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

https://www.gavagai.se/blog/2015/09/30/a-brief-history-of-word-embeddings/

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

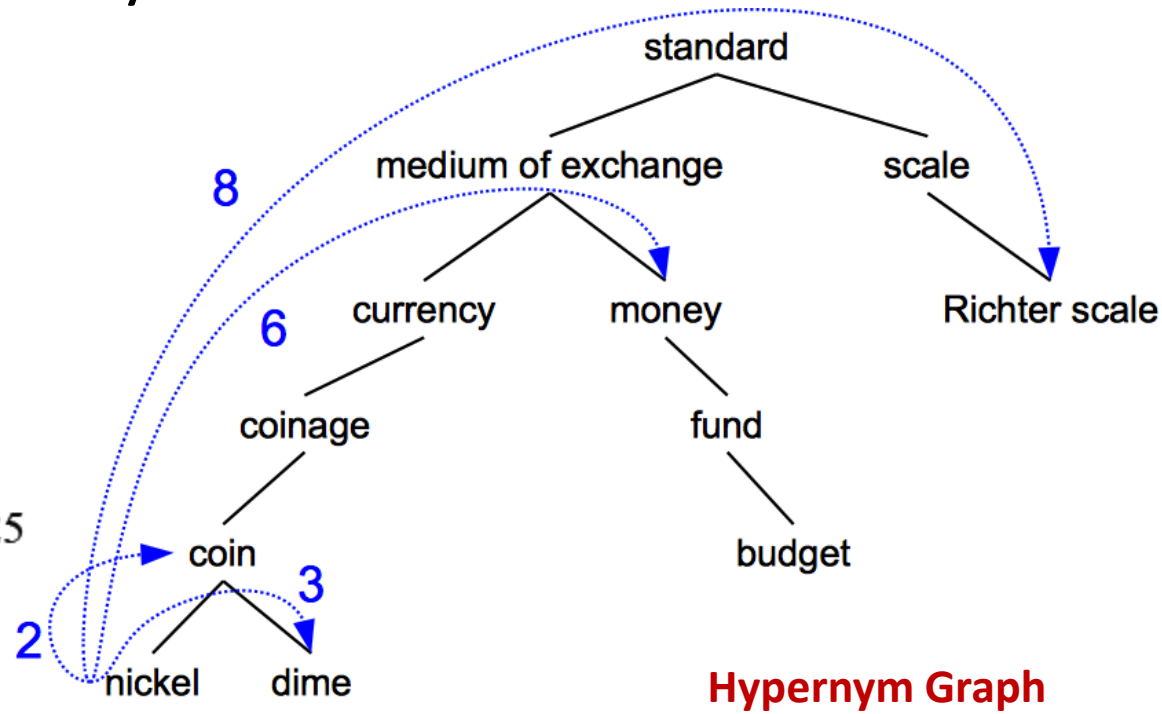
$$\text{simpath}(c_1,c_2) = \frac{1}{\text{pathlen}(c_1,c_2)}$$

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

simpath(*nickel,coin*) = 1/2 = .5

simpath(*fund,budget*) = 1/2 = .5

simpath(*nickel,currency*) = 1/4 = .25



Hypernym Graph

Dan Jurafsky

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 lingustics 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**

Dan Jurafsky

From Dan Jurafsky's course slide: Word Meaning and Similarity

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 your infer something about **"tidak"** and **"nggak"** ?...and why?

Why Word Embeddings?

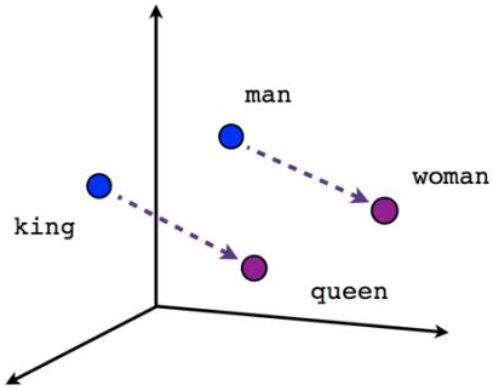
- Word representations are a critical component of many NLP systems. It is common to represent words as indices in a vocabulary, but this fails to capture the rich relational structure of the lexicon.
  - Representing words as unique, discrete ids furthermore leads to data sparsity, and usually means 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.

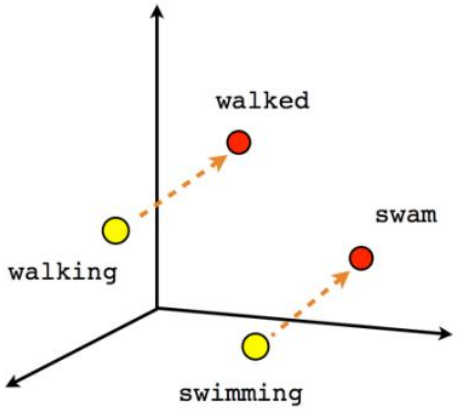
[1] Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y. & Potts, C. Learning word vectors for sentiment analysis. Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1, 2011. Association for Computational Linguistics, 142-150.

Why Word Embeddings?

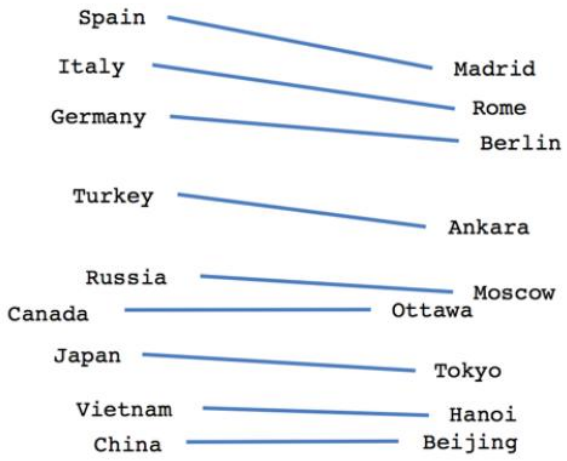
- Can capture the rich relational structure of the lexicon



Male-Female



Verb tense



Country-Capital

https://www.tensorflow.org/tutorials/word2vec

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**.

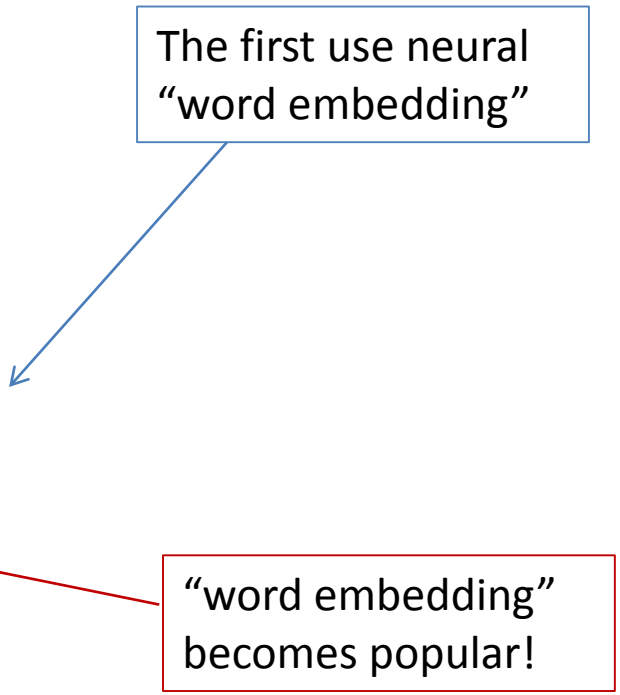
- Vectors whose relative similarities correlate with semantic similarity

- 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)



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)**

Baroni et la., "Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors". ACL 2014

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

https://www.gavagai.se/blog/2015/09/30/a-brief-history-of-word-embeddings/

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 **inflasi** ...
2	... obat membuat sakit **pusing** ...
3	... sakit **pening** di kepala ...
4	... **keuangan** 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.

$$V_{king} - V_{father} + V_{mother} = V_{queen}$$

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. NIPS, 1–9

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.

