Word Embeddings and NLP

Janu Verma

Data Scientist, Hike

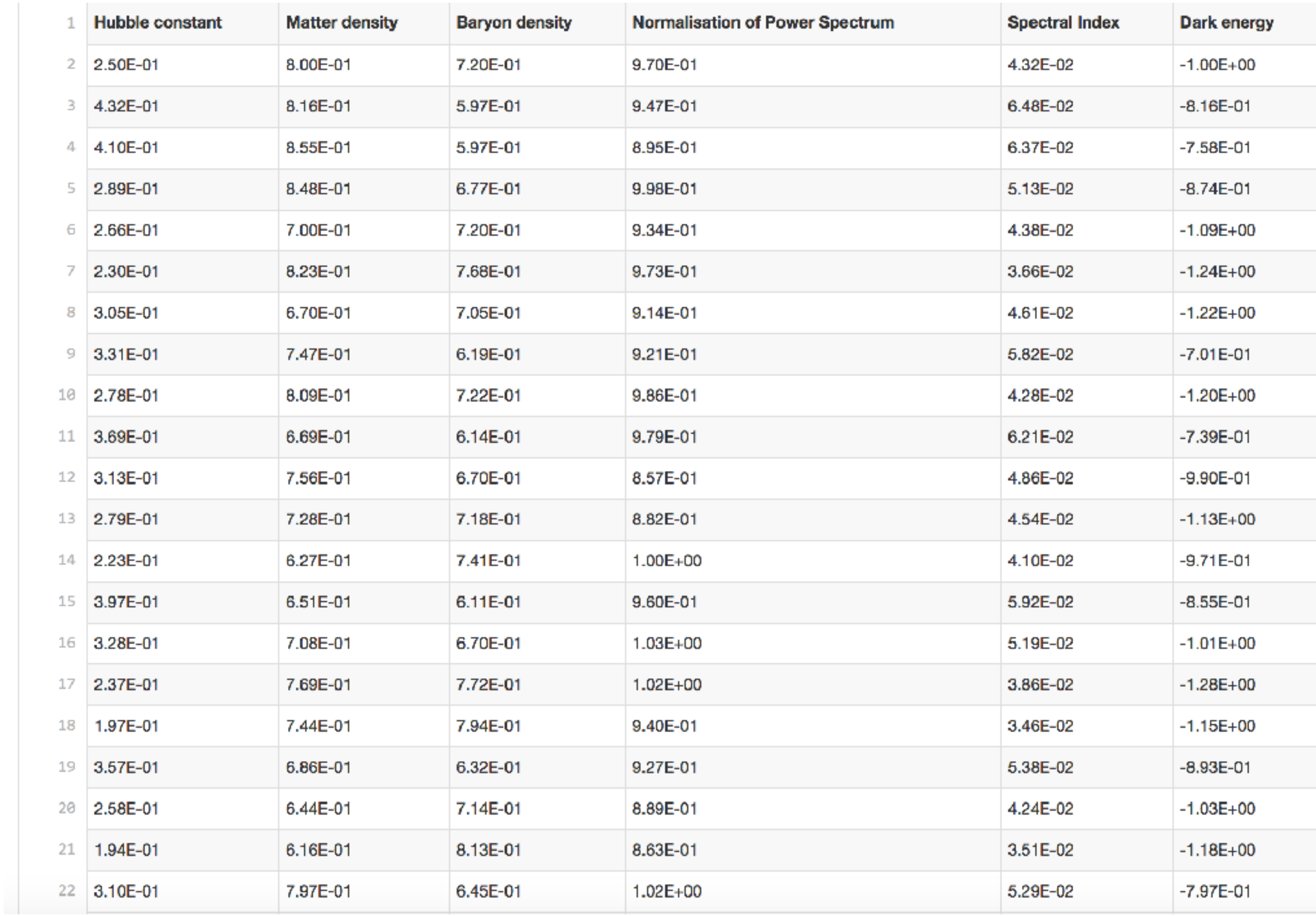
http://jverma.github.io/

janu@hike.in

@januverma

Motivation

- Main Motivation - *Natural Language Processing* (NLP) and *Information Retrieval* (IR).
- NLP essentially means enabling computers to derive meaning from human-input or machine-input natural languages like English, Hindi, Mandarin
- Some standard NLP tasks are tokenization, classification and clustering of textual documents, speech recognition, machine translation, auto-summarization of documents, language generation, chat-bots, etc.
- A retrieval system is basically a *search engine* e.g. Google, Bing.
- The most popular approach to NLP and IR is *statistical machine learning*, e.g. *supervised* where a model is trained on data labeled for the task to make predictions on un-labeled data or *unsupervised* where no labeled data is available
- Standard ML algorithms expect a data instance as a vector, in fact, when we say data, we mean a matrix (a row/vector for each data point) - *csv/tsv* file, excel sheet, data frame, *numpy* array etc.



Hubble constant	Matter density	Baryon density	Normalisation of Power Spectrum	Spectral Index	Dark energy
2.50E-01	8.00E-01	7.20E-01	9.70E-01	4.32E-02	-1.00E+00
4.32E-01	8.16E-01	5.97E-01	9.47E-01	6.48E-02	-8.16E-01
4.10E-01	8.55E-01	5.97E-01	8.95E-01	6.37E-02	-7.58E-01
2.89E-01	8.48E-01	6.77E-01	9.98E-01	5.13E-02	-8.74E-01
2.66E-01	7.00E-01	7.20E-01	9.34E-01	4.38E-02	-1.09E+00
2.30E-01	8.23E-01	7.68E-01	9.73E-01	3.66E-02	-1.24E+00
3.05E-01	6.70E-01	7.05E-01	9.14E-01	4.61E-02	-1.22E+00
3.31E-01	7.47E-01	6.19E-01	9.21E-01	5.82E-02	-7.01E-01
2.78E-01	8.09E-01	7.22E-01	9.86E-01	4.28E-02	-1.20E+00
3.69E-01	6.69E-01	6.14E-01	9.79E-01	6.21E-02	-7.39E-01
