Neural Machine Translation

Manish Shrivastava

Reference Materials

- Deep Learning for NLP by Richard Socher http://cs224d.stanford.edu/
- Tutorial and Visualization tool by Xin Rong http://www-personal.umich.edu/~ronxin/pdf/w2vexp.pdf
- Word2vec in Gensim by Radim Řehůřek http://rare-technologies.com/deep-learning-with-word2vec-and-gensim/
- Slides by Girish K, Vagelis Hristidis, Richard Socher and many others.

Deep Learning

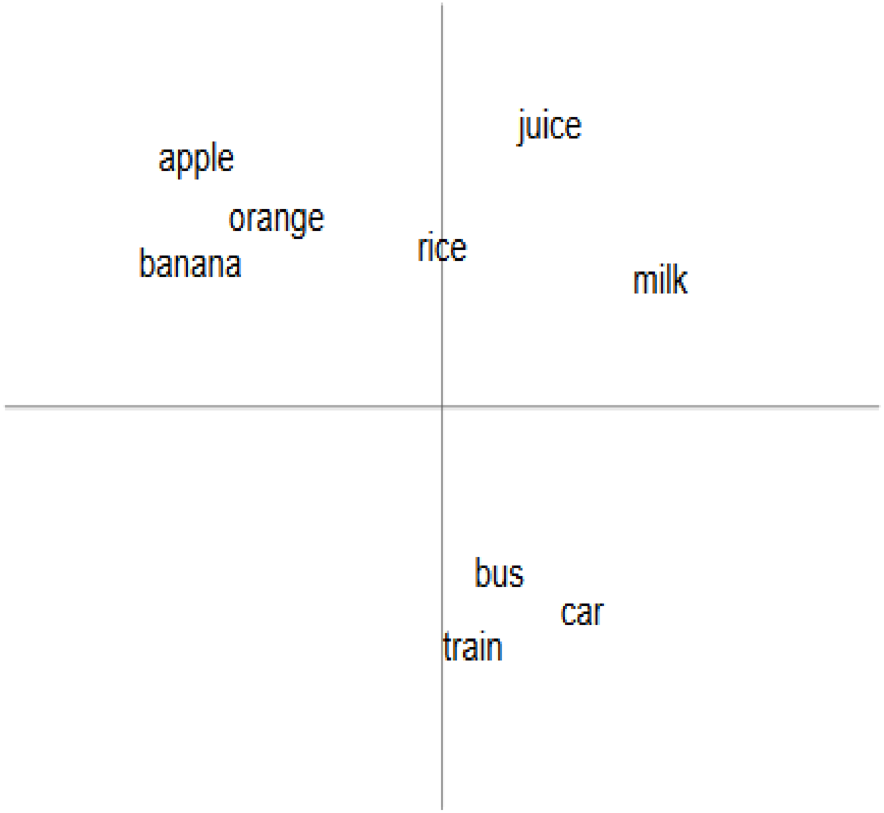
- Number of Applications
- Primary premise:
  - Word Representation : Words represented as dense vectors embedded in a vector space
- Many different approaches to learn
- Many different applications
  - Some new ones : Word Analogies, Semantic Similarity etc.

Role of Deep Learning in NLP

- Similarity?
  - France:Paris::Russia:Moscow
  - E(King)-E(Man)+E(Woman) =E(Queen)

“You shall know a word by the company it keeps!”

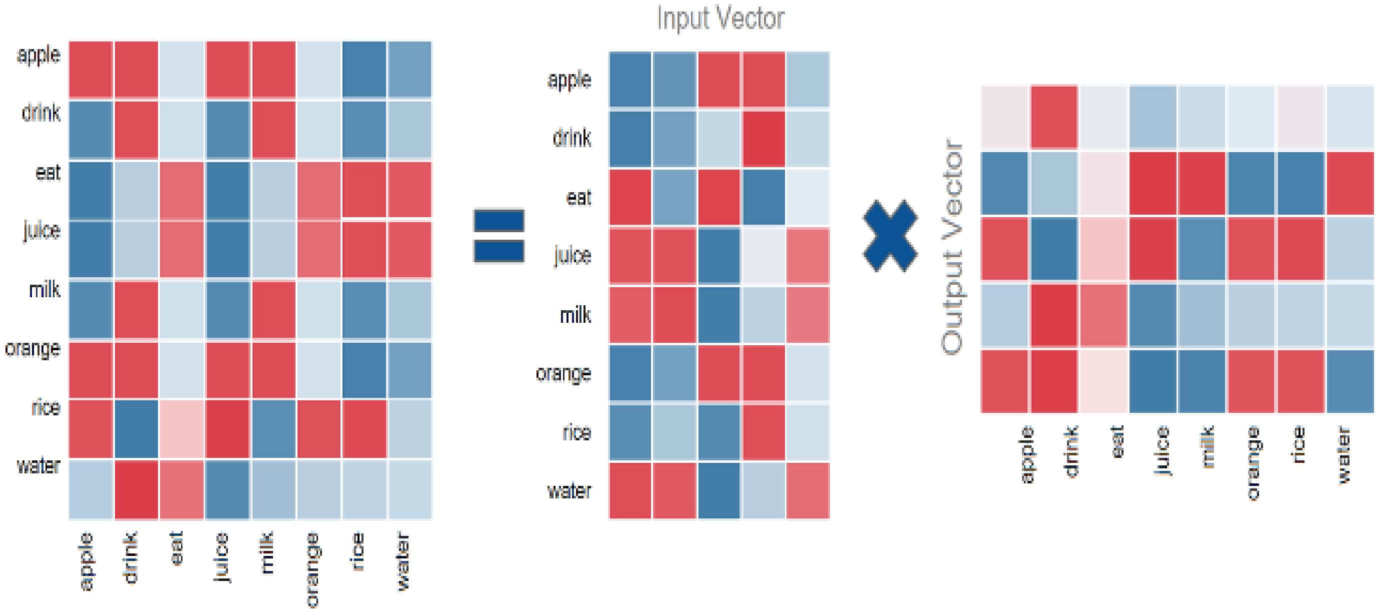
Firth (1957)



Role of Deep Learning in NLP

- Similarity?
    - Not Really!!
- Methods for Semantic Similarity have existed for a long time.
    - Distributional Semantics

Singular Value Decomposition



The problem with this method, is that we may end up with matrices having billions of rows and columns, which makes SVD computationally restrictive.

Singular Value Decomposition (SVD)

- Handy mathematical technique that has application to many problems
- Given any $m \times n$ matrix $\mathbf{A}$, algorithm to find matrices $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{W}$ such that

$$\mathbf{A} = \mathbf{U}\,\mathbf{W}\,\mathbf{V}^T$$

$\mathbf{U}$ is $m \times n$ and orthonormal

$\mathbf{W}$ is $n \times n$ and diagonal

$\mathbf{V}$ is $n \times n$ and orthonormal

SVD

$$\begin{pmatrix} & & \\ & \mathbf{A} & \\ & & \end{pmatrix} = \begin{pmatrix} & & \\ & \mathbf{U} & \\ & & \end{pmatrix} \begin{pmatrix} w_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & w_n \end{pmatrix} \begin{pmatrix} & & \\ & \mathbf{V} & \\ & & \end{pmatrix}^{\mathrm{T}}$$

- Treat as black box: code widely available

SVD

- The w_i are called the singular values of **A**
- If **A** is singular, some of the w_i will be 0
- In general *rank*(**A**) = number of nonzero w_i
- SVD is mostly unique (up to permutation of singular values, or if some w_i are equal)

SVD and Inverses

- Why is SVD so useful?
- Application #1: inverses
- $\mathbf{A}^{-1} = (\mathbf{V}^T)^{-1} \mathbf{W}^{-1} \mathbf{U}^{-1} = \mathbf{V} \mathbf{W}^{-1} \mathbf{U}^T$
  - Using fact that inverse = transpose for orthogonal matrices
  - Since $\mathbf{W}$ is diagonal, $\mathbf{W}^{-1}$ also diagonal with reciprocals of entries of $\mathbf{W}$

SVD and Inverses

- $\mathbf{A}^{-1} = (\mathbf{V}^T)^{-1}\,\mathbf{W}^{-1}\,\mathbf{U}^{-1} = \mathbf{V}\,\mathbf{W}^{-1}\,\mathbf{U}^T$
- This fails when some w_i are 0
  - It's *supposed* to fail – singular matrix
- Pseudoinverse: if w_i=0, set $1/w_i$ to 0 (!)
  - "Closest" matrix to inverse
  - Defined for all (even non-square, singular, etc.) matrices
  - Equal to $(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$ if $\mathbf{A}^T\mathbf{A}$ invertible

SVD and Eigenvectors

- Let $\mathbf{A}=\mathbf{UWV}^T$, and let x_i be i^{th} column of $\mathbf{V}$
- Consider $\mathbf{A}^T\mathbf{A}\,x_i$:

$$\mathbf{A}^T\mathbf{A}x_i = \mathbf{VW}^T\mathbf{U}^T\mathbf{UWV}^T x_i = \mathbf{VW}^2\mathbf{V}^T x_i = \mathbf{VW}^2\begin{pmatrix}0\\ \vdots\\ 1\\ \vdots\\ 0\end{pmatrix} = \mathbf{V}\begin{pmatrix}0\\ \vdots\\ w_i^{\,2}\\ \vdots\\ 0\end{pmatrix} = w_i^{\,2}x_i$$

- So elements of **W** are sqrt(eigenvalues) and columns of **V** are eigenvectors of $\mathbf{A}^T\mathbf{A}$
  - What we wanted for robust least squares fitting!

