Vector Embedding of Words

- A word is represented as a vector.
- Word embeddings depend on a notion of ***word similarity***.
  - Similarity is computed using cosine.
- A very useful definition is paradigmatic similarity:
  - ***Similar words*** occur in ***similar contexts.*** They are ***exchangeable.***
  - Yesterday { POTUS / The President / Trump } called a press conference.
  - "POTUS: President of the United States."

Vector Embedding of Words

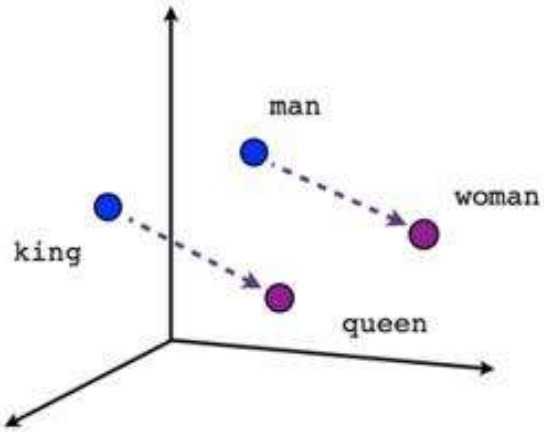
Traditional Method - Bag of Words Model

- Either uses one hot encoding.
  - Each word in the vocabulary is represented by one bit position in a HUGE vector.
  - For example, if we have a vocabulary of 10000 words, and "Hello" is the 4th word in the dictionary, it would be represented by: 0 0 0 1 0 0 . . . . . 0 0 0
- Or uses document representation.
  - Each word in the vocabulary is represented by its presence in documents.
  - For example, if we have a corpus of 1M documents, and "Hello" is in 1th, 3th and 5th documents ***only***, it would be represented by: 1 0 1 0 1 0 . . . . . 0 0 0
- Context information is not utilized.

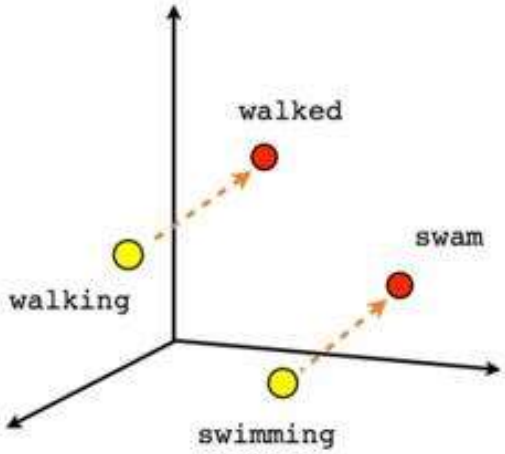
Word Embeddings

- Stores each word in as a point in space, where it is represented by a dense 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.

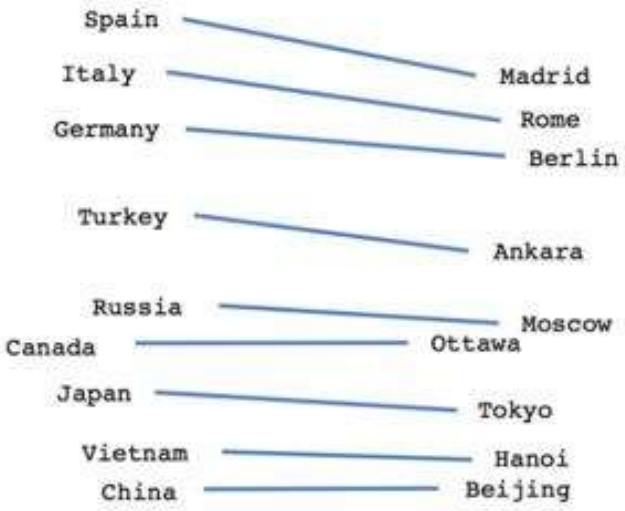
Example



Male-Female



Verb tense



Country-Capital

- vector[Queen] ≈ vector[King] - vector[Man] + vector[Woman]
- vector[Paris] ≈ vector[France] - vector[Italy] + vector[Rome]
  - This can be interpreted as "France is to Paris as Italy is to Rome".

Applications of Word Vectors

- Word Similarity
- Machine Translation
- Part-of-Speech and Named Entity Recognition
- Relation Extraction
- Sentiment Analysis
- Co-reference Resolution
  - Chaining entity mentions across multiple documents - can we find and unify the multiple contexts in which mentions occurs?
- 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.
- Semantic Analysis of Documents
  - Build word distributions for various topics, etc.

Vector Embedding of Words

- Three main methods described :
  - Word2Vec (2013)
    - Prediction-based model.
    - Consider occurrences of terms at context level.
  - GloVe (2014)
    - Count-based model.
    - Consider occurrences of terms at context level.
  - ELMo (2018)
    - Language model-based.
    - A different embedding for each word for each task.

Word2Vec

Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781 (2013).

- The word2vec objective function causes the words that occur in similar contexts to have similar embeddings.

Example: The **kid** said he would grow up to be superman.
The **child** said he would grow up to be superman.

The words kid and child will have similar word vectors due to a similar context.

word2Vec: Local contexts

- Instead of entire documents, ***Word2Vec*** uses words ***k*** positions away from each center word.
  - These words are called **context words**.
- Example for ***k=3***: (skip ngram)
  - "It was a bright cold day in April, and the clocks were striking".
  - Center word: red (also called focus word).
  - Context words: blue (also called target words).
- Word2Vec considers all words as center words, and all their context words.

Word2Vec: Data generation (window size = 2)

- Example: d1 = "king brave man" , d2 = "queen beautiful women"

word	Word one hot encoding	neighbor	Neighbor one hot encoding
king	[1,0,0,0,0,0]	brave	[0,1,0,0,0,0]
king	[1,0,0,0,0,0]	man	[0,0,1,0,0,0]
brave	[0,1,0,0,0,0]	king	[1,0,0,0,0,0]
brave	[0,1,0,0,0,0]	man	[0,0,1,0,0,0]
man	[0,0,1,0,0,0]	king	[1,0,0,0,0,0]
man	[0,0,1,0,0,0]	brave	[0,1,0,0,0,0]
queen	[0,0,0,1,0,0]	beautiful	[0,0,0,0,1,0]
queen	[0,0,0,1,0,0]	women	[0,0,0,0,0,1]
beautiful	[0,0,0,0,1,0]	queen	[0,0,0,1,0,0]
beautiful	[0,0,0,0,1,0]	women	[0,0,0,0,0,1]
woman	[0,0,0,0,0,1]	queen	[0,0,0,1,0,0]
woman	[0,0,0,0,0,1]	beautiful	[0,0,0,0,1,0]