3.13E-01	7.56E-01	6.70E-01	8.57E-01	4.86E-02	-9.90E-01
2.79E-01	7.28E-01	7.18E-01	8.82E-01	4.54E-02	-1.13E+00
2.23E-01	6.27E-01	7.41E-01	1.00E+00	4.10E-02	-9.71E-01
3.97E-01	6.51E-01	6.11E-01	9.60E-01	5.92E-02	-8.55E-01
3.28E-01	7.08E-01	6.70E-01	1.03E+00	5.19E-02	-1.01E+00
2.37E-01	7.69E-01	7.72E-01	1.02E+00	3.86E-02	-1.28E+00
1.97E-01	7.44E-01	7.94E-01	9.40E-01	3.46E-02	-1.15E+00
3.57E-01	6.86E-01	6.32E-01	9.27E-01	5.38E-02	-8.93E-01
2.58E-01	6.44E-01	7.14E-01	8.89E-01	4.24E-02	-1.03E+00
1.94E-01	6.16E-01	8.13E-01	8.63E-01	3.51E-02	-1.18E+00
3.10E-01	7.97E-01	6.45E-01	1.02E+00	5.29E-02	-7.97E-01

Text Classification

- Consider a task where we have to classify news articles into 20 different categories e.g. finance, politics, entertainment, sports etc.

- Supervised ML: Train a classifier on given labeled data which learns to make predictions on new, unseen documents.

- Text Document —-> Category

- Need a way to *vectorize* the text documents.

Traditional Approaches

- *Goal: A vector representation of words in a document.*

- *Vector Space Models:* Most popular models of NLP and IR. Expect a word/sentence/document to be a vector.

- *One Hot Encoding (OHE):*

    ★ Words are fundamental and indestructible.

    ★ A word is represented as a point in a vector space of dimension equal to the size of vocabulary (# of words in the corpus) i.e. each word contributes a dimension to the vector. i.e the word vectors are a basis for the vector space.

    ★ The entries in the vector representation of a word are all zeros except the one corresponding to the word under consideration.

One Hot Encoding - binary vectors

- Consider the following sentence -

  "Donald Trump called BuzzFeed a failing piece of garbage."

