

# Word Embeddings: History & Behind the Scene

Alfan F. Wicaksono  
Information Retrieval Lab.  
Faculty of Computer Science  
Universitas Indonesia

# References

- Bengio, Y., Ducharme, R., Vincent, P., & Janvin, C. (2003). A Neural Probabilistic Language Model. *The Journal of Machine Learning Research*, 3, 1137–1155.
- Mikolov, T., Corrado, G., Chen, K., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. *Proceedings of the International Conference on Learning Representations (ICLR 2013)*, 1–12
- Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. *NIPS*, 1–9.
- Morin, F., & Bengio, Y. (2005). Hierarchical Probabilistic Neural Network Language Model. *Aistats*, 5.

# References

Good weblogs for high-level understanding:

- <http://sebastianruder.com/word-embeddings-1/>
- *Sebastian Ruder. On word embeddings - Part 2: Approximating the Softmax.* <http://sebastianruder.com/word-embeddings-softmax>
- <https://www.tensorflow.org/tutorials/word2vec>
- <https://www.gavagai.se/blog/2015/09/30/a-brief-history-of-word-embeddings/>

Some slides were also borrowed from **From Dan Jurafsky's course slide: Word Meaning and Similarity. Stanford University.**

# Terminology

- The term **“Word Embedding”** came from **deep learning community**
- For **computational linguistic community**, they prefer **“Distributional Semantic Model”**
- Other terms:
  - Distributed Representation
  - Semantic Vector Space
  - Word Space

Before we learn Word Embeddings...

# Semantic Similarity

- Word Similarity
  - Near-synonyms
  - “boat” and “ship”, “car” and “bicycle”
- Word Relatedness
  - Can be related any way
  - **Similar:** “boat” and “ship”
  - **Topical Similarity:**
    - “boat” and “water”
    - “car” and “gasoline”

# Why Word Similarity?

- Document Classification
- Document Clustering
- Language Modeling
- Information Retrieval
- ...

# How to compute word similarity?

- Thesaurus-based Approach
  - Using lexical resource, such as **WordNet**, to compute similarity between words
  - For example, two terms are similar if their glosses contain similar words
  - For example, two terms are similar if they are near each other in the thesaurus hierarchy
- Distributional-based Approach
  - Do words have **similar distributional contexts**?



# Thesaurus-based Approach

- Path-based similarity
- Two terms are similar if they are near each other in the thesaurus hierarchy

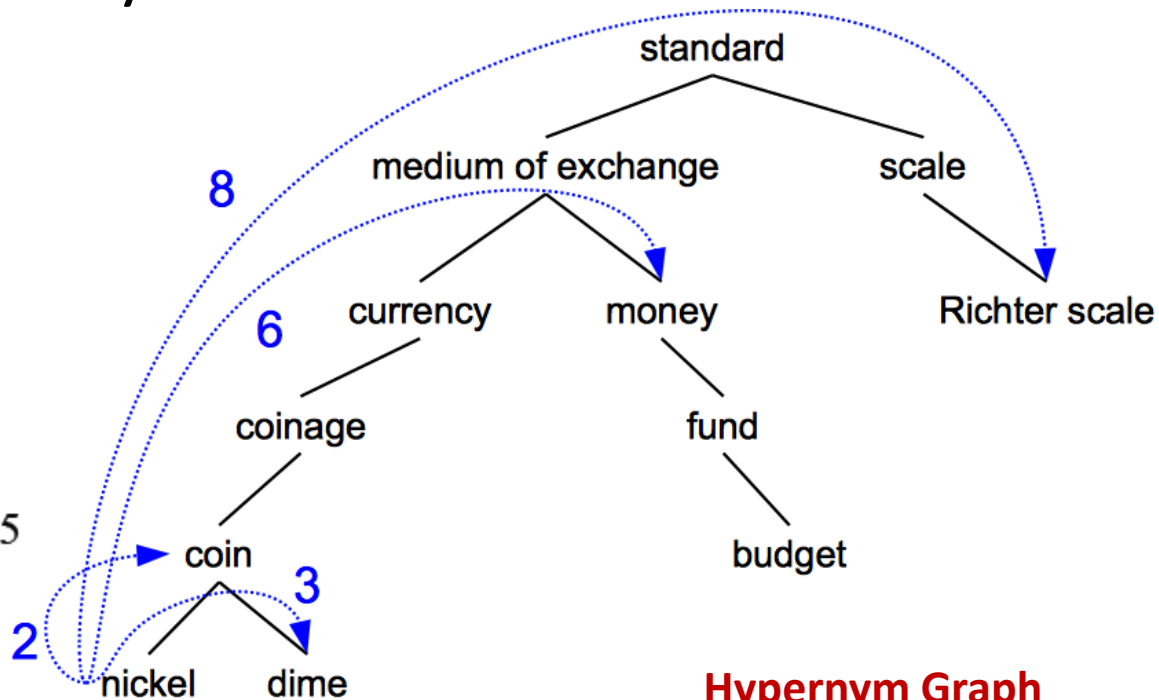
$$\text{simpath}(c_1, c_2) = \frac{1}{\text{pathlen}(c_1, c_2)}$$

$$\text{wordsim}(w_1, w_2) = \max_{c_1 \in \text{senses}(w_1), c_2 \in \text{senses}(w_2)} \text{sim}(c_1, c_2)$$

$$\text{simpath}(\text{nickel}, \text{coin}) = 1/2 = .5$$

$$\text{simpath}(\text{fund}, \text{budget}) = 1/2 = .5$$

$$\text{simpath}(\text{nickel}, \text{currency}) = 1/4 = .25$$



**Hypernym Graph**

# Thesaurus-based Approach

## Other Approach:

- Resnik. 1995, using information content to evaluate semantic similarity in a taxonomy. IJCAI.
- Dekang Lin. 1998. An information-theoretic definition of similarity. ICML.
- Lesk Algorithm

# Thesaurus-based Approach

## Problems...

- We don't have a thesaurus for every language
- For Indonesian, our WordNet is not complete
  - Many words are missing
  - Connections between senses are missing
  - ...

# Distributional-based Approach

- Based on the idea that **contextual information alone constitutes a viable representation of linguistic items.**
- As opposed to formal linguistics and the Chomsky tradition.
- Zellig Haris (1954): **“...if A and B have almost identical environments, we say that they are synonyms...”**

**Word Embeddings are based on this idea!**

# Distributional-based Approach

A bottle of *tesgüino* is on the table  
Everybody likes *tesgüino*  
*Tesgüino* makes you drunk  
We make *tesgüino* out of corn

- From context words humans can guess *tesgüino* means
  - an alcoholic beverage Like **beer**
- Intuition for algorithm:
  - **Two words are similar if they have similar word contexts**

# Distributional-based Approach

Suppose, in our large corpus, we have:

- ... sepertinya berita **tidak** begitu disukai oleh ...
- ... mungkin mereka **tidak** mengira bahwa selama ini ...
- ... gw **nggak** mengira kalau selama ini ...
- ... menurut gw berita itu **nggak** begitu adil sih ...

Suppose you don't know the meaning of "**nggak**", but you know "**tidak**".

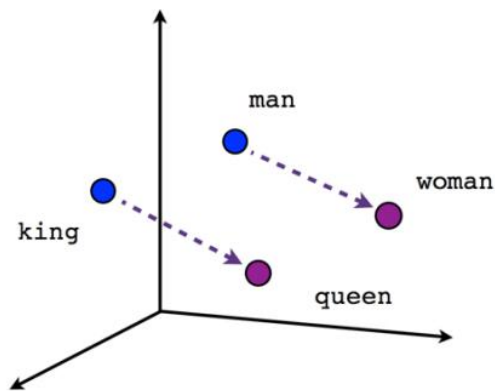
Can you infer something about "**tidak**" and "**nggak**" ?...and why?

# Why Word Embeddings?

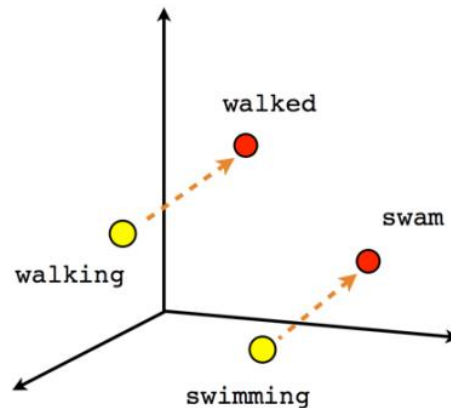
- Word representations are a critical component of many NLP systems. It is common to represent words as indices in a vocabulary, **but this fails to capture the rich relational structure of the lexicon.**
  - Representing words as unique, discrete ids furthermore leads to data sparsity, and usually means that we may need more data in order to successfully train statistical models
- **Vector-based models** do much better in this regard. They encode continuous similarities between words as distance or angle between word vectors in a high-dimensional space.

# Why Word Embeddings?

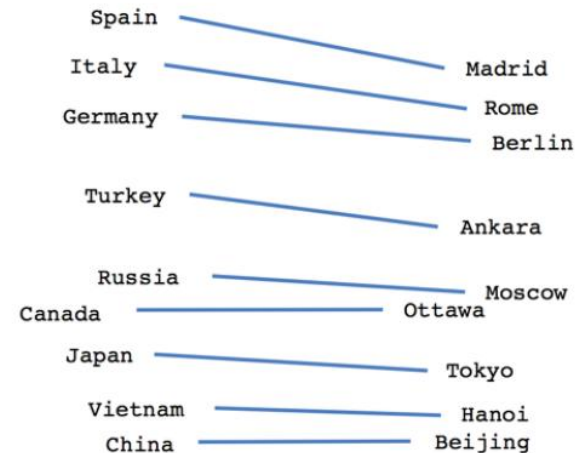
- Can capture the rich relational structure of the lexicon



Male-Female



Verb tense



Country-Capital



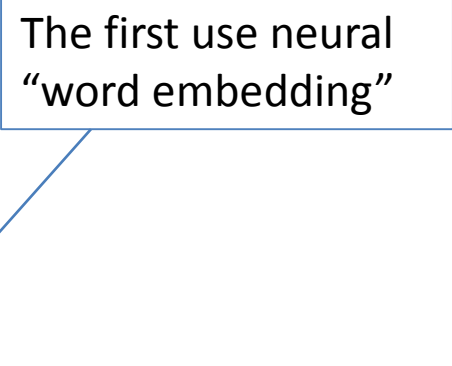
# Word Embeddings

- Any technique that maps a word (or phrase) from **it's original high-dimensional sparse input** space to **a lower-dimensional dense vector space**.
- Vectors whose relative similarities correlate with semantic similarity
- Such vectors are used both as an end in itself (for computing similarities between terms), and as a representational basis for downstream NLP tasks, such as POS tagging, NER, text classification, etc.

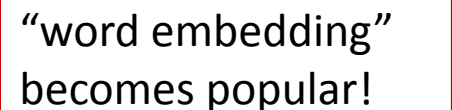
# Continuous Representation of Words

- In research on Information Retrieval and Topic Modeling
  - Simple Vector Space Model (**Sparse**)
  - Latent Semantic Analysis
  - Probabilistic LSA
  - Latent Dirichlet Allocation
- **[Distributional Semantic]**
  - VSMs, LSA, SVD, etc.
  - Self Organizing Map
  - Bengio et al's **Word Embedding (2003)**
  - Mikolov et al's **Word2Vec (2013)**
  - Pennington et al's **GloVe (2014)**

The first use neural  
"word embedding"



"word embedding"  
becomes popular!



# Continuous Representation of Words

The Differences:

- In **information retrieval**, LSA and topic models use *documents as contexts*.
  - Capture semantic relatedness (“boat” and “water”)
- **Distributional semantic models** use *words as contexts* (more natural in linguistic perspective)
  - Capture semantic similarity (“boat” and “ship”)

**Word Embedding**



# Distributional Semantic Models

Other classification based on (Baroni et al., ACL 2014)

- Count-based models
  - Simple VSMs
  - LSA
  - Singular Value Decomposition (Golub & VanLoan, 1996)
  - Non-negative Matrix Factorization (Lee & Seung, 2000)
- Predictive-based models (**neural network**)
  - Self Organizing Map
  - Bengio et al's **Word Embedding (2003)**
  - Mikolov et al's **Word2Vec (2013)**

# DSMs or Word Embeddings

- Count-based model
  - first collecting context vectors and then reweighting these vectors based on various criteria
- Predictive-based model (neural network)
  - vector weights are directly set to optimally predict the contexts in which the corresponding words tend to appear
  - Similar words occur in similar contexts, the system naturally learns to assign similar vectors to similar words.

# DSMs or Word Embeddings

- There is no need for *deep* neural networks in order to build good word embeddings.
  - the Skipgram and CBoW models included in the [word2vec](#) library – are shallow neural networks
- There is no qualitative difference between (current) *predictive* neural network models and *count-based* distributional semantics models.
  - they are different computational means to arrive at the same type of semantic model

# Vector Space Model

## Term-Document Matrix

Each cell is the count of word  $t$  in document  $d$

	D1	D2	D3	D4	D5
ekonomi	0	1	40	38	1
pusing	4	5	1	3	30
keuangan	1	2	30	25	2
sakit	4	6	0	4	25
inflasi	8	1	15	14	1

Vector of D2 = **[1, 5, 2, 6, 1]**

# Vector Space Model

## Term-Document Matrix

Each cell is the count of word  $t$  in document  $d$

	D1	D2	D3	D4	D5
ekonomi	0	1	40	38	1
pusing	4	5	1	3	30
keuangan	1	2	30	25	2
sakit	4	6	0	4	25
inflasi	8	1	15	14	1

Two documents are similar if they have similar vector!

