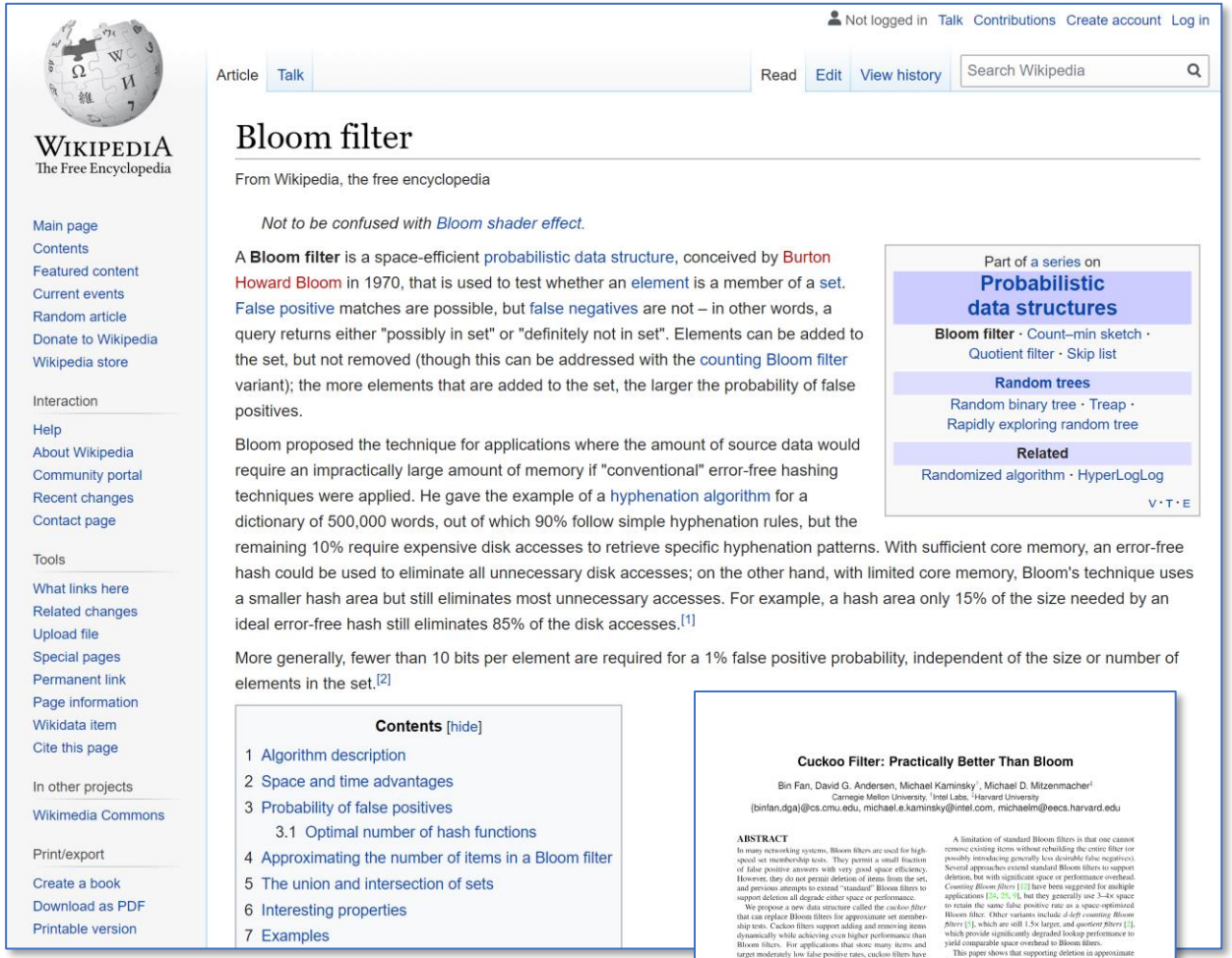
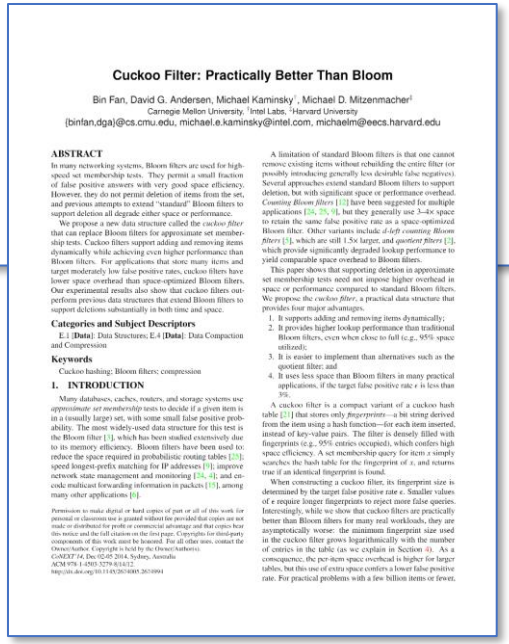
Topic Medley

