(WORD2VEC) DISTRIBUTED REPRESENTATIONS OF WORDS & PHRASES AND THEIR COMPOSITIONALITY

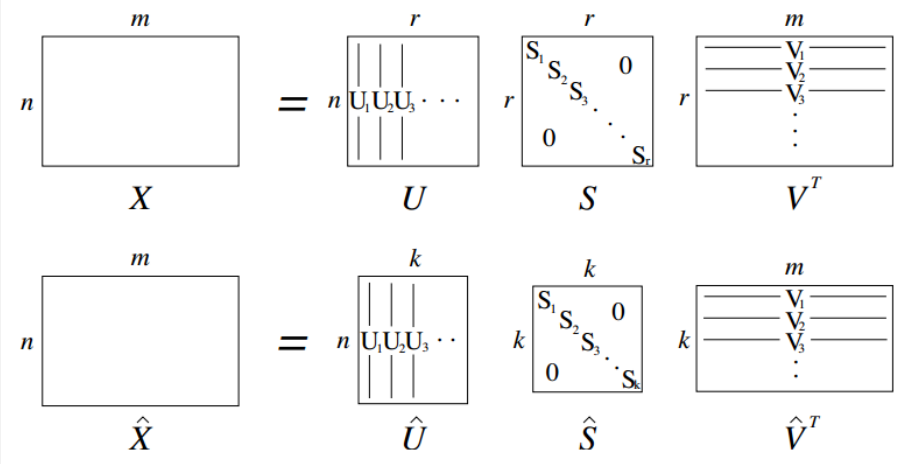
Muhammad Atif Qureshi

TO COMPARE PIECES OF TEXT

- We need effective representation of :
  - Words
  - Sentences
  - Text
- Approach 1: Use existing thesauri or ontologies like WordNet and Snomed CT (for medical). Drawbacks:
  - Manual
  - Not context specific
- Approach 2: Use co-occurrences for word similarity. Drawbacks:
  - Quadratic space needed
  - Relative position and order of words not considered

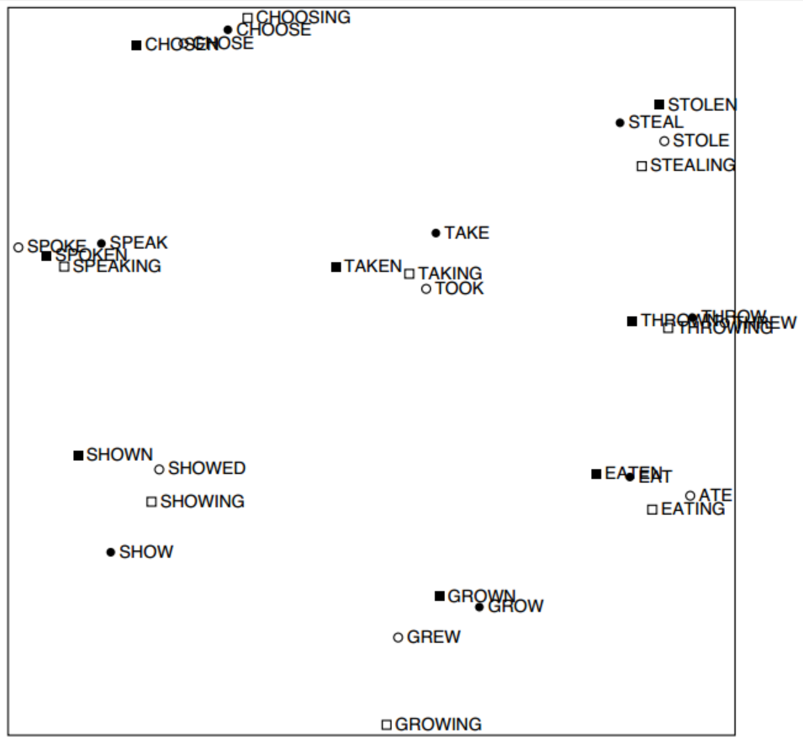
APPROACH 3: LOW DIMENSIONAL VECTORS

- Store only "important" information in fixed, low dimensional vector.
- Single Value Decomposition (SVD) on co-occurrence matrix
  - $\hat{X}$ is the best rank k approximation to X , in terms of least squares
  - Motel = [0.286, 0.792, -0.177, -0.107, 0.109, -0.542, 0.349, 0.271]
- m = n = size of vocabulary



APPROACH 3: LOW DIMENSIONAL 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 $n<m$)
- Hard to incorporate new words or documents
- Does not consider order of words

WORD EMBEDDING

- Definition
  - It is a technique in NLP that quantifies a concept (word or phrase) as a vector of real numbers.
- Simple application scenario
  - How similar are two words?
  - Similarity(vector(good), vector(best))
- Applications
  - Topic Modelling
  - Information Retrieval
  - Document Classification

WORD ANALOGIES

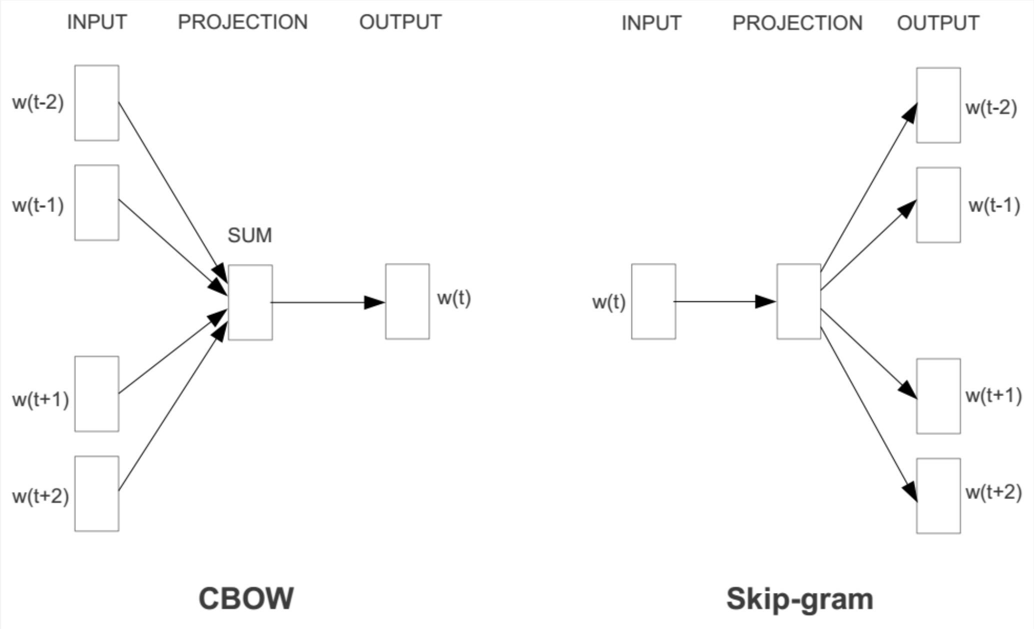
- Man is to Woman, King is to _____ ?
- London is to England, Dublin is to _____ ?
- Using vectors, we can say
  - King – Man + Woman → Queen
  - Dublin – London + England → Ireland

