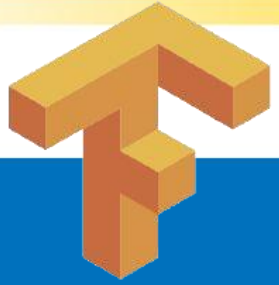


Word Embedding

Keras for Deep Learning Research

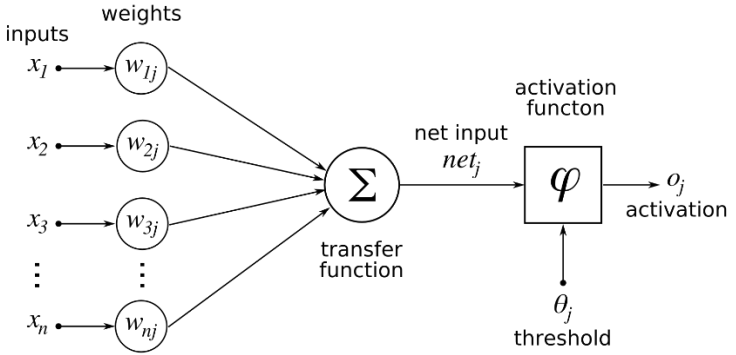
UMKC

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

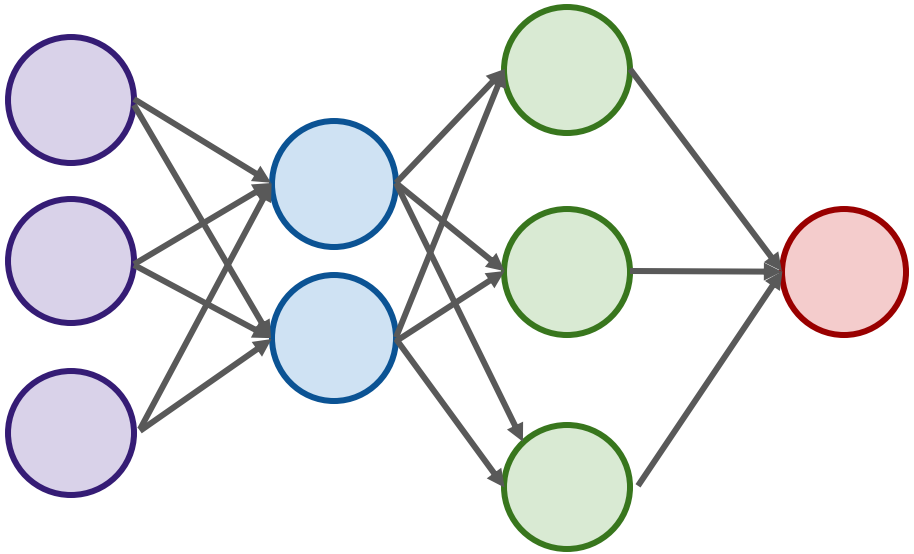
- Activation Functions
  - Threshold Function/Step Function
  - Sigmoid
  - Rectifier
  - Hyperbolic Tangent(tanh)

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

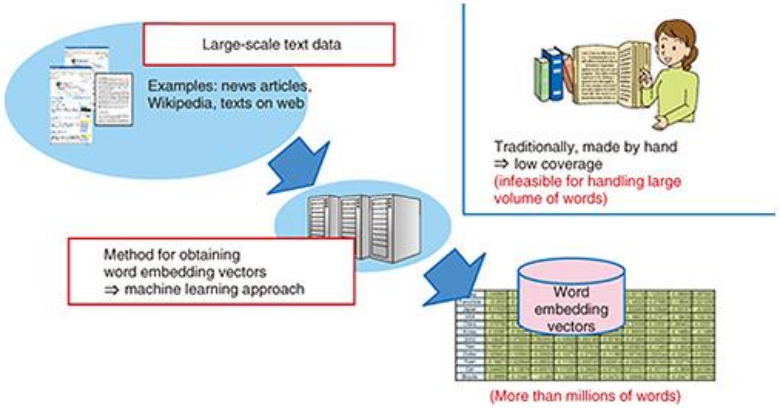
- We can then backpropagate the gradient descent through multiple layers, from the output layer back to the input layer.

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

UMKC

Why Word Embeddings: Essential for text processing with DL

- It is an essential part of text processing with Deep Learning
- All Deep Learning architectures, are incapable of processing strings or plain text in their raw form



TEXT processing

Source: https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/
https://www.ntt-review.jp/archive/ntttechnical.php?contents=ntr201711fa4.html

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

How represent word as vector?

UMKC

One-hot-encoding

Vocabulary:
Man, woman, boy, girl, prince, princess, queen, king, monarch

	1	2	3	4	5	6	7	8	9
man	1	0	0	0	0	0	0	0	0
woman	0	1	0	0	0	0	0	0	0
boy	0	0	1	0	0	0	0	0	0
girl	0	0	0	1	0	0	0	0	0
prince	0	0	0	0	1	0	0	0	0
princess	0	0	0	0	0	1	0	0	0
queen	0	0	0	0	0	0	1	0	0
king	0	0	0	0	0	0	0	1	0
monarch	0	0	0	0	0	0	0	0	1

