COMP4901K/Math4824B
Machine Learning for Natural Language Processing

Lecture 14: Word Embeddings

Instructor: Yangqiu Song

Slides credits: Richard Socher, Vagelis Hristidis

Semantic Similarity/Relatedness

- Similarity is a specific type of relatedness: graded
  - car vs. automobile -> 1.0
  - car vs. vehicle -> 0.6
  - car vs. tire -> 0.2
  - car vs. street -> 0.1
- Similarity: **synonyms**, **hyponyms/hyperonyms**, and **siblings** are highly similar
  - doctor vs. surgeon, bike vs. bicycle
- Relatedness: **topically related** or based on any other **semantic relation**
  - heart vs. surgeon, tire vs. car

Computational Approaches

- Knowledge base based
  - WordNet Similarity
  - …
- Corpus based
  - Distributional similarity
  - Deep learning

Corpus based Approach

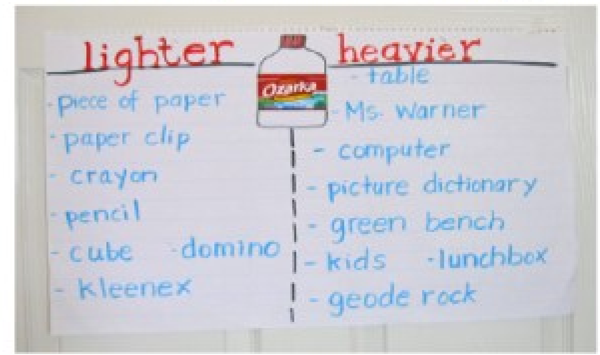
- Roots in linguistics:
  - Distributional hypothesis: Semantically similar words occur in similar contexts (Harris (1954))
  - You shall know a word by the company it keeps." (Firth (1957))
- Distributional semantics
  - The **distributional hypothesis** in linguistics is derived from the semantic theory of language usage, i.e. words that are used and occur in the same contexts tend to purport similar meanings.
  - The basic idea of distributional semantics can be summed up in the so-called Distributional hypothesis: *linguistic items with similar distributions have similar meanings*.

We will mention **distributed representation** based neural language models in this class

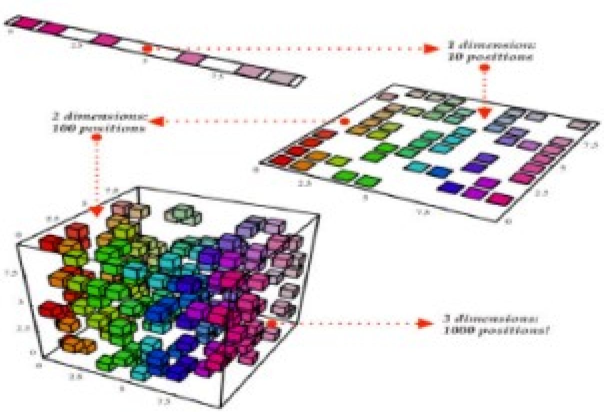
Corpus based Approach

1) Corpus

2) Preprocessing



3) Dimensionality Reduction



4) Post Processing

Let's try to keep the kitchen ______ .

- Observation: context can tell us a lot about word meaning

- **Context**: local window around a word occurrence (for now)

- Roots in linguistics:
  - **Distributional hypothesis**: Semantically similar words occur in similar contexts [Harris, 1954]
  - "You shall know a word by the company it keeps." [Firth, 1957]

- Pros: data-driven, easy to implement
- Cons: ambiguity

Window based co-occurrence matrix

- Window length 1 (more common: 5 - 10)
- Symmetric (irrelevant whether left or right context)
- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

Window based co-occurrence matrix

- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

Problems with simple co-occurrence vectors

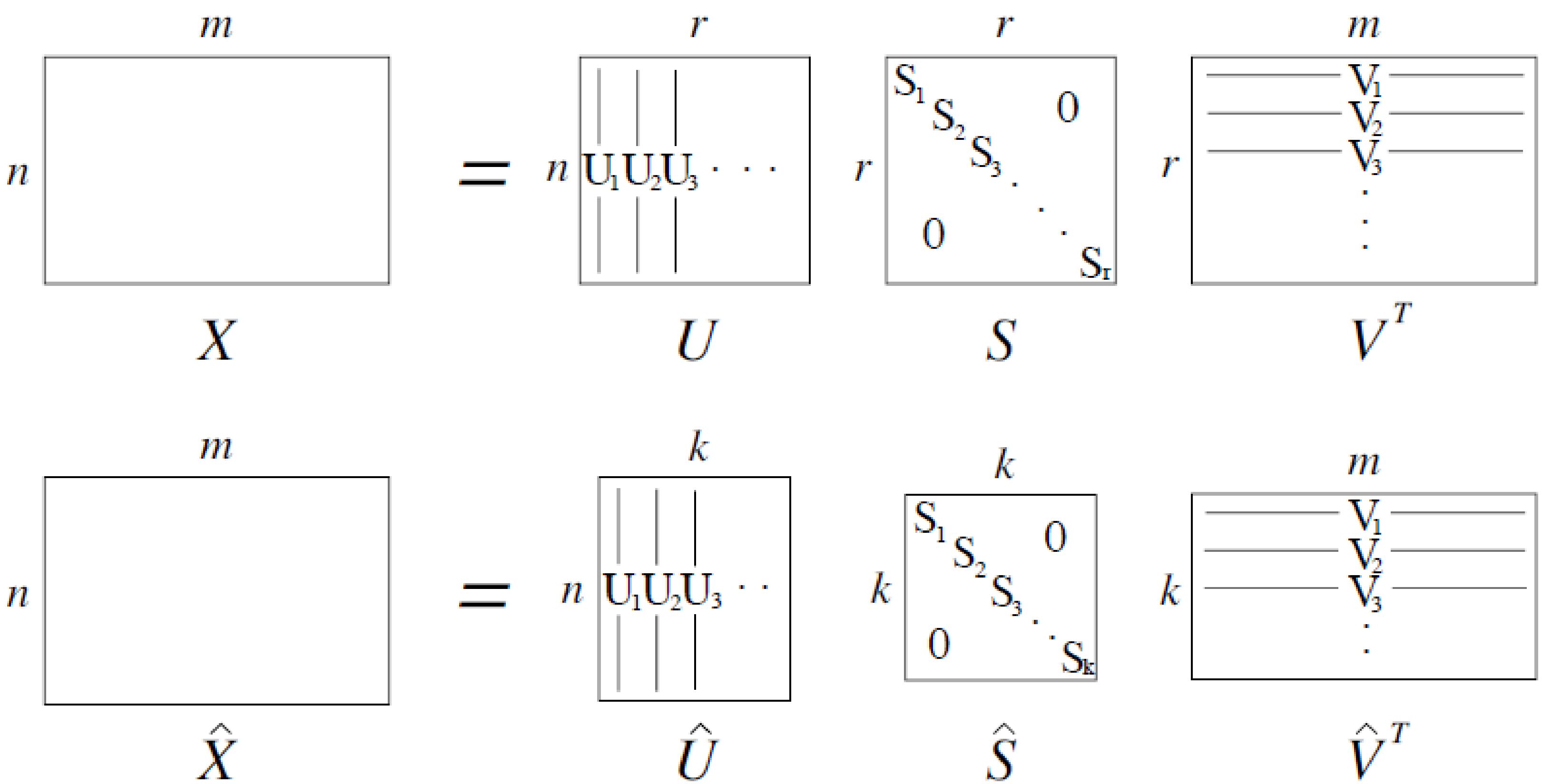
- Increase in size with vocabulary
- Very high dimensional: require a lot of storage
- Subsequent classification models have sparsity issues
- →Models are less robust

Solution: Low dimensional vectors

- Idea: store "most" of the important information in a fixed, small number of dimensions: a dense vector
- Usually around 25 – 1000 dimensions
- How to reduce the dimensionality?

Method 1: Dimensionality Reduction on X

- Singular Value Decomposition of co-occurrence matrix *X*.



best rank k approximation to X , in terms of least squares.

Simple SVD word vectors in Python

- Corpus:
- I like deep learning. I like NLP. I enjoy flying.

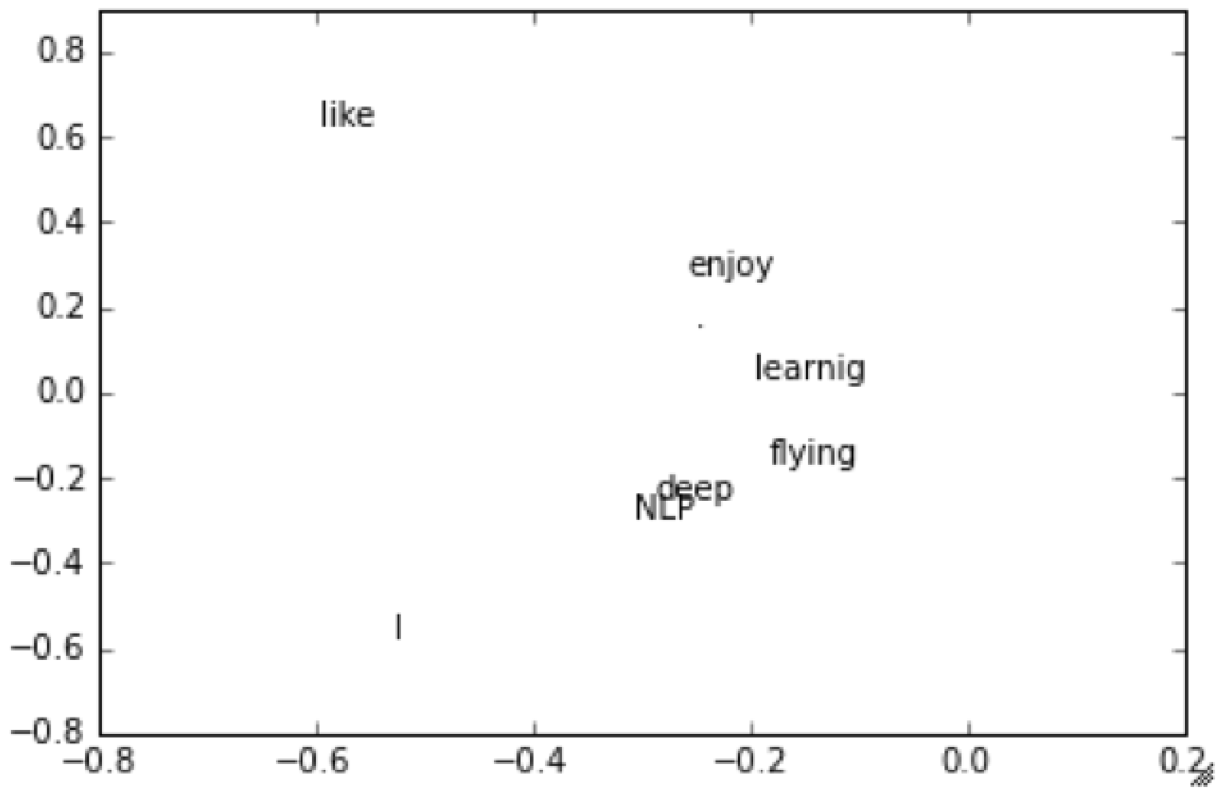
```
import numpy as np
la = np.linalg
words = ["I", "like", "enjoy",
         "deep","learnig","NLP","flying","."]
X = np.array([[0,2,1,0,0,0,0,0],
              [2,0,0,1,0,1,0,0],
              [1,0,0,0,0,0,1,0],
              [0,1,0,0,1,0,0,0],
              [0,0,0,1,0,0,0,1],
              [0,1,0,0,0,0,0,1],
              [0,0,1,0,0,0,0,1],
              [0,0,0,0,1,1,1,0]])

U, s, Vh = la.svd(X, full_matrices=False)
```