Word Representations

Traditional Method - Bag of Words Model	Word Embeddings
• Uses one hot encoding • Each word in the vocabulary is represented by neighbouring words • For example, if we have a vocabulary of 10000 words, and “Hello” is a word in the dictionary, it would be represented by: [12 3 1 0 0 4 1 50 … 0] • Each number (in this instance) is the frequency of surrounding words in all contexts • Dimensions are words.	• Stores each word in as a point in space, where it is represented by a vector of fixed number of dimensions (generally 300) • Unsupervised, built just by reading huge corpus • For example, “Hello” might be represented as : [0.4, -0.11, 0.55, 0.3 . . . 0.1, 0.02] • Dimensions are basically projections along different axes, more of a mathematical concept.

Applications of Word Vectors

1. <u>Word Similarity</u>

- Easily identifies similar words and synonyms since they occur in similar contexts
- Stemming (thought -> think)
- Inflections, Tense forms
- *eg. Think, thought, ponder, pondering,*
- *eg. Plane, Aircraft, Flight*

Applications of Word Vectors

2. Machine Translation

- I am a cook .
  - நான் ஒரு சமையல்காரன்.
  - मैं एक बावर्ची हूं ।
  - Je suis un cuisinier .

Applications of Word Vectors

3. <u>Part-of-Speech and Named Entity Recognition</u>

	POS WSJ (acc.)	NER CoNLL (F1)
State-of-the-art*	97.24	89.31
Supervised NN	**96.37**	**81.47**
Unsupervised pre-training followed by supervised NN**	**97.20**	**88.87**
+ hand-crafted features***	97.29	89.59

Applications of Word Vectors

4. Relation Extraction

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Applications of Word Vectors

5. Sentiment Analysis

Cassifying sentences as positive and negative

- Building sentiment lexicons using seed sentiment sets
- No need for classifiers, we can just use cosine distances to compare unseen reviews to known reviews.

```
Enter word or sentence (EXIT to break): sad

Word: sad  Position in vocabulary: 4067

                                          Word       Cosine distance
--------------------------------------------------------------------
                                     saddening              0.727309
                                           Sad              0.661083
                                      saddened              0.660439
                                  heartbreaking             0.657351
                                  disheartening             0.650732
                                  Meny_Friedman             0.648706
                       parishioner_Pat_Patello              0.647586
                                     saddens_me             0.640712
                                    distressing             0.639909
                              reminders_bobbing             0.635772
                                Turkoman_Shiites            0.635577
                                       saddest              0.634551
                                    unfortunate             0.627209
                                         sorry              0.619405
                                   bittersweet              0.617521
                                        tragic              0.611279
                                     regretful              0.603472
```

Applications of Word Vectors

6. Co-reference Resolution

- Chaining entity mentions across multiple documents - can we find and unify the multiple contexts in which mentions occurs?

7. Clustering

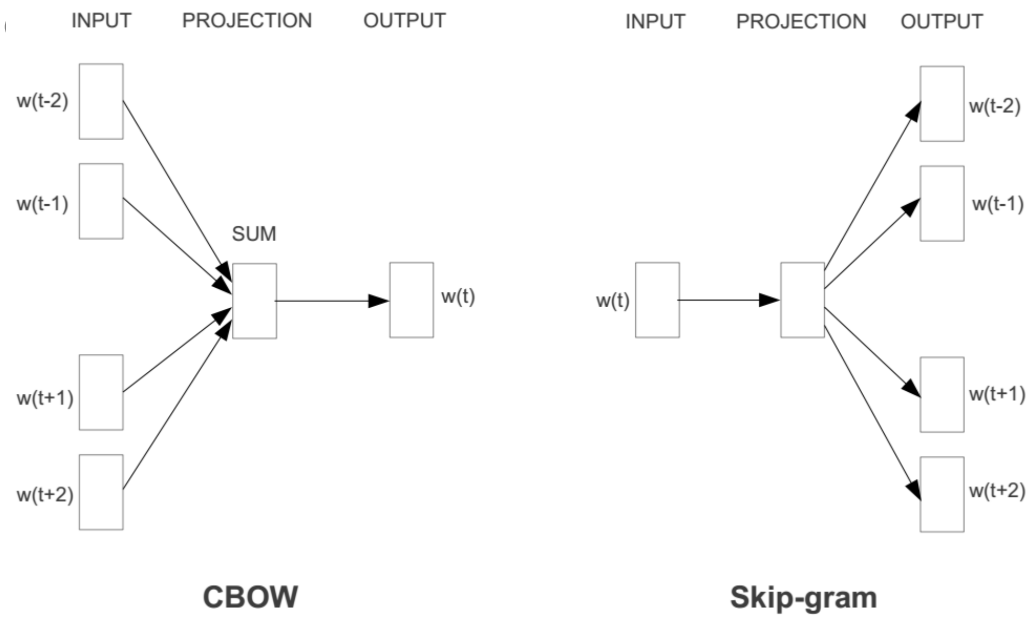
- Words in the same class naturally occur in similar contexts, and this feature vector can directly be used with any conventional clustering algorithms (K-Means, agglomerative, etc). Human doesn’t have to waste time hand-picking useful word features to cluster on.

8. Semantic Analysis of Documents

- Build word distributions for various topics, etc.

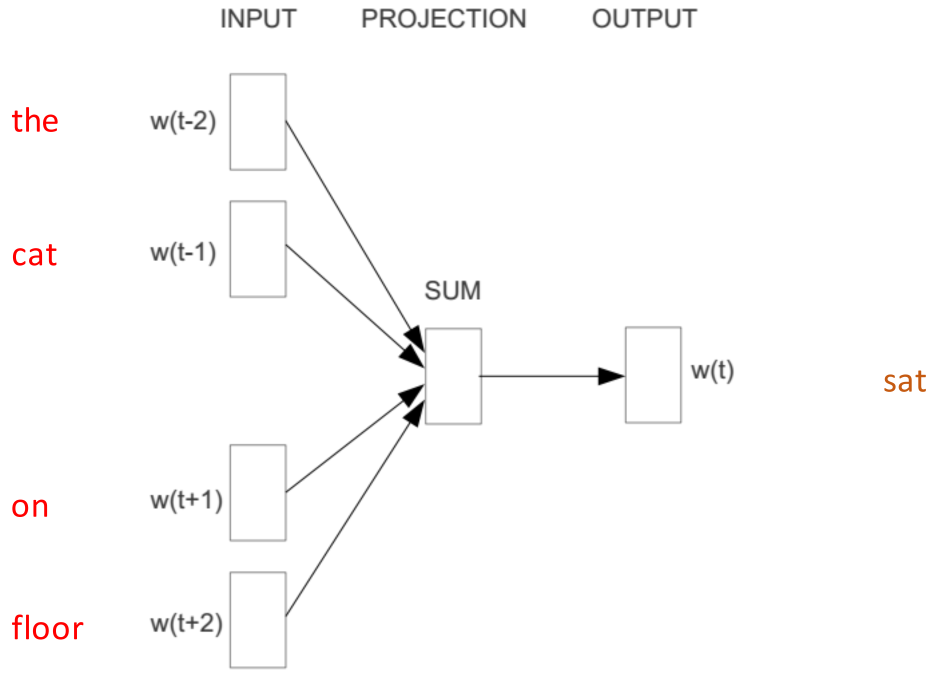
Represent the meaning of word **Word2vec**

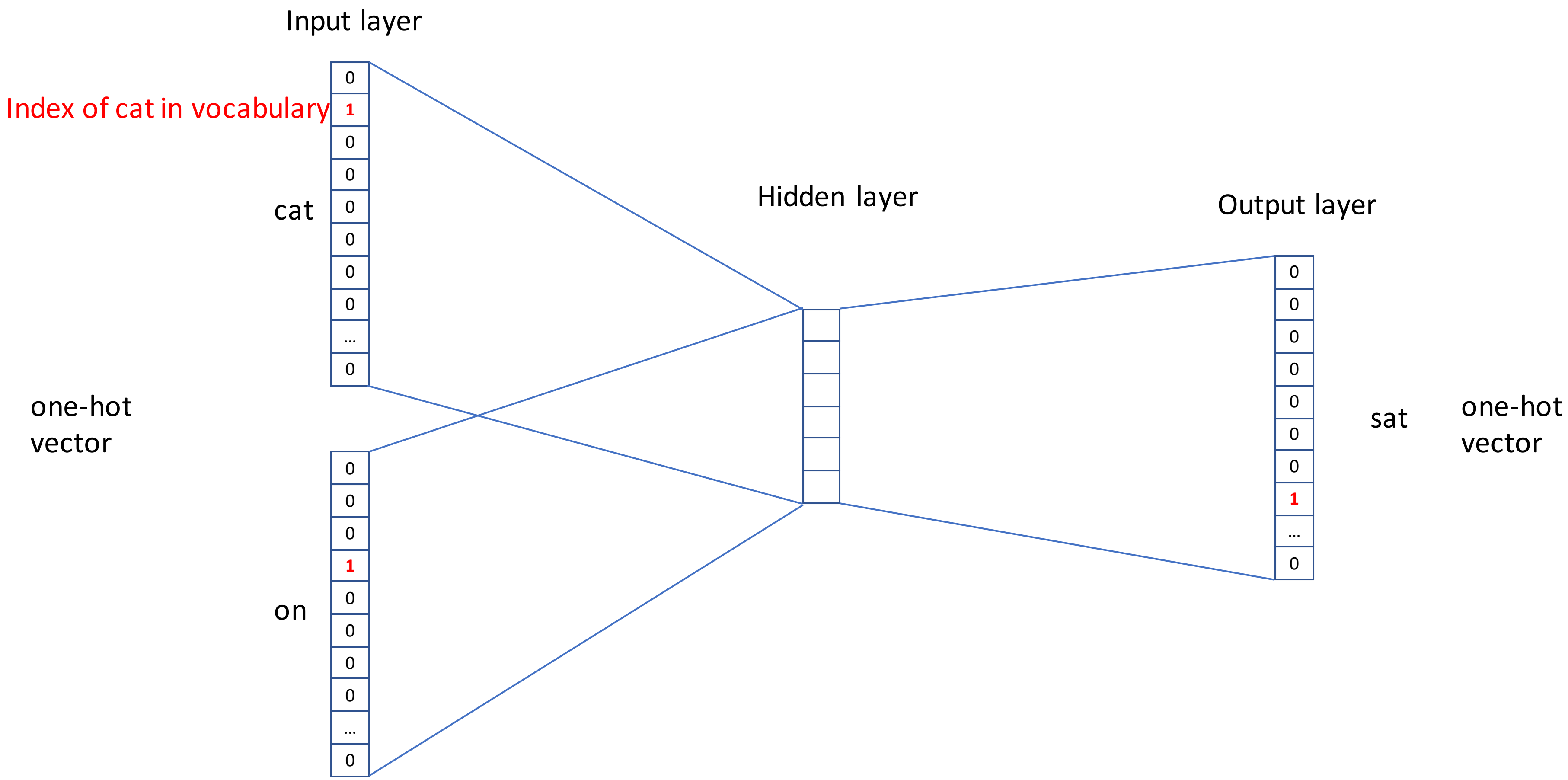
- 2 basic neural network models:
  - Continuous Bag of Word (CBOW): use a window of word to predict the middle word
  - Skip-gram (SG): use a word to predict the surrounding ones in win