WORD2VEC APPROACH TO REPRESENT THE MEANING OF WORD

- Represent each word with a low-dimensional vector
- Word similarity = vector similarity
- Key idea: Predict surrounding words of every word
- 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

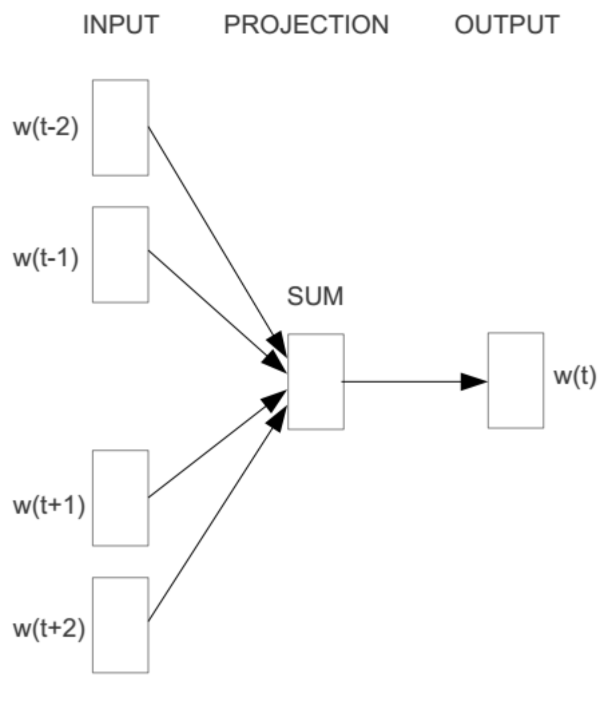
- E.g. "The cat sat on floor"
  - Window size = 2

