1. Get the dataset

In [1]:
```python
import pandas as pd
df = pd.read_csv('papers.csv')
df.head()
```

Out[1]:

	id	year	title	event_type	pdf_name	abstract	paper_text
0	1	1987	Self-Organization of Associative Database and ...	NaN	1-self-organization-of-associative-database-an...	Abstract Missing	767\n\nSELF-ORGANIZATION OF ASSOCIATIVE DATABA...
1	10	1987	A Mean Field Theory of Layer IV of Visual Cort...	NaN	10-a-mean-field-theory-of-layer-iv-of-visual-c...	Abstract Missing	683\n\nA MEAN FIELD THEORY OF LAYER IV OF VISU...
2	100	1988	Storing Covariance by the Associative Long-Ter...	NaN	100-storing-covariance-by-the-associative-long...	Abstract Missing	394\n\nSTORING COVARIANCE BY THE ASSOCIATIVE\n...
3	1000	1994	Bayesian Query Construction for Neural Network...	NaN	1000-bayesian-query-construction-for-neural-ne...	Abstract Missing	Bayesian Query Construction for Neural\nNetwor...
4	1001	1994	Neural Network Ensembles, Cross Validation, an...	NaN	1001-neural-network-ensembles-cross-validation...	Abstract Missing	Neural Network Ensembles, Cross\nValidation, a...

In [2]:
```python
df.tail()
```

Out[2]:

	id	year	title	event_type	pdf_name	abstract	paper_text
7236	994	1994	Single Transistor Learning Synapses	NaN	994-single-transistor-learning-synapses.pdf	Abstract Missing	Single Transistor Learning Synapses\n\nPaul Ha...
7237	996	1994	Bias, Variance and the Combination of Least Sq...	NaN	996-bias-variance-and-the-combination-of-least...	Abstract Missing	Bias, Variance and the Combination of\nLeast S...
7238	997	1994	A Real Time Clustering CMOS Neural Engine	NaN	997-a-real-time-clustering-cmos-neural-engine.pdf	Abstract Missing	A Real Time Clustering CMOS\nNeural Engine\nT....
7239	998	1994	Learning direction in global motion: two class...	NaN	998-learning-direction-in-global-motion-two-cl...	Abstract Missing	Learning direction in global motion: two\nclas...
7240	999	1994	Correlation and Interpolation Networks for Rea...	NaN	999-correlation-and-interpolation-networks-for...	Abstract Missing	Correlation and Interpolation Networks for\nRe...

In [3]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7241 entries, 0 to 7240
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   id          7241 non-null   int64
 1   year        7241 non-null   int64
 2   title       7241 non-null   object
 3   event_type  2422 non-null   object
 4   pdf_name    7241 non-null   object
 5   abstract    7241 non-null   object
 6   paper_text  7241 non-null   object
dtypes: int64(2), object(5)
memory usage: 396.1+ KB
```

In [4]:
```python
df.shape
```

Out[4]:
```
(7241, 7)
```

In [5]:
```python
df.describe()
```

Out[5]:

	id	year
count	7241.000000	7241.000000
mean	3655.912167	2006.439718
std	2098.435219	8.759919
min	1.000000	1987.000000
25%	1849.000000	2000.000000
50%	3659.000000	2009.000000
75%	5473.000000	2014.000000
max	7284.000000	2017.000000

2. Dropping the unnecessary columns

In [6]:
```python
df.drop(['title', 'event_type', 'pdf_name', 'paper_text'], axis = 1, inp
lace = True)
df.head()
```

Out[6]:

	id	year	abstract
0	1	1987	Abstract Missing
1	10	1987	Abstract Missing
2	100	1988	Abstract Missing
3	1000	1994	Abstract Missing
4	1001	1994	Abstract Missing

3. Dropping the rows where Abstract is missing

In [7]:
```python
dataset = df[df.abstract != 'Abstract Missing']
dataset.head()
```

Out[7]:

	id	year	abstract
941	1861	2000	Non-negative matrix factorization (NMF) has pr...
1067	1975	2001	Spike-triggered averaging techniques are effec...
2384	3163	2007	It is known that determinining whether a DEC-P...
2385	3164	2007	We present the first truly polynomial algorith...
2388	3167	2007	Semi-supervised inductive learning concerns ho...

In [8]:
```python
dataset.tail()
```

Out[8]:

	id	year	abstract
6943	7280	2017	We revisit the classical analysis of generativ...
6944	7281	2017	PAC maximum ...
6945	7282	2017	Community detection, which focuses on clusteri...
6946	7283	2017	We propose a general framework for interactive...
6947	7284	2017	We consider maximum likelihood estimation of l...

In [9]:
```python
dataset.shape
```

Out[9]:
```
(3924, 3)
```

4. Getting the word count in each abstract

In [10]:
```python
dataset['word_count'] = dataset['abstract'].apply(lambda x: len(str(x).s
plit(" ")))
dataset[['abstract','word_count']].head()
```