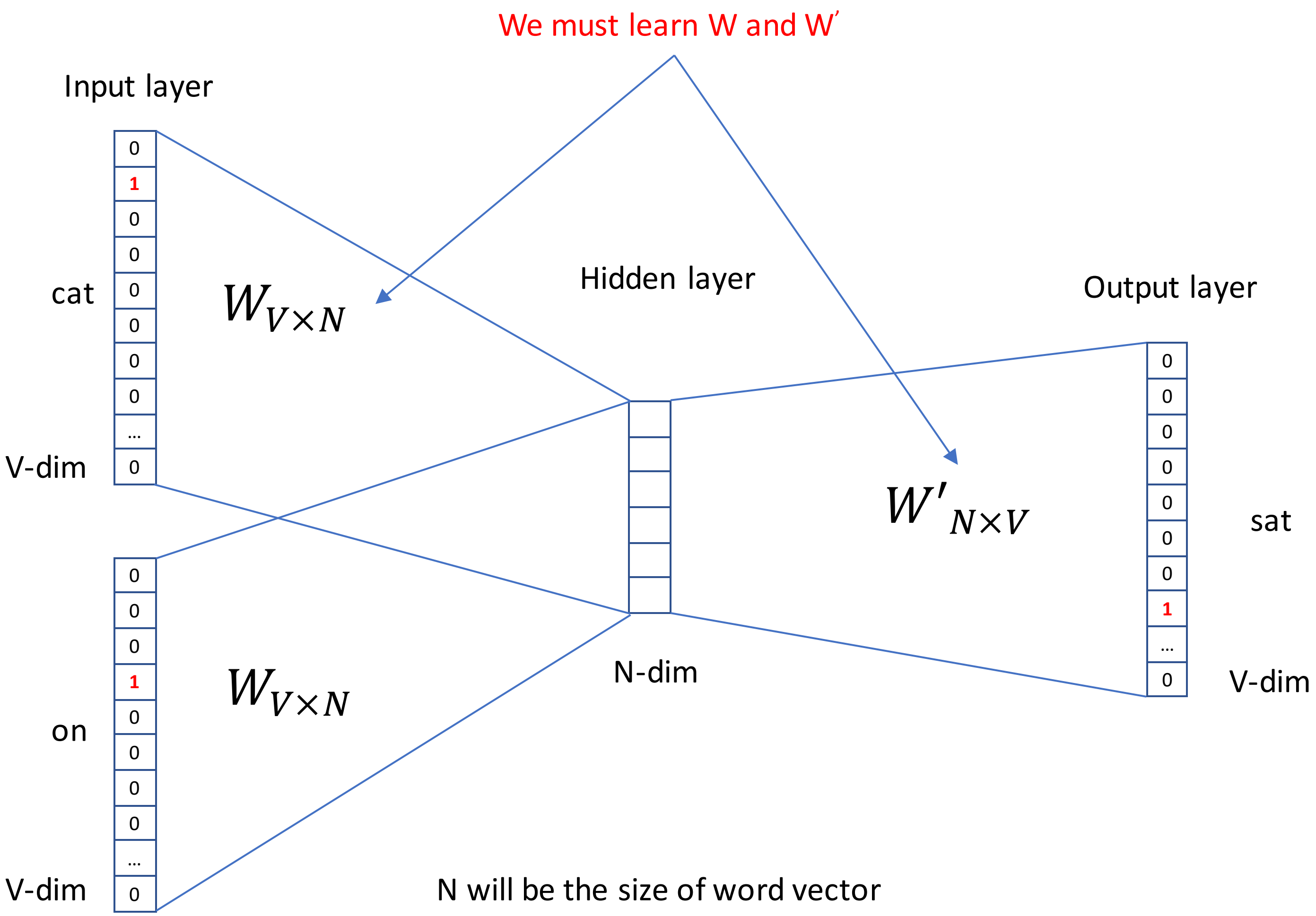


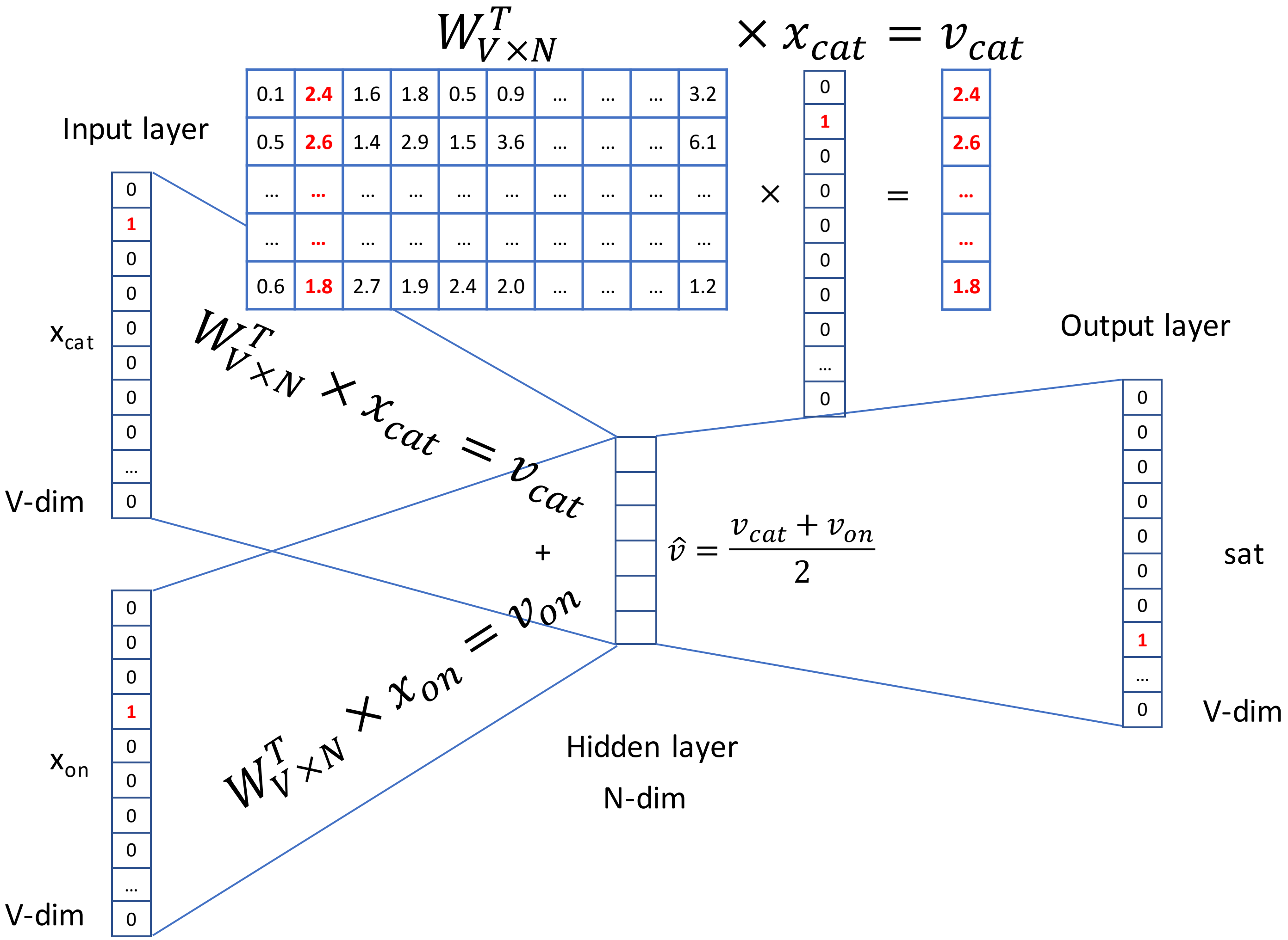
Word2vec – Continuous Bag of Word

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

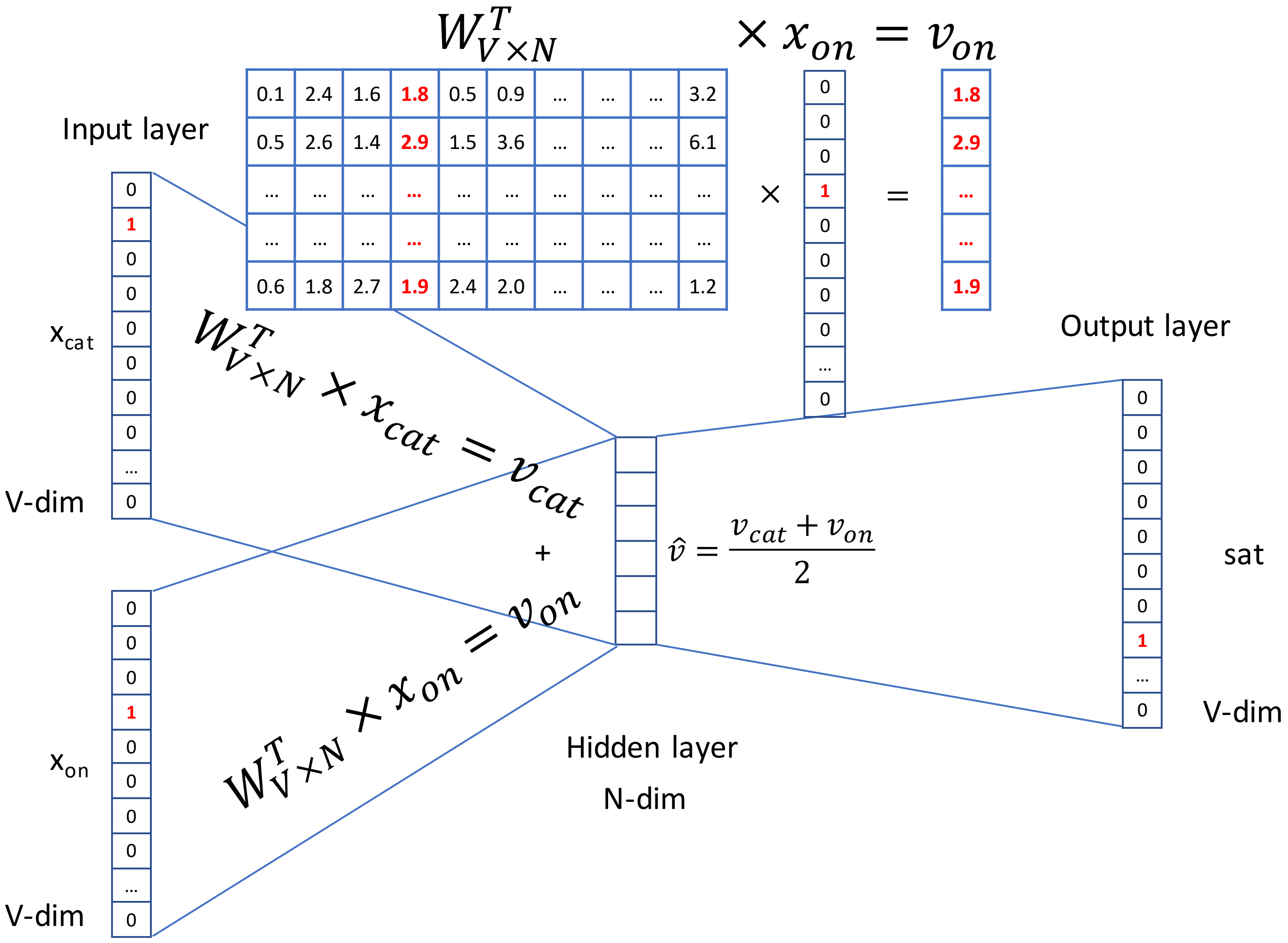



Input layer
Index of cat in vocabulary
0
1
0
0
0
0
0
0
...
0
cat
Hidden layer
Output layer
0
0
0
0
0
0
0
1
...
0
sat
