Explaining & Exploring word2vec

Universität des Saarlandes
November 2017
Andrew Johnson

The original paper
(Sept. 2013)

The follow-up paper
(Oct. 2013)
Not covered here

https://arxiv.org/pdf/1310.4546.pdf

https://arxiv.org/pdf/1301.3781.pdf

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

Distributed Representations of Words and Phrases and their Compositionality

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

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

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

Abstract

ous Skip-gram model is an efficient method for
vector representations that capture a large num-
ntic word relationships. In this paper we present
both the quality of the vectors and the training
equent words we obtain significant speedup and
resentations. We also describe a simple alterna-
alled negative sampling.

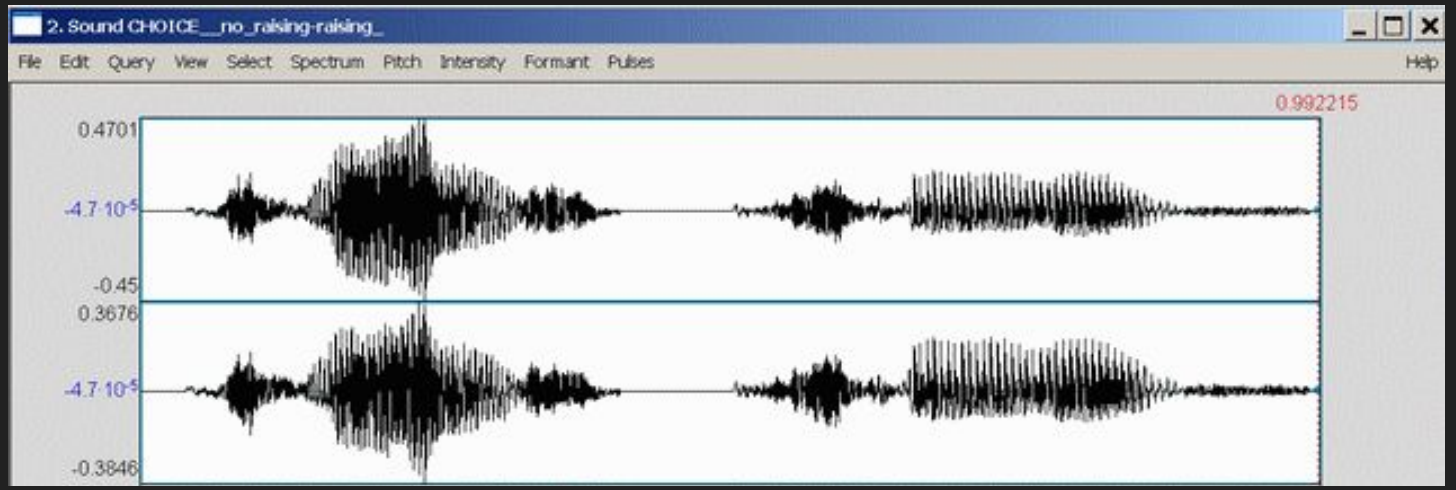
epresentations is their indifference to word order
diomatic phrases. For example, the meanings of
sily combined to obtain "Air Canada". Motivated
mple method for finding phrases in text, and show
ntations for millions of phrases is possible.

vector space help learning algorithms to achieve better

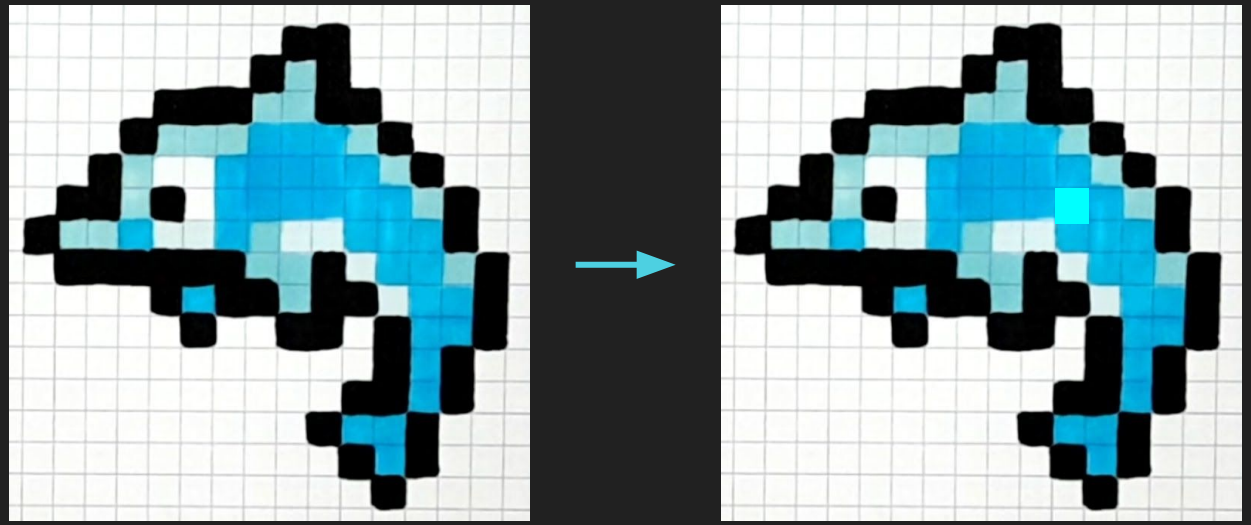
Outline

- Why word vectors?
  - Why word to vec?
- Training complexity
- Previous Approaches
  - NNLM
    - 1-of-V encoding
    - Hierarchical softmax
  - RNNLM
- New Approaches
  - CBOW
  - Skip-gram
  - Similarity Arithmetic
- Evaluation
  - Results
  - Further improvements
  - Playing around with word2vec

Audio is continuous …



Images are continuous …



… but text is discrete

Homemade Crab Dip → Homemade Crap Dip

Why use word embeddings?

Can calculate similarity between terms

To incrementally adjust (eg. training a NN)

Applications for MT, IR, QA, NLG, …

What can be done with continuous data

5 epochs

100 epochs

Why word2vec? (Answer: Big Data)

	Previous approaches	Their target
Dimensionality	50 to 100	600+
Training words	a few 100 million	Billions

Goal 1: Maximize Accuracy

Goal 2: Minmize computational cost

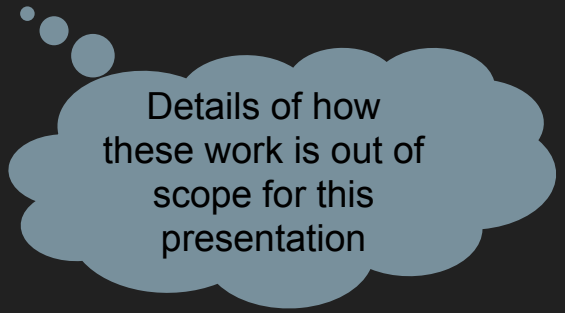
Previous Approaches for Estimating Continuous Representations

Not Neural Network-based:

- Latent Semantic Analysis (LSA) - *Significantly worse performance*
- Linear Dirichlet Allocation (LDA) - *Computationally very expensive*

Neural Network-based:

- Feedforward Neural Network Language Model
- Recursive Neural Network Language Model