**D3 = [40, 1, 30, 0, 15]**

**D4 = [38, 3, 25, 4, 14]**



# Vector Space Model

## Term-Document Matrix

Each cell is the count of word  $t$  in document  $d$

	D1	D2	D3	D4	D5
ekonomi	0	1	40	38	1
pusing	4	5	1	3	30
keuangan	1	2	30	25	2
sakit	4	6	0	4	25
inflasi	8	1	15	14	1

Vector of word “sakit” = **[4, 6, 0, 4, 25]**

# Vector Space Model

## Term-Document Matrix

Each cell is the count of word  $t$  in document  $d$

	D1	D2	D3	D4	D5
ekonomi	0	1	40	38	1
pusing	4	5	1	3	30
keuangan	1	2	30	25	2
sakit	4	6	0	4	25
inflasi	8	1	15	14	1

**Two words are similar if they have similar vector!**

**pusing = [4, 5, 1, 3, 30]**

**sakit = [4, 6, 0, 4, 25]**

# Vector Space Model

## Term-Context Matrix

- Previously, we use entire **Documents as our Context** of word
  - document-based models capture **semantic relatedness** (e.g. “boat” – “water”), **NOT semantic similarity** (e.g. “boat” – “ship”)
- We can get precise vector representation of word (for semantic similarity task) if we use **smaller context, i.e, Words as our Context!**
  - Window of **N** words
- A word is defined by a **vector of over counts of context words.**

# Vector Space Model

Sample context of **4 words** ...

No	Potongan konteks 4 kata
1	... kekuatan ekonomi dan <b>inflasi</b> ...
2	... obat membuat sakit <b>pusing</b> ...
3	... sakit <b>pening</b> di kepala ...
4	... <b>keuangan</b> menipis serta inflasi ...
5	...

# Vector Space Model

## Term-Context Matrix

- Sample context of **4 words** ...

	ekonomi	obat	sakit	kepala	keuangan	...
inflasi	2	0	0	0	3	...
pusing	0	1	6	6	1	...
pening	0	2	6	6	0	...
keuangan	2	0	0	0	4	...
...						

**Two words are similar in meaning if they have similar context vector!**

**Context Vector of “pusing” = [0, 1, 6, 6, 1, ...]**

**Context Vector of “pening” = [0, 2, 6, 6, 0, ...]**

# Vector Space Model

- **Weighting:** it practically works well... instead of just using raw counts.
- For Term-Document matrix
  - We usually use **TF-IDF**, instead of Raw Counts (only TF)
- For Term-Context matrix
  - We usually use **Pointwise Mutual Information** (PMI)

# Word Analogy Task

- **Father** is to **Mother** as **King** is to \_\_\_\_\_ ?
- **Good** is to **Best** as **Smart** is to \_\_\_\_\_ ?
- **Indonesia** is to **Jakarta** as **Malaysia** is to \_\_\_\_\_ ?
  
- It turns out that the previous Word-Context based vector model is good for such analogy task.

$$\mathbf{V}_{\text{king}} - \mathbf{V}_{\text{father}} + \mathbf{V}_{\text{mother}} = \mathbf{V}_{\text{queen}}$$

# Word Embeddings for Information Retrieval

- The two passages are indistinguishable for the query term “Albuquerque”!
  - Both contains the same number of “Albuquerque”, i.e. Only one!
- Word embeddings can help us to distinguish them!
- Which one is the real one talking about “Albuquerque”??
  - **#terms in the document related to “Albuquerque”**

*Albuquerque* is the most populous city in the U.S. state of New Mexico. The high-altitude city serves as the county seat of Bernalillo County, and it is situated in the central part of the state, straddling the Rio Grande. The city population is 557,169 as of the July 1, 2014, population estimate from the United States Census Bureau, and ranks as the 32nd-largest city in the U.S. The Metropolitan Statistical Area (or MSA) has a population of 902,797 according to the United States Census Bureau's most recently available estimate for July 1, 2013.

(a)

Allen suggested that they could program a BASIC interpreter for the device; after a call from Gates claiming to have a working interpreter, MITS requested a demonstration. Since they didn't actually have one, Allen worked on a simulator for the Altair while Gates developed the interpreter. Although they developed the interpreter on a simulator and not the actual device, the interpreter worked flawlessly when they demonstrated the interpreter to MITS in Albuquerque, New Mexico in March 1975; MITS agreed to distribute it, marketing it as Altair BASIC.



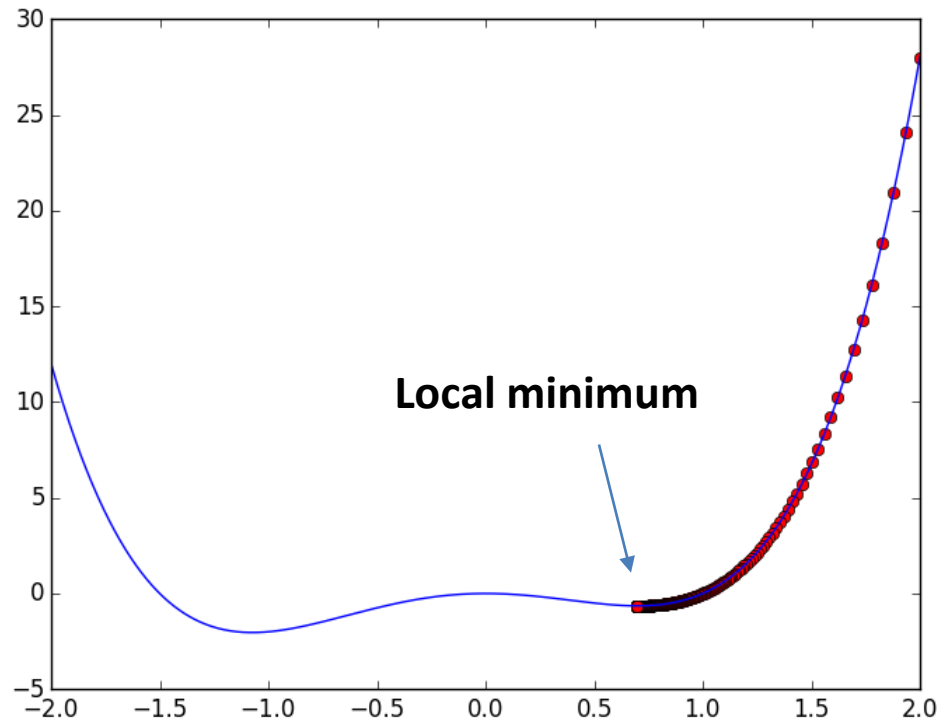
Some basics ...

# Gradient Descent (GD)

**Problem:** carilah nilai  $x$  sehingga fungsi  $f(x) = 2x^4 + x^3 - 3x^2$  mencapai titik *local minimum*.

Misal, kita pilih  $x$  dimulai dari  $x=2.0$ :

Algoritma GD konvergen pada titik  $x = 0.699$ , yang merupakan local minimum.



# Gradient Descent (GD)

## Algorithm:

For  $t = 1, 2, \dots, N_{\max}$  :

$$x_{t+1} \leftarrow x_t - \alpha_t f'(x_t)$$

If  $\|f'(x_{t+1})\| < \varepsilon$  then return "converged on critical point"

If  $\|x_t - x_{t+1}\| < \varepsilon$  then return "converged on  $x$  value"

If  $f(x_{t+1}) > f(x_t)$  then return "diverging"

$\alpha_t$  : **learning rate** atau **step size** pada iterasi ke- $t$

$\varepsilon$ : sebuah bilangan yang sangat kecil

$N_{\max}$ : batas banyaknya iterasi, atau disebut **epoch** jika iterasi selalu sampai akhir

Algoritma dimulai dengan menebak nilai  $x_1$  !

**Tips:** pilih  $\alpha_t$  yang tidak terlalu kecil, juga tidak terlalu besar.

# Gradient Descent (GD)

Kalau parameter-nya ada banyak ?

Carilah  $\theta = \theta_1, \theta_2, \dots, \theta_n$  sehingga  $f(\theta_1, \theta_2, \dots, \theta_n)$  mencapai local minimum !

*while not converged :*

$$\theta_1^{(t+1)} \leftarrow \theta_1^{(t)} - \alpha_t \frac{\partial}{\partial \theta_1^{(t)}} f(\theta^{(t)})$$

$$\theta_2^{(t+1)} \leftarrow \theta_2^{(t)} - \alpha_t \frac{\partial}{\partial \theta_2^{(t)}} f(\theta^{(t)})$$

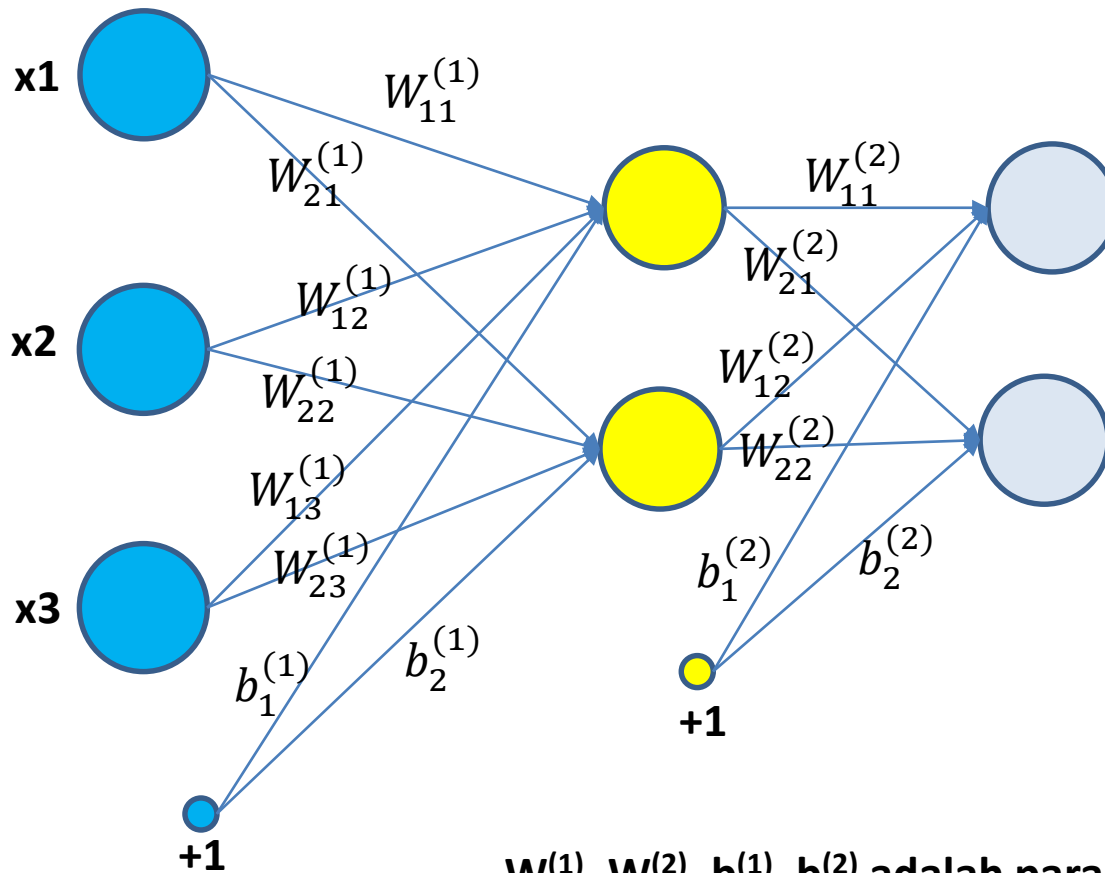
...

$$\theta_n^{(t+1)} \leftarrow \theta_n^{(t)} - \alpha_t \frac{\partial}{\partial \theta_n^{(t)}} f(\theta^{(t)})$$

Dimulai dengan menebak nilai awal  $\theta = \theta_1, \theta_2, \dots, \theta_n$

# Multilayer Neural Network (Multilayer Perceptron)

Misal, ada 3-layer NN, dengan 3 input unit, 2 hidden unit, dan 2 output unit.



$W^{(1)}, W^{(2)}, b^{(1)}, b^{(2)}$  adalah parameter !