Simple SVD word vectors in Python

- Corpus: I like deep learning. I like NLP. I enjoy flying.
- Printing first two columns of U corresponding to the 2 biggest singular values

```
for i in xrange(len(words)):
    plt.text(U[i,0], U[i,1], words[i])
```



Word meaning is defined in terms of vectors

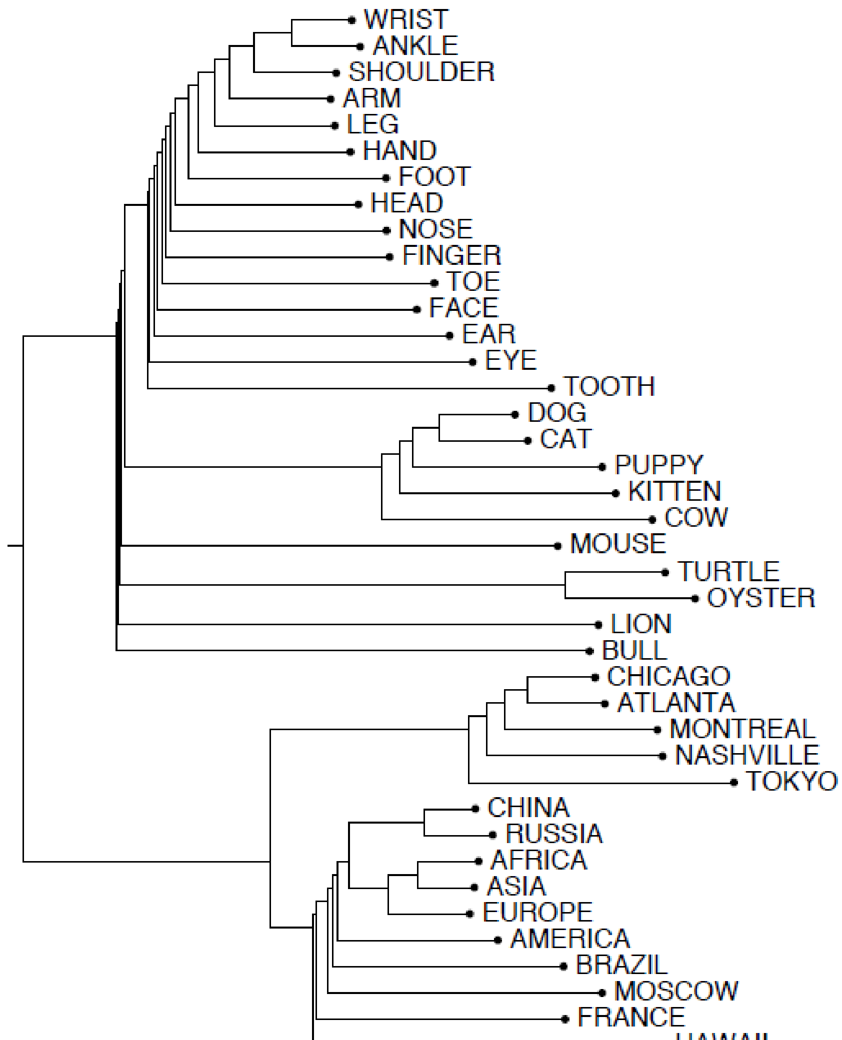
- In most deep learning models, a word is represented as a dense vector

$$\textit{linguistics} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

Hacks to X

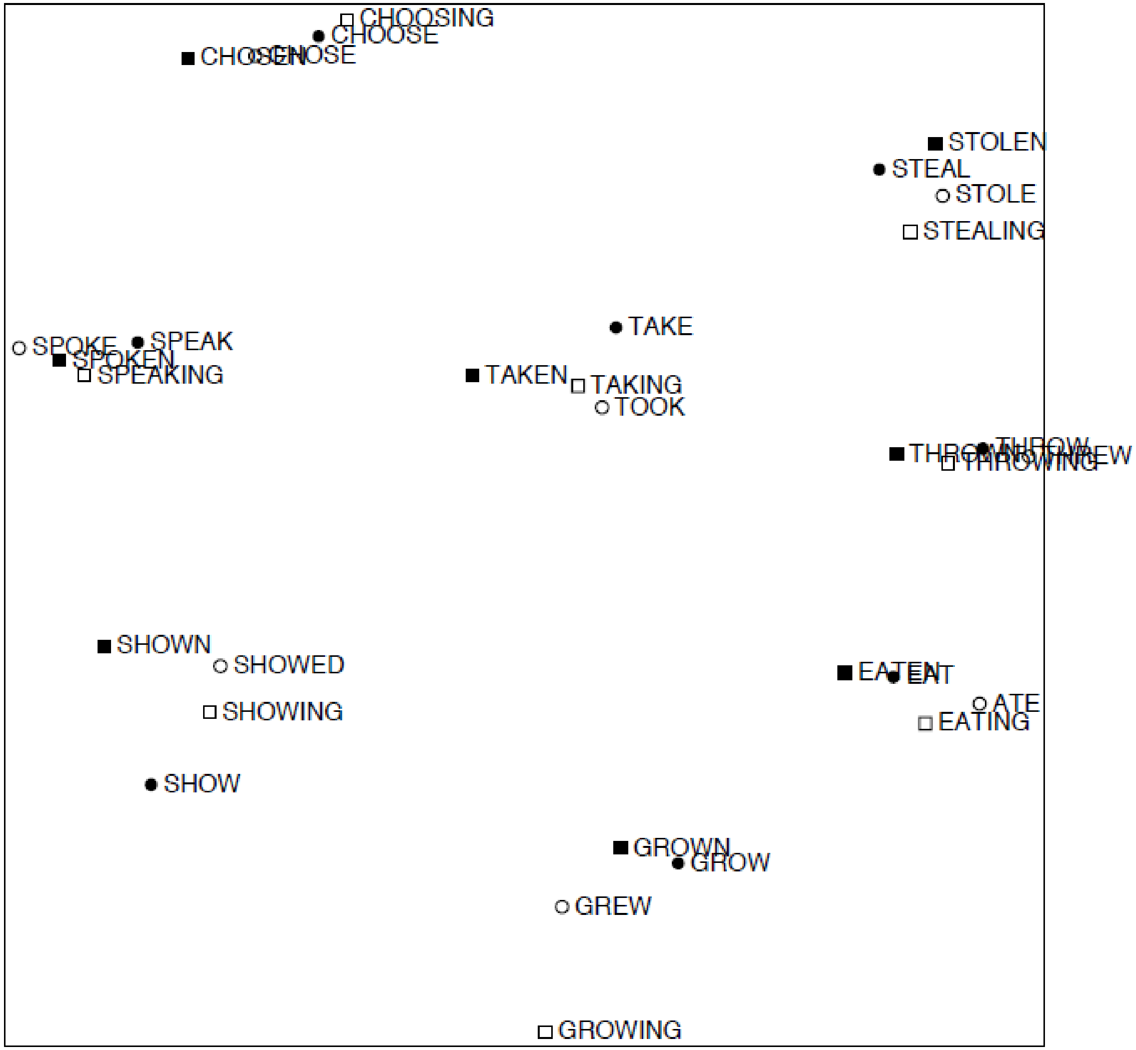
- Problem: function words (the, he, has) are too frequent → syntax has too much impact. Some fixes:
  - min(X,t), with t~100
  - Ignore them all
- Ramped windows that count closer words more
- Use Pearson correlations instead of counts, then set negative values to 0

Interesting semantic patters emerge in the vectors



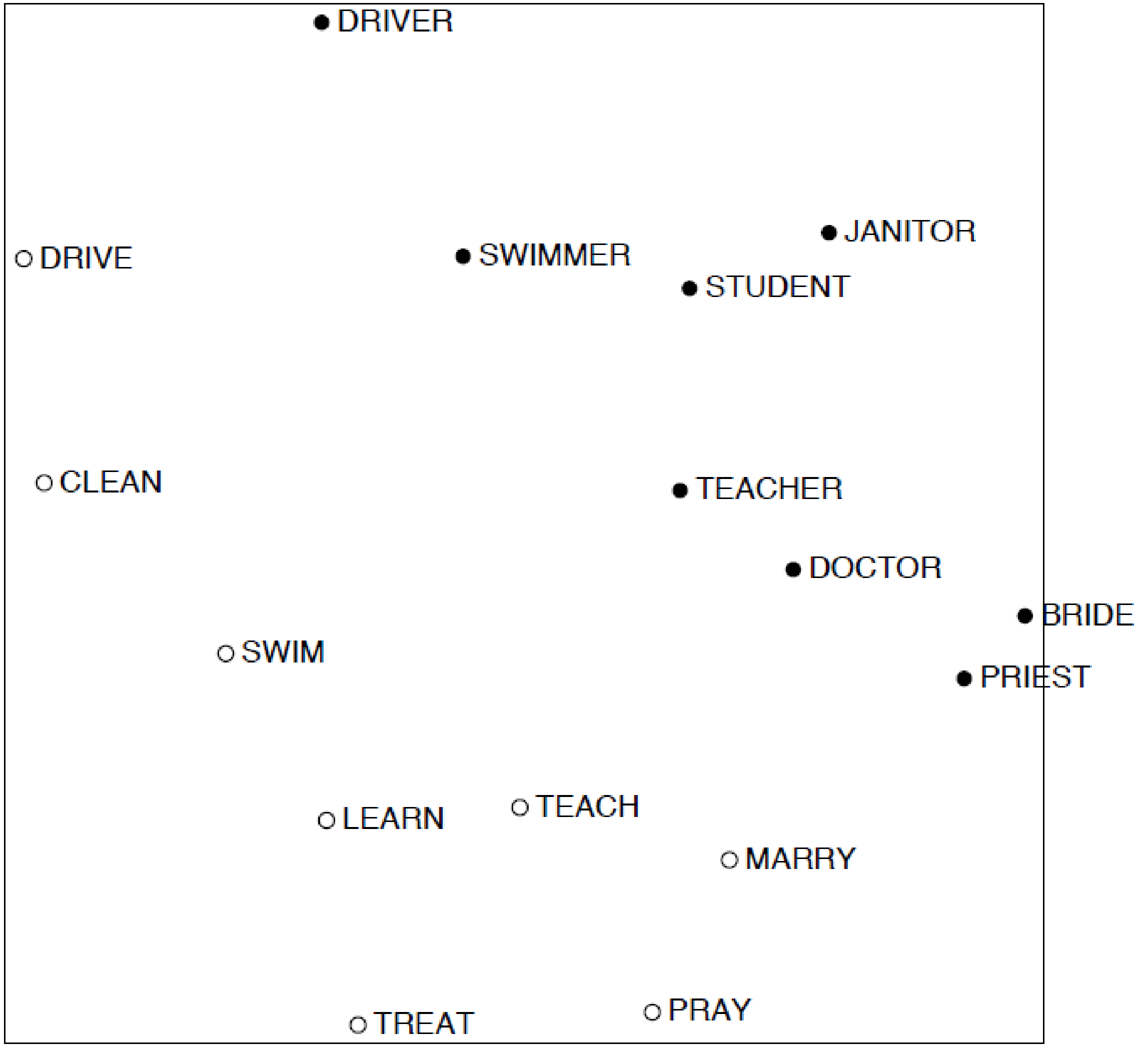
- An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence (Rohde et al. 2005)