```
C:\Users\kruti\anaconda3\lib\site-packages\ipykernel_launcher.py:1: Sett
ingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-d
ocs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
```

Out[10]:

	abstract	word_count
941	Non-negative matrix factorization (NMF) has pr...	107
1067	Spike-triggered averaging techniques are effec...	81
2384	It is known that determinining whether a DEC-P...	67
2385	We present the first truly polynomial algorith...	143
2388	Semi-supervised inductive learning concerns ho...	119

In [11]:
```python
dataset.word_count.describe()
```

Out[11]:
```
count    3924.000000
mean      148.390928
std        45.605755
min        19.000000
25%       116.000000
50%       143.000000
75%       177.000000
max       317.000000
Name: word_count, dtype: float64
```

5. Finding the most common words

In [12]:
```python
frequency = pd.Series(' '.join(dataset['abstract']).split()).value_count
s()[:20]
frequency
```

Out[12]:
```
the         29793
of          20918
a           16339
and         13626
to          12869
in           8980
that         7838
is           7666
for          7169
We           6238
on           5579
we           5167
with         4512
this         3677
as           3643
are          3529
an           3366
by           3197
can          2953
learning     2825
dtype: int64
```

6. Finding the most uncommon words

In [13]:
```python
frequency1 = pd.Series(' '.join(dataset['abstract']).split()).value_coun
ts()[-20:]
frequency1
```

Out[13]:
```
Meep,                  1
bound-based            1
(GCG)-type             1
communities/blocks.    1
(STA),                 1
non-metric,            1
justification:         1
non-additive           1
registered,            1
\textit{et             1
bels.                  1
LSVMs                  1
l                      1
confidence-set         1
``correct"             1
Gaussian)              1
KBP.                   1
STRADS,                1
$\E[Y]$                1
(AG)                   1
dtype: int64
```

7. Text Preprocessing

7. 1. Importing the necessary libraries for text preprocessing

In [14]:
```python
import re
import nltk
#nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.tokenize import RegexpTokenizer
#nltk.download('wordnet')
from nltk.stem.wordnet import WordNetLemmatizer
```

7. 2. Creating a list of 'Custom' stop words

In [15]:
```python
##Creating a list of stop words and adding custom stopwords
stop_words = set(stopwords.words("english"))
##Creating a list of custom stopwords
new_words = ["using", "show", "result", "large", "also", "iv", "one", "t
wo", "new", "previously", "shown"]
stop_words = stop_words.union(new_words)
```

7.3. Cleaning the text

In [18]:
```python
corpus = []
for i in range(0, 3924):
    #Remove punctuations
    text = dataset.iloc[i, 2].replace(r'[^\w\s]', ' ')

    #Convert to lowercase
    text = text.lower()

    #remove tags
    text=re.sub("&lt;/?.*?&gt;"," &lt;&gt; ",text)

    # remove special characters and digits
    text=re.sub("(\\d|\\W)+"," ",text)

    ##Convert to list from string
    text = text.split()