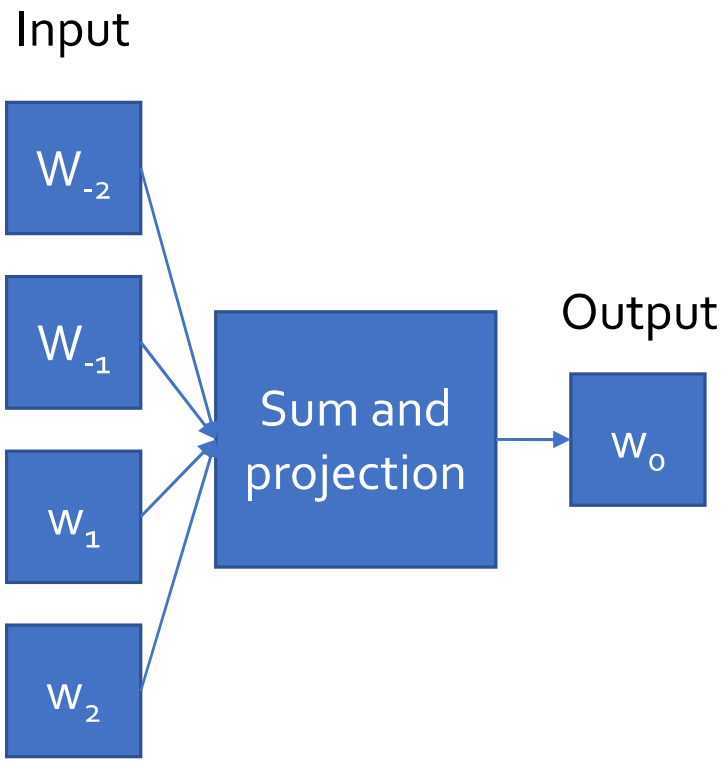
Word2Vec: Data generation (window size = 2)

- Example: d1 = "king brave man" , d2 = "queen beautiful women"

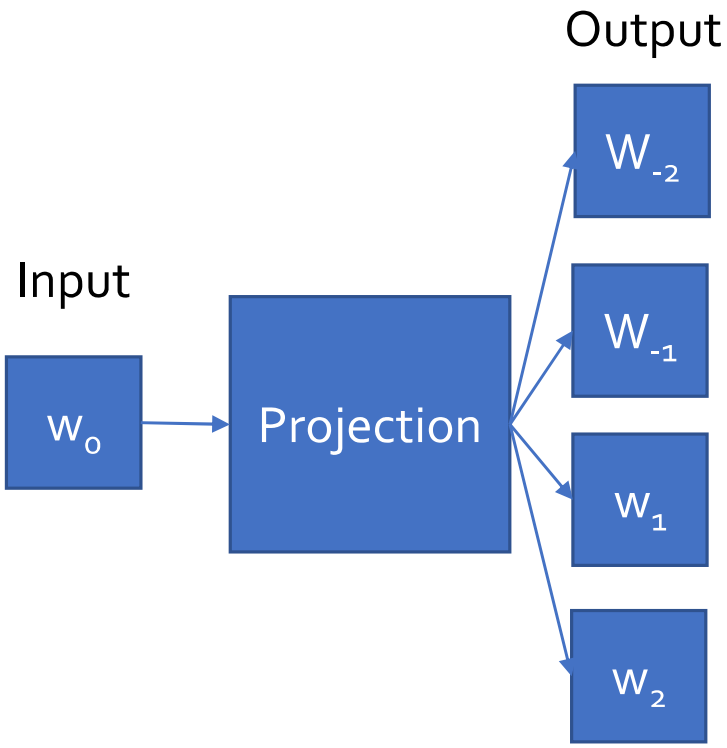
word	Word one hot encoding	neighbor	Neighbor one hot encoding
king	[1,0,0,0,0,0]	brave	[0,1,1,0,0,0]
		man	
brave	[0,1,0,0,0,0]	king	[1,0,1,0,0,0]
		man	
man	[0,0,1,0,0,0]	king	[1,1,0,0,0,0]
		brave	
queen	[0,0,0,1,0,0]	beautiful	[0,0,0,0,1,1]
		women	
beautiful	[0,0,0,0,1,0]	queen	[0,0,0,1,0,1]
		women	
woman	[0,0,0,0,0,1]	queen	[0,0,0,1,1,0]
		beautiful	

Word2Vec: main context representation models

Continuous Bag of Words (CBOW)