Each word gets a 1x9 vector representation

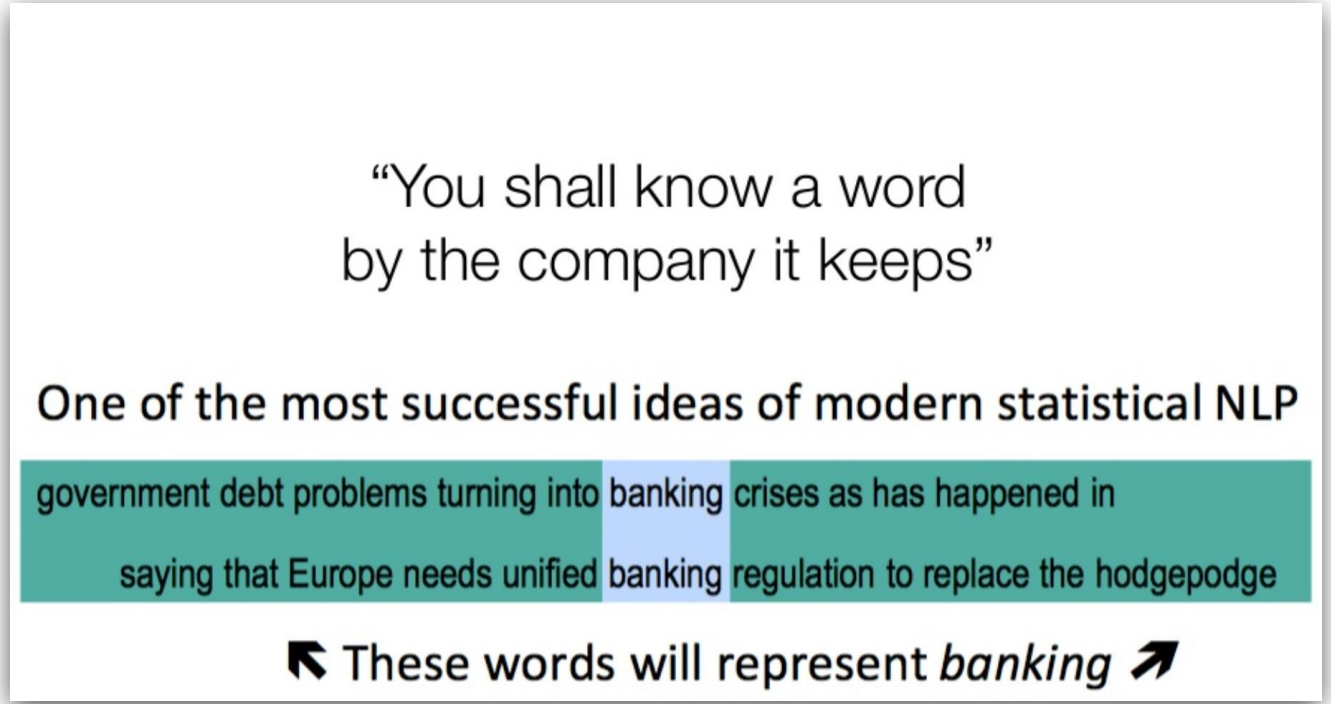
UMKC

Problems of one-hot-encoding

.

1. One-hot vectors are high-dimensional and sparse
2. Feature vector grows with the vocabulary size
3. Semantic and syntactic information are lost

word2vec

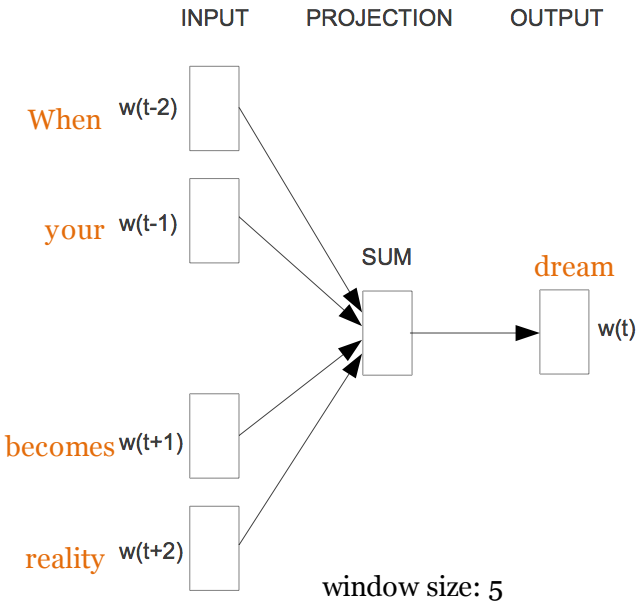

"You shall know a word
by the company it keeps"
One of the most successful ideas of modern statistical NLP
government debt problems turning into banking crises as has happened in
saying that Europe needs unified banking regulation to replace the hodgepodge
These words will represent *banking*

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)



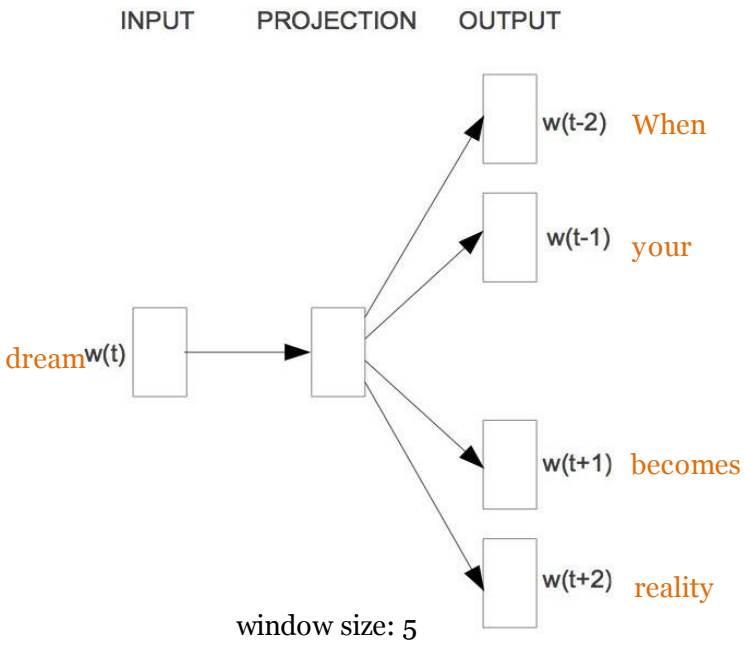
The way CBOW work is that it tends to predict the probability of a word given a context.

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.