    ##Stemming
    ps=PorterStemmer()
    #Lemmatisation
    lem = WordNetLemmatizer()
    text = [lem.lemmatize(word) for word in text if not word in
            stop_words]
    text = " ".join(text)
    corpus.append(text)
```

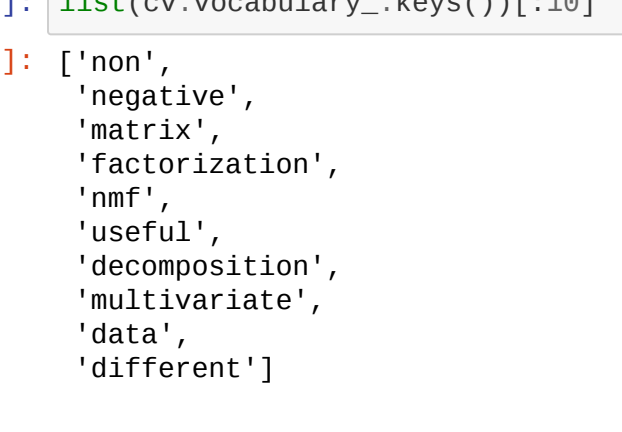
In [19]:
```python
#View corpus item
corpus[222]
```

Out[19]:
```
'actor critic algorithm reinforcement learning achieving renewed popular
ity due good convergence property situation approach often fail e g func
tion approximation involved interestingly growing evidence actor critic
approach based phasic dopamine signal play key role biological learning
cortical basal ganglion derive temporal difference based actor critic le
arning algorithm convergence proved without assuming separate time scale
actor critic approach demonstrated applying network spiking neuron estab
lished relation phasic dopamine temporal difference signal lends support
biological relevance algorithm'
```

8. Word Cloud

In [21]:
```python
#Word cloud
from os import path
from PIL import Image
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
import matplotlib.pyplot as plt
%matplotlib inline
wordcloud = WordCloud(
                          background_color='white',
                          stopwords=stop_words,
                          max_words=100,
                          max_font_size=50,
                          random_state=42
                         ).generate(str(corpus))
print(wordcloud)
fig = plt.figure(1)
plt.imshow(wordcloud)
plt.axis('off')
plt.show()
fig.savefig("word1.png", dpi=900)
```