Nalisnick et al., "Improving Document Ranking with Dual Word Embedding", WWW 2016

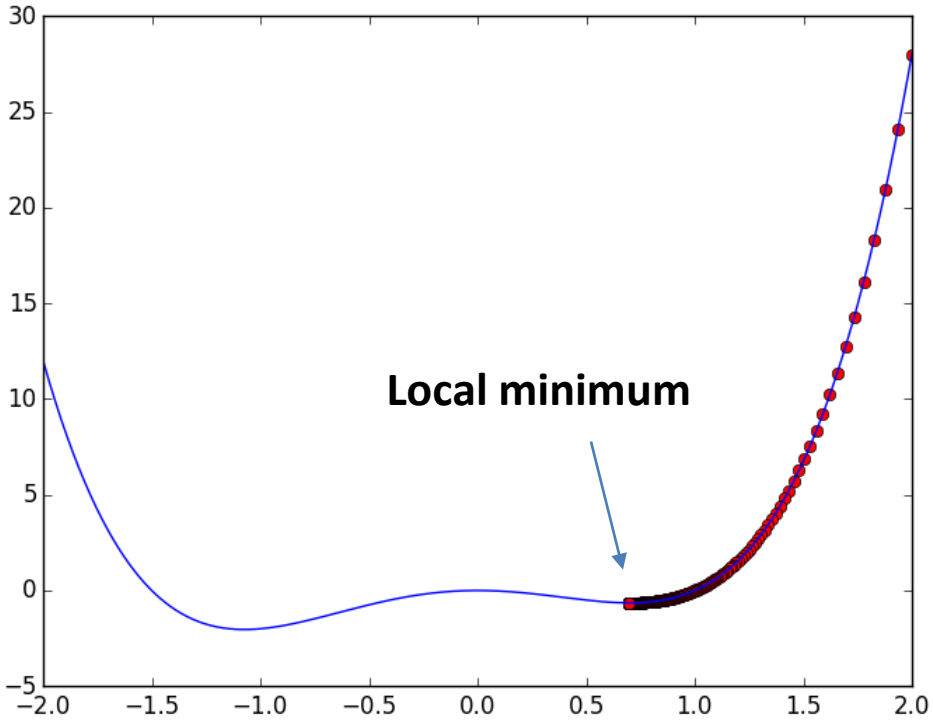
Some basics …

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:

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

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

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

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

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

α_t : ***learning rate*** atau ***step size*** pada iterasi ke-***t***
ϵ: 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 **α_t** yang tidak terlalu kecil, juga tidak terlalu besar.

Gradient Descent (GD)

Kalau parameter-nya ada banyak ?

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

while not converged :

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

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

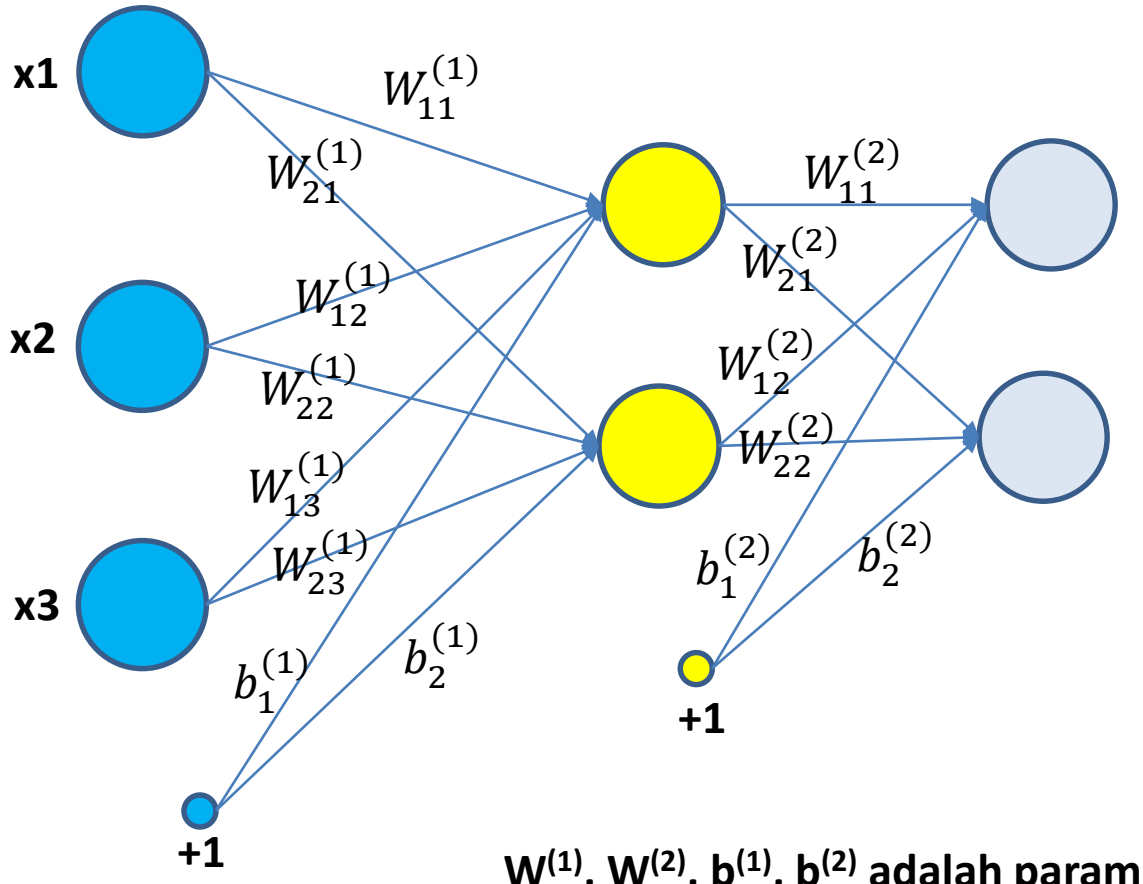
$$\ldots$$

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

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

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:

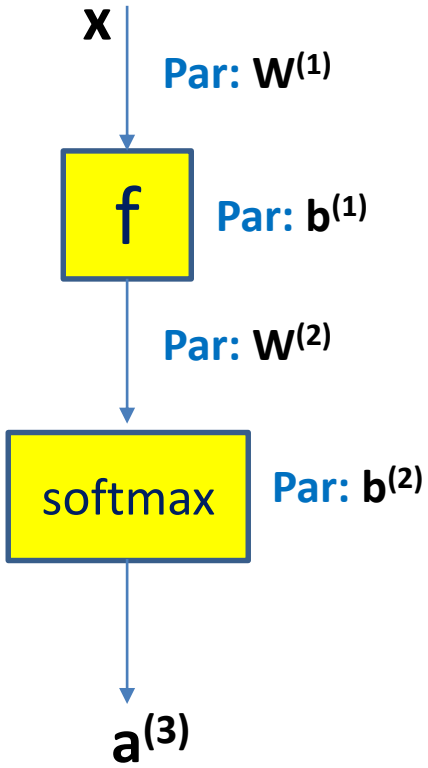
$$z^{(2)} = W^{(1)}x + b^{(1)}$$

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

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

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

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



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:

$$P(w_t \mid w_{t-1} w_{t-2} \ldots w_2 w_1)$$

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

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

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

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):

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

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

$$P(i, learn, word, embedding) =$$
$$P(i \mid < start >)P(learn \mid i)P(word \mid learn)P(embedding \mid word)P(< end > \mid embedding)$$

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

$$\hat{P}(w_t \mid w_{t-1}...w_{t-n+1}) = \frac{count(w_t w_{t-1}...w_{t-n+1})}{count(w_{t-1}...w_{t-n+1})}$$

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 $w_1, w_2, w_3, \ldots, w_{T-1}, w_T$