one-hot
vector
one-hot
vector
0
0
0
1
0
0
0
0
...
0
on

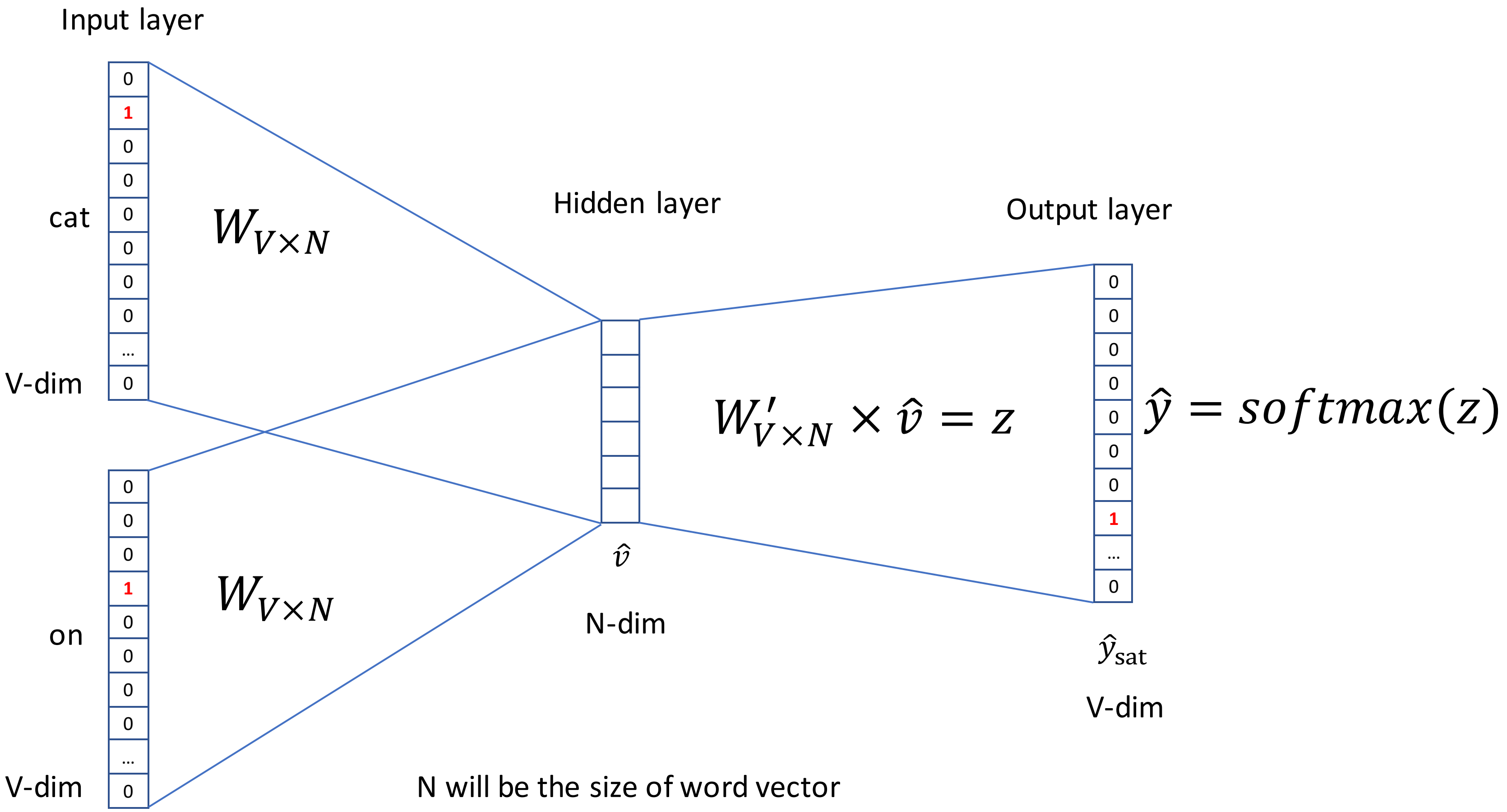

We must learn W and W′
Input layer
cat
$W_{V \times N}$
V-dim
on
$W_{V \times N}$
V-dim
Hidden layer
N-dim
Output layer
$W'_{N \times V}$
sat
V-dim
0
1
0
0
0
0
0
0
…
0
0
0
0
1
0
0
0
0
…
0
0
0
0
0
0
0
0
1
…
0
N will be the size of word vector