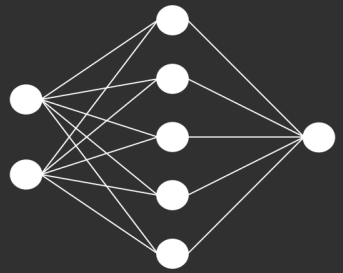
Training Complexity

is proportional to:

$$O = E \times T \times Q$$

Number of epochs

Number of training words

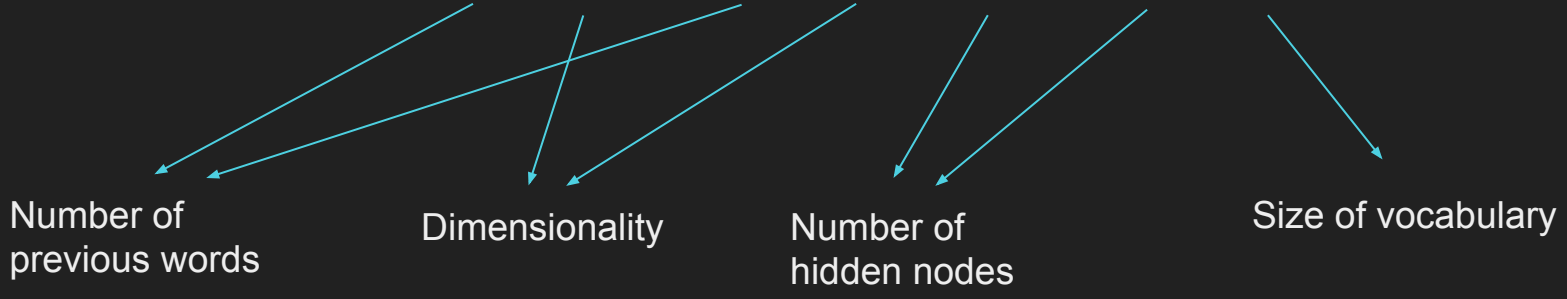


Complexity per training example
(depends on model architecture)

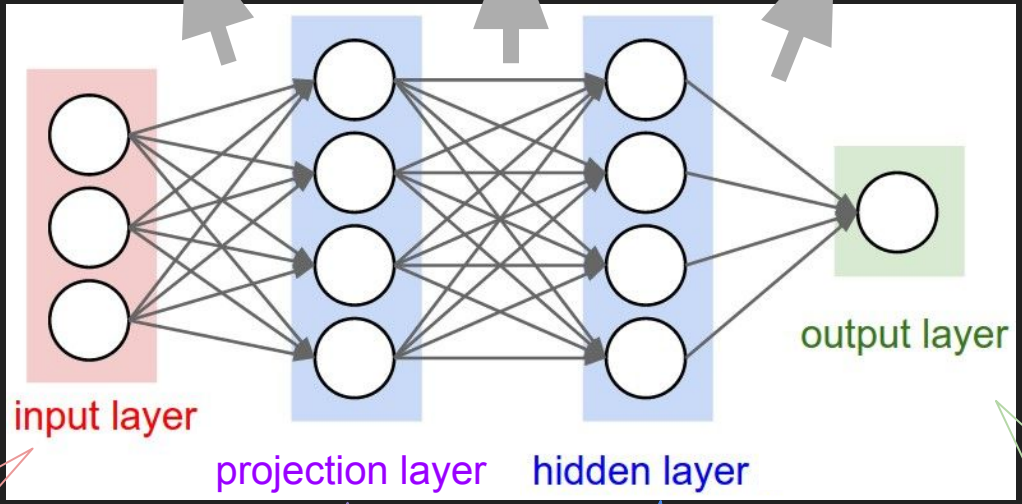
NNLM: Q Value

Probabilistic feedforward neural network language model

$$Q = (N \times D) + (N \times D \times H) + (H \times V)$$



$$Q = (N \times D) + (N \times D \times H) + (H \times V)$$



(output not just one node)

1-of-V encoding over N previous words

Dimensionality N×D

Dense connection to previous layer

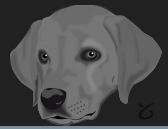
Dimensionality V

One-of-V Encoding:

Aka “One hot encoding”, example:

V = { it, is, raining, cats, and, dogs, today }

[0 0 0 0 0 1 0]



Encoding for the word “dogs”

But math operations are meanignless, eg:

$V_{dog} \cdot V_{cat}$

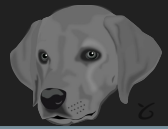
= [0,0,0,0,0,1,0]·[0,0,0,1,0,0,0]
= [0,0,0,0,0,0,0]

One-of-V Encoding:

Aka “One hot encoding”, example:

V = { it, is, raining, cats, and, dogs, today }

0	0	0	0	0	1	0



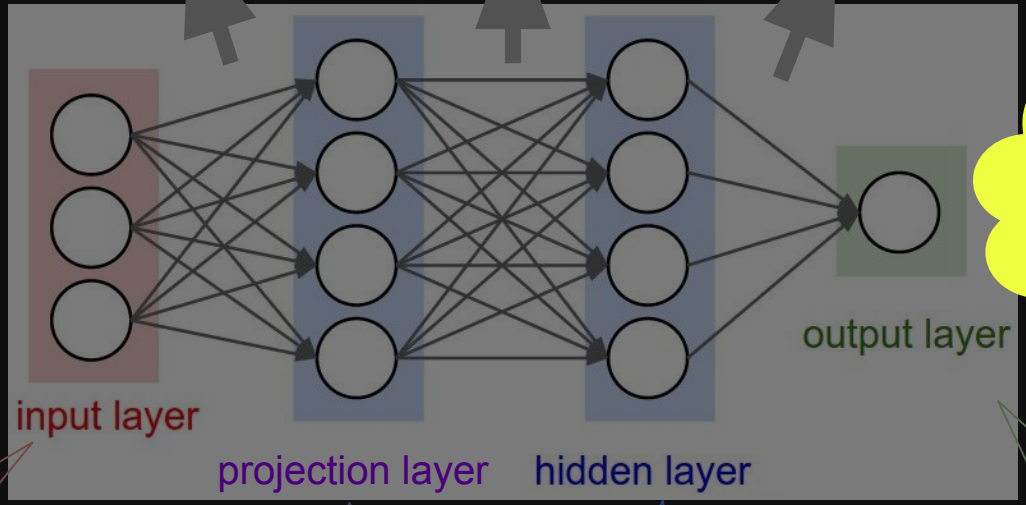
Encoding for the word “dogs”

Compare to “distributed representation” (word vec):

0.02	0.12	0.01	0.22	0.47	0.88	0.79
0.31	0.46	0.25	0.18	0.25	0.08	0.14

...

$$Q = (N \times D) + (N \times D \times H) + (H \times V)$$



Which is the "computational bottleneck"?