- Size of vocabulary is 9, 7 ignoring *stop-words* (a, of).
- **An** OHE for the words will be
  - ★ Donald = [1,0,0,0,0,0,0]
  - ★ Trump = [0,1,0,0,0,0,0]
  - ★ called = [0,0,1,0,0,0,0]
  - ★ BuzzFeed = [0,0,0,1,0,0,0]
  - ★ failing = [0,0,0,0,1,0,0]
  - ★ piece = [0,0,0,0,0,1,0]
  - ★ garbage = [0,0,0,0,0,0,1]

- This is not the only possibility for OHE. Any basis for the vector space can be an OHE. Popular approaches are *count vectors* and *tf-idf vectors.*

- In count vector representation, the entry corresponding to each word is it’s raw frequency in the document.

- Consider the paragraph -

  “Donald Trump called BuzzFeed a failing piece of garbage. Buzzfeed is apparently so pleased at being called such by Trump that they started selling ‘failing piece of garbage’ shirts.”

- The size of vocab is 12, ignoring stop-words.

★ Donald = [1,0,0,0,0,0,0,0,0,0,0,0]

★ Trump = [0,2,0,0,0,0,0,0,0,0,0,0]

★ called = [0,0,2,0,0,0,0,0,0,0,0,0]

★ BuzzFeed = [0,0,0,2,0,0,0,0,0,0,0,0]

★ failing = [0,0,0,0,2,0,0,0,0,0,0,0]

★ piece = [0,0,0,0,0,2,0,0,0,0,0,0]

★ garbage = [0,0,0,0,0,0,2,0,0,0,0,0]

★ apparently = [0,0,0,0,0,0,0,1,0,0,0,0]

★ pleased = [0,0,0,0,0,0,0,0,1,0,0,0]

★ started = [0,0,0,0,0,0,0,0,0,1,0,0]

★ selling = [0,0,0,0,0,0,0,0,0,01,0]

★ shirts = [0,0,0,0,0,0,0,0,0,0,01]

Vector Space Model

- Using a vector representation of words, we can build a representation of documents.

- A document is represented as a point in a vector space of dimension equal to the size of the vocabulary i.e. again the words form the basis, and the contribution of each word in the vector is the OHE contribution of the word.

- e.g. the previous document can be represented by the following vector

  basis = [Donald, Trump, called, BuzzFeed, failing, piece, garbage, apparently, pleased, started, selling, shirts]

  document = [1,2,2,2,2,2,2,1,1,1,1,1]

- This way we obtain a vector representation of the text documents, which can be fed into a machine learning model.

- We can extend this kind of encoding to a corpus of documents, which is the original motivation.
- Consider the following sentences -

d1 = “Donald Trump called BuzzFeed a failing piece of garbage.”

d2 = “ Buzzfeed is apparently so pleased at being called such by Trump that they started selling ‘failing piece of garbage’ shirts.”

d3 = “Donald Trump is the new President-Elect of the US”

d4 = “The Trump University has such a bad reputation.”

d5 = “BuzzFeed is growing at an alarming pace.”

- The vocabulary is of size 21. Each document can be represented as a vector of length 21, where each word corresponds to a dimension.

- In OHE, the entry corresponding to a word in a document vector is non-zero if the word is present in the document.

- For binary encoding, the non-zero contribution is 1.

    - ★ d1 = [1,1,1,1,1,1,1,0,…,0]

    - ★ d2 = [0,1,1,1,1,1,1,1,1,1,1,1,0,…,0]

    - ★ d3 = [1,1,0,…,0,1,1,1,0,0,0,0,0,0]

    - ★ d4 = [0,1,0,..,0,1,1,1,0,0,0]

    - ★ d4 = [0,0,0,1,0,..,0,1,1,1]

tf-idf

- term frequency-inverse document frequency (tf-idf) is the most popular approach of vectorizing documents.
- The problem with raw frequency approach is that the high-frequent words will skew the model.
- Term-frequencies can be normalised by the max frequency of a term in the document.
- tf-idf is the product of the normalised term frequency and the inverse of the document frequency (i.e. the fraction of documents that contain the term).
- This attempts to give high weightage to the discerning words.
- E.g. article about computer science and physics.

