NLP/SkipGram Model

WWC-ML/CSE ML Community

Tim Scarfe

Commercial Software Engineering (CSE)

@ecsquendor

youtube.com/machinelearningatmicrosoft

MACHINE LEARNING @ MICROSOFT

Azure

Microsoft

Microsoft

What is a word vector?

“Tim” = [0.1 0.4 0.3 … 0.2 0.9 0.2 0.1]

Microsoft

Word vectors are cool!

https://projector.tensorflow.org

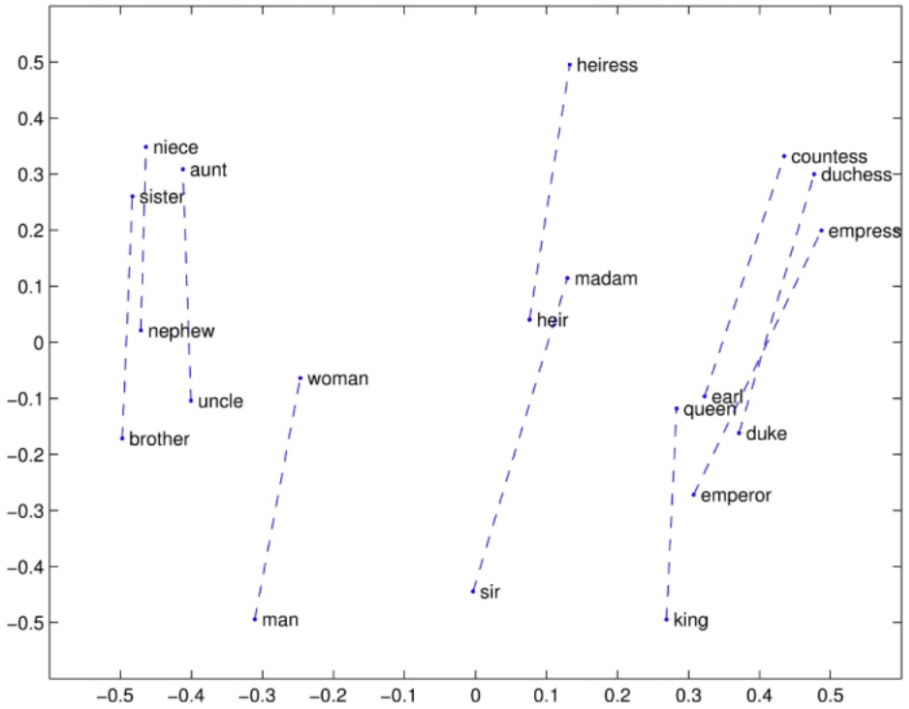
Microsoft

First names

https://projector.tensorflow.org

Microsoft

Word vector relationships

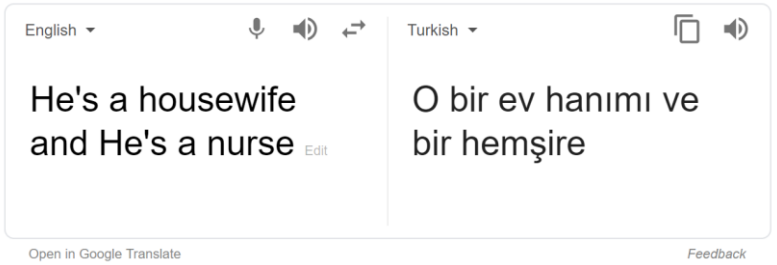


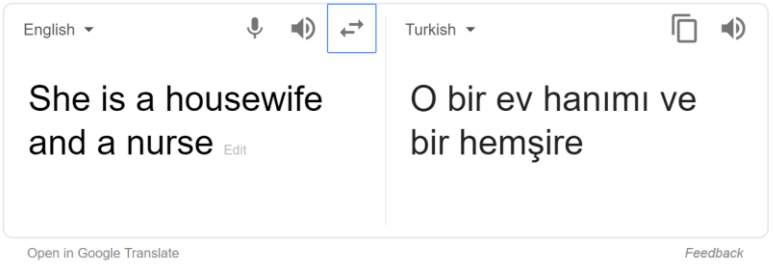
The word representations computed using neural networks are very interesting because the learned vectors explicitly encode many linguistic regularities and patterns. Somewhat surprisingly, many of these patterns can be represented as linear translations. For example, the result of a vector calculation vec("Madrid") - vec("Spain") + vec("France") is closer to vec("Paris") than to any other word vector [9, 8].

Microsoft

A note on bias in word embeddings

- word embeddings exhibit female/male gender stereotypes to a disturbing extent





Microsoft

What about Glove, ELMO etc?

Microsoft

https://arxiv.org/abs/1310.4546

Distributed Representations of Words and Phrases and their ...

https://arxiv.org › cs ▼

by T Mikolov - 2013 - Cited by 10992 - Related articles

16 Oct 2013 - **Distributed Representations of Words and Phrases and their Compositionality**. 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.

Cite as: arXiv:1310.4546

https://arxiv.org/abs/1301.3781

Efficient Estimation of Word Representations in Vector Space

https://arxiv.org › cs ▼

by T Mikolov - 2013 - Cited by 8981 - Related articles

16 Jan 2013 - **Efficient Estimation of Word Representations in Vector Space**. ... We observe large improvements in accuracy at much lower computational cost, i.e. it takes less than a day to learn high quality **word vectors** from a 1.6 billion **words** data set.

You've visited this page 3 times. Last visit: 17/01/19

Microsoft

Efficient Estimation of Word Representations in Vector Space

Tomas Mikolov
Google Inc., Mountain View, CA
tmikolov@google.com

Kai Chen
Google Inc., Mountain View, CA
kaichen@google.com

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

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

Abstract

We propose two novel model architectures for computing continuous vector representations of words from very large data sets. The quality of these representations is measured in a word similarity task, and the results are compared to the previously best performing techniques based on different types of neural networks. We observe large improvements in accuracy at much lower computational cost, i.e. it takes less than a day to learn high quality word vectors from a 1.6 billion words data set. Furthermore, we show that these vectors provide state-of-the-art performance on our test set for measuring syntactic and semantic word similarities.

Microsoft

Efficient Estimation of Word Representations in Vector Space

Abstract

We propose two novel model architectures for computing continuous vector representations of words from very large data sets. The quality of these representations is measured in a word similarity task, and the results are compared to the previously best performing techniques based on different types of neural networks. We observe large improvements in accuracy at much lower computational cost, i.e. it takes less than a day to learn high quality word vectors from a 1.6 billion words data set. Furthermore, we show that these vectors provide state-of-the-art performance on our test set for measuring syntactic and semantic word similarities.

- Introduces the Skip-Gram and the CBOW (continuous bag of words)

Microsoft

Efficient Estimation of Word Representations in Vector Space

Abstract

We propose two novel model architectures for computing continuous vector representations of words from very large data sets. The quality of these representations is measured in a word similarity task, and the results are compared to the previously best performing techniques based on different types of neural networks. We observe large improvements in accuracy at much lower computational cost, i.e. it takes less than a day to learn high quality word vectors from a 1.6 billion words data set. Furthermore, we show that these vectors provide state-of-the-art performance on our test set for measuring syntactic and semantic word similarities.

- Introduces the Skip-Gram and the CBOW (continuous bag of words)
- The results are measured
- Large improvements seen in accuracy
- Lower computational cost

Microsoft

Efficient Estimation of Word Representations in Vector Space

Abstract

We propose two novel model architectures for computing continuous vector representations of words from very large data sets. The quality of these representations is measured in a word similarity task, and the results are compared to the previously best performing techniques based on different types of neural networks. We observe large improvements in accuracy at much lower computational cost, i.e. it takes less than a day to learn high quality word vectors from a 1.6 billion words data set. Furthermore, we show that these vectors provide state-of-the-art performance on our test set for measuring syntactic and semantic word similarities.

- Introduces the Skip-Gram and the CBOW (continuous bag of words)
- The results are measured
- Large improvements seen in accuracy
- Lower computational cost
- Lower time to create embeddings
- Semantic and syntactic testing

Microsoft

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.

Microsoft

Distributed Representations of Words and Phrases and their Compositionality

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.

- Improvements to accuracy and training speed

Microsoft

Distributed Representations of Words and Phrases and their Compositionality

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.

- Improvements to accuracy and training speed
- Sub-sampling frequent words
- Negative sampling

Microsoft

Distributed Representations of Words and Phrases and their Compositionality

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.