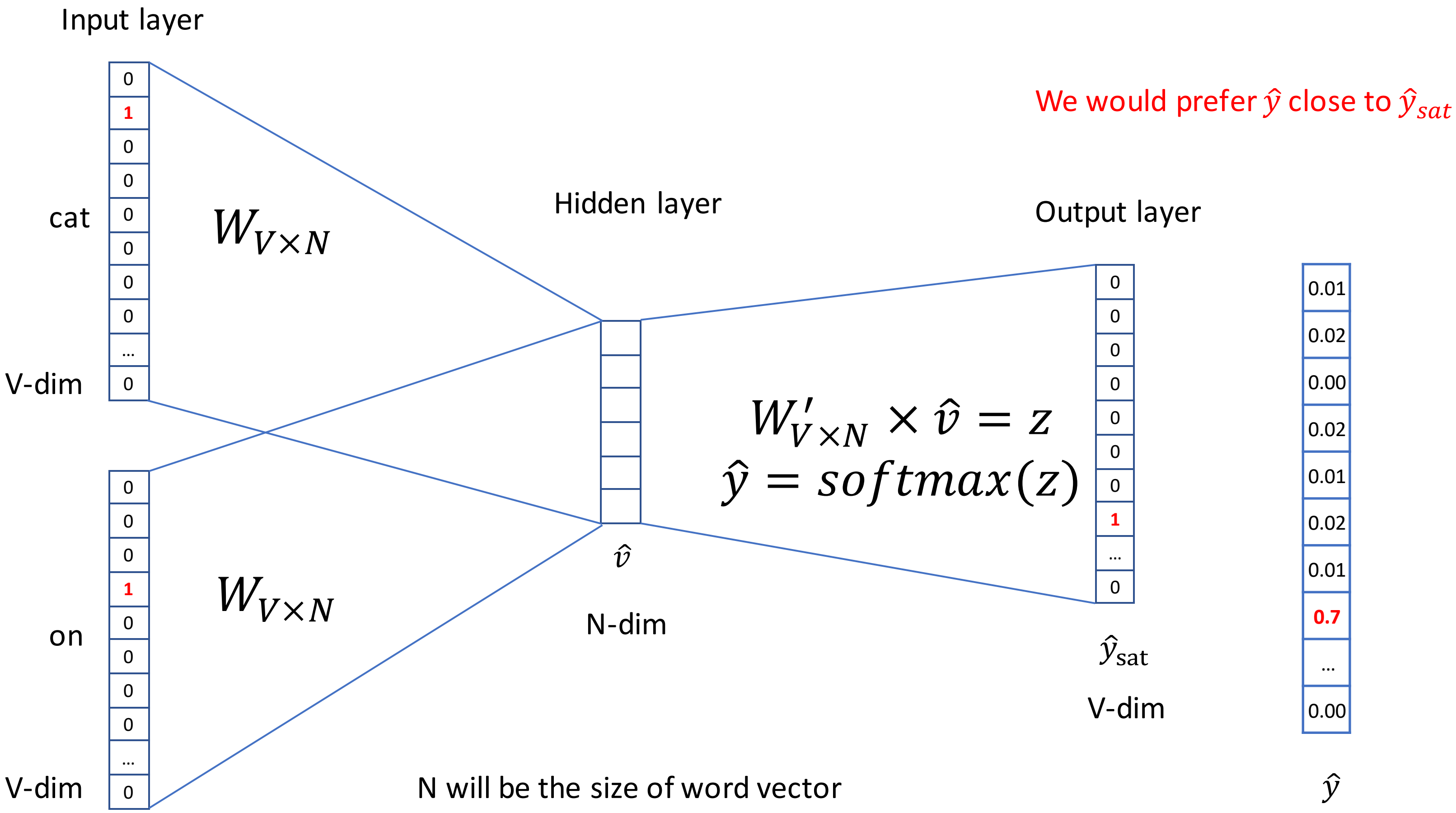
$W_{V\times N}^T \times x_{cat} = v_{cat}$
Input layer
x_{cat}
V-dim
x_{on}
V-dim
$W_{V\times N}^T \times x_{cat} = v_{cat}$
+
$W_{V\times N}^T \times x_{on} = v_{on}$
$\hat{v} = \frac{v_{cat} + v_{on}}{2}$
Hidden layer
N-dim
Output layer
sat
V-dim

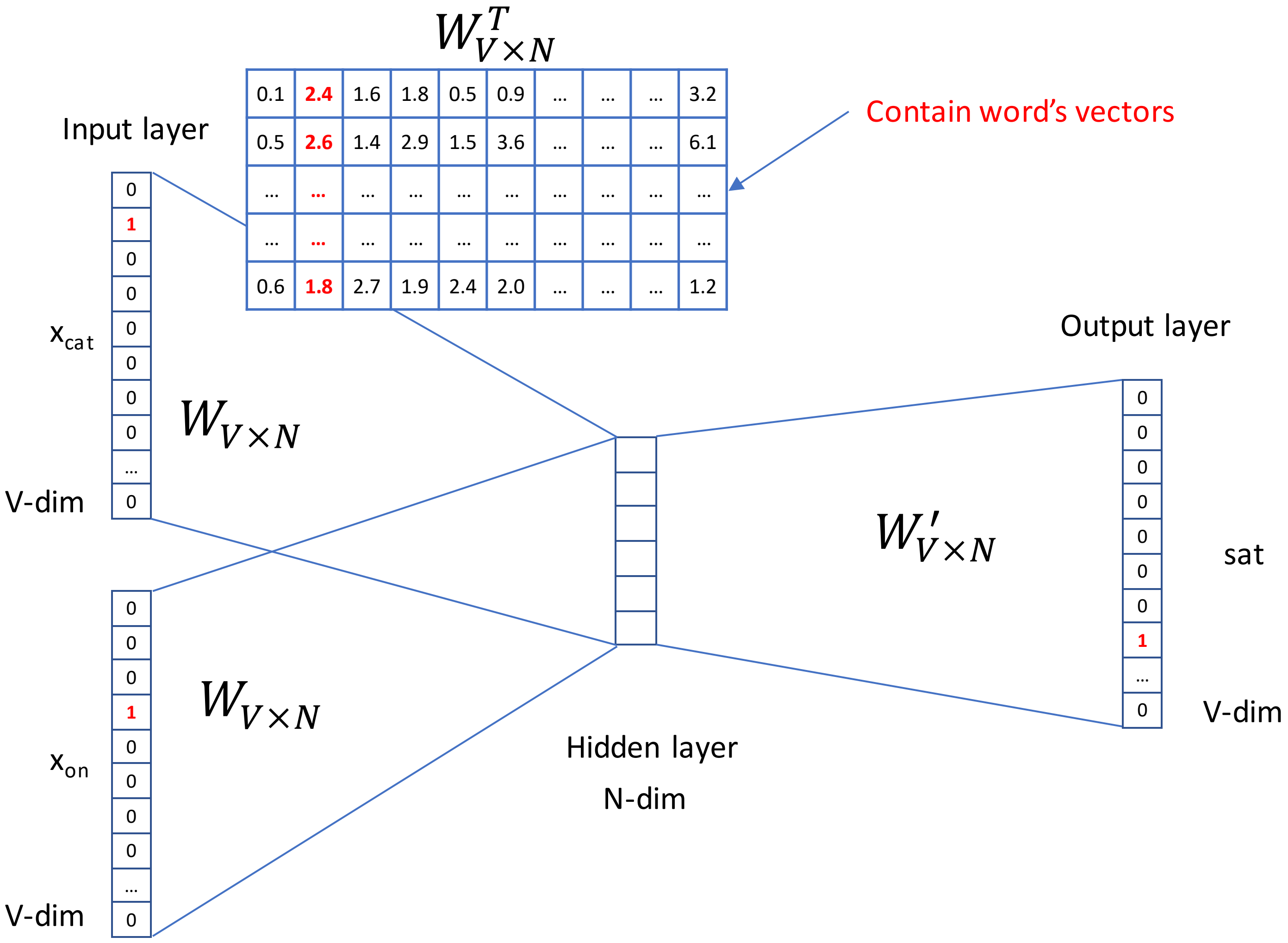


$W_{V\times N}^{T}$
$\times x_{on} = v_{on}$
Input layer
x_{cat}
V-dim
$W_{V\times N}^{T} \times x_{cat} = v_{cat}$
+
$W_{V\times N}^{T} \times x_{on} = v_{on}$
x_{on}
V-dim
$\hat{v} = \frac{v_{cat} + v_{on}}{2}$
Hidden layer
N-dim
Output layer
sat
V-dim



N will be the size of word vector


We would prefer $\hat{y}$ close to $\hat{y}_{sat}$
Input layer
cat
V-dim
on
V-dim
$W_{V\times N}$
$W_{V\times N}$
Hidden layer
$\hat{v}$
N-dim
Output layer
$W'_{V\times N} \times \hat{v} = z$
$\hat{y} = softmax(z)$
$\hat{y}_{sat}$
V-dim
$\hat{y}$
0.01
0.02
0.00
0.02
0.01
0.02
0.01
0.7
...
0.00
N will be the size of word vector



