Import necessary dependencies and settings

In [1]:

```
import pandas as pd
import numpy as np
import re
import nltk
import matplotlib.pyplot as plt

pd.options.display.max_colwidth = 200
%matplotlib inline
```

Sample corpus of text documents

In [2]:

```
corpus = ['The sky is blue and beautiful.',
          'Love this blue and beautiful sky!',
          'The quick brown fox jumps over the lazy dog.',
          "A king's breakfast has sausages, ham, bacon, eggs, toast and beans
          'I love green eggs, ham, sausages and bacon!',
          'The brown fox is quick and the blue dog is lazy!',
          'The sky is very blue and the sky is very beautiful today',
          'The dog is lazy but the brown fox is quick!'
]
labels = ['weather', 'weather', 'animals', 'food', 'food', 'animals', 'weathe

corpus = np.array(corpus)
corpus_df = pd.DataFrame({'Document': corpus,
                          'Category': labels})
corpus_df = corpus_df[['Document', 'Category']]
corpus_df
```

Out[2]:

	Document	Category
0	The sky is blue and beautiful.	weather
1	Love this blue and beautiful sky!	weather
2	The quick brown fox jumps over the lazy dog.	animals
3	A king's breakfast has sausages, ham, bacon, eggs, toast and beans	food
4	I love green eggs, ham, sausages and bacon!	food
5	The brown fox is quick and the blue dog is lazy!	animals
6	The sky is very blue and the sky is very beautiful today	weather
7	The dog is lazy but the brown fox is quick!	animals

Simple text pre-processing

In [3]:
```
wpt = nltk.WordPunctTokenizer()
stop_words = nltk.corpus.stopwords.words('english')

def normalize_document(doc):
    # lower case and remove special characters\whitespaces
    doc = re.sub(r'[^a-zA-Z\s]', '', doc, re.I|re.A)
    doc = doc.lower()
    doc = doc.strip()
    # tokenize document
    tokens = wpt.tokenize(doc)
    # filter stopwords out of document
    filtered_tokens = [token for token in tokens if token not in stop_words]
    # re-create document from filtered tokens
    doc = ' '.join(filtered_tokens)
    return doc

normalize_corpus = np.vectorize(normalize_document)
```

In [4]:
```
norm_corpus = normalize_corpus(corpus)
norm_corpus
```

Out[4]:
```
array(['sky blue beautiful', 'love blue beautiful sky',
       'quick brown fox jumps lazy dog',
       'kings breakfast sausages ham bacon eggs toast beans',
       'love green eggs ham sausages bacon',
       'brown fox quick blue dog lazy', 'sky blue sky beautiful today',
       'dog lazy brown fox quick'], dtype='<U51')
```

Word Embeddings

Load up sample corpus - Bible

In [5]:
```
from nltk.corpus import gutenberg
from string import punctuation

bible = gutenberg.sents('bible-kjv.txt')
remove_terms = punctuation + '0123456789'

norm_bible = [[word.lower() for word in sent if word not in remove_terms] for
norm_bible = [' '.join(tok_sent) for tok_sent in norm_bible]
norm_bible = filter(None, normalize_corpus(norm_bible))
norm_bible = [tok_sent for tok_sent in norm_bible if len(tok_sent.split()) >

print('Total lines:', len(bible))
print('\nSample line:', bible[10])
print('\nProcessed line:', norm_bible[10])
```

```
Total lines: 30103

Sample line: ['1', ':', '6', 'And', 'God', 'said', ',', 'Let', 'there', 'b
e', 'a', 'firmament', 'in', 'the', 'midst', 'of', 'the', 'waters', ',', 'an
d', 'let', 'it', 'divide', 'the', 'waters', 'from', 'the', 'waters', '.']

Processed line: god said let firmament midst waters let divide waters water
s
```

Implementing a word2vec model using a CBOW (Continuous Bag of Words) neural network architecture

Build Vocabulary

In [6]:

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

tokenizer = text.Tokenizer()
tokenizer.fit_on_texts(norm_bible)
word2id = tokenizer.word_index

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 norm

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

print('Vocabulary Size:', vocab_size)
print('Vocabulary Sample:', list(word2id.items())[:10])
```

```
Using TensorFlow backend.

Vocabulary Size: 12425
Vocabulary Sample: [('shall', 1), ('unto', 2), ('lord', 3), ('thou', 4),
('thy', 5), ('god', 6), ('ye', 7), ('said', 8), ('thee', 9), ('upon', 10)]
```

Build (context_words, target_word) pair generator

In [7]:

```
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)
```

In [8]:

```
i = 0
for x, y in generate_context_word_pairs(corpus=wids, window_size=window_size,
    if 0 not in x[0]:
        print('Context (X):', [id2word[w] for w in x[0]], '-> Target (Y):', i

        if i == 10:
            break
        i += 1
```

```
Context (X): ['old', 'testament', 'james', 'bible'] -> Target (Y): king
Context (X): ['first', 'book', 'called', 'genesis'] -> Target (Y): moses
Context (X): ['beginning', 'god', 'heaven', 'earth'] -> Target (Y): created
Context (X): ['earth', 'without', 'void', 'darkness'] -> Target (Y): form
Context (X): ['without', 'form', 'darkness', 'upon'] -> Target (Y): void
Context (X): ['form', 'void', 'upon', 'face'] -> Target (Y): darkness
Context (X): ['void', 'darkness', 'face', 'deep'] -> Target (Y): upon
Context (X): ['spirit', 'god', 'upon', 'face'] -> Target (Y): moved
Context (X): ['god', 'moved', 'face', 'waters'] -> Target (Y): upon
Context (X): ['god', 'said', 'light', 'light'] -> Target (Y): let
Context (X): ['god', 'saw', 'good', 'god'] -> Target (Y): light
```

Build CBOW Deep Network Model

In [10]:

```
import keras.backend as K
from keras.models import Sequential
from keras.layers import Dense, Embedding, Lambda

cbow = Sequential()
cbow.add(Embedding(input_dim=vocab_size, output_dim=embed_size, input_length=
cbow.add(Lambda(lambda x: K.mean(x, axis=1), output_shape=(embed_size,)))
cbow.add(Dense(vocab_size, activation='softmax'))

cbow.compile(loss='categorical_crossentropy', optimizer='rmsprop')
print(cbow.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 4, 100)            1242500
_________________________________________________________________
lambda_2 (Lambda)            (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 12425)             1254925
=================================================================
Total params: 2,497,425
Trainable params: 2,497,425
Non-trainable params: 0
_________________________________________________________________
None
```

In [ ]:
```
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot

SVG(model_to_dot(cbow, show_shapes=True, show_layer_names=False,
                 rankdir='TB').create(prog='dot', format='svg'))
```

Train model for 5 epochs

In [11]:
```
for epoch in range(1, 6):
    loss = 0.
    i = 0
    for x, y in generate_context_word_pairs(corpus=wids, window_size=window_s
        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()
```

```
WARNING:tensorflow:From c:\users\finan\appdata\local\programs\python\python
37\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a futu
re version.
Instructions for updating:
Use tf.cast instead.
Processed 100000 (context, word) pairs
Processed 200000 (context, word) pairs
Processed 300000 (context, word) pairs
Epoch: 1        Loss: 4040066.8243356827

Processed 100000 (context, word) pairs
Processed 200000 (context, word) pairs
Processed 300000 (context, word) pairs
Epoch: 2        Loss: 4252849.414020351

Processed 100000 (context, word) pairs
Processed 200000 (context, word) pairs
Processed 300000 (context, word) pairs
Epoch: 3        Loss: 4221319.320678599

Processed 100000 (context, word) pairs
Processed 200000 (context, word) pairs
Processed 300000 (context, word) pairs
Epoch: 4        Loss: 4374675.963517152

Processed 100000 (context, word) pairs
Processed 200000 (context, word) pairs
Processed 300000 (context, word) pairs
Epoch: 5        Loss: 4467728.309007732
```

Get word embeddings

In [12]:
```
weights = cbow.get_weights()[0]
weights = weights[1:]
print(weights.shape)

pd.DataFrame(weights, index=list(id2word.values())[1:]).head()
```

```
(12424, 100)
```

Out[12]:

	0	1	2	3	4	5	6	7	
unto	-1.558268	0.480393	0.007157	-1.124351	0.349770	-0.177132	0.429398	0.510010	-0
lord	-2.782228	0.594669	-0.572333	-1.998984	-0.467158	1.464037	-0.388773	0.573977	-1
thou	-2.037416	-0.173713	1.115885	-1.430719	-0.205978	1.162578	0.658434	-0.623687	-0
thy	-1.139119	0.730314	-0.308474	-1.363923	0.689318	0.746845	0.102472	-0.284844	-0
god	-1.642045	-0.098926	0.760895	-0.941579	0.043936	-0.016048	1.278149	-0.154688	-0

5 rows × 100 columns

Build a distance matrix to view the most similar words (contextually)

In [13]:
```
from sklearn.metrics.pairwise import euclidean_distances

# compute pairwise distance matrix
distance_matrix = euclidean_distances(weights)
print(distance_matrix.shape)

# view contextually similar words
similar_words = {search_term: [id2word[idx] for idx in distance_matrix[word2i
                   for search_term in ['god', 'jesus', 'noah', 'egypt', 'john

similar_words
```

```
(12424, 12424)
```

Out[13]:
```
{'god': ['things', 'ye', 'also', 'might', 'therefore'],
 'jesus': ['faith', 'world', 'law', 'glory', 'spirit'],
 'noah': ['adam', 'barnabas', 'lamech', 'shem', 'cain'],
 'egypt': ['pharaoh', 'camp', 'led', 'money', 'darkness'],
 'john': ['galilee', 'peter', 'angels', 'ship', 'violence'],
 'gospel': ['preached', 'crucified', 'apostles', 'preach', 'powers'],
 'moses': ['aaron', 'pharaoh', 'rose', 'died', 'throughout'],
 'famine': ['pestilence', 'portion', 'shadow', 'chaldeans', 'possession']}
```

Implementing a word2vec model using a skip-gram neural network architecture

Build Vocabulary

In [14]:

```
from keras.preprocessing import text

tokenizer = text.Tokenizer()
tokenizer.fit_on_texts(norm_bible)

word2id = tokenizer.word_index
id2word = {v:k for k, v in word2id.items()}

vocab_size = len(word2id) + 1
embed_size = 100

wids = [[word2id[w] for w in text.text_to_word_sequence(doc)] for doc in norm
print('Vocabulary Size:', vocab_size)
print('Vocabulary Sample:', list(word2id.items())[:10])
```