- Improvements to accuracy and training speed
- Sub-sampling frequent words
- Introduces "Negative sampling"
- Finding phrases in text i.e. "Air Canada"

Microsoft

Introduction

1 Introduction

Many current NLP systems and techniques treat words as atomic units - there is no notion of similarity between words, as these are represented as indices in a vocabulary. This choice has several good reasons - simplicity, robustness and the observation that simple models trained on huge amounts of data outperform complex systems trained on less data. An example is the popular N-gram model used for statistical language modeling - today, it is possible to train N-grams on virtually all available data (trillions of words [3]).

However, the simple techniques are at their limits in many tasks. For example, the amount of relevant in-domain data for automatic speech recognition is limited - the performance is usually dominated by the size of high quality transcribed speech data (often just millions of words). In machine translation, the existing corpora for many languages contain only a few billions of words or less. Thus, there are situations where simple scaling up of the basic techniques will not result in any significant progress, and we have to focus on more advanced techniques.

With progress of machine learning techniques in recent years, it has become possible to train more complex models on much larger data set, and they typically outperform the simple models. Probably the most successful concept is to use distributed representations of words [10]. For example, neural network based language models significantly outperform N-gram models [1, 27, 17].

Microsoft

Introduction

1 Introduction

Many current NLP systems and techniques treat words as atomic units - there is no notion of similarity between words, as these are represented as indices in a vocabulary. This choice has several good reasons - simplicity, robustness and the observation that simple models trained on huge amounts of data outperform complex systems trained on less data. An example is the popular N-gram model used for statistical language modeling - today, it is possible to train N-grams on virtually all available data (trillions of words [3]).

However, the simple techniques are at their limits in many tasks. For example, the amount of relevant in-domain data for automatic speech recognition is limited - the performance is usually dominated by the size of high quality transcribed speech data (often just millions of words). In machine translation, the existing corpora for many languages contain only a few billions of words or less. Thus, there are situations where simple scaling up of the basic techniques will not result in any significant progress, and we have to focus on more advanced techniques.

With progress of machine learning techniques in recent years, it has become possible to train more complex models on much larger data set, and they typically outperform the simple models. Probably the most successful concept is to use distributed representations of words [10]. For example, neural network based language models significantly outperform N-gram models [1, 27, 17].

- (Most) NLP systems in industry model words atomically
- Doing so is simple and robust
- Simple models work better with less data which is why they were once better
- In-domain data is limited, so scaling up the basic methods won’t yield improvements
- Now possible to train complex models on WAY more data
- Complex models on vast data out-perform simple models
- Word vectors are hugely successful
- NNLMs significantly out-perform n-gram/tf-idf models

Microsoft

Language processing is about to explode

- NLP just had its ImageNet moment, thanks to the “Attention is all you need” and “Bidirectional transformers for language understanding” papers recently released.
- Transfer learning for NLP is here
- Coming to a paper review call soon!

BERT is conceptually simple and empirically powerful. It obtains new state-of-the-art results on eleven natural language processing tasks, including pushing the GLUE benchmark to **80.4%** (**7.6%** absolute improvement), MultiNLI accuracy to **86.7%** (**5.6%** absolute improvement) and the SQuAD v1.1 question answering Test F1 to **93.2** (**1.5** absolute improvement), outperforming human performance by **2.0**.

Microsoft

Paper goals

- Learn high quality word representations from billions of words with a vocabulary size in the millions
- Previous methods have been computationally intractable for this number of words and vocabulary size

Microsoft

Previous work

[1] Y. Bengio, R. Ducharme, P. Vincent. A neural probabilistic language model. Journal of Machine Learning Research, 3:1137-1155, 2003.

1.2 Previous Work

Representation of words as continuous vectors has a long history [10, 26, 8]. A very popular model architecture for estimating neural network language model (NNLM) was proposed in [1], where a feedforward neural network with a linear projection layer and a non-linear hidden layer was used to learn jointly the word vector representation and a statistical language model. This work has been followed by many others.

Another interesting architecture of NNLM was presented in [13, 14], where the word vectors are first learned using neural network with a single hidden layer. The word vectors are then used to train the NNLM. Thus, the word vectors are learned even without constructing the full NNLM. In this work, we directly extend this architecture, and focus just on the first step where the word vectors are learned using a simple model.

It was later shown that the word vectors can be used to significantly improve and simplify many NLP applications [4, 5, 29]. Estimation of the word vectors itself was performed using different model architectures and trained on various corpora [4, 29, 23, 19, 9], and some of the resulting word vectors were made available for future research and comparison[2]. However, as far as we know, these architectures were significantly more computationally expensive for training than the one proposed in [13], with the exception of certain version of log-bilinear model where diagonal weight matrices are used [23].

[4] R. Collobert and J. Weston. A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. In International Conference on Machine Learning, ICML, 2008.

[5] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu and P. Kuksa. Natural Language Processing (Almost) from Scratch. Journal of Machine Learning Research, 12:2493-2537, 2011.

[29] J. Turian, L. Ratinov, Y. Bengio. Word Representations: A Simple and General Method for Semi-Supervised Learning. In: Proc. Association for Computational Linguistics, 2010.

[13] T. Mikolov. Language Modeling for Speech Recognition in Czech, Masters thesis, Brno University of Technology, 2007.

[14] T. Mikolov, J. Kopecký, L. Burget, O. Glembek and J. Černocký. Neural network based language models for higly inflective languages, In: Proc. ICASSP 2009.

The classic neural language model proposed by Bengio et al. [1] in 2003 consists of a one-hidden layer feed-forward neural network that predicts the next word in a sequence as in Figure 2.

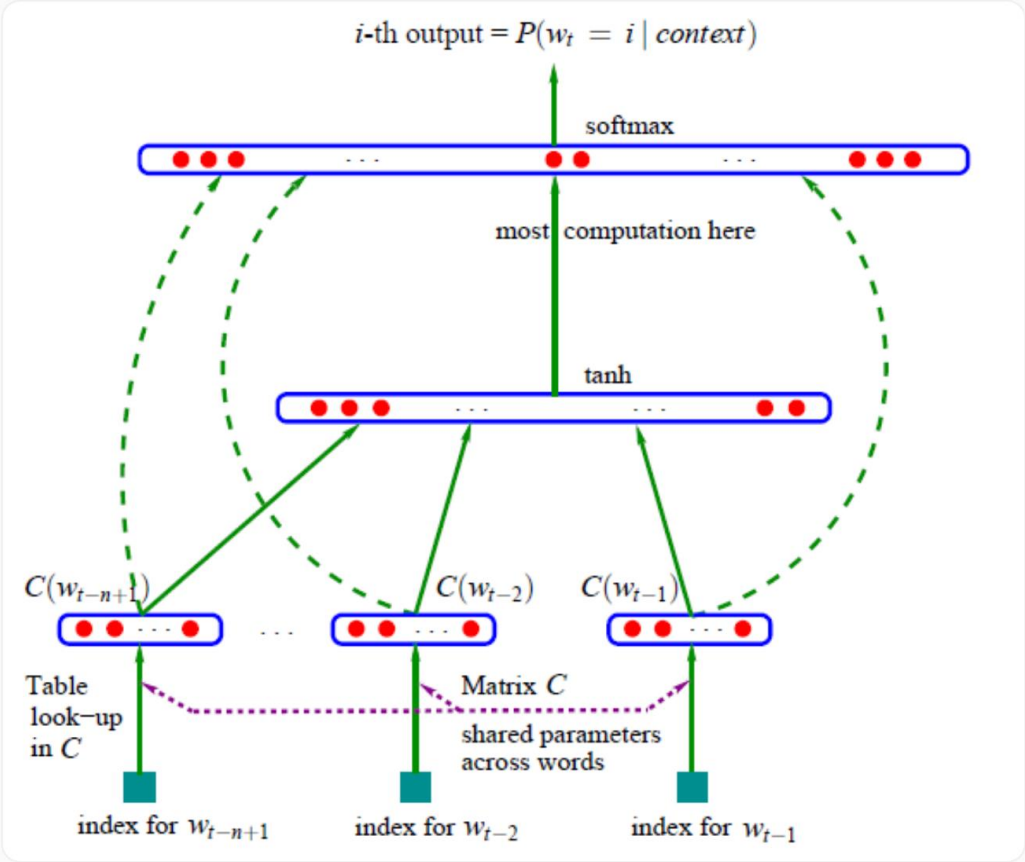


Figure 2: Classic neural language model (Bengio et al., 2003)