```
<wordcloud.wordcloud.WordCloud object at 0x000001879CDF4DC8>
```



9. Text Preparation

- Tokenization - for this we use the Bag of Words model
- Vectorization - The list of words is then converted to a matrix of integers by the process of vectorisation. Vectorisation is also called feature extraction.
- We will use the CountVectoriser to tokenise the text and build a vocabulary of known words.

In [22]:
```python
from sklearn.feature_extraction.text import CountVectorizer
import re
cv=CountVectorizer(max_df=0.8,stop_words=stop_words, max_features=10000,
ngram_range=(1,3))
X=cv.fit_transform(corpus)
```

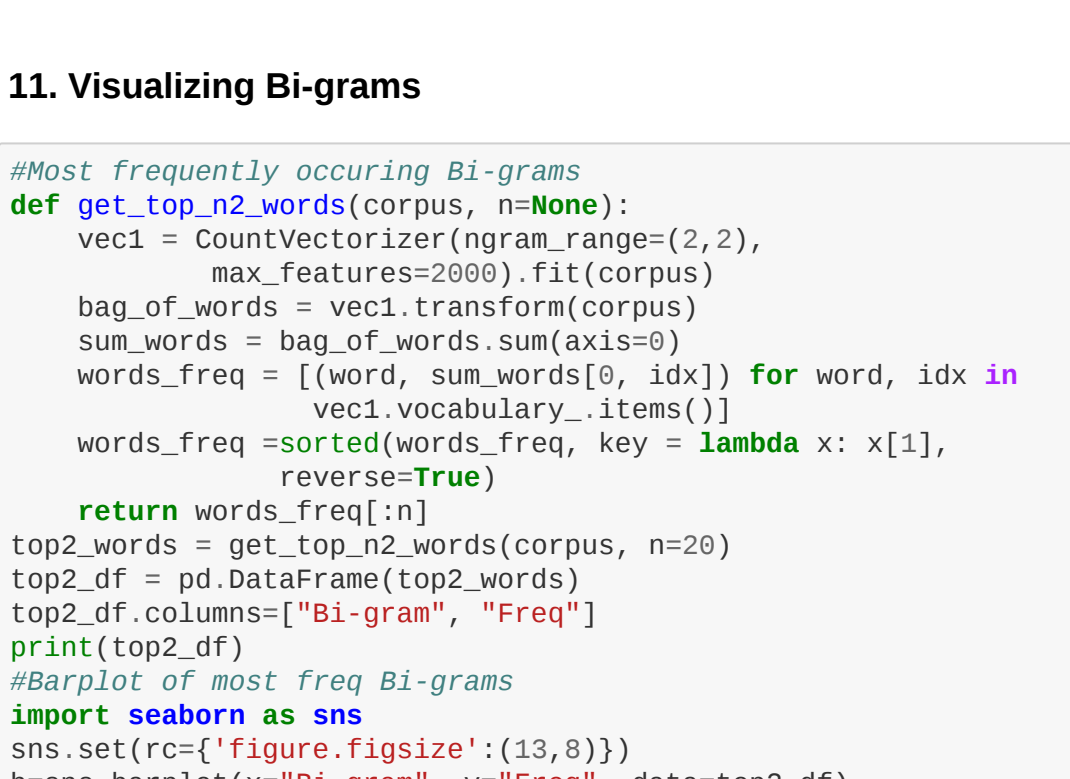
In [23]:
```python
list(cv.vocabulary_.keys())[:10]
```

Out[23]:
```
['non',
 'negative',
 'matrix',
 'factorization',
 'nmf',
 'useful',
 'decomposition',
 'multivariate',
 'data',
 'different']
```

10. Visualizing Uni-grams

In [25]:
```python
#Most frequently occuring words
def get_top_n_words(corpus, n=None):
    vec = CountVectorizer().fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in
                  vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1],
                       reverse=True)
    return words_freq[:n]
#Convert most freq words to dataframe for plotting bar plot
top_words = get_top_n_words(corpus, n=20)
top_df = pd.DataFrame(top_words)
top_df.columns=["Word", "Freq"]
#Barplot of most freq words
import seaborn as sns
sns.set(rc={'figure.figsize':(13,8)})
g = sns.barplot(x="Word", y="Freq", data=top_df)
g.set_xticklabels(g.get_xticklabels(), rotation=30)
```

Out[25]:
```
[Text(0, 0, 'model'),
 Text(0, 0, 'algorithm'),
 Text(0, 0, 'learning'),
 Text(0, 0, 'method'),
 Text(0, 0, 'problem'),
 Text(0, 0, 'data'),
 Text(0, 0, 'approach'),
 Text(0, 0, 'function'),
 Text(0, 0, 'based'),
 Text(0, 0, 'network'),
 Text(0, 0, 'time'),
 Text(0, 0, 'paper'),
 Text(0, 0, 'result'),
 Text(0, 0, 'task'),
 Text(0, 0, 'distribution'),
 Text(0, 0, 'propose'),
 Text(0, 0, 'state'),
 Text(0, 0, 'feature'),
 Text(0, 0, 'image'),
 Text(0, 0, 'performance')]
```

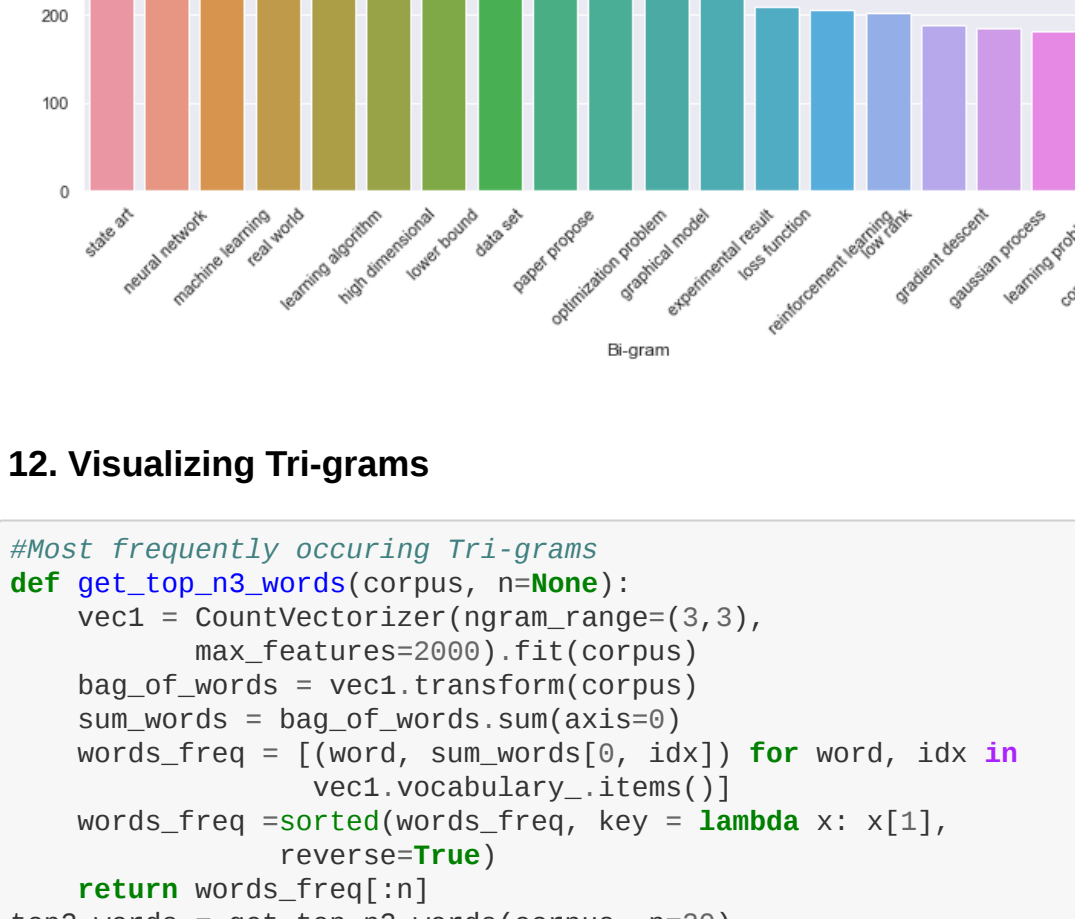


11. Visualizing Bi-grams

In [26]:
```python
#Most frequently occuring Bi-grams
def get_top_n2_words(corpus, n=None):
    vec1 = CountVectorizer(ngram_range=(2,2),
           max_features=2000).fit(corpus)
    bag_of_words = vec1.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in
                  vec1.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1],
                reverse=True)
    return words_freq[:n]
top2_words = get_top_n2_words(corpus, n=20)
top2_df = pd.DataFrame(top2_words)
top2_df.columns=["Bi-gram", "Freq"]
print(top2_df)
#Barplot of most freq Bi-grams
import seaborn as sns
sns.set(rc={'figure.figsize':(13,8)})
h=sns.barplot(x="Bi-gram", y="Freq", data=top2_df)
h.set_xticklabels(h.get_xticklabels(), rotation=45)
```

