Deep Learning for Natural Language Processing

Dr. Minlie Huang (黄民烈), Asso/Prof.

aihuang@tsinghua.edu.cn

Computer Science Department

Tsinghua University

Homepage: http://coai.cs.tsinghua.edu.cn/hml

My group: http://coai.cs.tsinghua.edu.cn

Outline

- Background
- Word Vectors
- Recursive Neural Network
- Recurrent Neural Network
- Convolution Neural Network

Natural Language Processing

- **Natural language processing** (**NLP**) is a field of computer science, artificial intelligence and computational linguistics concerned with the interactions between computers and human (natural) languages, and, in particular, concerned with programming computers to fruitfully process large natural language corpora.

- Involve natural language understanding, natural language generation (frequently from formal, machine-readable logical forms), connecting language and machine perception, managing human-computer dialog systems, or some combination thereof.

From https://en.wikipedia.org/wiki/Natural_language_processing

Background: Natural Language Processing

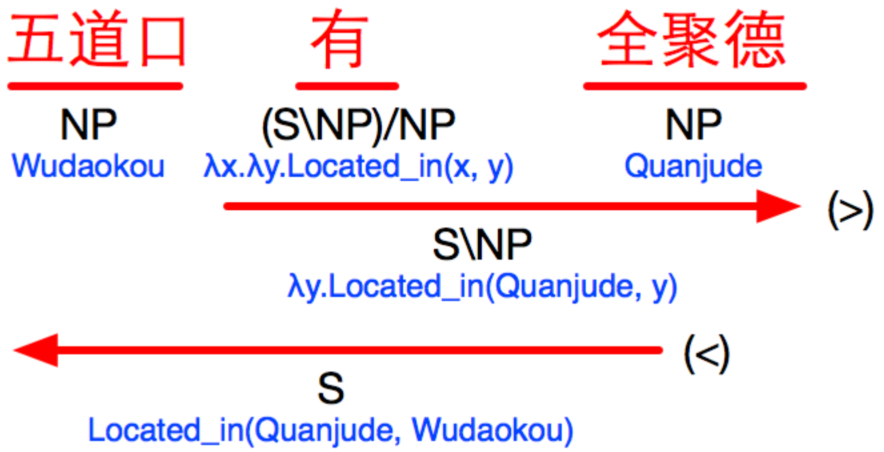
- Tagging and parsing
- Question answering and dialogue systems
- Text/document classification
- Sentiment analysis and opinion mining
- Machine translation

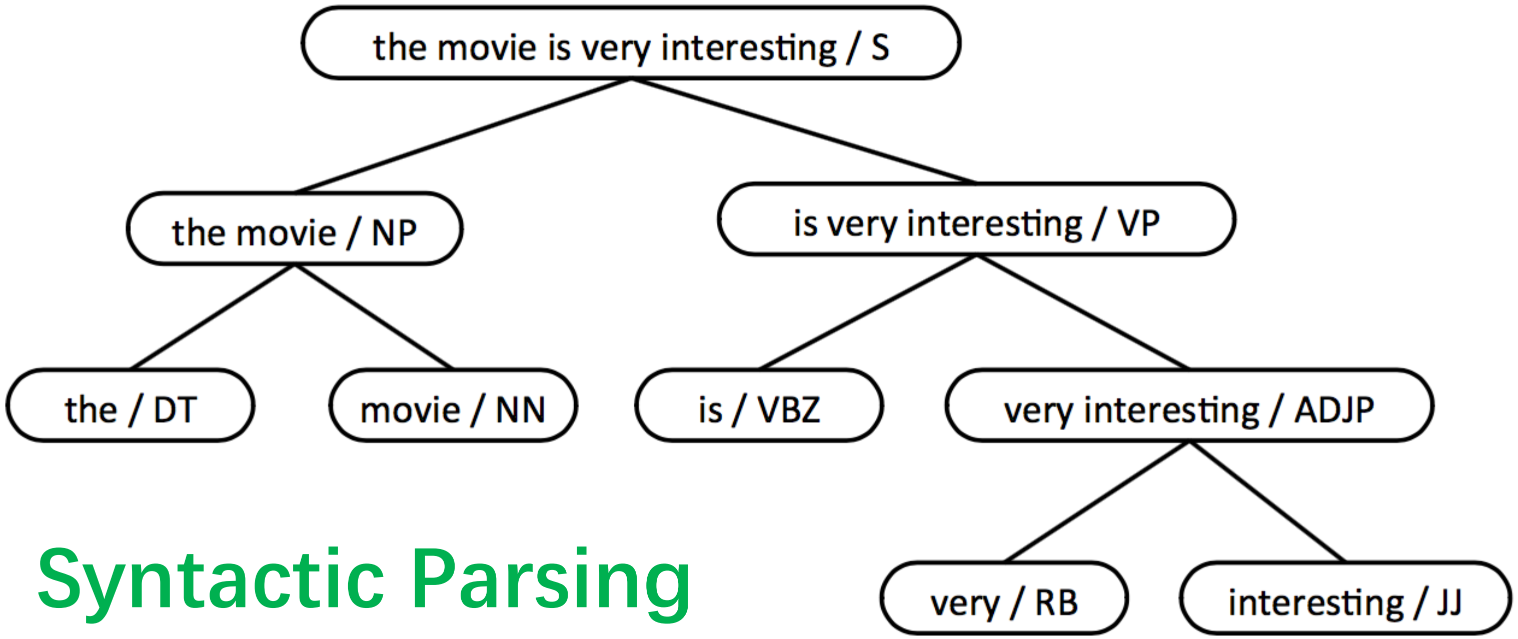
Tagging and Parsing

Part-of-speech Tagging

The	movie	is	very	interesting
DT	NN	VBZ	RB	JJ

Semantic Parsing





Syntactic Parsing

Question Answering and Dialogue Systems

Q: How many people live in Beijing?

A: *About* ***22 million****.*

Q: How long is the Great Wall?

A: About ***8800 km****.*

Q: What creates a cloud

A: *in meteorology , a cloud is* ***a visible mass of liquid droplets*** *or frozen crystals made of water or various chemicals suspended in the atmosphere above the surface of a planetary body.*

x_0	*good morning*
y_0	*hello what can i help you with today*
x_1	*can you book a table in a cheap price range in london*
y_1	*i'm on it*
x_2	*<SILENCE>*
y_2	*any preference on a type of cuisine*
x_3	*with french food*
y_3	*how many people would be in your party*
x_4	*for four please*
y_4	*ok let me look into some options for you*
x_5	*<SILENCE>*
y_5	*api_call french london four cheap*

IBM DeepQA: Understanding Human Languages

Sentiment Analysis and Opinion Mining

Sentiment analysis (SA) or opinion mining
computational study of opinion, sentiment, appraisal, evaluation, and emotion.

- *Sentiment*: an attitude, thought, or judgment prompted by feeling.
  - A sentiment is more of a feeling.
  - "*I am concerned about the current state of the economy.*"
- *Opinion:* a view, judgment, or appraisal formed in the mind about a particular matter.
  - a concrete view of a person about something.
  - "*I think the economy is not doing well.*"

Slides from Prof. Bing Liu

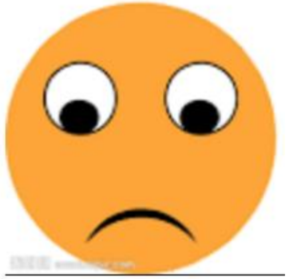
Sentiment Analysis and Opinion Mining

Sentiment analysis (SA) or opinion mining

computational study of opinion, sentiment, appraisal, evaluation, and emotion.