the

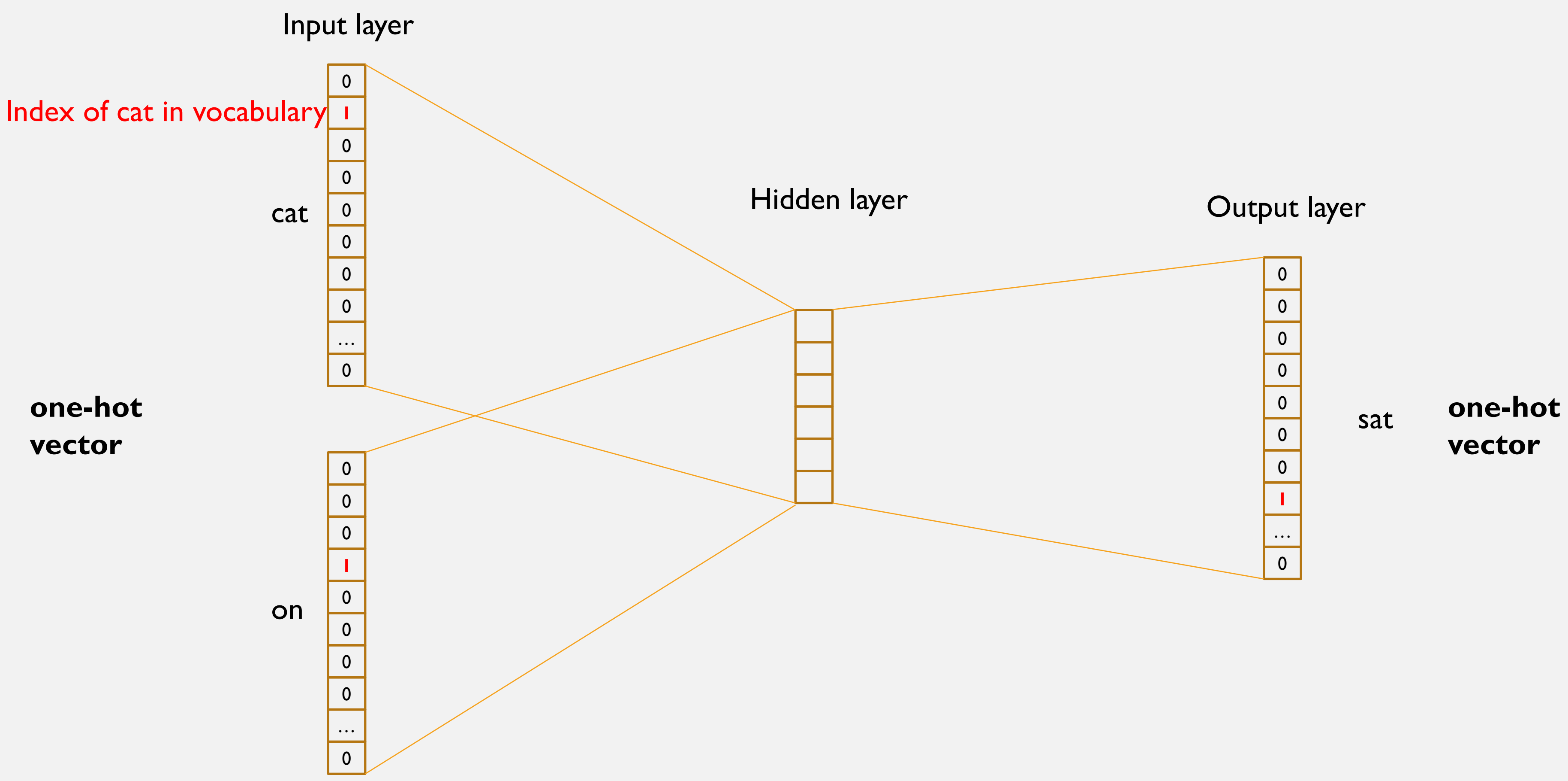
cat

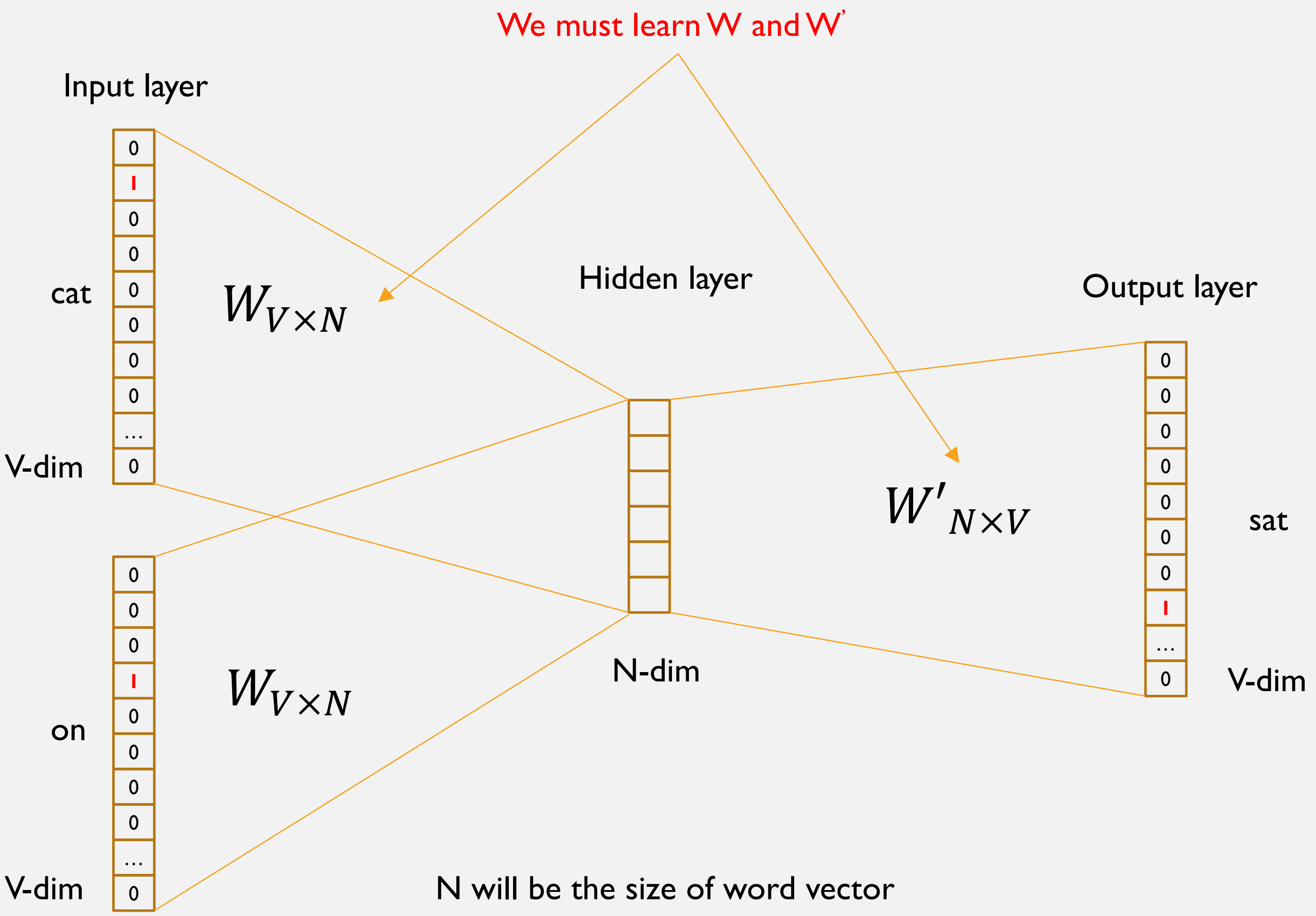
on

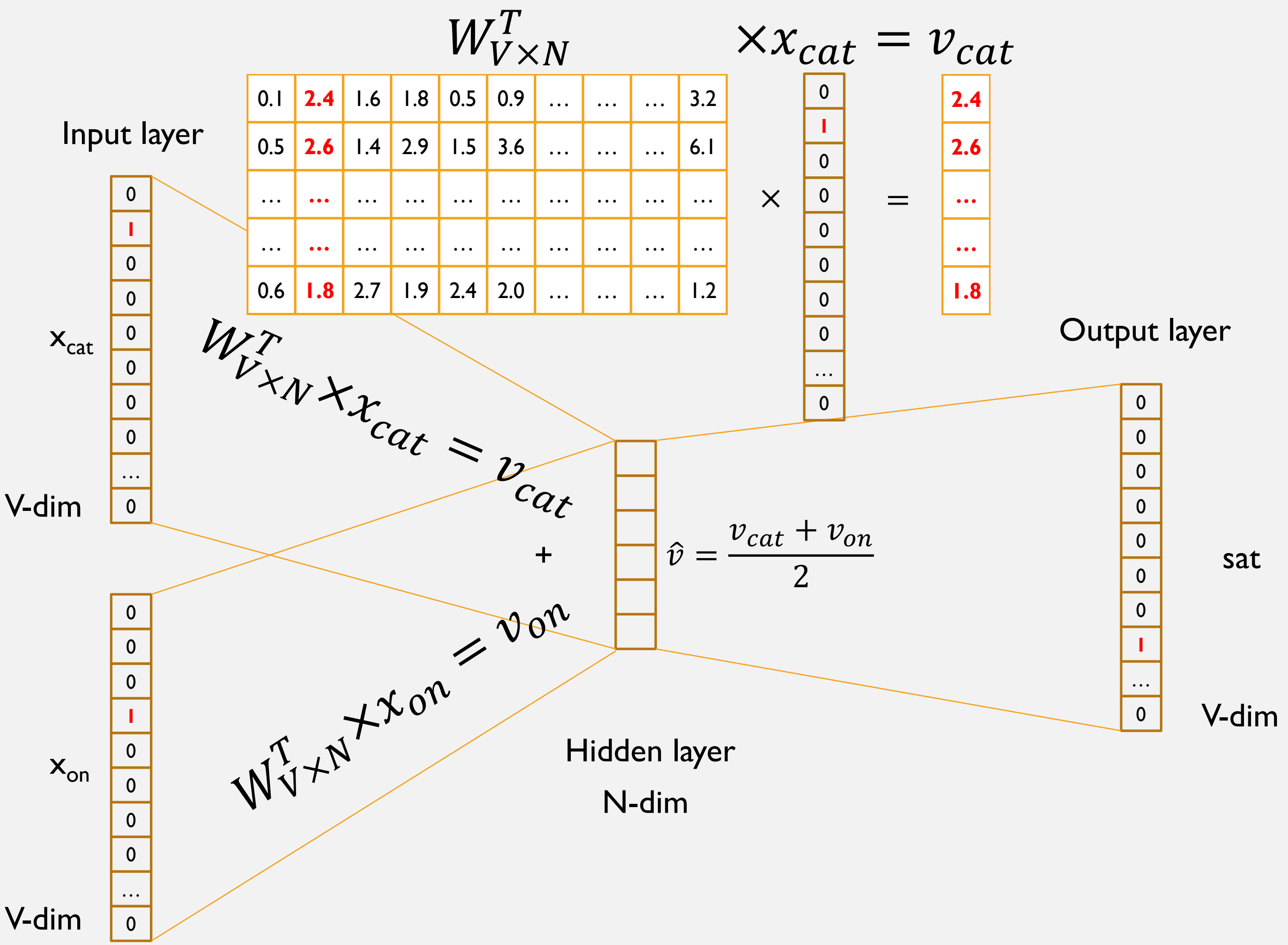
floor

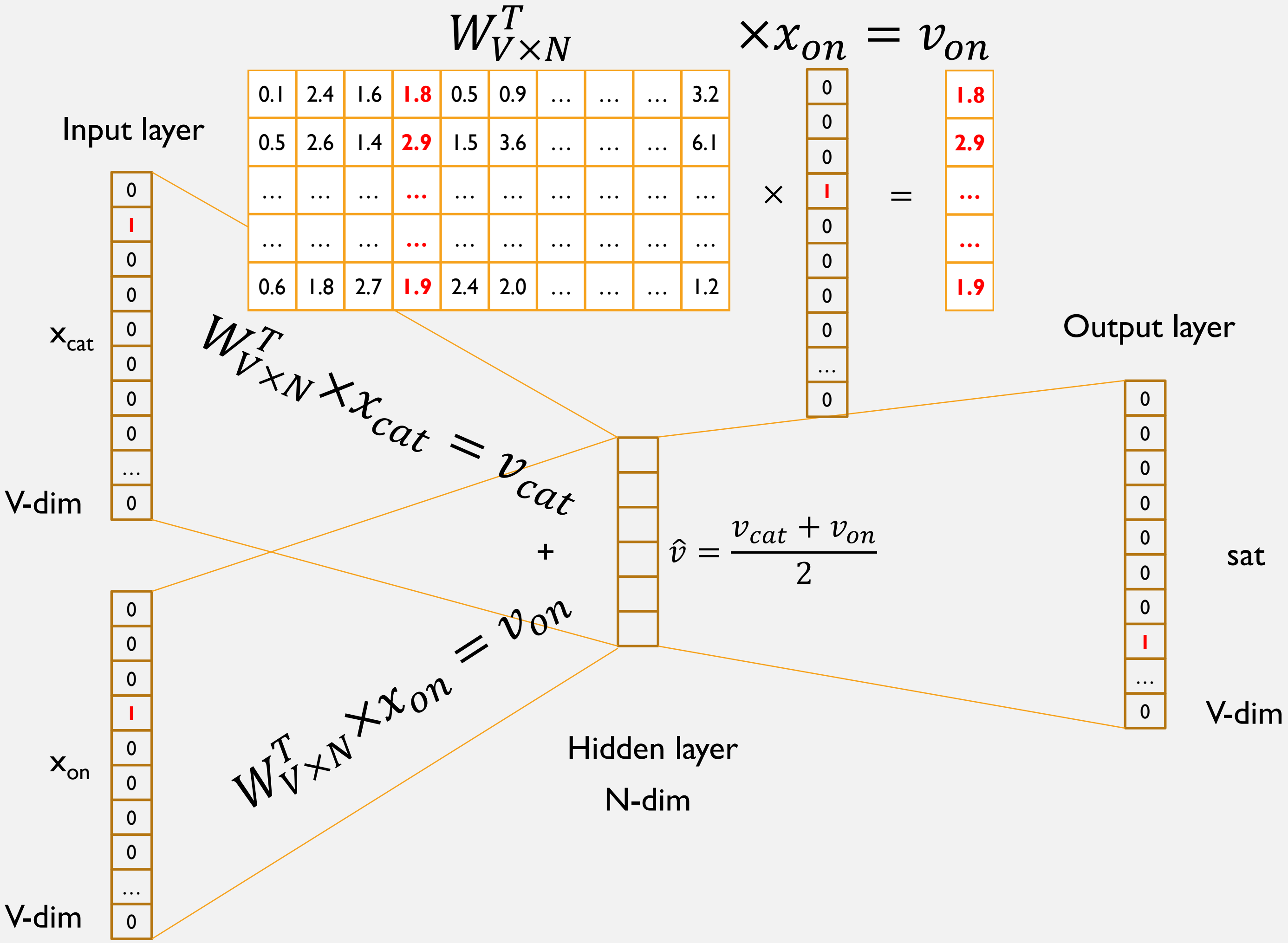


