Deep Learning for Natural Language Processing

Dr. Minlie Huang (黄民烈)

aihuang@tsinghua.edu.cn

Computer Science Department

Tsinghua University

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

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

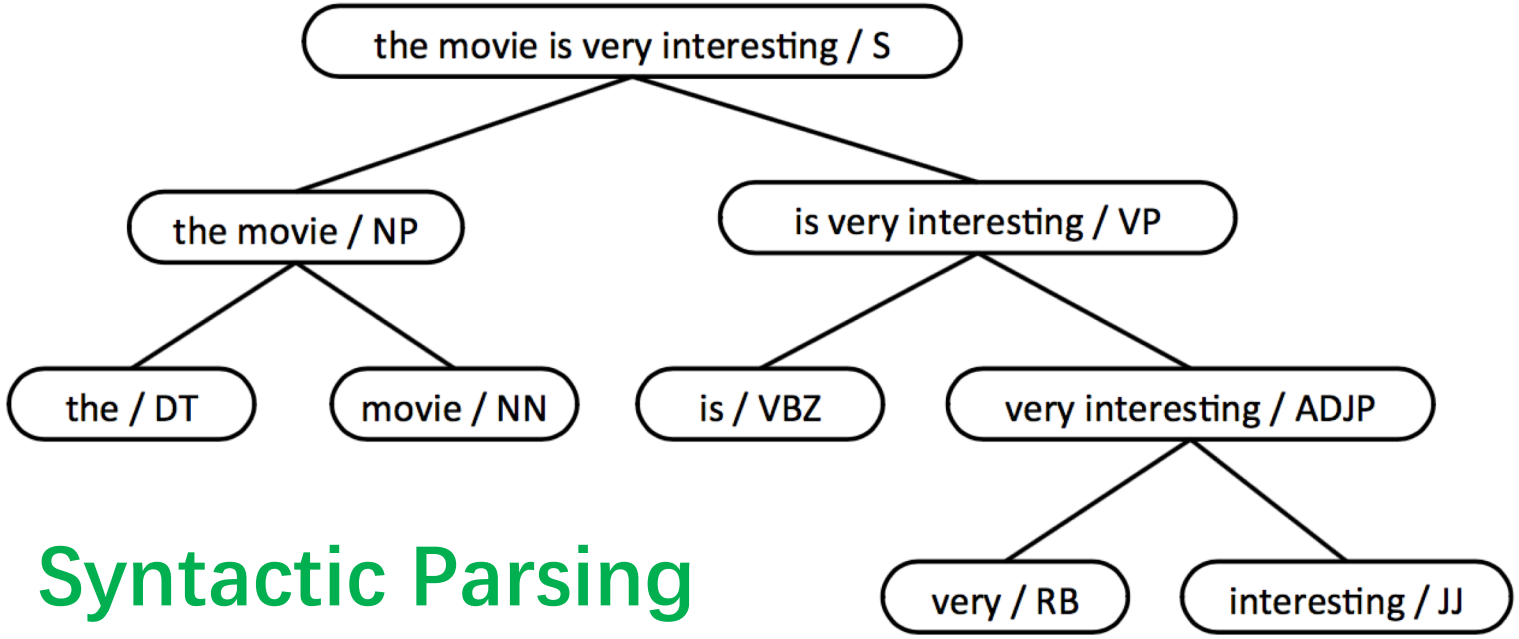
Tagging and Parsing

Part-of-speech Tagging

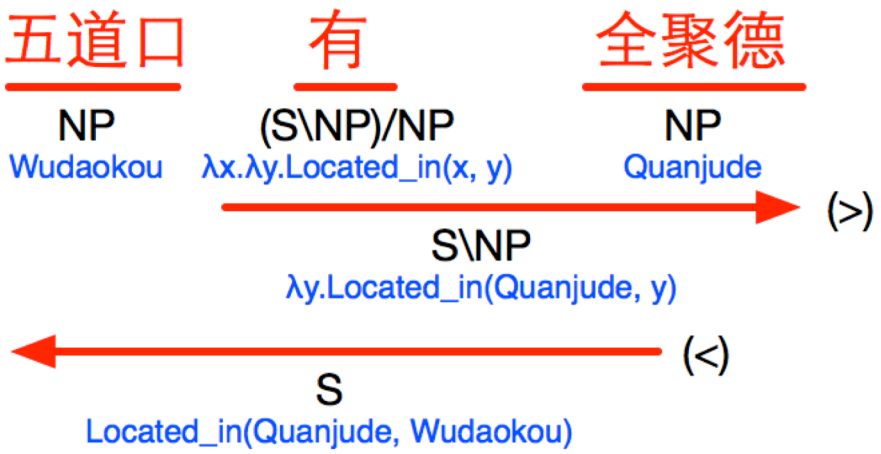
The movie is very interesting

DT NN VBZ RB JJ

Syntactic Parsing



Semantic 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*

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

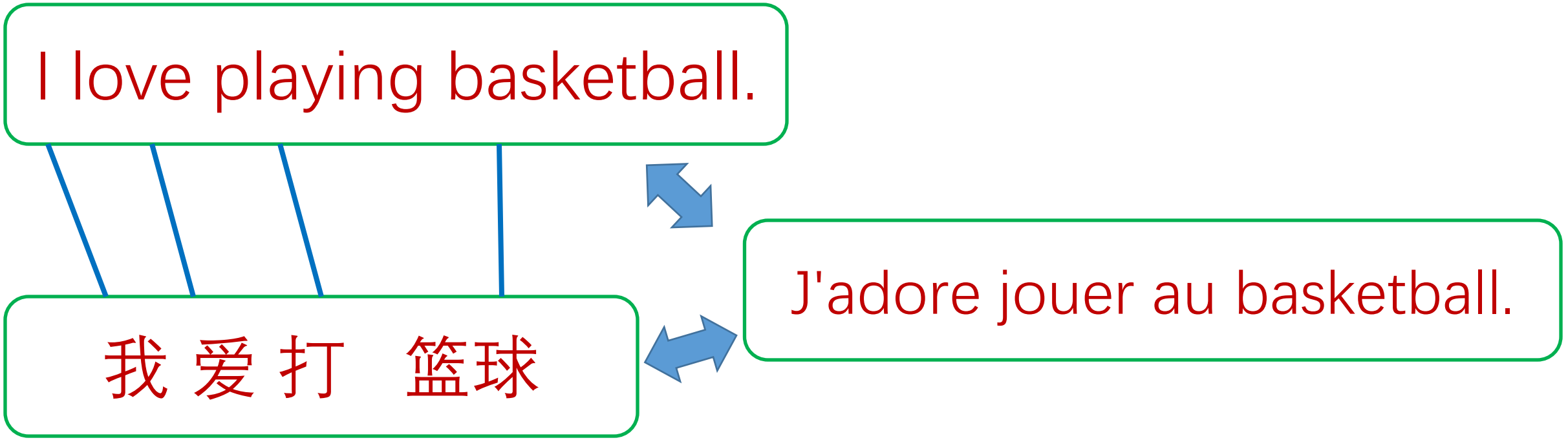