Staffs are not that friendly, but the taste covers all."

Staff



Taste

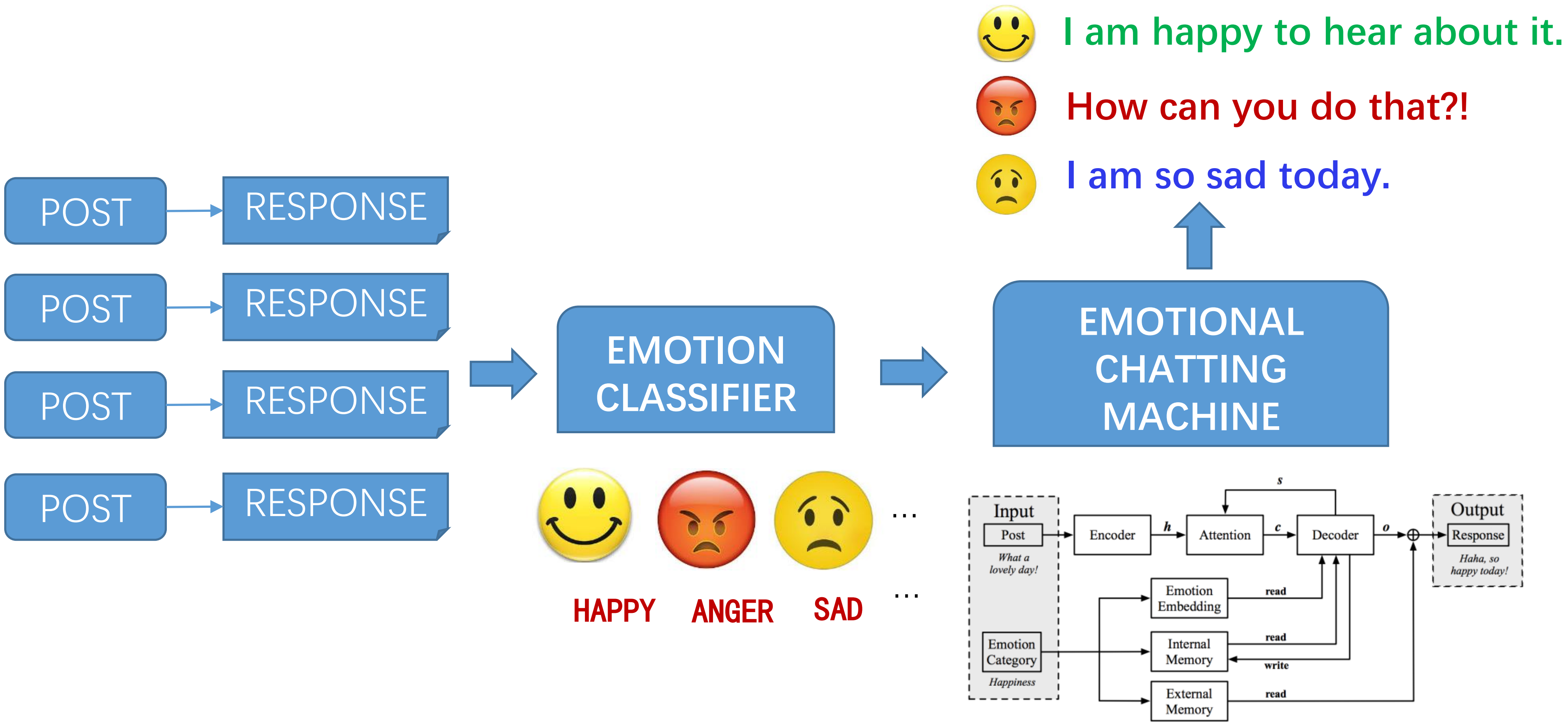
Emotion/Sentiment Generation

Varying the code of sentiment
this movie was awful and boring . this movie was funny and touching .
jackson is n't very good with documentary jackson is superb as a documentary productions
you will regret it you will enjoy it

Varying the unstructured code z
("negative", "past") the acting was also kind of hit or miss . i wish i 'd never seen it by the end i was so lost i just did n't care anymore
("negative", "present") the movie is very close to the show in plot and characters the era seems impossibly distant i think by the end of the film , it has confused itself
("negative", "future") i wo n't watch the movie and that would be devastating ! i wo n't get into the story because there really is n't one

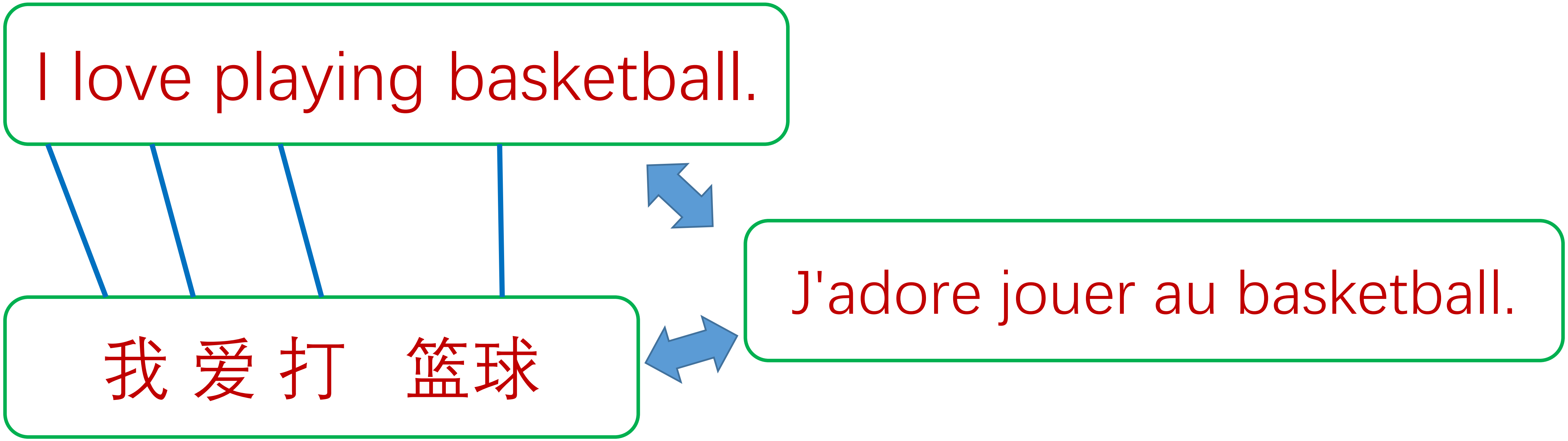
Hu Z, Yang Z, Liang X, et al. Controllable Text Generation[J]. arXiv preprint arXiv:1703.00955, 2017.