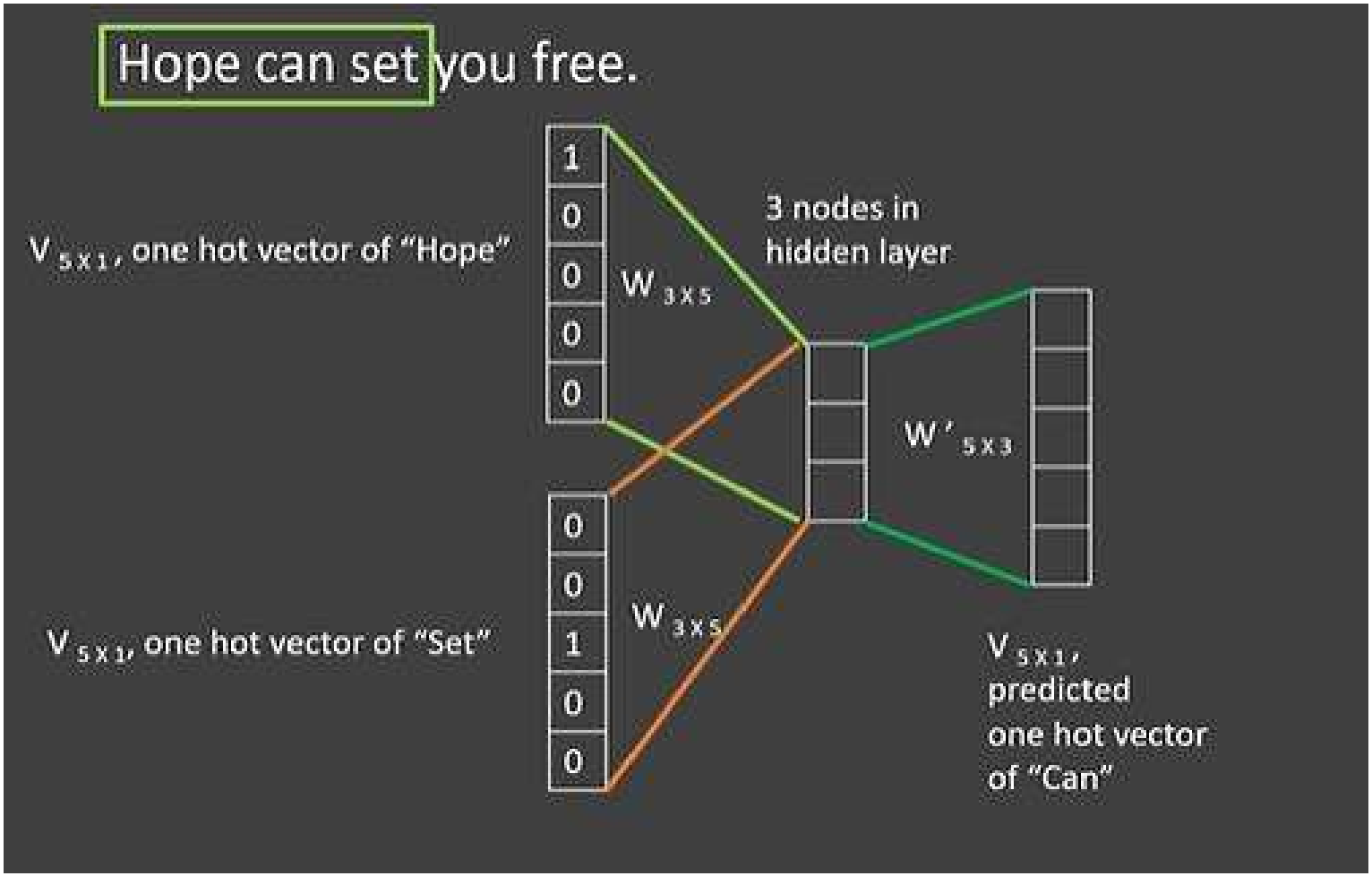
Skip-Ngram



- **Word2Vec is a predictive model.**
- **Will focus on Skip-Ngram model**

http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

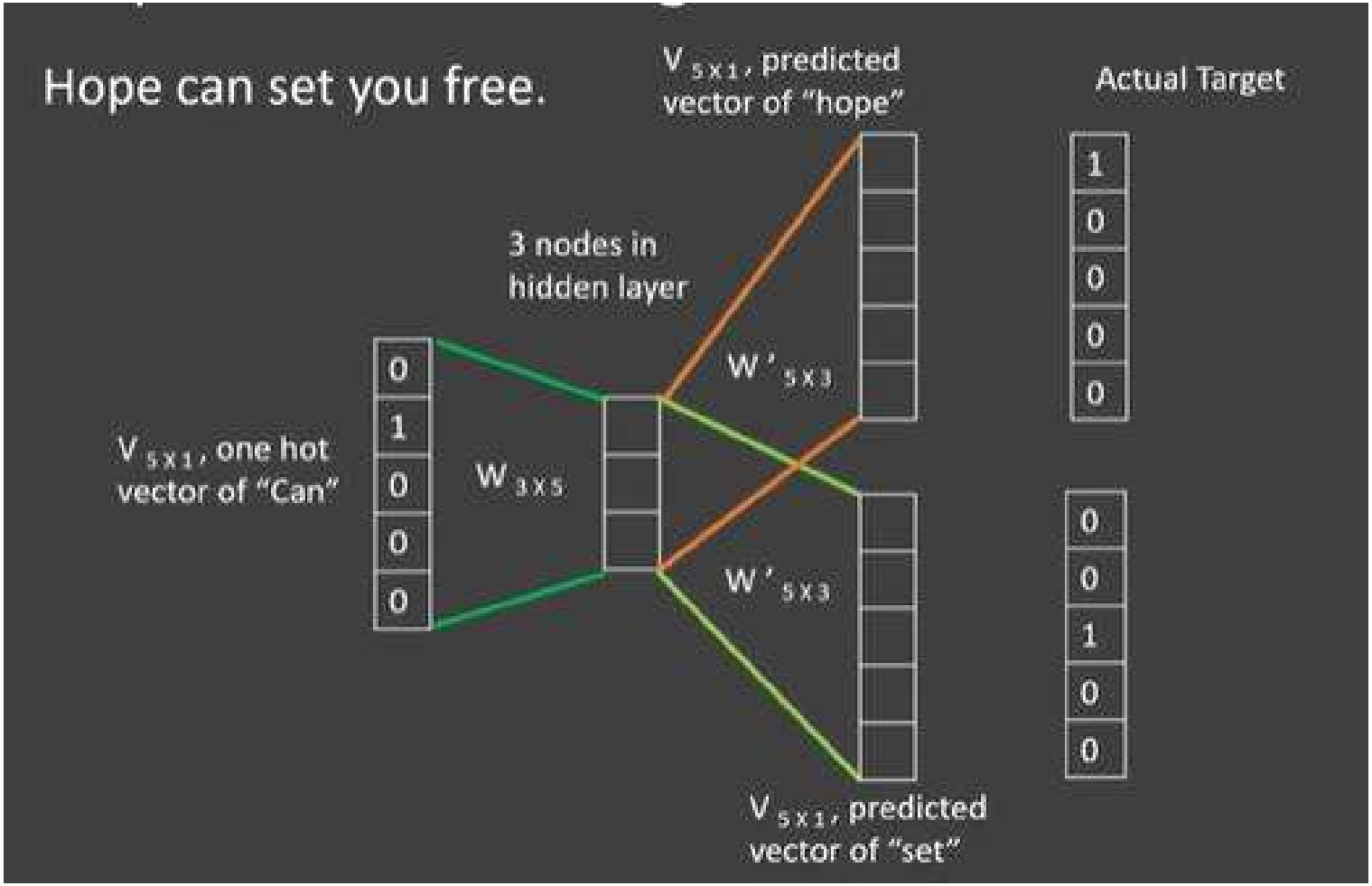
CBOW Working



- Word embeddings are created using a neural network with one input layer, one hidden layer and one output layer.

- The hidden layer dimension is the size of our word embedding.
- The output layers activation function is **softmax.**
- The activation function of the hidden layer is **linear.**
- Each input node will have weights connecting it to the hidden layer. These updated weights are the word embeddings!

Skip NGram



We will now have 2 1xV error vectors and will perform an element-wise sum to get a 1xV vector. The weights of the hidden layer will be updated based on this cumulative 1xV error vector.