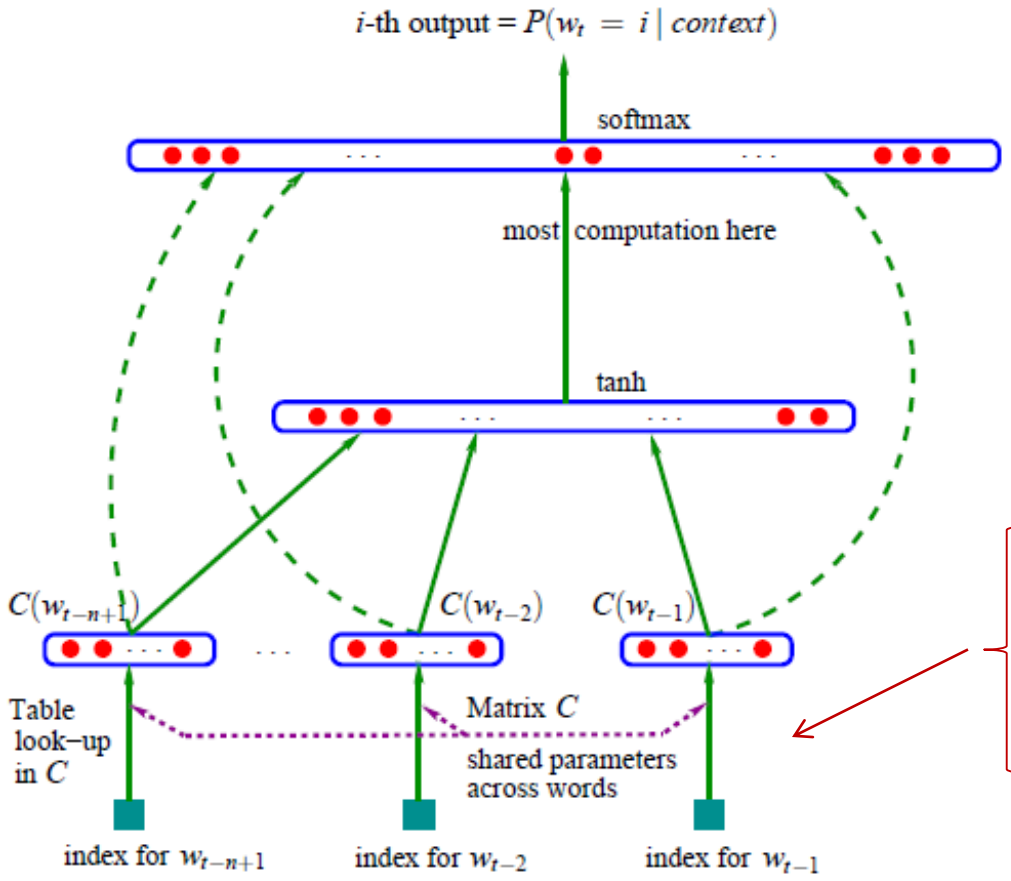
Where each word belongs to a Vocabulary, $w_i \in V$

We want to learn good model for:

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

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

Neural Language Model



We have **Embedding/Projection Matrix**

$$C_{|V| \times m}$$

$$|V| \left\{ \begin{bmatrix} \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \end{bmatrix} \right. \quad m$$

m is **size** of **word vector**.

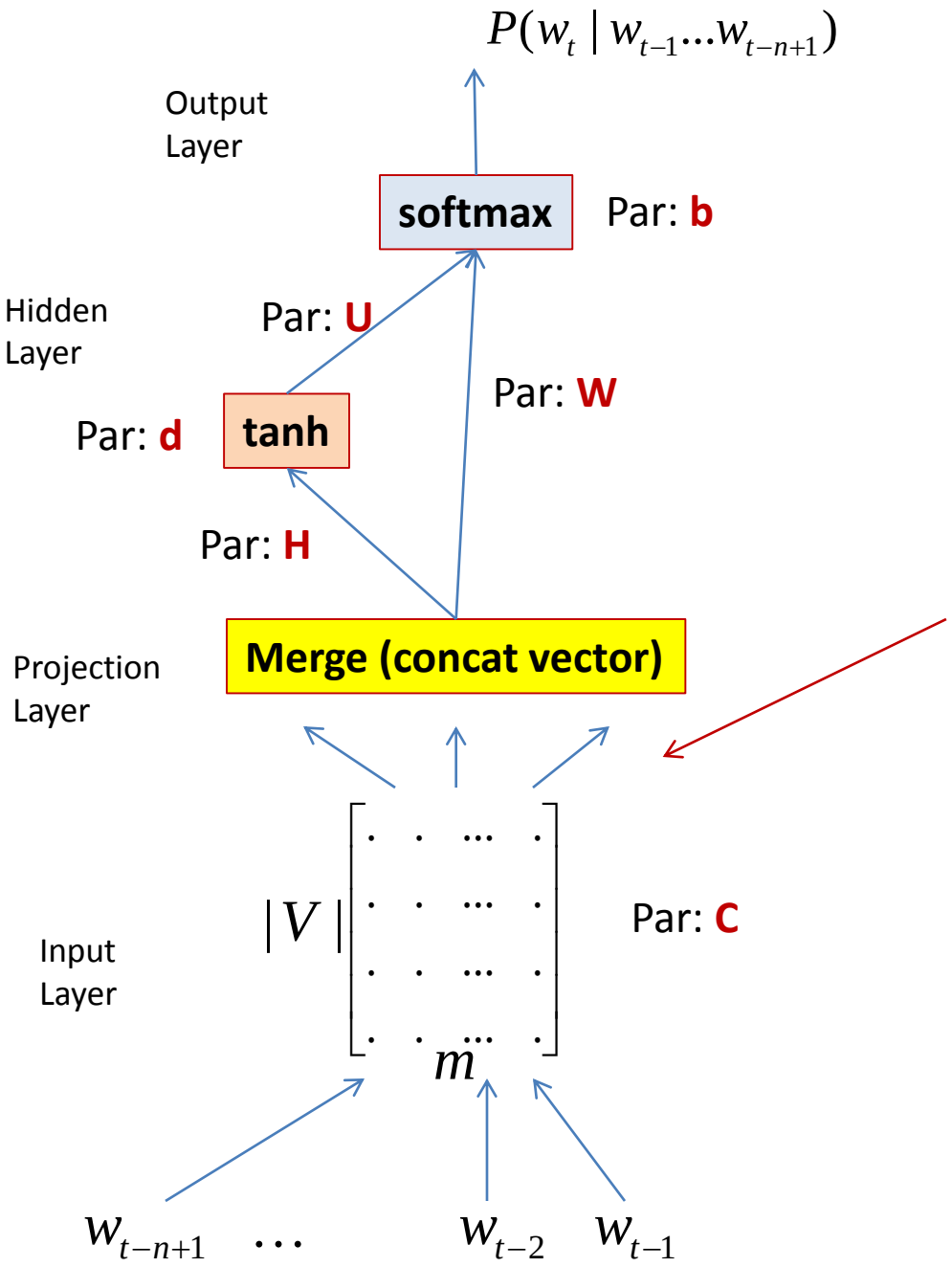
$$C(i) \in R^m$$

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

$$f(i, w_{t-1}, \ldots, w_{t-n+1}) = P(w_t = i \mid context)$$
$$= g(i, C(w_{t-1}), \ldots, C(w_{t-n+1}))$$

Score for a particular output:

g is a neural network function.



Feed-Forward Process

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

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

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

Total Parameters:

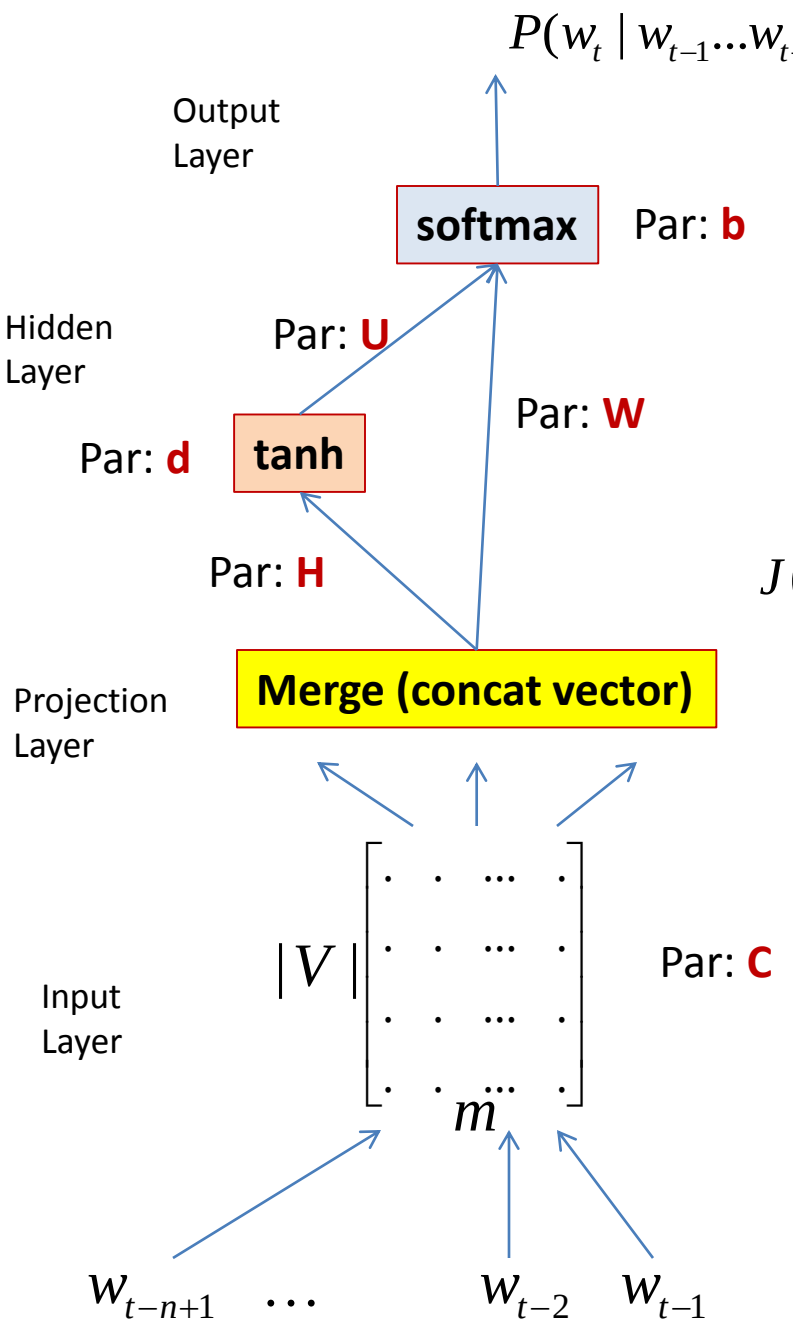
$$\theta = (b, d, W, U, H, C)$$

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

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

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

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



Training

Training is achieved by looking **θ** that maximizes the following Cost Function:

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

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

Regularization terms

How? We can use Gradient Ascent!

Iterative update using the following formula:

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

Learning rate

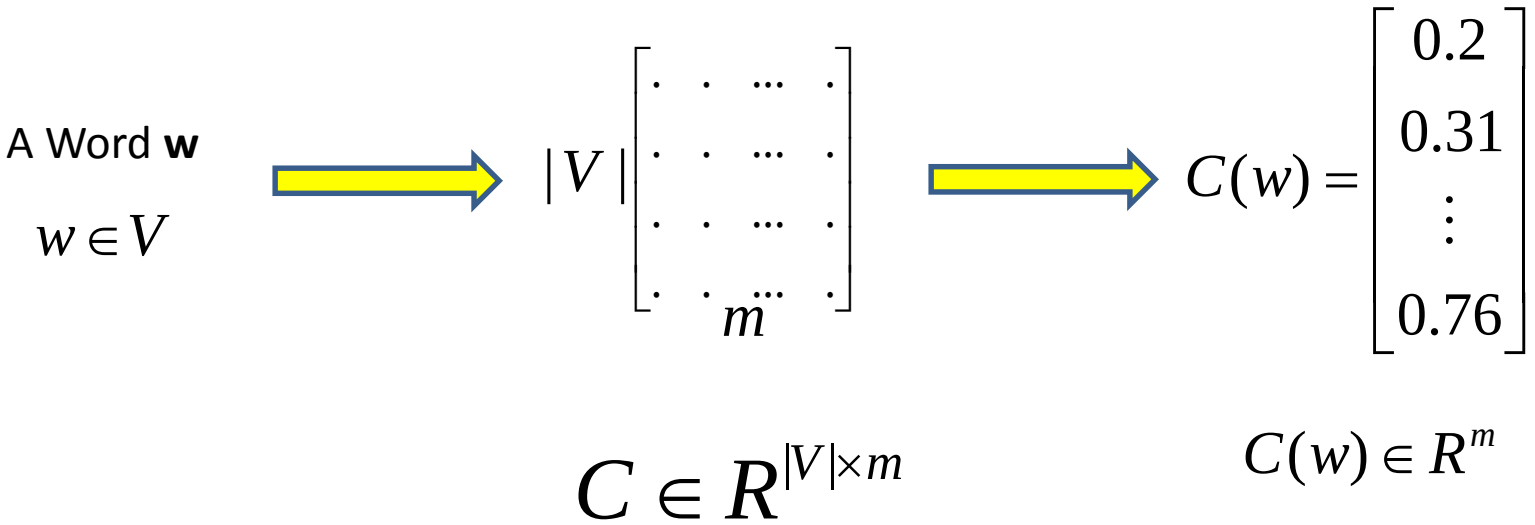
Where are the Word Embeddings?

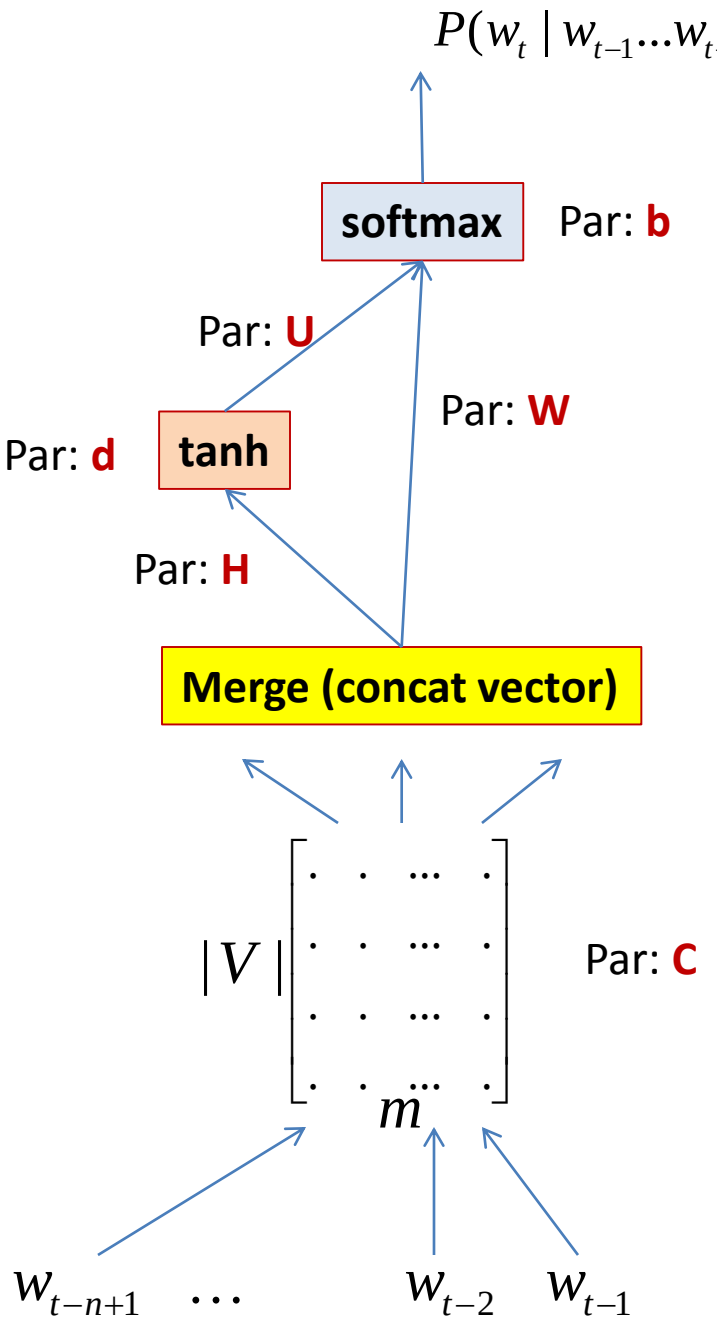
- The previous model actually aims at building the language model.
  - So, **where is the Word Embedding model that we need?**

- **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 **C**) are optimized, then we can use **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 **X** size of vector m

Between projection layer & hidden layer

Dominating Term!