```
Vocabulary Size: 12425
Vocabulary Sample: [('shall', 1), ('unto', 2), ('lord', 3), ('thou', 4),
('thy', 5), ('god', 6), ('ye', 7), ('said', 8), ('thee', 9), ('upon', 10)]
```

Build and View sample skip grams ((word1, word2) -> relevancy)

In [15]:

```
from keras.preprocessing.sequence import skipgrams

# generate skip-grams
skip_grams = [skipgrams(wid, vocabulary_size=vocab_size, window_size=10) for

# view sample skip-grams
pairs, labels = skip_grams[0][0], skip_grams[0][1]
for i in range(10):
    print("({:s} ({:d}), {:s} ({:d})) -> {:d}".format(
          id2word[pairs[i][0]], pairs[i][0],
          id2word[pairs[i][1]], pairs[i][1],
          labels[i]))
```

```
(bible (5766), king (13)) -> 1
(bible (5766), james (1154)) -> 1
(king (13), james (1154)) -> 1
(bible (5766), knowledge (374)) -> 0
(james (1154), railings (12195)) -> 0
(king (13), machnadebai (10146)) -> 0
(king (13), bible (5766)) -> 1
(james (1154), bible (5766)) -> 1
(bible (5766), log (4765)) -> 0
(james (1154), mine (84)) -> 0
```

Build Skip-gram Deep Network Model

In [ ]:
```
from keras.layers import dot
from keras.layers.core import Dense, Reshape
from keras.layers.embeddings import Embedding
from keras.models import Sequential

word_model = Sequential()
word_model.add(Embedding(vocab_size, embed_size,
                         embeddings_initializer="glorot_uniform",
                         input_length=1))
word_model.add(Reshape((embed_size, )))

context_model = Sequential()
context_model.add(Embedding(vocab_size, embed_size,
                  embeddings_initializer="glorot_uniform",
                  input_length=1))
context_model.add(Reshape((embed_size,)))

model = Sequential()
model.add(dot([word_model, context_model], axes=1,normalize=False))
model.add(Dense(1, kernel_initializer="glorot_uniform", activation="sigmoid")
model.compile(loss="mean_squared_error", optimizer="rmsprop")
print(model.summary())
```

In [ ]:
```
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot

SVG(model_to_dot(model, show_shapes=True, show_layer_names=False,
                 rankdir='TB').create(prog='dot', format='svg'))
```

Train the model for 5 epochs

In [ ]:
```
for epoch in range(1, 6):
    loss = 0
    for i, elem in enumerate(skip_grams):
        pair_first_elem = np.array(list(zip(*elem[0]))[0], dtype='int32')
        pair_second_elem = np.array(list(zip(*elem[0]))[1], dtype='int32')
        labels = np.array(elem[1], dtype='int32')
        X = [pair_first_elem, pair_second_elem]
        Y = labels
        if i % 10000 == 0:
            print('Processed {} (skip_first, skip_second, relevance) pairs'.f
        loss += model.train_on_batch(X,Y)

    print('Epoch:', epoch, 'Loss:', loss)
```

Get word embeddings

In [ ]:

```
merge_layer = model.layers[0]
word_model = merge_layer.layers[0]
word_embed_layer = word_model.layers[0]
weights = word_embed_layer.get_weights()[0][1:]

print(weights.shape)
pd.DataFrame(weights, index=id2word.values()).head()
```

Build a distance matrix to view the most similar words (contextually)

In [34]:

```
from sklearn.metrics.pairwise import euclidean_distances

distance_matrix = euclidean_distances(weights)
print(distance_matrix.shape)

similar_words = {search_term: [id2word[idx] for idx in distance_matrix[word2i
                   for search_term in ['god', 'jesus', 'noah', 'egypt', 'johr

similar_words
```

```
(12424, 12424)
```

Out[34]:

```
{'god': ['things', 'ye', 'also', 'might', 'therefore'],
 'jesus': ['faith', 'world', 'law', 'glory', 'spirit'],
 'noah': ['adam', 'barnabas', 'lamech', 'shem', 'cain'],
 'egypt': ['pharaoh', 'camp', 'led', 'money', 'darkness'],
 'john': ['galilee', 'peter', 'angels', 'ship', 'violence'],
 'gospel': ['preached', 'crucified', 'apostles', 'preach', 'powers'],
 'moses': ['aaron', 'pharaoh', 'rose', 'died', 'throughout'],
 'famine': ['pestilence', 'portion', 'shadow', 'chaldeans', 'possession']}
```

Visualize word embeddings

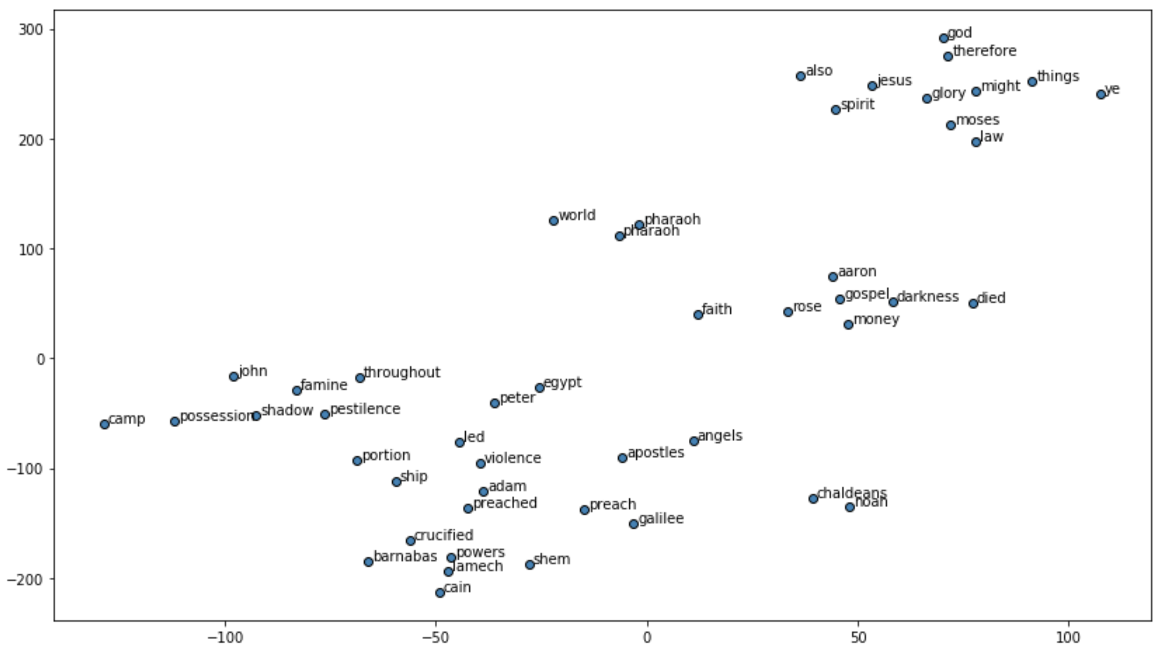
In [35]:
```
from sklearn.manifold import TSNE

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.sh

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=(14, 8))
plt.scatter(T[:, 0], T[:, 1], c='steelblue', edgecolors='k')
for label, x, y in zip(labels, T[:, 0], T[:, 1]):
    plt.annotate(label, xy=(x+1, y+1), xytext=(0, 0), textcoords='offset poi
```

```
Total words: 48         Word Embedding shapes: (48, 100)
```



Leveraging gensim for building a word2vec model

In [48]:

```
from gensim.models import word2vec

# tokenize sentences in corpus
wpt = nltk.WordPunctTokenizer()
tokenized_corpus = [wpt.tokenize(document) for document in norm_bible]

# Set values for various parameters
feature_size = 100    # Word vector dimensionality
window_context = 30          # Context window size
min_word_count = 1   # Minimum word count
sample = 1e-3   # Downsample setting for frequent words

w2v_model = word2vec.Word2Vec(tokenized_corpus, size=feature_size,
                          window=window_context, min_count=min_word_count,
                          sample=sample, iter=50)

# view similar words based on gensim's model
similar_words = {search_term: [item[0] for item in w2v_model.wv.most_similar(
                  for search_term in ['god', 'jesus', 'noah', 'egypt', 'john'
similar_words
```

Out[48]:

```
{'god': ['lord', 'worldly', 'saving', 'ever', 'us'],
 'jesus': ['peter', 'messias', 'immediately', 'apelles', 'impotent'],
 'noah': ['shem', 'ham', 'japheth', 'kenan', 'enosh'],
 'egypt': ['egyptians', 'pharaoh', 'bondage', 'flowing', 'rod'],
 'john': ['baptist', 'james', 'peter', 'galilee', 'devine'],
 'gospel': ['christ', 'repentance', 'faith', 'preach', 'afflictions'],
 'moses': ['congregation', 'children', 'joshua', 'aaron', 'doctor'],
 'famine': ['pestilence', 'peril', 'sword', 'deaths', 'blasting']}
```

Visualizing word embeddings

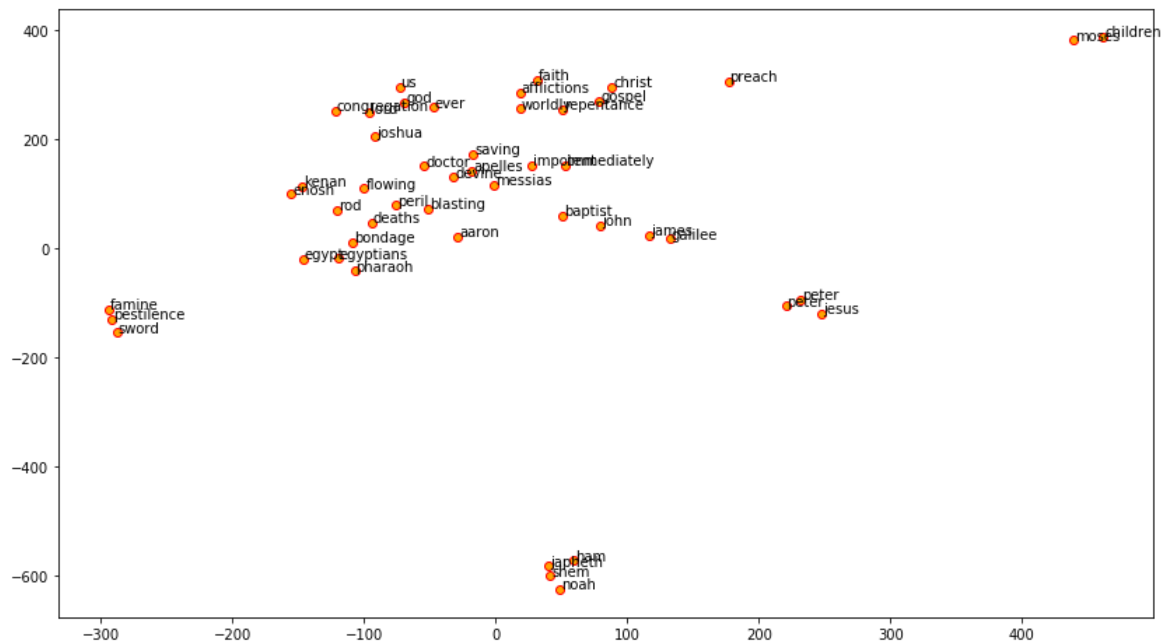
In [49]:

```
from sklearn.manifold import TSNE

words = sum([[k] + v for k, v in similar_words.items()], [])
wvs = w2v_model.wv[words]

tsne = TSNE(n_components=2, random_state=0, n_iter=10000, perplexity=2)
np.set_printoptions(suppress=True)
T = tsne.fit_transform(wvs)
labels = words

plt.figure(figsize=(14, 8))
plt.scatter(T[:, 0], T[:, 1], c='orange', edgecolors='r')
for label, x, y in zip(labels, T[:, 0], T[:, 1]):
    plt.annotate(label, xy=(x+1, y+1), xytext=(0, 0), textcoords='offset poir
```



Applying the word2vec model on our sample corpus

In [50]:

```
wpt = nltk.WordPunctTokenizer()
tokenized_corpus = [wpt.tokenize(document) for document in norm_corpus]

# Set values for various parameters
feature_size = 10    # Word vector dimensionality
window_context = 10          # Context window size
min_word_count = 1   # Minimum word count
sample = 1e-3   # Downsample setting for frequent words

w2v_model = word2vec.Word2Vec(tokenized_corpus, size=feature_size,
                              window=window_context, min_count = min_word_cou
                              sample=sample, iter=100)
```

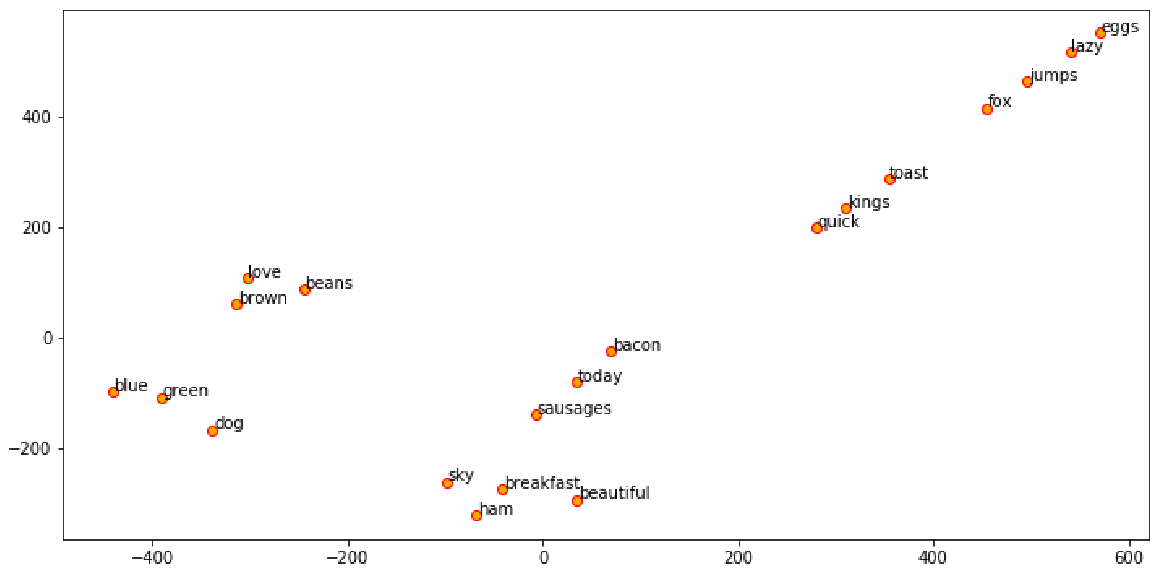
Visualize word embeddings

In [51]:
```
from sklearn.manifold import TSNE

words = w2v_model.wv.index2word
wvs = w2v_model.wv[words]

tsne = TSNE(n_components=2, random_state=0, n_iter=5000, perplexity=2)
np.set_printoptions(suppress=True)
T = tsne.fit_transform(wvs)
labels = words

plt.figure(figsize=(12, 6))
plt.scatter(T[:, 0], T[:, 1], c='orange', edgecolors='r')
for label, x, y in zip(labels, T[:, 0], T[:, 1]):
    plt.annotate(label, xy=(x+1, y+1), xytext=(0, 0), textcoords='offset poir
```



Sample word embedding

In [52]:
```
w2v_model.wv['sky']
```

Out[52]:
```
array([ 0.00130841, -0.03155484, -0.02961808, -0.0304446 , -0.02777019,
       -0.0281617 , -0.02667127, -0.02629152,  0.02053194, -0.03976242],
      dtype=float32)
```

Build framework for getting document level embeddings

In [53]:

```
def average_word_vectors(words, model, vocabulary, num_features):

    feature_vector = np.zeros((num_features,),dtype="float64")
    nwords = 0.

    for word in words:
        if word in vocabulary:
            nwords = nwords + 1.
            feature_vector = np.add(feature_vector, model[word])

    if nwords:
        feature_vector = np.divide(feature_vector, nwords)

    return feature_vector


def averaged_word_vectorizer(corpus, model, num_features):
    vocabulary = set(model.wv.index2word)
    features = [average_word_vectors(tokenized_sentence, model, vocabulary, r
                    for tokenized_sentence in corpus]
    return np.array(features)
```

In [54]:

```
w2v_feature_array = averaged_word_vectorizer(corpus=tokenized_corpus, model=w
                                             num_features=feature_size)
pd.DataFrame(w2v_feature_array)
```

```
c:\users\finan\appdata\local\programs\python\python37\lib\site-packages\ipy
kernel_launcher.py:9: DeprecationWarning: Call to deprecated `__getitem__`
(Method will be removed in 4.0.0, use self.wv.__getitem__() instead).
  if __name__ == '__main__':
```

Out[54]:

	0	1	2	3	4	5	6	7	
0	0.002783	-0.022939	-0.022471	0.004143	-0.029013	-0.003862	-0.010397	-0.020175	0.02
1	0.001214	-0.018318	-0.024470	-0.002171	-0.009240	-0.001448	-0.018517	-0.016627	0.00
2	-0.014307	0.000737	0.002849	0.011757	0.015777	-0.010199	0.015807	-0.016422	0.00
3	-0.002899	0.022346	0.005972	0.009551	0.014934	-0.004253	0.010546	0.001638	0.01
4	0.002709	0.014457	-0.007672	0.001225	0.008650	-0.000316	0.002582	-0.002047	-0.00
5	-0.006682	-0.001316	-0.001212	0.012979	0.014086	0.004563	0.012740	-0.014526	0.00
6	0.008368	-0.027867	-0.015179	-0.006986	-0.019475	-0.004753	-0.005497	-0.007990	0.02
7	-0.015013	-0.003183	0.008187	0.016703	0.019562	-0.002188	0.009358	-0.013956	-0.00

Clustering with word embeddings

In [55]:

```
from sklearn.cluster import AffinityPropagation

ap = AffinityPropagation()
ap.fit(w2v_feature_array)
cluster_labels = ap.labels_
cluster_labels = pd.DataFrame(cluster_labels, columns=['ClusterLabel'])
pd.concat([corpus_df, cluster_labels], axis=1)
```

Out[55]:

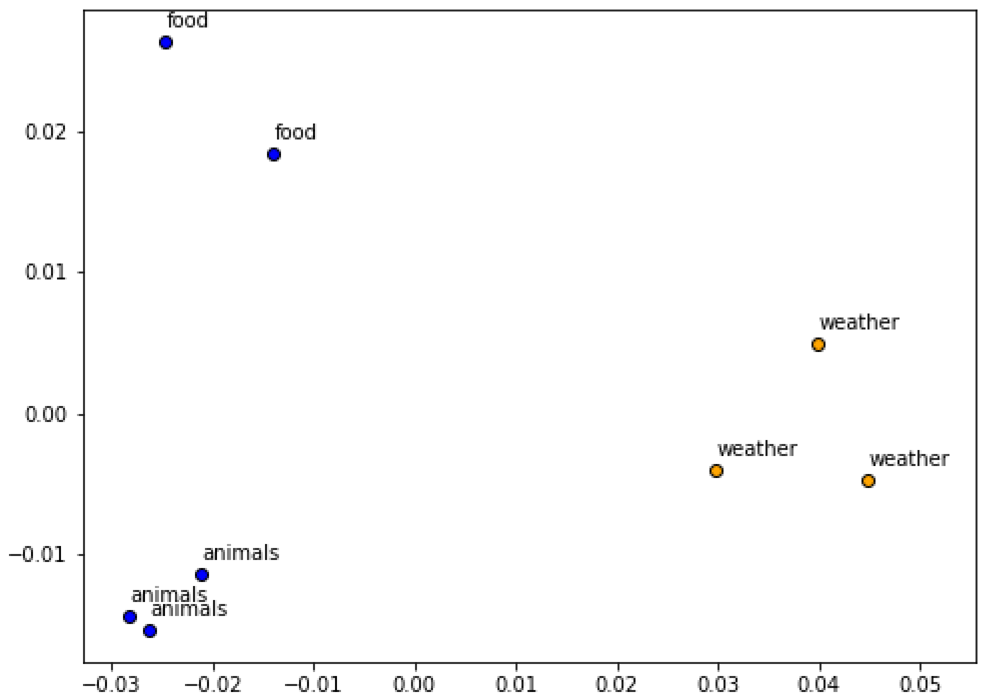
	Document	Category	ClusterLabel
0	The sky is blue and beautiful.	weather	0
1	Love this blue and beautiful sky!	weather	0
2	The quick brown fox jumps over the lazy dog.	animals	1
3	A king's breakfast has sausages, ham, bacon, eggs, toast and beans	food	1
4	I love green eggs, ham, sausages and bacon!	food	1
5	The brown fox is quick and the blue dog is lazy!	animals	1
6	The sky is very blue and the sky is very beautiful today	weather	0
7	The dog is lazy but the brown fox is quick!	animals	1

In [56]:

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2, random_state=0)
pcs = pca.fit_transform(w2v_feature_array)
labels = ap.labels_
categories = list(corpus_df['Category'])
plt.figure(figsize=(8, 6))

for i in range(len(labels)):
    label = labels[i]
    color = 'orange' if label == 0 else 'blue' if label == 1 else 'green'
    annotation_label = categories[i]
    x, y = pcs[i]
    plt.scatter(x, y, c=color, edgecolors='k')
    plt.annotate(annotation_label, xy=(x+1e-4, y+1e-3), xytext=(0, 0), textcc
```