sat

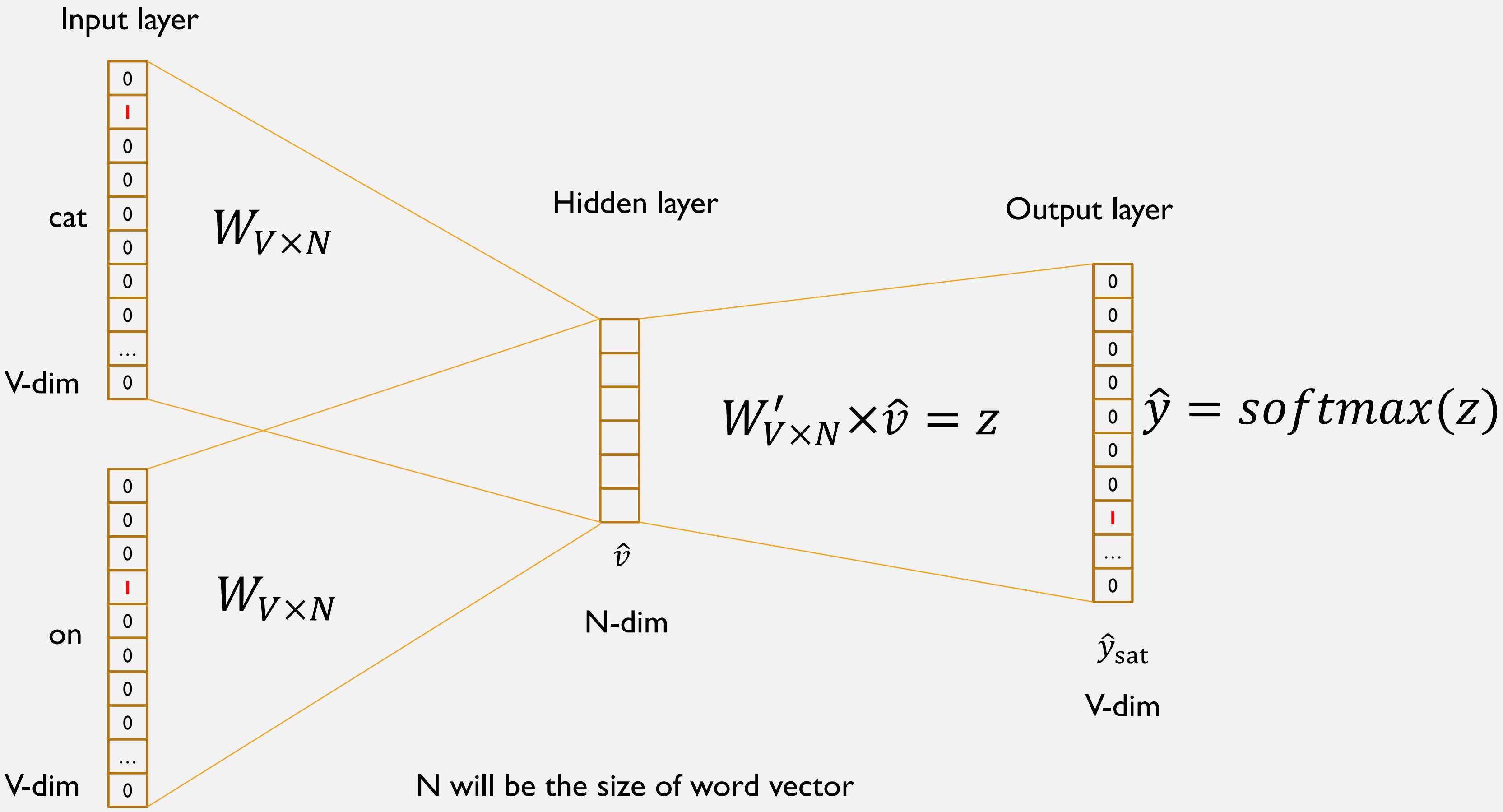

Input layer
Index of cat in vocabulary
cat
on
one-hot
vector
Hidden layer
Output layer
sat
one-hot
vector