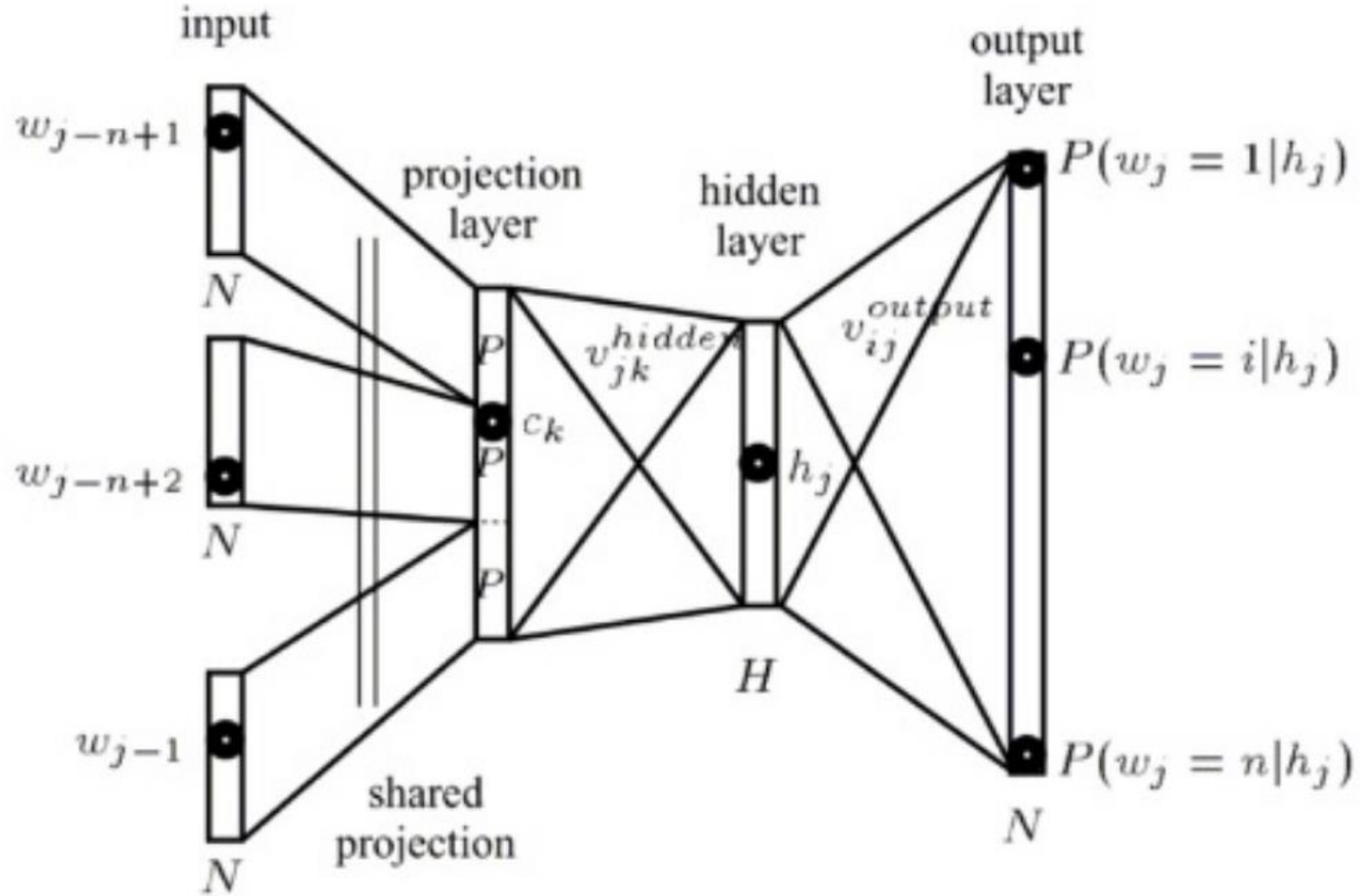
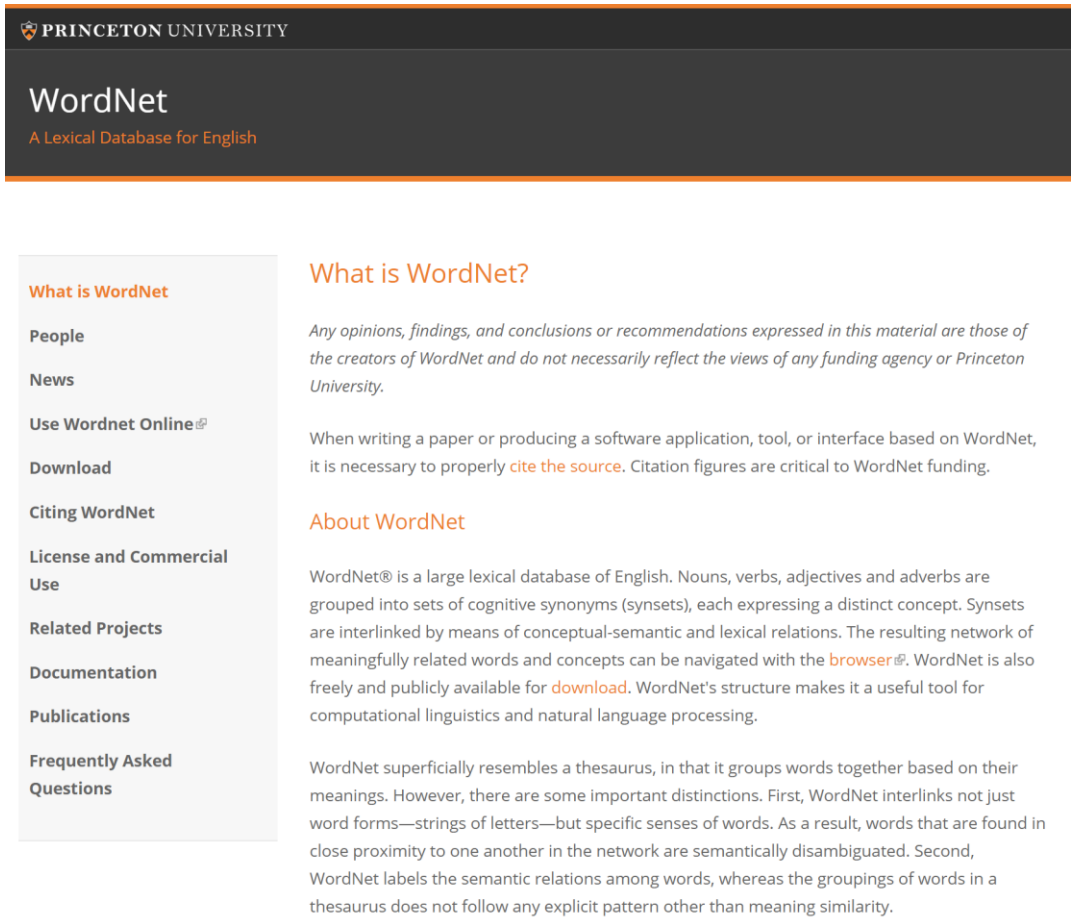


Figure 1: A neural language model (Bengio et al., 2006)

Microsoft

Capturing meaning of words in a computer (1)



PRINCETON UNIVERSITY

WordNet

A Lexical Database for English

- What is WordNet
- People
- News
- Use Wordnet Online
- Download
- Citing WordNet
- License and Commercial Use
- Related Projects
- Documentation
- Publications
- Frequently Asked Questions

What is WordNet?

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the creators of WordNet and do not necessarily reflect the views of any funding agency or Princeton University.

When writing a paper or producing a software application, tool, or interface based on WordNet, it is necessary to properly cite the source. Citation figures are critical to WordNet funding.

About WordNet

WordNet® is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. The resulting network of meaningfully related words and concepts can be navigated with the browser. WordNet is also freely and publicly available for download. WordNet's structure makes it a useful tool for computational linguistics and natural language processing.

WordNet superficially resembles a thesaurus, in that it groups words together based on their meanings. However, there are some important distinctions. First, WordNet interlinks not just word forms—strings of letters—but specific senses of words. As a result, words that are found in close proximity to one another in the network are semantically disambiguated. Second, WordNet labels the semantic relations among words, whereas the groupings of words in a thesaurus does not follow any explicit pattern other than meaning similarity.