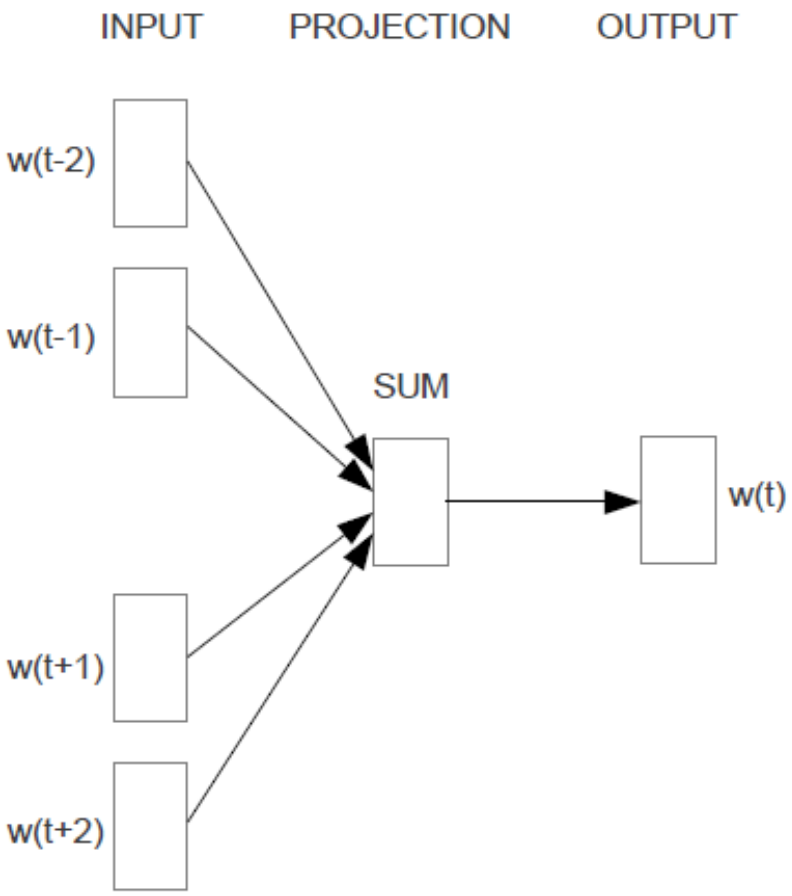
Word2Vec (Mikolov et al., 2013)

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

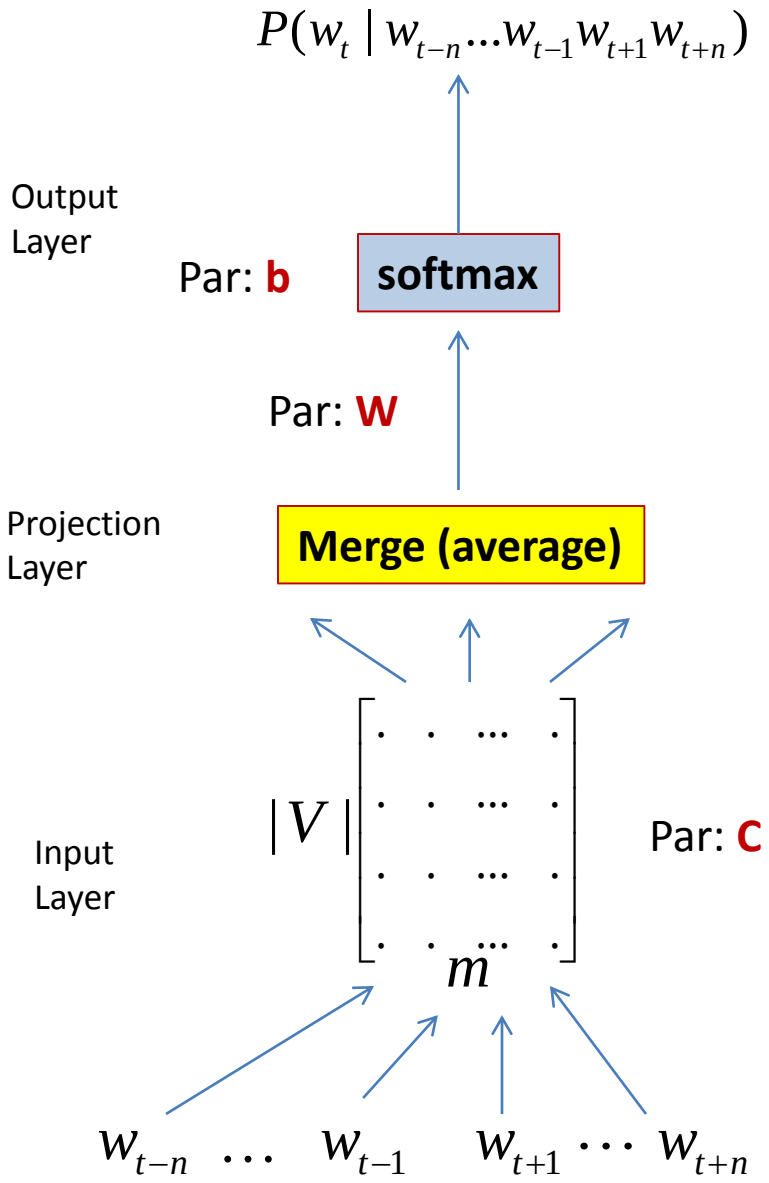
Continuous Bag-of-Words

- **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 $P(w_t \mid w_{t-n} \ldots w_{t-1} w_{t+1} \ldots w_{t+n})$

Continuous Bag-of-Words



Feed-Forward Process

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

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

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

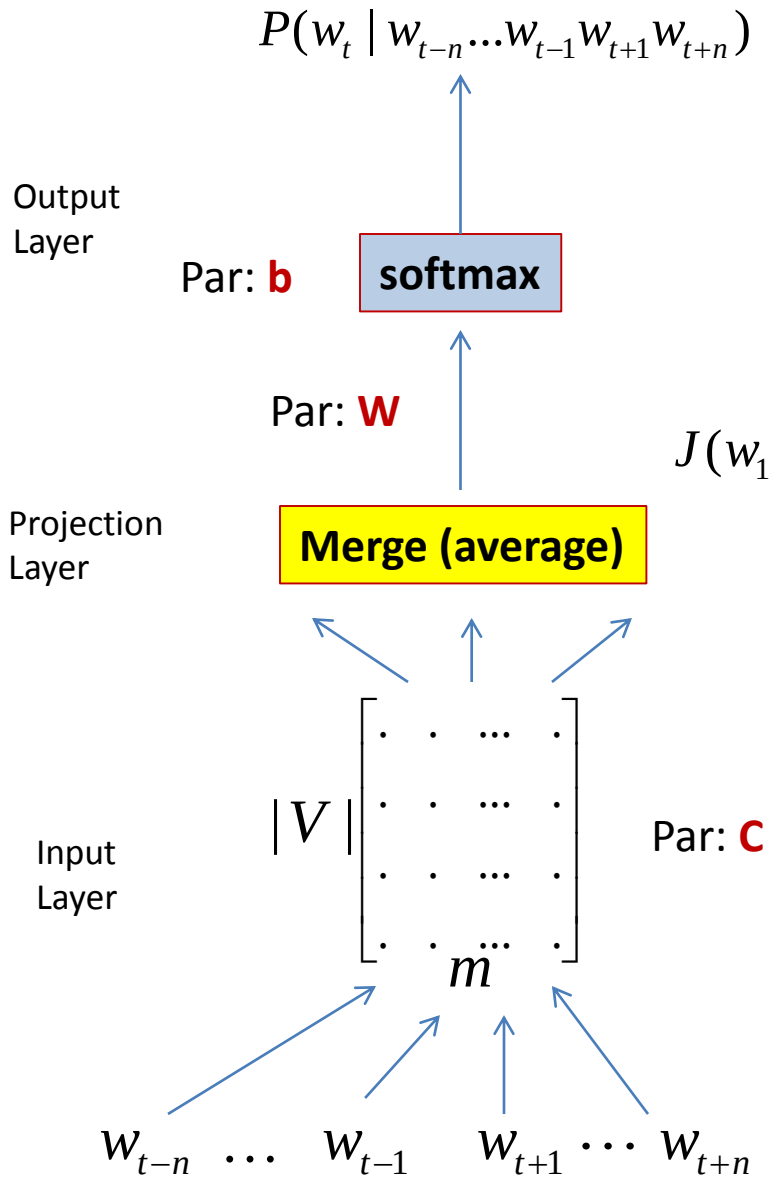
Total Parameters:

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

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

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

Continuous Bag-of-Words



Training

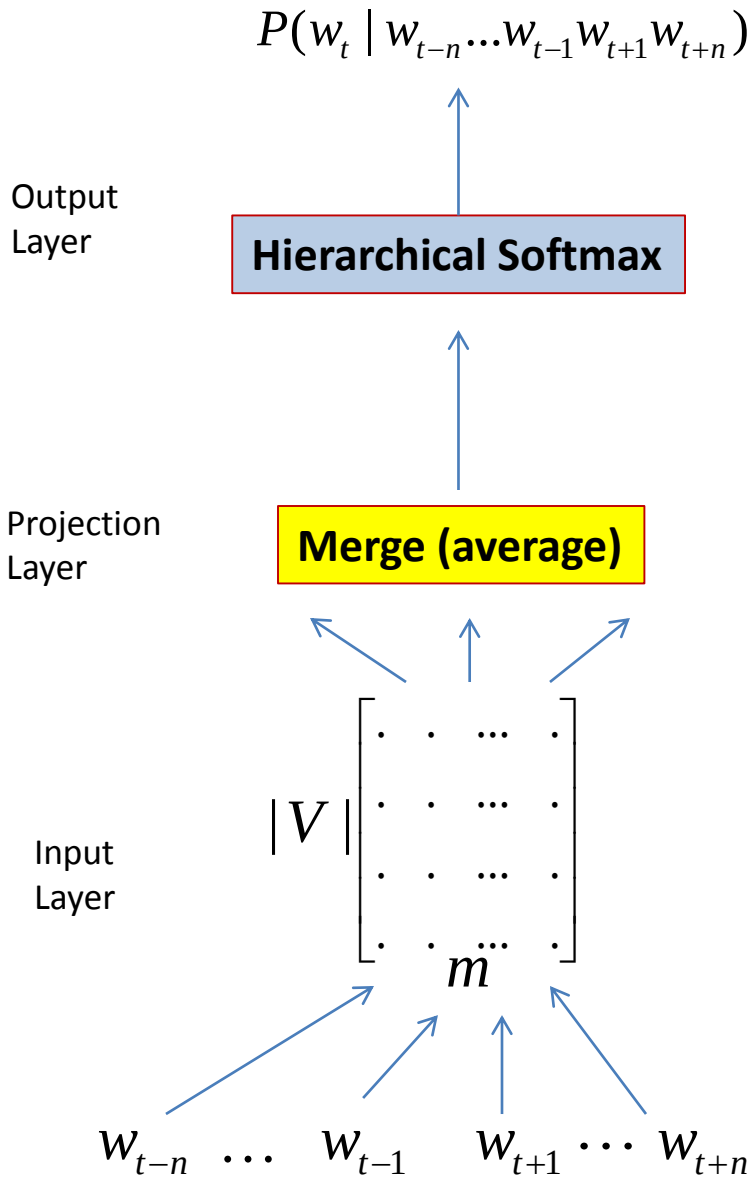
Training is achieved by looking **θ** that maximizes the following Cost Function:

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

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

Regularization terms

Continuous Bag-of-Words



Training

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

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

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

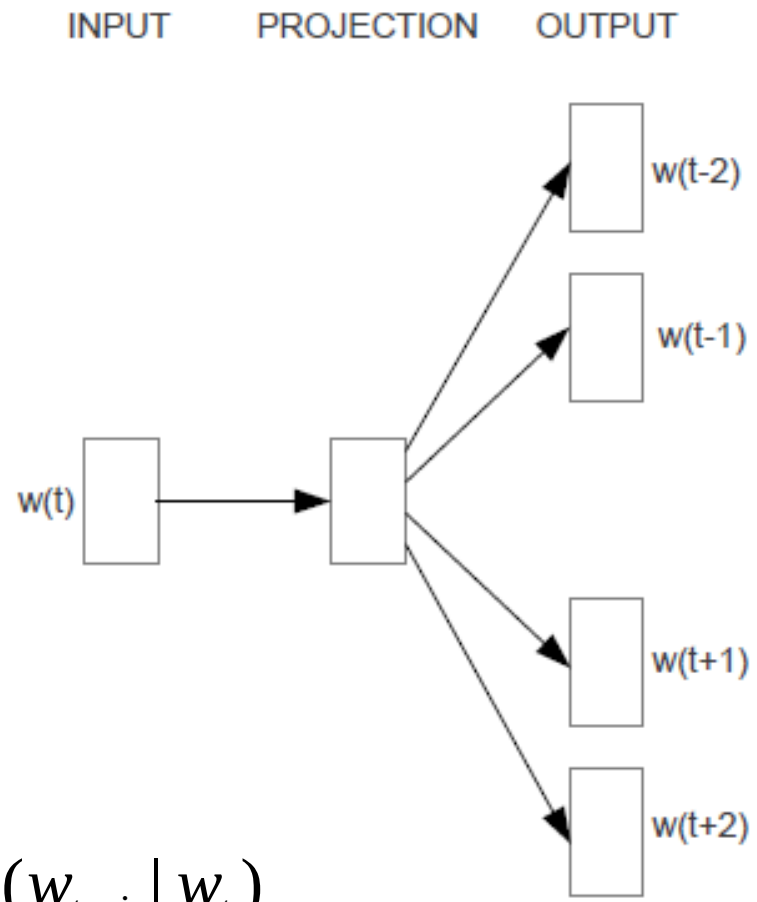
$$Q = (2N \times m) + (m \times^2 \log(|V|))$$

(Morin & Bengio, 2005)

Skip-Gram

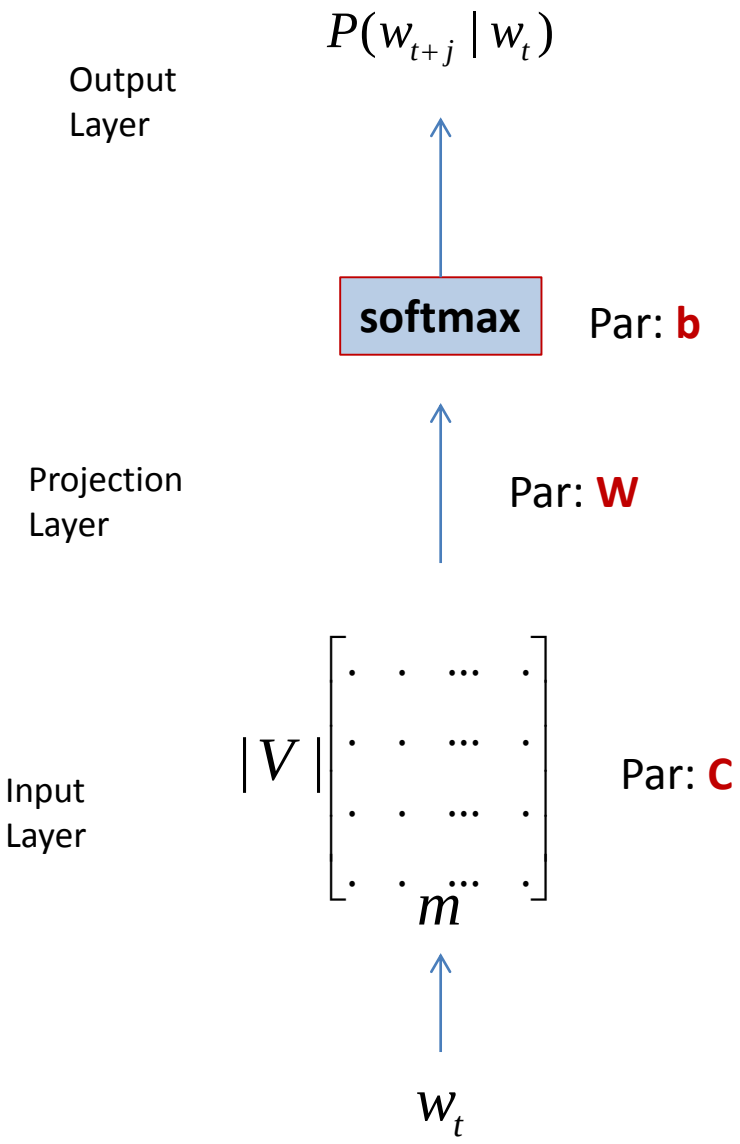
- Instead of predicting the current word based on the context, it tries to maximize classification of a word based on another word in the same sentence.