Now we have vectors for each of the documents, we can do NLP and IR using machine learning e.g. cluster the documents using standard algorithms (e.g. k-means), classification of documents etc.

Text classification model

- Define the vocabulary i.e. all the terms that will be considered in the model (feature engineering). e.g. remove stop-words, or remove non-domain words etc.
- Obtain a vectorization of the documents in the corpus. Each doc becomes a vector in a high-dimensional space, with dimension equal to the total length of the vocabulary. tf-idf is a very successful approach.
- Choose a machine learning algorithm e.g. SVM, Naive Bayes and Logistic regression have been proven to be very effective in different situations.
- Speaker always starts with a tf-idf based SVM.

Limitations of OHE

- The words are treated atomic.
- No information about the relationship between the words.

  distance(apple, banana) = 0

- Not a good result, both apple and banana are fruits. Int fact, we so often use apple-banana as a phrase.

  distance(apple, banana) = distance(apple, airplane) = 0

- Only comparison supported is equality.
- Furthermore, such a representation results in word vector which are extremely sparse.

Distributed Representations

- **Assumption:** *Words that appear in same context are semantic closer than the words which do not share same context.*
- A word can be represented as points in a continuous vector space where semantically similar words corresponds to nearby points.
- This representation is also called *word embeddings*, since we are embedding word vectors in the distributed vector space.
- Essentially, the weight of each word in the vector is distributed across many dimensions.
- Instead of a one-to-one mapping between a word and a basis vector (dimension), the word contribution is spread across all the dimensions of the vector.
- The dimensions are believed to capture the semantic properties of the words.

Distributed Reps

- Distributed reps take the following form
    - ★ Friends = [0.73,0.34,0.52,0.01]
    - ★ Work = [0.65,0.79,0.22,0.1]
    - ★ And = [0.87,0.94,0.14,0.7]
    - ★ Play = [0.73, 0.69, 0.89, 0.4]
    - ★ Together = [0.87,0.79,0.22,0.09]
- Please notice that these vectors are chosen arbitrarily, and do not show an actual representation. The sole is purpose is to give an example.

“Somewhat surprisingly, it was found that similarity of word representations goes beyond simple syntactic regularities. Using a word offset technique where simple algebraic operations are performed on the word vectors, it was shown for example that vector(“King”) – vector(“Man”) + vector(“Woman”) results in a vector that is closest to the vector representation of the word Queen.”

–Mikolov et al

Examples/Applications

- Machine Translation has been shown to achieve much higher accuracy using distributed representations.
- One can make following assertions :
  - *Distance(France, Germany) < Distance(France, Spain)*
  - *Vector('Paris') - Vector('France') + Vector('Italy') ~ Vector(Rome)*
  - *Vector('king') - Vector('man') + Vector('woman') ~ Vector('queen')*
- The odd one in *[staple, hammer, saw, drill]* is *staple*.
- Item2vec: word2vec for collaborative filtering and recommendation system. e.g one can infer:
  - *Vector(David Guetta) - Vector(Avicii) + Vector(Beyonce) -> Vector(Rihanna)*
- BioVectors: Word vectors for Bioinformatics.
- BioVectors can characterise biological sequences in terms of biochemical and biophysical interpretations of the underlying patterns.