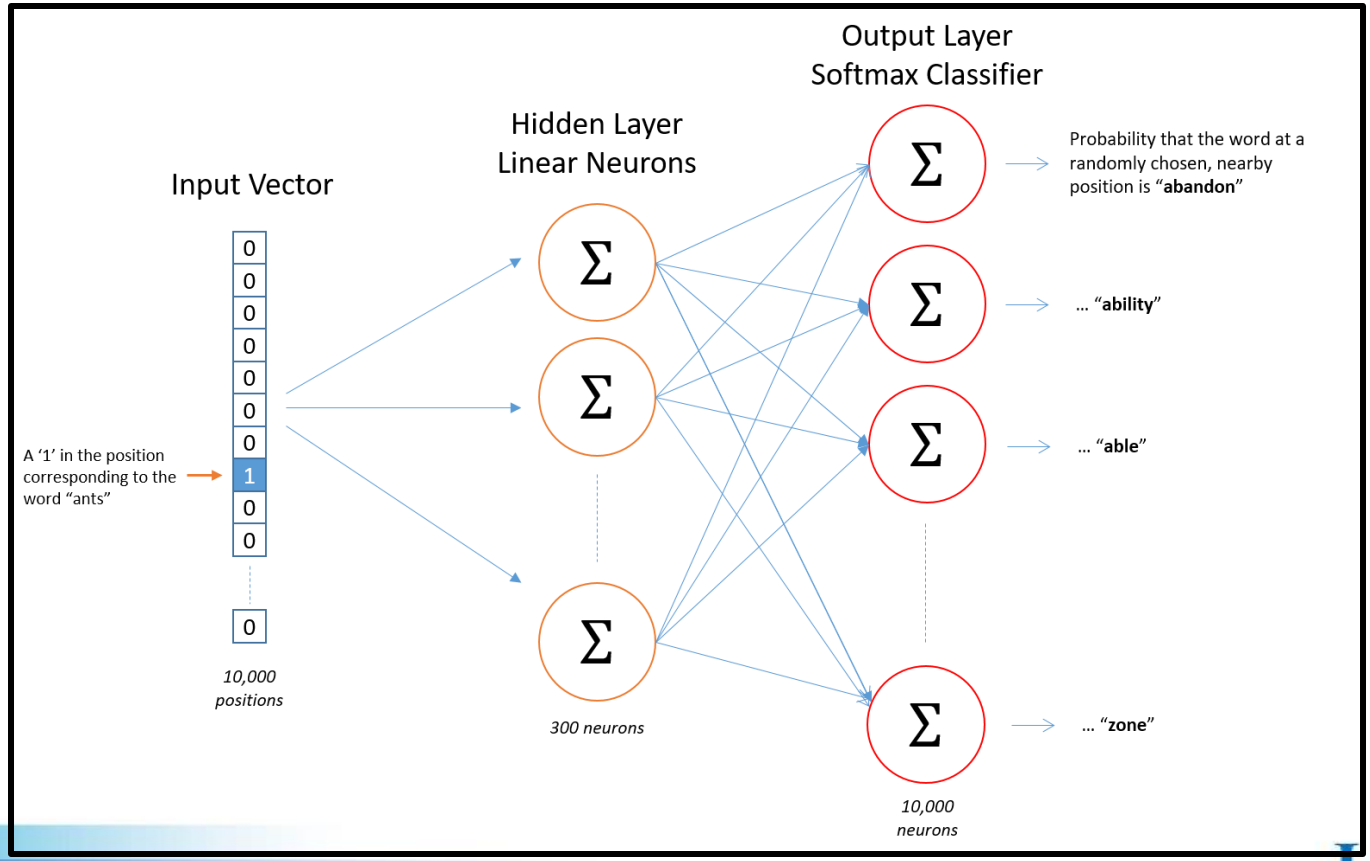
Let us check with same example (*Tweet*):
"When your dream becomes reality"