1 hot ("1-of-V") encoding over N previous words

Dimensionality N×D

Dense connection to previous layer

Dimensionality V

NNLM: Dominating Term

$$Q = (N \times D) + (N \times D \times H) + (H \times V)$$

But there are remedies:

- Using hierarchical softmax
- Avoiding normalization during training
- Representing vocab as huffman tree

Activation Functions

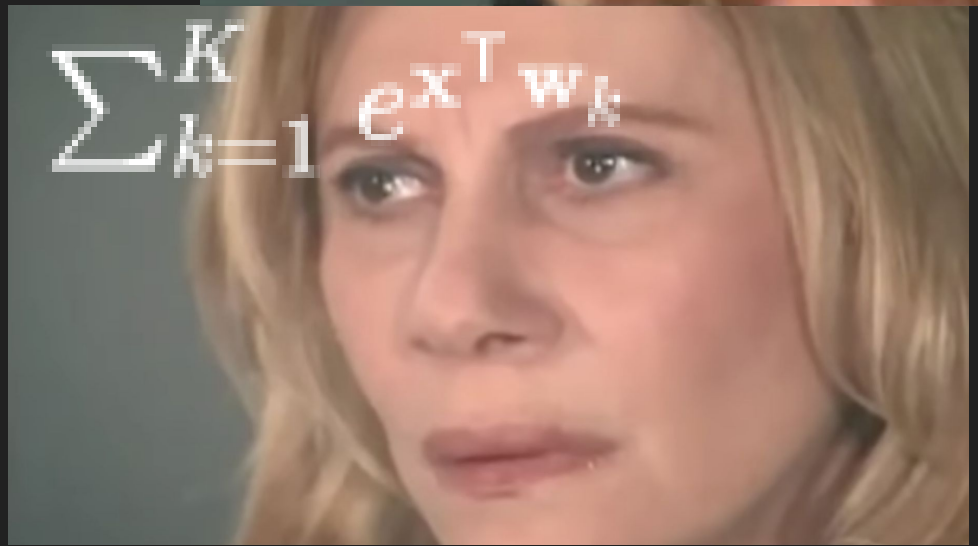
Softmax:

Normalized using sigmoidal function on *every word* in the vocabulary → computationally expensive

$$P(y = j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\top \mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\top \mathbf{w}_k}}$$

Solution:

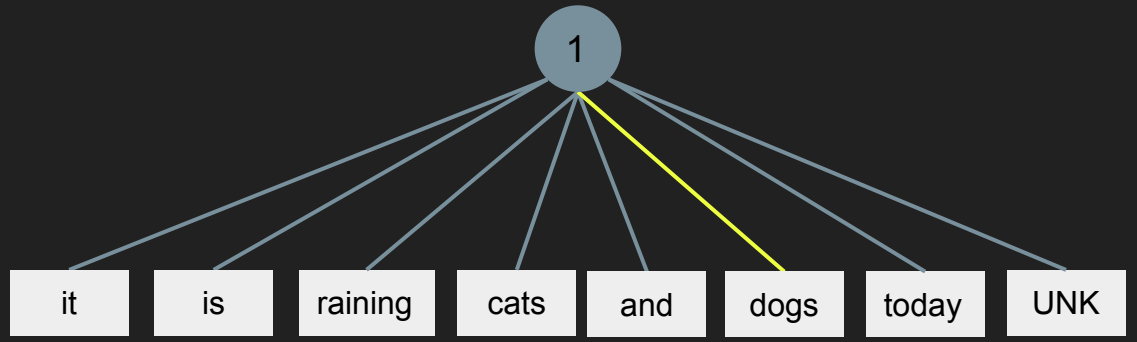
Hierarchical softmax



Activation Functions

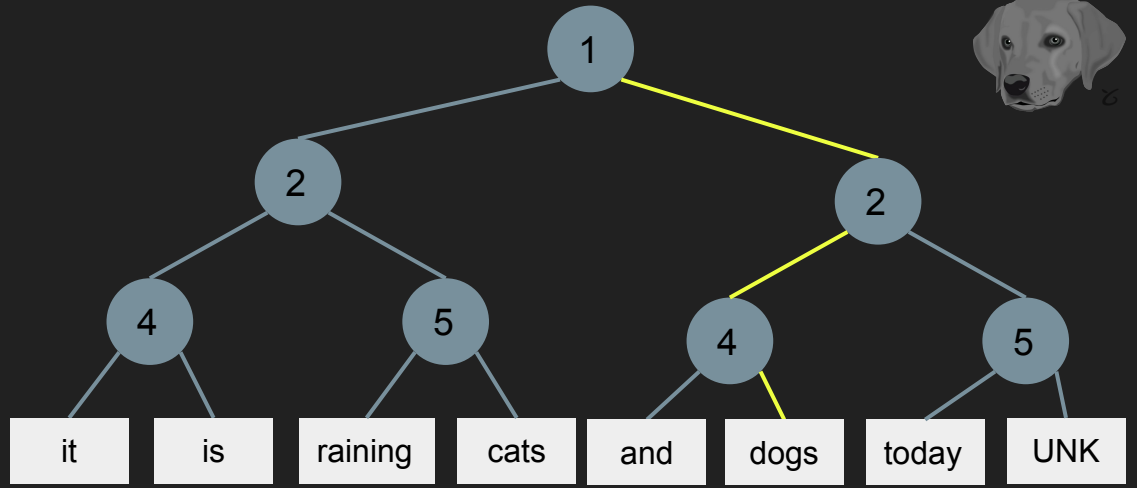
- (Flat) Softmax:

  |V| computations
  eg: p(dog|context)



- Hierarchical Softmax:

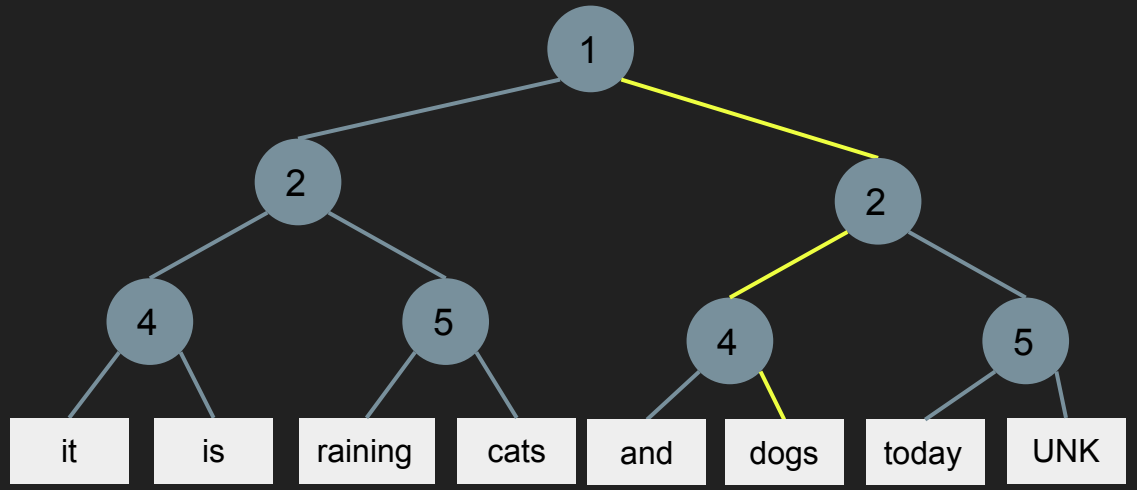
  $\log_2(|V|)$ computations
  eg: p(RLR|context)