Emotional Content Generation



Hao Zhou, Minlie Huang, Tianyang Zhang, Xiaoyan Zhu, Bing Liu Emotional Chatting Machine: Emotional Conversation Generation with Internal and External Memory. arXiv:1704.01074

Machine Translation



Knowledge Graph: Storing Human Knowledge

Freebase
1.9 billion triples

Representing Textual Data

Traditional, Fixed Representation

- [1/0, 1/0, ⋯, 1/0]
- [tf*idf, ⋯, tf*dif]
- [$\#w_1$, $\#w_2$, $\#w_3$, ⋯. , $\#w_n$]

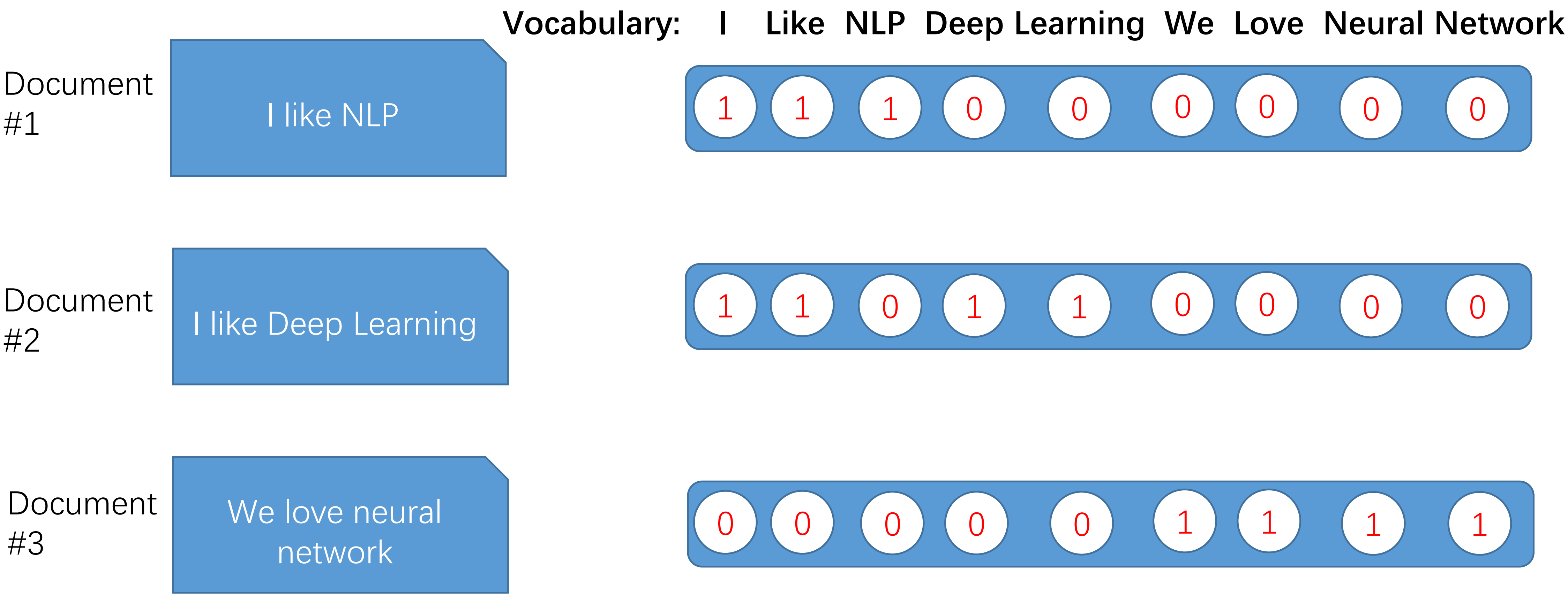
- High-dimensional, sparse
- Heavy domain expertise
- Expensive engineering

New Feature Representation

Trainable, Learnable

- Low-dimensional, dense
- Data and model driven
- Task oriented

Traditional Representation



Similarity(DOC#2, DOC#3)???

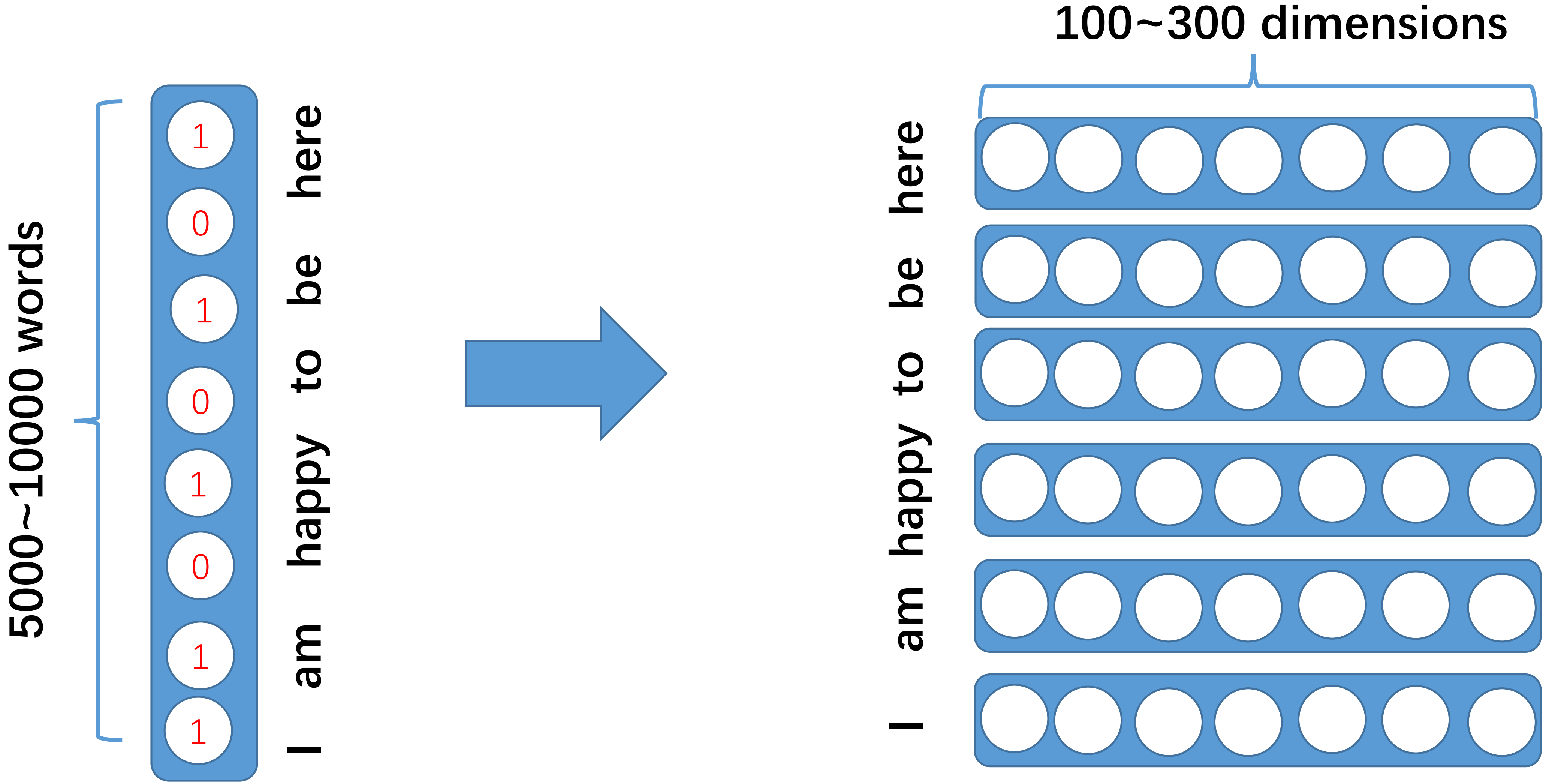
Traditional Representation

	Document	Vocabulary: I	Like	NLP	Deep	Learning	We	Love	Neural	Network
Document #1	I like NLP	TF=1 DF=2	TF=1 DF=2	TF=1 DF=1						
Document #2	I like Deep Learning	TF=1 DF=2	TF=1 DF=2		TF=1 DF=1	TF=1 DF=1				
Document #3	We love neural network						TF=1 DF=1	TF=1 DF=1	TF=1 DF=1	TF=1 DF=1

TF(w,d): count of word **w** in document **d**
DF(w): #docs containing **w**

Similarity(DOC#2, DOC#3)???