Microsoft

Capturing meaning of words in a computer (2)

In [14]:
```
import nltk

nltk.download('wordnet')
from nltk.corpus import wordnet as wn
wn.synset('dog.n.01').hypernyms()
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Chomsky\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[14]:
```
[Synset('canine.n.02'), Synset('domestic_animal.n.01')]
```

Microsoft

Capturing meaning of words in a computer (3)

```
In [14]: import nltk

         nltk.download('wordnet')
         from nltk.corpus import wordnet as wn
         wn.synset('dog.n.01').hypernyms()

         [nltk_data] Downloading package wordnet to
         [nltk_data]     C:\Users\Chomsky\AppData\Roaming\nltk_data...
         [nltk_data]   Package wordnet is already up-to-date!

Out[14]: [Synset('canine.n.02'), Synset('domestic_animal.n.01')]
```

Problems:

- Labour intensive
- Missing synonyms
- Subjective
- No similarity score between words
- Out of date

Microsoft

Discrete representations of words (1)

- Almost all NLP approaches still use this

"one-hot representation"

Animal [0 1 0 0 0 0 0 0]

Dog [0 0 0 1 0 0 0 0]

dot (Animal, Dog) = 0

Microsoft

Discrete representations of words (2)

- “N-grams” were designed to capture some word context in a discrete representation
- Vocabulary size skyrockets
- Also “term frequencies” i.e. tf-idf
- Ideas from information retrieval to vectorise text based on the document context instead of immediate context

- Todo: draw picture

Microsoft

Discrete representations of words (3)

Demonstrate loading a small corpus into memory and "one-hot" encoding it

In [29]:
```
from keras.datasets import imdb

(x_train, y_train), (x_test, y_test) = imdb.load_data(path="imdb.npz",
                                                      num_words=1000,
                                                      skip_top=0,
                                                      maxlen=None,
                                                      seed=113,
                                                      start_char=1,
                                                      oov_char=2,
                                                      index_from=3)
```

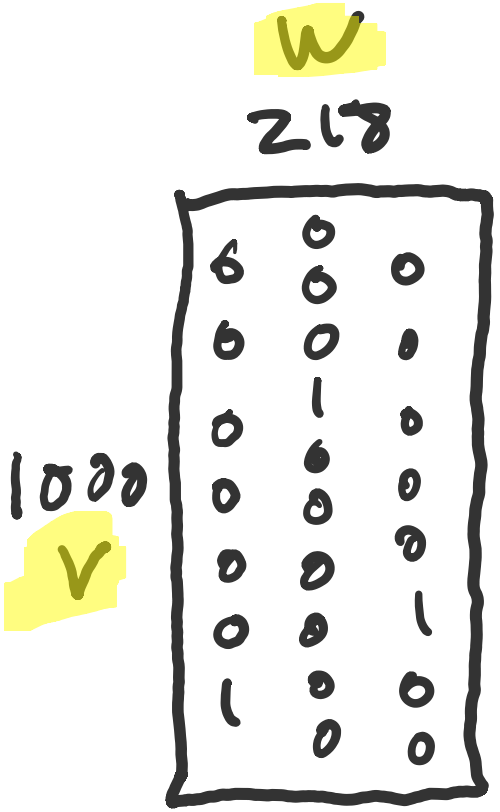
In [39]:
```
x_train[0] | take(10) | as_list
```

Out[39]:
```
[1, 14, 22, 16, 43, 530, 973, 2, 2, 65]
```

In [30]:
```
from keras.utils import to_categorical

onehot = to_categorical(x_train[0],1000)
```

In [32]:
```
onehot.shape
```

Out[32]:
```
(218, 1000)
```



onehot

"I really hated this film!"

Microsoft

Discrete representations of words (4)

In [3]:
```
review = 200
```

In [4]:
```
word_index = imdb.get_word_index()

reverse_word_index = list( word_index.items() ) \
    | select( lambda i: (i[1],i[0]) ) \
    | as_dict

decoded_review = train_data[review] \
    | select( lambda i: reverse_word_index.get(i - 3, '?') ) \
    | as_list