Activation Functions

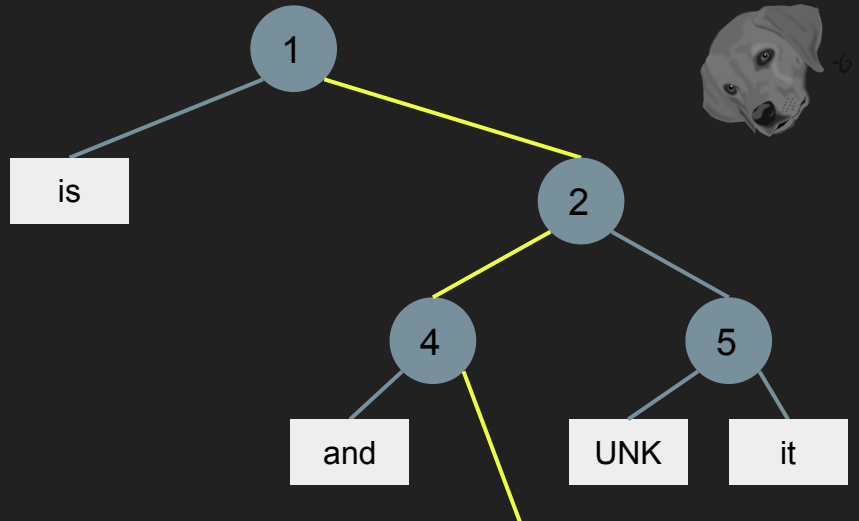
- Hierarchical Softmax:

  Balanced tree → every word at same depth



- Hierarchical Softmax, with Vocabulary as Huffman Tree:

  Most frequent words at shallowest leaves



NNLM: Dominating Term

$Q = (N \times D) + (N \times D \times H) + (H \times V)$

But there are remedies:

- Using hierarchical softmax
- Avoiding normalization during training
- Representing vocab as huffman tree

	Complexity	V = 1mil
Balanced binary tree	$\log_2(\|V\|)$	≈ 20
Huffman tree	$\log_2(PP_{1gram}(\|V\|))$	≈10 (assuming 1500 PP)

So the *real* bottleneck is: $Q = (N \times D) + (N \times D \times H) + (H \times V)$

RNNLM: Q Value

Uses a recurrent neural network (“remembers” previously seen inputs)

Differences from regular NNLM:

$$Q = (\cancel{N \times D}) + \cancel{(N \times D \times H)} + (H \times V)$$
$$H \times H$$

Don’t have to specify context length N.

Word representations have same dimensionality as HL

No projection layer

Which is the “computational bottleneck”?

Dominating Terms:

As before, it would be (H × V), but there are remedies,

so in effect it is (H × H).

Observation: Most of the complexity comes from the non-linear HL

Insight: A simpler model without a HL should be able to be trained more quickly

(This is no longer a proper NN, so may not represent data as precisely)

Mikolov et. al.’s Approach

Two-step approach:

- Simple model learns vectors
- NNLM is trained on these vectors (input is no longer discrete)

Two approaches to training the NNLM:

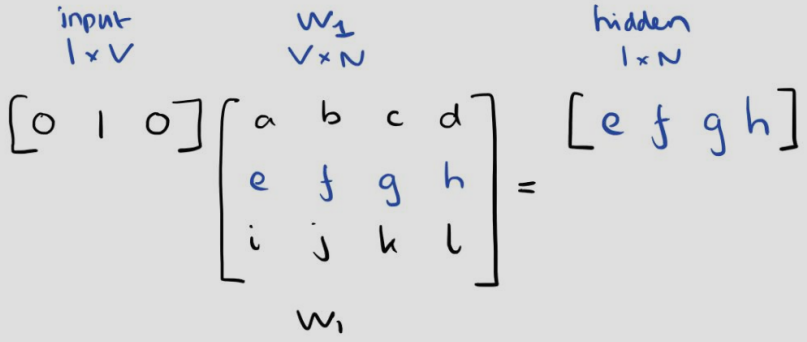
- CBOW
- Skip-gram

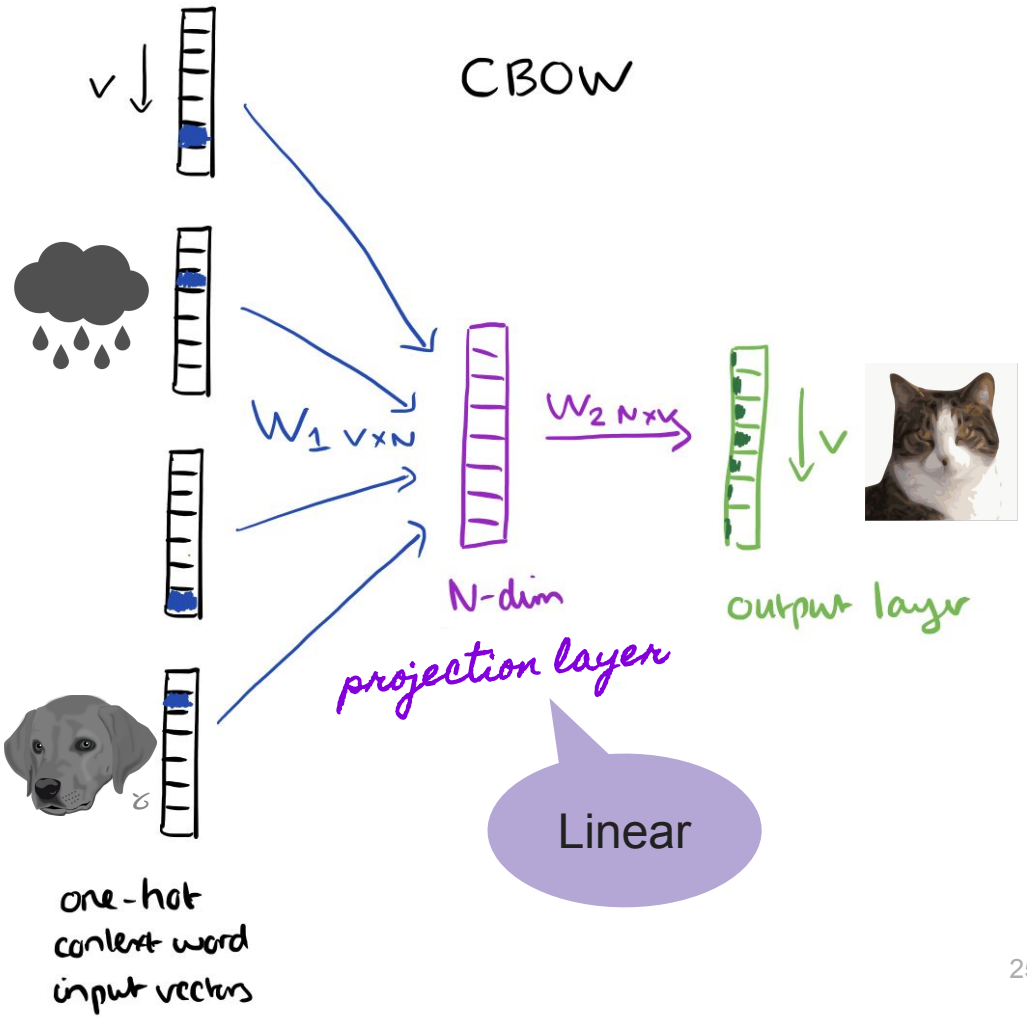
CBOW	Skip-gram
“Contiguous bag of words”	“Continuous skip-gram”
Training goal	
Given context, predict a word	Given word, predict context
Intuition (simplification)	
“it’s raining ___ and dogs”	*“___ ___ cats ___ ___”*
Advantages	
+Faster +Better for large datasets	+Better for small datasets +Better for infrequent terms