Getting word embeddings

Weights after training

W_{3X5}

w00	w01	w02	w03	w04
w10	w11	w12	w13	w14
w20	w21	w22	w23	w24

One Hot vector of words V_{5X1}

Hope	can	set	you	free
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

Word Vector for hope = W_{3X5} X V_{5X1}

w00	w01	w02	w03	w04
w10	w11	w12	w13	w14
w20	w21	w22	w23	w24

X

1
0
0
0
0

=

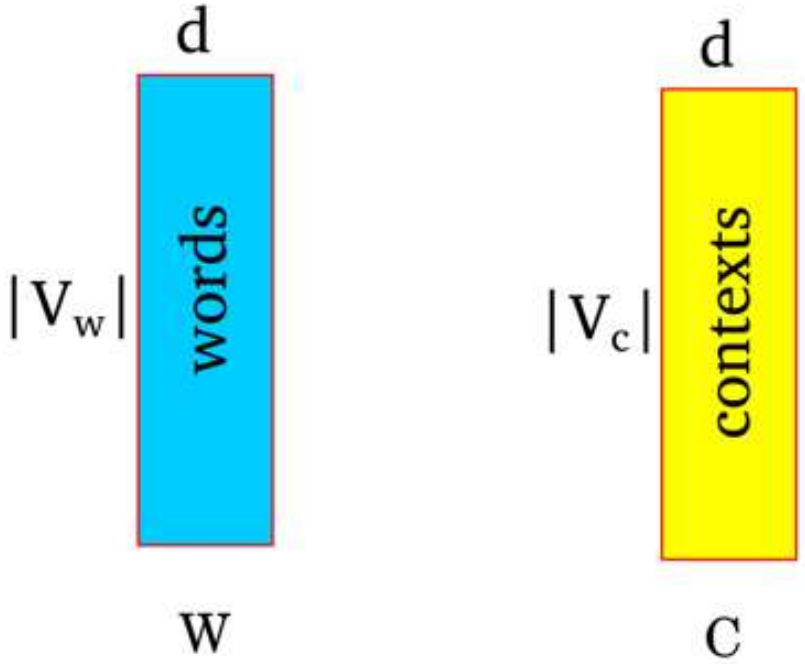
V_{3X1}

w00
w10
w20

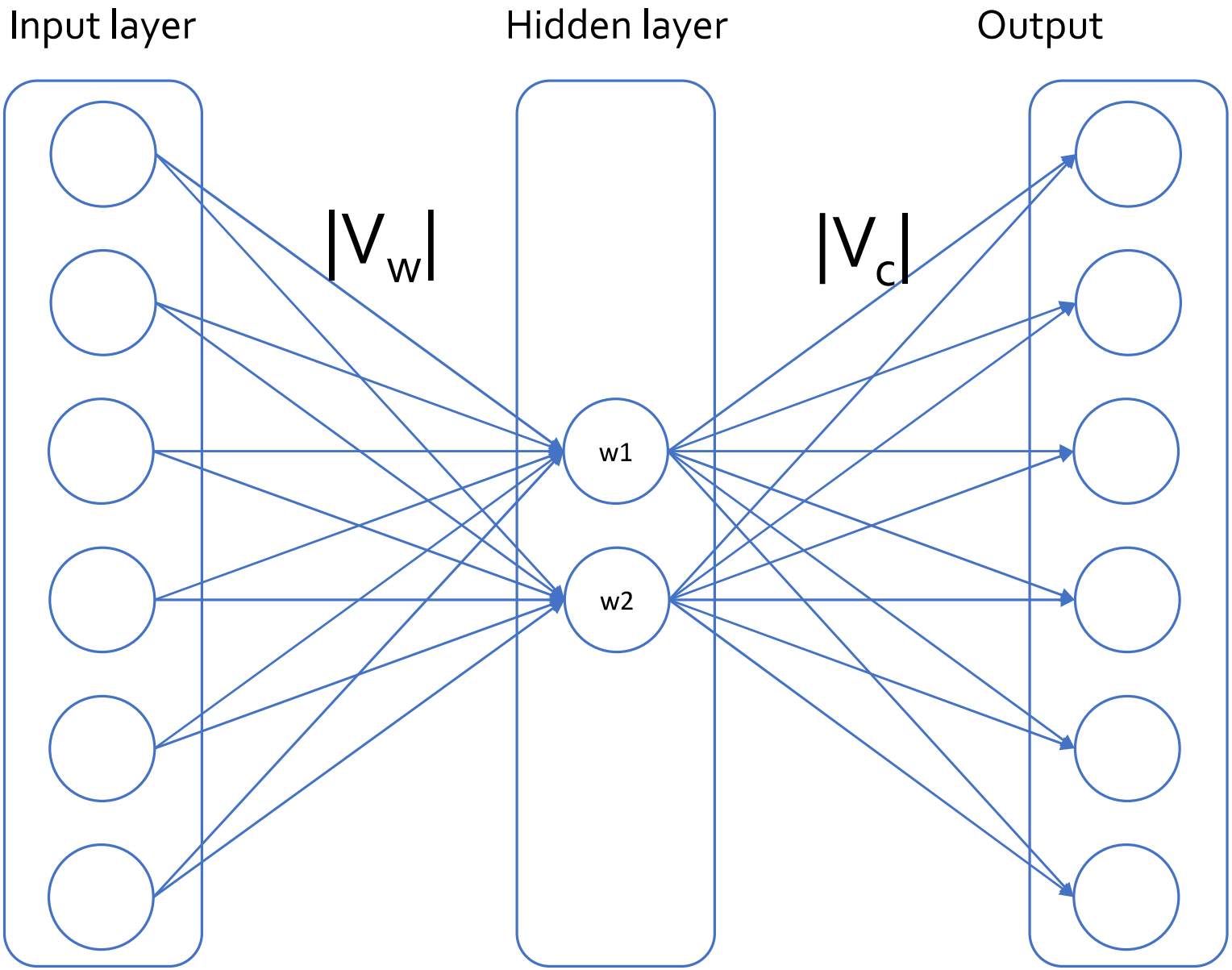
Word Vector for Hope

How does word2Vec work?