print( ' '.join(decoded_review) )
```

```
? this is a bit long 2 hours 20 minutes but it had a a lot of the famous ? ? novel in it in other words a lot of ? to ? br br i
t was ? ? at times but had some ? dramatic moments too ? off by a ? ? at the end of the film that was ? to view ? this film is
about ? years old the special effects ? on this film did a ? job br br paul ? and ? ? were ? ? actors in their day and they do
n't ? here both giving powerful performances the only problem is ? as all the ? are played by ? and some of them like ? ? just
don't look real i'd like to see a re make of this movie with all ? actors not for ? ? but to simply make the story look and sou
nd more ?
```

Microsoft

Discrete representations of words (5)

- Almost all NLP approaches still use the discrete approach
- No notion of similarity
- Inflectional languages like English i.e. nouns have different endings
- This is a “localised” encoding of a word i.e. not spread out
- All vectors are orthogonal to each other and have a dot product (kind of similarity score) of zero
- Vocab sizes (high dimensionality)
  - 20K standard speech
  - 400K/6B Wikipedia corpus (2014)
  - 2.2M/840B common crawl corpus
  - 13M Google 1TB corpus

Advances in Pre-Training Distributed Word Representations — Mivolov ‘18

Corpus	Size [billion]
Wikipedia meta-pages	9.2
Statmt.org News	4.2
UMBC News	3.2
Gigaword	3.3
Common Crawl	630

Table 1: Training corpora and their size in billions of words after tokenization and sentence de-duplication.

Microsoft

Representing words by their similarity

- Core idea: **A word's meaning is given by the words that frequently appear close-by**
  - *"You shall know a word by the company it keeps"* (J. R. Firth 1957: 11)
  - One of the most successful ideas of modern statistical NLP!
- When a word *w* appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).
- Use the many contexts of *w* to build up a representation of *w*

…government debt problems turning into **banking** crises as happened in 2009…
…saying that Europe needs unified **banking** regulation to replace the hodgepodge…
…India has just given its **banking** system a shot in the arm…

These context words will represent ***banking***

http://cs224d.stanford.edu

Microsoft

Model architectures

https://medium.com/nanonets/topic-modeling-with-lsa-psla-lda-and-lda2vec-555ff65b0b05

2 Model Architectures

Many different types of models were proposed for estimating continuous representations of words, including the well-known Latent Semantic Analysis (LSA) and Latent Dirichlet Allocation (LDA). In this paper, we focus on distributed representations of words learned by neural networks, as it was previously shown that they perform significantly better than LSA for preserving linear regularities among words [20, 31]; LDA moreover becomes computationally very expensive on large data sets.

Similar to [18], to compare different model architectures we define first the computational complexity of a model as the number of parameters that need to be accessed to fully train the model. Next, we will try to maximize the accuracy, while minimizing the computational complexity.

For all the following models, the training complexity is proportional to

$$O = E \times T \times Q, \tag{1}$$

where E is number of the training epochs, T is the number of the words in the training set and Q is defined further for each model architecture. Common choice is $E = 3 - 50$ and T up to one billion. All models are trained using stochastic gradient descent and backpropagation [26].

Microsoft

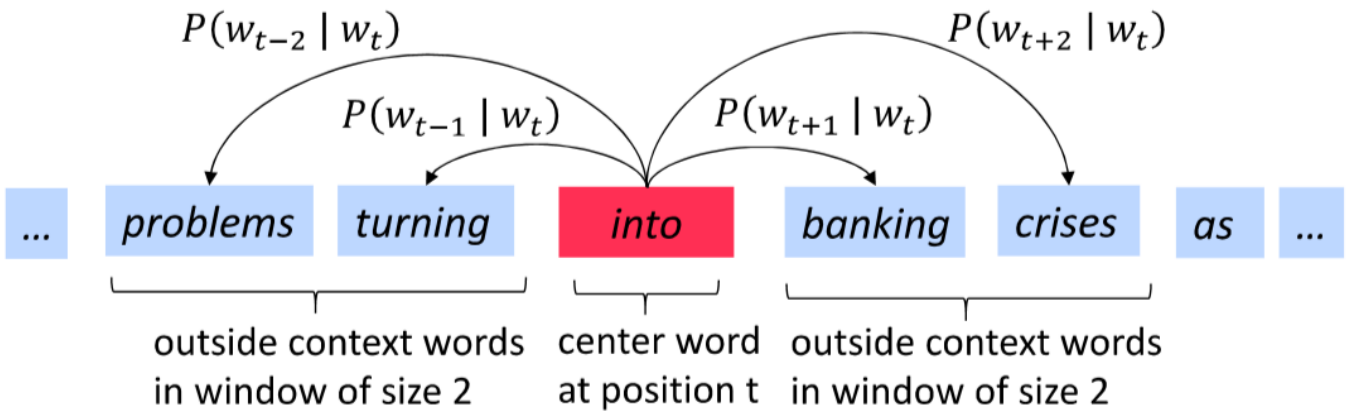
Skip-Gram model overview

- For a corpus of text
- Every word in the dictionary represented by an atomic vector (1-h)
- Go through the text word by word, select centre word and context words
