundreds of directions
encoding hundreds of ideas



UMKC

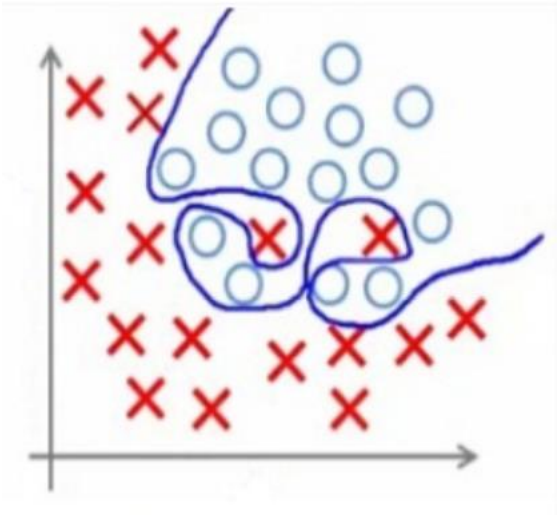
Overfitting in machine learning

- Overfitting refers to a model that models the training data too well.
- Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data

UMKC

Overfitting

- Overfitting is the case where the overall cost is really small, but the generalization of the model is unreliable.
- This is due to the model learning "too much" from the training data set.

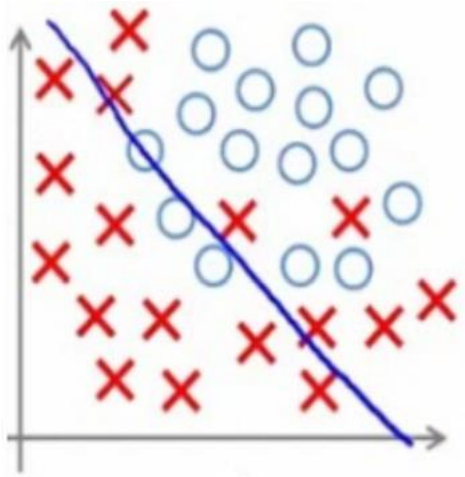


Underfitting in Machine Learning

- Underfitting refers to a model that can neither model the training data nor generalize to new data.
- An underfit machine learning model is not a suitable model and will be obvious as it will have poor performance on the training data.

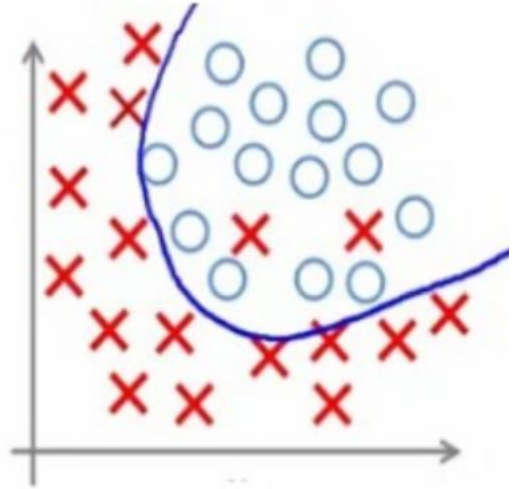
Underfitting

- when a machine learning model is not complex enough to accurately capture relationships between a dataset's features and a target variable



- Arguably, Machine Learning models have one sole purpose; to generalize well
- *Generalization is the model's ability to give sensible outputs to sets of input that it has never seen before*

Best fitted

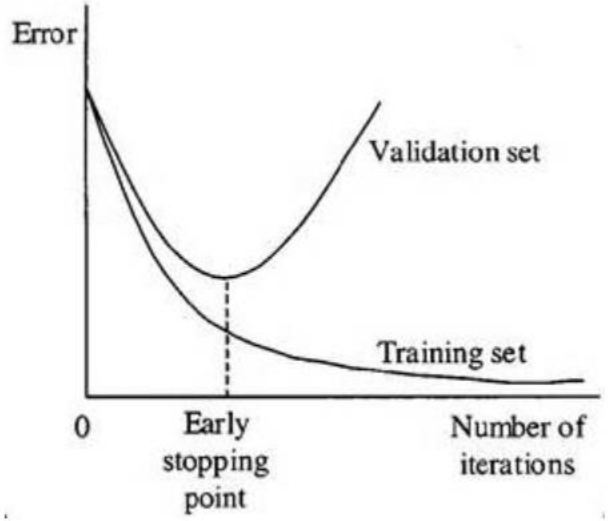


How to overcome it?