# Multilayer Neural Network (Multilayer Perceptron)

Misal, untuk activation function, kita gunakan fungsi **hyperbolic tangent**.  $f(x) = \tanh(x)$

Untuk menghitung **output di hidden layer**:

$$z_1^{(2)} = W_{11}^{(1)} x_1 + W_{12}^{(1)} x_2 + W_{13}^{(1)} x_3 + b_1^{(1)}$$

$$z_2^{(2)} = W_{21}^{(1)} x_1 + W_{22}^{(1)} x_2 + W_{23}^{(1)} x_3 + b_2^{(1)}$$

$$a_1^{(2)} = f(z_1^{(2)})$$

$$a_2^{(2)} = f(z_2^{(2)})$$

Ini hanyalah perkalian matrix !

$$z^{(2)} = W^{(1)} x + b^{(1)} = \begin{bmatrix} W_{11}^{(1)} & W_{12}^{(1)} & W_{13}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} & W_{23}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix}$$

# Multilayer Neural Network (Multilayer Perceptron)

Jadi, Proses **feed-forward** secara keseluruhan hingga menghasilkan output di kedua output unit adalah:

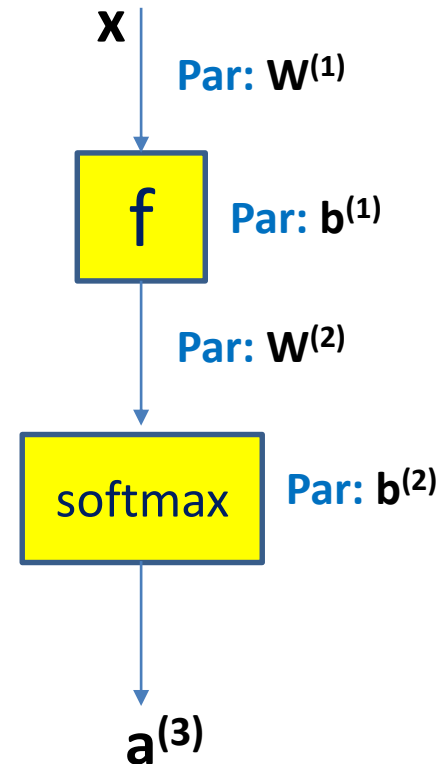
$$z^{(2)} = W^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$

$$h_{W,b}(x) = a^{(3)} = \text{softmax}(z^{(3)})$$

$$a_i^{(3)} = \frac{\exp(z_i^{(3)})}{\sum_i \exp(z_i^{(3)})}$$



# Multilayer Neural Network (Multilayer Perceptron)

## Learning

Misal, Cost function kita adalah **cross-entropy loss**:  
**m** adalah banyaknya training examples.

$$J(W, b) = -\frac{1}{m} \sum_{i=1}^m \sum_{j \in C} y_{i,j} \log(p_{i,j}) + \frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (W_{ji}^{(l)})^2$$

Regularization terms



# Multilayer Neural Network (Multilayer Perceptron)

## Learning

### Batch Gradient Descent

inisialisasi  $W$ ,  $b$

while not converged :

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b)$$

$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W, b)$$

**Bagaimana cara menghitung gradient ??**

**Backpropagation Algorithm**

Neural Language Model (Bengio et al., 2003)

# Main Goal: Language Model

Actually, their main goal is to develop a language model, i.e., **conditional probability of the next word given all the previous ones**:

$$P(w_t | w_{t-1} w_{t-2} \dots w_2 w_1)$$

So that, we can compute the **likelihood of a whole document or sentence** by the product of the probabilities each word given its previous words:

Given a document or sentence  $w_1, w_2, w_3, \dots, w_{T-1}, w_T$

$$P(w_1 w_2 \dots w_{T-1} w_T) = \prod_{t=1}^T P(w_t | w_{t-1} \dots w_2 w_1)$$

# N-Gram Language Model

In **N-Gram** language model, we often relax the conditional probability of a word into just its **n-1 previous words** (Markov Assumption):

$$P(w_t | w_{t-1} \dots w_1) \approx P(w_t | w_{t-1} \dots w_{t-n+1})$$

For example, using **Bi-Gram**:

$P(i, learn, word, embedding) =$

$$P(i | \langle start \rangle) P(learn | i) P(word | learn) P(embedding | word) P(\langle end \rangle | embedding)$$

**Using MLE**, we can estimate the parameter using **Frequency Counts**:

$$\hat{P}(w_t | w_{t-1} \dots w_{t-n+1}) = \frac{\text{count}(w_t w_{t-1} \dots w_{t-n+1})}{\text{count}(w_{t-1} \dots w_{t-n+1})}$$

# Problem with N-Gram

- There should be much more information in the sequence that immediately precedes the word to predict **than just the identity of the previous couple of words**.
- It's **NOT** taking into account the **“similarity”** between words.
- For example, in the training data:

The cat is walking in the bedroom

- Then, the model should generalize for the following sentence:

A dog was running in a room

**WHY ?** Dog-Cat, The-A, Bedroom-Room have similar semantic and grammatical roles !

# Neural Language Model

Given a document or sentence  $w_1, w_2, w_3, \dots, w_{T-1}, w_T$

Where each word belongs to a Vocabulary,  $w_i \in V$

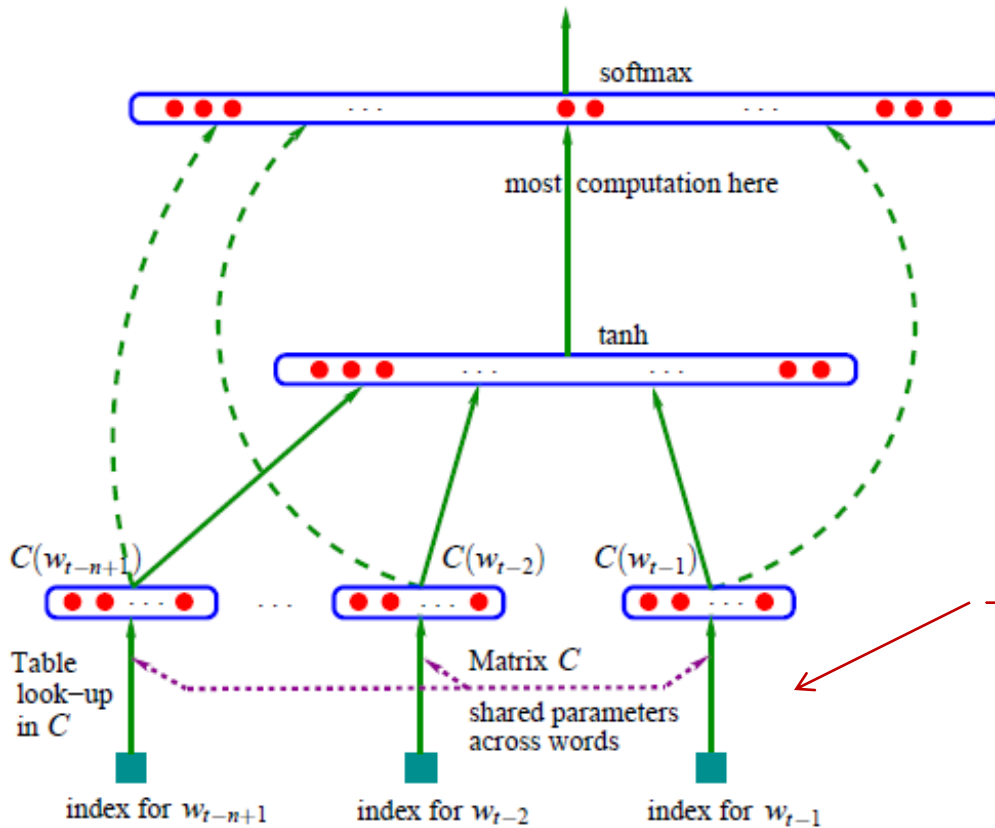
We want to learn good model for:

$$\begin{aligned} f(w_t, w_{t-1}, \dots, w_{t-n+1}) &= P(w_t \mid w_{t-1} \dots w_{t-n+1}) \\ &= \text{soft max}(\text{score}(w_t, \text{context})) \\ &= \frac{\exp(\text{score}(w_t, \text{context}))}{\sum_{j \in V} \exp(\text{score}(w_j, \text{context}))} \end{aligned}$$

$$\text{context} = w_{t-n+1}, \dots, w_{t-1}$$

# Neural Language Model

$i$ -th output =  $P(w_t = i | context)$



We have **Embedding/Projection Matrix**

$$|V| \left\{ \begin{array}{c} \left[ \begin{array}{cccc} \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \end{array} \right] \\ m \end{array} \right. C_{|V| \times m}$$

$m$  is size of word vector.

$$C(i) \in R^m$$

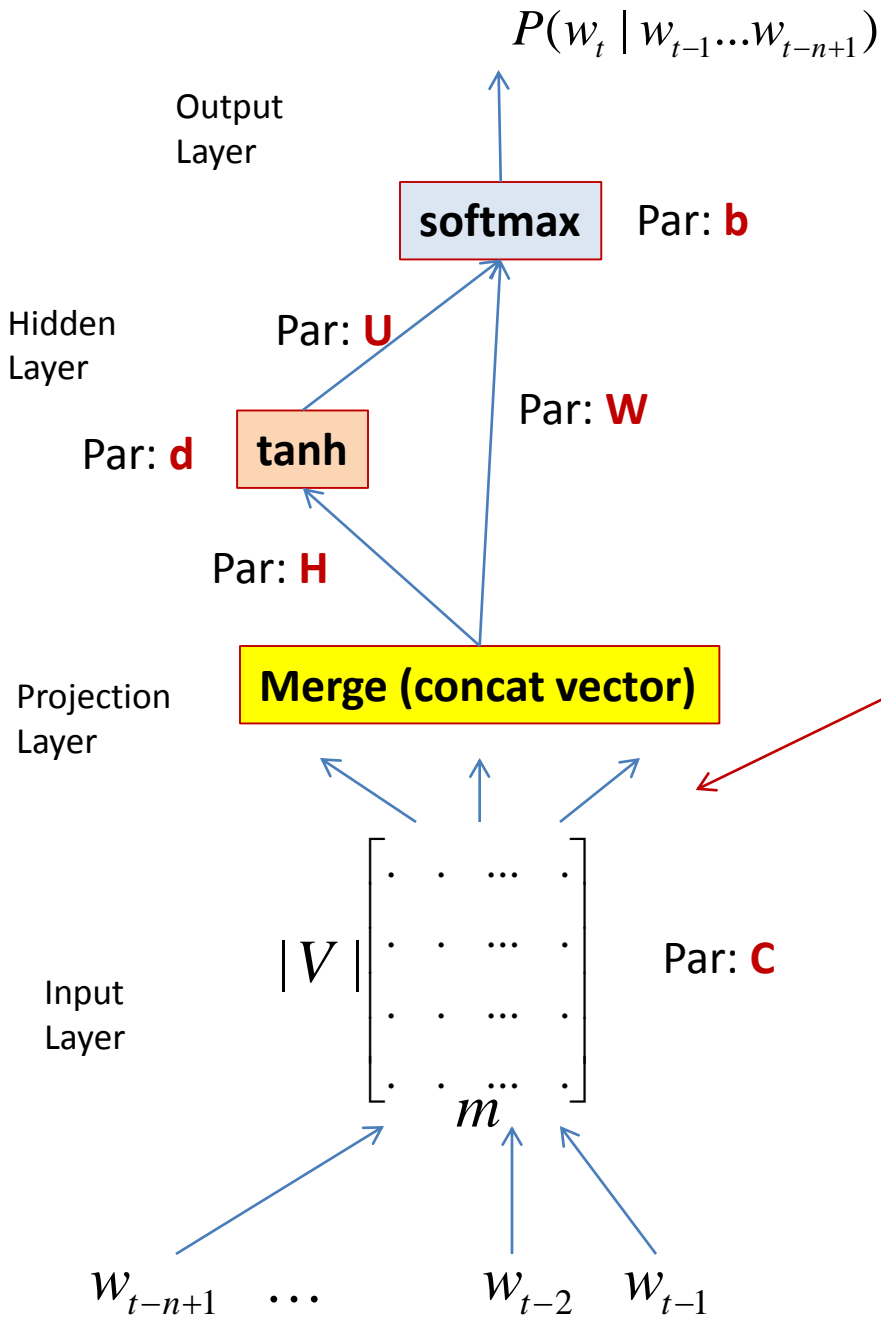
$C(i)$  is a function that maps **A Word  $i$**  into **Its Vector**

$$f(i, w_{t-1}, \dots, w_{t-n+1}) = P(w_t = i | context)$$

$$= g(i, C(w_{t-1}), \dots, C(w_{t-n+1}))$$

**Score for a particular output:**

$g$  is a neural network function.