GloVe Embeddings with spaCy

```
# Use the following command to install spaCy
> pip install -U spacy

OR

> conda install -c conda-forge spacy


# Download the following language model and store it in disk
https://github.com/explosion/spacy-models/releases/tag/en_vectors_web_lg-2.0.0


# Link the same to spacy
```

```
> python -m spacy link ./spacymodels/en_vectors_web_lg-2.0.0/en_vectors_web_lg
en_vecs

    Linking successful
    ./spacymodels/en_vectors_web_lg-2.0.0/en_vectors_web_lg -->
./Anaconda3/lib/site-packages/spacy/data/en_vecs

    You can now load the model via spacy.load('en_vecs')
```

In [ ]:

```
import spacy

#nlp = spacy.load('en_vecs')
nlp = spacy.load('en_vectors_web_lg')

total_vectors = len(nlp.vocab.vectors)
print('Total word vectors:', total_vectors)
```

Visualize GloVe word embeddings

In [ ]:

```
unique_words = list(set([word for sublist in [doc.split() for doc in norm_cor

word_glove_vectors = np.array([nlp(word).vector for word in unique_words])
pd.DataFrame(word_glove_vectors, index=unique_words)
```

In [ ]:

```
from sklearn.manifold import TSNE

tsne = TSNE(n_components=2, random_state=0, n_iter=5000, perplexity=3)
np.set_printoptions(suppress=True)
T = tsne.fit_transform(word_glove_vectors)
labels = unique_words

plt.figure(figsize=(12, 6))
plt.scatter(T[:, 0], T[:, 1], c='orange', edgecolors='r')
for label, x, y in zip(labels, T[:, 0], T[:, 1]):
    plt.annotate(label, xy=(x+1, y+1), xytext=(0, 0), textcoords='offset poir
```

Cluster documents with GloVe Embeddings

In [ ]:

```
doc_glove_vectors = np.array([nlp(str(doc)).vector for doc in norm_corpus])

km = KMeans(n_clusters=3, random_state=0)
km.fit_transform(doc_glove_vectors)
cluster_labels = km.labels_
cluster_labels = pd.DataFrame(cluster_labels, columns=['ClusterLabel'])
pd.concat([corpus_df, cluster_labels], axis=1)
```

Leveraging gensim for building a FastText model

In [66]:
```
from gensim.models.fasttext import FastText

wpt = nltk.WordPunctTokenizer()
tokenized_corpus = [wpt.tokenize(document) for document in norm_bible]

# Set values for various parameters
feature_size = 100    # Word vector dimensionality
window_context = 50          # Context window size
min_word_count = 5   # Minimum word count
sample = 1e-3   # Downsample setting for frequent words


ft_model = FastText(tokenized_corpus, size=feature_size, window=window_conte
                    min_count=min_word_count,sample=sample, sg=1, iter=50)
```

In [67]:
```
# view similar words based on gensim's model
similar_words = {search_term: [item[0] for item in ft_model.wv.most_similar([
                  for search_term in ['god', 'jesus', 'noah', 'egypt', 'john'
similar_words
```

Out[67]:
```
{'god': ['lord', 'jesus', 'therefore', 'unto', 'christ'],
 'jesus': ['christ', 'god', 'faith', 'disciples', 'grace'],
 'noah': ['methuselah', 'milcah', 'flood', 'adam', 'shem'],
 'egypt': ['land', 'pharaoh', 'egyptians', 'israel', 'shur'],
 'john': ['baptist', 'baptize', 'peter', 'baptized', 'philip'],
 'gospel': ['preached', 'preach', 'christ', 'preaching', 'grace'],
 'moses': ['aaron', 'commanded', 'congregation', 'tabernacle', 'spake'],
 'famine': ['pestilence', 'sword', 'sojourn', 'blasted', 'die']}
```