Emotion/Sentiment Generation

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

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

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

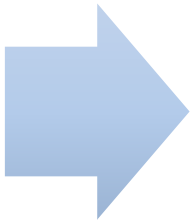
Machine Translation



Representing Textual Data

Traditional, Fixed Representation

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



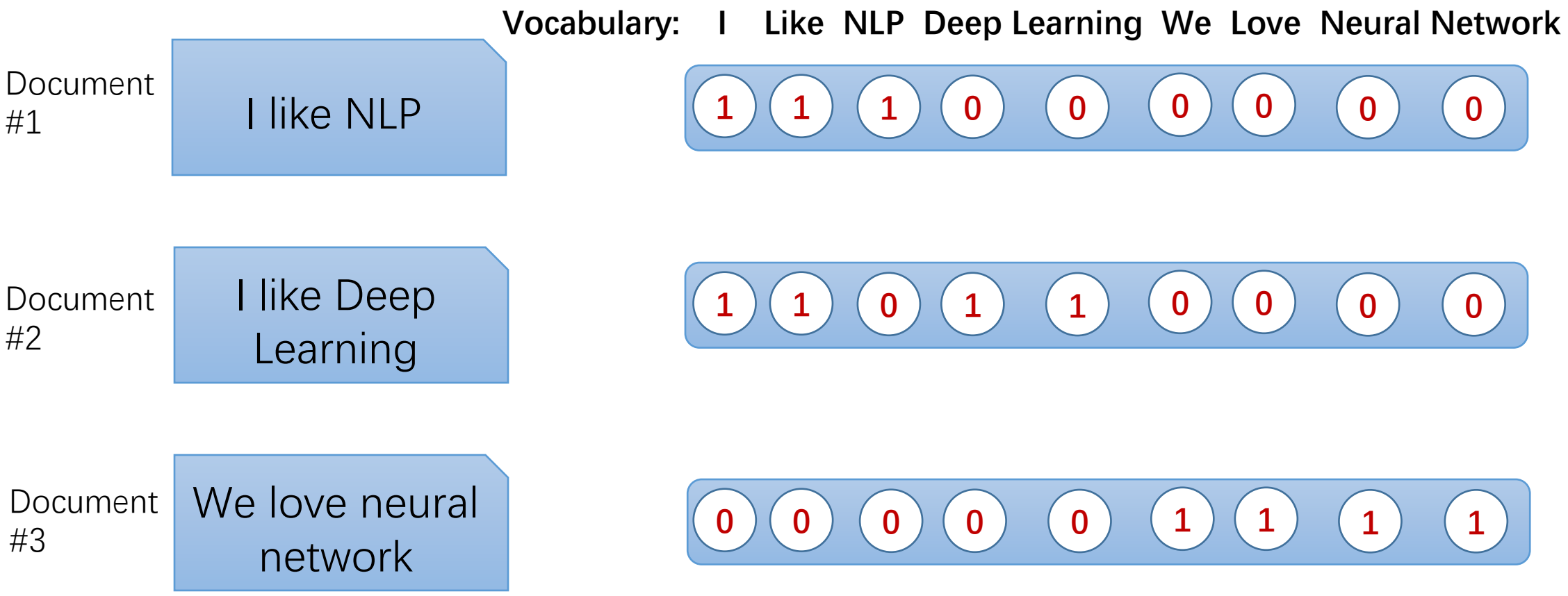
Trainable, Learnable

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

New Feature Representation

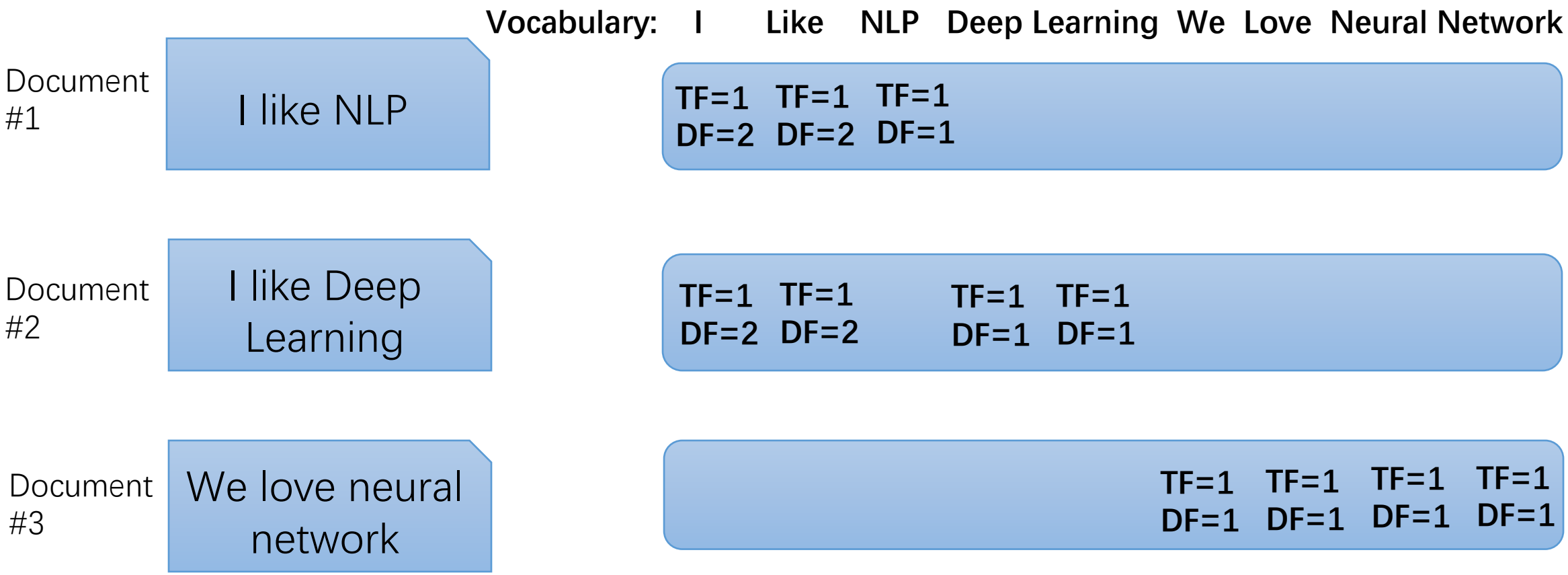
- Low-dimensional, dense
- Data and model driven
- Task oriented