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/

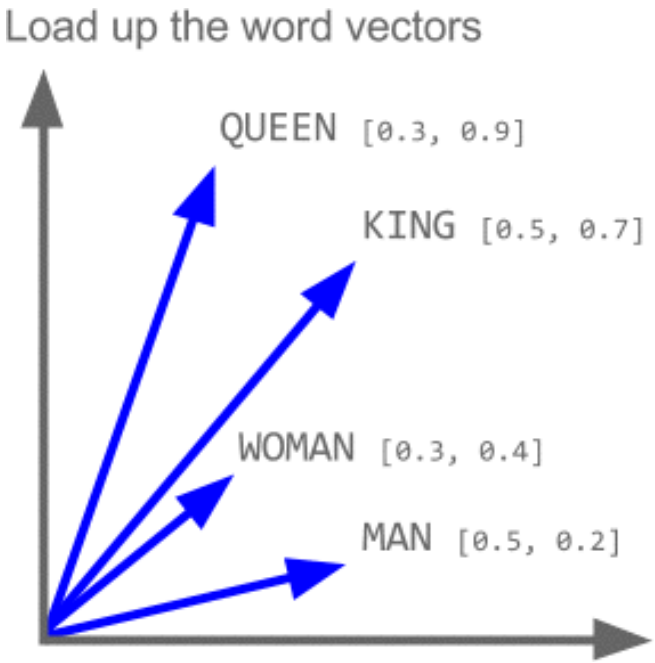
Word2Vec Skip-gram model

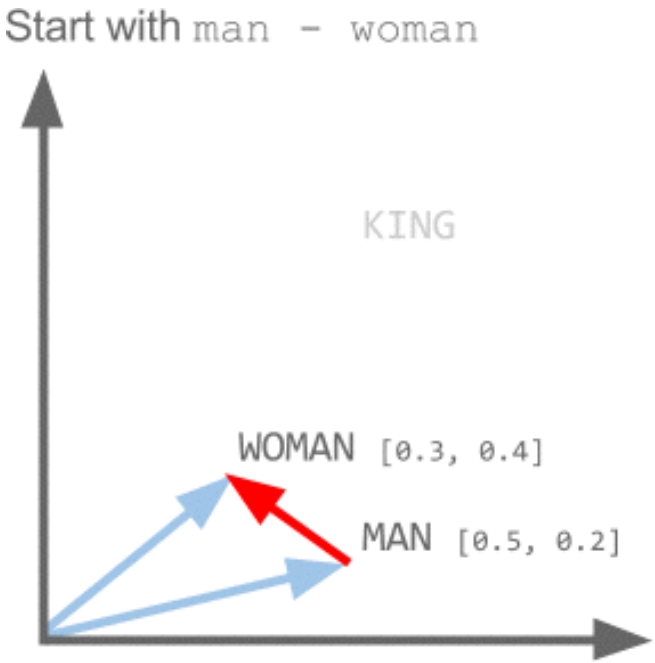
word2vec. skip-gram model

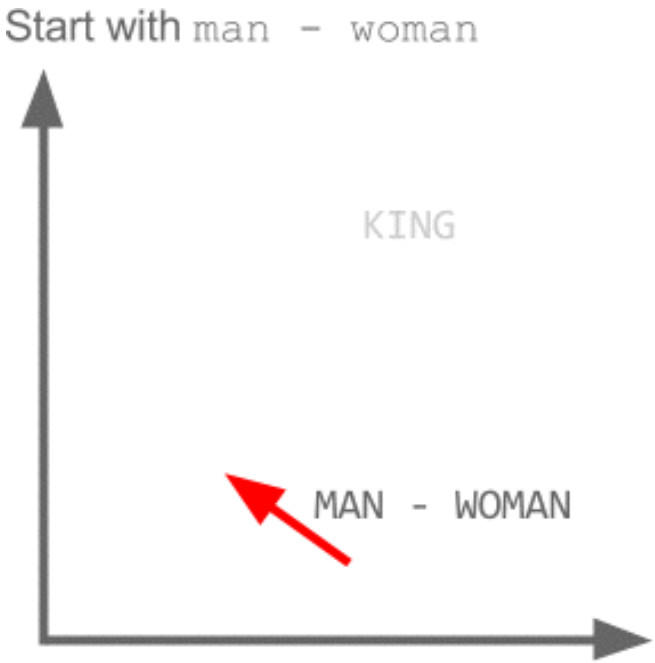


Properties of word2vec

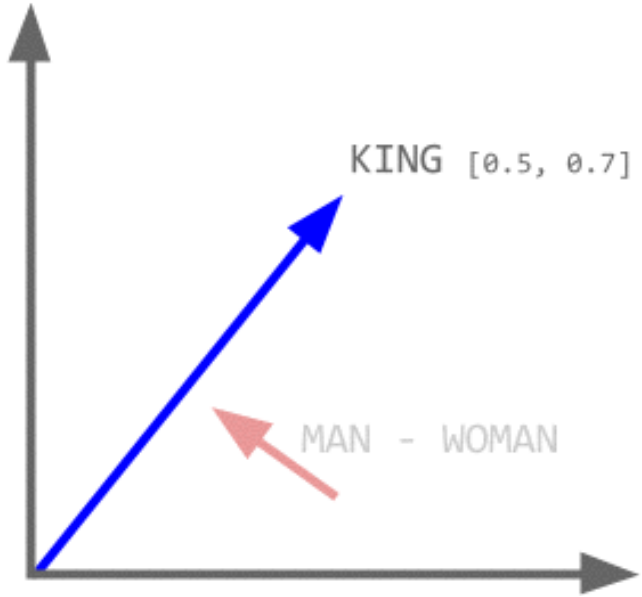
- .



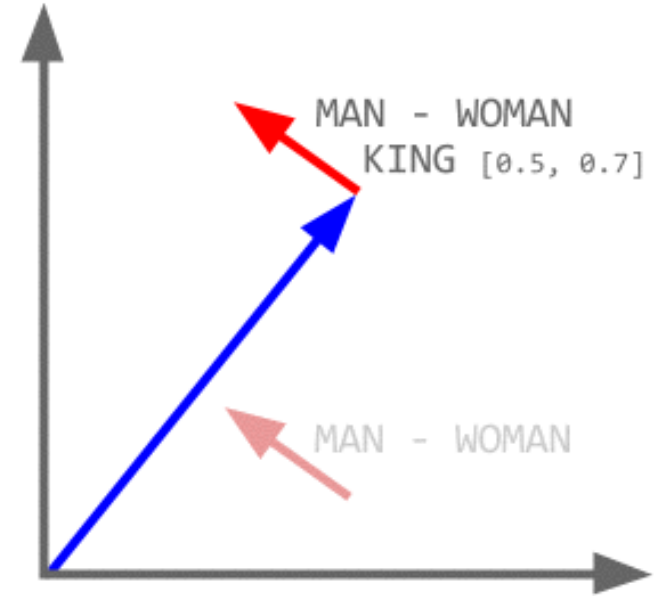

Start with man - woman
KING
WOMAN [0.3, 0.4]
MAN [0.5, 0.2]