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

Some interesting results

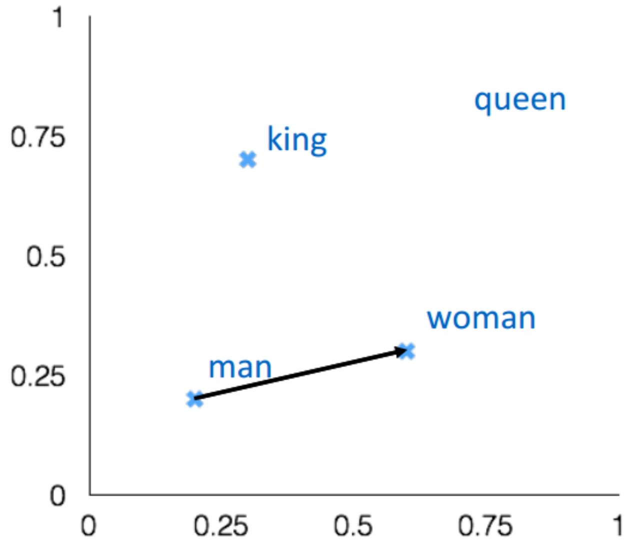
Word Analogies

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

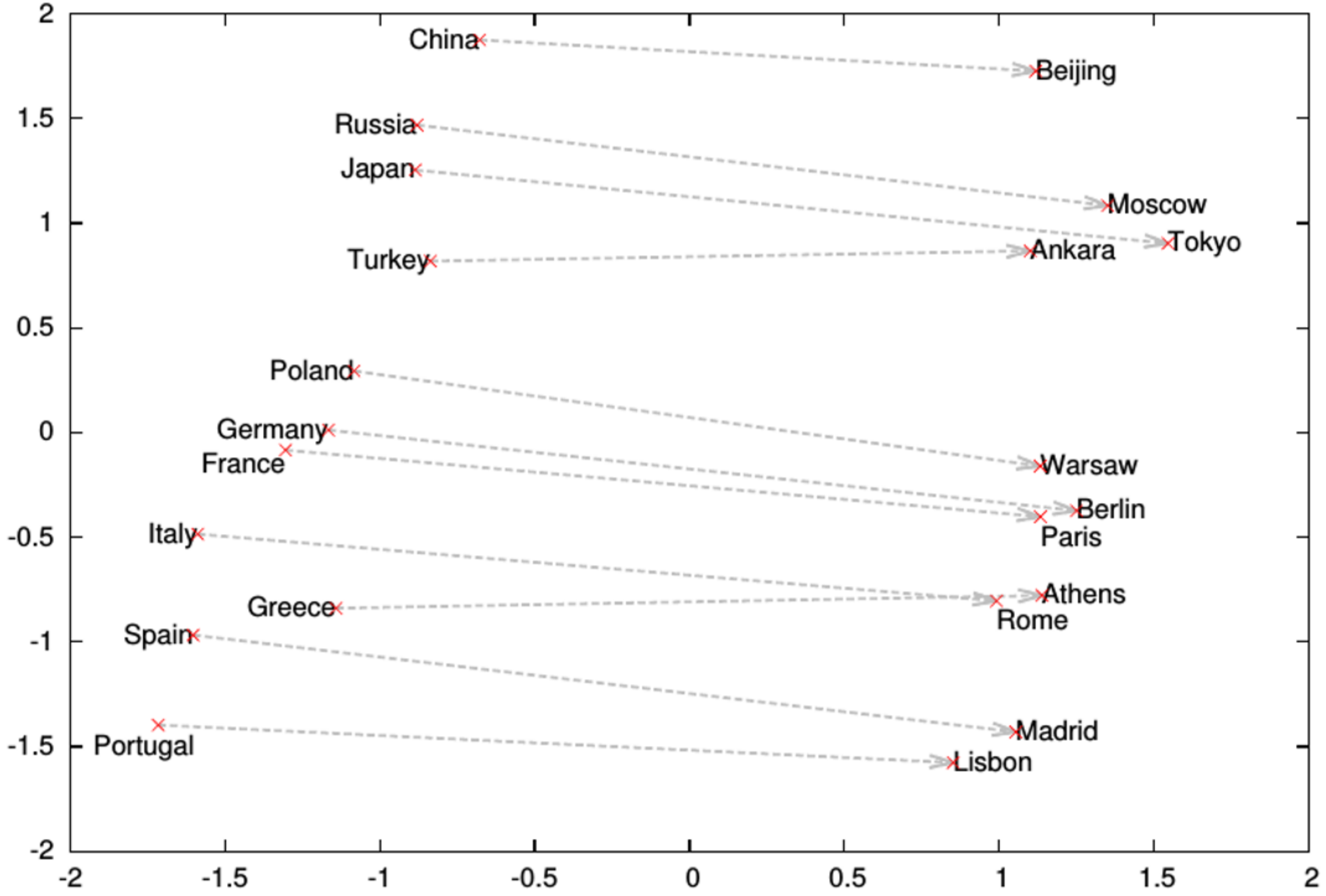
a:b :: c:? $\longrightarrow$ $$d = \arg\max_x \frac{(w_b - w_a + w_c)^T w_x}{||w_b - w_a + w_c||}$$

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



Now what?

- Word2Vec == Deep Learning
  - No!
  - The first step
- Embeddings are dense vectors
  - Can encapsulate context information
  - Can serve as information points similar to pixels
  - Serve as input to Deep learning models
  - CNNs, RNNs and many more

Deep Learning and NLP

- Tasks
  - Language Modeling
  - Semantic encoding of larger sentential units
  - Semantic similarity at larger granularity for various tasks

Language Modelling Problem

- Aim is to calculate the probability of a sequence (sentence) P(X)
- Can be decomposed into product of conditional probabilities of tokens (works):

$$P(\boldsymbol{W}) = P(\mathrm{w}(1),\mathrm{w}(2),\mathrm{w}(3),\ldots,\mathrm{w}(\mathrm{M}+1)) = \prod_{k=1}^{\mathrm{M}+1} P\big(w(k)|w(1),\ldots,w(k-1)\big)$$

- In practice, only finite content used

$$P\big(w(k)|w(1),\ldots,w(k-1)\big) \approx P\big(w(k)|w(1),\ldots,w(k-N+1)\big)$$

N-Gram Language Model

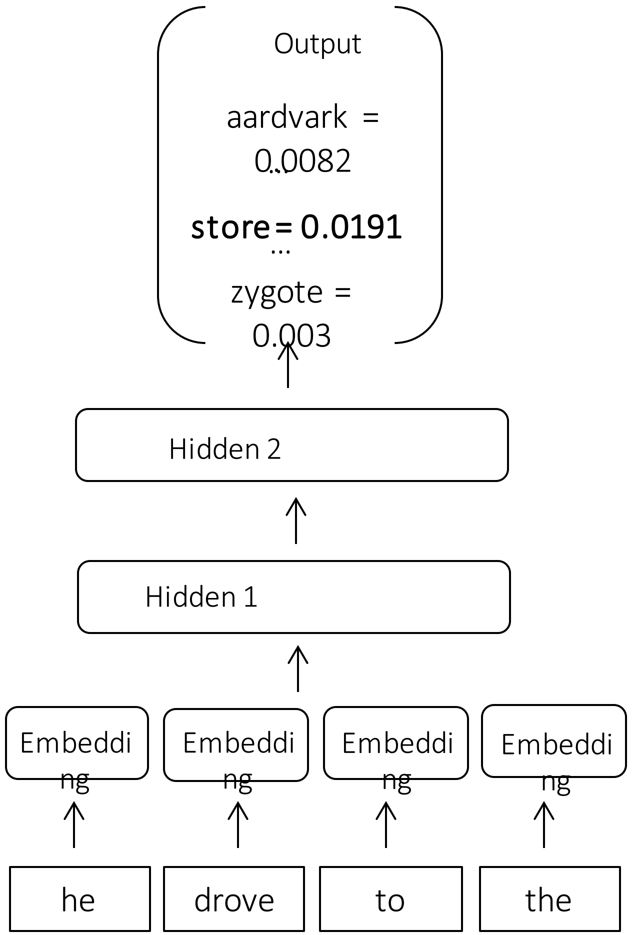
- N-Grams estimate word conditional probabilities via counting:

$$P\left(w(k)|h_{k-N+1}^{k-1}\right) = \frac{Count\left[w(k), h_{k-N+1}^{k-1}\right]}{Count\left[\, h_{k-N+1}^{k-1}\right]}$$

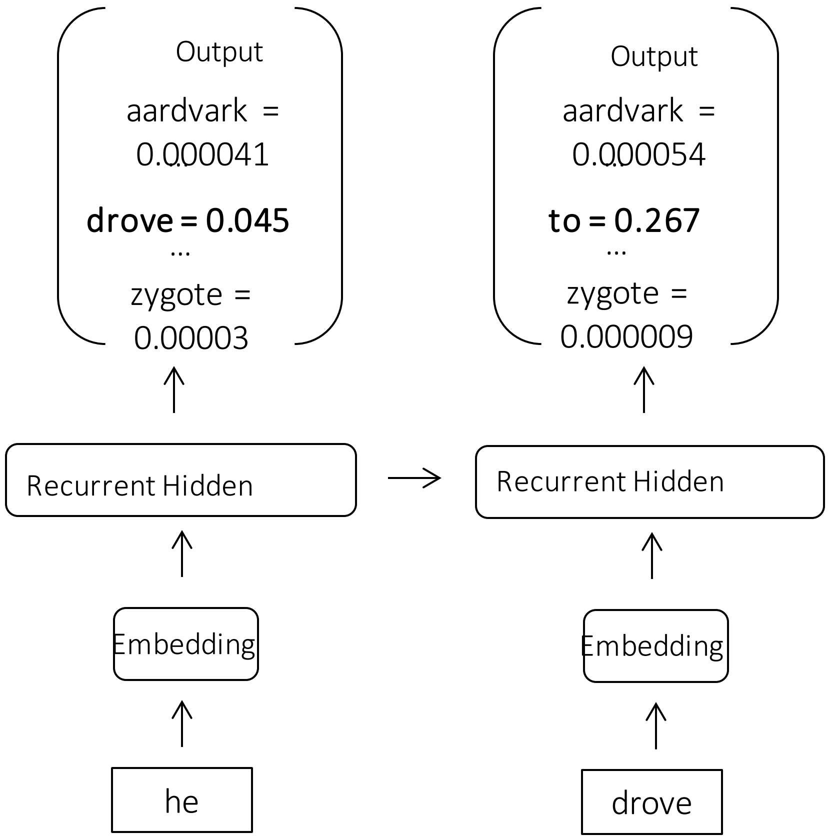
- Sparse (alleviated by back-off, but not entirely)
- Doesn't exploit word similarity
- Finite Context

Neural Network Language Models (NNLMs)