CBOW

eg: *“it’s raining ___ and dogs”*

Vectors are just summed and averaged in projection layer, so order of words from input doesn’t matter





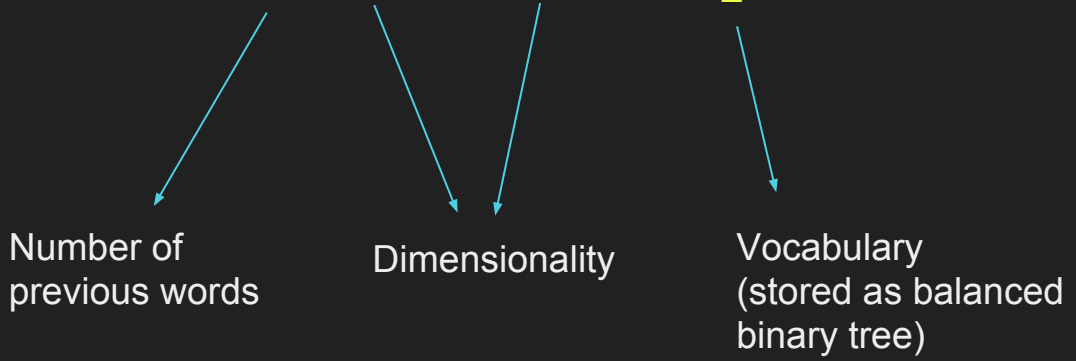
CBOW: Q Value

Reminder for NNLM:

$$Q = (N \times D) + (N \times D \times H) + (H \times \log_2(V))$$

Similar to NNLM, with non-linear HL removed

$$Q = (N \times D) + (D \times \log_2(V))$$



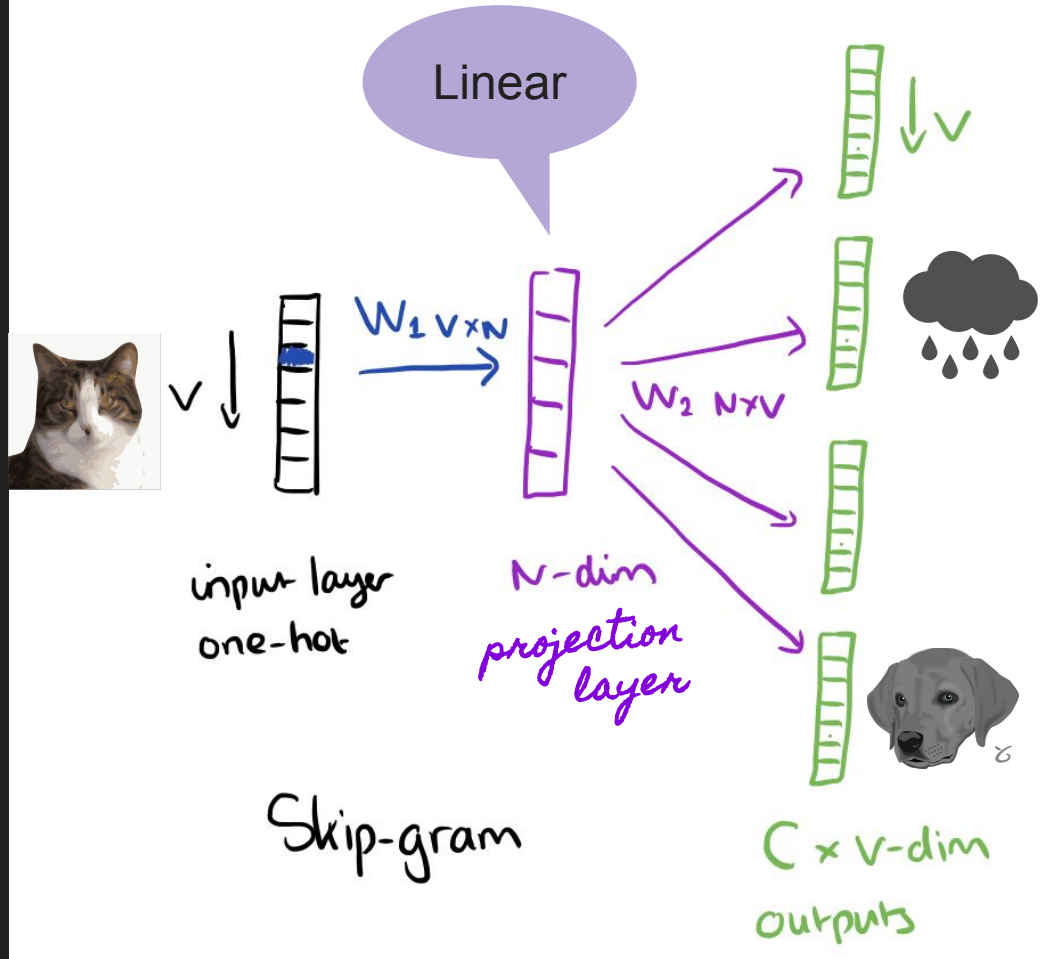
Skip-gram

eg: “___ ___ *cats* ___ ___”

Predict context words in a range R, which is between 1 and C.

In this example R=2, although perhaps C=4.

Sampling distant words less gives them lower weight and makes computations faster.



Skip-gram: Q Value

Reminder for CBOW:

$$Q = (N \times D) + (D \times \log_2(V))$$

Predict words within a certain range before and after

- Weighted by distance
- Set max distance, and sample random distance *up to* max

$$Q = C \times [D + (D \times \log_2(V)]$$

Max distance

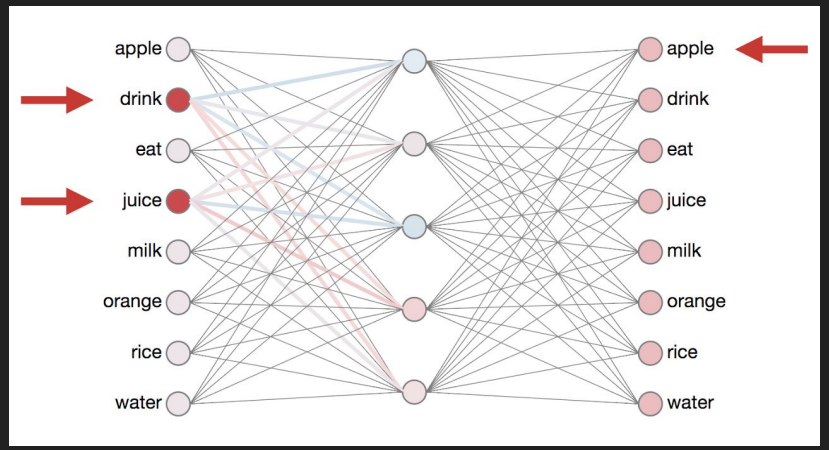
Dimensionality

Vocabulary (stored as balanced binary tree)