Start with man - woman
KING
MAN - WOMAN

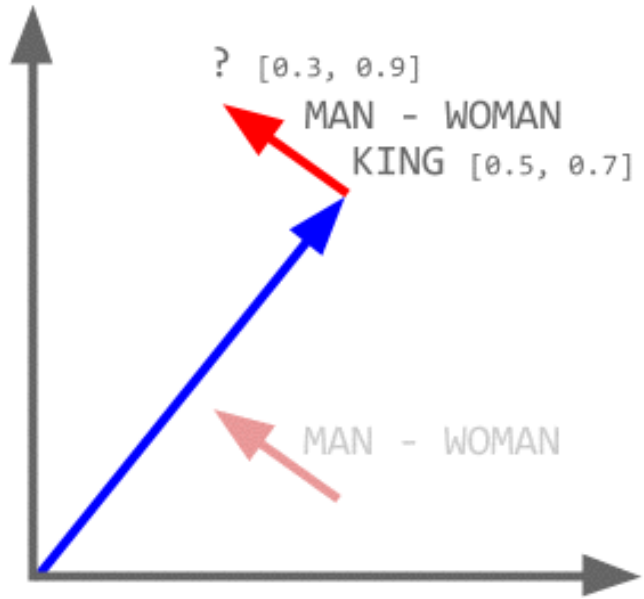
Then take `king`



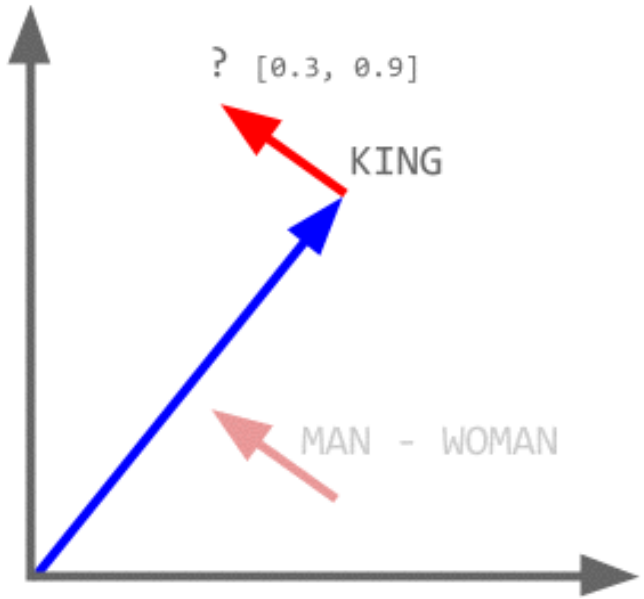
And add `man - woman`



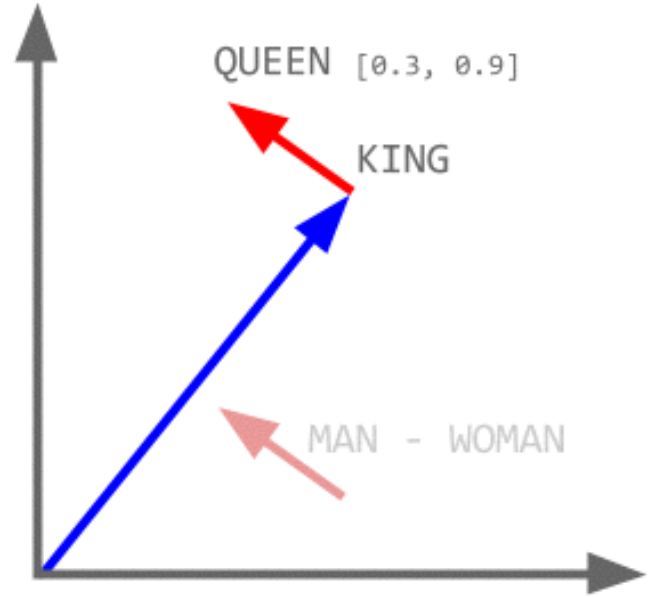
And add `man - woman`



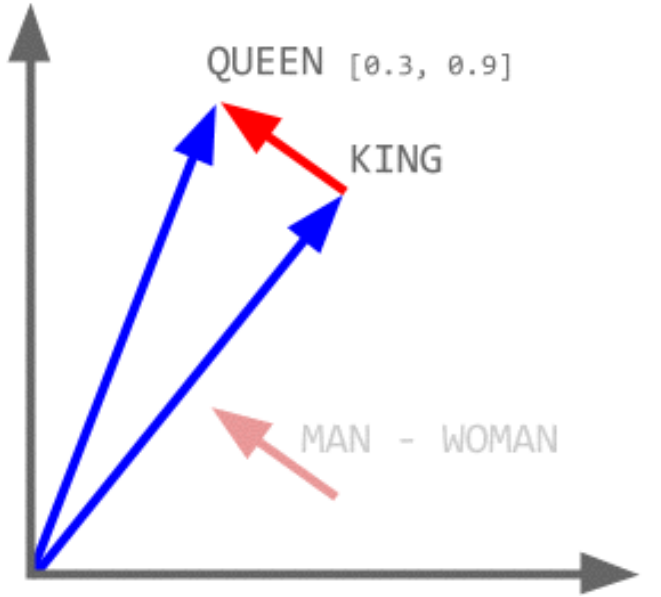
Find nearest word to result



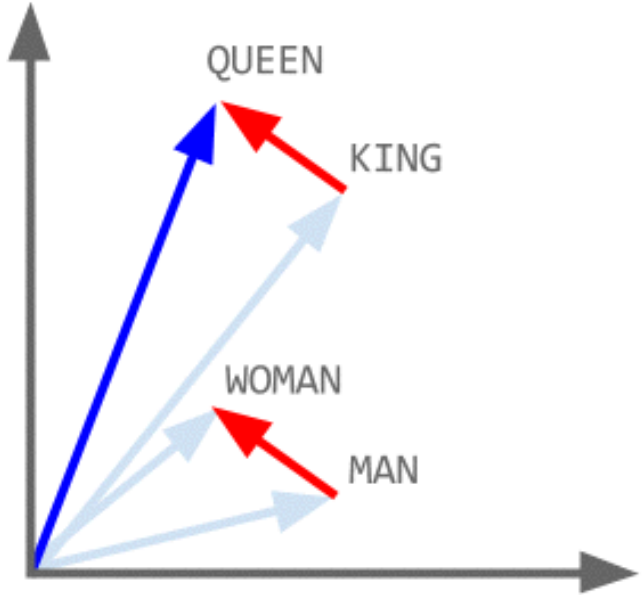
`queen` is closest to resulting vector



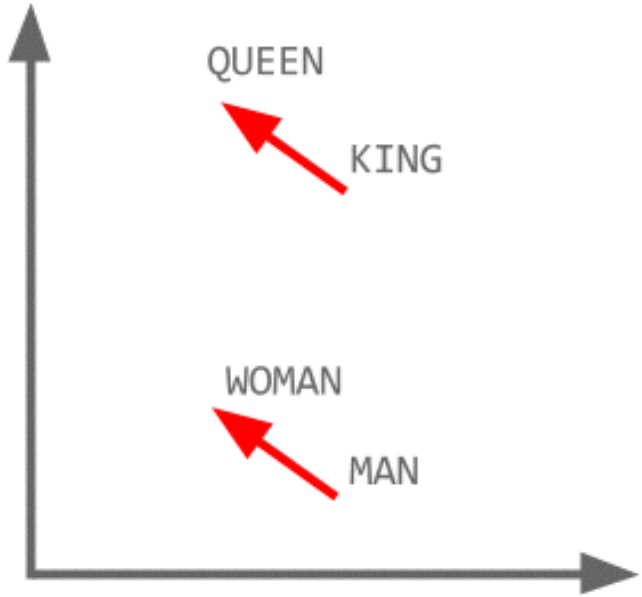
`queen` is closest to resulting vector



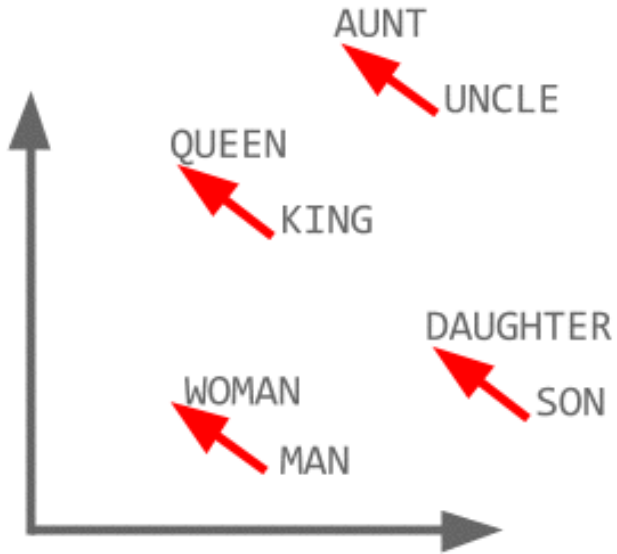
So `king + man - woman = queen!`



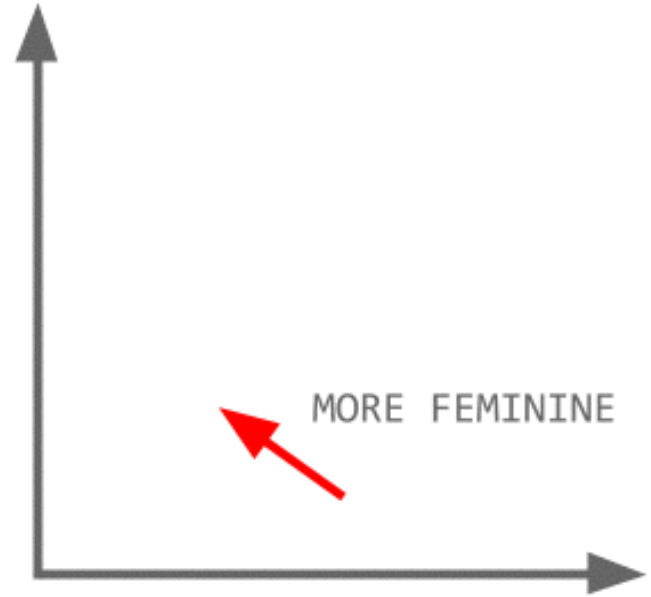
The red direction encodes gender



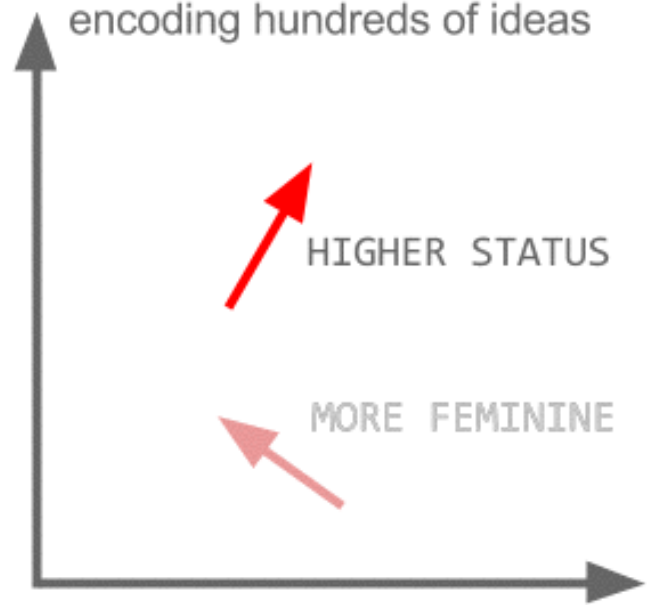