## Feed-Forward Process

$$P(w_t | w_{t-1} \dots w_{t-n+1}) = \frac{\exp(y_{w_t})}{\sum_{i \in V} \exp(y_i)}$$

$$y_{w_t} = b + Wx + U \tanh(d + Hx)$$

$$x = \text{concat}(C(w_{t-1}), \dots, C(w_{t-n+1}))$$

## Total Parameters:

$$\theta = (b, d, W, U, H, C)$$

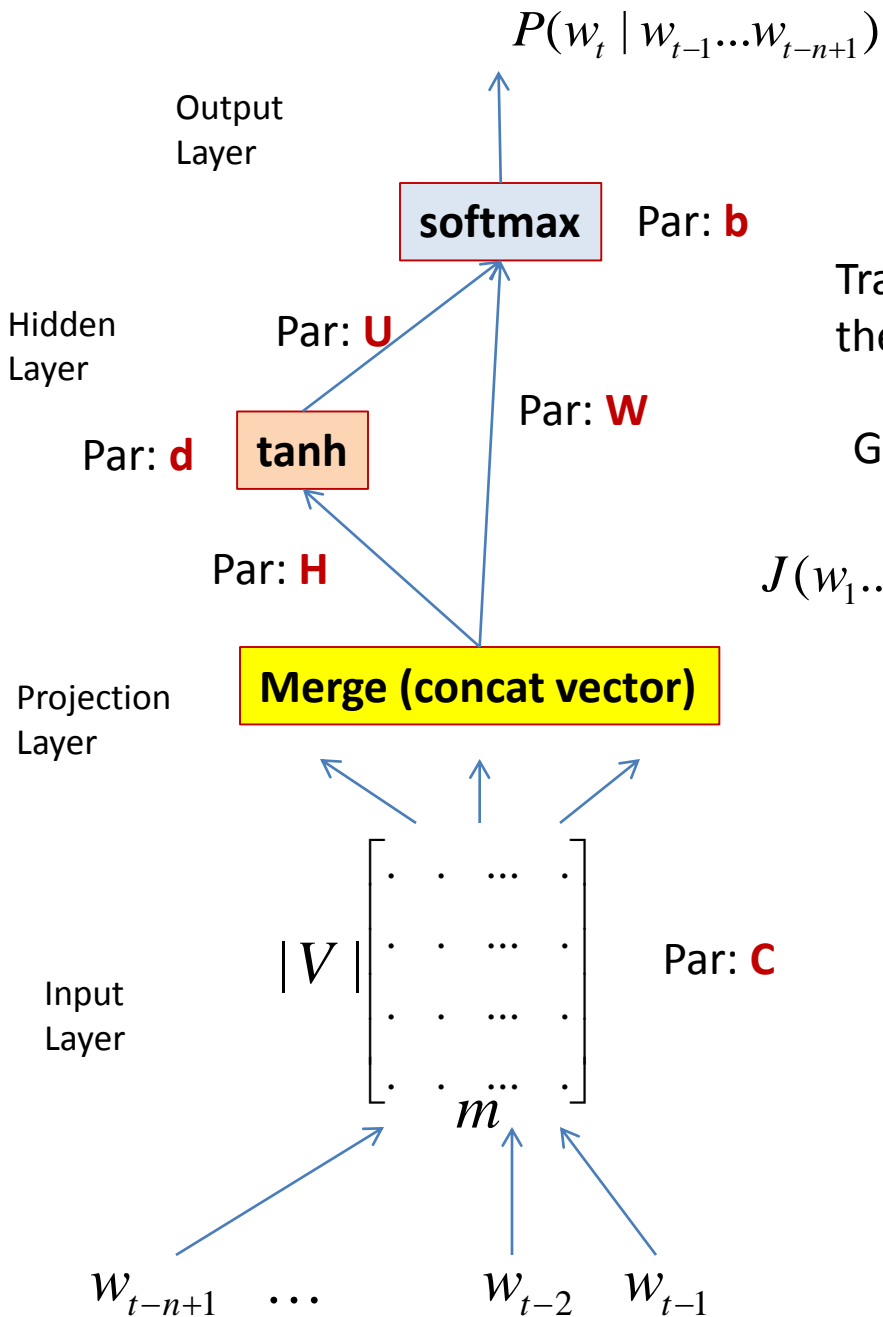
$$U \in \mathbb{R}^{|V| \times h} \quad b \in \mathbb{R}^{|V|}$$

$$C \in \mathbb{R}^{|V| \times m} \quad d \in \mathbb{R}^h$$

$$W \in \mathbb{R}^{|V| \times (n-1)m}$$

$$H \in \mathbb{R}^{h \times (n-1)m}$$





## Training

Training is achieved by looking  $\theta$  that maximizes the following Cost Function:

Given training data  $w_1, w_2, w_3, \dots, w_{T-1}, w_T$

$$J(w_1 \dots w_T; \theta) = \frac{1}{T} \sum_{t=1}^T \log f(w_t, w_{t-1}, \dots, w_{t-n+1}; \theta) + R(\theta)$$

Regularization terms


**How? We can use Gradient Ascent!**

**Iterative update using the following formula:**

$$\theta^{new} := \theta^{old} + \alpha \frac{\partial}{\partial \theta} J(w_1 \dots w_T; \theta)$$

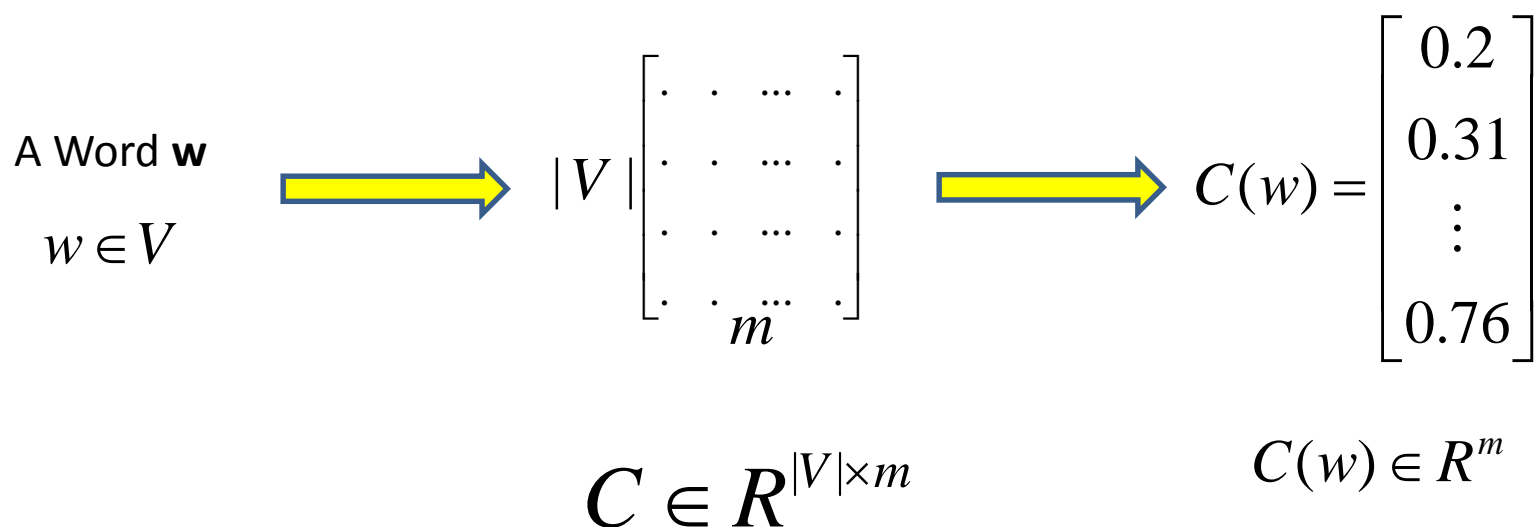
Learning rate

# Where are the Word Embeddings?

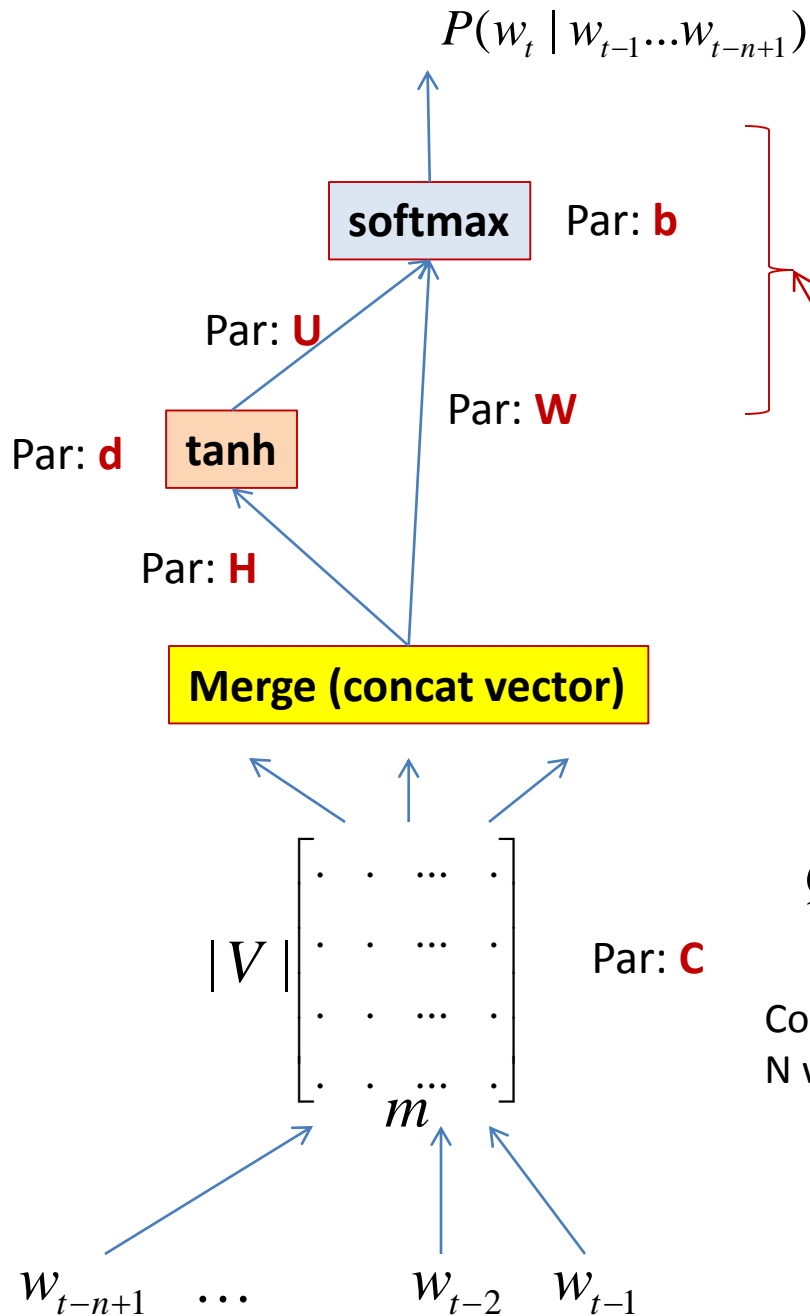
- The previous model actually aims at building the language model.
  - So, **where is the Word Embedding model that we need?**
- **The answer is:** If you just need the Word Embedding model, you just need the matrix **C**  


# Where are the Word Embeddings?

- After all parameters (including  $\mathbf{C}$ ) are optimized, then we can use  $\mathbf{C}$  to map a word into its vector !



# Problem?



Computation in this area is very **COSTLY!**

Size of Vocabulary can reach **100.000 !**

(Mikolov et al., 2013)

Computational Complexity of this model per each training sample:

$$Q = (N \times m) + (N \times m \times h) + (h \times |V|)$$

Composition of projection layer  
N words  $\times$  size of vector m

Between projection layer & hidden layer

**Dominating Term!**

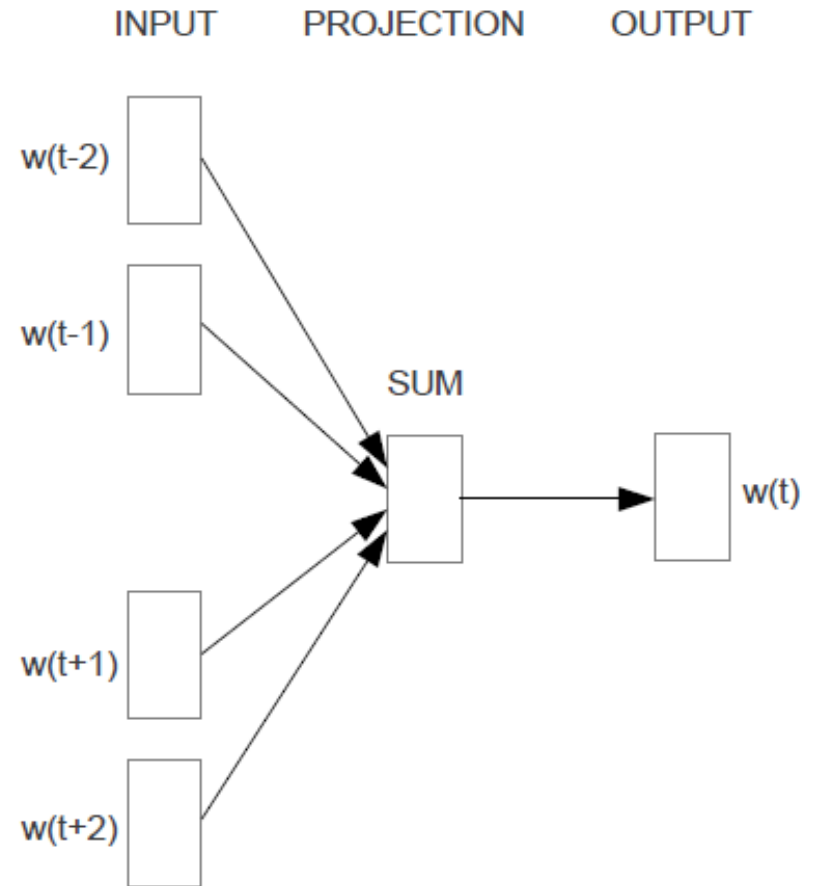
Word2Vec (Mikolov et al., 2013)