New Representation with Deep Learning



Roadmap of this Lecture

Representing Words → **Representing Phrases/Sentences** → **Representing Documents**

Neural language model
Word2vect
GloVe

Recursive Neural Network
Recurrent Neural Network
Convolutional Neural Network

Convolutional Neural Network

Statistical Language Models

- A ***statistical language model*** is a probability distribution over sequences of words.
  A sequence ***W*** consists of ***L*** words (eg: *I like deep learning*):

$$
\begin{aligned}
P(W) &= P(w_{1:L}) = P(w_1, \cdots, w_L) \\
&= P(w_1)P(w_2|w_1)P(w_3|w_1w_2)\cdots P(w_L|w_{1:(L-1)}) \\
&= \prod_{i=1}^{L} P(w_i|w_{1:(i-1)}).
\end{aligned}
$$

- ***N***-gram language model

$$
P(W) = \prod_{i=1}^{L} P(w_i|w_{(i-n+1):(i-1)}).
$$

Statistical Language Models

- P(I like deep learning)
  =P(I)*
  P(like|I)*
  P(deep|I like)*
  P(learning|I like deep)

For 1-gram LM (unigram LM):

- P(I like deep learning)
  =P(I)* P(like)* P(deep)* P(learning)

For 2-gram LM (bigram LM):

- P(I like deep learning)
  =P(I)*
  P(like|I)*
  P(deep|like)*
  P(learning|deep)

$$P(like|I) = \frac{\#(I, like)}{\sum_w \#(I, w)}$$

$$P(\mathrm{deep}|I, like) = \frac{\#(I, like, deep)}{\sum_w \#(I, like, w)}$$

Word order matters

Issues with Traditional Language Models

- Data sparsity: ***n*** cannot be too large
  - Model size grows exponentially with ***the size of vocabulary***.
  - Trigram: Model size = **$|V|^3$**
- Reliable probability estimation: smoothing techniques

$$P(I) = \frac{\#(I)}{\sum_w \#(w)}$$

$$P(I) = \frac{\#(I) + k}{\sum_w [\#(w) + k]}$$

Neural Probabilistic Language Model

Evolve from traditional language models

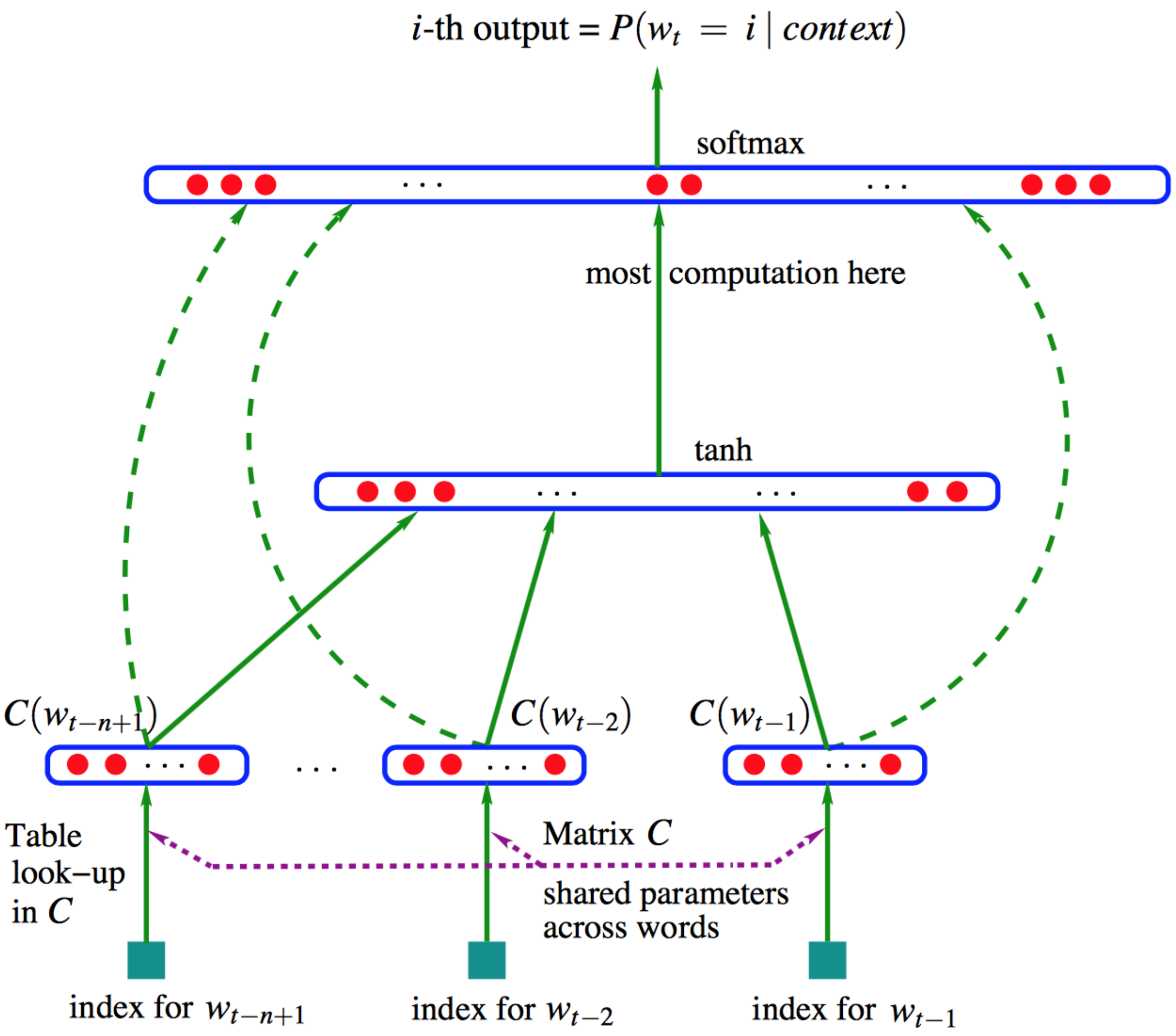
$$\hat{P}(w_1^T) = \prod_{t=1}^{T} \hat{P}(w_t | w_1^{t-1}),$$

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

$$\hat{P}(w_t | w_{t-1}, \cdots w_{t-n+1}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}}$$