```
                  Bi-gram  Freq
0               state art   718
1          neural network   561
2        machine learning   413
3              real world   376
4      learning algorithm   350
5        high dimensional   326
6             lower bound   259
7                data set   239
8           paper propose   237
9    optimization problem   225
10        graphical model   224
11    experimental result   220
12          loss function   208
13 reinforcement learning   205
14               low rank   202
15       gradient descent   188
16       gaussian process   184
17       learning problem   180
18       convergence rate   178
19    stochastic gradient   175
```

Out[26]:
```
[Text(0, 0, 'state art'),
 Text(0, 0, 'neural network'),
 Text(0, 0, 'machine learning'),
 Text(0, 0, 'real world'),
 Text(0, 0, 'learning algorithm'),
 Text(0, 0, 'high dimensional'),
 Text(0, 0, 'lower bound'),
 Text(0, 0, 'data set'),
 Text(0, 0, 'paper propose'),
 Text(0, 0, 'optimization problem'),
 Text(0, 0, 'graphical model'),
 Text(0, 0, 'experimental result'),
 Text(0, 0, 'loss function'),
 Text(0, 0, 'reinforcement learning'),
 Text(0, 0, 'low rank'),
 Text(0, 0, 'gradient descent'),
 Text(0, 0, 'gaussian process'),
 Text(0, 0, 'learning problem'),
 Text(0, 0, 'convergence rate'),
 Text(0, 0, 'stochastic gradient')]
```



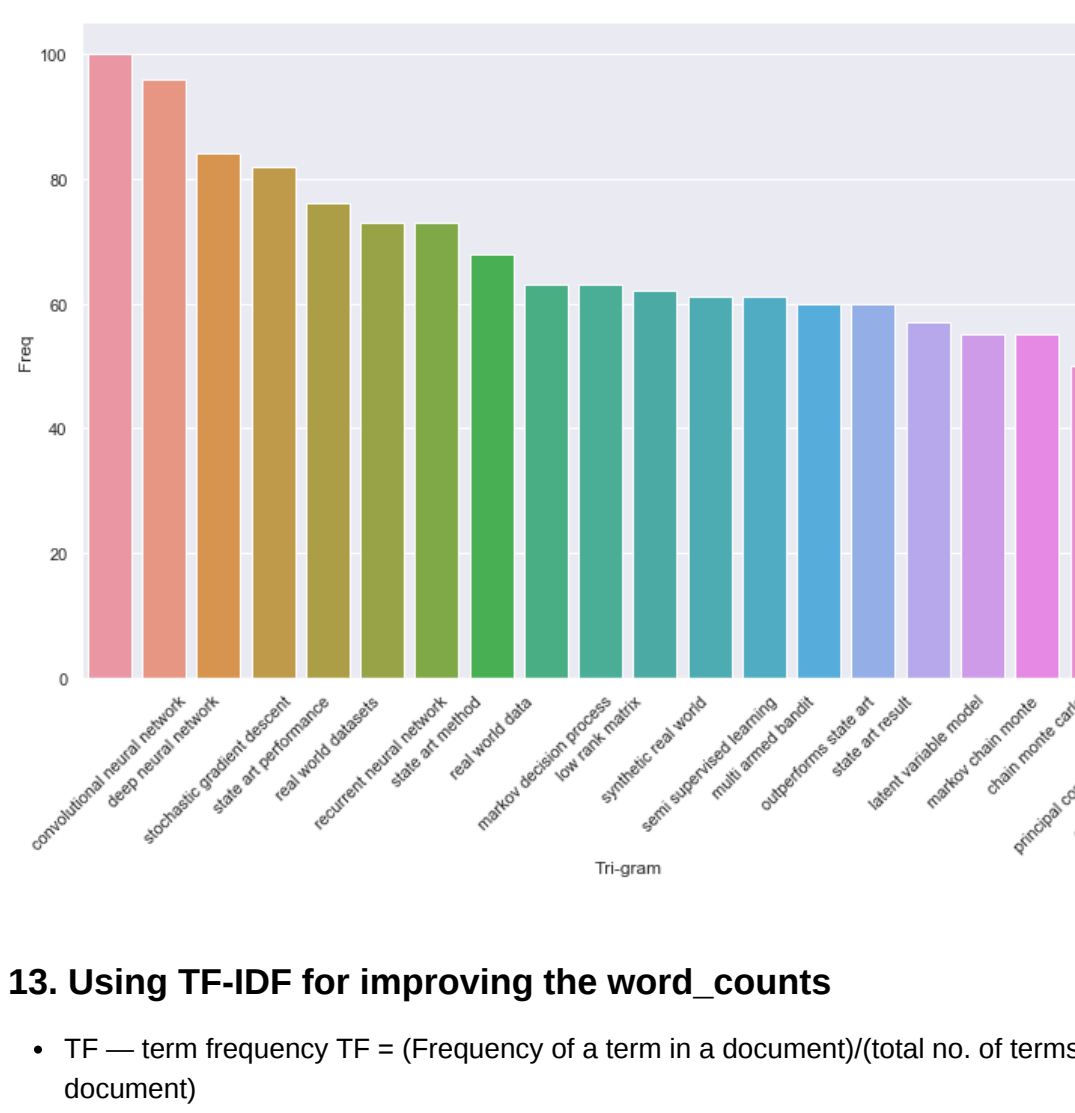
Freq
700
600
500
400
300
200
100
0
state art
neural network
machine learning
real world
learning algorithm
high dimensional
lower bound
data set
paper propose
optimization problem
graphical model
experimental result
loss function
reinforcement learning
low rank
gradient descent
gaussian process
learning problem
convergence rate
stochastic gradient
Bi-gram

12. Visualizing Tri-grams

In [28]:
```python
#Most frequently occuring Tri-grams
def get_top_n3_words(corpus, n=None):
    vec1 = CountVectorizer(ngram_range=(3,3),
           max_features=2000).fit(corpus)
    bag_of_words = vec1.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in
                  vec1.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1],
                reverse=True)
    return words_freq[:n]
top3_words = get_top_n3_words(corpus, n=20)
top3_df = pd.DataFrame(top3_words)
top3_df.columns=["Tri-gram", "Freq"]
print(top3_df)
#Barplot of most freq Tri-grams
import seaborn as sns
sns.set(rc={'figure.figsize':(13,8)})
j=sns.barplot(x="Tri-gram", y="Freq", data=top3_df)
j.set_xticklabels(j.get_xticklabels(), rotation=45)
```