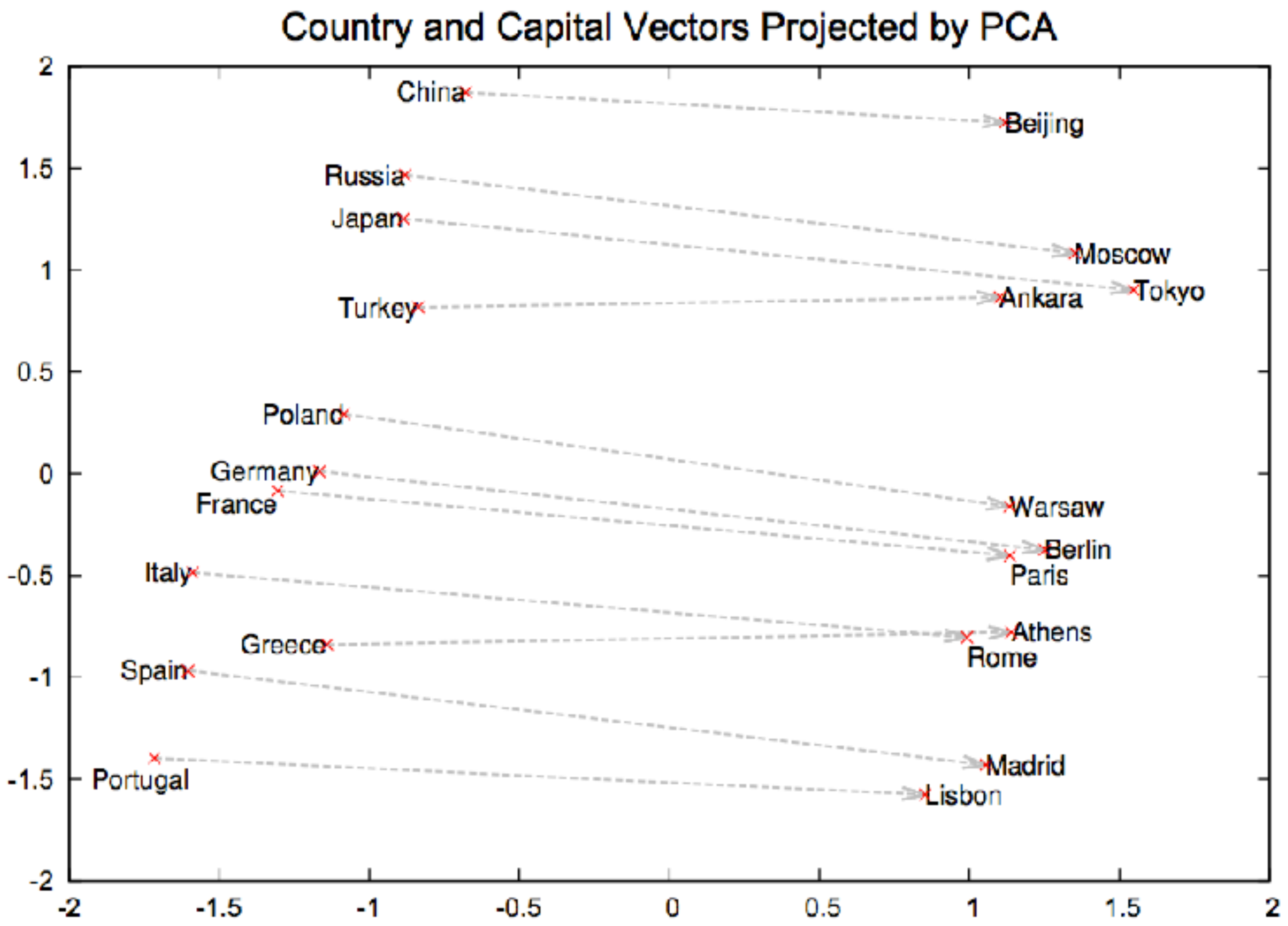


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

Czech + currency	Vietnam + capital	German + airlines	Russian + river	French + actress
koruna	Hanoi	airline Lufthansa	Moscow	Juliette Binoche
Check crown	Ho Chi Minh City	carrier Lufthansa	Volga River	Vanessa Paradis
Polish zolty	Viet Nam	flag carrier Lufthansa	upriver	Charlotte Gainsbourg
CTK	Vietnamese	Lufthansa	Russia	Cecile De

Table 5: Vector compositionality using element-wise addition. Four closest tokens to the sum of two vectors are shown, using the best Skip-gram model.

Newspapers			
New York	New York Times	Baltimore	Baltimore Sun
San Jose	San Jose Mercury News	Cincinnati	Cincinnati Enquirer
NHL Teams			
Boston	Boston Bruins	Montreal	Montreal Canadiens
Phoenix	Phoenix Coyotes	Nashville	Nashville Predators
NBA Teams			
Detroit	Detroit Pistons	Toronto	Toronto Raptors
Oakland	Golden State Warriors	Memphis	Memphis Grizzlies
Airlines			
Austria	Austrian Airlines	Spain	Spainair
Belgium	Brussels Airlines	Greece	Aegean Airlines
Company executives			
Steve Ballmer	Microsoft	Larry Page	Google
Samuel J. Palmisano	IBM	Werner Vogels	Amazon

Table 2: Examples of the analogical reasoning task for phrases (the full test set has 3218 examples). The goal is to compute the fourth phrase using the first three. Our best model achieved an accuracy of 72% on this dataset.