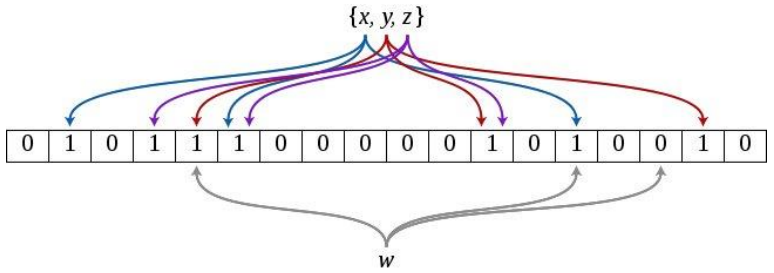
IN3120/IN4120

Bloom Filters

- A probabilistic data structure for checking set membership
  - “No” means “no”
  - “Yes” means “probably”
- A bit vector with *m* bits, combined with *k* independent hash functions, used to store *n* membership values
  - Pr(false positive) = *f*(*m*, *k*, *n*)
  - Pr(false negative) = 0
- Extensions exists that allow for deleting elements
  - E.g., cuckoo filters or counting Bloom filters

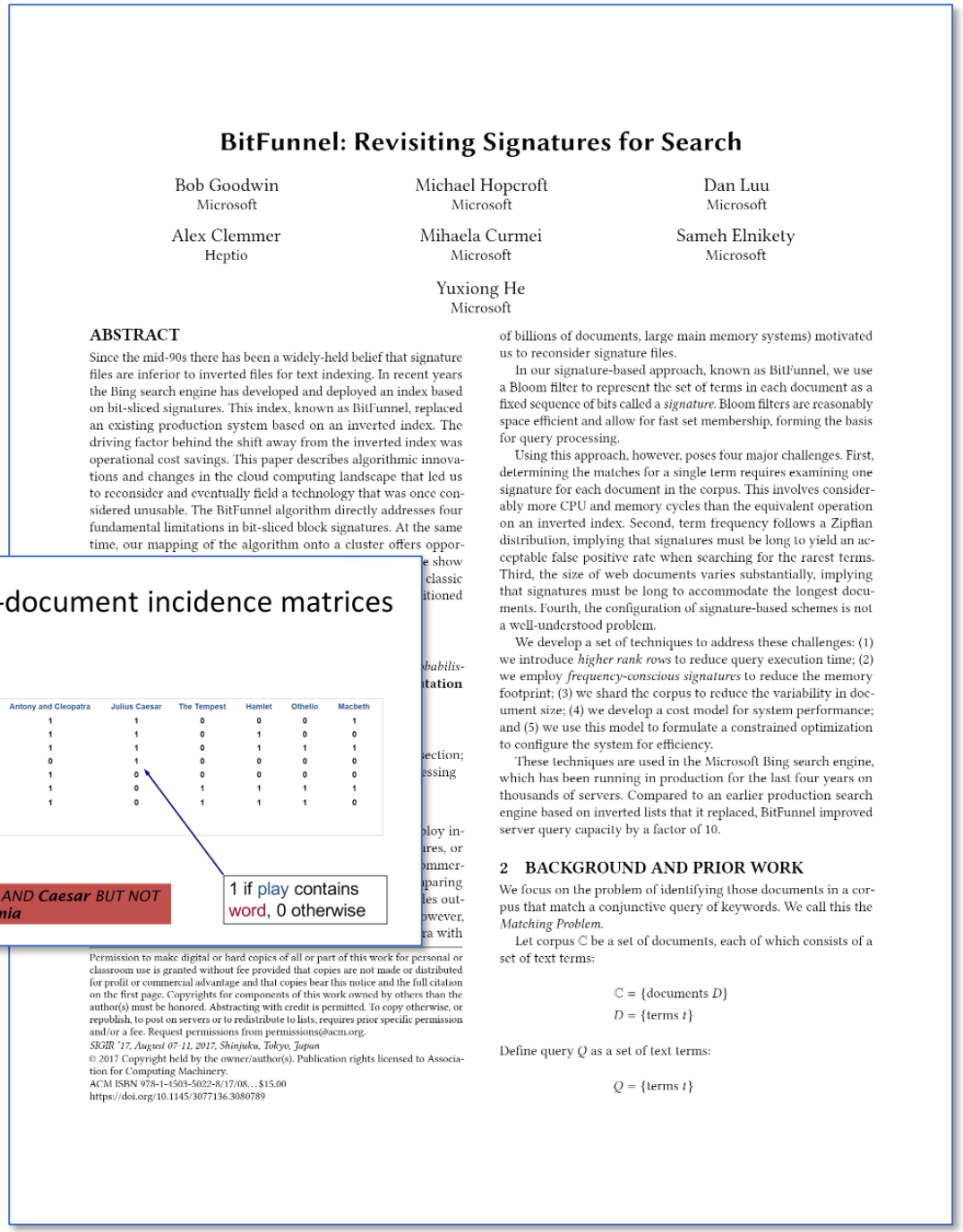






Bit-Signature Indices

- Signature: Sequence of bits in Bloom filter representing the set of terms in the string
- Intersect the document and query signatures to find matches quickly and cost-efficiently
- Cost of false positives “negligible” for web search
  - Results don’t have to be “correct”
  - Rankers save the day
- Plenty of engineering tricks involved
  - E.g., frequency-conscious signatures, higher rank rows, sharding by document length, …



BitFunnel: Revisiting Signatures for Search

Bob Goodwin
Microsoft

Michael Hopcroft
Microsoft

Dan Luu
Microsoft

Alex Clemmer
Heptio

Mihaela Curmei
Microsoft

Sameh Elnikety
Microsoft

Yuxiong He
Microsoft

ABSTRACT

Since the mid-90s there has been a widely-held belief that signature files are inferior to inverted files for text indexing. In recent years the Bing search engine has developed and deployed an index based on bit-sliced signatures. This index, known as BitFunnel, replaced an existing production system based on an inverted index. The driving factor behind the shift away from the inverted index was operational cost savings. This paper describes algorithmic innovations and changes in the cloud computing landscape that led us to reconsider and eventually field a technology that was once considered unusable. The BitFunnel algorithm directly addresses four fundamental limitations in bit-sliced block signatures. At the same time, our mapping of the algorithm onto a cluster offers opportunities to