Interesting semantic patters emerge in the vectors



- An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence (Rohde et al. 2005)

Interesting semantic patters emerge in the vectors



- An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence (Rohde et al. 2005)

Problems with SVD

- Computational cost scales quadratically for n x m matrix:
  - $O(mn^2)$ flops (when n<m)
  - →Bad for millions of words or documents

- Hard to incorporate new words or documents

- Different learning regime than other DL models

Idea: Directly learn low-dimensional word vectors

- Old idea. Relevant for this lecture & deep learning:
  - Learning representations by back-propagating errors. (Rumelhart et al., 1986)
  - A neural probabilistic language model (Bengio et al., 2003)
    - Multilayer perceptron
  - NLP (almost) from Scratch (Collobert & Weston, 2008)
    - CNN
  - An even simpler and faster model:
    - word2vec (Mikolov et al. 2013) → intro now

Distributed Representations

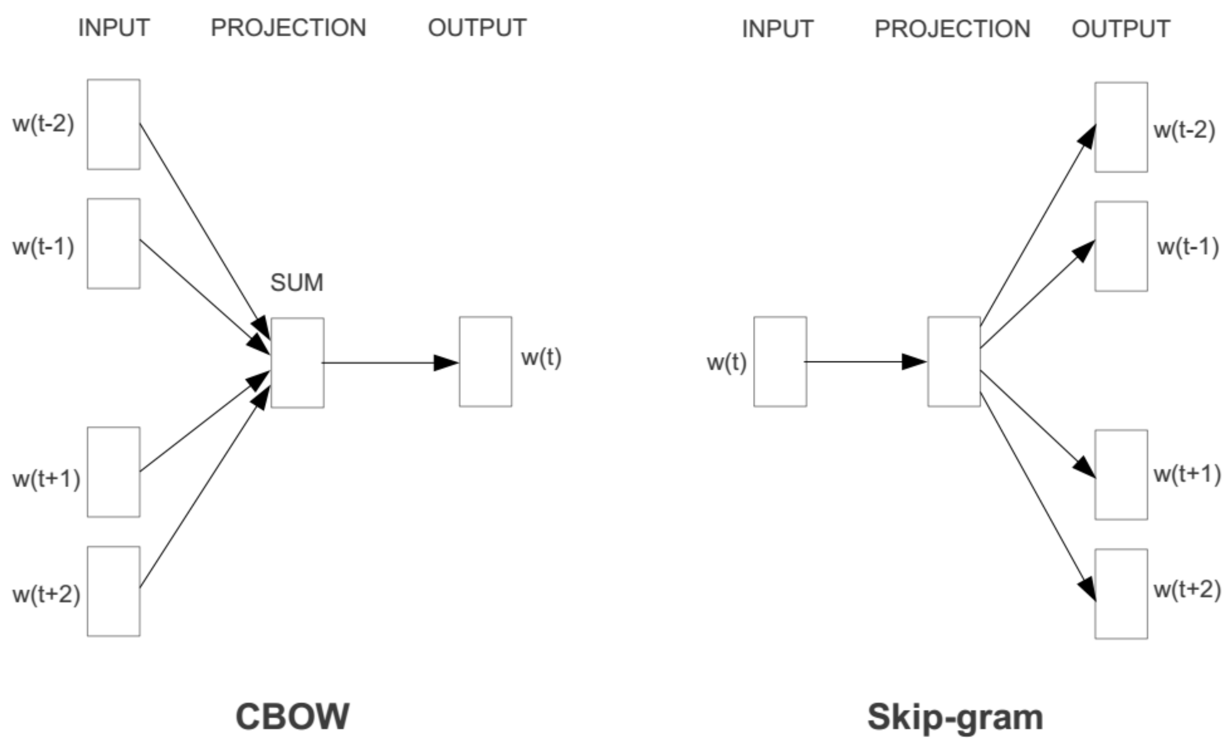
- This is usually called distributed representations in the context of deep learning
  - Vector representation does not represent a distribution, but distributed over the space
  - Term widely used in connectionism (Learning distributed representations of concepts, Hinton (1986))
    - "In the componential approach each concept is simply a set of features and so a neural net can be made to implement a set of concepts by assigning a unit to each feature and setting the strengths of the connections between units so that each concept corresponds to a stable pattern of activity distributed over the whole network."
- Compared to distributional semantics
  - The **distributional hypothesis** in linguistics is derived from the semantic theory of language usage, i.e. words that are used and occur in the same contexts tend to purport similar meanings.

Main Idea of word2vec

- Instead of capturing co-occurrence counts directly,
- Predict surrounding words of every word
- Both are quite similar, see *"Glove: Global Vectors for Word Representation"* by Pennington et al. (2014) and Levy and Goldberg (2014) … more later
- Faster and can easily incorporate a new sentence/document or add a word to the vocabulary

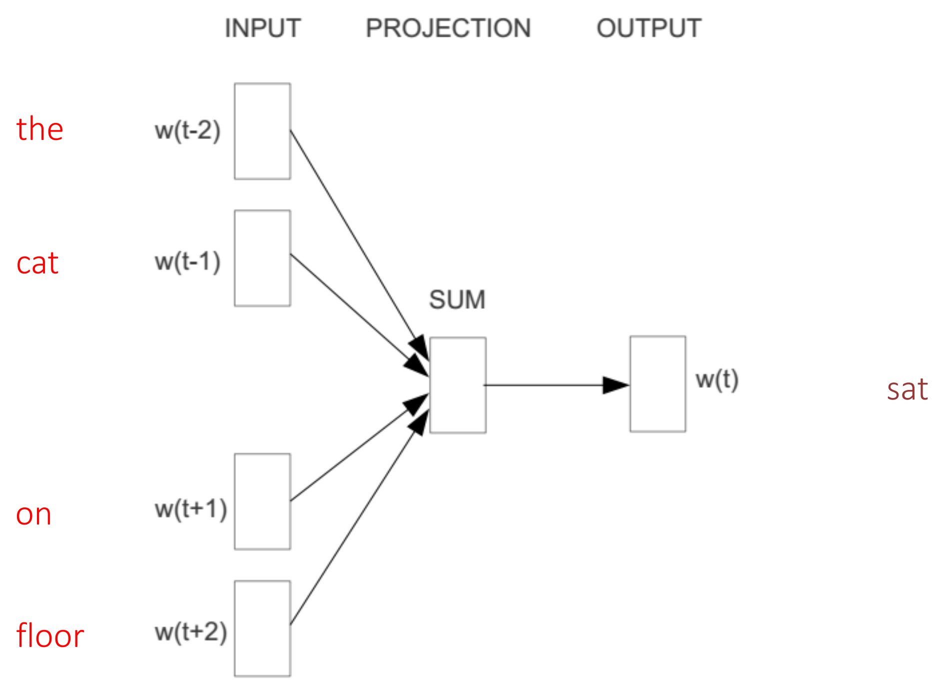
Represent the meaning of word – word2vec

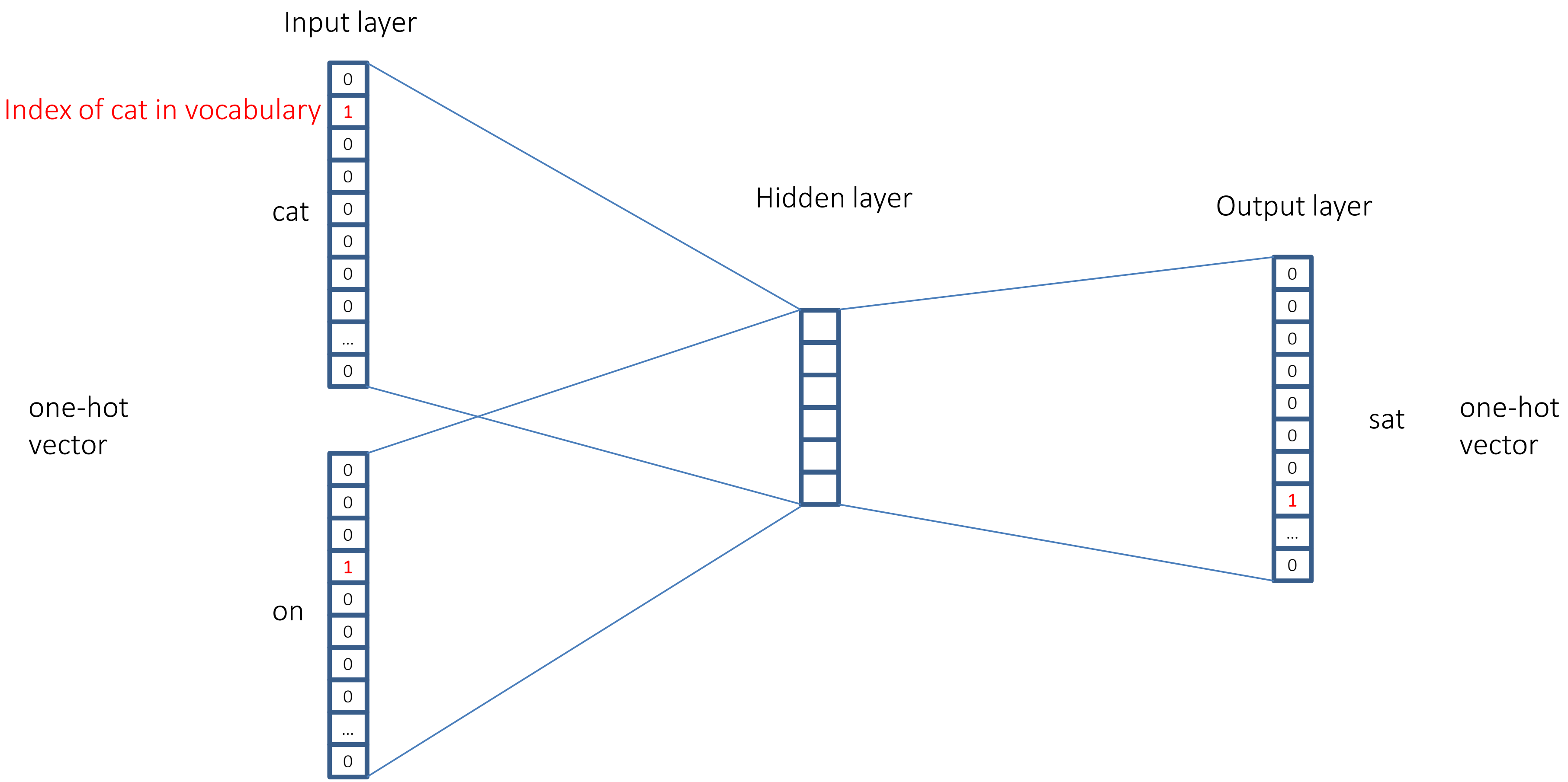
- 2 basic neural network models:
  - Continuous Bag of Word (CBOW): use a window of word to predict the middle word
  - Skip-gram (SG): use a word to predict the surrounding ones in window.