```
                      Tri-gram  Freq
0    convolutional neural network   100
1            deep neural network    96
2     stochastic gradient descent    84
3          state art performance    82
4              real world datasets    76
5        recurrent neural network    73
6                state art method    73
7                  real world data    68
8         markov decision process    63
9                  low rank matrix    63
10            synthetic real world    62
11        semi supervised learning    61
12               multi armed bandit    61
13            outperforms state art    60
14                 state art result    60
15           latent variable model    57
16              markov chain monte    55
17               chain monte carlo    55
18  principal component analysis    50
19    empirical risk minimization    50
```

Out[28]:
```
[Text(0, 0, 'convolutional neural network'),
 Text(0, 0, 'deep neural network'),
 Text(0, 0, 'stochastic gradient descent'),
 Text(0, 0, 'state art performance'),
 Text(0, 0, 'real world datasets'),
 Text(0, 0, 'recurrent neural network'),
 Text(0, 0, 'state art method'),
 Text(0, 0, 'real world data'),
 Text(0, 0, 'markov decision process'),
 Text(0, 0, 'low rank matrix'),
 Text(0, 0, 'synthetic real world'),
 Text(0, 0, 'semi supervised learning'),
 Text(0, 0, 'multi armed bandit'),
 Text(0, 0, 'outperforms state art'),
 Text(0, 0, 'state art result'),
 Text(0, 0, 'latent variable model'),
 Text(0, 0, 'markov chain monte'),
 Text(0, 0, 'chain monte carlo'),
 Text(0, 0, 'principal component analysis'),
 Text(0, 0, 'empirical risk minimization')]
```



Freq
100
80
60
40
20
0
convolutional neural network
deep neural network
stochastic gradient descent
state art performance
real world datasets
recurrent neural network
state art method
real world data
markov decision process
low rank matrix
synthetic real world
semi supervised learning
multi armed bandit
outperforms state art
state art result
latent variable model
markov chain monte
chain monte carlo
principal component analysis
empirical risk minimization
Tri-gram

13. Using TF-IDF for improving the word_counts

- TF — term frequency TF = (Frequency of a term in a document)/(total no. of terms in document)
- IDF — Inverse document frequency IDF = (log(Total documents))/(no. of documents with the term)

In [29]:
```python
from sklearn.feature_extraction.text import TfidfTransformer