Text Classification revisited

- As we have seen, word2vec gives another way to vectorize the textual documents, they can be employed for text classification tasks.
- One approach is to get word vectors for each word in the document and then take an average of these vectors to arrive at the word2vec representation of the document.
- There are also tools like Doc2vec which compute an embedded representation from documents.
- Another approach is to concatenate the word vectors for each word to get the document representation.
- After we have a vector rep, we can use SVM/Logistic regression/feed-forward neural network etc. to build a classification model.
- There are pre-trained word2vec available which can be used in lieu of training a new model.

- Word embeddings as input to a sequence model (e.g. RNN, LSTM) which take a sequence of vectors as input (e.g. a sentence/document) as opposed to fixed-length vectors.

- LSTM has been very effective in text classification problems. e.g. movie review sentiment classification

- Convolutional Neural networks for text classification.

- **Language Modeling :** Sequence-to-Sequence models for language translation, language generation ,chat bot/QA system etc.

- Document summarization

- Caption generation for an image.

Search engines

- Given a query, retrieve the most relevant pages.
- Base of all search engines is the method to find docs most similar to the input query.
- Vectorize the query using word-embeddings and look for similar vectors representing pages.
- Dog v/s canine, dog v/s cat.
- Classification into relevant and non-relevant pages.
- Personalization: relevant or non-relevant for the current user.

Named Entity Recognition

- Locate and classify *named entities* in text into pre-defined categories such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc.

  Jim bought 300 shares of Acme Corp. in 2006.

  [Jim](person) bought 300 shares of [Acme Corp.](org) in [2006](time)

- Very hard problem. Build set of labeled NEs and learn the word embeddings, tag new entities based on their similarity to the earlier tags.
- e.g. Virat Kohli and MS Dhoni appear in the same context and would have similar word-embeddings.

Learning Distributed Representations

- **Question:** *How do we compute such a representation ?*
- Distributed representations of words can be learned by training a model on a corpus of textual data
- Thomas Mikolov et al (Google, Inc.) proposed an efficient method to learn these embeddings, making it feasible to learn high-quality word vectors on a huge corpus of data.
- Basically, we train a Neural Network on a huge collection of textual files (e.g. wikipedia, google news, google books) and learn a representation of each word possible.
- An implementation of this model, word2vec, is made public by Google. It has a few pre-trained models which can be used directly on documents of interest without building/training neural nets.
- Two architectures were proposed for training word embeddings - CBOW and Skip-gram.

Neural Network Setup

- Consider our favourite sentence -

  "Donald Trump called BuzzFeed a failing piece of garbage."

- We choose a sliding window to quantify the context. One of the parameters of the neural net is the length of the sliding window.

- In each sliding window, there is a *central word* which is under attention, and few words preceding and following the central word.

- In the above example if we choose the length of sliding window to be 3, the the context of BuzzFeed, e.g. is

  [Donald, Trump, called, failing, piece, garbage]

CBOW

- The context words form the input layer of the CBOW neural network, and each word is represented as a vector using one-hot schema.
- There is one hidden layer, and one output layer. The output layer is formed by the central words (i.e. each element in the vocabulary). This way we learn a representation for each word in terms of the context words.
- The actual ordering of the context words is irrelevant, this is called bag-of-words assumption.
- The training objective is to maximize the conditional probability of observing the actual output word (the focus word) given the input context words, with regard to the weights.
- In our example, given the input [Donald, Trump, called, failing, piece, garbage], we want to maximize the probability of getting "Buzzfeed" as the output.

Skip gram

- The skip-gram method is completely opposite of the CBOW method. Here the central word is the input layer, and the context words are now at the output layer.
- Again there is one hidden layer.
- At the output layer, we now output multinomial distributions instead of just one.
- The training objective is to minimize the summed prediction error across all context words in the output layer.
- In our example, the input would be “buzzfeed”, and we hope to see [Donald, Trump, called, failing, piece, garbage] at the output layer.
- CBOW is faster, but skip-gram does a better job for not-so-frequent words.