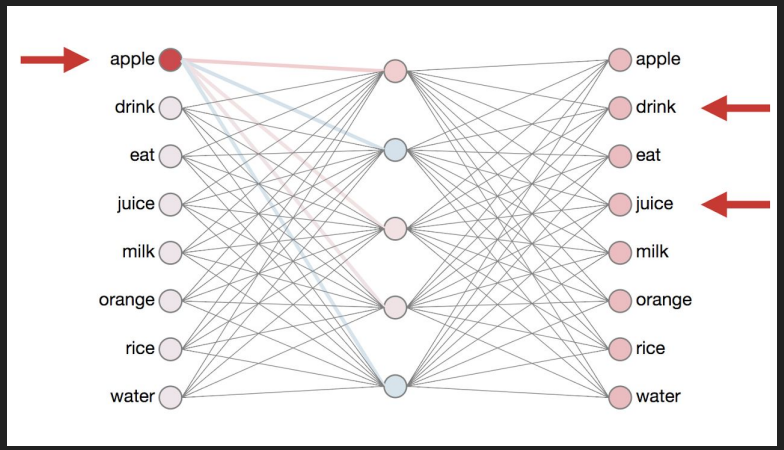
Wevi: word2vec Visualization Tool

https://ronxin.github.io/wevi/ (Thanks to Iliana for sharing)

CBOW



Skip-gram

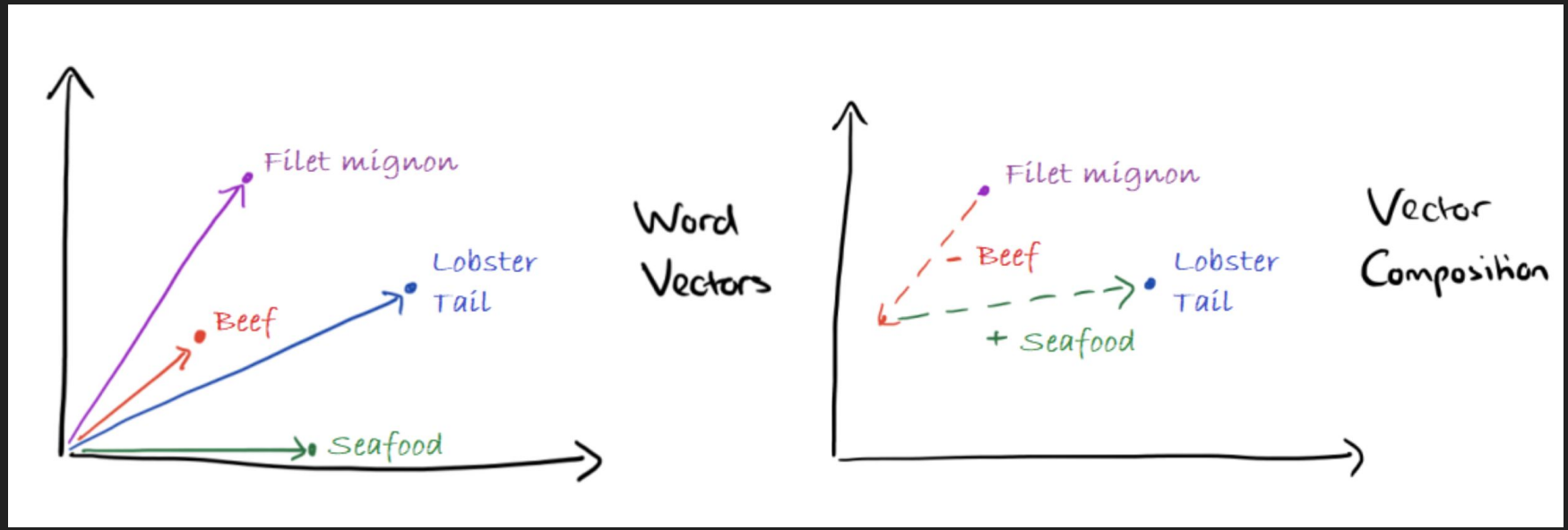


Similarity Arithmetic

Potential direct applications:

- Automatically extending facts in Knowledge Bases
- Verifying correctness of existing facts

Similarity Arithmetic



Similarity Arithmetic

Captures syntactic and semantic similarity

Also captures different multiple degrees of similarity

			Model output:
big	bigger	small	smaller
Copper	cu	Gold	au
Obama	Barak	Putin	Medvedev
Einstein	scientist	Messi	midfielder

Guessing Game:

Model output:

Japan	sushi	Germany	??
Japan	sushi		
Japan	sushi		

Guessing Game:

Model output:

Japan	sushi	Germany	bratwurst
Japan	sushi	USA	??
Japan	sushi		

Guessing Game:

Model output:

Japan	sushi	Germany	bratwurst
Japan	sushi	USA	pizza
Japan	sushi	France	??

Guessing Game:

Model output:

Japan	sushi	Germany	bratwurst
Japan	sushi	USA	pizza
Japan	sushi	France	tapas

(But averaging 10 examples of a relationship leads to 10% improvement)

Not perfect …

Evaluation

Just like our game: use vector artihmetic to try to complete the analogy

Only exact matches
are counted as accurate

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

Training Details

Used Google News Corpus to train vectors

- 1 billion tokens
- Vocab restricted to 1 million most frequent words (or fewer for some comparisons)

Dimensionality of up to 640

Trained via mini-batch asynchronous stochastic gradient descent and backpropigation, adaptive learning rate (0.025, decreasing linearly) with Adagrad
Trained in parallel through distributed computed using DistBelief

Results

"It is currently popular to train word vectors on relatively large amounts of data, but with insufficient size"

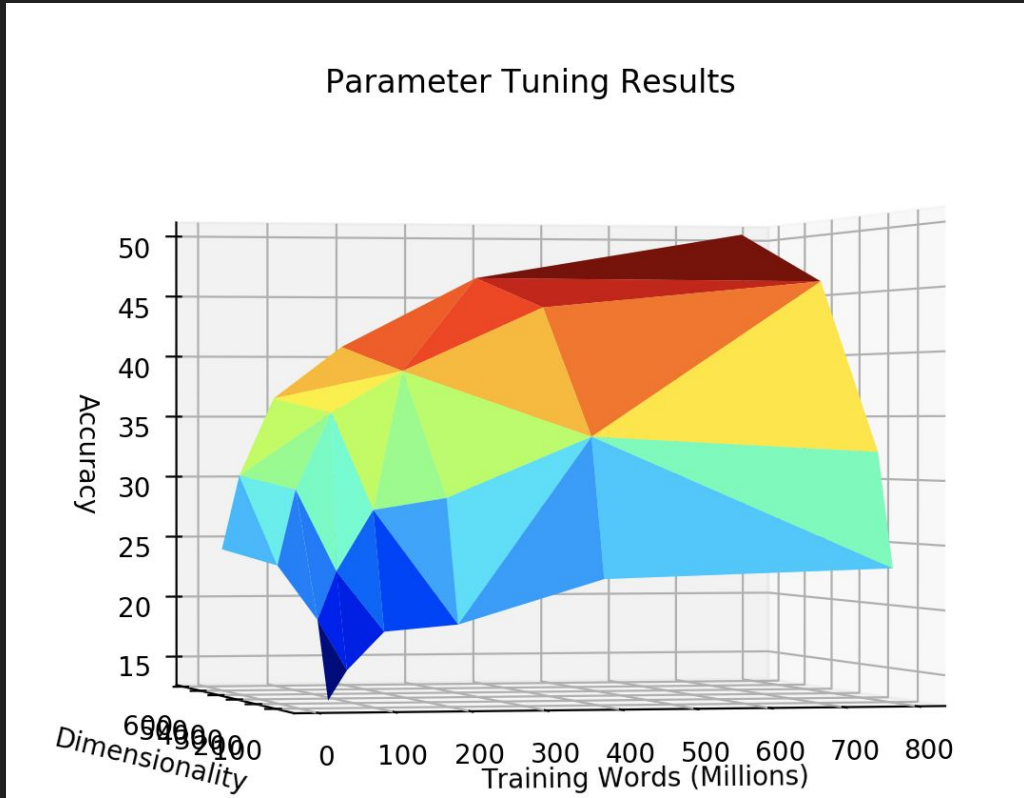
Results using CBOW, only top 30k vocab, and 3 training epochs:

Dimensionality / Training words	24M	49M	98M	196M	391M	783M
50	13.4	15.7	18.6	19.1	22.5	23.2
100	19.4	23.1	27.8	28.7	33.4	32.2
300	23.2	29.2	35.3	38.6	43.7	45.9
600	24.0	30.1	36.5	40.8	46.6	50.4

Results

“It is currently popular to train word vectors on relatively large amounts of data, but with insufficient size”

Just increasing training data size or just increasing dimensionality is not as good as increasing both.



Results: Accuracy across Models

Semantic relations are more difficult than syntactic ones

Word2vec outperforms previous models

CBOW is slightly better with syntactic accuracy, but
Skip-gram is much better at semantic accuracy

Model Architecture	Semantic-Syntactic Word Relationship test set		MSR Word Relatedness Test Set [20]
	Semantic Accuracy [%]	Syntactic Accuracy [%]	
RNNLM	9	36	35
NNLM	23	53	47
CBOW	24	64	61
Skip-gram	55	59	56

Results: Speed across Models

Word2vec generates vectors with 10 times the dimensionality than NNLM in one-tenth the time as it.

Model	Vector Dimensionality	Training words	Accuracy [%]			Training time [days x CPU cores]
			Semantic	Syntactic	Total	
NNLM	100	6B	34.2	64.5	50.8	14 x 180
CBOW	1000	6B	57.3	68.9	63.7	2 x 140
Skip-gram	1000	6B	66.1	65.1	65.6	2.5 x 125

Further Potential Improvements

- Include word structure information for building vectors
- Handle multi-word expressions, eg: vector for “New York”
- Training on limited data

Preview of their Follow-up Paper

- Finding phrases in a text
- Speedup by subsampling frequent words
- Negative sampling instead of hierarchical softmax

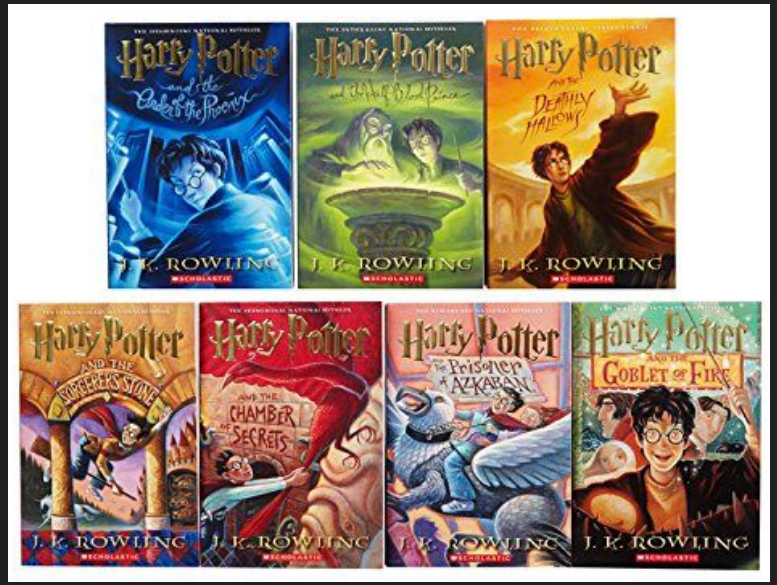
Playing Around with word2vec

gensim
topic modelling for humans

https://radimrehurek.com/gensim/models/word2vec.html

Training data: All 7 Harry Potter books

63.9k sentences
1.4 mil tokens
0.96 mil tokens excluding
hapax legomena and stopwords



Playing Around with word2vec

```
In [8]: import gensim, logging
        from gensim.models import Word2Vec

        model = gensim.models.Word2Vec(iter=10, size=1000, window=15, min_count=2, sg=1)
        model.build_vocab(all_sents)
        model.train(iter(all_sents), total_examples=model.corpus_count, epochs=20)
```

Tried to compensate for small dataset with beefy model (1000 dimensions)

Took less than one minute to train on a laptop CPU

Still, results not that great → Need more data

Tried with skip-gram and CBOW, but skip-gram performed considerably better

Playing Around with word2vec

Name : Magical Status

```
In [22]: model.most_similar(positive=['harry', 'wizard'], negative=['dursley'], topn=5)

Out[22]: [('man', 0.6315226554870605),
          ('incredibly', 0.6138944625854492),
          ('killer', 0.609815239906311),
          ('snorkack', 0.6069245338439941),
          ('voiced', 0.6059046983718872)]
```

Other answers: muggle, non-magical

Name : School House

```
In [12]: model.most_similar(positive=['harry', 'gryffindor'], negative=['malfoy'], topn=5)

Out[12]: [('quidditch', 0.7258402109146118),
          ('ravenclaw', 0.6982583999633789),
          ('slytherin', 0.6807438135147095),
          ('march', 0.6543607115745544),
          ('godric', 0.6498055458068848)]
```

Correct answer

Playing Around with word2vec

First Name : Last Name

```
In [14]: model.most_similar(positive=['harry', 'potter'], negative=['hermione'], topn=5)

Out[14]: [('thief', 0.718835711479187),
          ('tom', 0.7104202508926392),
          ('tm', 0.7091629505157471),
          ('deathly', 0.7062216401100159),
          ('seek', 0.7046765089035034)]
```

Correct answer:
granger

Teacher Name : Subject Taught

```
In [23]: model.most_similar(positive=['snape', 'potions'], negative=['mcgonagall'], topn=5)

Out[23]: [('brother', 0.8145629167556763),
          ('killing', 0.8054770827293396),
          ('james', 0.7979542016983032),
          ('cruciatus', 0.7953594923019409),
          ('interview', 0.7802930474281311)]
```

Correct answer:
transfiguration

Summary

- Why word vectors? →
  - Why word to vec? →
- Training complexity →
- Previous Approaches
  - NNLM
    - 1-of-V encoding →
    - Hierarchical softmax →
  - RNNLM
- New Approaches →
  - CBOW →
  - Skip-gram →
  - Similarity Arithmetic →
- Evaluation →
  - Results →
  - Further improvements
  - Playing around with word2vec →