Which is consistent across all words



This direction always means gender



We have hundreds of directions
encoding hundreds of ideas

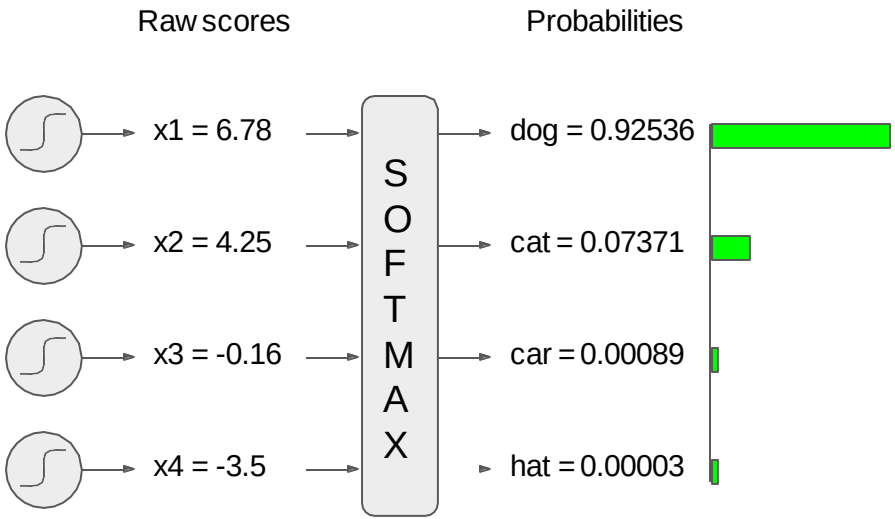


Activation functions

- We learned that non-linear activation function gives non-linearity to the model which is good
- Sigmoid: range (0,1) can be used for the binary classification
- Tangent: range (-1, 1)
- Relu: range (0, infinity)
- Softmax: The **softmax function** is a more generalized logistic activation function which is used for multiclass classification.

UMKC

Softmax

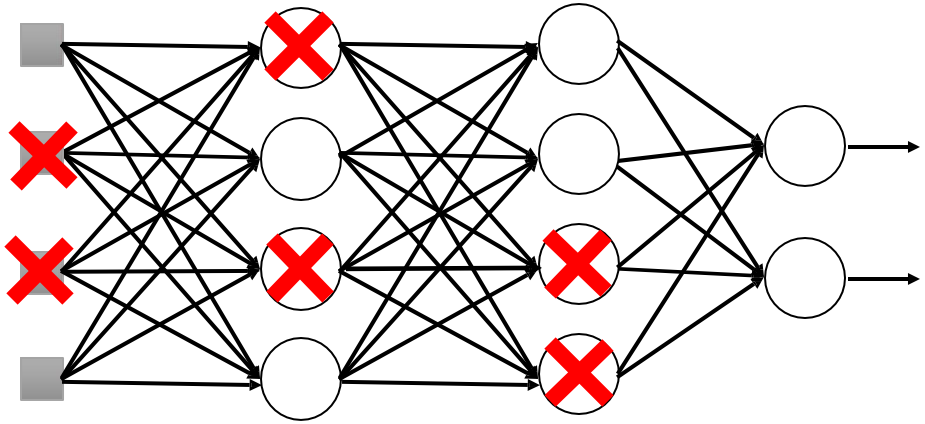


$$p_i = \frac{e^{x_i}}{e^{x_1} + e^{x_2} + e^{x_3} + e^{x_4}}$$

Regulizer: Dropout

- .