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


$W_{V \times N}^{T}$
$\times x_{cat} = v_{cat}$
Input layer
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
×
0 1 0 0 0 0 0 0 … 0
=
2.4 2.6 … … 1.8
x_{cat}
0 1 0 0 0 0 0 0 … 0
V-dim
$W_{V \times N}^{T} \times x_{cat} = v_{cat}$
+
$W_{V \times N}^{T} \times x_{on} = v_{on}$
x_{on}
0 0 0 1 0 0 0 0 … 0
V-dim
$\hat{v} = \frac{v_{cat} + v_{on}}{2}$
Hidden layer
N-dim
Output layer
0 0 0 0 0 0 0 1 … 0
sat
V-dim

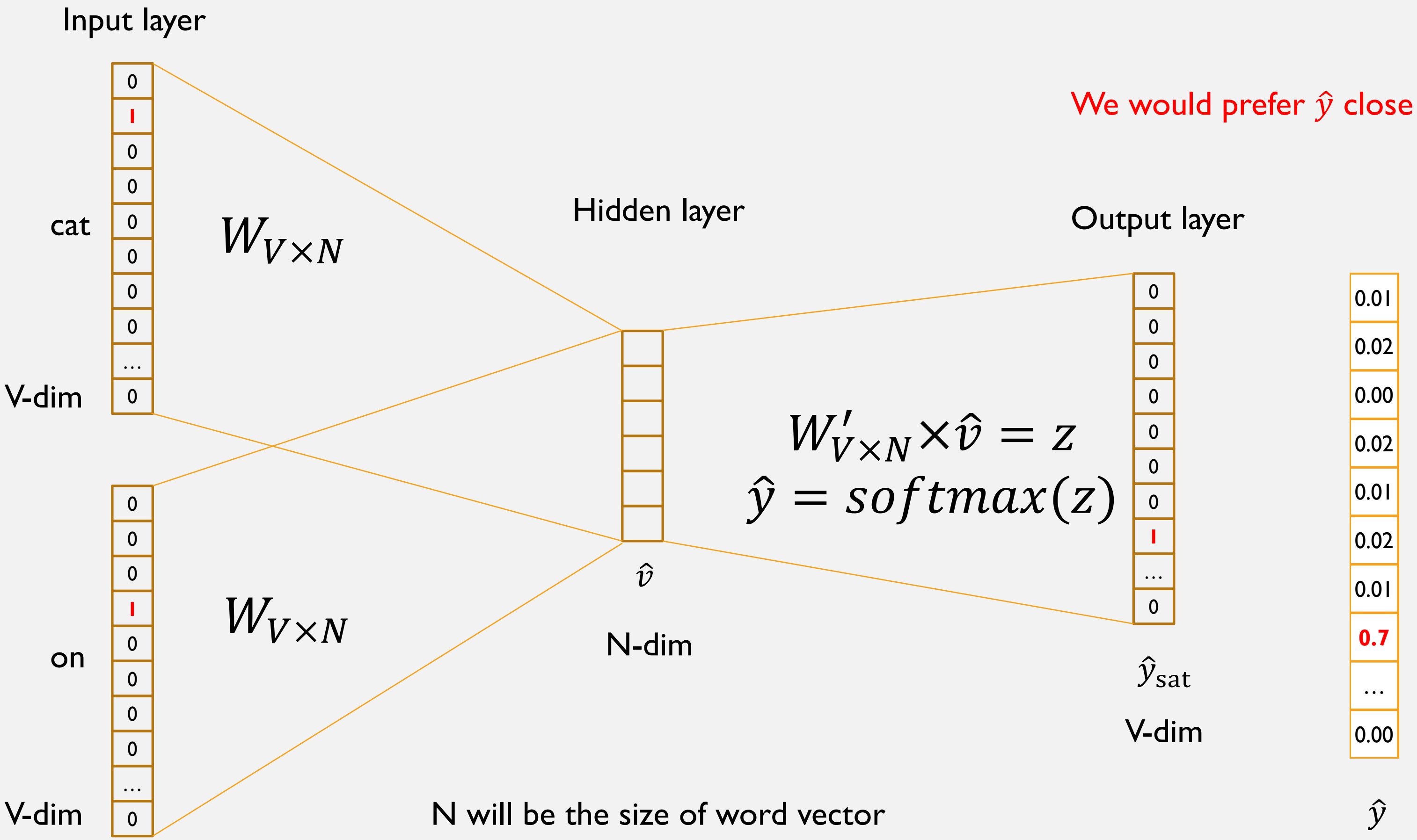


$W^T_{V\times N}$
$\times x_{on} = v_{on}$
0.1 2.4 1.6 1.8 0.5 0.9 … … … 3.2
0.5 2.6 1.4 2.9 1.5 3.6 … … … 6.1
0.6 1.8 2.7 1.9 2.4 2.0 … … … 1.2
1.8
2.9
1.9
Input layer
x_{cat}
V-dim
$W^T_{V\times N} \times x_{cat} = v_{cat}$
+
$W^T_{V\times N} \times x_{on} = v_{on}$
x_{on}
V-dim
$\hat{v} = \frac{v_{cat}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