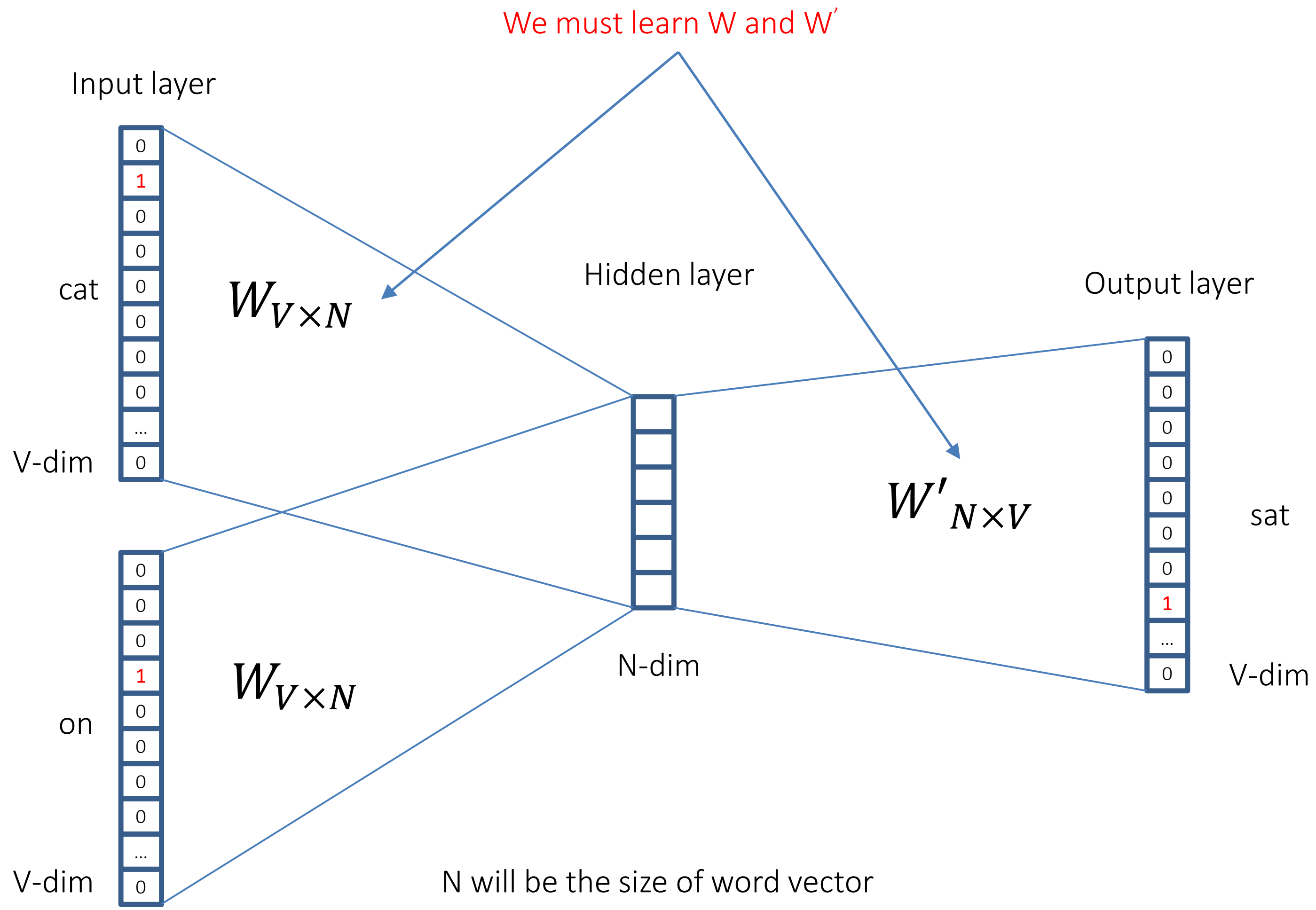


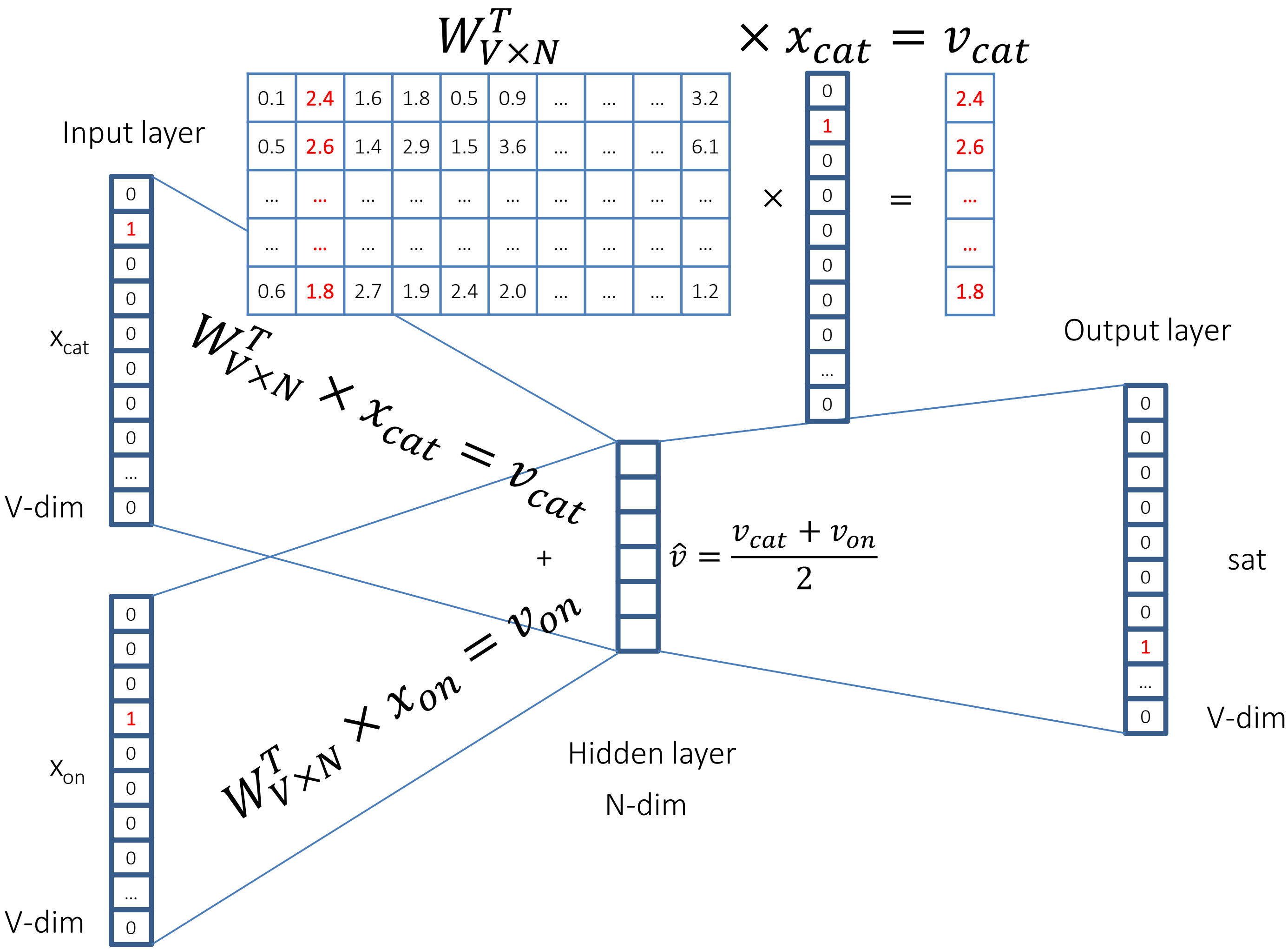
Word2vec – Continuous Bag of Word

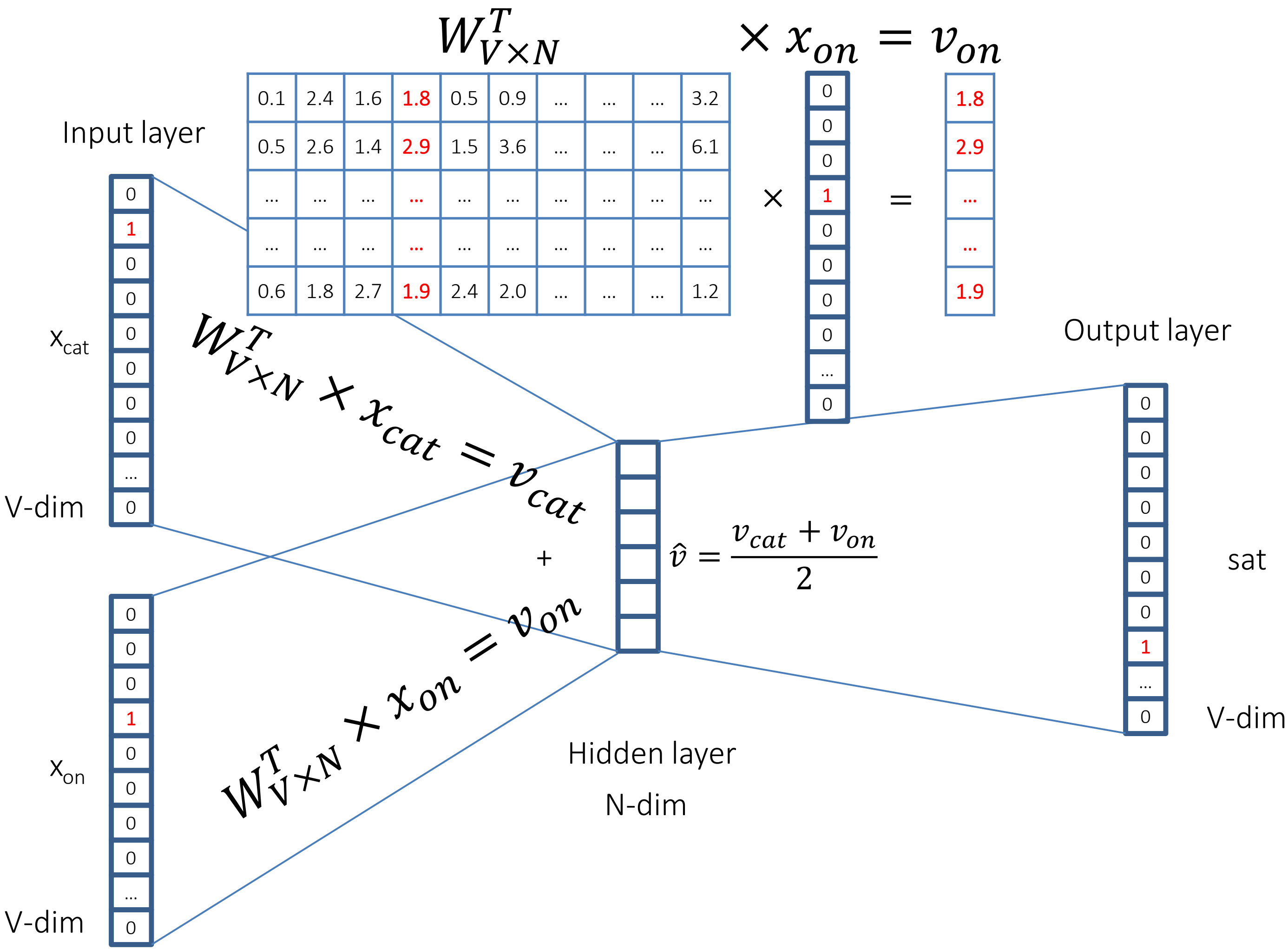
- E.g. "The cat sat on floor"
  - Window size = 2

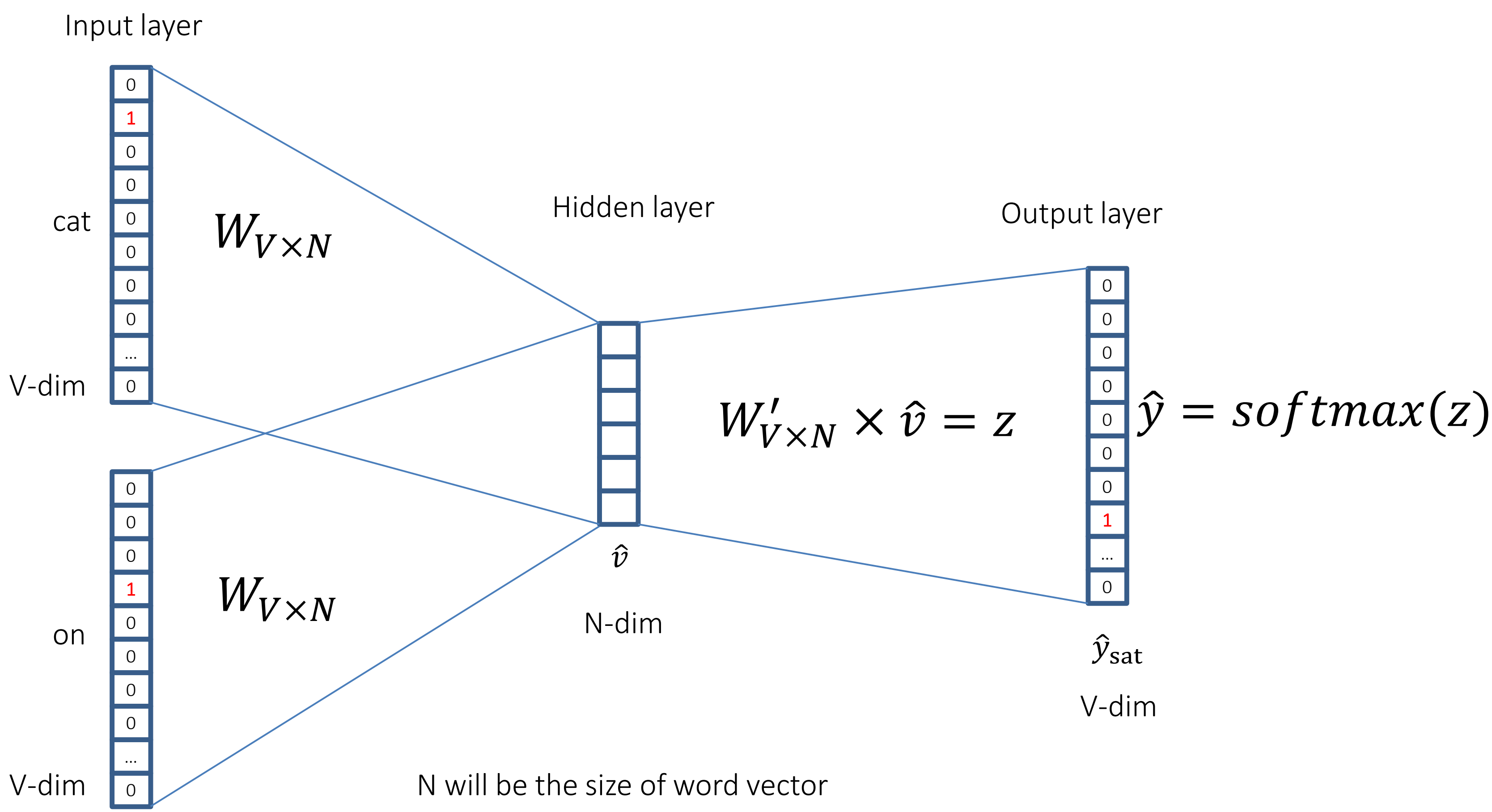



Input layer
Index of cat in vocabulary
0
1
0
0
0
0
0
0
...
0
cat
one-hot
vector
0
0
0
1
0
0
0
0
...