# Word2Vec

- One of the most popular Word Embedding models nowadays!
- Computationally less expensive compared to the previous model
- There are two types of models:
  - Continuous Bag of Words Model (**CBOW**)
  - Skip-Gram Model

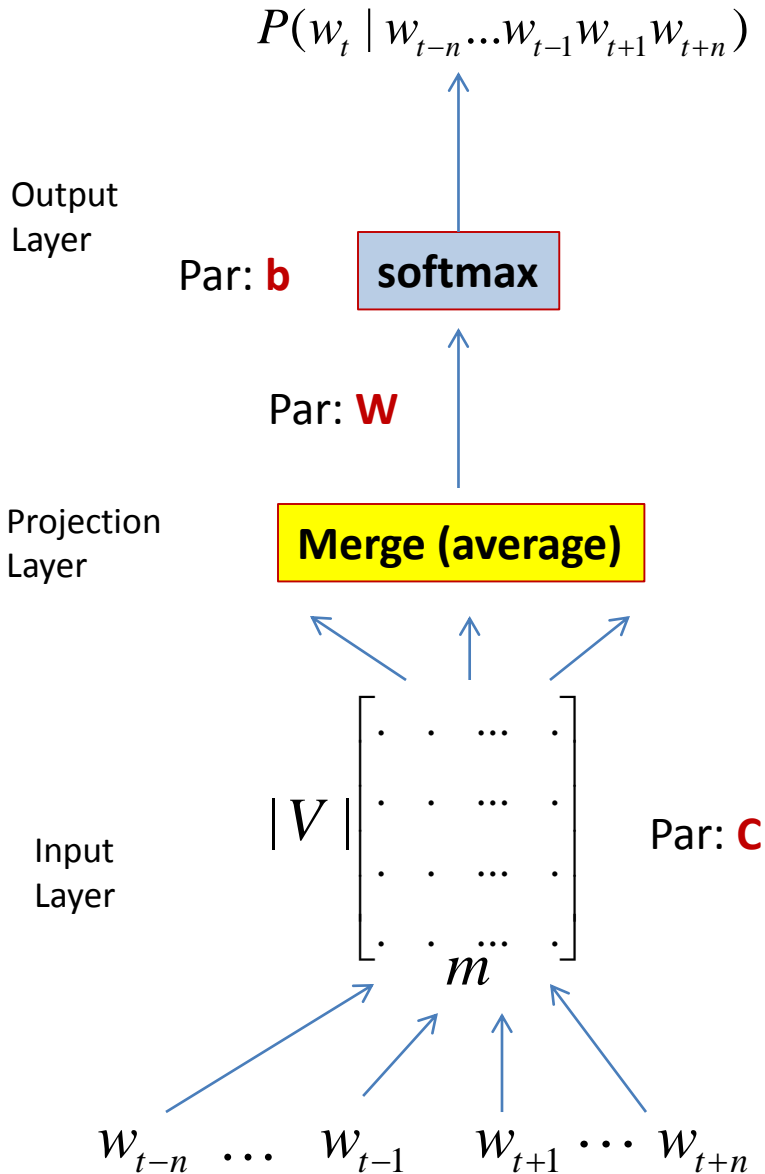
# Continuous Bag-of-Words

- **Bengio's language model** only looks at previous words as a context for predictions.
- **Mikolov's CBOW** looks at **n words before and after the target words**.
  - Non-linear hidden layer is also removed.
  - All word vectors get projected into the same position (**their vectors are averaged**)
- “Bag-of-Words” is because **the order of words in the history does not influence the projection.**



We seek a model for  $P(w_t | w_{t-n} \dots w_{t-1} w_{t+1} \dots w_{t+n})$

# Continuous Bag-of-Words



## Feed-Forward Process

$$P(w_t | w_{t-n} \dots w_{t-1} w_{t+1} \dots w_{t+n}) = \frac{\exp(y_{w_t})}{\sum_{i \in V} \exp(y_i)}$$

$$y_{w_t} = b + Wx$$

$$x = \text{average}(C(w_{t-n}), C(w_{t-1}), \dots, C(w_{t+1}), C(w_{t+n}))$$

## Total Parameters:

$$\theta = (b, W, C)$$

$$C \in R^{|V| \times m}$$

$$W \in R^{|V| \times (2n)m}$$



# Continuous Bag-of-Words

$$P(w_t | w_{t-n} \dots w_{t-1} w_{t+1} w_{t+n})$$

**Training**

Output Layer

Par: **b**

softmax

Training is achieved by looking  $\theta$  that maximizes the following Cost Function:

Par: **W**

Merge (average)

Given training data  $w_1, w_2, w_3, \dots, w_{T-1}, w_T$

$$J(w_1 \dots w_T; \theta) = \frac{1}{T} \sum_{t=1}^T \log P(w_t | w_{t-n} \dots w_{t-1} w_{t+1} \dots w_{t+n}) + R(\theta)$$

Projection Layer

Regularization terms

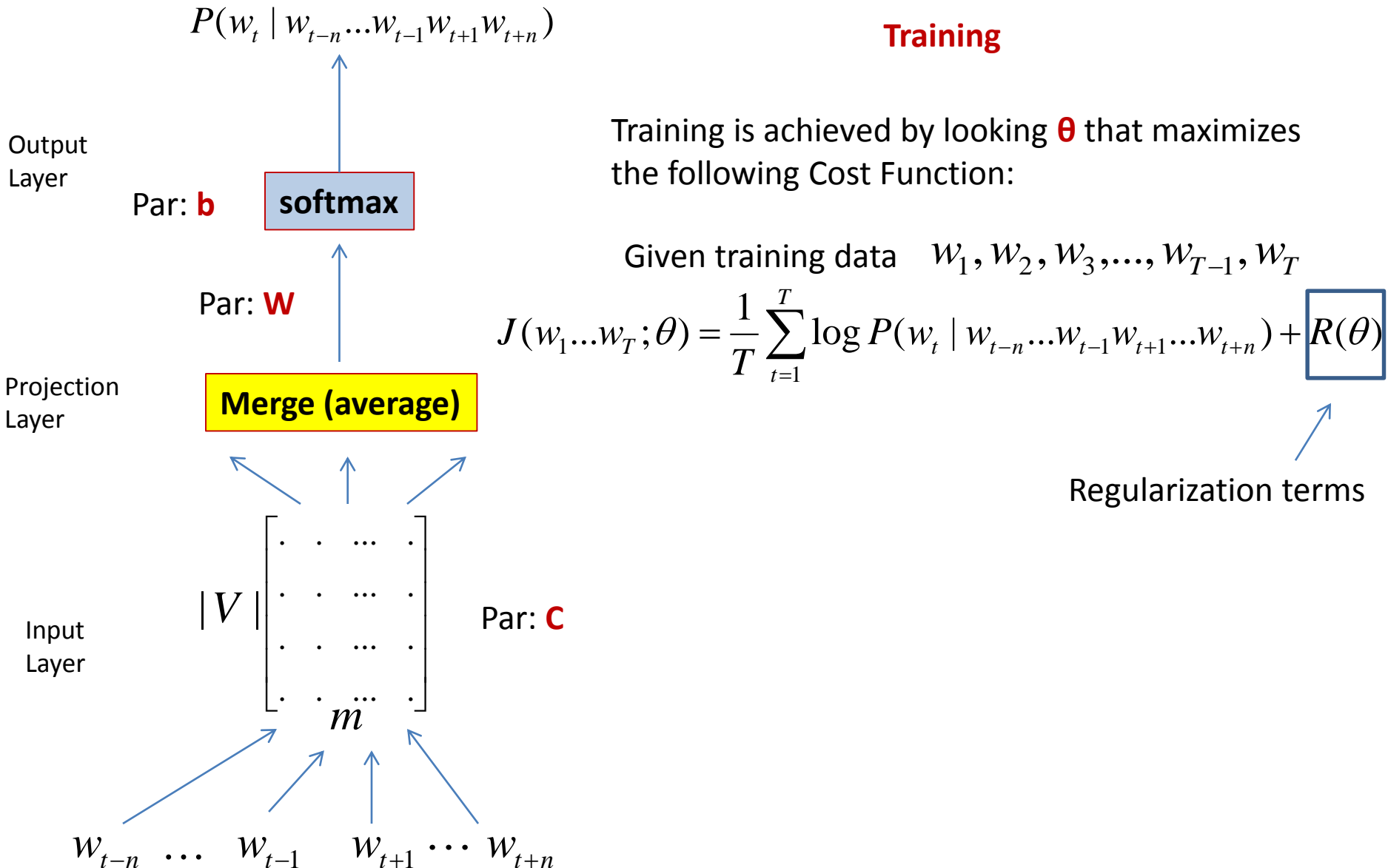
Input Layer

$$|V| \begin{bmatrix} \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \end{bmatrix}$$

Par: **C**

$m$

$w_{t-n} \dots w_{t-1} w_{t+1} \dots w_{t+n}$



# Continuous Bag-of-Words

$$P(w_t | w_{t-n} \dots w_{t-1} w_{t+1} w_{t+n})$$

**Training**

Output  
Layer

**Hierarchical Softmax**

Actually, if we use **vanilla softmax**, then the computational complexity is still costly.

$$Q = (2N \times m) + (m \times |V|)$$

Projection  
Layer

**Merge (average)**

To solve this problem, they use **Hierarchical Softmax layer**. This layer uses a **binary tree representation** of the output layer with  $|V|$  units.

Input  
Layer

$$|V| \begin{bmatrix} \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \end{bmatrix}$$

$m$

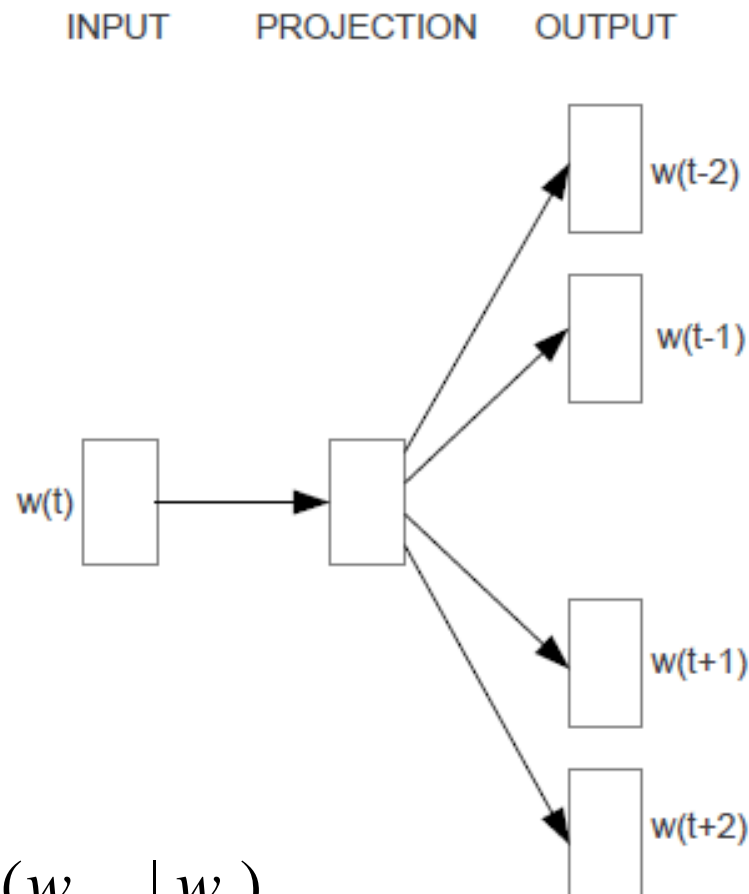
$$Q = (2N \times m) + (m \times \log(|V|))$$

**(Morin & Bengio, 2005)**

$$w_{t-n} \dots w_{t-1} w_{t+1} \dots w_{t+n}$$

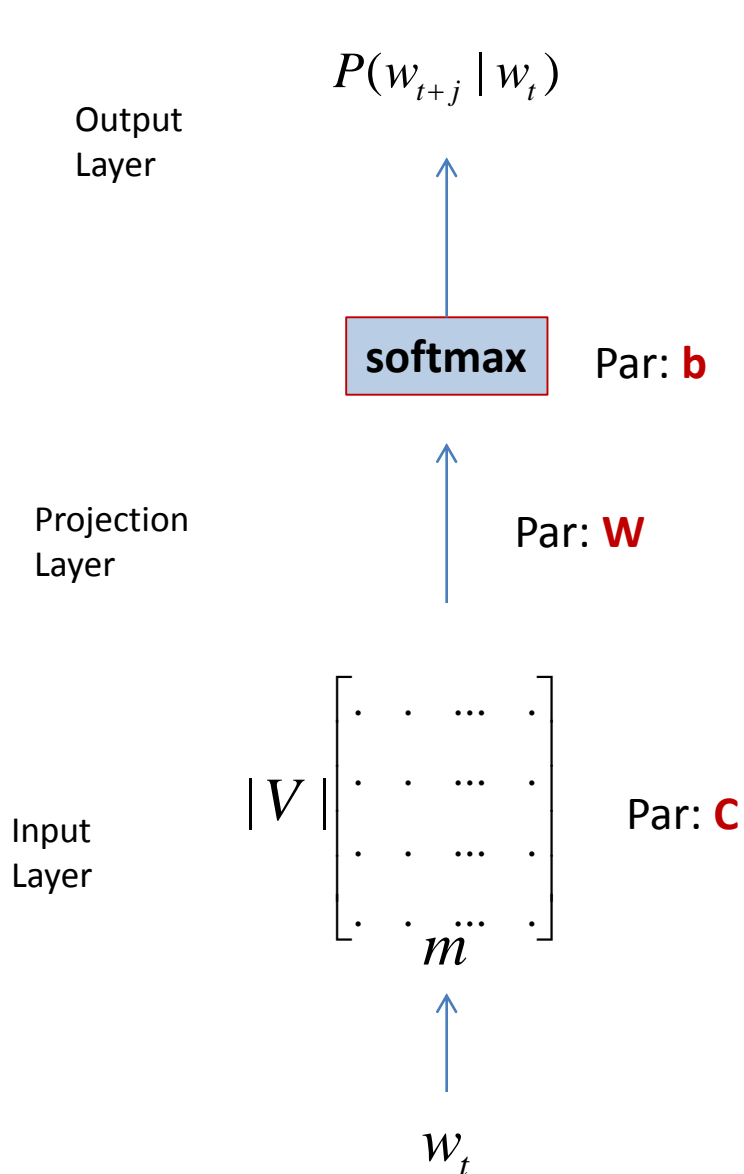
# Skip-Gram

- Instead of predicting the current word based on the context, it tries to maximize classification of a word based on another word in the same sentence.
- Use each current word as an input to a log-linear classifier with continuous projection layer, and predict words within a certain range before and after the current word.