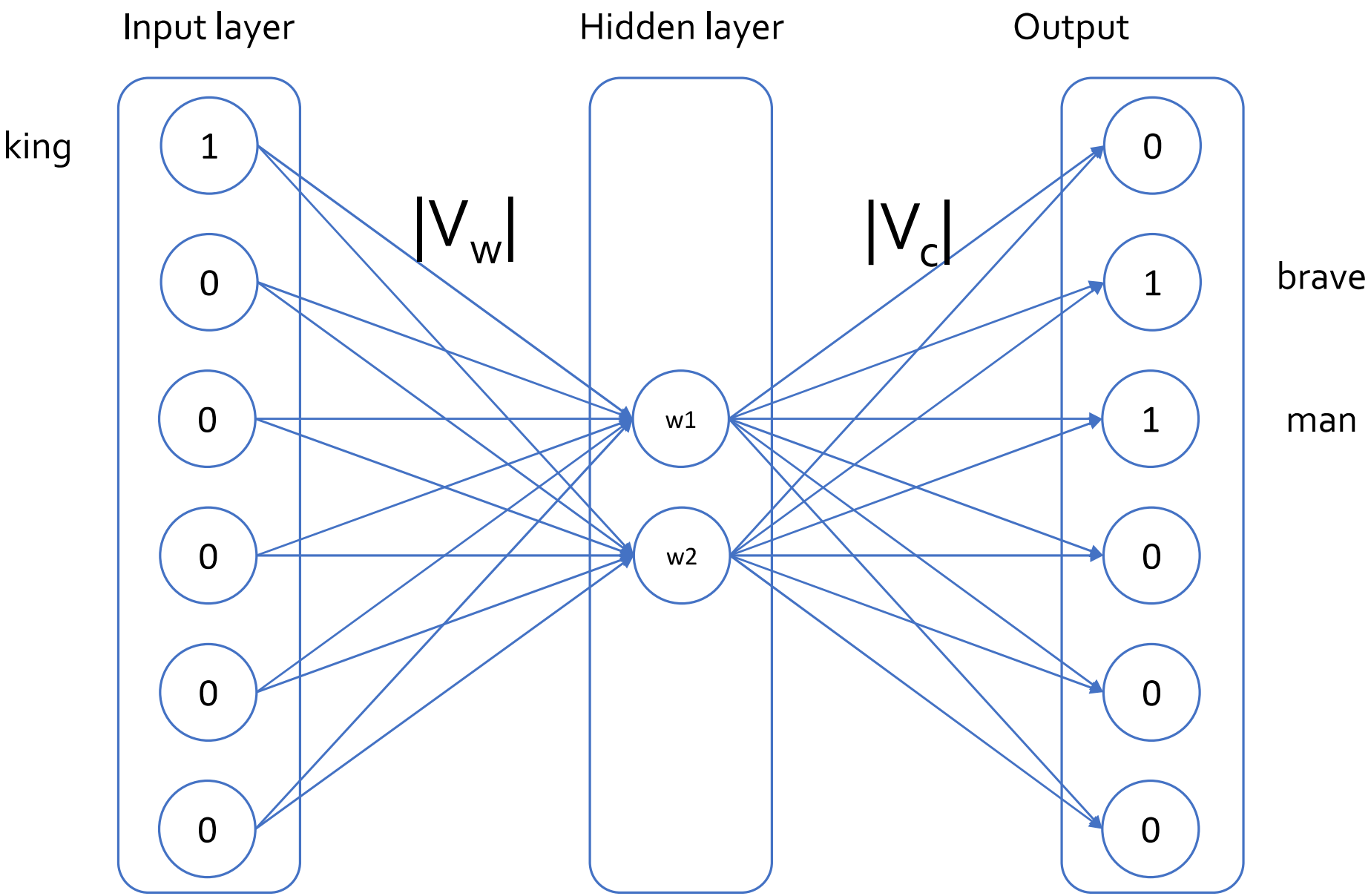
- Represent each word as a $\boldsymbol{d}$ dimensional vector.
- Represent each context as a $\boldsymbol{d}$ dimensional vector.
- Initialize all vectors to random weights.
- Arrange vectors in two matrices, W and C.



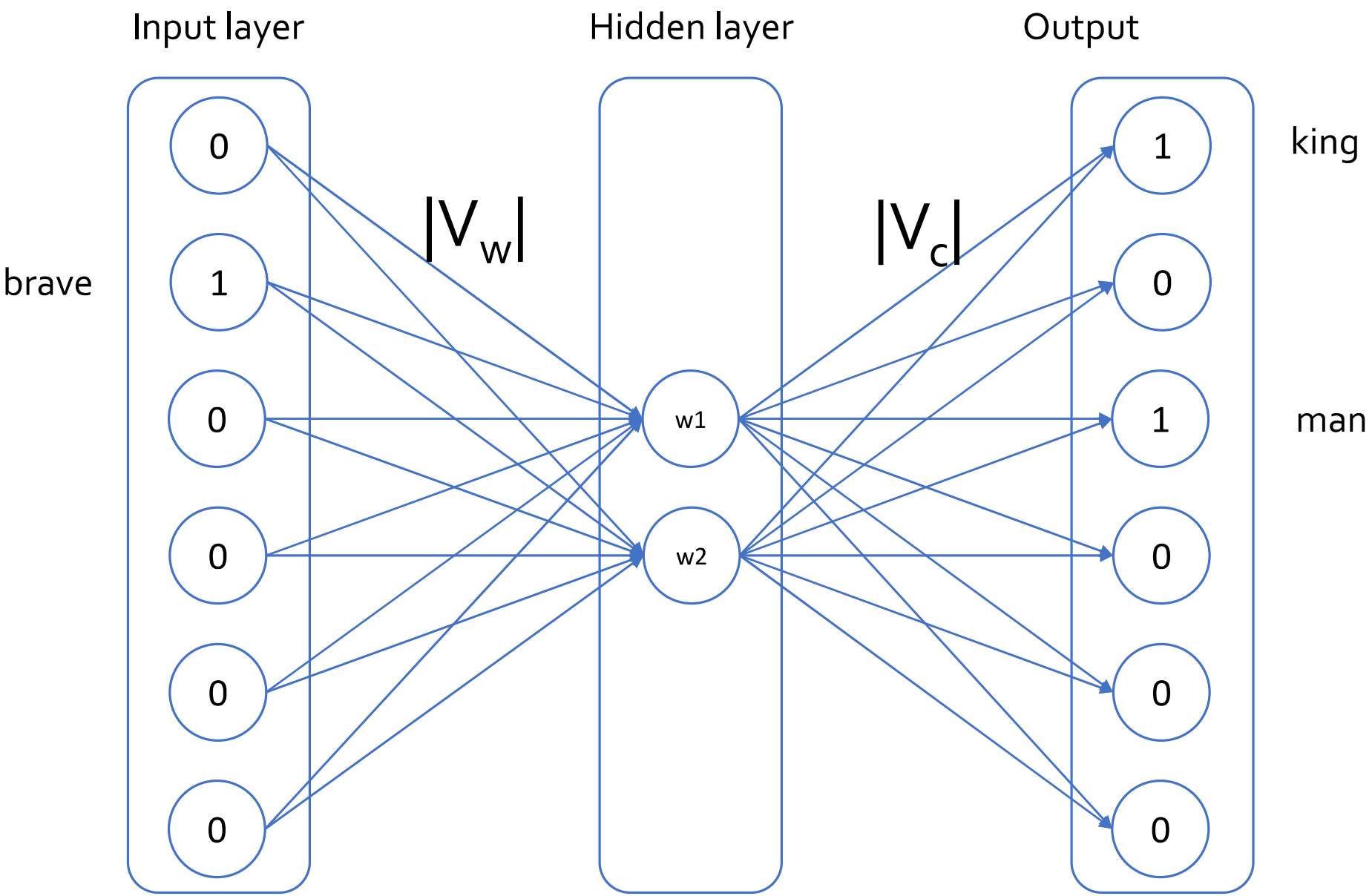
Word2Vec : Neural Network representation