Feed-forward NNLM

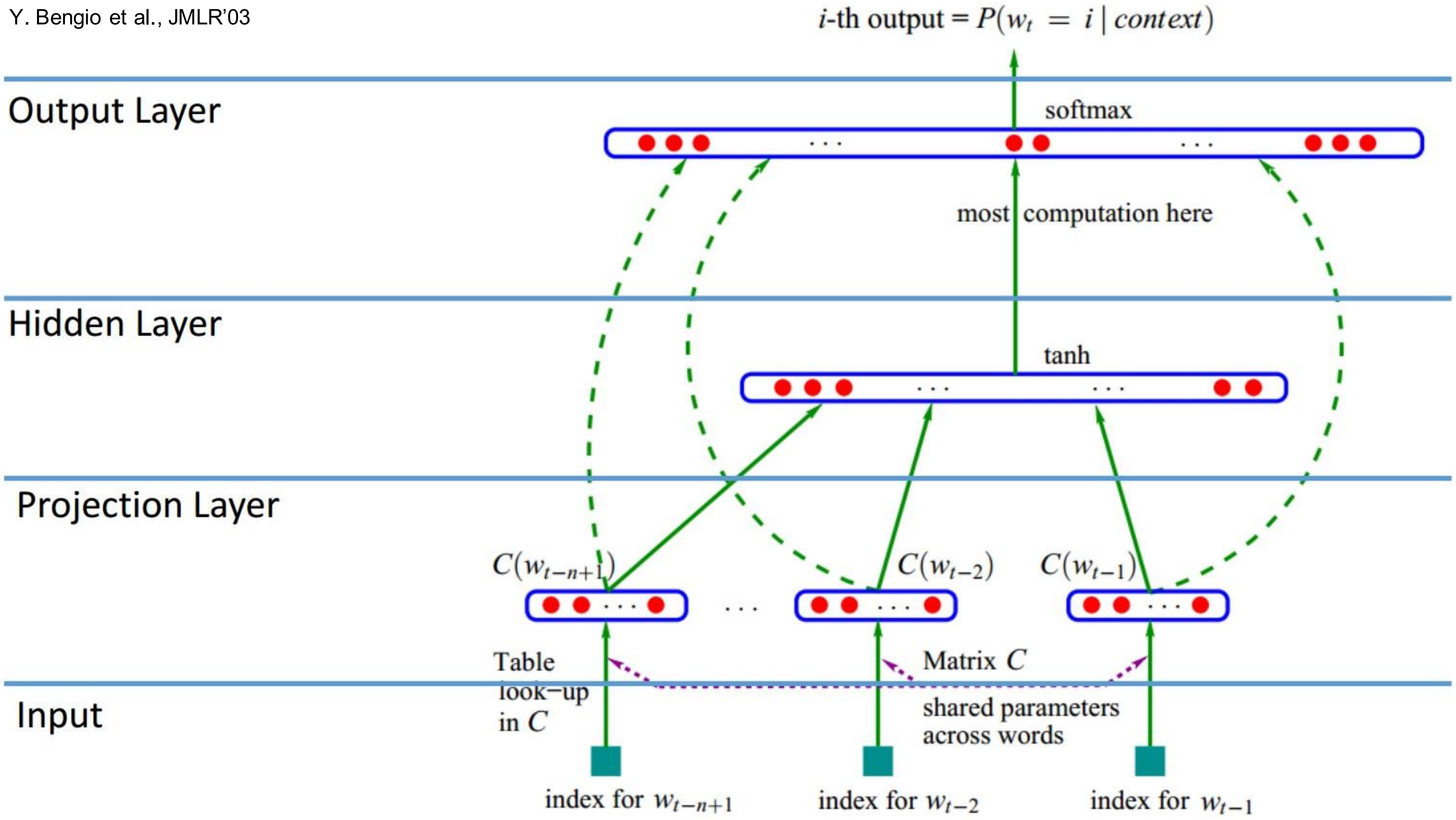


Recurrent NNLM



Neural Network Language Model

Y. Bengio et al., JMLR'03

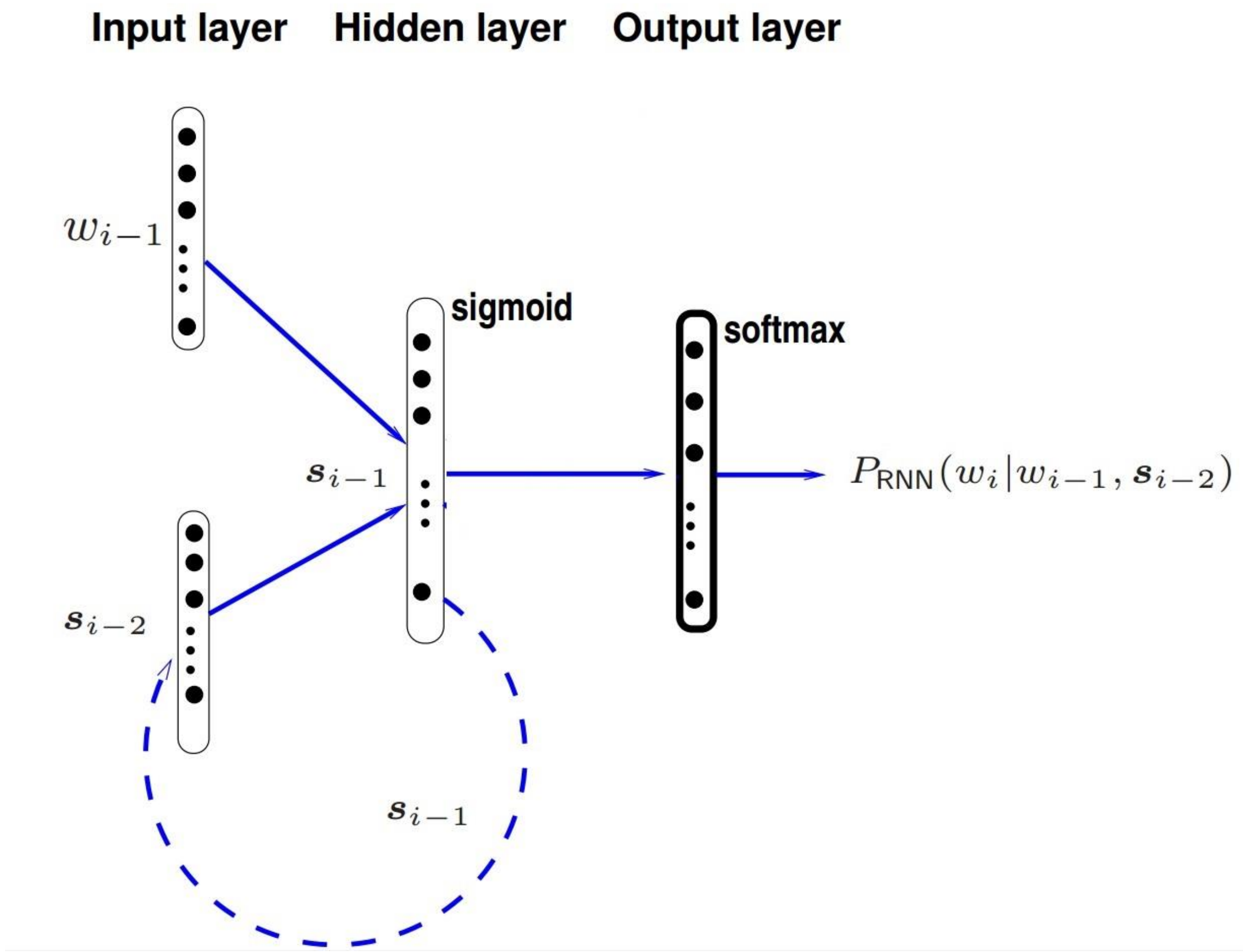


Limitation of Neural Network Language Model

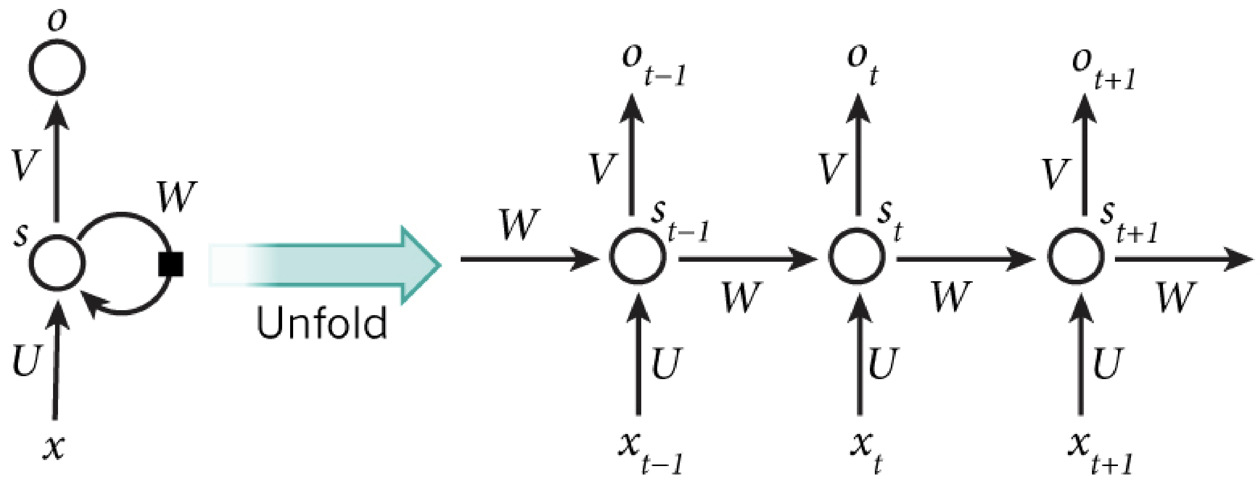
- Sparsity – Solved
- World Similarity – Solved
- Finite Context – Not
- Computational Complexity - Softmax

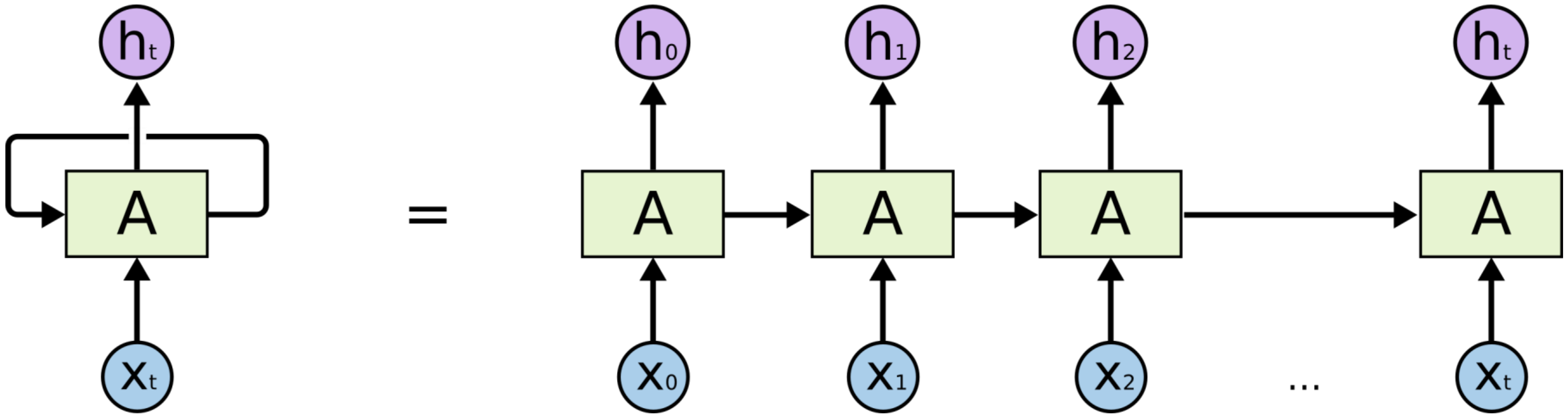
Recurrent Neural Network Language Model

[X. Liu, et al.]