of billions of documents, large main memory systems) motivated us to reconsider signature files.

In our signature-based approach, known as BitFunnel, we use a Bloom filter to represent the set of terms in each document as a fixed sequence of bits called a *signature*. Bloom filters are reasonably space efficient and allow for fast set membership, forming the basis for query processing.

Using this approach, however, poses four major challenges. First, determining the matches for a single term requires examining one signature for each document in the corpus. This involves considerably more CPU and memory cycles than the equivalent operation on an inverted index. Second, term frequency follows a Zipfian distribution, implying that signatures must be long to yield an acceptable false positive rate when searching for the rarest terms. Third, the size of web documents varies substantially, implying that signatures must be long to accommodate the longest documents. Fourth, the configuration of signature-based schemes is not a well-understood problem.

We develop a set of techniques to address these challenges: (1) we introduce *higher rank rows* to reduce query execution time; (2) we employ *frequency-conscious signatures* to reduce the memory footprint; (3) we shard the corpus to reduce the variability in document size; (4) we develop a cost model for system performance; and (5) we use this model to formulate a constrained optimization to configure the system for efficiency.

These techniques are used in the Microsoft Bing search engine, which has been running in production for the last four years on thousands of servers. Compared to an earlier production search engine based on inverted lists that it replaced, BitFunnel improved server query capacity by a factor of 10.