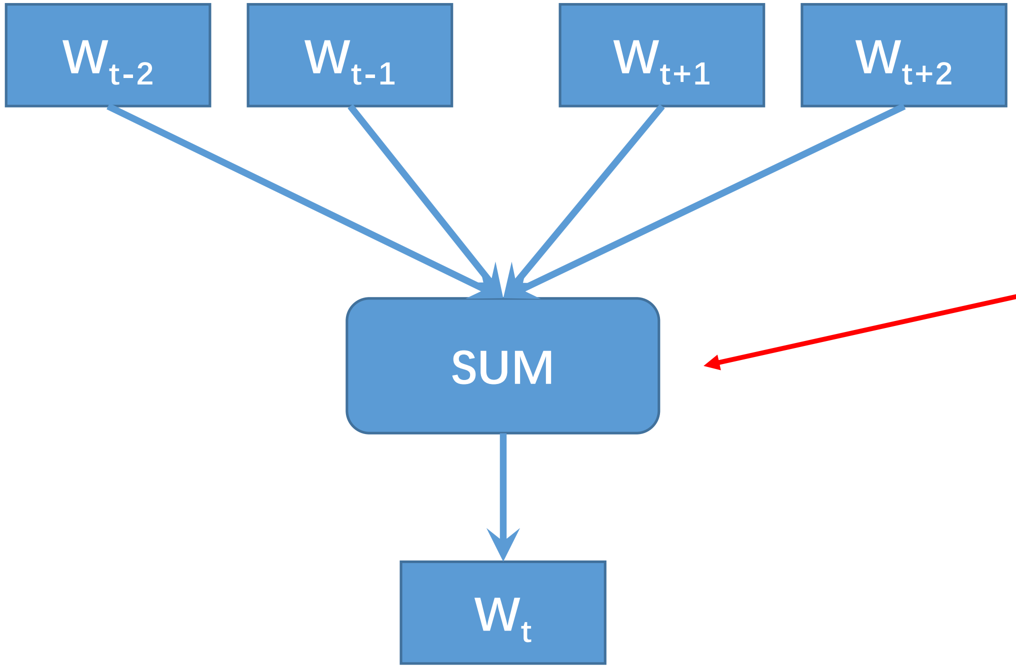
$$y = b + Wx + U \tanh(d + Hx)$$

Bengio et al 2003. **A Neural Probabilistic Language Model.** JMLR 3 (2003) 1137–1155



Representing Words

- CBOW: Continuous Bag-Of-Words
- Predicting the current word from contextual words



Context: $c_t = w_{t-n}, \cdots, w_{t+n}$

$P(w_t|c_t) = softmax(\mathbf{V}'^{T}_{w_t}\mathbf{c}_t)$

$\boxed{\mathbf{c}_t = \sum_{t-n \leq j \leq t+n, j \neq 0} \mathbf{V}_j}$

Loss function for $w_1 w_2 \cdots w_T$

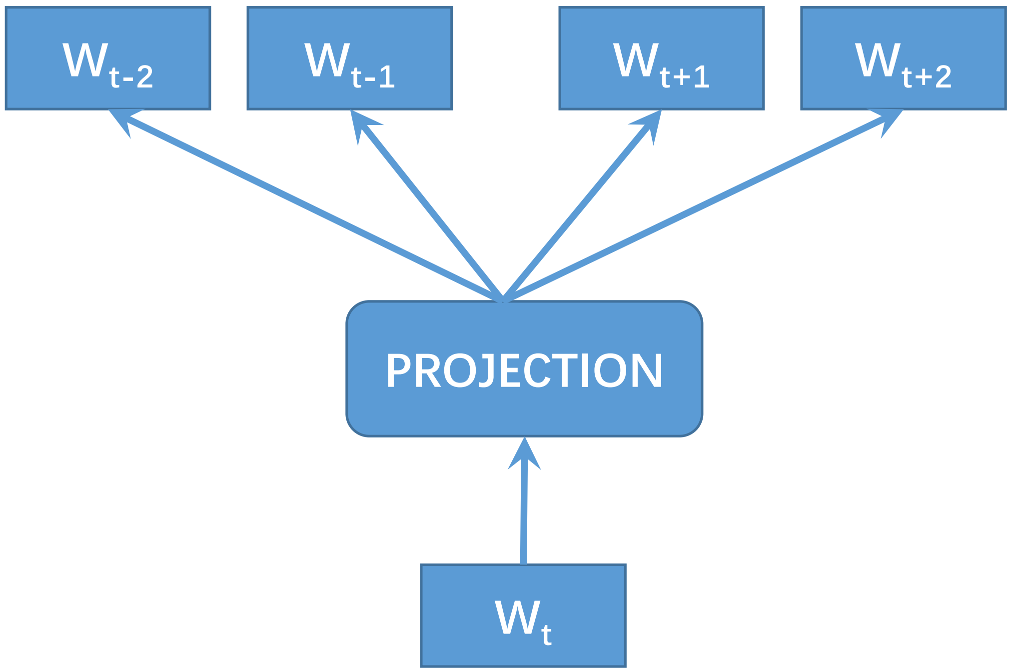
$\mathcal{L}_\theta = -\frac{1}{T}\sum_{t=1}^{T} logP(w_t|c_t)$

Each word has an input vector V and output vector V'

Mikolov et al 2013. Efficient Estimation of Word Representations in Vector Space

Representing Words

- Skip-gram: predicting the contextual words from the current word



$$\mathrm{P}(w_{t+j}|w_t) = softmax(\mathbf{V}_{w_t}^T \mathbf{V}'_{w_{t+j}})$$

$$= \frac{exp(\mathbf{V}_{w_t}^T \mathbf{V}'_{w_{t+j}})}{\sum_w exp(\mathbf{V}_{w_t}^T \mathbf{V}'_w)}$$

$$\mathscr{L}_\theta = -\frac{1}{T}\sum_{t=1}^{T} \sum_{t-n \leq j \leq t+n, j \neq 0} logP(w_{t+j}|w_t)$$

NO hidden layer
Each word has an input vector V and output vector V'

Mikolov et al 2013. Efficient Estimation of Word Representations in Vector Space

What If Very Large Vocabulary

- The normalization factor is too expensive for computation

$$\hat{P}(w_t | w_{t-1}, \cdots w_{t-n+1}) = \frac{e^{y_{w_t}}}{\boxed{\sum_i e^{y_i}}} = softmax(y_{w_t})$$

- Solution
  - Hierarchical Softmax: a tree-structured vocabulary
  - Negative Sampling: sample some random words for the context

A. Mnih and G. Hinton. A scalable hierarchical distributed language model . In: Advances in neural information processing systems (2009).

B. F. Morin and Y. Bengio. Hierarchical Probabilistic Neural Network Language Model. In: Aistats. Vol. 5. Citeseer. 2005, pp. 246 252.

C. T. Mikolov et al. Efficient estimation of word representations in vector space . In: arXiv preprint arXiv:1301.3781 (2013).