- Use the similarity of word and context words to compute the probability of the context given the word and vice versa
- Keep adjusting word vectors to maximise this probability

Microsoft

Skip-Gram overview (2)

- Example windows and process for computing $P(w_{t+j} \mid w_t)$

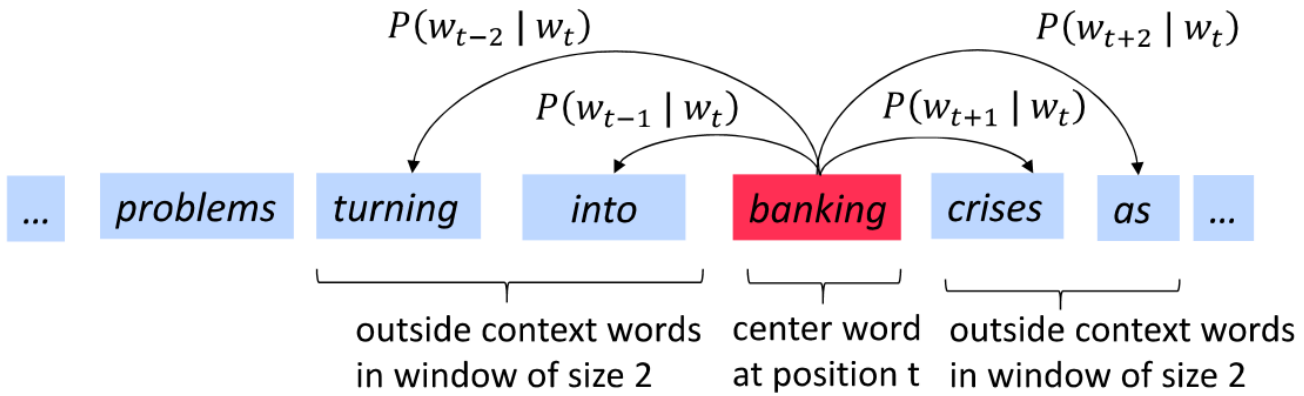


http://cs224d.stanford.edu

Microsoft

Skip-Gram overview (3)

- Example windows and process for computing $P(w_{t+j} \mid w_t)$



http://cs224d.stanford.edu

Microsoft

Skip-gram objective function

For each position $t = 1, \ldots, T$, predict context words within a window of fixed size m, given center word w_j.

$$L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P(w_{t+j} \mid w_t; \theta)$$

Likelihood =

θ is all variables to be optimized

sometimes called *cost* or *loss* function

The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P(w_{t+j} \mid w_t; \theta)$$

Minimizing objective function ⟺ Maximizing predictive accuracy

http://cs224d.stanford.edu

Microsoft

Softmax probability

- We want to minimize the objective function:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m \\ j\neq 0}} \log P(w_{t+j} \mid w_t; \theta)$$

- Question: How to calculate $P(w_{t+j} \mid w_t; \theta)$?
- Answer: We will *use two* vectors per word *w*:
  - v_w when *w* is a center word
  - u_w when *w* is a context word
- Then for a center word *c* and a context word *o*:

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w\in V}\exp(u_w^T v_c)}$$

http://cs224d.stanford.edu

Microsoft

Hierarchical softmax

Hierarchical softmax (H-Softmax) is an approximation inspired by binary trees that was proposed by Morin and Bengio (2005) [3]. H-Softmax essentially replaces the flat softmax layer with a hierarchical layer that has the words as leaves, as can be seen in Figure 1.

This allows us to decompose calculating the probability of one word into a sequence of probability calculations, which saves us from having to calculate the expensive normalization over all words. Replacing a softmax layer with H-Softmax can yield speedups for word prediction tasks of at least 50× and is thus critical for low-latency tasks such as real-time communication in Google's new messenger app Allo.

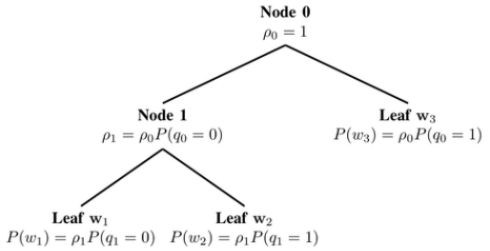


Figure 1: Hierarchical softmax (Quora)

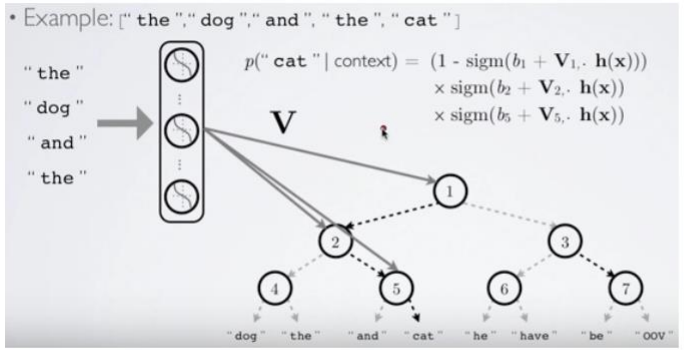
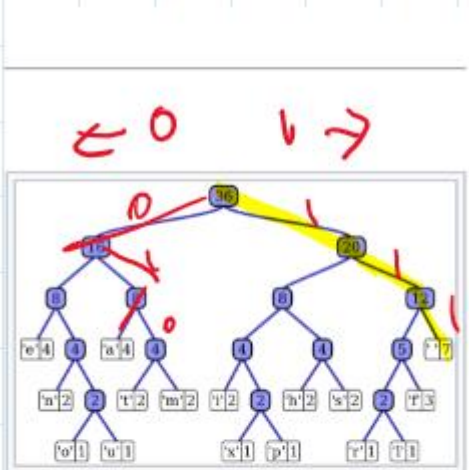


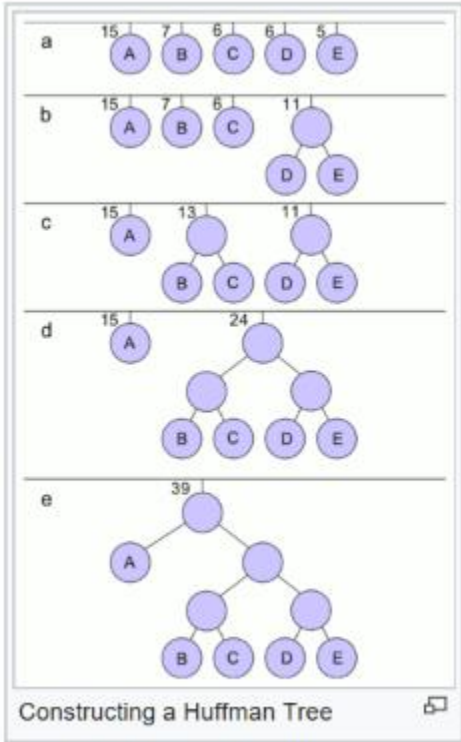
Figure 2: Hierarchical softmax computations (Hugo Lachorelle's Youtube lectures)

Hierarchical softmax (2)



Huffman tree generated from the exact frequencies of the text "this is an example of a huffman tree". The frequencies and codes of each character are below. Encoding the sentence with this code requires 195 (or 147) bits, as opposed to 288 (or 180) bits if 36 characters of 8 (or 5) bits were used. (This assumes that the code tree structure is known to the decoder and thus does not need to be counted as part of the transmitted information.)

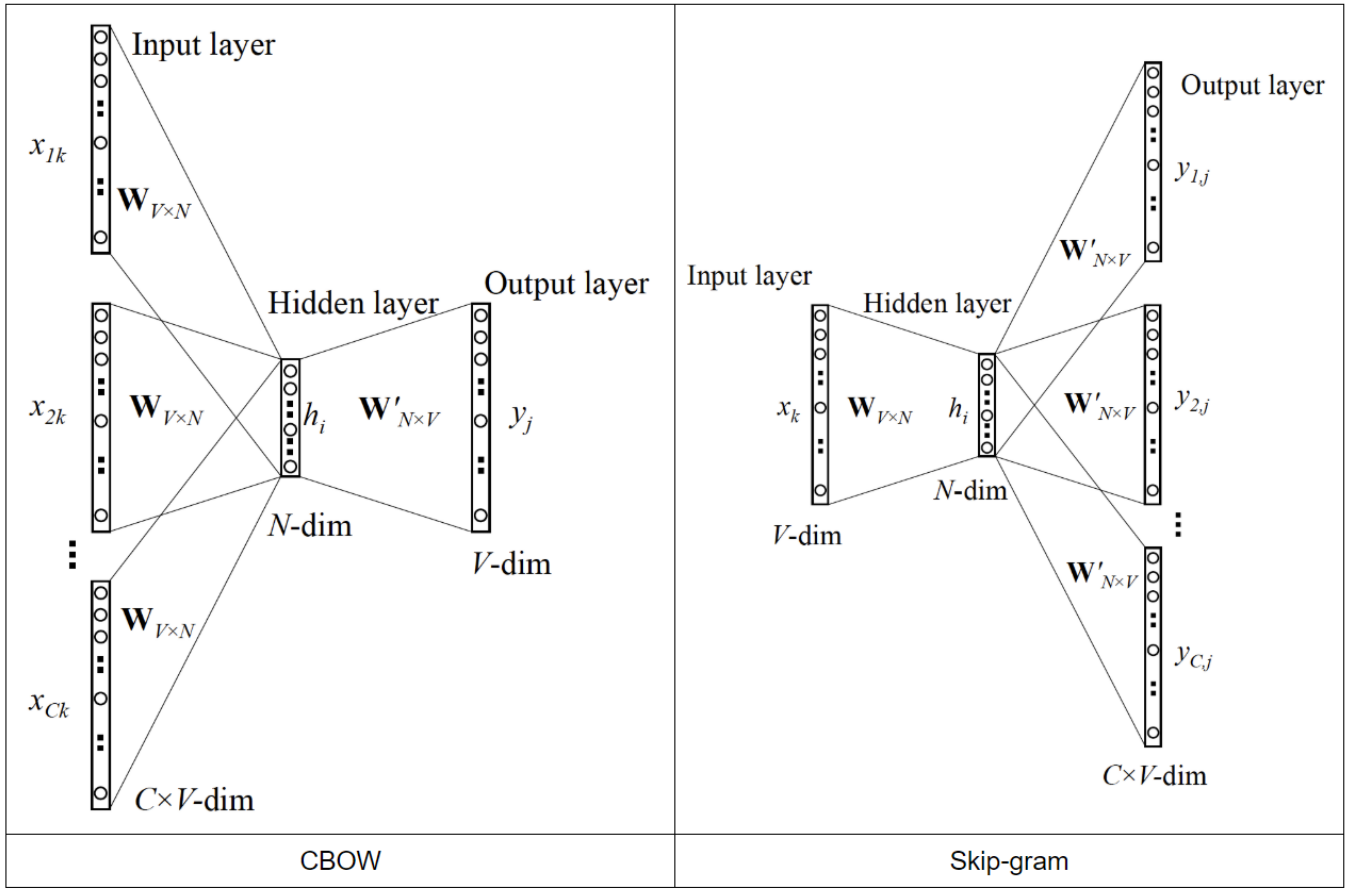
Char	Freq	Code
space	7	111
a	4	010
e	4	000
f	3	1101
h	2	1010
i	2	1000
m	2	0111
n	2	0010
s	2	1011
t	2	0110
l	1	11001
o	1	00110
p	1	10011
r	1	11000
u	1	00111
x	1	10010



Constructing a Huffman Tree

Microsoft

CBOW vs Skipgram



http://www.cs.nthu.edu.tw/~shwu/courses/ml/labs/10_Keras_Word2Vec/10_Keras_Word2Vec.html

Microsoft

CBOW vs Skipgram (from paper)

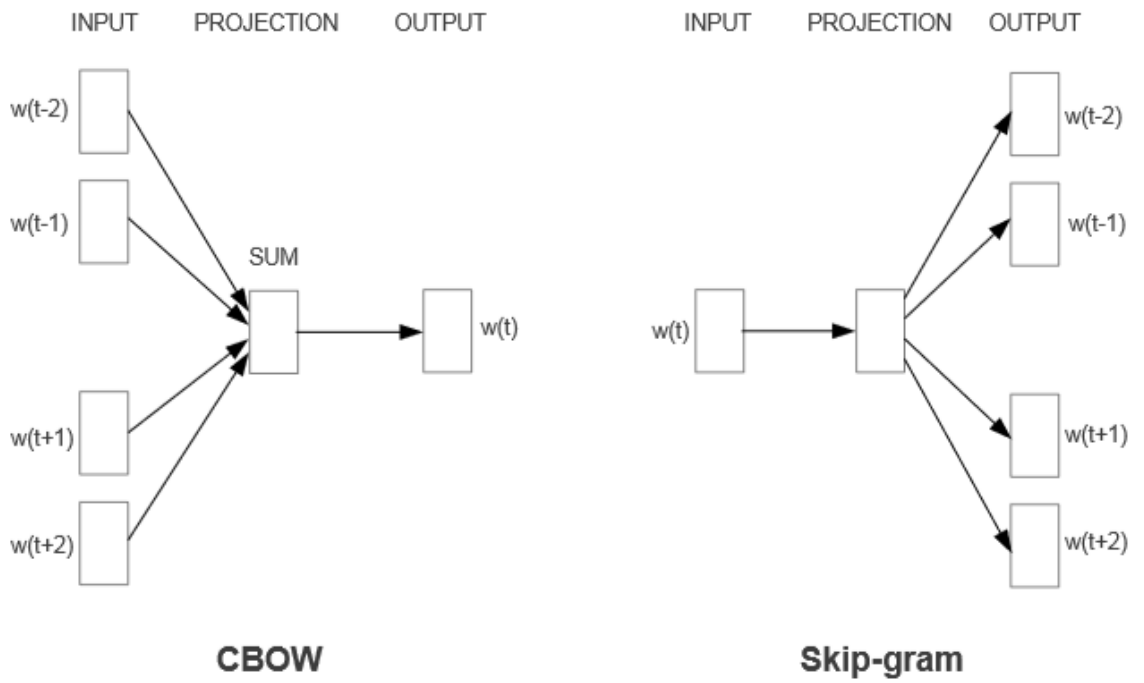
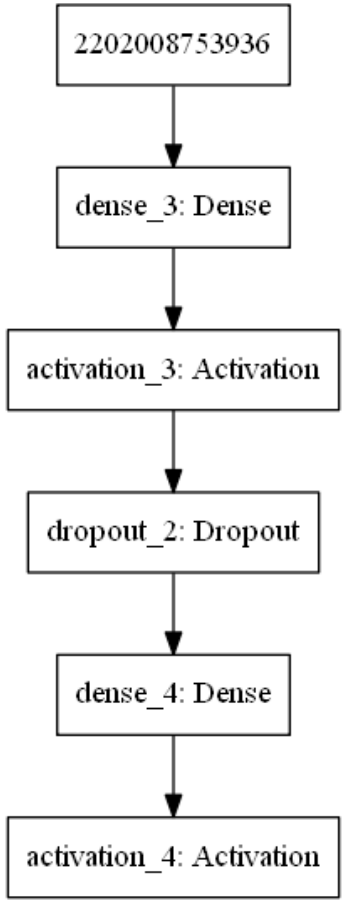


Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

Microsoft

Trivial skipgram with softmax



In [35]:
```
ohe = OneHotEncoder(n_values=vocab_size)
X = ohe.fit_transform(np.array(xs).reshape(-1, 1)).todense()
Y = ohe.fit_transform(np.array(ys).reshape(-1, 1)).todense()
Xtrain, Xtest, Ytrain, Ytest = train_test_split(X, Y, test_size=0.3,
                                                random_state=42)