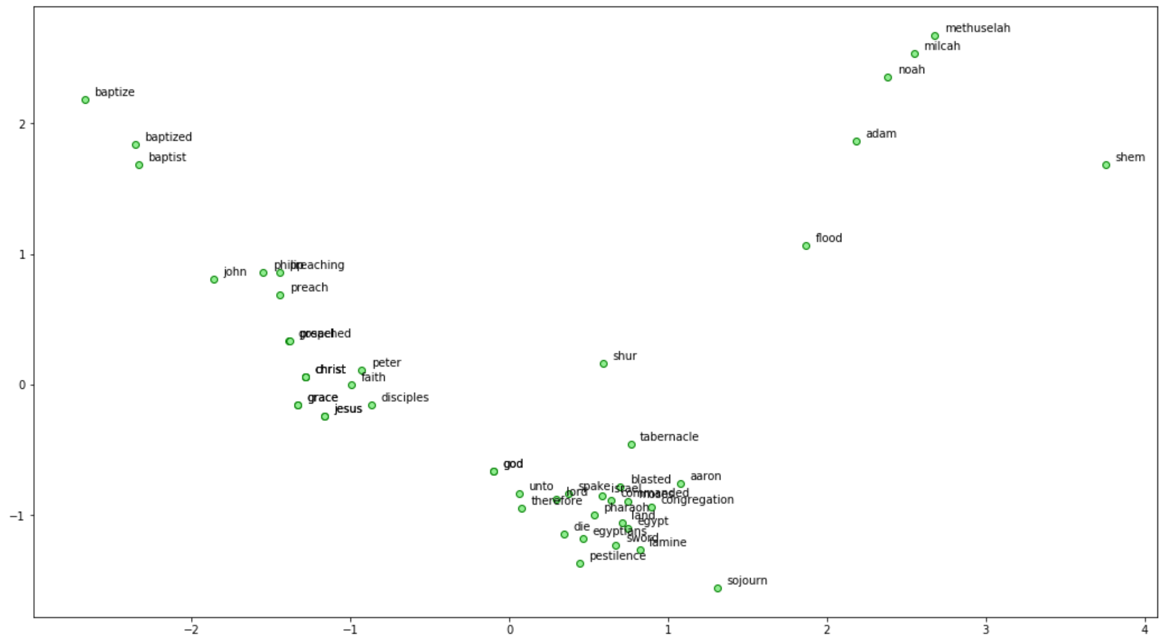
In [68]:

```
from sklearn.decomposition import PCA

words = sum([[k] + v for k, v in similar_words.items()], [])
wvs = ft_model.wv[words]

pca = PCA(n_components=2)
np.set_printoptions(suppress=True)
P = pca.fit_transform(wvs)
labels = words

plt.figure(figsize=(18, 10))
plt.scatter(P[:, 0], P[:, 1], c='lightgreen', edgecolors='g')
for label, x, y in zip(labels, P[:, 0], P[:, 1]):
    plt.annotate(label, xy=(x+0.06, y+0.03), xytext=(0, 0), textcoords='offse
```



In [69]:
```
ft_model.wv['jesus']
```

Out[69]:
```
array([ 0.13958982, -0.05803353,  0.00735883,  0.30419916,  0.28101364,
       -0.09217732, -0.35205486, -0.10779286, -0.20662811,  0.01127647,
        0.56522685, -0.25434008, -0.11027662, -0.09649172,  0.01176434,
        0.4307362 ,  0.28582764, -0.0331421 , -0.07566007, -0.0571156 ,
       -0.09087794, -0.11904453, -0.1753413 , -0.33740595,  0.10848472,
       -0.00883908, -0.04714192,  0.19704778,  0.09685205,  0.21058671,
       -0.08167323, -0.30198124,  0.31116888,  0.07161883,  0.37063324,
        0.6008908 , -0.13673909, -0.4890262 , -0.52823555,  0.28040478,
       -0.05382667,  0.47069886, -0.0484414 ,  0.61672145,  0.24615462,
       -0.43410167,  0.11218455,  0.30876148,  0.13442726, -0.21992294,
        0.11206417,  0.00722101, -0.45872113, -0.16935116,  0.29874218,
        0.19664103, -0.61964315, -0.25932878,  0.07407966,  0.2578632 ,
        0.08950499, -0.27822822,  0.32461262, -0.22445315,  0.1997942 ,
        0.27103156,  0.08593477, -0.09900978,  0.0666374 ,  0.4836858 ,
        0.08429956,  0.08149047,  0.08276018,  0.04630444,  0.21469755,
        0.03319091, -0.10793024, -0.26813093, -0.04301734,  0.09764177,
       -0.22126143,  0.1392126 , -0.41733798,  0.28070012, -0.19648044,
        0.14717151, -0.21817796,  0.40246776, -0.02561598, -0.3541444 ,
        0.05279887,  0.27027574, -0.18075562,  0.00881244,  0.04249534,
        0.05854722,  0.45185554,  0.3729318 , -0.06063852, -0.02820194],
      dtype=float32)
```

In [70]:
```
print(ft_model.wv.similarity(w1='god', w2='satan'))
print(ft_model.wv.similarity(w1='god', w2='jesus'))
```

```
0.33519396
0.70369
```

In [71]:
```
st1 = "god jesus satan john"
print('Odd one out for [',st1, ']:', ft_model.wv.doesnt_match(st1.split()))

st2 = "john peter james judas"
print('Odd one out for [',st2, ']:', ft_model.wv.doesnt_match(st2.split()))
```

```
c:\users\finan\appdata\local\programs\python\python37\lib\site-packages\gen
sim\models\keyedvectors.py:858: FutureWarning: arrays to stack must be pass
ed as a "sequence" type such as list or tuple. Support for non-sequence ite
rables such as generators is deprecated as of NumPy 1.16 and will raise an
error in the future.
  vectors = vstack(self.word_vec(word, use_norm=True) for word in used_word
s).astype(REAL)

Odd one out for [ god jesus satan john ]: satan
Odd one out for [ john peter james judas ]: judas
```