We seek a model for  $P(w_{t+j} | w_t)$

# Skip-Gram



## Feed-Forward Process

$$P(w_{t+j} | w_t) = \frac{\exp(y_{w_{t+j}})}{\sum_{i \in V} \exp(y_i)}$$

$$y_{w_{t+j}} = b + Wx$$

$$x = C(w_t)$$

## Total Parameters:

$$\theta = (b, W, C)$$

$$C \in R^{|V| \times m} \quad b \in R^{|V|}$$

$$W \in R^{|V| \times m}$$

# Skip-Gram

## Training

Training is achieved by looking  $\theta$  that maximizes the following Cost Function:

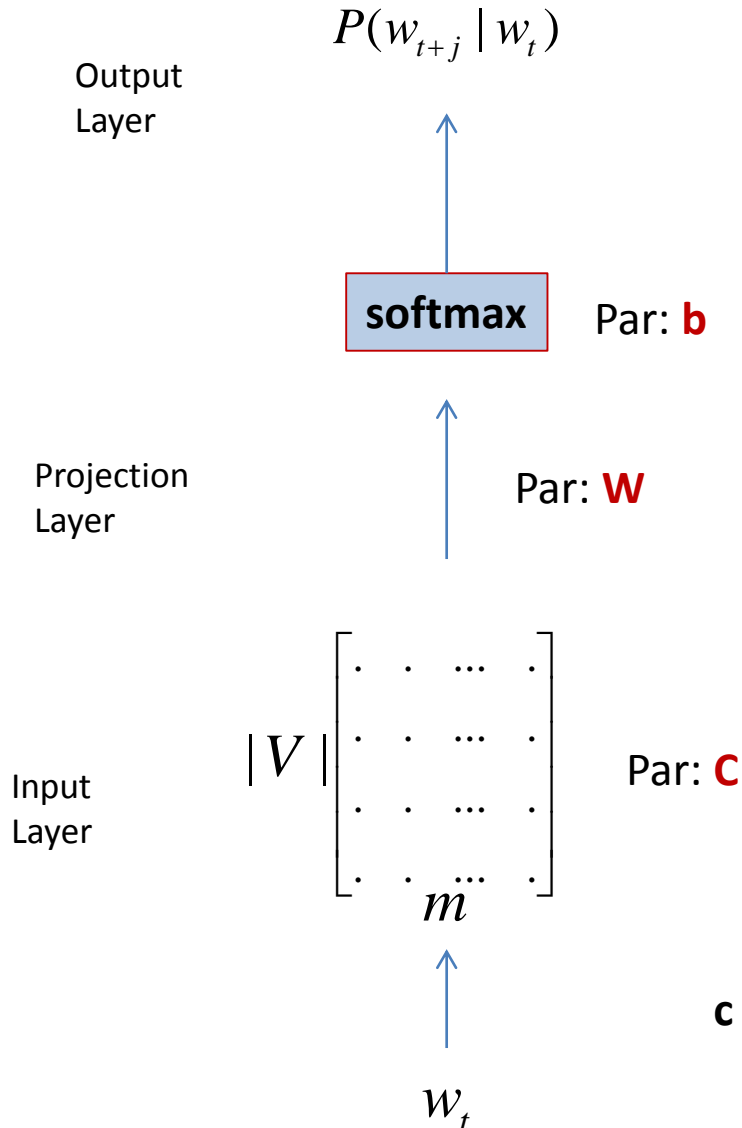
Given training data  $w_1, w_2, w_3, \dots, w_{T-1}, w_T$

$$J(w_1 \dots w_T; \theta)$$

$$= \frac{1}{T} \sum_{t=1}^T \sum_{-c \leq j \leq c, j \neq 0} \log P(w_{t+j} | w_t) + R(\theta)$$

Regularization Terms

$c$  is the maximum distance of the words, or **WINDOW size**



# Skip-Gram

## How to develop dataset?

For example, let's consider the following dataset:

**the quick brown fox jumped over the lazy dog**

Using **c = 1 (or window = 1)**, we then have dataset:

**([the, brown], quick), ([quick, fox], brown), ([brown, jumped], fox), ...**

Therefore, our **(input, output)** dataset becomes:

**(quick, the), (quick, brown), (brown, quick), (brown, fox), ...**

For example,  $P(w_{t+1} = \textit{brown} \mid w_t = \textit{quick})$

Use this dataset to learn  $P(w_{t+j} \mid w_t)$

So that, the cost function is optimized!

$$J(w_1 \dots w_T; \theta) = \frac{1}{T} \sum_{t=1}^T \sum_{-c \leq j \leq c, j \neq 0} \log P(w_{t+j} \mid w_t)$$

# Skip-Gram

## Training

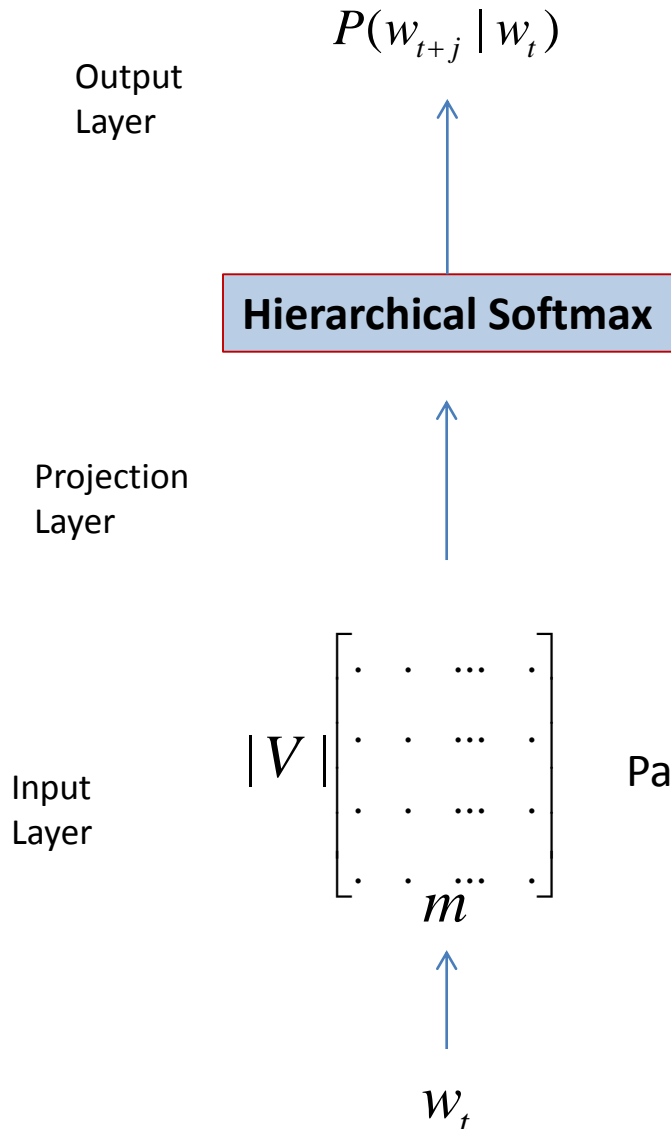
Actually, if we use **vanilla softmax**, then the computational complexity is still costly.

$$Q = C \times (m + m \times |V|)$$

To solve this problem, they use **Hierarchical Softmax layer**. This layer uses a **binary tree representation** of the output layer with  $|V|$  units.

$$Q = C \times (m + m \times \log(|V|))$$

**(Morin & Bengio, 2005)**



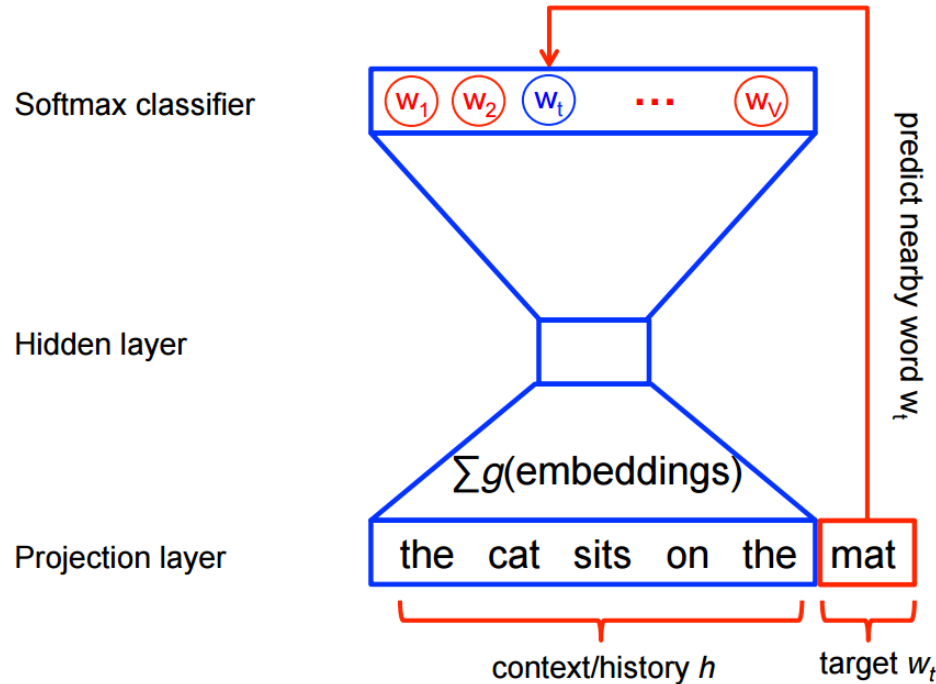
Word2Vec + H-Softmax + Noise Contrastive Estimation (Mikolov et al., 2013)



SOURCE: <https://www.tensorflow.org/tutorials/word2vec>

# Previously

- Standard CBOW and Skip-gram models use **standard flat softmax layer** in the output.
- This is very expensive, because we need to compute and normalize each probability using the score for all other words in the Vocabulary in the current context, at every training step.



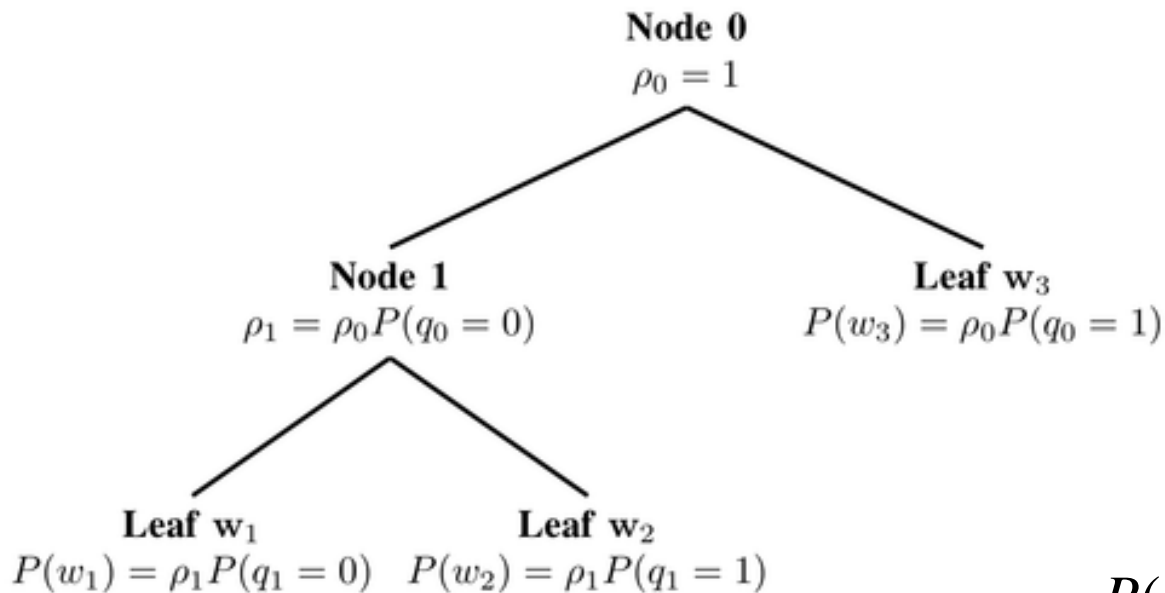
This picture is only CBOW.