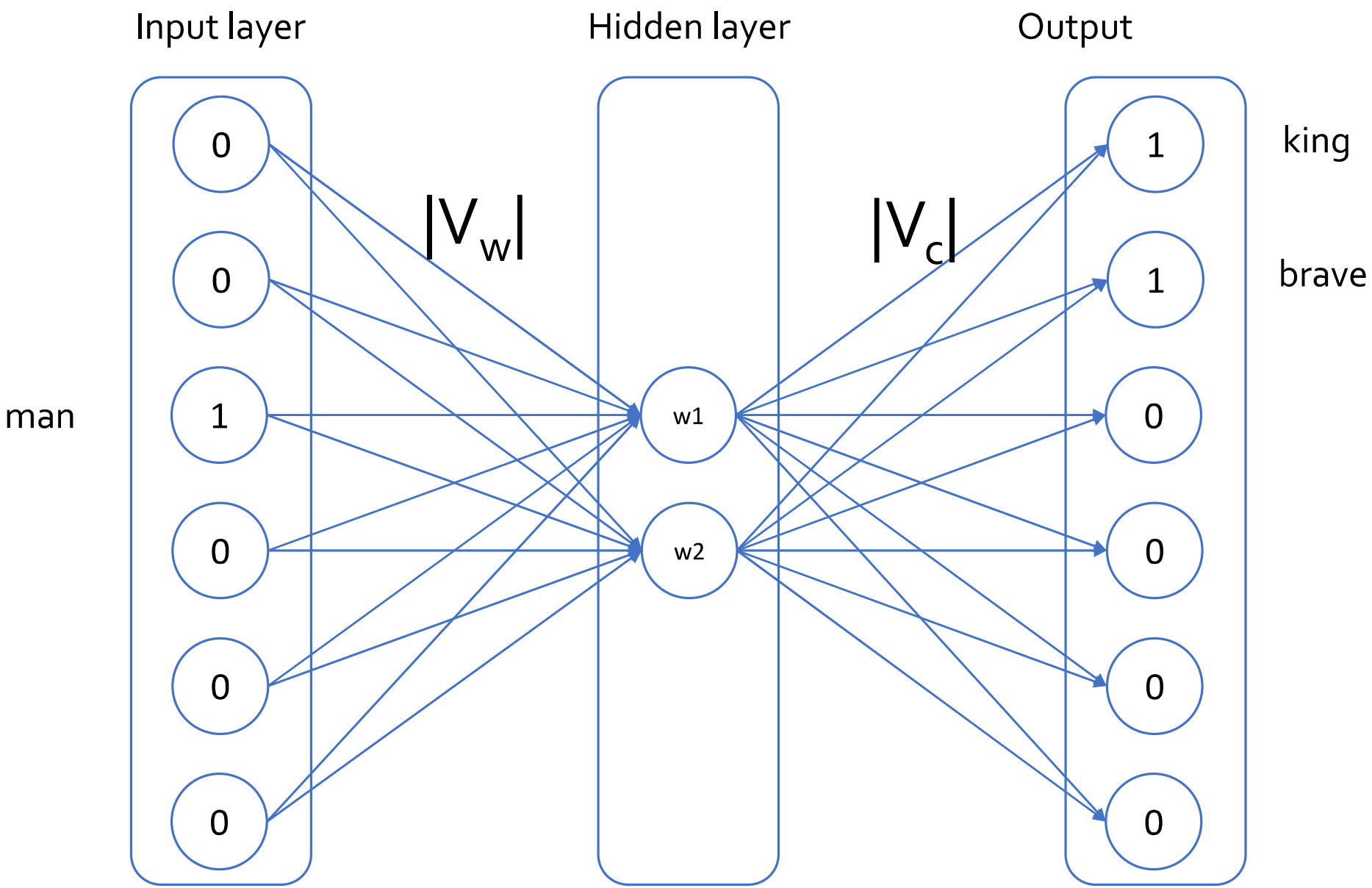
Word2Vec : Neural Network representation



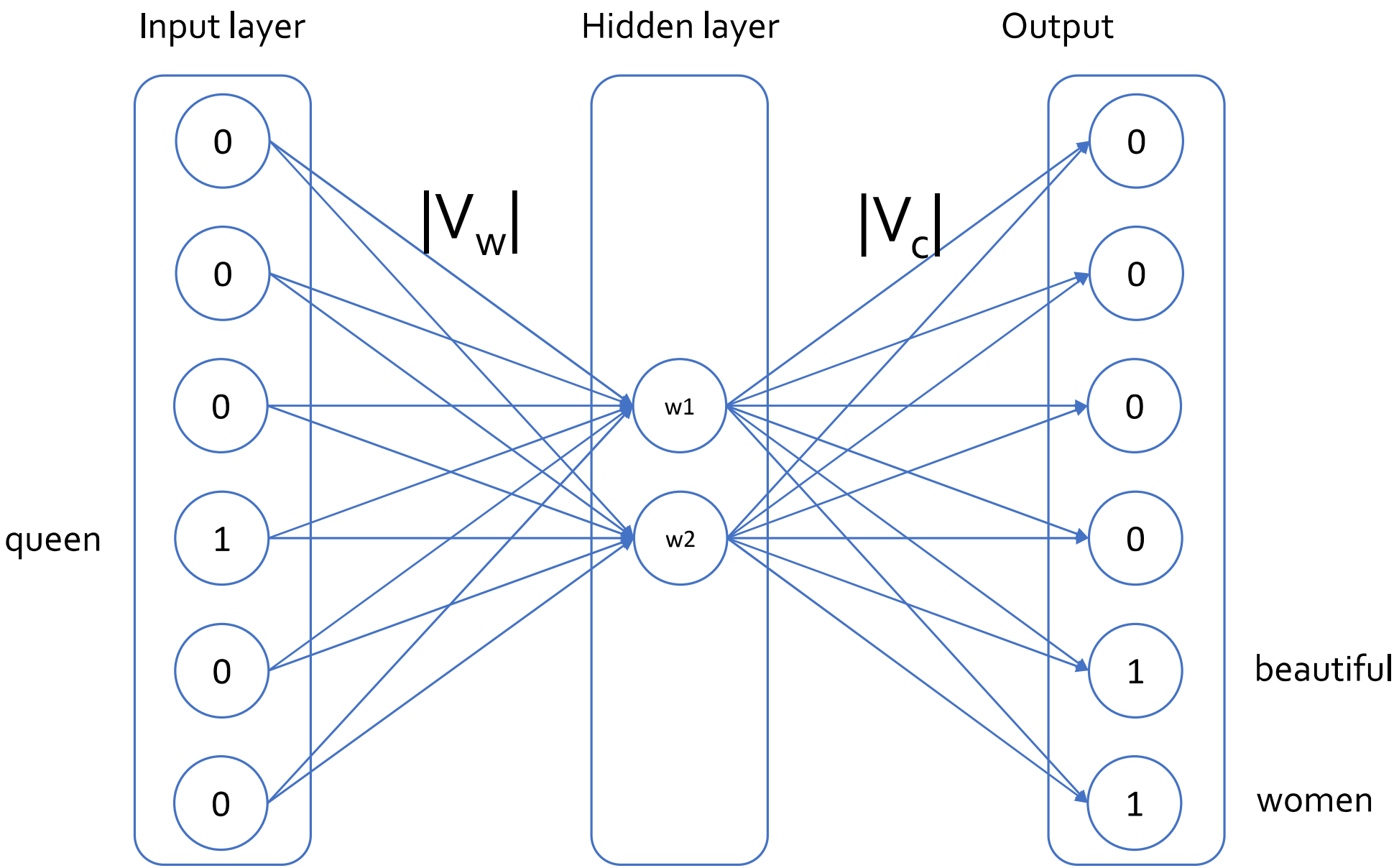
Word2Vec : Neural Network representation