- Cross-validation (K-fold cross validation)
- Train with more data
- Early stopping (mostly in deep learning model)
- Regularization:
  - Regularization refers to a broad range of techniques for artificially forcing your model to be simpler.
    - The method will depend on the type of learner you're using.
    - For example, you could prune a decision tree,
    - use dropout on a neural network,
    - or add a penalty parameter to the cost function in regression
- Ensembling

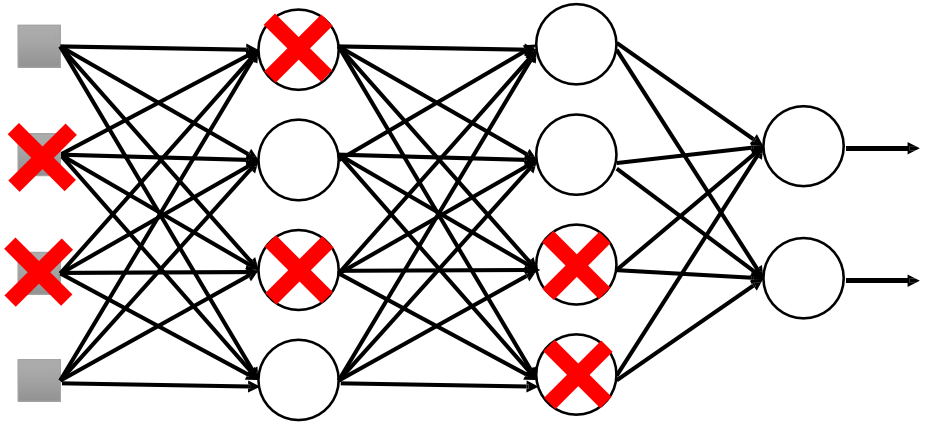
UMKC

Early stopping



Regulizer: Dropout

- .



- **Each time before updating the parameters**
  - Each neuron has p% to dropout

Programming

Embedding layer in Keras

- **input_dim**: This is the size of the vocabulary in the text data.
- For example, if your data is integer encoded to values between 0-10, then the size of the vocabulary would be 11 words.
- **output_dim**: This is the size of the vector space in which words will be embedded. It defines the size of the output vectors from this layer for each word. For example, it could be 32 or 100 or even larger.
- **input_length**: This is the length of input sequences, as you would define for any input layer of a Keras model. For example, if all of your input documents are comprised of 1000 words, this would be 1000.

UMKC

Embedding layer in Keras

- Preparing data for embedding layer:

```
max_review_len = max([len(s.split()) for s in sentences])
vocab_size = len(tokenizer.word_index)+1
sentences = tokenizer.texts_to_sequences(sentences)
padded_docs = pad_sequences(sentences,maxlen=max_review_len)
```

- Adding embedding layer in keras:

```
model.add(Embedding(vocab_size, 50, input_length=max_review_len))
```

UMKC

Use case: Sentiment Classification, IMDB dataset

	ID0	review	label
0	0	Once again Mr. Costner has dragged out a movie...	neg
1	3	Not even the Beatles could write songs everyon...	neg
2	9	Wealthy horse ranchers in Buenos Aires have a ...	neg
3	10	Cage plays a drunk and gets high critically pr...	neg
4	11	First of all, I would like to say that I am a ...	neg

UMKC

Reading the data

```
df = pd.read_csv('imdb_master.csv',encoding='latin-1')
sentences = df['review'].values
y = df['label'].values
```

Steps to prepare the text data (review column)

- Keras provides the Tokenizer class for preparing text documents for deep learning.
- The Tokenizer must be constructed and then fit on either raw text documents or integer encoded text documents.

```
tokenizer = Tokenizer(num_words=2000)
tokenizer.fit_on_texts(sentences)
```

UMKC

Encoding the target column

```
le = preprocessing.LabelEncoder()
y = le.fit_transform(y)
```

Creating and fitting the model

What is input_dim here?

```
model = Sequential()
model.add(layers.Dense(300,input_dim=input_dim, activation='relu'))
```

Is this number of neurons for the last layer is correct?

Is this activation function for the last layer is correct?

```
model.add(layers.Dense(5, activation='sigmoid'))
model.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['acc'])
model.fit(X_train,y_train, epochs=5, verbose=True, validation_data=(X_test,y_test), batch_size=256)
```

UMKC

References

https://deeplearning4j.org/word2vec.html
https://towardsdatascience.com/tagged/word2vec
https://medium.com/@Aj.Cheng/word2vec-3b2cc79d674
https://medium.com/tag/word2vec
https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/
http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/
https://becominghuman.ai/how-does-word2vecs-skip-gram-work-f92e0525def4
https://machinelearningmastery.com/prepare-text-data-deep-learning-keras/

UMKC