Recurrent Neural Network

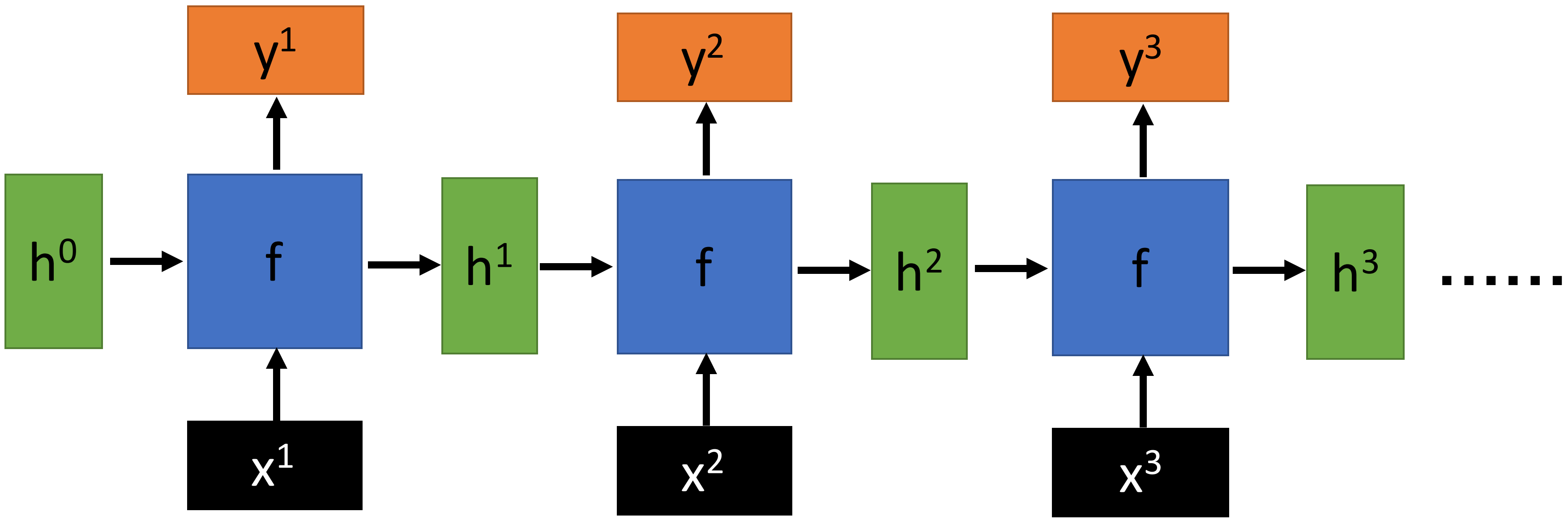




How does RNN reduce complexity?

- Given function f: h’,y=f(h,x)

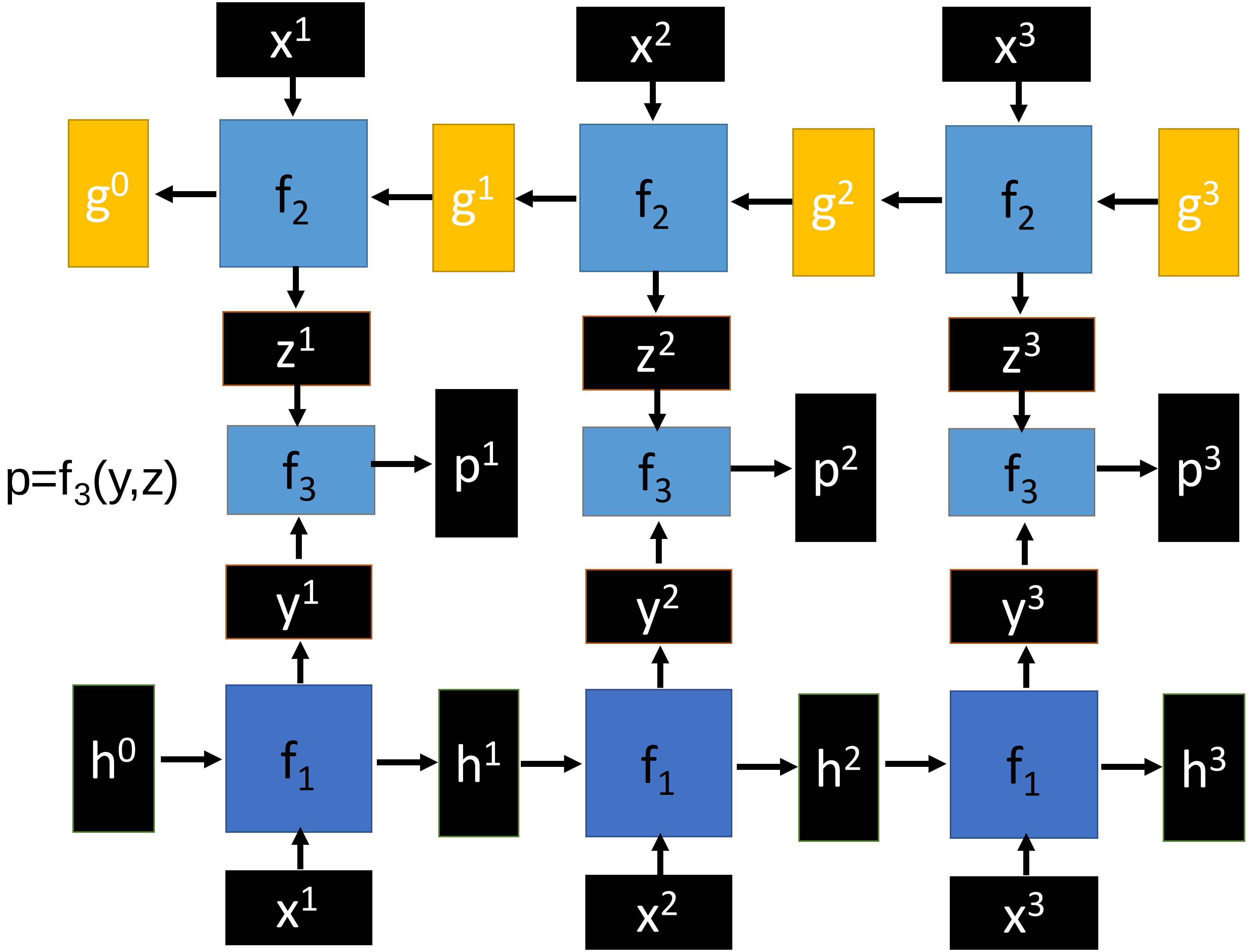
h and h’ are vectors with the same dimension



No matter how long the input/output sequence is, we only need one function f. If f’s are different, then it becomes a feedforward NN. This may be treated as another compression from fully connected network.

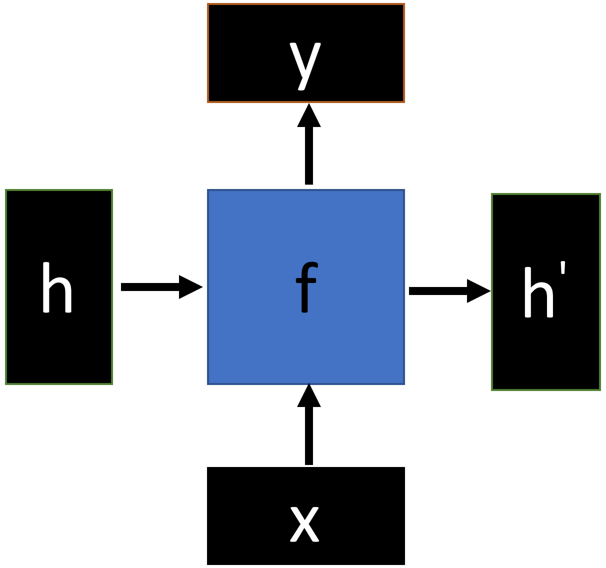
Bidirectional RNN

$y,h=f_1(x,h)$ $z,g = f_2(g,x)$



Naïve RNN

- Given function f: h’,y=f(h,x)



$$h' = \sigma(W^h h + W^i x)$$

$$y = \sigma(W^o h')$$

softmax

Note, y is computed from h’

We have ignored the bias

Problems with naive RNN

- When dealing with a time series, it tends to forget old information. When there is a distant relationship of unknown length, we wish to have a "memory" to it.
- Vanishing gradient problem.