0
on
Hidden layer
Output layer
0
0
0
0
0
0
0
1
...
0
sat
one-hot
vector

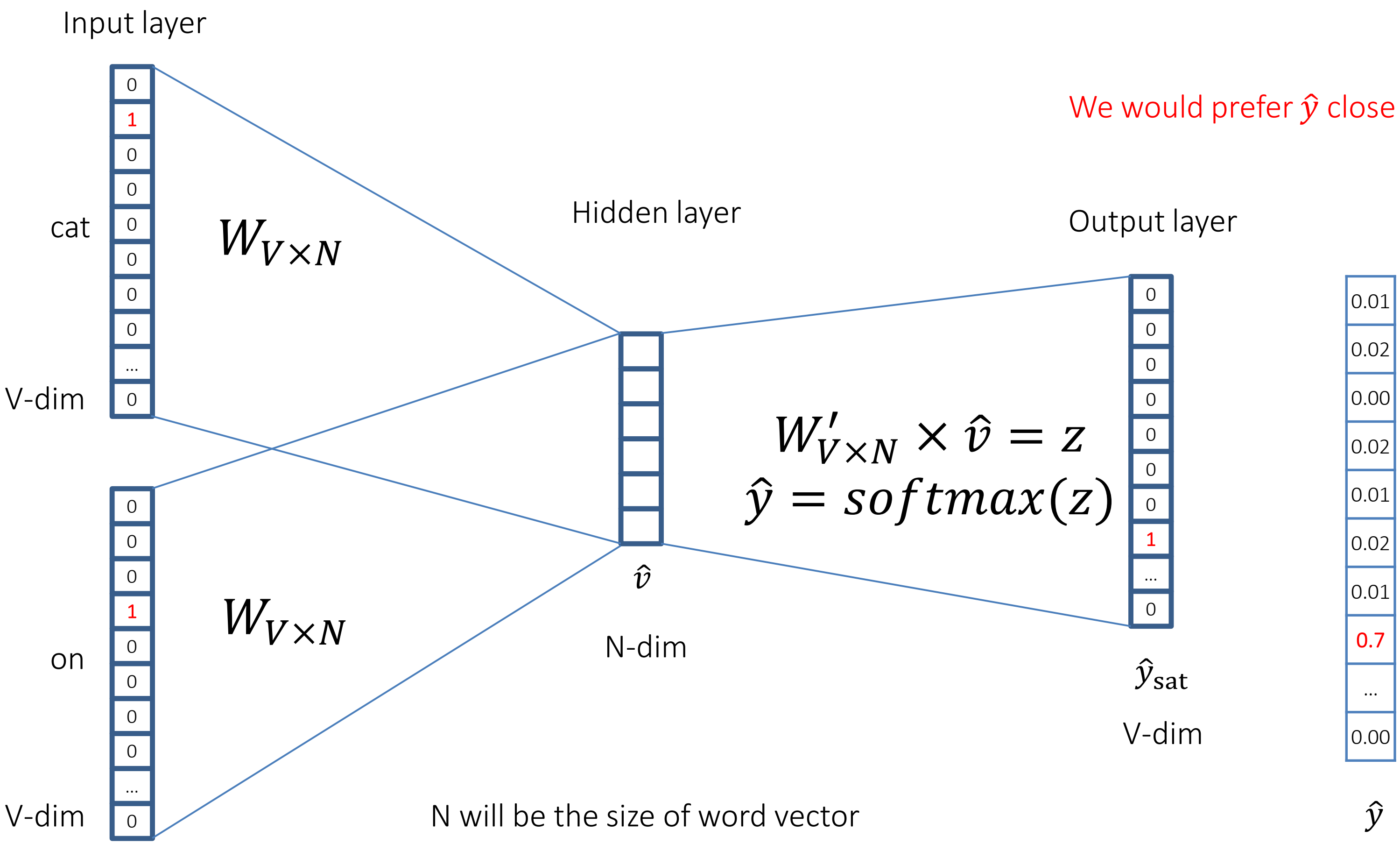

We must learn W and W′
Input layer
cat
V-dim
on
V-dim
$W_{V\times N}$
$W_{V\times N}$
Hidden layer
N-dim
Output layer
$W'_{N\times V}$
sat
V-dim
0
1
0
0
0
0
0
0
…
0
0
0
0
1
0
0
0
0
…
0
0
0
0
0
0
0
0
1
…
0
N will be the size of word vector