Hierarchical Softmax

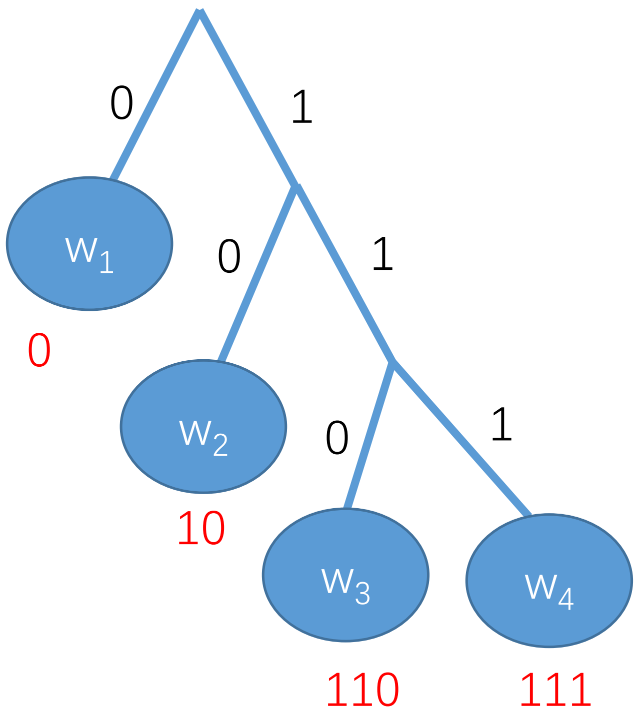
- Huffman tree to encode each word
  - More frequent words closer to root
  - Each word corresponds to a unique code path: **b(w,1),b(w,2),⋯,b(w,m)** where b(w,i) in {0,1}

For w_3: b(w,1)=1, b(w,2)=1, b(w,3)=0

- Given context **h**, the prob. of observing **w**:

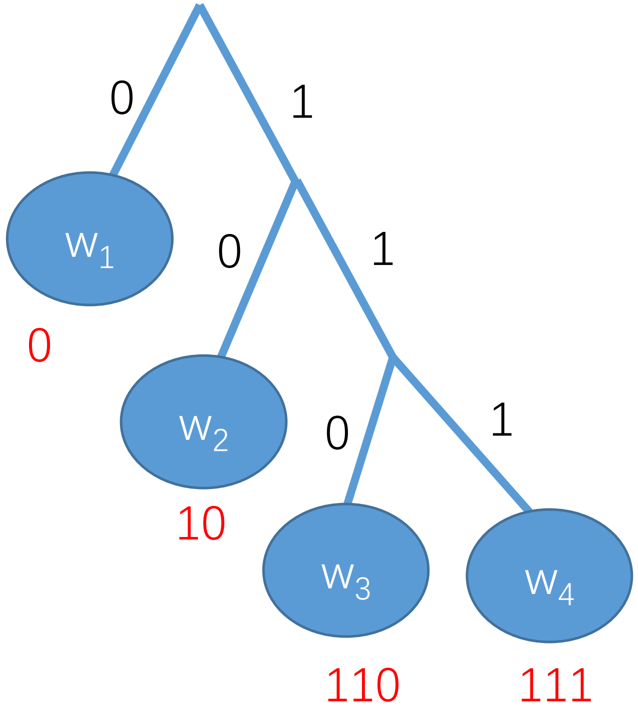
P(w|h)=P(b(w,1),b(w,2),⋯,b(w,m) |h)

$$=\prod_{j=1}^{m} P(b(w,j)|b(w,1),\ldots,b(w,j-1),h)$$



Hierarchical Softmax

- Denote b(w,1),b(w,2),⋯,b(w,j-1)=**n(w,j-1)**, the (**j-1**)th non-leaf node starting from root for word ***w***
- Given context **h**, the prob. of observing **w**:



$$P(w|h) = \prod_{j=1}^{m} P(b(w,j)|b(w,1),\dots,b(w,j-1),h)$$

$$= \prod_{j=1}^{m} P(b(w,j)|n(w,j-1),h)$$

- Since ***b(w,j)*** in {0,1}, sigmoid function can be used:

$$P(b(w,j)=1|n(w,j-1),h)=sigmoid(\boldsymbol{W}_{n(w,j-1)}h+\boldsymbol{b}_{n(w,j-1)})$$

Computation Reduction in Hierarchical Softmax

- For each word **w**, at most **log|V|** nodes needs to compute
  - Each word has at most **log|V|** path codes (**n(w,i)** is prefix)
  - **log|V|** is the height of Huffman tree
  - More frequent words have short paths – More accelerations!
- Speed up from **|V|** to **log|V|**

$$P(w_{t+j}|w_t) = softmax(\mathbf{V}_{w_t}^T \mathbf{V}'_{w_{t+j}}) = \frac{exp(\mathbf{V}_{w_t}^T \mathbf{V}'_{w_{t+j}})}{\boxed{\sum_w exp(\mathbf{V}_{w_t}^T \mathbf{V}'_w)}}$$

$$\Rightarrow$$

$$P(w|h) = \prod_{j=1}^{m} P(b(w,j)|n(w,j-1),h)$$

Computation Reduction in Hierarchical Softmax

- Each word is associated with more parameters
  - The number of nodes in the code path

$$P(b(w,j)=1|n(w,j-1),h)=sigmoid(\boldsymbol{W}_{n(w,j-1)}h+\boldsymbol{b}_{n(w,j-1)})$$

  - At most **log|V|** nodes
  - Total #parameters of the model= **|V|*log|V|**
- Better time efficiency at the cost of space!

Accelerating the Training Process

- **Negative Sampling**

Observed, true samples

I like deep learning

I love deep learning

I code with deep learning

Skip-gram
P(like|I,Deep)

Generated, fake samples

I hate deep learning

I play deep learning

A. **Mnih and G. Hinton. A scalable hierarchical distributed language model . In: Advances in neural information processing systems (2009).**
B. **F. Morin and Y. Bengio. Hierarchical Probabilistic Neural Network Language Model. In: Aistats. Vol. 5. Citeseer. 2005, pp. 246 252.**

Skip-gram Model

- Given a pair of <word, context> (*w,c*), the probability that word *w* is observed in the context *c* is given by:

$$Pr(D=1|w,c) = \frac{1}{1+\exp(-\mathbf{w}^T\mathbf{c})}$$

- where ***w*** and ***c*** are embedding vectors of *w* and *c* respectively. The probability of not observing word *w* in the context *c* is given by:

$$Pr(D=0|w,c) = 1 - \frac{1}{1+\exp(-\mathbf{w}^T\mathbf{c})}.$$

Skip-gram Model

- Given a training set ***D***, the word embeddings are learned by maximizing the following objective function:

$$J(\theta) = \sum_{w,c \in \mathcal{D}} Pr(D = 1|w, c) + \sum_{w,c \in \mathcal{D}'} Pr(D = 0|w, c)$$

- where the set ***D′*** is randomly sampled negative examples, assuming they are all incorrect.