# Solution?

- In the first paper, Mikolov et al. (2013) use **Hierarchical Softmax Layer** (Morin & Bengio, 2005).
- **H-Softmax** replaces the flat softmax layer with a hierarchical layer that has the words as leaves.
- This allows us to decompose calculating the probability of one word into a sequence of probability calculations, which saves us from having to calculate the expensive normalization over all words.

# H-Softmax

- We can reason that at a tree's root node (Node 0), the probabilities of branching decisions must sum to 1.
- At each subsequent node, the probability mass is then split among its children, until it eventually ends up at the leaf nodes, i.e. the words.



We can now calculate the probability of going right (or left) at a given node  $\mathbf{n}$  given the context  $\mathbf{c}$ .

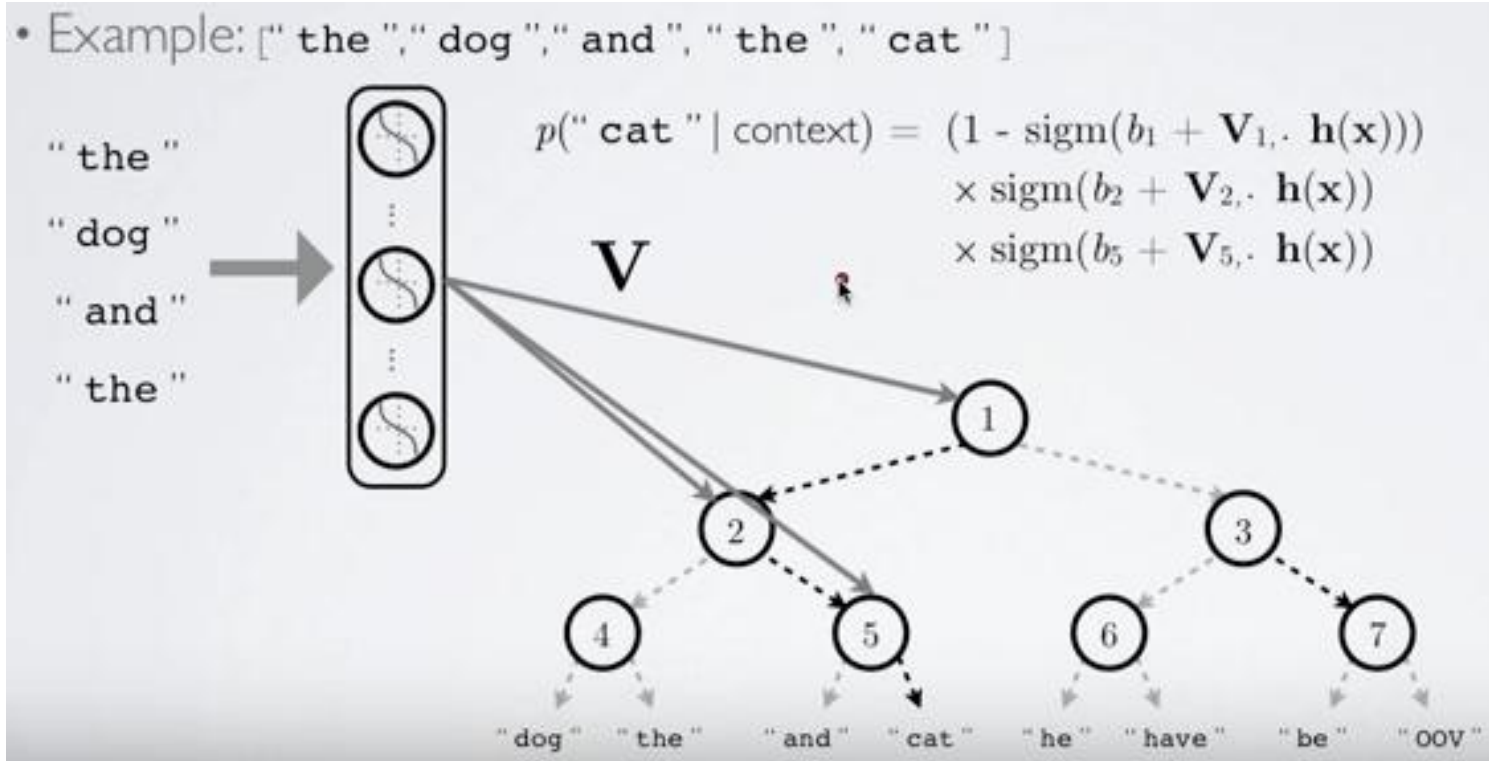
$\mathbf{v}_n$  is embedding in node  $\mathbf{n}$ .

$$P(\text{right} \mid n, \mathbf{c}) = \sigma(W \cdot \mathbf{v}_n)$$

$$P(\text{left} \mid n, \mathbf{c}) = 1 - P(\text{right} \mid n, \mathbf{c})$$

# H-Softmax

- The probability of a word  $w$  given its context  $c$ , is then simply the product of the probabilities of taking right and left turns respectively that lead to its leaf node.

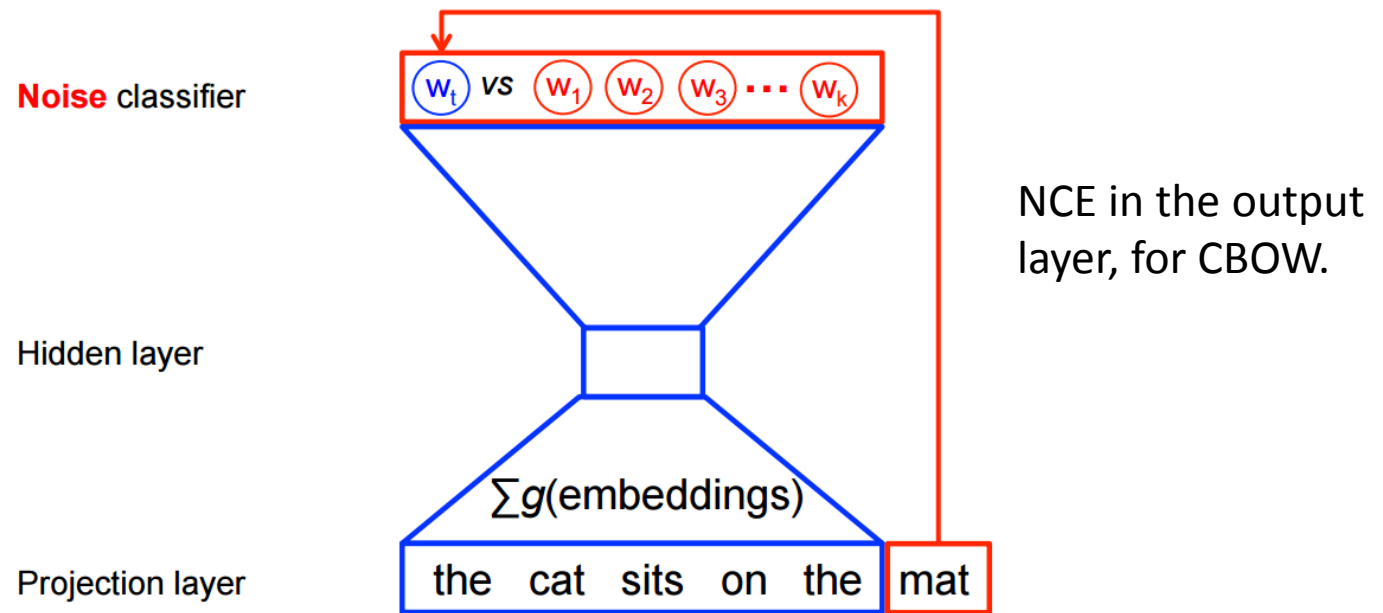


([Hugo Lachorelle's Youtube lectures](#))

SOURCE: <https://www.tensorflow.org/tutorials/word2vec>

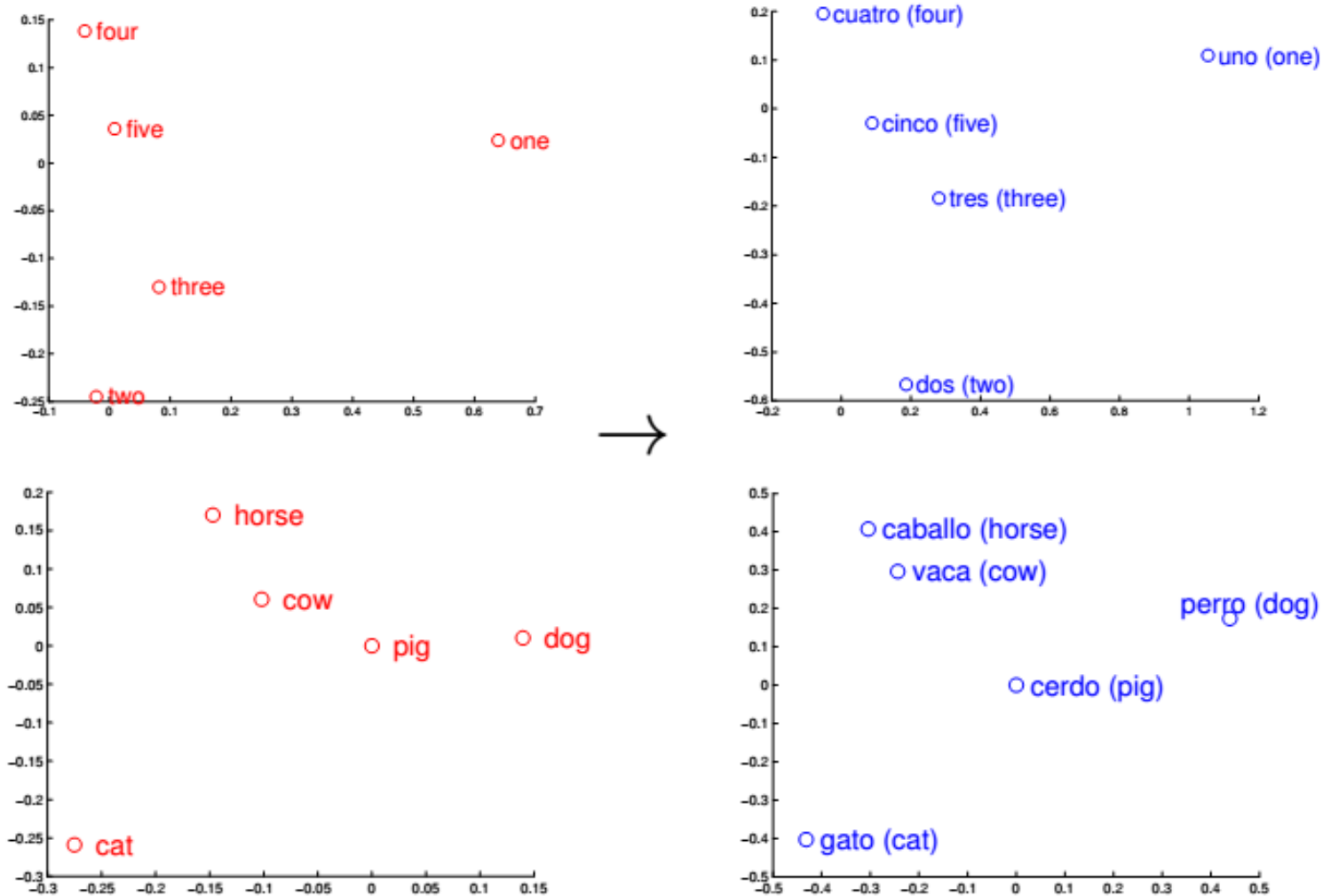
# Noise Contrastive Estimation (NCE)

- In their second paper, the CBOW and skip-gram models are instead trained using a binary classification objective ([logistic regression](#)) to discriminate the real target words  $w_t$  from  $k$  imaginary (noise) words  $w$ .