tfidf_transformer=TfidfTransformer(smooth_idf=True,use_idf=True)
tfidf_transformer.fit(X)
# get feature names
feature_names=cv.get_feature_names()

# fetch document for which keywords needs to be extracted
doc=corpus[532]

#generate tf-idf for the given document
tf_idf_vector=tfidf_transformer.transform(cv.transform([doc]))
```

In [30]:
```python
#Function for sorting tf_idf in descending order
from scipy.sparse import coo_matrix
def sort_coo(coo_matrix):
    tuples = zip(coo_matrix.col, coo_matrix.data)
    return sorted(tuples, key=lambda x: (x[1], x[0]), reverse=True)

def extract_topn_from_vector(feature_names, sorted_items, topn=10):
    """get the feature names and tf-idf score of top n items"""

    #use only topn items from vector
    sorted_items = sorted_items[:topn]

    score_vals = []
    feature_vals = []

    # word index and corresponding tf-idf score
    for idx, score in sorted_items:

        #keep track of feature name and its corresponding score
        score_vals.append(round(score, 3))
        feature_vals.append(feature_names[idx])

    #create a tuples of feature,score
    #results = zip(feature_vals,score_vals)
    results= {}
    for idx in range(len(feature_vals)):
        results[feature_vals[idx]]=score_vals[idx]

    return results
#sort the tf-idf vectors by descending order of scores
sorted_items=sort_coo(tf_idf_vector.tocoo())
#extract only the top n; n here is 10
keywords=extract_topn_from_vector(feature_names,sorted_items,5)

# now print the results
print("\nAbstract:")
print(doc)
print("\nKeywords:")
for k in keywords:
    print(k,keywords[k])
```

```
Abstract:
present theory compositionality stochastic optimal control showing task
optimal controller constructed certain primitive primitive feedback cont
roller pursuing agenda mixed proportion much progress making towards age
nda compatible agenda present task resulting composite control law prova
bly optimal problem belongs certain class class rather general yet numbe
r unique property bellman equation made linear even non linear discrete
dynamic give rise compositionality developed special case linear dynamic
gaussian noise framework yield analytical solution e non linear mixture
linear quadratic regulator without requiring final cost quadratic genera
lly natural set control primitive constructed applying svd green functio
n bellman equation illustrate theory context human arm movement idea opt
imality compositionality prominent field motor control yet hard reconcil
e work make possible

Keywords:
compositionality 0.33
primitive 0.295
```