Traditional Representation



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

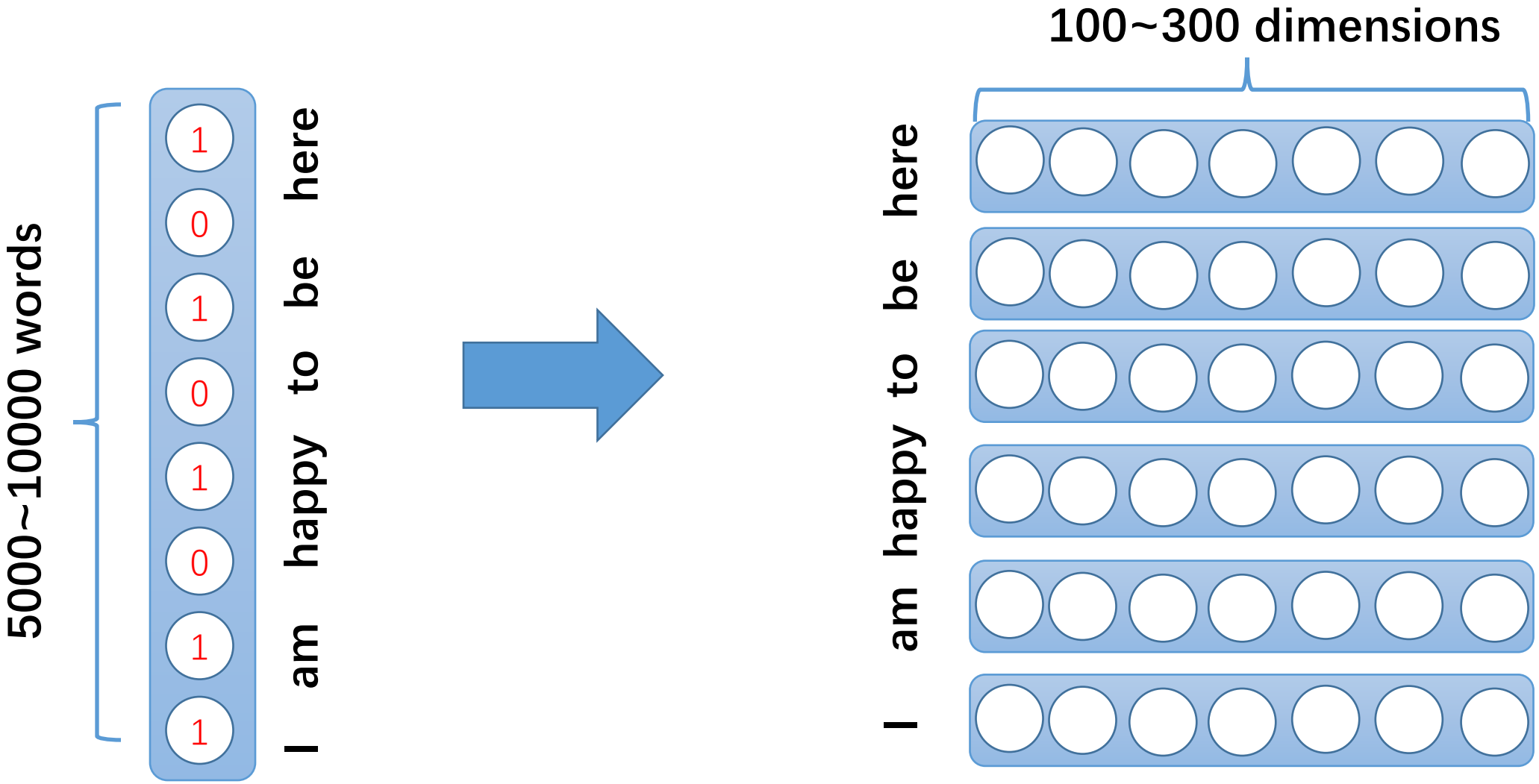
Traditional Representation (Discrete Symbols)



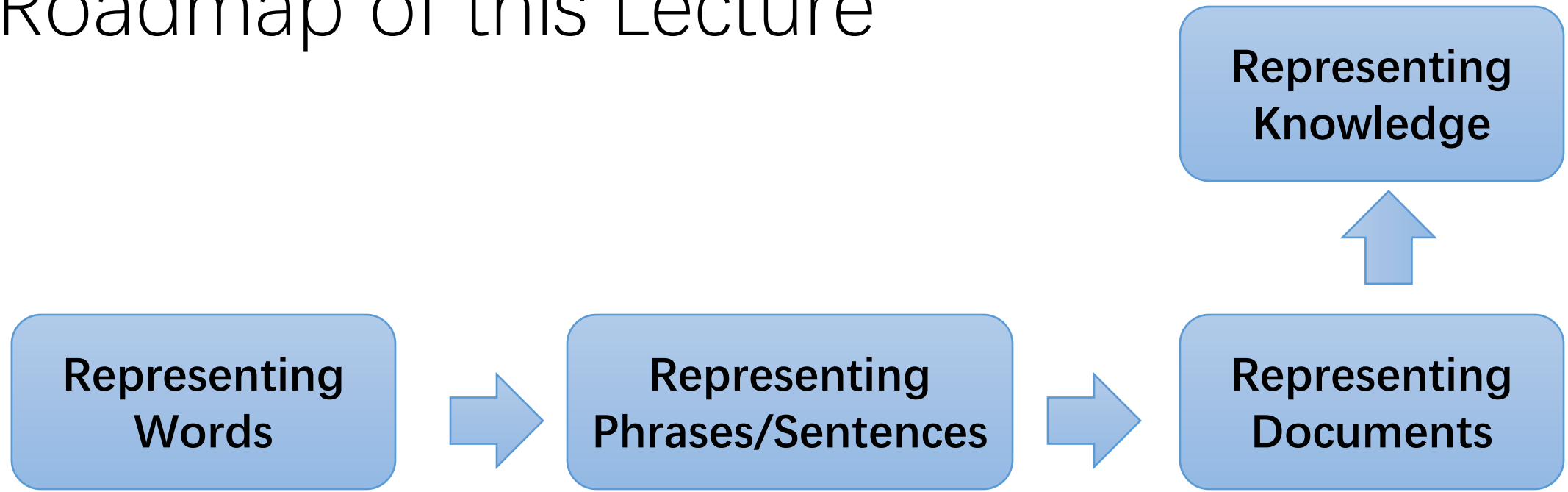
TF(w,d): count of word **w** in document **d**
DF(w): #docs containing **w**

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

New Representation with Deep Learning



Roadmap of this Lecture



Neural language model
Word2vec
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)