Summary

- Why word vectors? → Continuous, can measure similarity
  - Why word to vec? →
- Training complexity →
- Previous Approaches
  - NNLM
    - 1-of-V encoding →
    - Hierarchical softmax →
  - RNNLM
- New Approaches →
  - CBOW →
  - Skip-gram →
  - Similarity Arithmetic →
- Evaluation →
  - Results →
  - Further improvements
  - Playing around with word2vec →

Summary

- Why word vectors? → Continuous, can measure similarity
  - Why word to vec? → Faster, better for larger datasets
- Training complexity →
- Previous Approaches
  - NNLM
    - 1-of-V encoding →
    - Hierarchical softmax →
  - RNNLM
- New Approaches →
  - CBOW →
  - Skip-gram →
  - Similarity Arithmetic →
- Evaluation →
  - Results →
  - Further improvements
  - Playing around with word2vec →

Summary

- Why word vectors? → Continuous, can measure similarity
  - Why word to vec? → Faster, better for larger datasets
- Training complexity → Depends on model architecture
- Previous Approaches
  - NNLM
    - 1-of-V encoding →
    - Hierarchical softmax →
  - RNNLM
- New Approaches →
  - CBOW →
  - Skip-gram →
  - Similarity Arithmetic →
- Evaluation →
  - Results →
  - Further improvements
  - Playing around with word2vec →

Summary

- Why word vectors? → Continuous, can measure similarity
  - Why word to vec? → Faster, better for larger datasets
- Training complexity → Depends on model architecture
- Previous Approaches
  - NNLM
    - 1-of-V encoding → Dog: [0,0,0,0,0,1,0]
    - Hierarchical softmax →
  - RNNLM
- New Approaches →
  - CBOW →
  - Skip-gram →
  - Similarity Arithmetic →
- Evaluation →
  - Results →
  - Further improvements
  - Playing around with word2vec →

Summary

- Why word vectors? → Continuous, can measure similarity
  - Why word to vec? → Faster, better for larger datasets
- Training complexity → Depends on model architecture
- Previous Approaches
  - NNLM
    - 1-of-V encoding → Dog: [0,0,0,0,0,1,0]
    - Hierarchical softmax → Probability of path on a binary tree. Much faster.
  - RNNLM
- New Approaches →
  - CBOW →
  - Skip-gram →
  - Similarity Arithmetic →
- Evaluation →
  - Results →
  - Further improvements
  - Playing around with word2vec →

Summary

- Why word vectors? → Continuous, can measure similarity
  - Why word to vec? → Faster, better for larger datasets
- Training complexity → Depends on model architecture
- Previous Approaches
  - NNLM
    - 1-of-V encoding → Dog: [0,0,0,0,0,1,0]
    - Hierarchical softmax → Probability of path on a binary tree. Much faster.
  - RNNLM
- New Approaches → Simpler model: linear activation in HL
  - CBOW →
  - Skip-gram →
  - Similarity Arithmetic →
- Evaluation →
  - Results →
  - Further improvements
  - Playing around with word2vec →

Summary

- Why word vectors? → Continuous, can measure similarity
  - Why word to vec? → Faster, better for larger datasets
- Training complexity → Depends on model architecture
- Previous Approaches
  - NNLM
    - 1-of-V encoding → Dog: [0,0,0,0,0,1,0]
    - Hierarchical softmax → Probability of path on a binary tree. Much faster.
  - RNNLM
- New Approaches → Simpler model: linear activation in HL
  - CBOW → Given context, guess word
  - Skip-gram →
  - Similarity Arithmetic →
- Evaluation →
  - Results →
  - Further improvements
  - Playing around with word2vec →

Summary

- Why word vectors? → Continuous, can measure similarity
  - Why word to vec? → Faster, better for larger datasets
- Training complexity → Depends on model architecture
- Previous Approaches
  - NNLM
    - 1-of-V encoding → Dog: [0,0,0,0,0,1,0]
    - Hierarchical softmax → Probability of path on a binary tree. Much faster.
  - RNNLM
- New Approaches → Simpler model: linear activation in HL
  - CBOW → Given context, guess word
  - Skip-gram → Given word, guess context
  - Similarity Arithmetic →
- Evaluation →
  - Results →
  - Further improvements
  - Playing around with word2vec →

Summary

- Why word vectors? → Continuous, can measure similarity
  - Why word to vec? → Faster, better for larger datasets
- Training complexity → Depends on model architecture
- Previous Approaches
  - NNLM
    - 1-of-V encoding → Dog: [0,0,0,0,0,1,0]
    - Hierarchical softmax → Probability of path on a binary tree. Much faster.
  - RNNLM
- New Approaches → Simpler model: linear activation in HL
  - CBOW → Given context, guess word
  - Skip-gram → Given word, guess context
  - Similarity Arithmetic → Man - King + Women = Queen
- Evaluation →
  - Results →
  - Further improvements
  - Playing around with word2vec →

Summary

- Why word vectors? → Continuous, can measure similarity
  - Why word to vec? → Faster, better for larger datasets
- Training complexity → Depends on model architecture
- Previous Approaches
  - NNLM
    - 1-of-V encoding → Dog: [0,0,0,0,0,1,0]
    - Hierarchical softmax → Probability of path on a binary tree. Much faster.
  - RNNLM
- New Approaches → Simpler model: linear activation in HL
  - CBOW → Given context, guess word
  - Skip-gram → Given word, guess context
  - Similarity Arithmetic → Man - King + Women = Queen
- Evaluation → Analogy completion, speed
  - Results →
  - Further improvements
  - Playing around with word2vec →

Summary

- Why word vectors? → Continuous, can measure similarity
  - Why word to vec? → Faster, better for larger datasets
- Training complexity → Depends on model architecture
- Previous Approaches
  - NNLM
    - 1-of-V encoding → Dog: [0,0,0,0,0,1,0]
    - Hierarchical softmax → Probability of path on a binary tree. Much faster.
  - RNNLM
- New Approaches → Simpler model: linear activation in HL
  - CBOW → Given context, guess word
  - Skip-gram → Given word, guess context
  - Similarity Arithmetic → Man - King + Women = Queen
- Evaluation → Analogy completion, speed
  - Results → Faster, better
  - Further improvements
  - Playing around with word2vec →

Summary

- Why word vectors? → Continuous, can measure similarity
  - Why word to vec? → Faster, better for larger datasets
- Training complexity → Depends on model architecture
- Previous Approaches
  - NNLM
    - 1-of-V encoding → Dog: [0,0,0,0,0,1,0]
    - Hierarchical softmax → Probability of path on a binary tree. Much faster.
  - RNNLM
- New Approaches → Simpler model: linear activation in projection layer
  - CBOW → Given context, guess word
  - Skip-gram → Given word, guess context
  - Similarity Arithmetic → Man - King + Women = Queen
- Evaluation → Analogy completion, speed
  - Results → Faster, better
  - Further improvements
  - Playing around with word2vec → Easy to do with gensim

End of Presentation – Any Questions?

Thank you