Computation Reduction in Negative Sampling

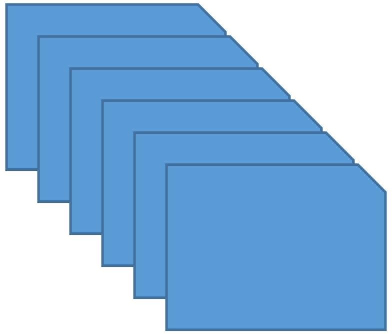
- From the normalization factor to a sigmoid function

$$P(w_{t+j}|w_t) = softmax(\mathbf{V}_{w_t}^T \mathbf{V}'_{w_{t+j}})$$

$$= \frac{exp(\mathbf{V}_{w_t}^T \mathbf{V}'_{w_{t+j}})}{\boxed{\sum_w exp(\mathbf{V}_{w_t}^T \mathbf{V}'_w)}}$$

$$\Rightarrow$$

$$Pr(D = 1|w, c) = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{c})}$$

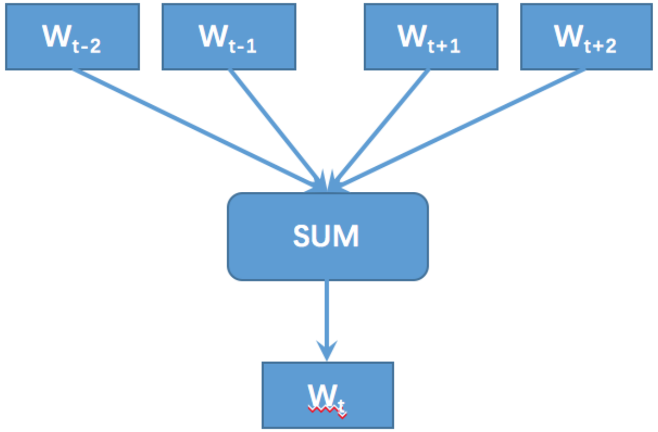


A very large corpus:
Many many documents

Train the model

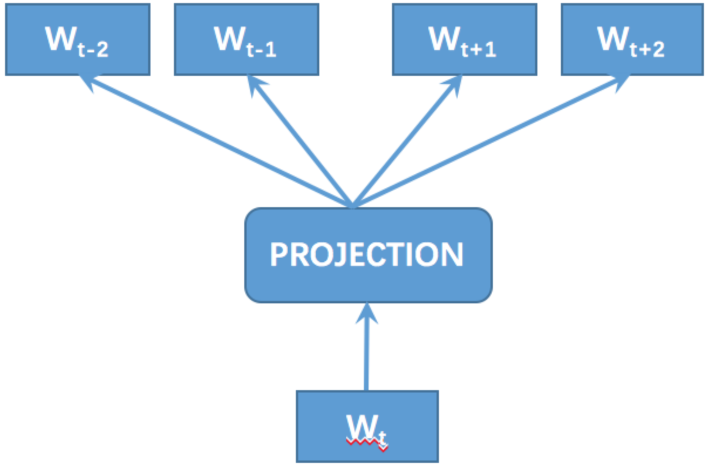
$$P(w_t|c_t) = softmax(\mathbf{V}'^{T}_{w_t}\mathbf{c}_t)$$

CBOW



$$\mathrm{P}(w_{t+j}|w_t) = softmax(\mathbf{V}^{T}_{w_t}\mathbf{V}'_{w_{t+j}})$$

Skip-gram



$$\mathscr{L}_\theta = -\frac{1}{T}\sum_{t=1}^{T}\sum_{t-n\le j\le t+n, j\ne 0} logP(w_{t+j}|w_t)$$

Word vectors are the parameters of the model

Language Regularity

Words similar to"Sweden"

word	Cosine similarity
norway	.760
denmark	.715
finland	.620
switzerland	.588
belgium	.585
netherlands	.575

Semantic:

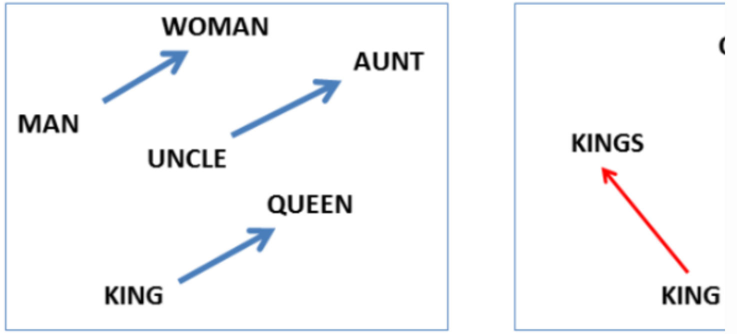
$$v_{brother} - v_{sister} = v_{grandson} - v_{granddaughter}$$

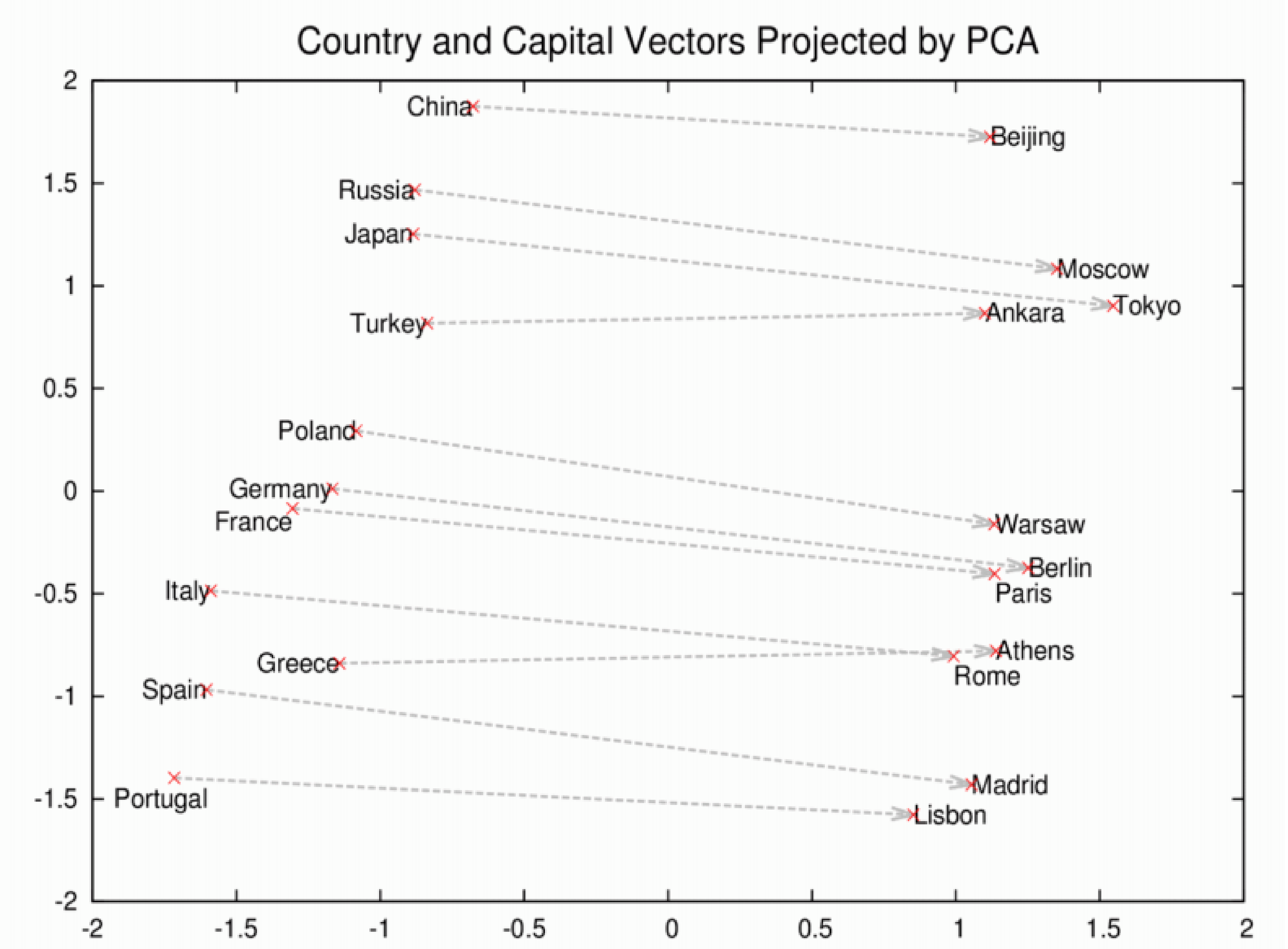
Syntactic:

$$v_{apparent} - v_{apparently} = v_{rapid} - v_{rapidly}$$

http://deeplearning4j.org/word2vec

Language Regularity





http://deeplearning4j.org/word2vec

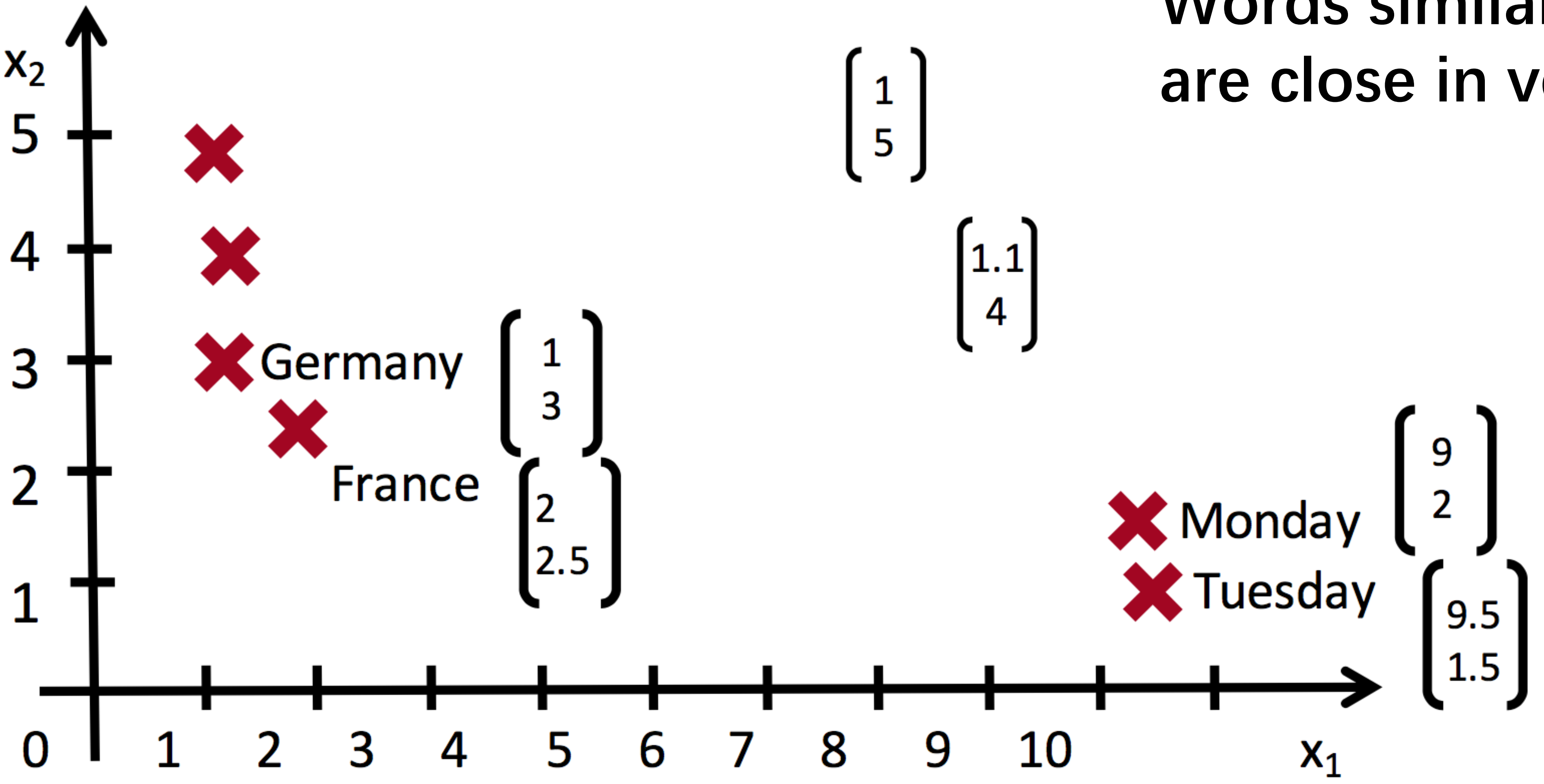
Now, each word can be represented by

- A low-dimensional, real-valued, and dense vectors

$$\textbf{Deep}= \begin{bmatrix} 0.123 \\ 0.243 \\ 0.789 \\ -0.150 \\ 0.893 \\ 0.163 \\ -0.876 \end{bmatrix} \quad \textbf{Learning}= \begin{bmatrix} 0.978 \\ 0.673 \\ -0.123 \\ -0.450 \\ 0.708 \\ 0.467 \\ -0.234 \end{bmatrix} \quad \textbf{NLP}= \begin{bmatrix} -0.360 \\ -0.445 \\ 0.809 \\ -0.961 \\ 0.537 \\ 0.189 \\ -0.776 \end{bmatrix}$$

Word Vectors

Words similar in semantics are close in vector space



Slides from Socher CS224d

Resources for Word Vectors

- download
  - glove
    - homepage: https://nlp.stanford.edu/projects/glove/
    - code: https://github.com/maciejkula/glove-python
    - word vectors: http://nlp.stanford.edu/data/glove.6B.zip
  - word2vec
    - homepage: https://code.google.com/archive/p/word2vec/
    - code: https://github.com/tmikolov/word2vec
    - word vectors: https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
  - Chinese corpus
    - https://dumps.wikimedia.org/zhwiki/latest/zhwiki-latest-pages-articles.xml.bz2

Word Vector Training with Word2vec

- Exemplar command
  - ./word2vec -train text8 -output vectors.bin -cbow 1 -size 200 -window 8 -negative 25 -hs 0 -sample 1e-4 -threads 20 -binary 1 -iter 15
- Parameters
  - -train: Use text data from <file> to train the model
  - -output: Use <file> to save the resulting word vectors
  - -size: Set size of word vectors; default is 100
  - -window: Set max skip length between words; default is 5
  - -sample: Set threshold for occurrence of words.
  - -hs: Use Hierarchical Softmax; default is 0
  - -negative: Number of negative examples; default is 5
  - -threads: Use <int> threads (default 12)
  - -iter: Run more training iterations (default 5)
  - -binary: Save the resulting vectors in binary moded; default is 0 (off)

Messages about Word Vectors

- Representing word with vector is a **precursor** for natural language understanding with deep learning

- **Well pre-trained** word vectors are **crucial** for the performance

- **Fine-tuning** word vectors generally **improve** the performance but a well pre-trained word vectors are good enough, and sometimes even degrade the performance.

Roadmap of this Lecture

Representing Words → **Representing Phrases/Sentences** → **Representing Documents**

Word2vect
GloVe

Recursive Neural Network
Recurrent Neural Network
Convolutional Neural Network

Convolutional Neural Network

Thanks for Attention

- Dr. Minlie Huang
- aihuang@tsinghua.edu.cn
- Homepage: http://coai.cs.tsinghua.edu.cn/hml