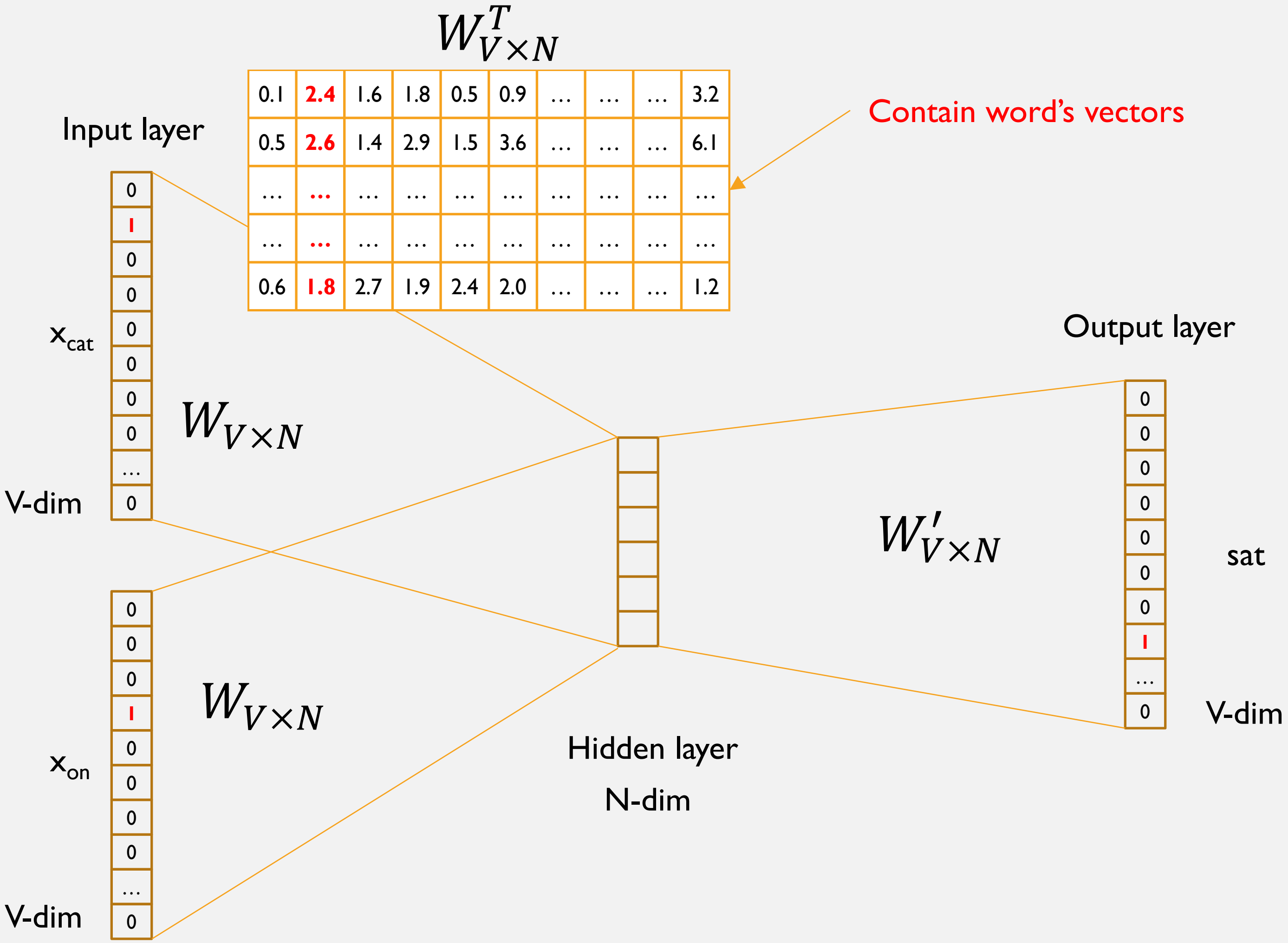


Input layer
cat
V-dim
on
V-dim
$W_{V\times N}$
$W_{V\times N}$
Hidden layer
$\hat{v}$
N-dim
$W'_{V\times N}\times\hat{v} = z$
Output layer
$\hat{y} = softmax(z)$
$\hat{y}_{\text{sat}}$
V-dim
N will be the size of word vector

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



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

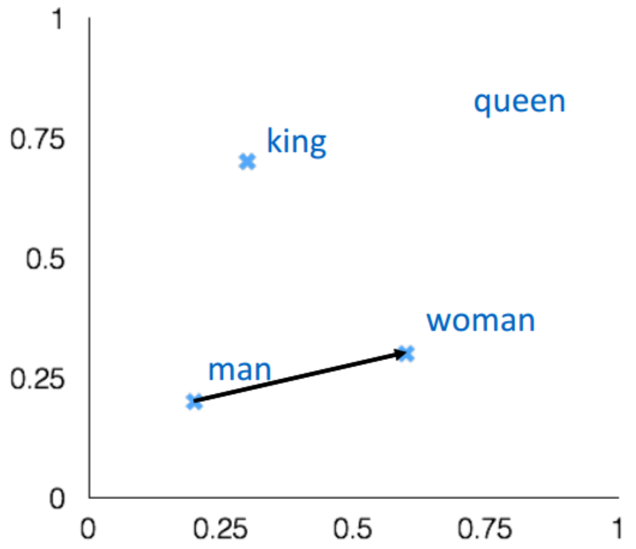
Word Analogies

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

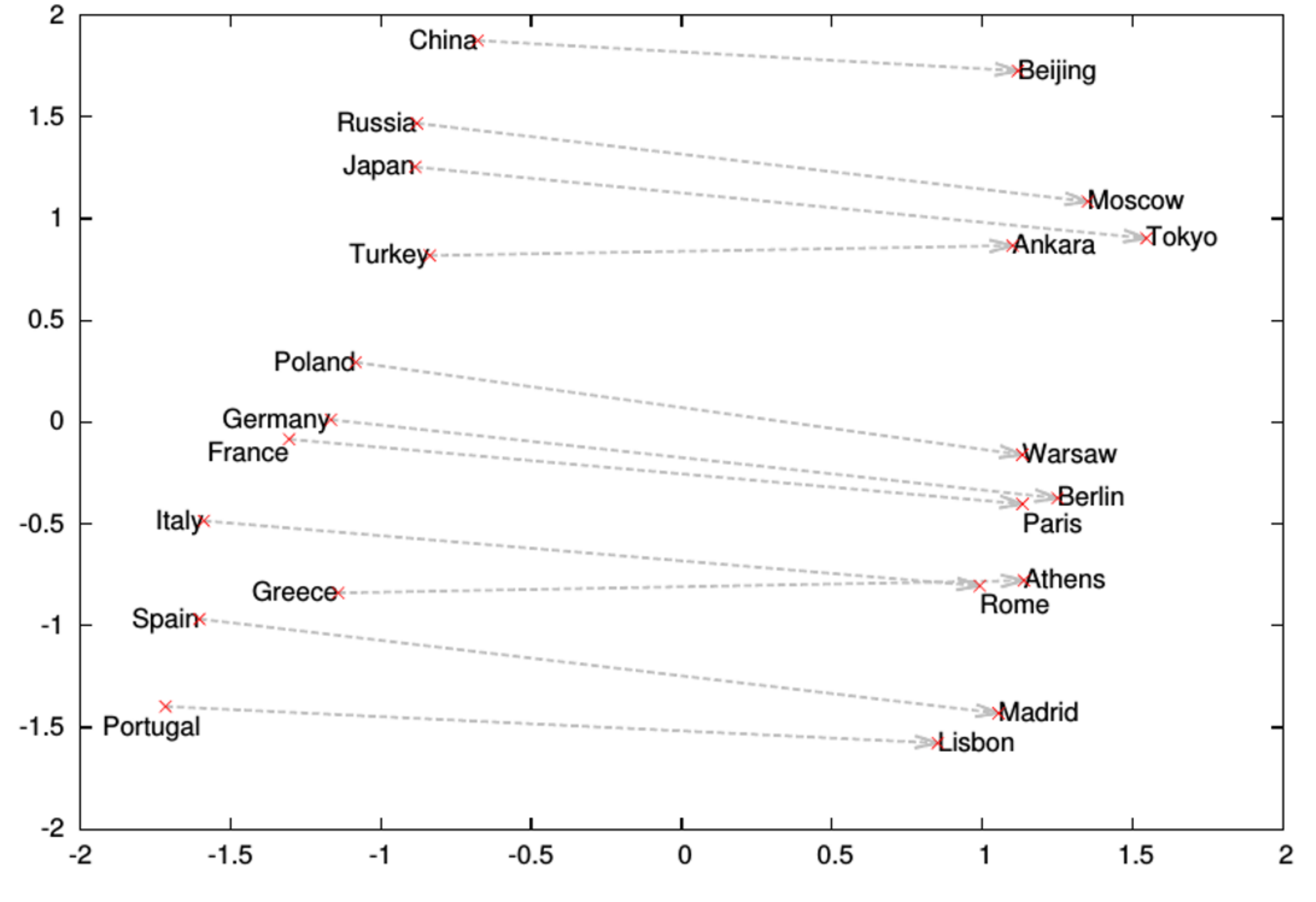
a:b :: c:? → $d = \arg\max_x \frac{(w_b - w_a + w_c)^T w_x}{||w_b - w_a + w_c||}$