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

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

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

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

Neural Probabilistic Language Model

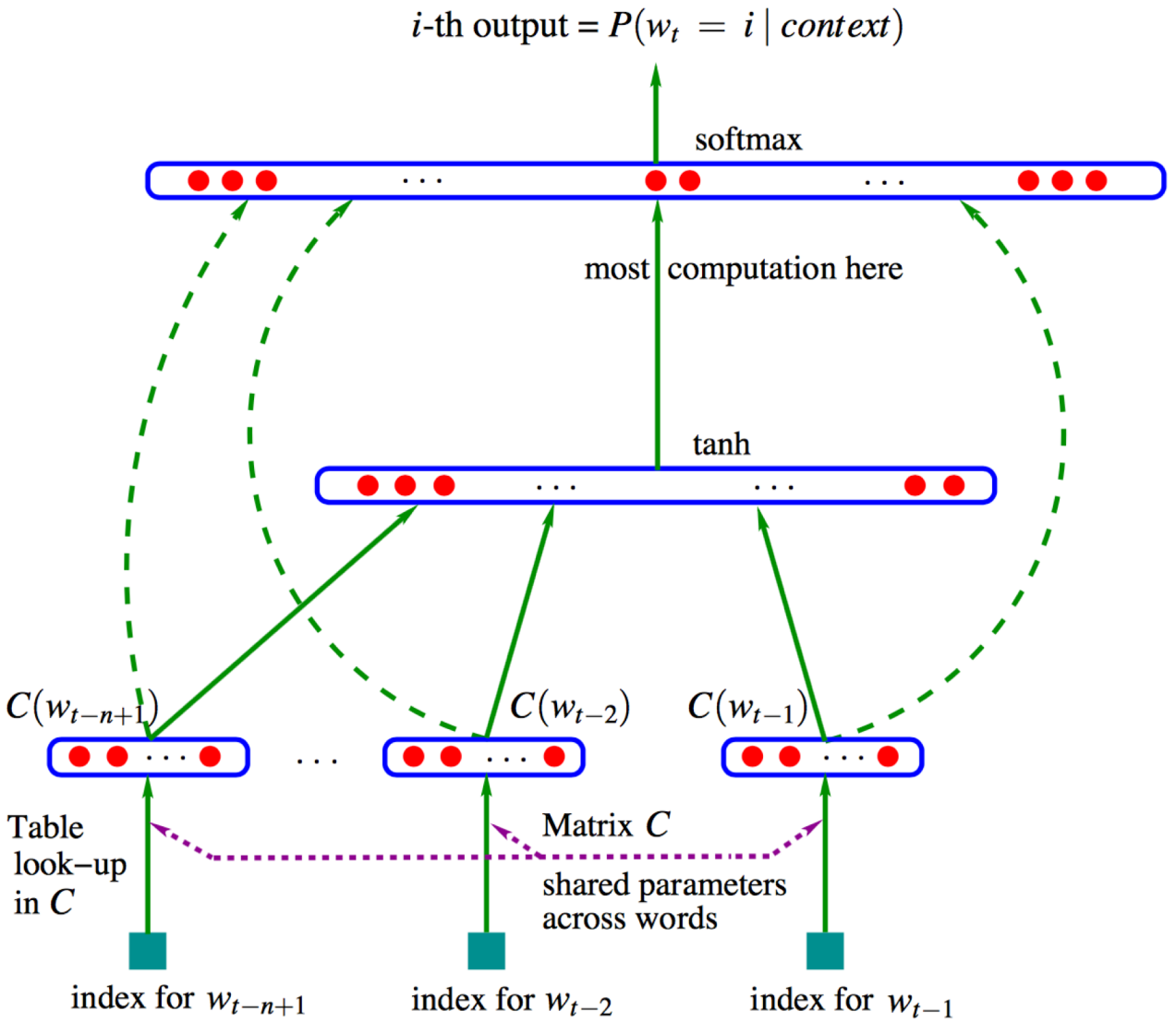
Evolve from traditional language models

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

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

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

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

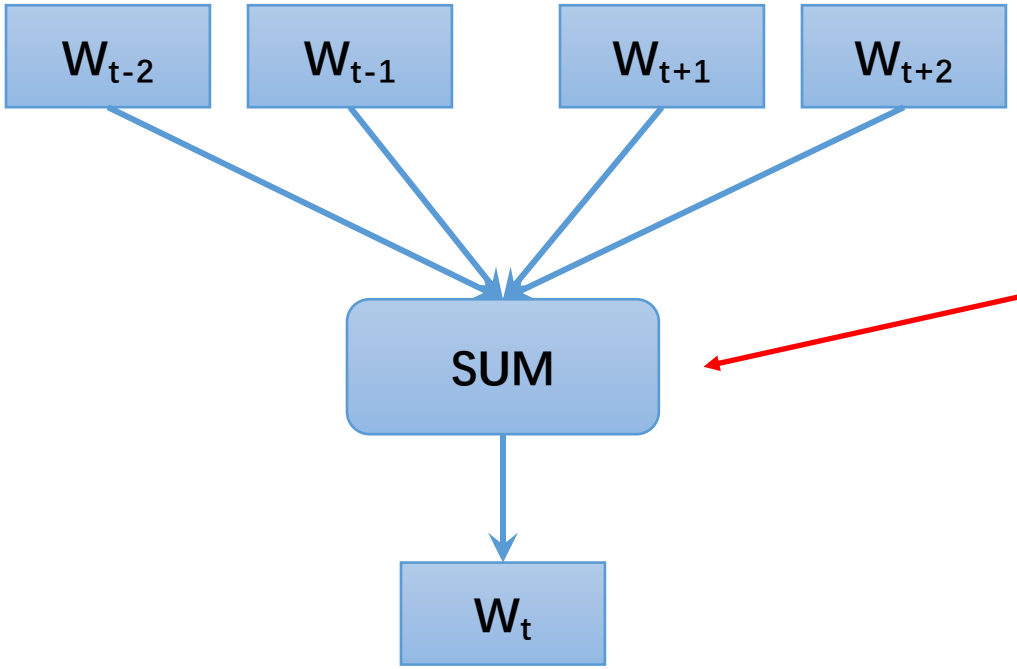


Bengio et al 2003. **A Neural Probabilistic Language Model.** JMLR 3 (2003) 1137–1155

Representing Words

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

V有两套是因为每个词都可能出现在条件概率的两边，即input和output



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

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

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

Loss function for $w_1 w_2 \cdots w_T$

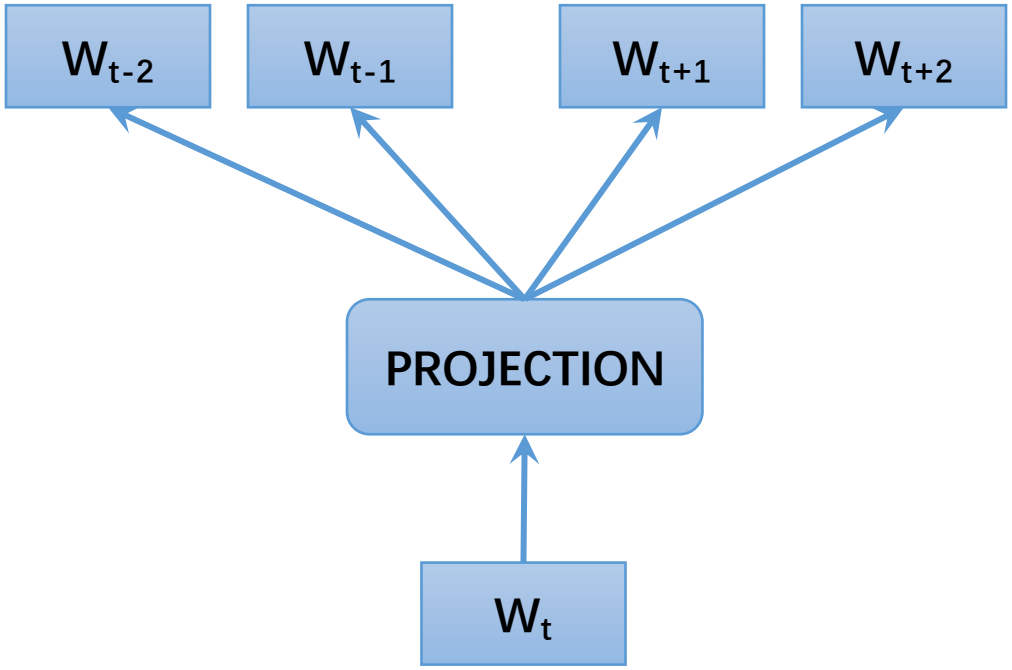
$$\mathscr{L}_\theta = -\frac{1}{T}\sum_{t=1}^{T} logP(w_t|c_t)$$