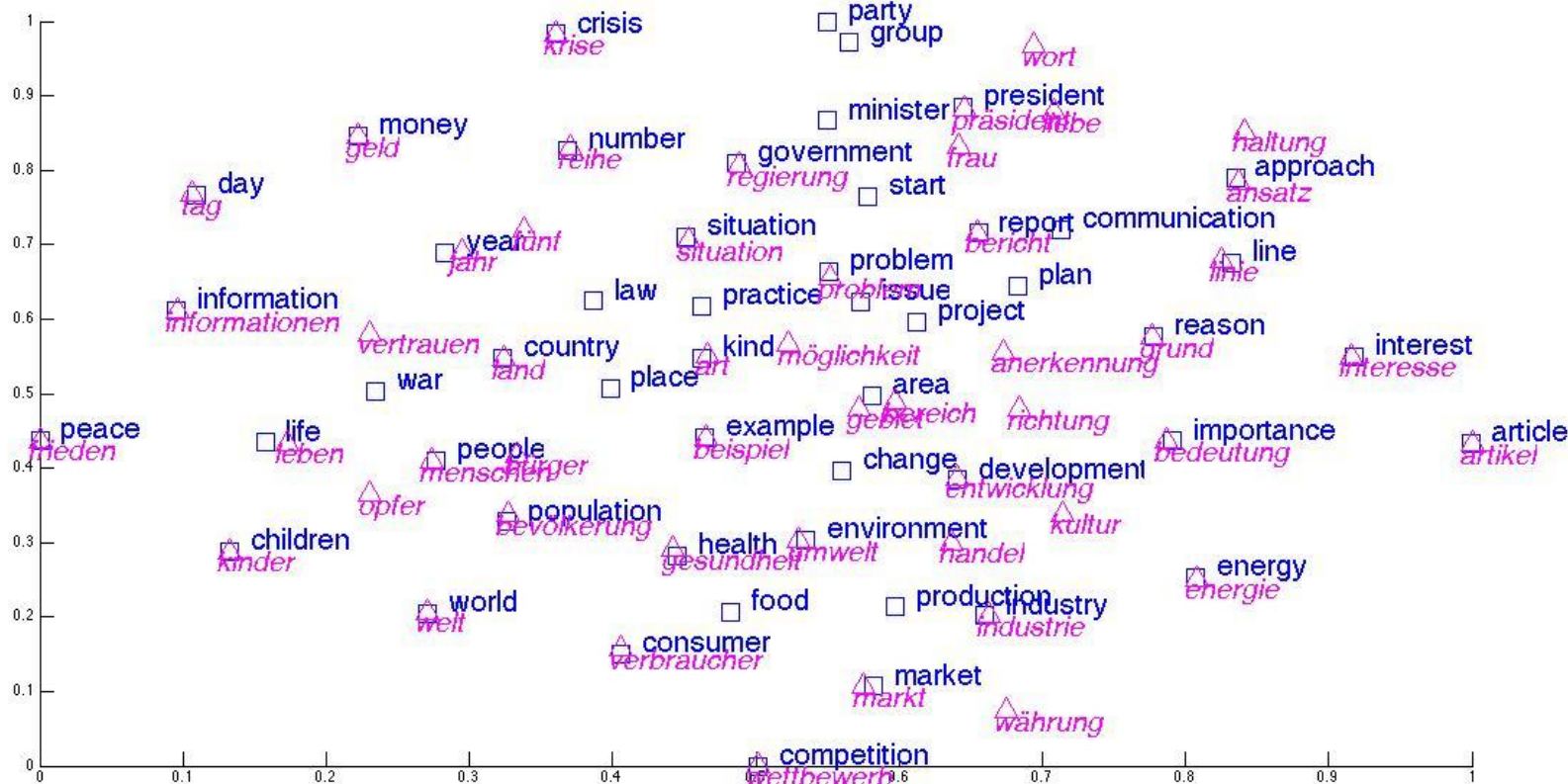
## Cross-Lingual Embedding Model

**Mikolov (2013)** observed that concepts have similar geometric arrangements in both spaces, suggesting that it is possible to learn an accurate linear mapping from one space to another.

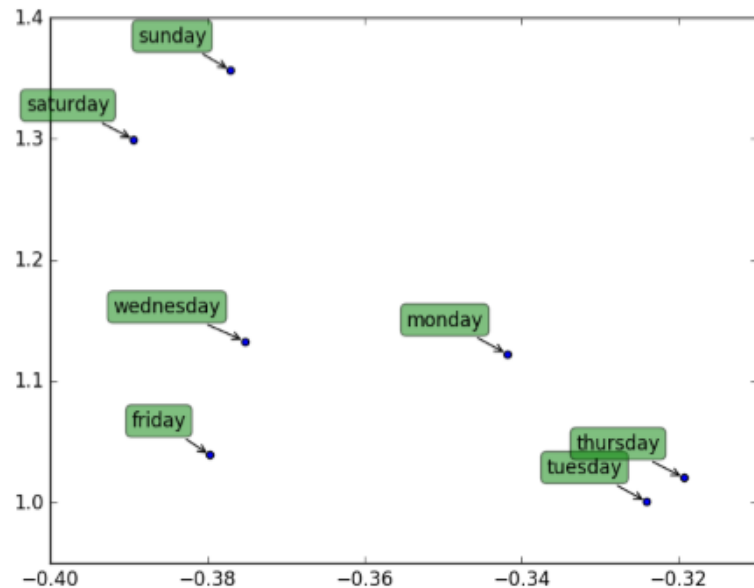
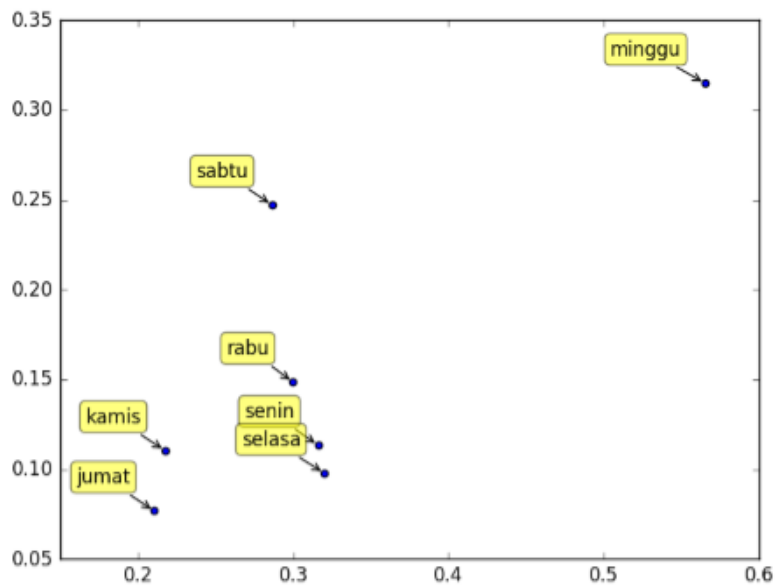
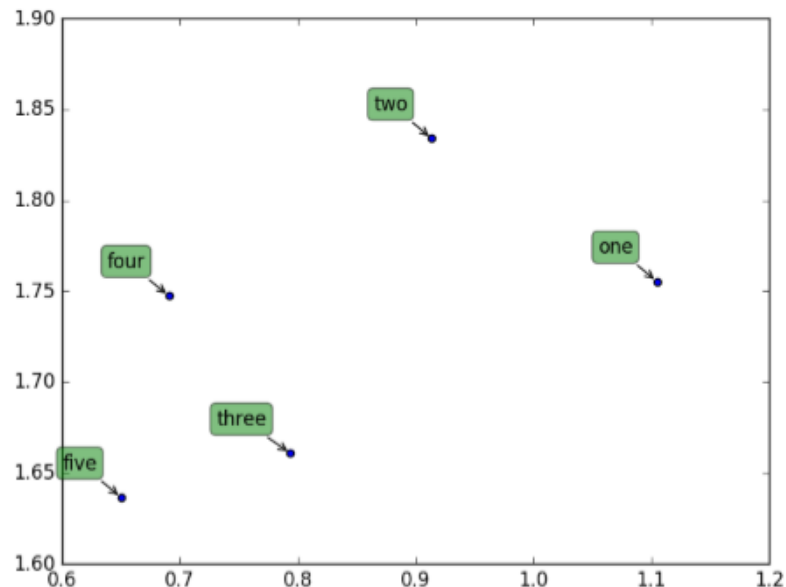
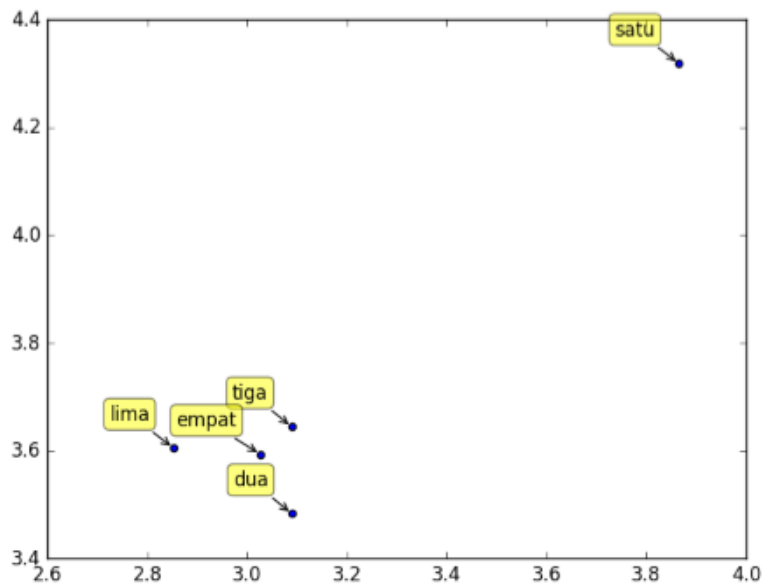


## Cross-Lingual Embedding Model

Ultimately, the goal is to learn a shared embedding space between words in *all* languages.

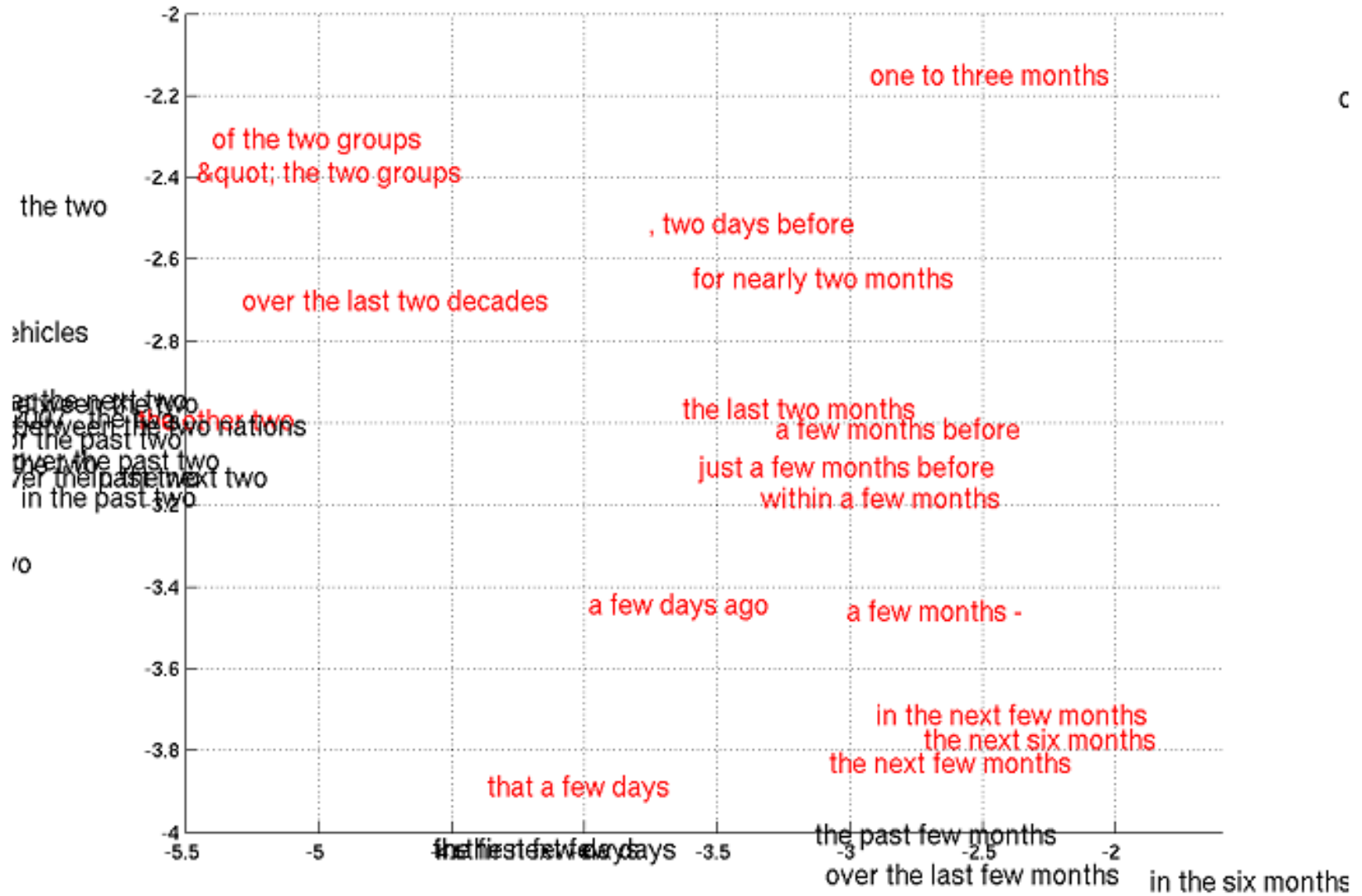


# Simple Observation for Indonesian-English Case; Trained on Comparable Corpus (Wikipedia)





# Learning Phrase Representations



# Learning Phrase Representations