$W_{V\times N}^T \times x_{cat} = v_{cat}$
Input layer
Output layer
Hidden layer
N-dim
V-dim
V-dim
V-dim
x_{cat}
x_{on}
sat
$W_{V\times N}^T \times x_{cat} = v_{cat}$
+
$W_{V\times N}^T \times x_{on} = v_{on}$
$\hat{v} = \frac{v_{cat} + v_{on}}{2}$
0.1 2.4 1.6 1.8 0.5 0.9 … … … 3.2
0.5 2.6 1.4 2.9 1.5 3.6 … … … 6.1
… … … … … … … … … …
… … … … … … … … … …
0.6 1.8 2.7 1.9 2.4 2.0 … … … 1.2
×
=
2.4
2.6
1.8


$W^T_{V\times N} \times x_{on} = v_{on}$
Input layer
x_{cat}
V-dim
x_{on}
V-dim
$W^T_{V\times N} \times x_{cat} = v_{cat}$
+
$W^T_{V\times N} \times x_{on} = v_{on}$
$\hat{v} = \frac{v_{cat} + v_{on}}{2}$
Hidden layer
N-dim
Output layer
sat
V-dim


Input layer
cat
V-dim
$W_{V\times N}$
on
V-dim
$W_{V\times N}$
Hidden layer
$\hat{v}$
N-dim
Output layer
$W'_{V\times N} \times \hat{v} = z$
$\hat{y} = softmax(z)$
$\hat{y}_{\text{sat}}$
V-dim
N will be the size of word vector

We would prefer $\hat{y}$ close to $\hat{y}_{sat}$



Cross Entropy Loss

- In binary classification, where the number of classes M equals 2, cross-entropy can be calculated as:

$$-(y\log(p) + (1-y)\log(1-p))$$

- If M>2 (i.e. multiclass classification), we calculate a separate loss for each class label per observation and sum the result.

$$-\sum_{c=1}^{M} y_{o,c}\log(p_{o,c})$$

- Remember the output of softmax function

$$\langle x_1, x_2, \ldots, x_k \rangle \mapsto \left\langle \frac{e^{x_1}}{\sum_{j=1}^{k} e^{x_j}}, \frac{e^{x_2}}{\sum_{j=1}^{k} e^{x_j}}, \ldots, \frac{e^{x_k}}{\sum_{j=1}^{k} e^{x_j}} \right\rangle$$

Example

0.1	0.3	0.3
0.2	0.4	0.3
0.7	0.3	0.4

"1"	"2"	"3"
1	0	0
0	1	0
0	0	1

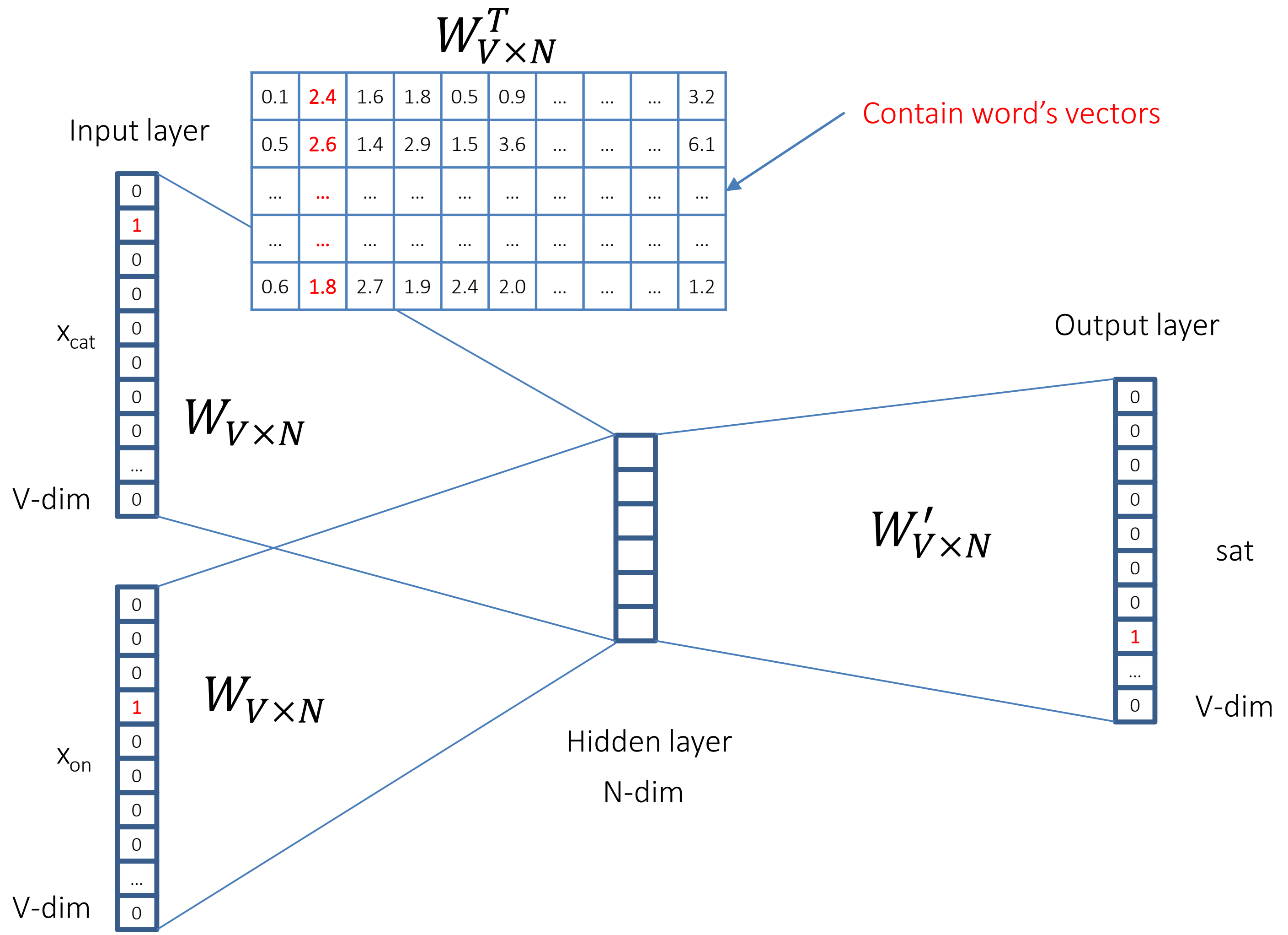
Classification accuracy = 2/3
Cross-entropy loss = 4.14

0.3	0.1	0.1
0.4	0.7	0.2
0.3	0.2	0.7

"1"	"2"	"3"
1	0	0
0	1	0
0	0	1

Classification accuracy = 2/3
Cross-entropy loss = 1.92

why-you-should-use-cross-entropy-error



We can consider either W or W' as the word's representation. Or even take the average.

Approximations

- With large vocabularies this objective function is not scalable and would train too slowly! → Why?
- Idea: approximate the normalization or
- Define negative prediction that only samples a few words that do not appear in the context
- Similar to focusing on mostly positive correlations
- More reading
  - https://canvas.ust.hk/courses/16504/files/1444107?module_item_id=214783

Linear Relationships in word2vec

- These representations are *very good* at encoding dimensions of similarity!
- Analogies testing dimensions of similarity can be solved quite well just by doing vector subtraction in the embedding space
  - Syntactically
    - *apple − apples ≈ car − cars ≈ family − families*
  - Similarly for verb and adjective morphological forms
  - Semantically (Semeval 2012 task 2)
    - *shirt − clothing ≈ chair − furniture*
    - *king − man ≈ queen − woman*

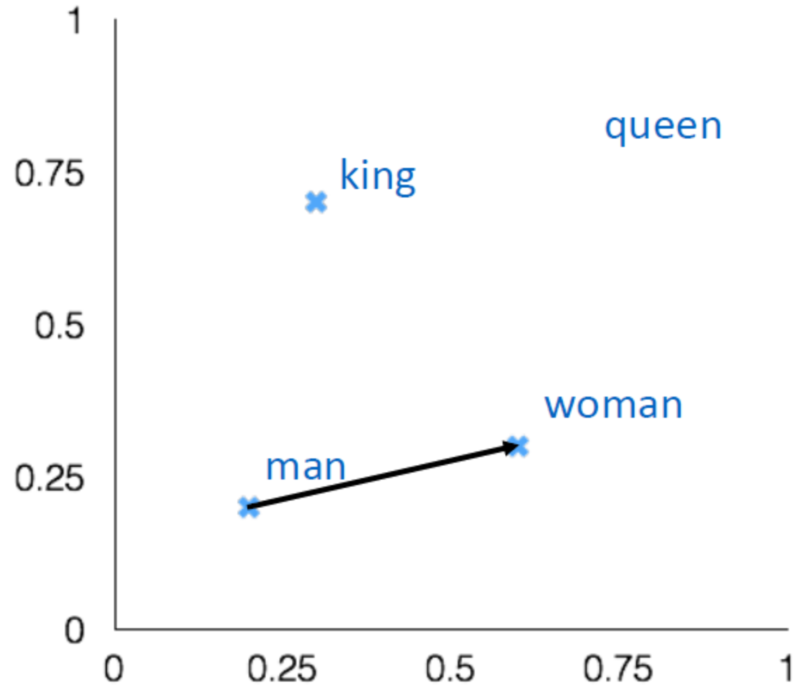
Word Analogies

- Test for linear relationships, examined by Mikolov et al. (2014)