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



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

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

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

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

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

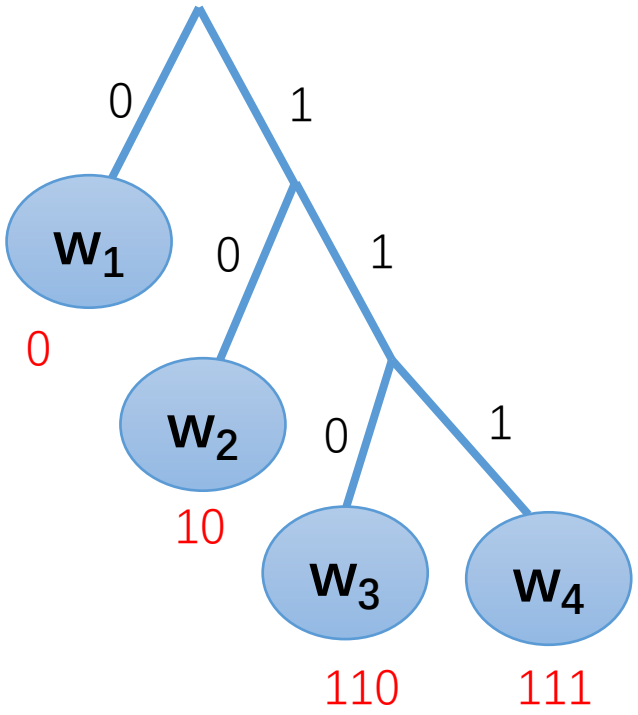
- Solution
  - Hierarchical Softmax: a tree-structured vocabulary
  - Negative Sampling: sample some random words for the context

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)

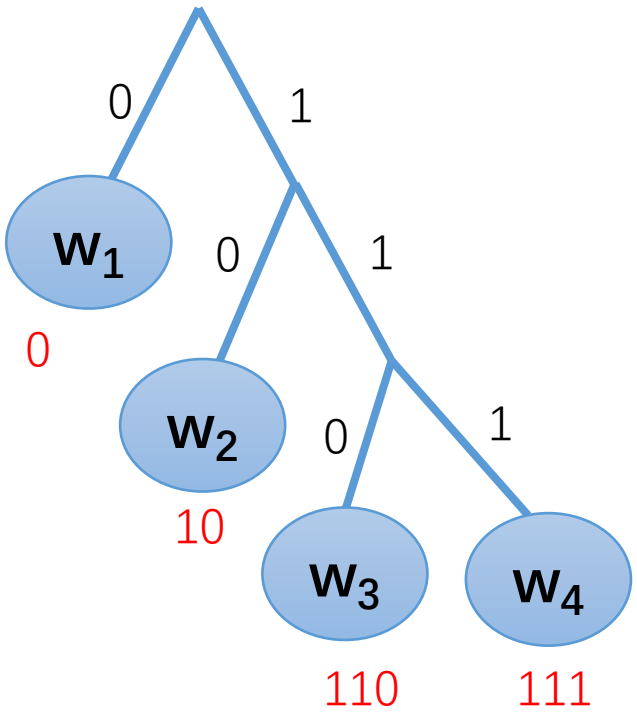
$$=\prod_{j=1}^{m} P(b(w,j)|b(w,1),\ldots,b(w,j-1),h)$$

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**:

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

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



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

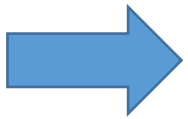
$$P(b(w,j)=1|n(w,j-1),h)=sigmoid(\boldsymbol{W}_{n(w,j-1)}h+\boldsymbol{b}_{n(w,j-1)})$$

Computation Reduction in Hierarchical Softmax

- For each word **w**, at most **log|V|** nodes needs to be computed
  - 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|**

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

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



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

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

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

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

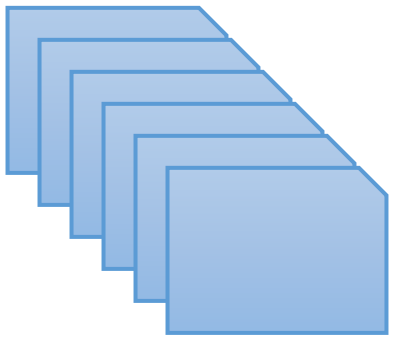
$$\Rightarrow \quad Pr(D = 1|w, c) = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{c})}$$

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

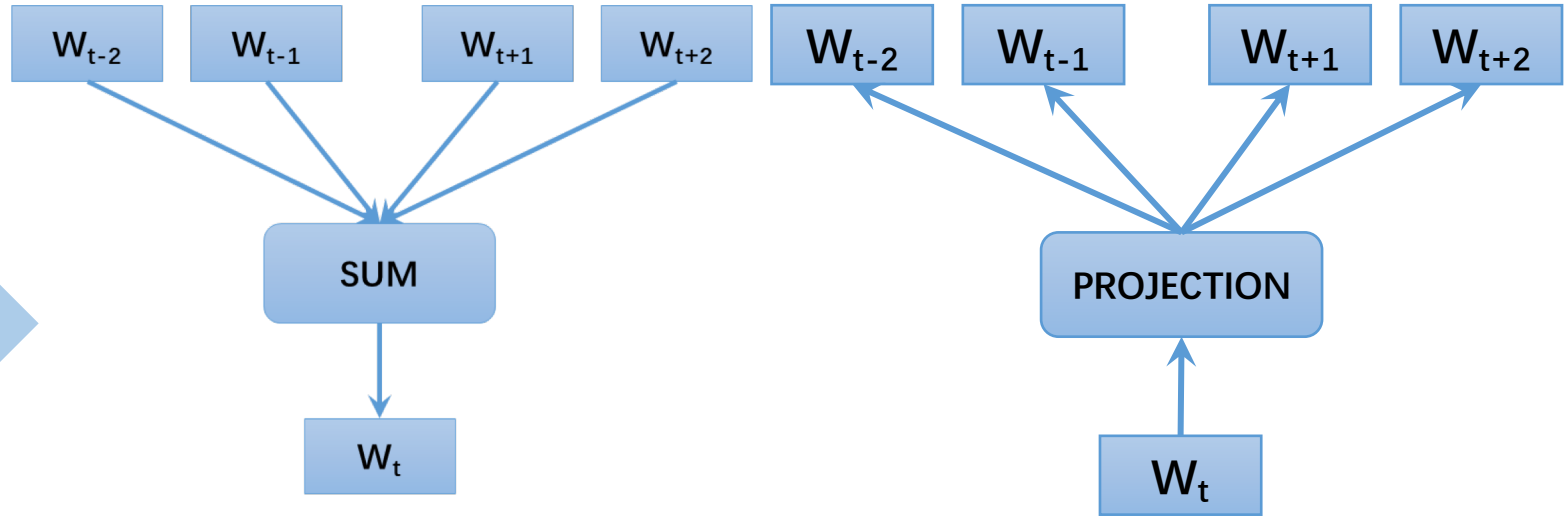
$$P(w_{t+j}|w_t) = softmax(\mathbf{V}^{T}_{w_t}\mathbf{V}'_{w_{t+j}})$$

CBOW

Skip-gram



Train the model



A very large corpus:
Many many documents

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

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:

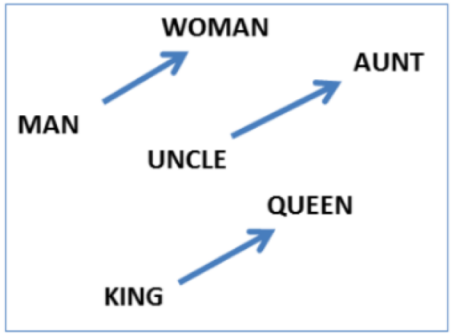
$$v_{brother} - v_{sister} = v_{grandson} - v_{granddaughter}$$

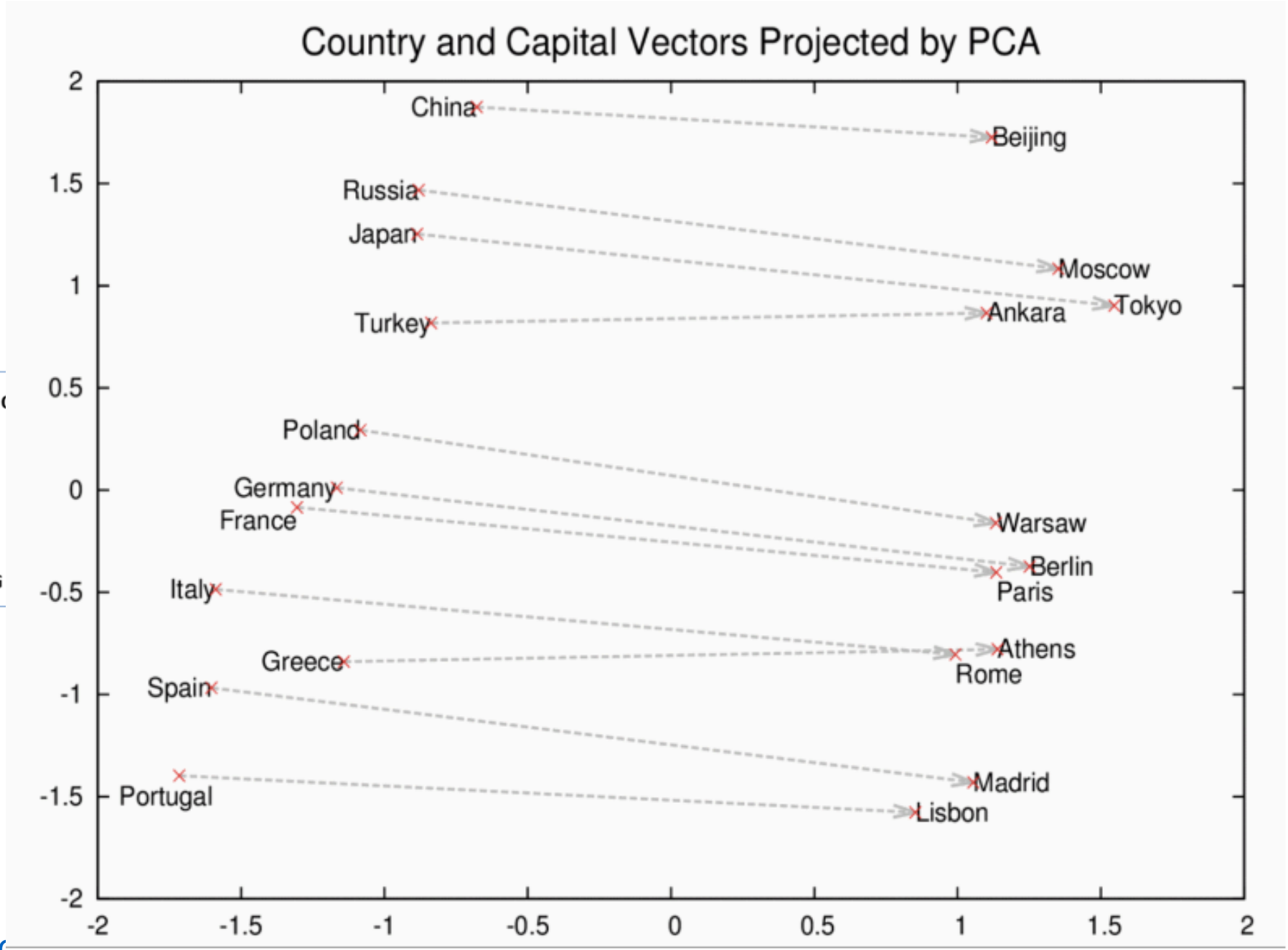
Syntactic:

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

http://deeplearning4j.org/word2vec

Language Regularity





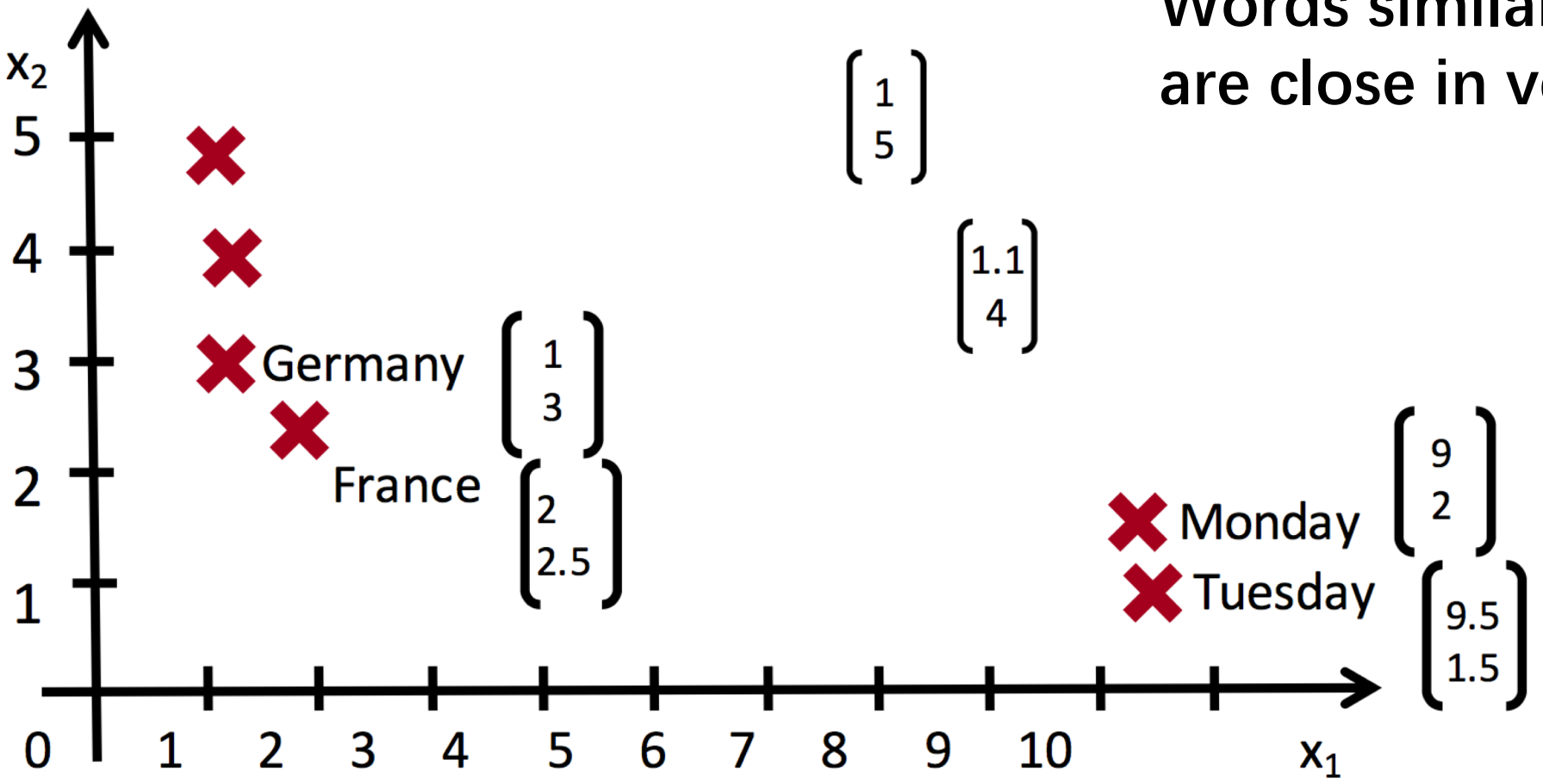
http://deeplearning4j.org/word2vec

Now, each word can be represented by

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

Deep= $\begin{bmatrix} 0.123 \\ 0.243 \\ 0.789 \\ -0.150 \\ 0.893 \\ 0.163 \\ -0.876 \end{bmatrix}$ **Learning=** $\begin{bmatrix} 0.978 \\ 0.673 \\ -0.123 \\ -0.450 \\ 0.708 \\ 0.467 \\ -0.234 \end{bmatrix}$ **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.

ELMo (Embeddings from Language Models)

- **Contextualized** word embeddings
  - Be aware of the entire input sentence
  - Not a context-independent vector

I go to the **bank** to deposit some money.
The little girl fell off the **bank**.

- Bidirectional language models
  - Train a forward LM and a backward LM

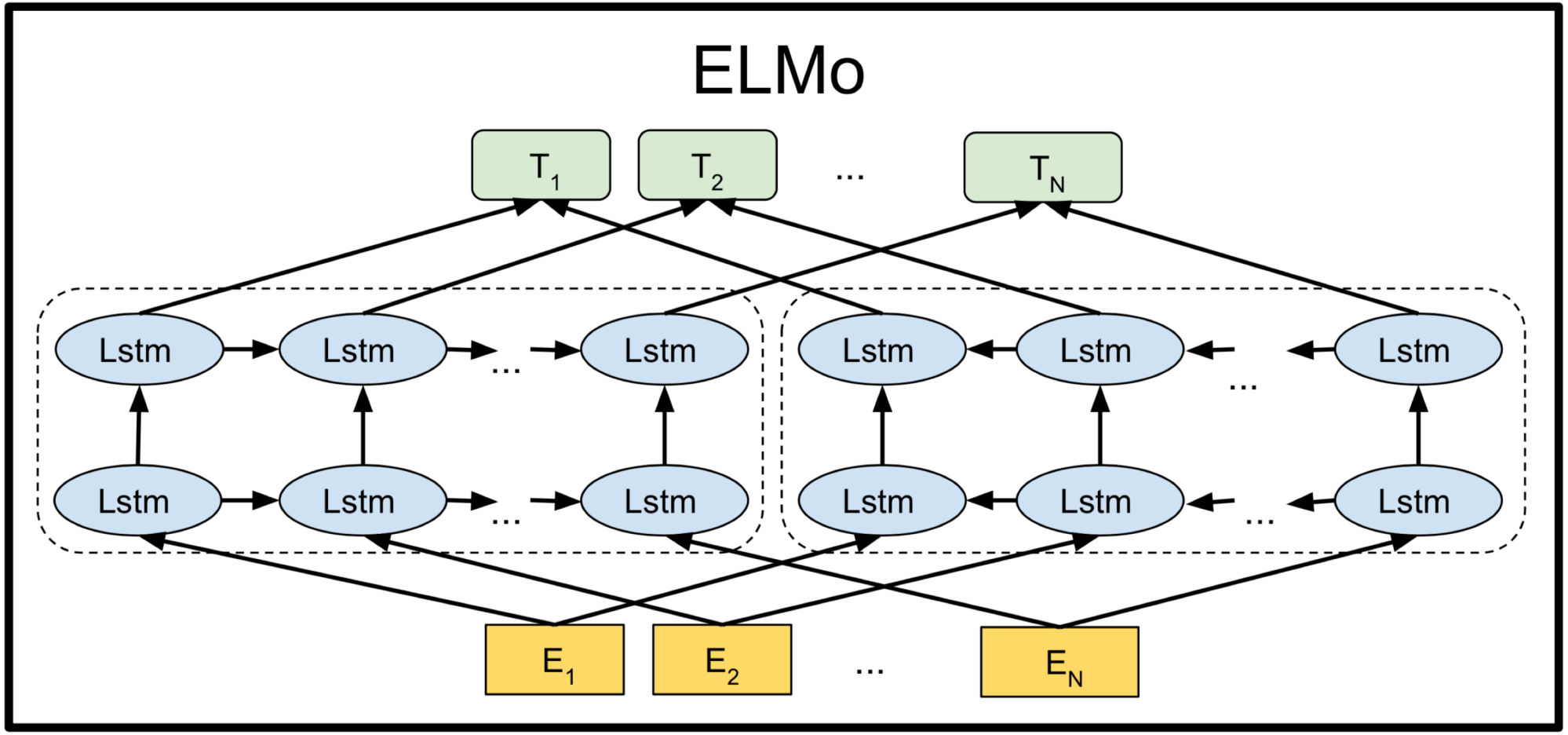
$$\text{Loss} = \sum_{k=1}^{N} \left(\log p(t_k | t_1, \ldots, t_{k-1}; \Theta_{forward}) + \log p(t_k | t_{k+1}, \ldots, t_N; \Theta_{backward}) \right)$$

- Deep representations
  - Stacked biLM
  - Combination of all representations in internal layers

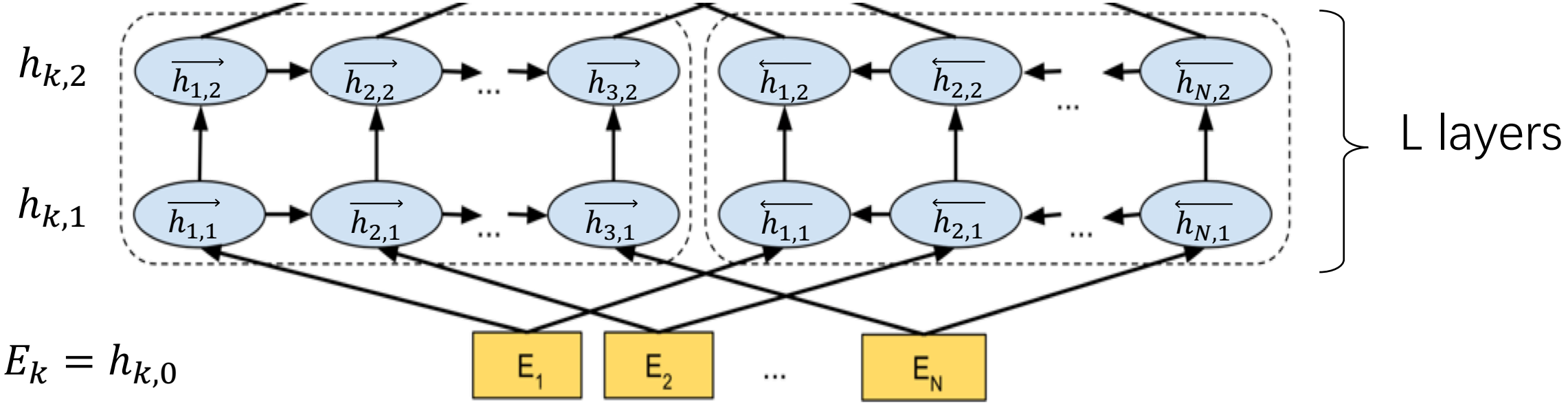
Peters M E, Neumann M, Iyyer M, et al. Deep contextualized word representations[J]. arXiv preprint arXiv:1802.05365, 2018.