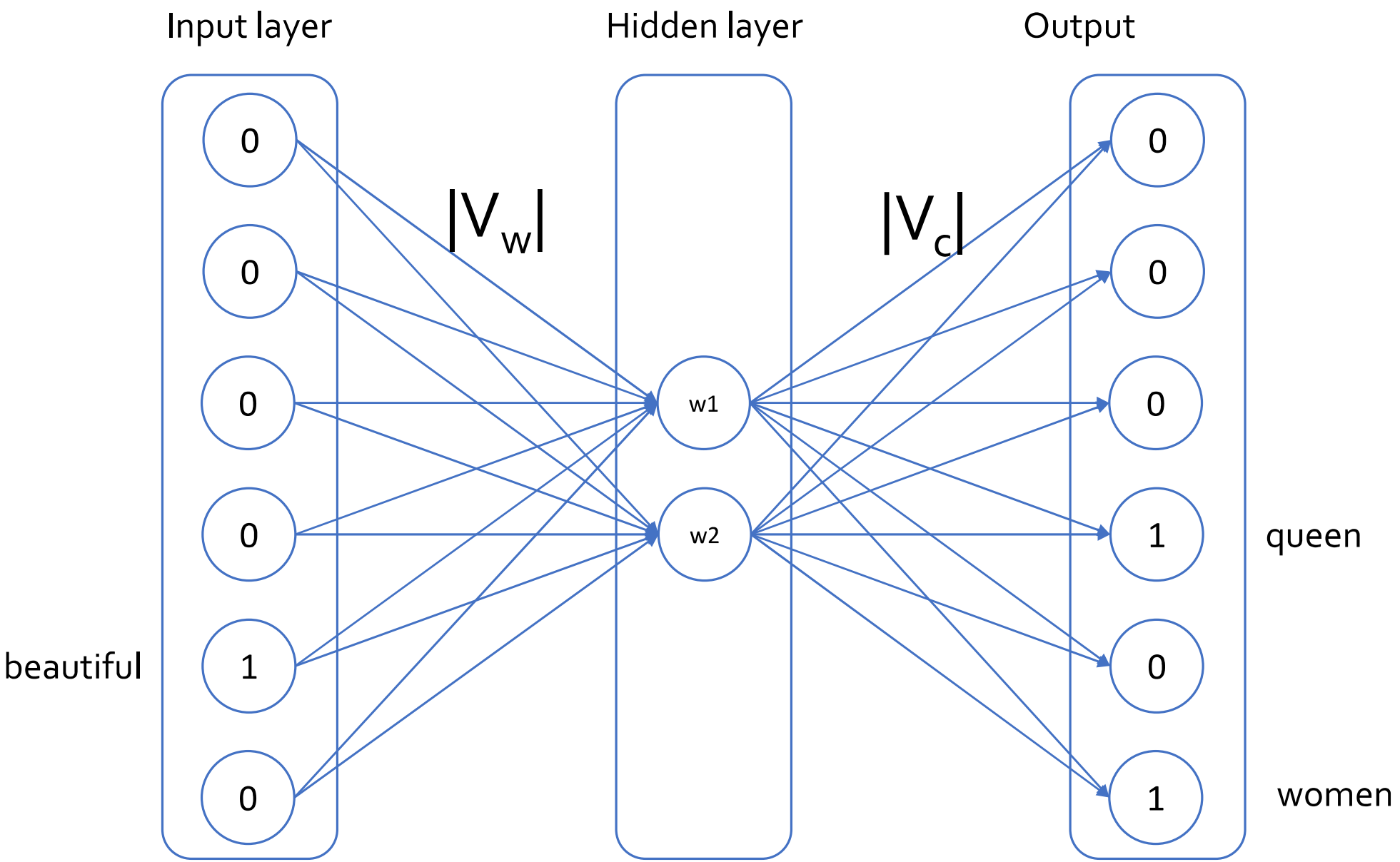
Word2Vec : Neural Network representation



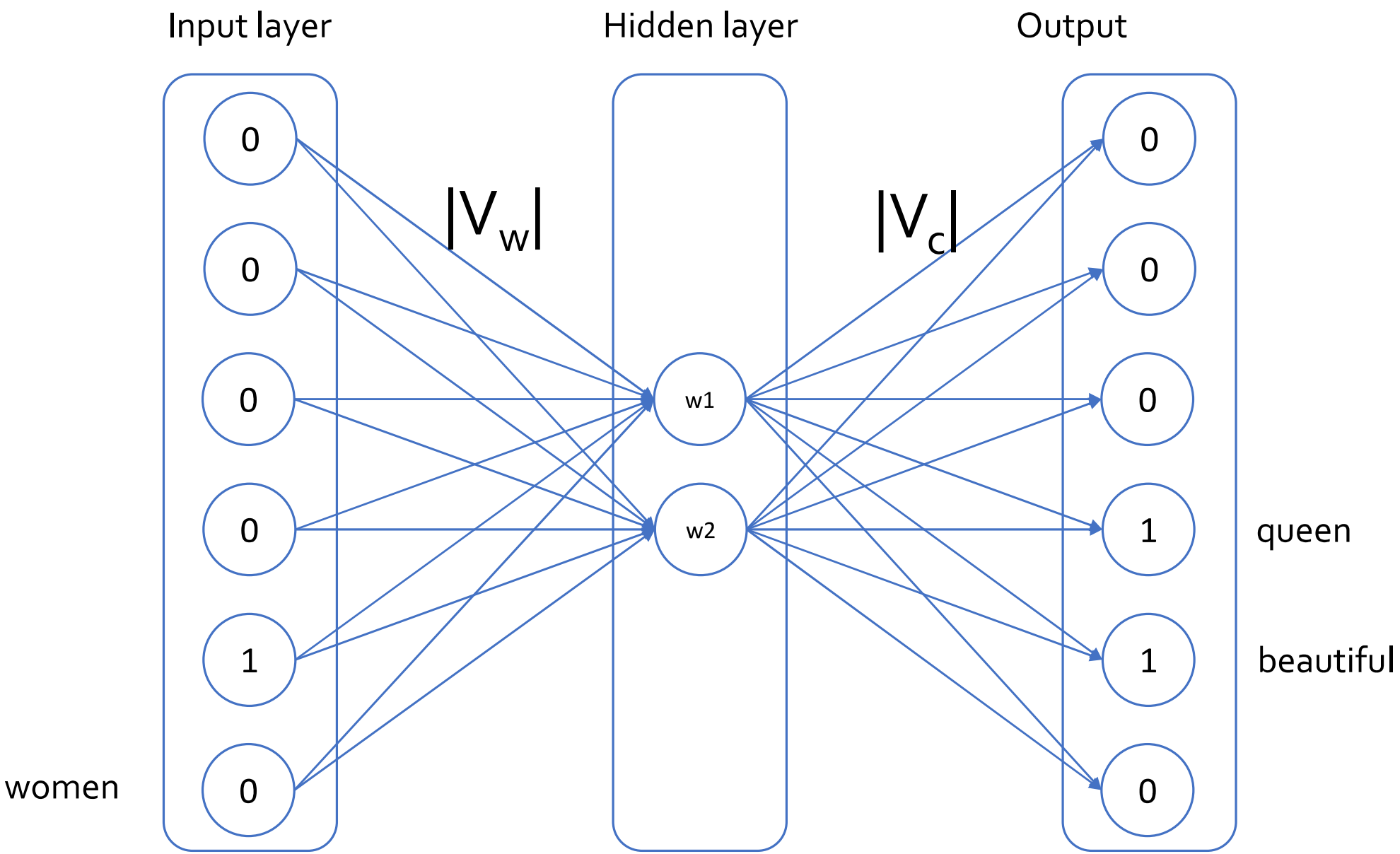
Word2Vec : Neural Network representation