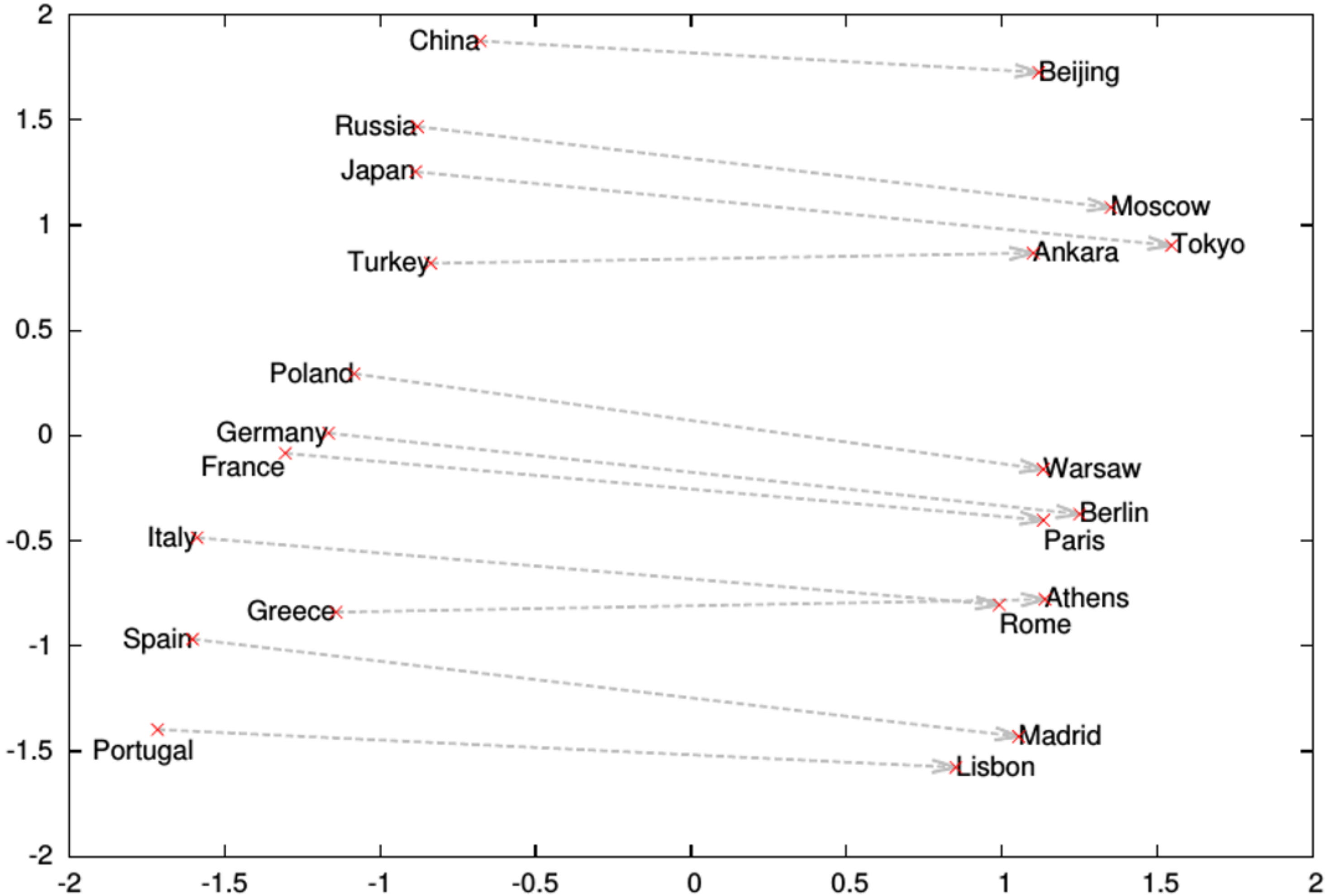
a:b :: c:? → $$d = \arg\max_x \frac{(w_b - w_a + w_c)^T w_x}{||w_b - w_a + w_c||}$$

man:woman :: king:?

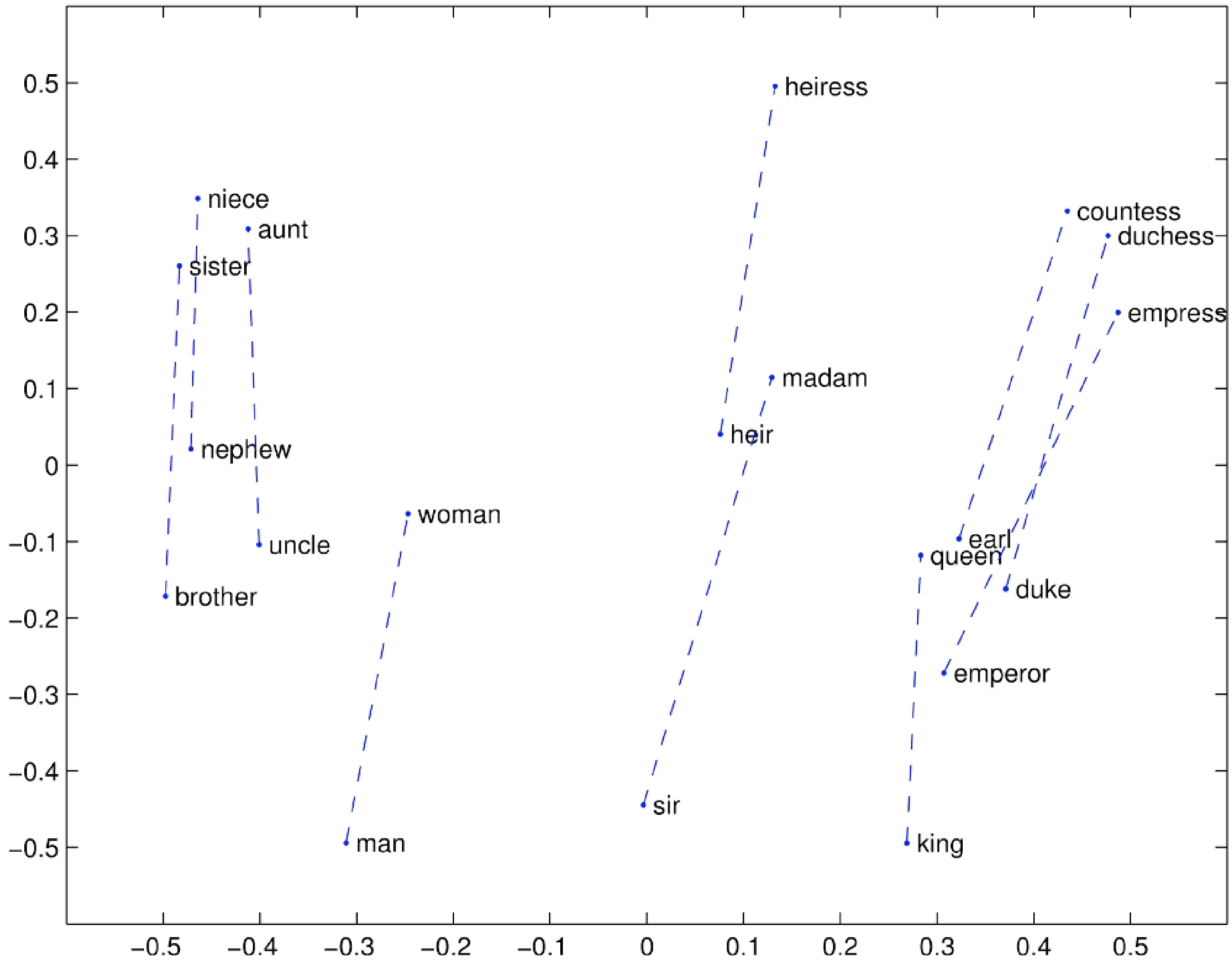
+	king	[0.30 0.70]
-	man	[0.20 0.20]
+	woman	[0.60 0.30]
	queen	[0.70 0.80]