2 BACKGROUND AND PRIOR WORK

We focus on the problem of identifying those documents in a corpus that match a conjunctive query of keywords. We call this the *Matching Problem*.

Let corpus $\mathbb{C}$ be a set of documents, each of which consists of a set of text terms:

$$\mathbb{C} = \{\text{documents } D\}$$
$$D = \{\text{terms } t\}$$

Define query Q as a set of text terms:

$$Q = \{\text{terms } t\}$$

SIGIR '17, August 07-11, 2017, Shinjuku, Tokyo, Japan

ACM ISBN 978-1-4503-5022-8/17/08…$15.00
https://doi.org/10.1145/3077136.3080789

Term-document incidence matrices

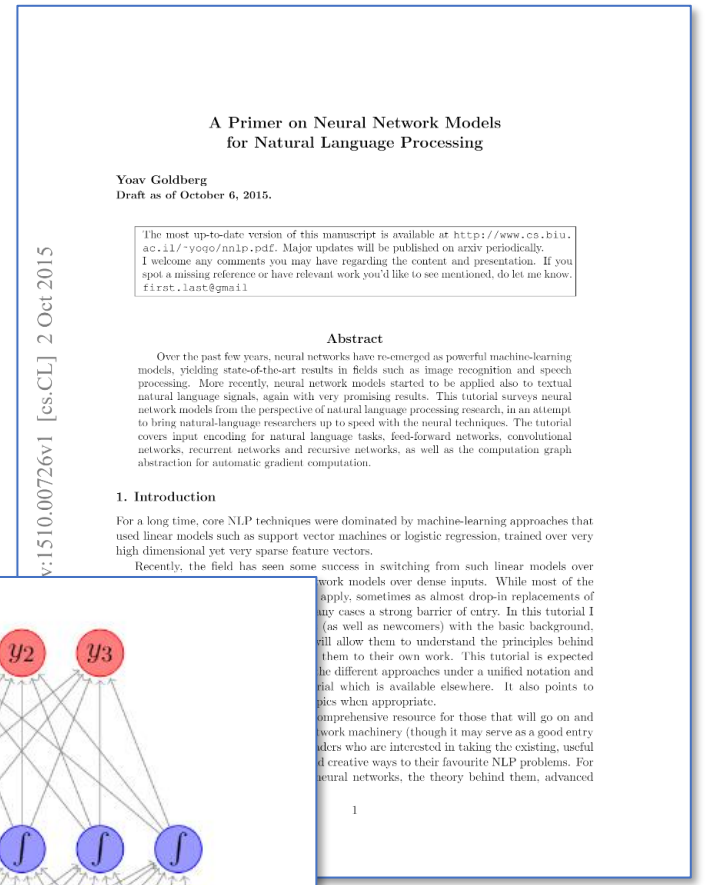
	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

1 if play contains word, 0 otherwise

Brutus** AND **Caesar** BUT NOT **Calpurnia

Neural Networks

- Propagate the signal from input layer to output layer
  - Edges have weights, nodes perform non-linear transformations on their weighted inputs
- Lots of architectures exist!
  - E.g., recurrent networks, LSTMs, convolutional networks, bi-directional transformers, ...
- Learns by adjusting weights so that the outputs mimic the labeled inputs
  - Typically using gradient descent to optimize some loss function



A Primer on Neural Network Models for Natural Language Processing

Yoav Goldberg
Draft as of October 6, 2015.

The most up-to-date version of this manuscript is available at http://www.cs.biu.ac.il/~yogo/nnlp.pdf. Major updates will be published on arxiv periodically.
I welcome any comments you may have regarding the content and presentation. If you spot a missing reference or have relevant work you'd like to see mentioned, do let me know.
first.last@gmail

Abstract

Over the past few years, neural networks have re-emerged as powerful machine-learning models, yielding state-of-the-art results in fields such as image recognition and speech processing. More recently, neural network models started to be applied also to textual natural language signals, again with very promising results. This tutorial surveys neural network models from the perspective of natural language processing research, in an attempt to bring natural-language researchers up to speed with the neural techniques. The tutorial covers input encoding for natural language tasks, feed-forward networks, convolutional networks, recurrent networks and recursive networks, as well as the computation graph abstraction for automatic gradient computation.