Word2Vec : Neural Network representation



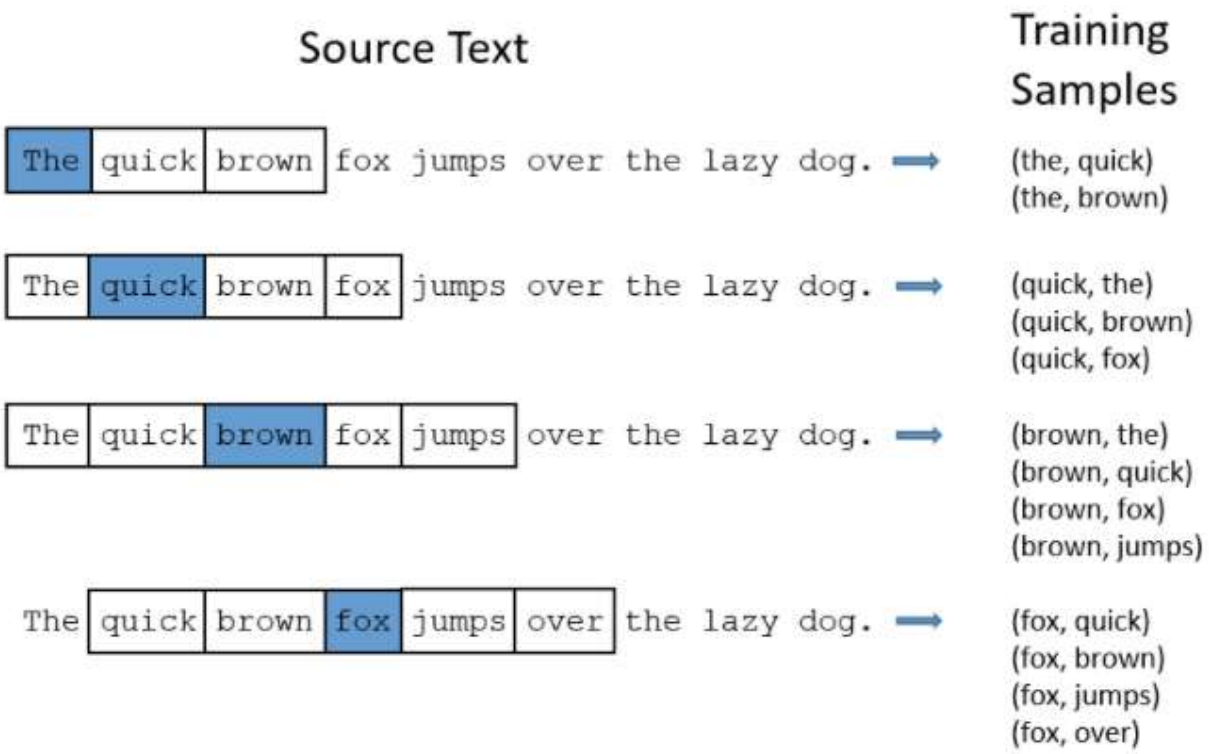
Word2Vec : Neural Network representation



Negative Sampling

https://towardsdatascience.com/nlp-101-negative-sampling-and-glove-936c88f3bc68

- **For each training sample, only the weights corresponding to the target word might get a significant update.**
  - While training a neural network model, in each back-propagation pass we try to update all the weights in the hidden layer. The weight corresponding to non-target words would receive a marginal or no change at all, i.e. in each pass we only make very sparse updates.
  - For every training sample, the calculation of the final probabilities using the softmax is quite an **expensive operation** as it involves a summation of scores over all the words in our vocabulary for **normalizing.**
- "**Negative Sampling**", causes each training sample to update only a tiny percentage of the model's weights.
- The neural network has a massive number of weights, all of which would be slightly changed by each of our billions of training examples!



The network's "correct output" is a one-hot vector when training it on the word pair ("**fox**", **"quick").**
Fox: input
Quick: output

That is, the output neuron associated with "quick" should have a higher prob, whereas the rest of the hundreds of output neurons should have a low prob.
Instead, with **negative sampling**, we'll pick a small number of "negative" words (let's say 5) at random to update the weights. (A "negative" term is one for which we want the network to output a 0 in this context.)

Relations Learned by Word2Vec

- A relation is defined by the vector displacement in the first column. For each start word in the other column, the closest displaced word is shown.

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

- "Efficient Estimation of Word Representations in Vector Space" Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, Arxiv 2013

- In pre trained models embedding has 300 dimension.
  - 300 dimension helps in capturing the semantic relationship b/w words
  - Transferability: Pre-trained embeddings can be useful for transfer learning, where a model is trained on one task and then fine-tuned on a different, related task. Embeddings with a higher dimensionality may be more transferable to a wider range of tasks and domains.
- Trained on google/wiki text with 13 million words
- Window size 5 to 10

Improving Accuracy

- Which one to chose?
  - Skipgram: for large corpus, higher dimensions though performs slower
  - CBOW: faster for small corpus, overfitting problem
- Increasing training dataset
- Increasing vector dimensions
  - The size of the vector is typically set to be between 50 and 1000 dimensions, with 300 being a common default choice.
- Increasing window size

Limitations of Word2vec

1. Word2Vec relies on local information about words, i.e. the context of a word relies only on its neighbors.
2. The obtained word embedding is a byproduct of training a neural network, hence the linear relationships between feature vectors are a black box (kind of).
3. Word2Vec cannot understand out-of-vocabulary (OOV) words, i.e. words not present in training data. You could assign a UNK token which is used for all OOV words or you could use other models that are robust to OOV words.
4. By assigning a distinct vector to each word, Word2Vec ignores the **morphology** of words. For example, *eat*, *eats*, and *eaten* are considered independently different words by Word2Vec, but they come from the same root: *eat*, which might contain useful information.

https://levelup.gitconnected.com/glove-and-fasttext-clearly-explained-extracting-features-from-text-data-1d227ab017b2