man:woman :: king:?

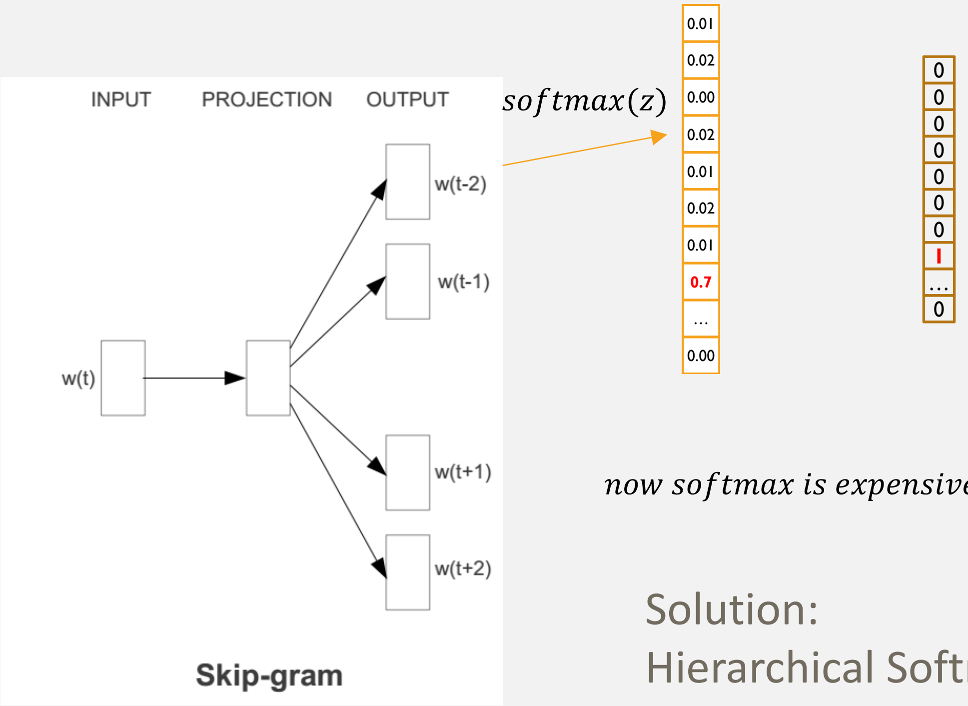
+	king	[0.30 0.70]
-	man	[0.20 0.20]
+	woman	[0.60 0.30]
	queen	[0.70 0.80]



WORD ANALOGIES



SKIP-GRAM



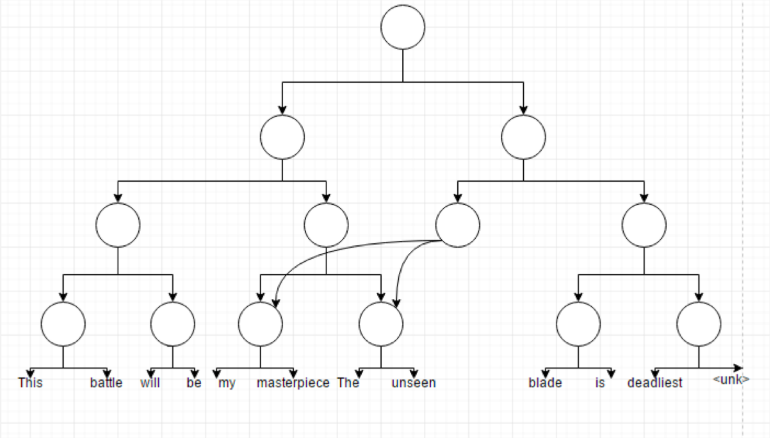
now softmax is expensive on multiple output layers

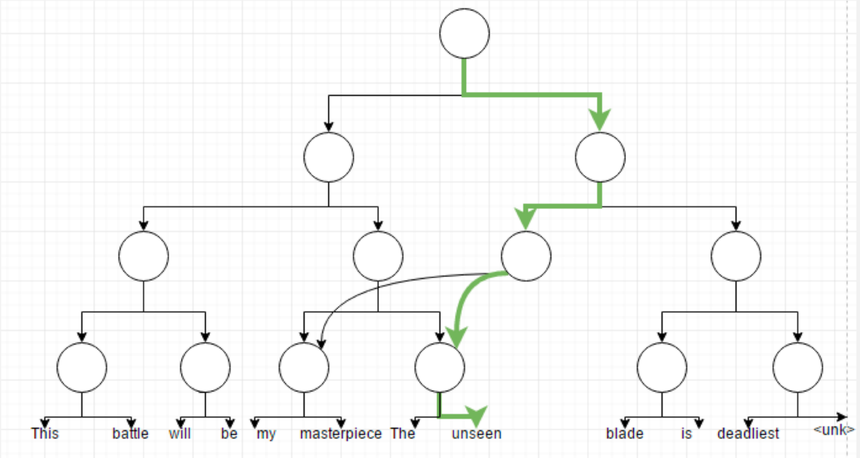
Solution:
Hierarchical Softmax and Negative Sampling

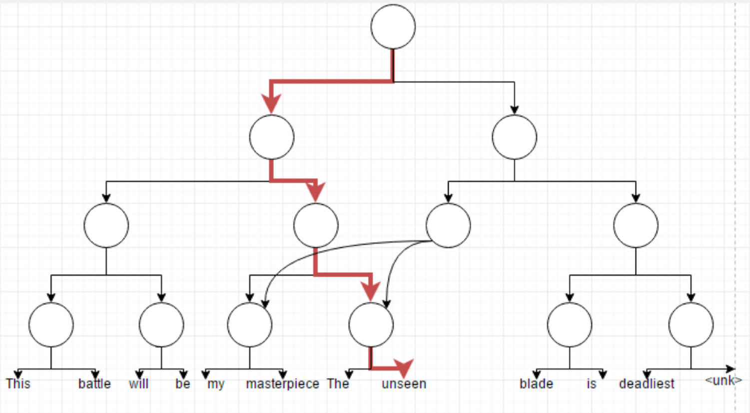
HIERARCHICAL SOFTMAX & NEGATIVE SAMPLING

- These are optimization strategies to speed up the process of training
- Subsampling faster training time and as well as improves accuracy.