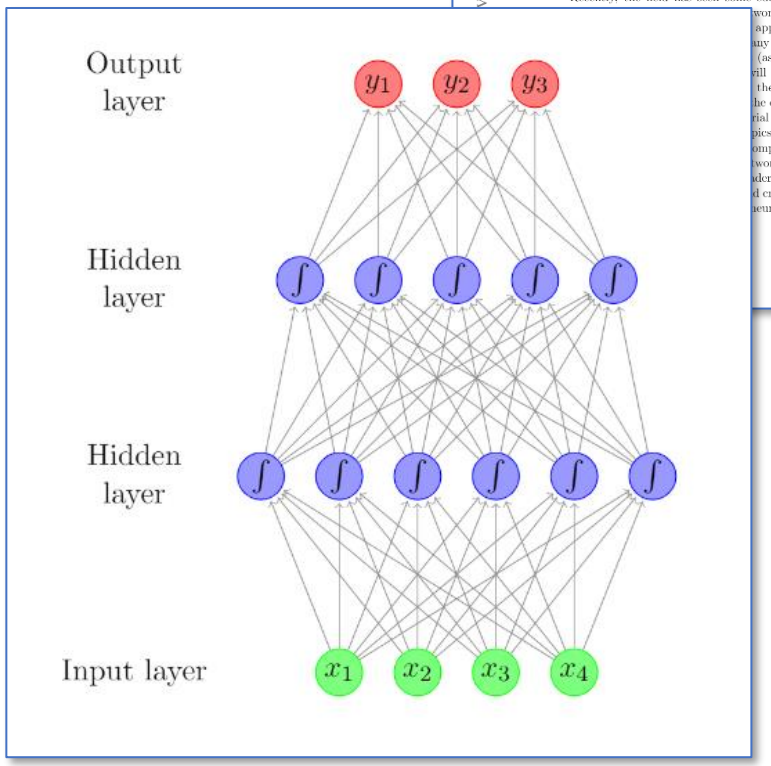
1. Introduction

For a long time, core NLP techniques were dominated by machine-learning approaches that used linear models such as support vector machines or logistic regression, trained over very high dimensional yet very sparse feature vectors.

Recently, the field has seen some success in switching from such linear models over work models over dense inputs. While most of the apply, sometimes as almost drop-in replacements of uy cases a strong barrier of entry. In this tutorial I (as well as newcomers) with the basic background, fill allow them to understand the principles behind them to their own work. This tutorial is expected he different approaches under a unified notation and rial which is available elsewhere. It also points to pics when appropriate.
omprehensive resource for those that will go on and work machinery (though it may serve as a good entry ders who are interested in taking the existing, useful d creative ways to their favourite NLP problems. For eural networks, the theory behind them, advanced