print(Xtrain.shape, Xtest.shape, Ytrain.shape, Ytest.shape)

model = Sequential()
model.add(Dense(300, input_shape=(vocab_size,)))
model.add(Activation("relu"))
model.add(Dropout(0.7))
model.add(Dense(vocab_size))
model.add(Activation("softmax"))

model.compile(optimizer="rmsprop", loss="categorical_crossentropy",
              metrics=["accuracy"])

history = model.fit(Xtrain, Ytrain, batch_size=BATCH_SIZE,
                    epochs=NUM_EPOCHS, verbose=1,
                    validation_data=(Xtest, Ytest))
```

```
(38525, 3338) (16511, 3338) (38525, 3338) (16511, 3338)
Train on 38525 samples, validate on 16511 samples
Epoch 1/20
38525/38525 [==============================] - 4s 110us/step - loss: 6.4479
Epoch 2/20
38525/38525 [==============================] - 4s 104us/step - loss: 5.9694
```

Microsoft

Skipgram with negative sampling

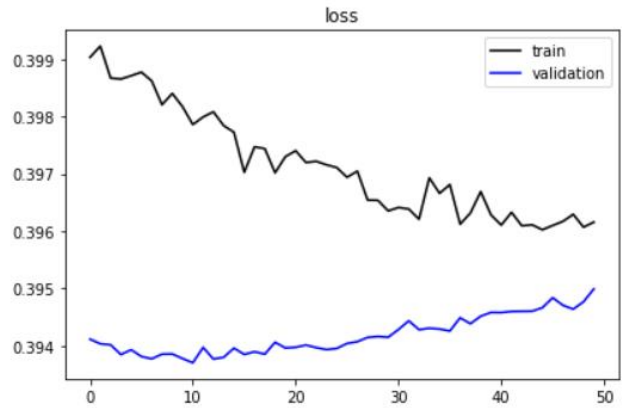
In [9]:
```
from keras.layers import Activation, Embedding, dot, Reshape, Input
from keras.models import Sequential, Model

input_targetword = Input(shape=(vocab_size,))
inputs_context = Input(shape=(vocab_size,))
target_word = Dense(30)(input_targetword)
context = Dense(30)(inputs_context)

drop_context = Dropout(0.4)(context)
drop_target = Dropout(0.4)(target_word)

ddot = dot([drop_target, drop_context], 1)
reshape = Reshape((1,), input_shape=(1, 1))(ddot)
net_output = Activation('sigmoid')(reshape)

model = Model(inputs=[input_targetword, inputs_context], outputs=[net_output])

model.compile(loss='binary_crossentropy', optimizer='adam')

model.summary()
```

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 1000)	0	
input_2 (InputLayer)	(None, 1000)	0	
dense_1 (Dense)	(None, 30)	30030	input_1[0][0]
dense_2 (Dense)	(None, 30)	30030	input_2[0][0]
dropout_2 (Dropout)	(None, 30)	0	dense_1[0][0]
dropout_1 (Dropout)	(None, 30)	0	dense_2[0][0]
dot_1 (Dot)	(None, 1)	0	dropout_2[0][0] dropout_1[0][0]
reshape_1 (Reshape)	(None, 1)	0	dot_1[0][0]
activation_1 (Activation)	(None, 1)	0	reshape_1[0][0]

Total params: 60,060
Trainable params: 60,060
Non-trainable params: 0

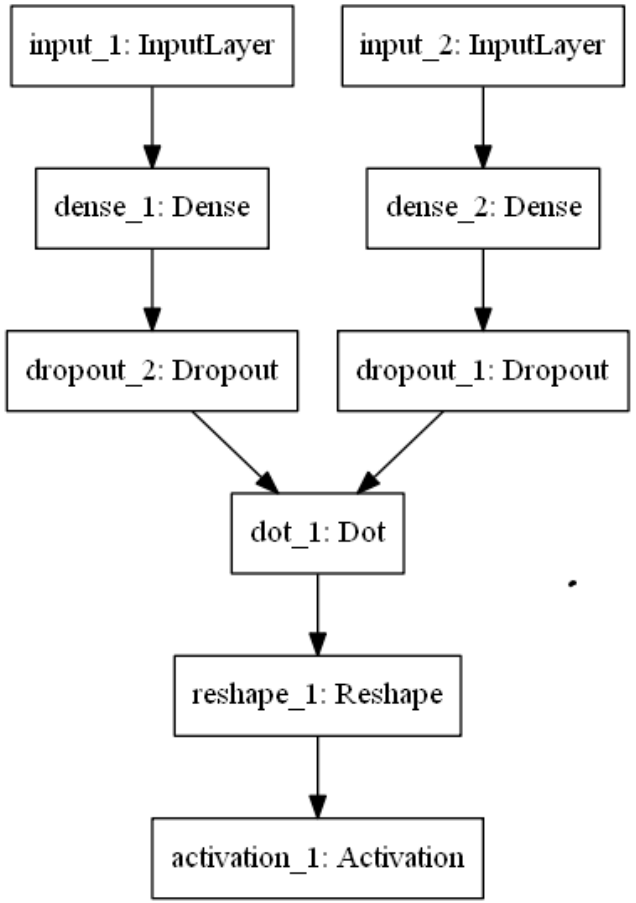
In [71]:
```
%%time
history = model.fit([Wtrain, Ctrain], Ytrain, batch_size=500,
                    epochs=50, verbose=1,
                    validation_data=([Wtest, Ctest], Ytest))
```

```
Train on 705980 samples, validate on 302564 samples
Epoch 1/50
705980/705980 [==============================] - 21s 30us/step - loss: 0.3990 - val_loss: 0.3941
Epoch 2/50
705980/705980 [==============================] - 21s 30us/step - loss: 0.3992 - val_loss: 0.3940
Epoch 3/50
705980/705980 [==============================] - 21s 30us/step - loss: 0.3987 - val_loss: 0.3940
Epoch 4/50
705980/705980 [==============================] - 21s 30us/step - loss: 0.3987 - val_loss: 0.3938
Epoch 5/50
705980/705980 [==============================] - 21s 30us/step - loss: 0.3987 - val_loss: 0.3939
Epoch 6/50
```



Microsoft

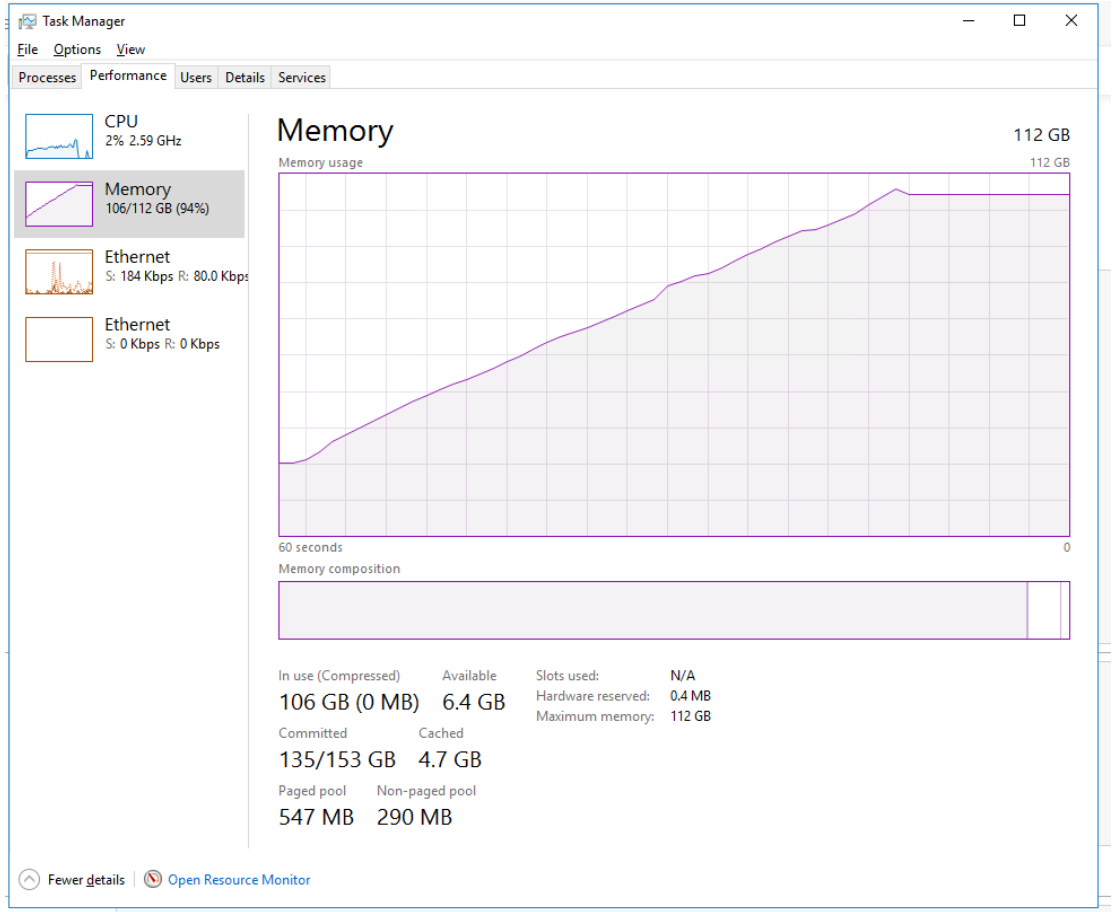
Skipgram with negative sampling (2)



Microsoft

Don’t one-hot encode everything at once unless you have enough memory!