HIERARCHICAL SOFTMAX







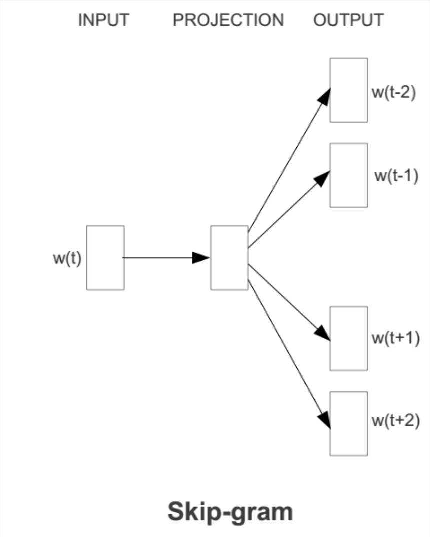
P(right) * P(left) * P(right) * P(right).

or **P(left) * P(right) * P(right) * P(right)**

We then use logistic regression and apply the Sigmoid function to get the probability

reduced from **O(N)** to **O(log(N))**

NEGATIVE SAMPLING



$softmax(z)$

0.01	0
0.02	0
0.00	0
0.02	0
0.01	0
0.02	0
0.01	0
0.7	1
…	…
0.00	0

Normally, after getting the error, we update the whole weights of the neural network

Idea: we don't have to do this

We know that we have all the corpus words in the training items, so, the words weights will be certainly updated during the training phase.

So, we can instead update some of the words weights for each training example

These words would contain the positive word (the actual word that shall be predicted) and other words that are considered to be negative samples (incorrect words that shall not predicted).

NEGATIVE SAMPLING

The question now is, how to select these negative samples?

Traditionally, choosing these negative samples depends on the words frequency in the given corpus. The higher frequency of the word means the higher chance of choosing it as a negative sample.

The following formula is used to determine the probability of selecting the word as a negative sample.

$$P(w_i) = \frac{frequency(w_i)^c}{\Sigma frequncey(w_j)^c})$$

Where **c** is a constant that is selected by the model creator. After applying that, we choose the maximum **N** words, where **N** is the number of the negative samples.

SUBSAMPLING

- Heuristically chosen the formula for subsampling to counter the imbalance between the rare and frequent words.
- Each word in the training is discarded with probability computed by:

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

Where $f(w_i)$ is the frequency of word w_i and t is a chosen threshold, typically around 10^{-5}.

RESULTS

Method	Time [min]	Syntactic [%]	Semantic [%]	Total accuracy [%]
NEG-5	38	63	54	59
NEG-15	97	63	58	**61**
HS-Huffman	41	53	40	47
NCE-5	38	60	45	53
The following results use 10^{-5} subsampling				
NEG-5	14	61	58	60
NEG-15	36	61	61	**61**
HS-Huffman	21	52	59	55

Table 1: Accuracy of various Skip-gram 300-dimensional models on the analogical reasoning task as defined in [8]. NEG-k stands for Negative Sampling with k negative samples for each positive sample; NCE stands for Noise Contrastive Estimation and HS-Huffman stands for the Hierarchical Softmax with the frequency-based Huffman codes.

LEARNING PHRASE

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

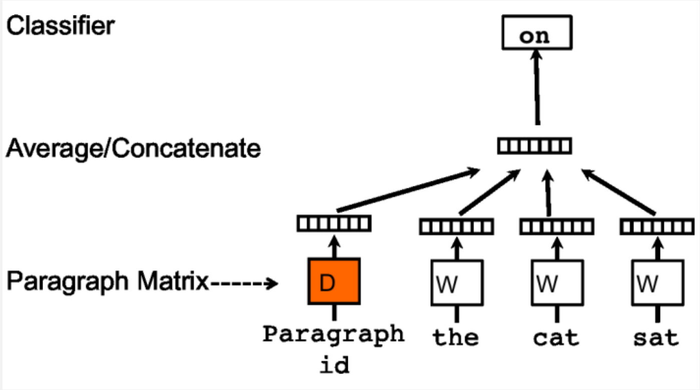
The δ is used as a discounting coefficient and prevents too many phrases consisting of very infrequent words to be formed. The bigrams with score above the chosen threshold are then used as phrases. Typically, we run 2-4 passes over the training data with decreasing threshold value, allowing longer phrases that consists of several words to be formed.

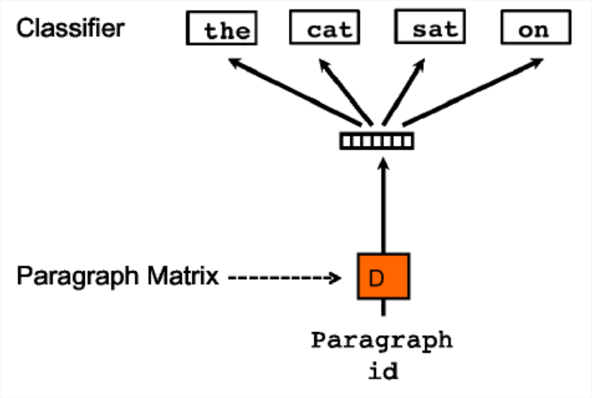
Method	Dimensionality	No subsampling [%]	10^{-5} subsampling [%]
NEG-5	300	24	27
NEG-15	300	27	42
HS-Huffman	300	19	**47**

Table 3: Accuracies of the Skip-gram models on the phrase analogy dataset. The models were trained on approximately one billion words from the news dataset.

REPRESENT THE MEANING OF **SENTENCE/TEXT**

- Paragraph vector (2014, Quoc Le, Mikolov)
  - Extend word2vec to text level
  - Also two models: add paragraph vector as the input





APPLICATIONS

- Search, e.g., query expansion
- Sentiment analysis
- Classification
- Clustering

RESOURCES

- Stanford CS224d: Deep Learning for NLP
  - http://cs224d.stanford.edu/index.html
  - The best
- "word2vec Parameter Learning Explained", Xin Rong
  - https://ronxin.github.io/wevi/
- GOOGLE vectors: https://code.google.com/archive/p/word2vec/
- Word2Vec Tutorial - The Skip-Gram Model
  - http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/
- https://www.youtube.com/watch?v=thLzt3D-A10
- Qureshi, M.A. and Greene, D., 2018. EVE: explainable vector based embedding technique using Wikipedia. *Journal of Intelligent Information Systems, pp 1-29*. https://doi.org/10.1007/s10844-018-0511-x