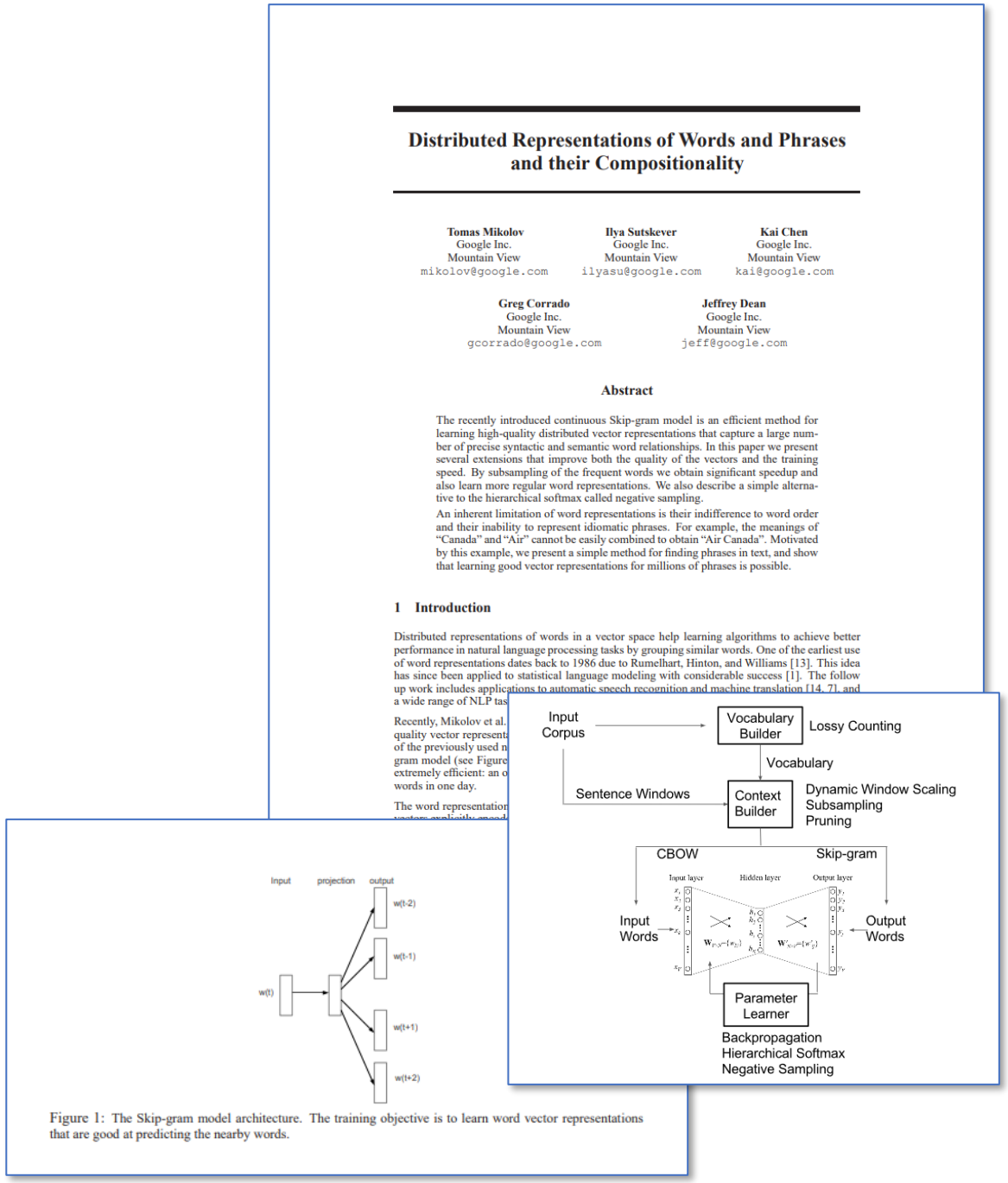
1

v:1510.00726v1 [cs.CL] 2 Oct 2015



Embedding Vectors

- Idea: Words that occur in the same contexts tend to have similar meanings
  - E.g., learn to predict a blanked-out word given its surrounding context, or learn to predict the surrounding context given the blanked-out word
- Word2Vec and friends
  - Further improved by BERT and friends
- "Embeddings": Values at internal layers
  - Idea: Autoencoders
  - A denser, lower-dimensional representation than very sparse, high-dimensional, one-hot representations
- Semantic algebra!
  - Analogies via vector differences



Distributed Representations of Words and Phrases and their Compositionality

Tomas Mikolov Google Inc. Mountain View mikolov@google.com

Ilya Sutskever Google Inc. Mountain View ilyasu@google.com

Kai Chen Google Inc. Mountain View kai@google.com

Greg Corrado Google Inc. Mountain View gcorrado@google.com

Jeffrey Dean Google Inc. Mountain View jeff@google.com

Abstract

The recently introduced continuous Skip-gram model is an efficient method for learning high-quality distributed vector representations that capture a large number of precise syntactic and semantic word relationships. In this paper we present several extensions that improve both the quality of the vectors and the training speed. By subsampling of the frequent words we obtain significant speedup and also learn more regular word representations. We also describe a simple alternative to the hierarchical softmax called negative sampling.

An inherent limitation of word representations is their indifference to word order and their inability to represent idiomatic phrases. For example, the meanings of "Canada" and "Air" cannot be easily combined to obtain "Air Canada". Motivated by this example, we present a simple method for finding phrases in text, and show that learning good vector representations for millions of phrases is possible.