```
In [40]: ohe = OneHotEncoder(n_values=vocab_size)
         X = ohe.fit_transform(np.array(xs).reshape(-1, 1)).todense()
         Y = ohe.fit_transform(np.array(ys).reshape(-1, 1)).todense()
         Xtrain, Xtest, Ytrain, Ytest = train_test_split(X, Y, test_size=0.3,
                                                         random_state=42)

         print(Xtrain.shape, Xtest.shape, Ytrain.shape, Ytest.shape)

         (599048, 3338) (256736, 3338) (599048, 3338) (256736, 3338)
```



Use lazy evaluated streams and mPyPl !
https://pypi.org/project/mPyPl/

Microsoft

You don’t have to implement it yourself

```
import gensim

# preprocess the text
corpus = []
for text in newsgroups.data:
    corpus.append(tokenizer_stem_nostop(preprocessor(text)))

# size : embed dimension
# min_count : filter words without min frequency
# sg : 0 for CBOW; 1 for skip-gram
# negative : how many noise words should be drawn
%time model_w2v = gensim.models.Word2Vec(corpus, size=64, min_count=5, sg=1, negative=5, workers=2)
```

```
CPU times: user 23.3 s, sys: 205 ms, total: 23.5 s
Wall time: 13.3 s
```

http://www.cs.nthu.edu.tw/~shwu/courses/ml/labs/10_Keras_Word2Vec/10_Keras_Word2Vec.html

Microsoft

You don’t have to implement it yourself (2)

```
display(model_w2v.most_similar('car'))
display(model_w2v.most_similar('baseball'))
display(model_w2v.most_similar('electronic'))
display(model_w2v.most_similar('american'))
```

```
[('dealer', 0.8815475702285767),
 ('cars', 0.8629363179206848),
 ('carnum', 0.8581898212432861),
 ('auto', 0.8568898439407349),
 ('tires', 0.8530106544494629),
 ('toyota', 0.8468404412269592),
 ('nissan', 0.837895929813385),
 ('taurus', 0.8325661420822144),
 ('driving', 0.8298842906951904),
 ('ford', 0.8290390968322754)]

[('football', 0.8473756313323975),
 ('hockey', 0.8451310396194458),
 ('manager', 0.8335633277893066),
 ('basketball', 0.8320602178573608),
 ('pro', 0.8268144130706787),
 ('fans', 0.8199853897094727),
 ('hockeynum', 0.8151143789291382),
 ('hall', 0.8130113482475281),
 ('predictions', 0.8118510246276855),
 ('stat', 0.8079237341880798)]
```

Microsoft

Advances in Pre-Training Distributed Word Representations

Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, Armand Joulin
Facebook AI Research
{tmikolov, egrave, bojanowski, cpuhrsch, ajoulin}@fb.com

Abstract

Many Natural Language Processing applications nowadays rely on pre-trained word representations estimated from large text corpora such as news collections, Wikipedia and Web Crawl. In this paper, we show how to train high-quality word vector representations by using a combination of known tricks that are however rarely used together. The main result of our work is the new set of publicly available pre-trained models that outperform the current state of the art by a large margin on a number of tasks.

Keywords: fastText, word2vec, word vectors, pre-trained

2.1. Standard cbow model

The cbow model as used in Mikolov et al. (2013a) learns

Word subsampling.

2.3. Phrase representations

2.2. Position-dependent Weighting

2.4. Subword information

Model	Sem	Syn	Tot
cbow + uniq	79	73	76
cbow + uniq + phrases	82	78	80
cbow + uniq + phrases + weighting	87	82	85

Table 2: Accuracies on semantic and syntactic analogy datasets for models trained on Common Crawl (630B words). By performing sentence-level de-duplication, adding position-dependent weighting and phrases, the model quality improves significantly.

https://arxiv.org/abs/1712.09405

Microsoft

Use pre-built word vectors

Model	Analogy	RW	Squad
GloVe Wiki + news	72	0.38	77.7%
fastText Wiki + news	**87**	0.50	78.8%
GloVe Crawl	75	0.52	78.9%
fastText Crawl	85	**0.58**	**79.8%**

Table 3: Results on Word Analogy, Rare Words and Squad datasets with fastText models trained on various corpora (see Table 1) or Common Crawl (see Table 2), and comparison to GloVe models trained on comparable datasets.

- https://fasttext.cc/docs/en/english-vectors.html

English word vectors

This page gathers several pre-trained word vectors trained using fastText.

Download pre-trained word vectors

Pre-trained word vectors learned on different sources can be downloaded below:

1. wiki-news-300d-1M.vec.zip: 1 million word vectors trained on Wikipedia 2017, UMBC webbase corpus and statmt.org news dataset (16B tokens).
2. wiki-news-300d-1M-subword.vec.zip: 1 million word vectors trained with subword infomation on Wikipedia 2017, UMBC webbase corpus and statmt.org news dataset (16B tokens).
3. crawl-300d-2M.vec.zip: 2 million word vectors trained on Common Crawl (600B tokens).
4. crawl-300d-2M-subword.zip: 2 million word vectors trained with subword information on Common Crawl (600B tokens).

```
@inproceedings{mikolov2018advances,
  title={Advances in Pre-Training Distributed Word Representations},
  author={Mikolov, Tomas and Grave, Edouard and Bojanowski, Piotr and Puhrsch, Christian and Joulin, Armand},
  booktitle={Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)},
  year={2018}
}
```

Microsoft

Improving Distributional Similarity with Lessons Learned from Word Embeddings

Omer Levy **Yoav Goldberg** **Ido Dagan**
Computer Science Department
Bar-Ilan University
Ramat-Gan, Israel
{omerlevy,yogo,dagan}@cs.biu.ac.il

Levy et al. find that SVD — and not one of the word embedding algorithms — performs best on similarity tasks, while SGNS performs best on analogy datasets. They furthermore shed light on the importance of hyperparameters compared to other choices:

1. Hyperparameters vs. algorithms:
   Hyperparameter settings are often more important than algorithm choice.
   No single algorithm consistently outperforms the other methods.

2. Hyperparameters vs. more data:
   Training on a larger corpus helps for some tasks.
   In 3 out of 6 cases, tuning hyperparameters is more beneficial.

Microsoft

2.3 Subsampling of Frequent Words

In very large corpora, the most frequent words can easily occur hundreds of millions of times (e.g., "in", "the", and "a"). Such words usually provide less information value than the rare words. For example, while the Skip-gram model benefits from observing the co-occurrences of "France" and "Paris", it benefits much less from observing the frequent co-occurrences of "France" and "the", as nearly every word co-occurs frequently within a sentence with "the". This idea can also be applied in the opposite direction; the vector representations of frequent words do not change significantly after training on several million examples.

To counter the imbalance between the rare and frequent words, we used a simple subsampling approach: each word w_i in the training set is discarded with probability computed by the formula

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}} \tag{5}$$

Microsoft

Phrases

This way, we can form many reasonable phrases without greatly increasing the size of the vocabulary; in theory, we can train the Skip-gram model using all n-grams, but that would be too memory intensive. Many techniques have been previously developed to identify phrases in the text; however, it is out of scope of our work to compare them. We decided to use a simple data-driven approach, where phrases are formed based on the unigram and bigram counts, using

$$\text{score}(w_i, w_j) = \frac{\text{count}(w_i w_j) - \delta}{\text{count}(w_i) \times \text{count}(w_j)}. \quad (6)$$

The δ is used as a discounting coefficient and prevents too many phrases consisting of very infrequent words to be formed. The bigrams with score above the chosen threshold are then used as phrases. Typically, we run 2-4 passes over the training data with decreasing threshold value, allowing longer phrases that consists of several words to be formed. We evaluate the quality of the phrase representations using a new analogical reasoning task that involves phrases. Table 2 shows examples of the five categories of analogies used in this task. This dataset is publicly available on the web[2].

Microsoft

Materials/Links

- http://web.stanford.edu/class/cs224n/
- https://www.manning.com/books/deep-learning-with-python
- https://arxiv.org/abs/1310.4546
- https://arxiv.org/abs/1301.3781

Tim Scarfe

@ecsquendor

youtube.com/machinelearningatmicrosoft

THANK YOU!

Azure

Microsoft