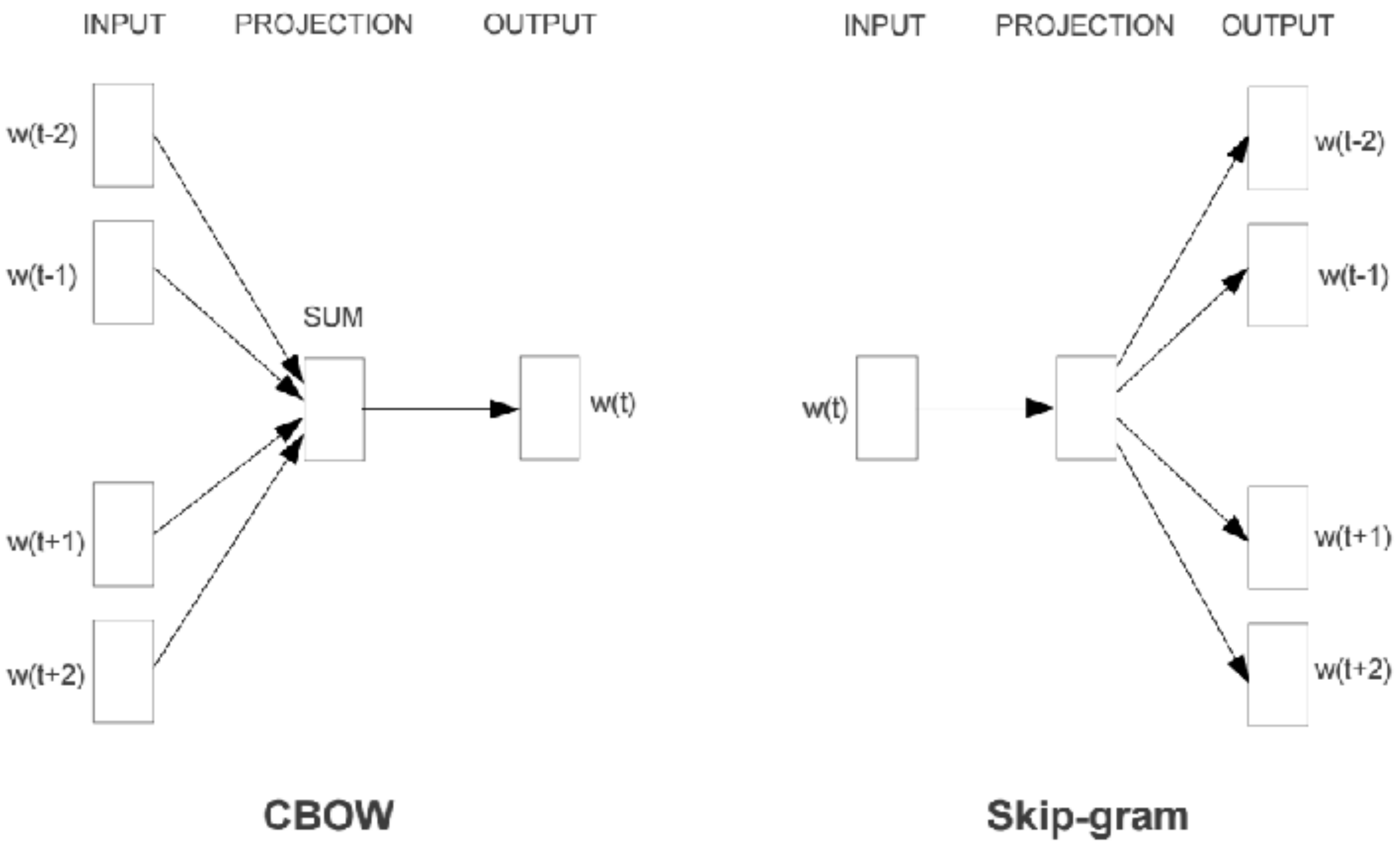


Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

Thanks