1 Introduction

Distributed representations of words in a vector space help learning algorithms to achieve better performance in natural language processing tasks by grouping similar words. One of the earliest use of word representations dates back to 1986 due to Rumelhart, Hinton, and Williams [13]. This idea has since been applied to statistical language modeling with considerable success [1]. The follow up work includes applications to automatic speech recognition and machine translation [14, 7], and a wide range of NLP tas

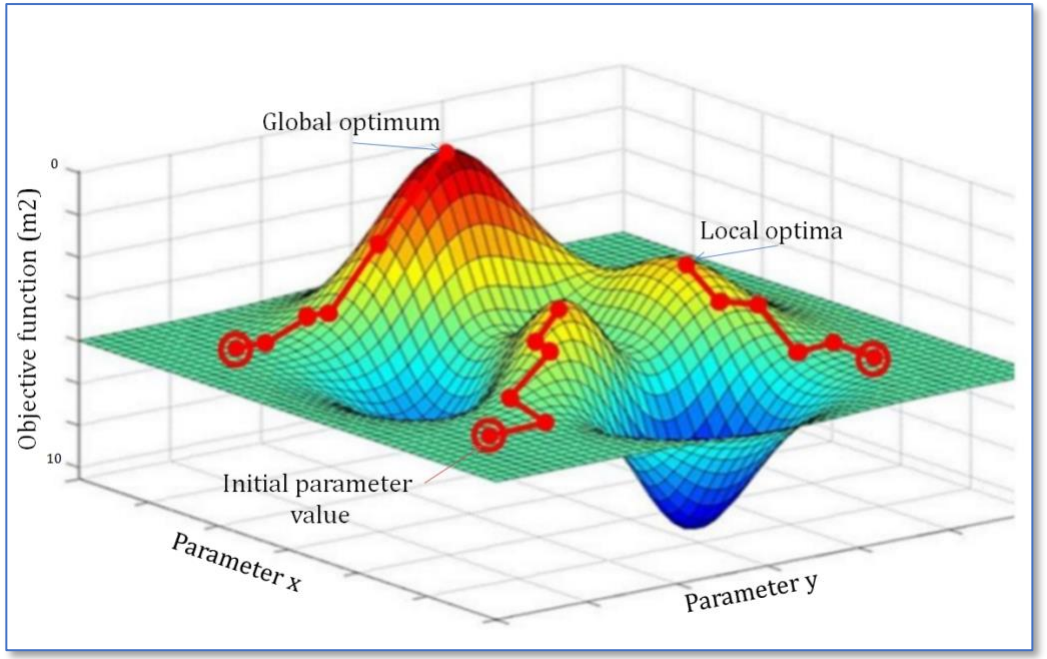
Recently, Mikolov et al.
quality vector represent
of the previously used n
gram model (see Figure
extremely efficient: an o
words in one day.

The word representation

Figure 1: The Skip-gram model architecture. The training objective is to learn word vector representations that are good at predicting the nearby words.

Gradient Descent

- Imagine walking on an “error surface” in parameter space
  - Defined by some loss function $L(\boldsymbol{\theta})$ evaluated over our training data
- Take a step in the direction where the gradient is the steepest
  - I.e., adjust parameters $\boldsymbol{\theta}$ slightly
  - Repeat until convergence, i.e., until we have the value of $\boldsymbol{\theta}$ such that L attains an optimal value for our training data
- Lots of variants exist!
  - E.g., being clever with step sizes, adding momentum terms, estimating gradients from randomly sampled mini-batches, avoiding local optima, …



Algorithm 1 Online Stochastic Gradient Descent Training

Input: Function $f(\mathbf{x};\theta)$ parameterized with parameters θ.
Input: Training set of inputs $\mathbf{x_1},\ldots,\mathbf{x_n}$ and outputs $\mathbf{y_1},\ldots,\mathbf{y_n}$.
Input: Loss function L.
while stopping criteria not met **do**
  Sample a training example $\mathbf{x_i},\mathbf{y_i}$
  Compute the loss $L(f(\mathbf{x_i};\theta),\mathbf{y_i})$
  $\hat{\mathbf{g}} \leftarrow$ gradients of $L(f(\mathbf{x_i};\theta),\mathbf{y_i})$ w.r.t θ
  $\theta \leftarrow \theta + \eta_k\hat{\mathbf{g}}$
return θ