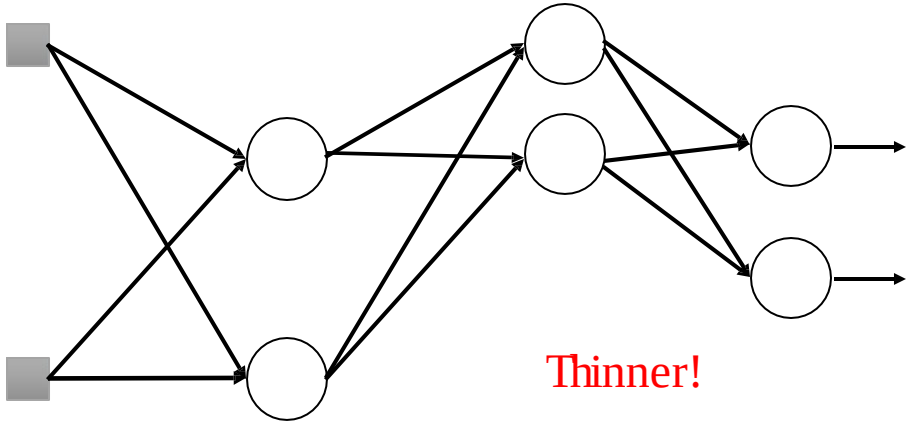
Training:



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

Dropout

Training:

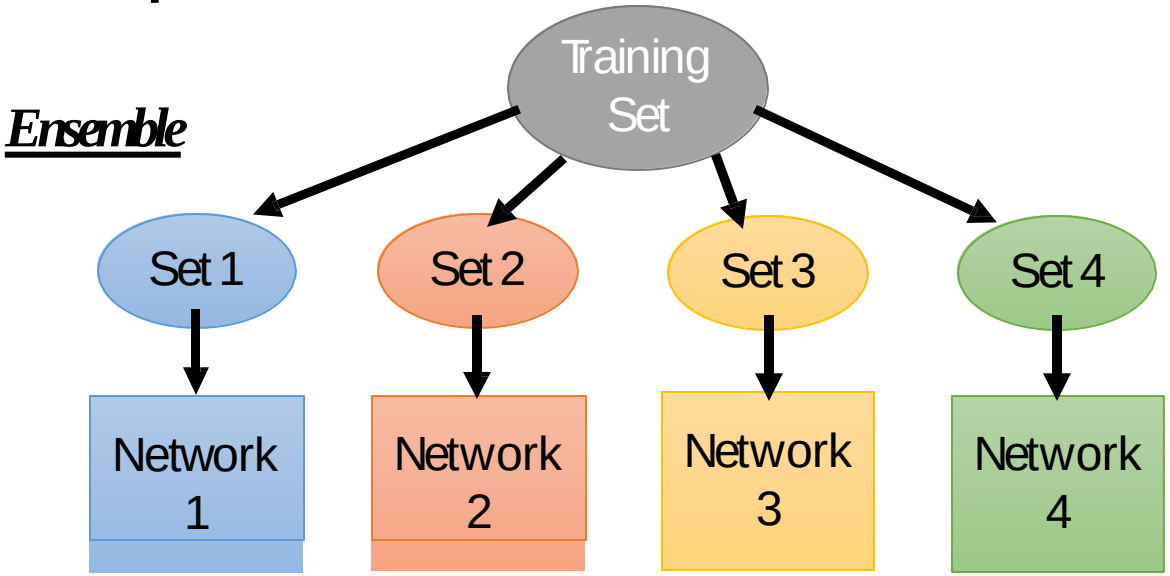


- **Each time before updating the parameters**
  - Each neuron has p% to dropout
    - **The structure of the network is changed.**
  - Using the new network for training

For each mini-batch, we resample the dropout neurons

UMKC

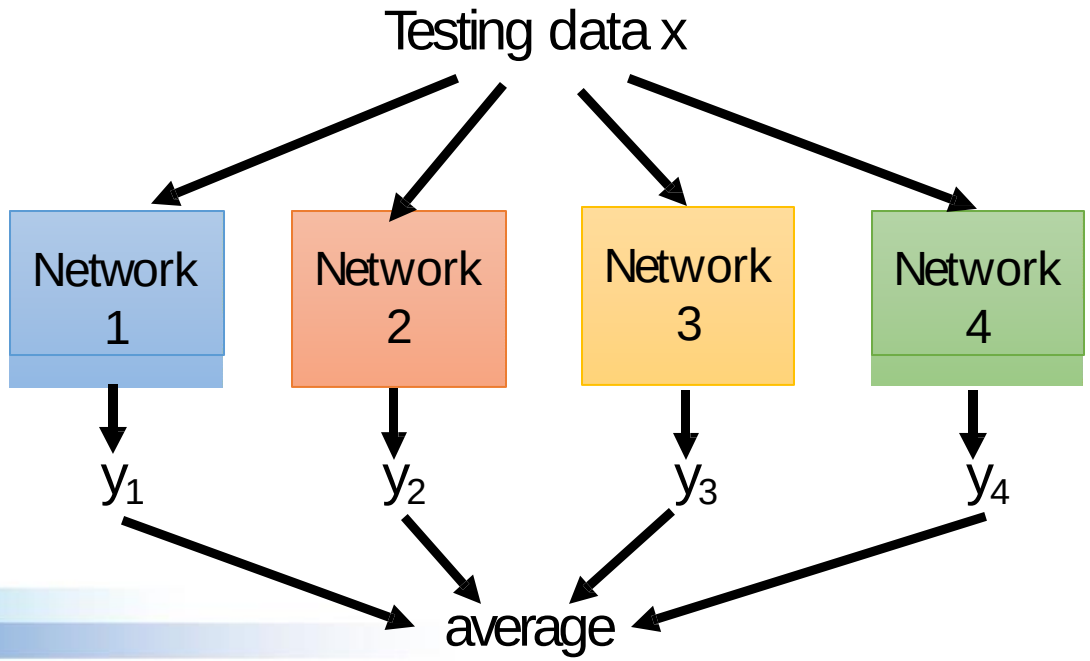
Dropout is a kind of ensemble.