- 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 $P(w_{t+j} \mid w_t)$

Skip-Gram



Feed-Forward Process

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

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

$$x = C(w_t)$$

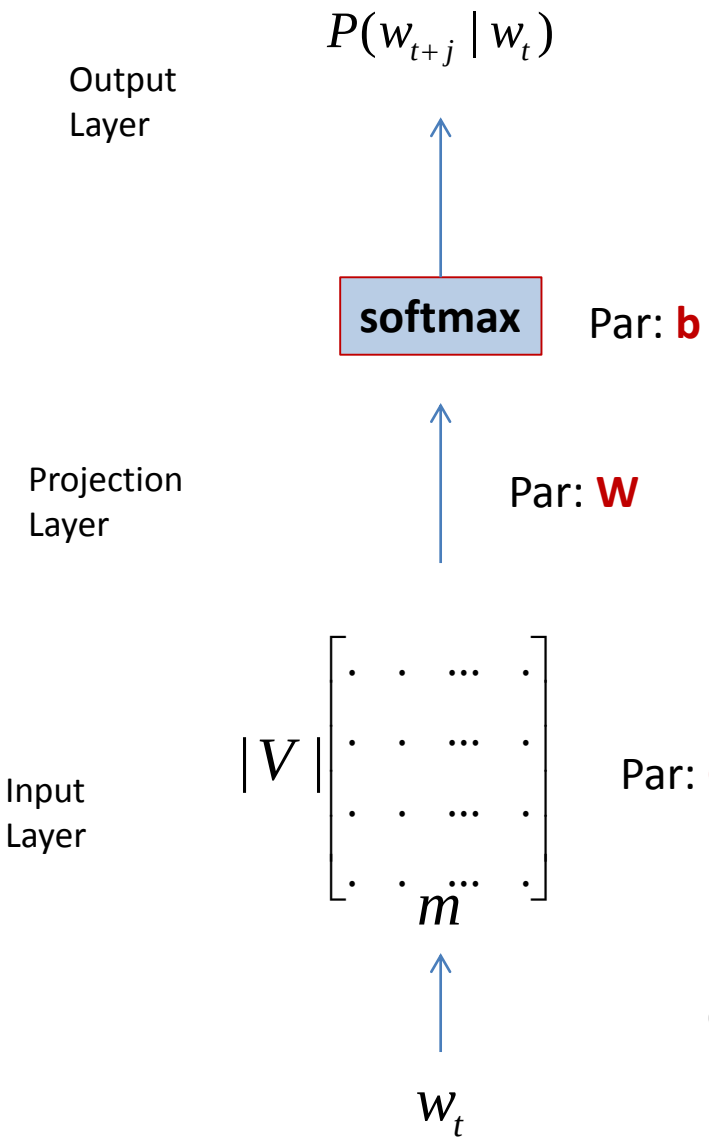
Total Parameters:

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

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

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

Skip-Gram



Training

Training is achieved by looking **θ** that maximizes the following Cost Function:

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

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

Regularization Terms

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

Reference: https://www.tensorflow.org/tutorials/word2vec

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), ...
```

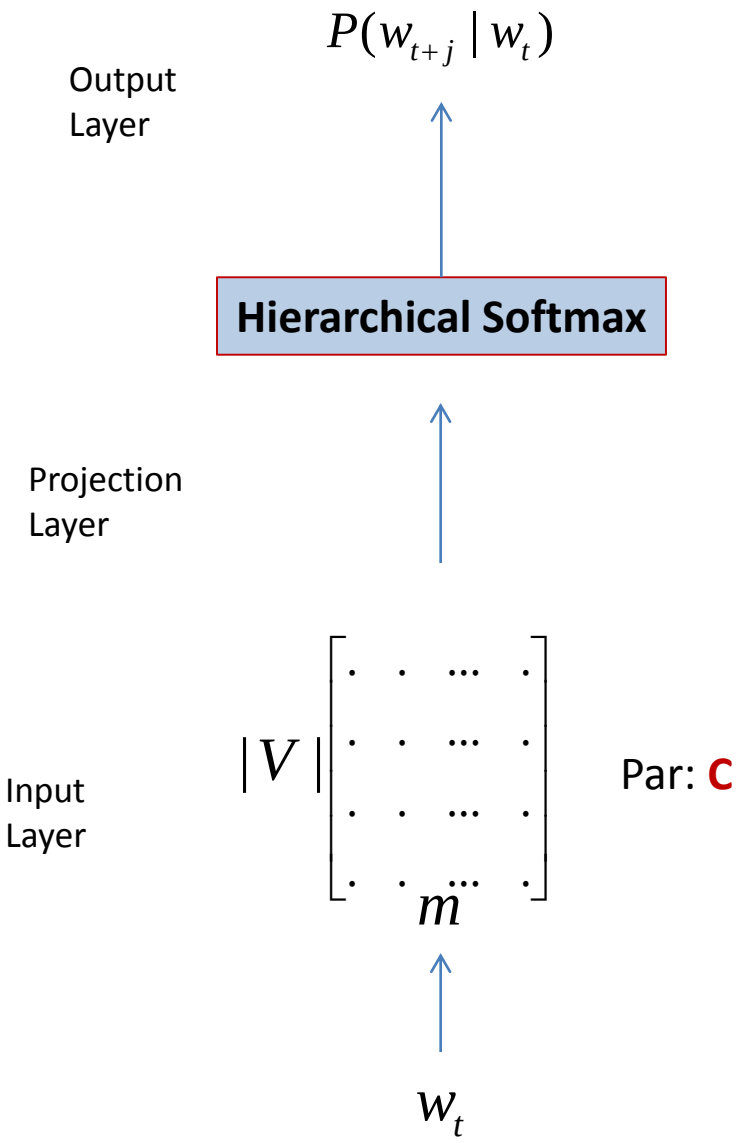
For example, $P(w_{t+1} = brown \mid w_t = quick)$

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

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

So that, the cost function is optimized!

Skip-Gram



Par: **C**

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.

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

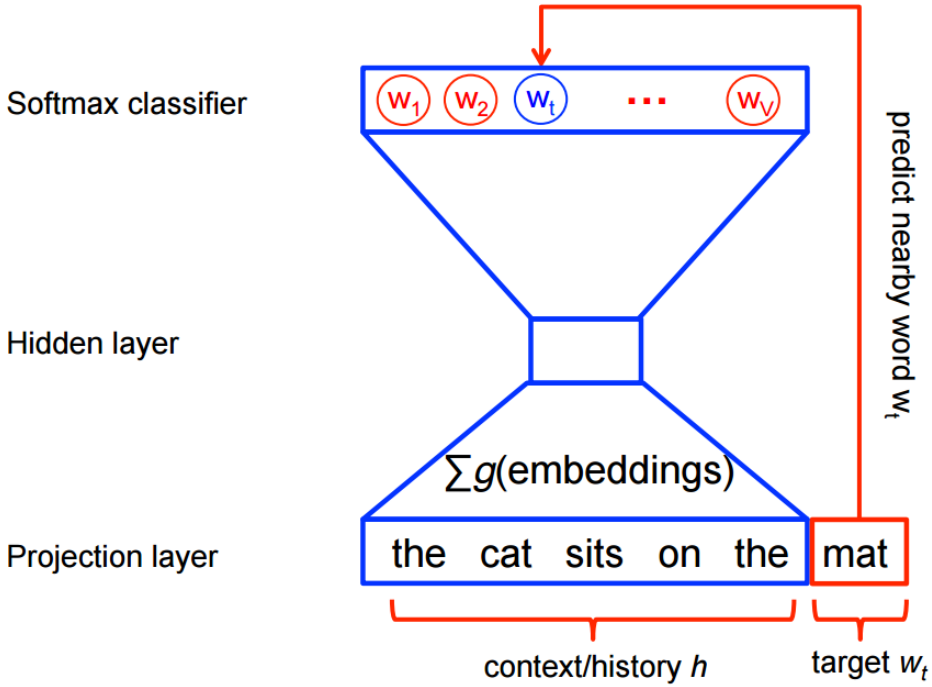
(Morin & Bengio, 2005)

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?

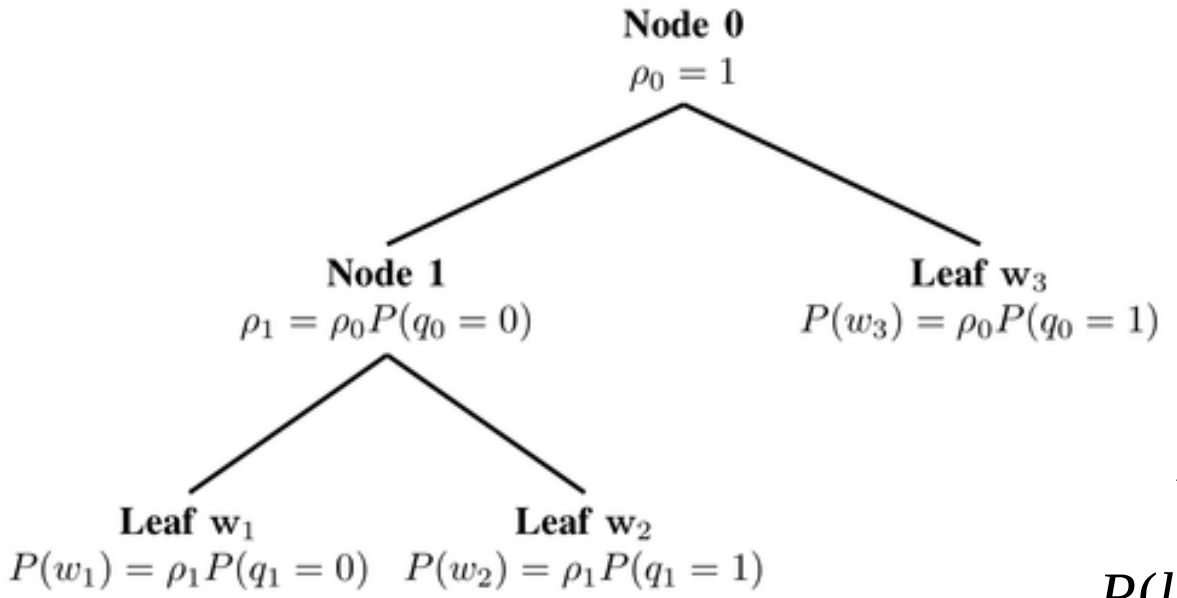
- In the first paper, Mikolov et al. (2013) use **Hierarchical Softmax Layer** (Morin & Bengio, 2005).
- **H-Softmax** replaces the flat softmax layer with a hierarchical layer that has the words as leaves.

- 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.

Sebastian Ruder. On word embeddings - Part 2: Approximating the Softmax. http://sebastianruder.com/word-embeddings-softmax

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 **n** given the context **c**.

$\mathbf{v_n}$ is embedding in node **n**.

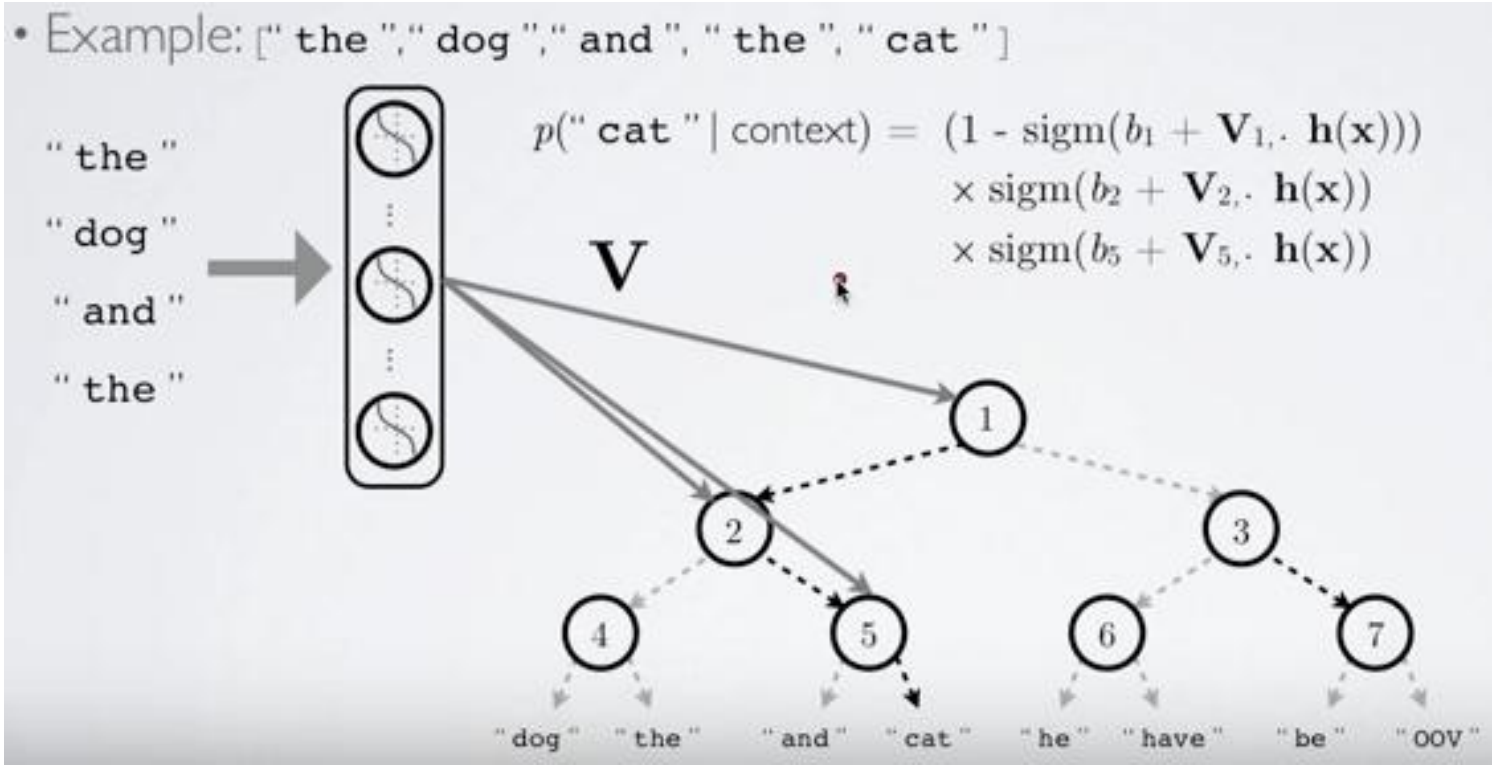
$$P(right \mid n, c) = \sigma(W.v_n)$$

$$P(left \mid n, c) = 1 - P(right \mid n, c)$$

Sebastian Ruder. On word embeddings - Part 2: Approximating the Softmax. http://sebastianruder.com/word-embeddings-softmax

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)

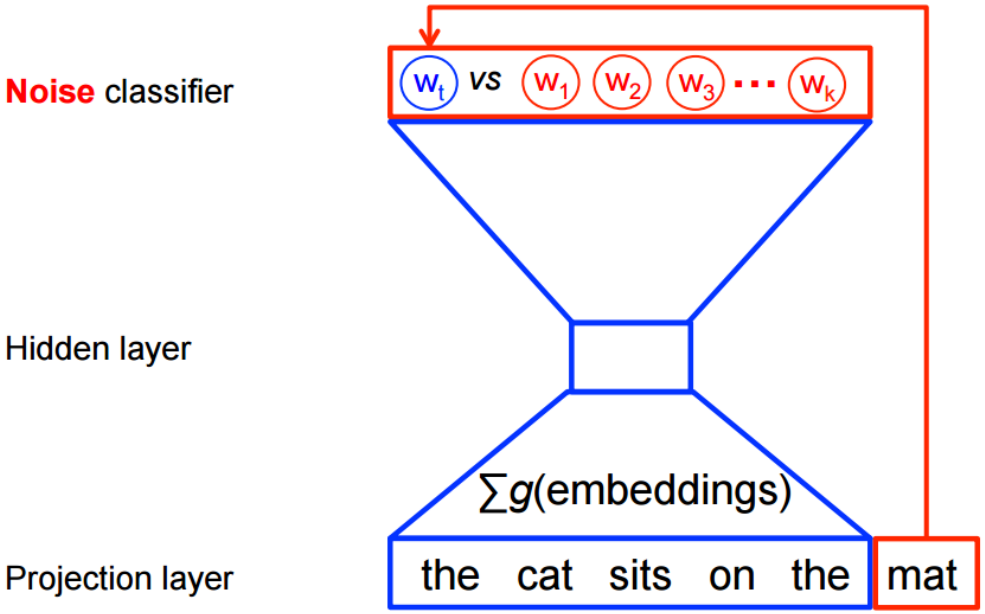
https://www.youtube.com/watch?v=B95LTf2rVWM

Sebastian Ruder. On word embeddings - Part 2: Approximating the Softmax. http://sebastianruder.com/word-embeddings-softmax

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 $\mathbf{w_t}$ from **k** imaginary (noise) words **w**.



NCE in the output layer, for CBOW.