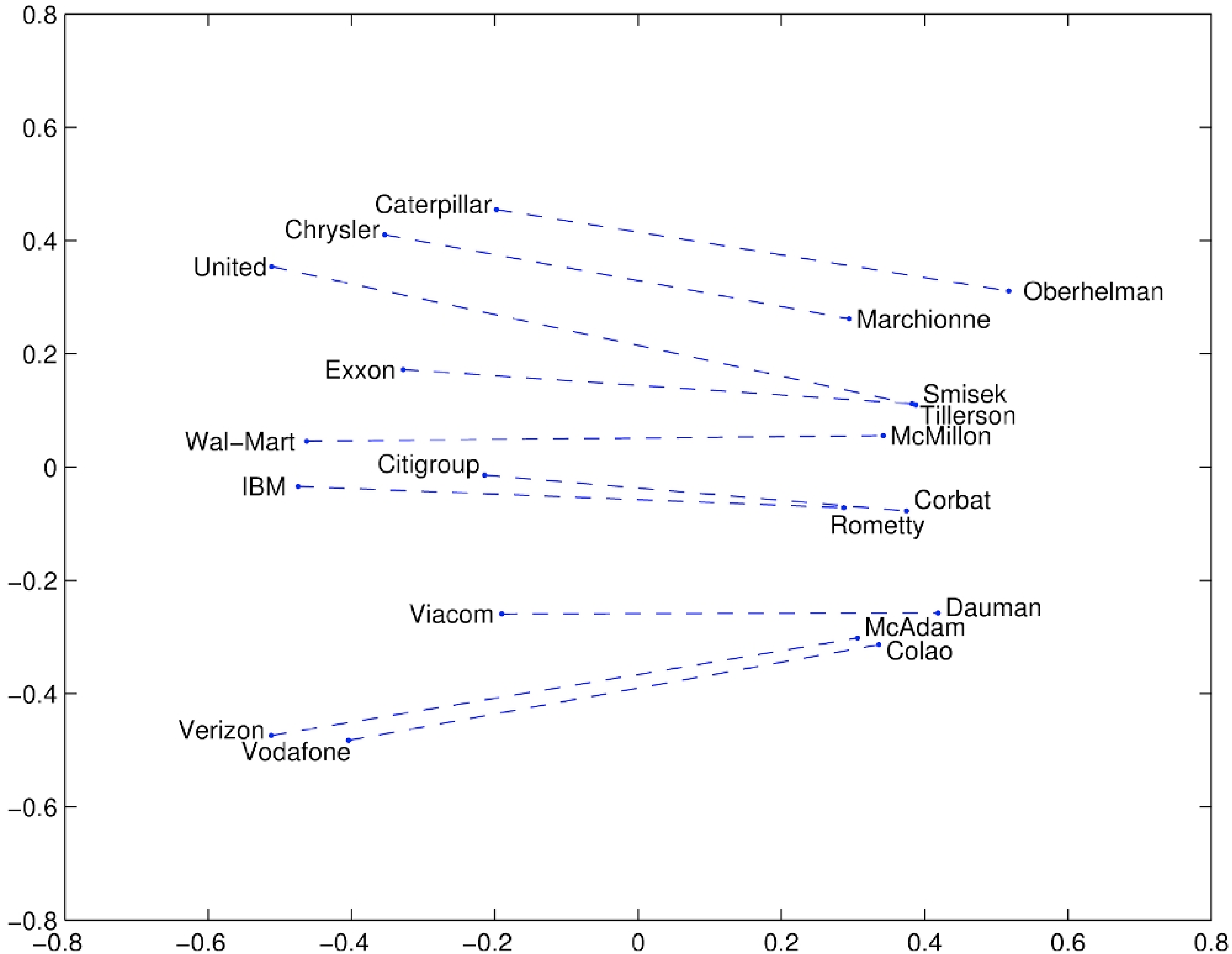
Word analogies



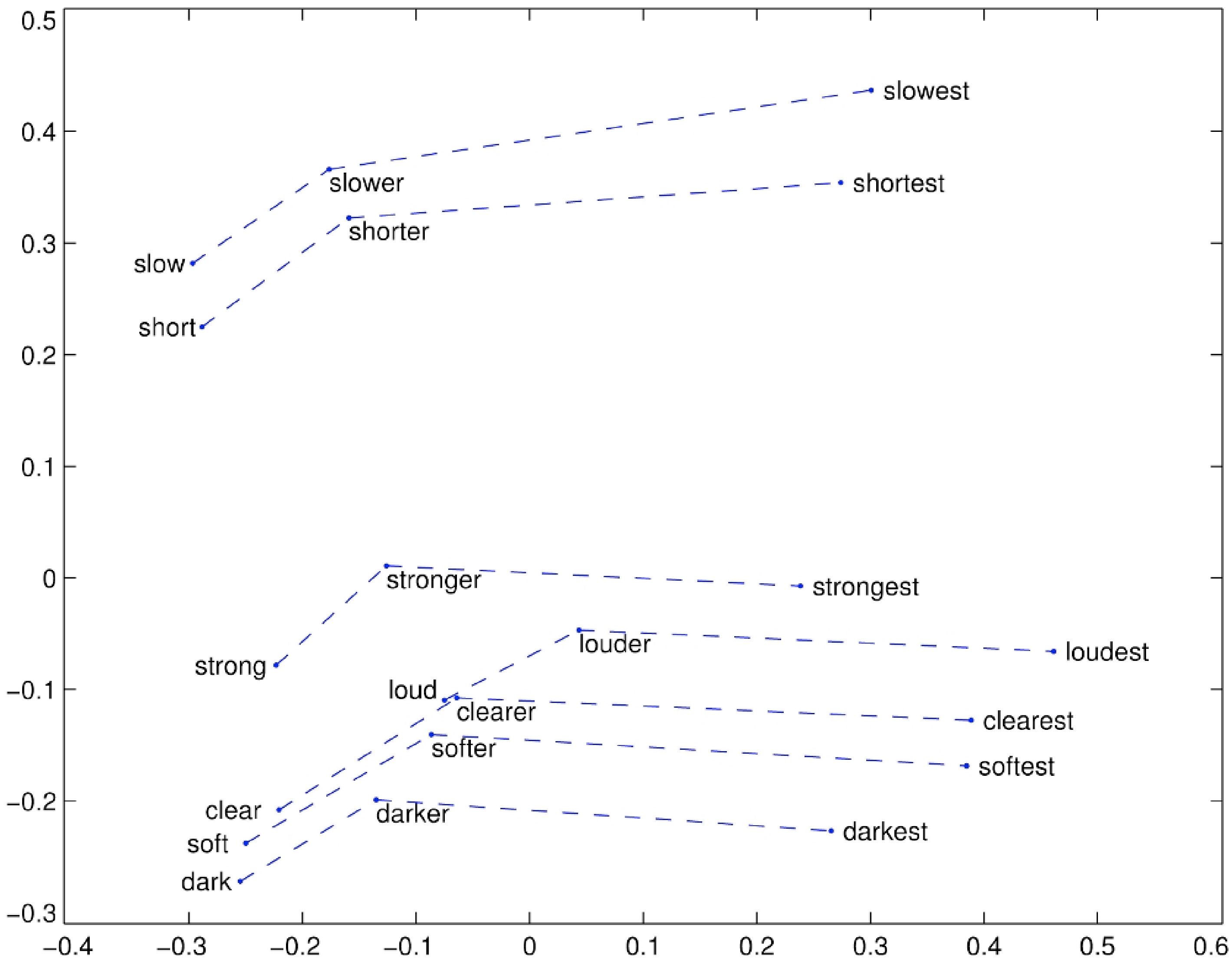
Glove Visualizations



Glove Visualizations: Company - CEO



Glove Visualizations: Superlatives



More Examples

- “word2vec Parameter Learning Explained”, Xin Rong
  - https://ronxin.github.io/wevi/
- Word2Vec Tutorial - The Skip-Gram Model
  - http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Deep Contextualized Word Representation

- As most NLP tasks are context related, most of existing methods would contextualize the word embedding before make the final prediction
- However, complicated neural models requires extensive training data:
  - Models pre-trained on the ImageNet are widely used for Computer Vision tasks
  - What's the proper way to conduct pre-training for NLP?
- Basic Idea
  - Leveraging Language Modeling to get pre-trained contextualized representation models
- Highlight:
  - 1. rely on large corpora, instead of human annotations
  - 2. works very well ---- improve the performance of existing SOA methods a lot

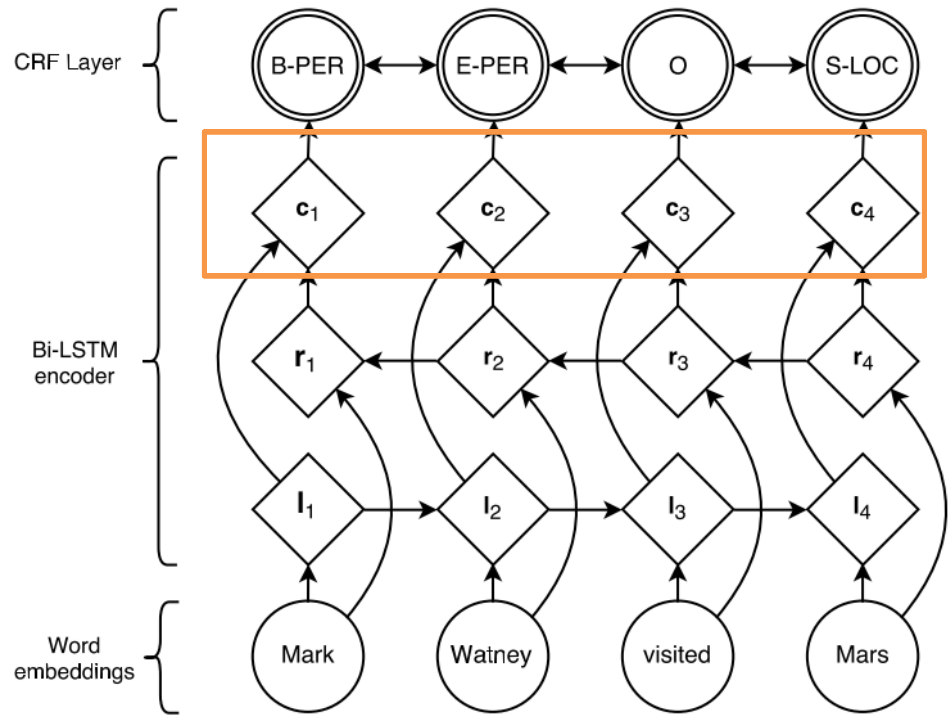


Figure 1: Main architecture of the network. Word embeddings are given to a bidirectional LSTM. $\mathbf{l}_i$ represents the word i and its left context, $\mathbf{r}_i$ represents the word i and its right context. Concatenating these two vectors yields a representation of the word i in its context, $\mathbf{c}_i$.

http://aclweb.org/anthology/N18-1202

Experiments

- Add ELMo at the input of RNN. For some tasks (SNLI, SQuAD), including ELMo at the output brings further improvements

TASK	PREVIOUS SOTA		OUR BASELINE	ELMO + BASELINE	INCREASE (ABSOLUTE/ RELATIVE)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

Table 1: Test set comparison of ELMo enhanced neural models with state-of-the-art single model baselines across six benchmark NLP tasks. The performance metric varies across tasks – accuracy for SNLI and SST-5; F_1 for SQuAD, SRL and NER; average F_1 for Coref. Due to the small test sizes for NER and SST-5, we report the mean and standard deviation across five runs with different random seeds. The "increase" column lists both the absolute and relative improvements over our baseline.

Google's BERT

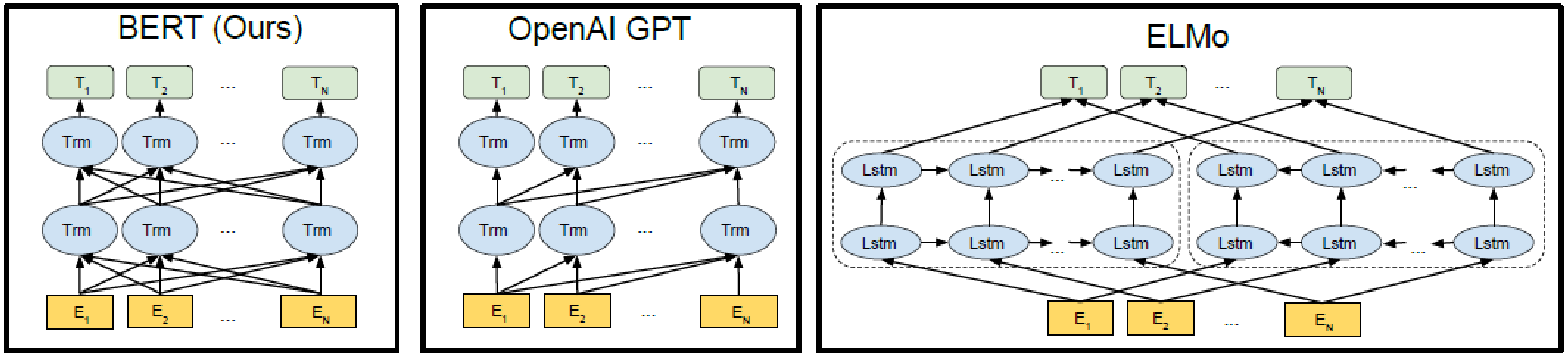


Figure 1: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTM to generate features for downstream tasks. Among three, only BERT representations are jointly conditioned on both left and right context in all layers.

- Training of BERT-BASE was performed on 4 Cloud TPUs in Pod configuration (16 TPU chips total).
- Training of BERT-LARGE was performed on 16 Cloud TPUs (64 TPU chips total).
- Each pre-training took 4 days to complete.

https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html
https://arxiv.org/pdf/1810.04805.pdf

Experiments

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**

Table 1: GLUE Test results, scored by the GLUE evaluation server. The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set. OpenAI GPT = (L=12, H=768, A=12); $\text{BERT}_{\text{BASE}}$ = (L=12, H=768, A=12); $\text{BERT}_{\text{LARGE}}$ = (L=24, H=1024, A=16). BERT and OpenAI GPT are single-model, single task. All results obtained from `https://gluebenchmark.com/leaderboard` and `https://blog.openai.com/language-unsupervised/`.