Train a bunch of networks with different structures

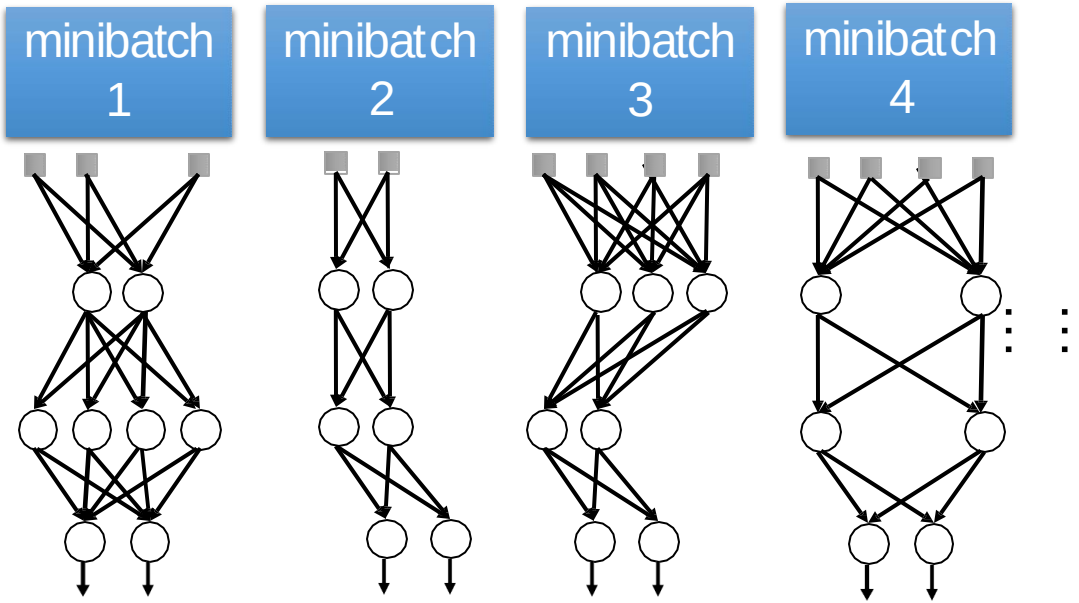
Dropout is a kind of ensemble.

Ensemble



UMKC

Dropout is a kind of ensemble.



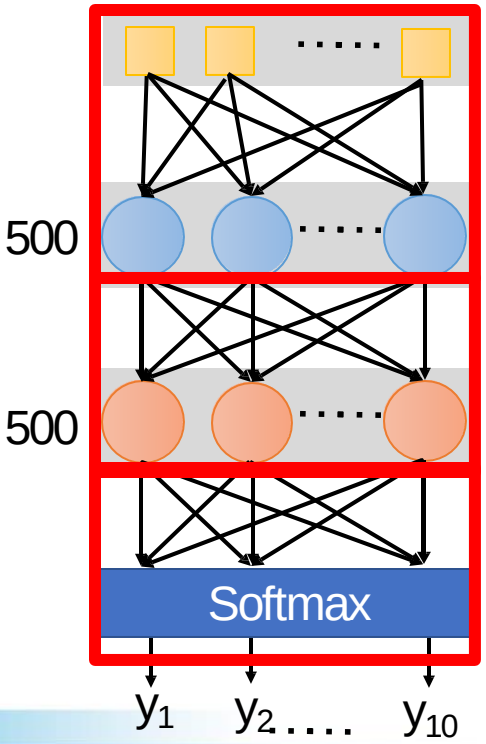
Training of Dropout

M neurons

2^M possible networks

- Using one mini-batch to train one network
- Some parameters in the network are shared

Let's try it




```
model = Sequential()
```

```
model.add( Dense( input_dim=28*28,
                  output_dim=500 ))
model.add( Activation('sigmoid') )
```

model.add(dropout(0.8))

```
model.add( Dense( output_dim=500 ) )
model.add( Activation('sigmoid') )
```

model.add(dropout(0.8))

```
model.add( Dense(output_dim=10 ) )
model.add( Activation('softmax') )
```

UMKC

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

Tensorboard

- Tensorflow, the deep learning framework from Google comes with a great tool to debug
- It hosts a website on your local machine in which you can monitor things like accuracy, cost functions
- and visualize the computational graph that Tensorflow is running based on what you defined in Keras
- Pip install tensorboard

Tensorboard

```
tensorboard = TensorBoard(log_dir="logs/{}".format(time())) model.fit(x_train, y_train, verbose=1, callbacks=[tensorboard])
```

- You need to create a new TensorBoard instance and point it to a log directory where data should be collected.
- Next you need to modify the fit call so that it includes the tensorboard callback.

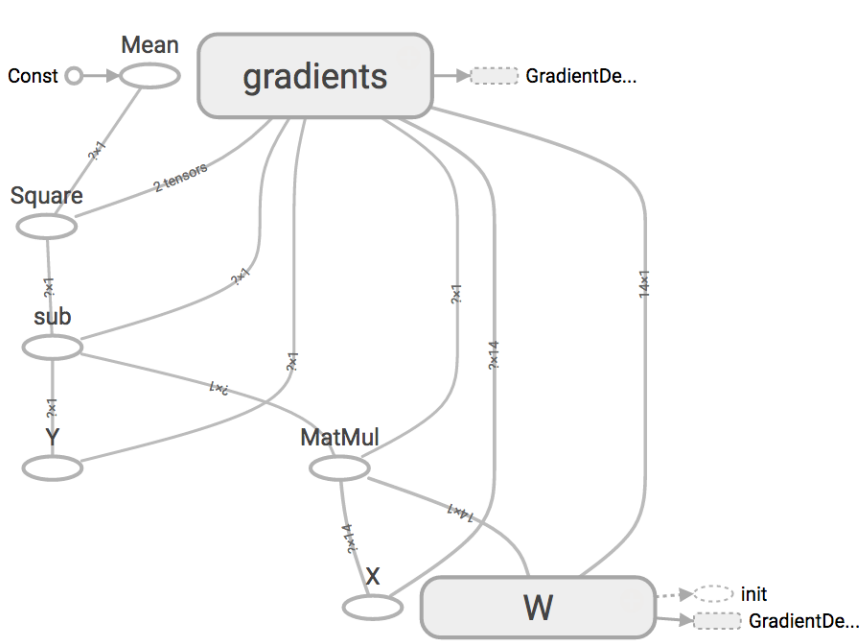
Then the following command in the separate terminal

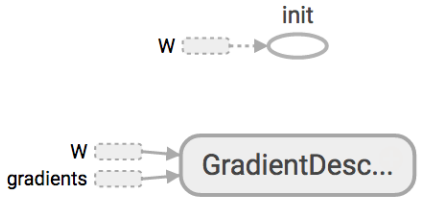
```
tensorboard --logdir=logs/
```

http://localhost:6006/

UMKC

Tensorboard





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

UMKC