ELMo (Embeddings from Language Models)

- Bidirectional language models



ELMo (Embeddings from Language Models)



- Stacked biLM layers L times
- For each word t_k, we can get 2L+1 representations:

$$R_k = \{h_{k,j} | j = 0, \ldots, L\} \qquad \text{where } h_{k,j} = \overrightarrow{h_{k,j}} \oplus \overleftarrow{h_{k,j}}$$

ELMo (Embeddings from Language Models)

ELMo is a linear combination of all internal representations

$$ELMo_k^{task} = f\left(R_k; \Theta^{task}\right) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} h_{k,j}$$

γ^{task} and s^{task} are learnable parameters

s^{task} are softmax-normalized weights and the scalar parameter γ^{task} allows the task model to scale the entire ELMo vector.

ELMo (Embeddings from Language Models)

ELMo is a linear combination of all internal representations

$$ELMo_k^{task} = f\left(R_k; \Theta^{task}\right) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} h_{k,j}$$

Using ELMo:

1. Pre-trained biLM on a large text corpus
2. Run the biLM to get $h_{k,j}$
3. Add $ELMo_k^{task}$ to an existing task specific model:
4. Update Θ^{task} (optional: update biLM)

ELMo (Embeddings from Language Models)

- Question answering(SQuAD), Textual entailment(SNLI), Semantic role labeling(SRL), Coreference resolution(Coref), Named entity extraction(NER), Sentiment analysis(SST-5)

Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/ relative)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 $\pm$ 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 $\pm$ 0.19	90.15	92.22 $\pm$ 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 $\pm$ 0.5	3.3 / 6.8%

ELMo (Embeddings from Language Models)

Alternate layer weighting schemes

- Regularize the ELMo weights by adding $\lambda||w||_2^2$ to the loss

Task	Baseline	Last Only	All layers	
			λ=1	λ=0.001
SQuAD	80.8	84.7	85.0	**85.2**
SNLI	88.1	89.1	89.3	**89.5**
SRL	81.6	84.1	84.6	**84.8**

ELMo (Embeddings from Language Models)

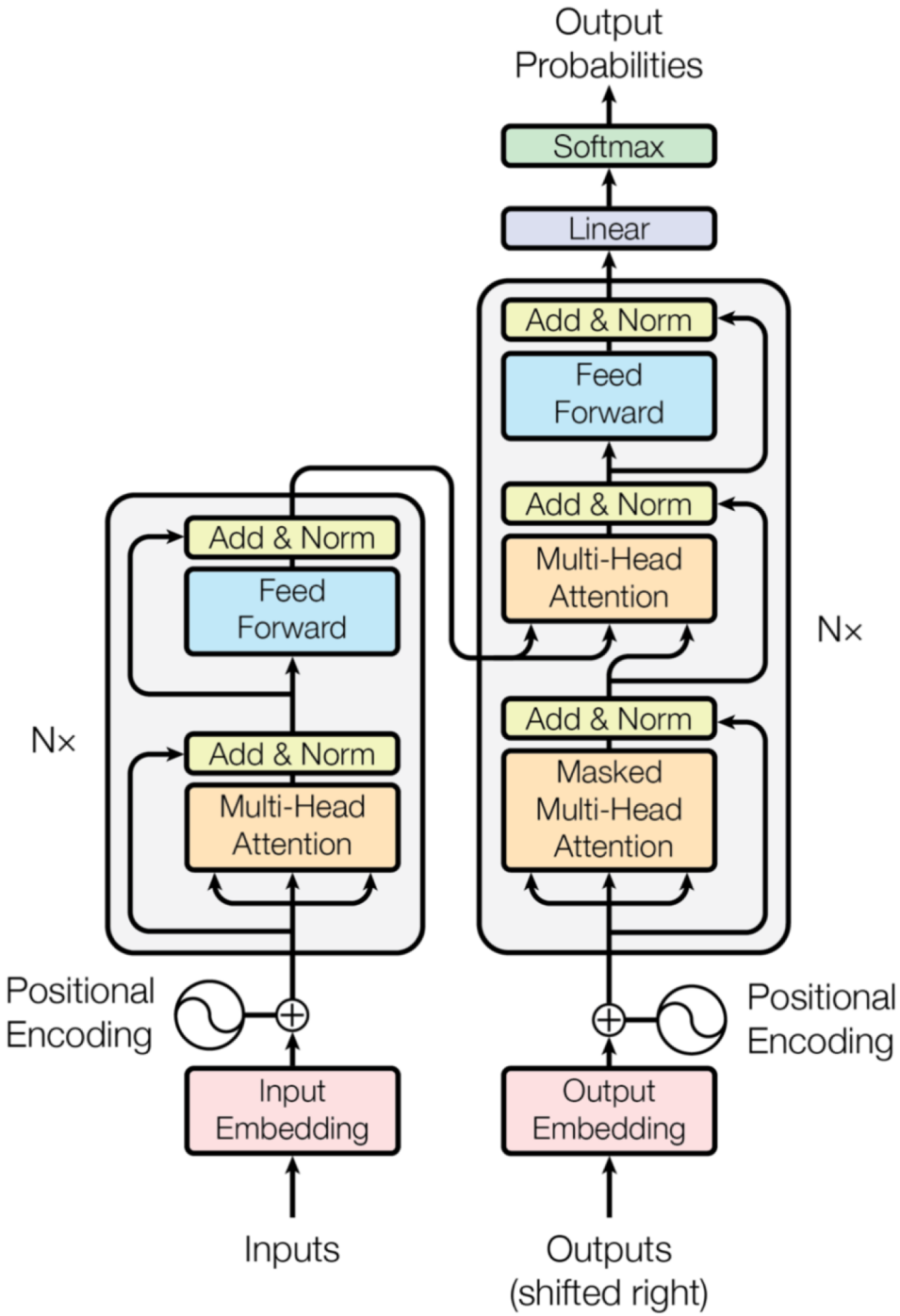
Where to use ELMo?

- We can use ELMo at both the input and output layers
  - Input: $x_k \rightarrow [x_k, ELMo_k^{task}]$
  - Output: $h_k \rightarrow [h_k, ELMo_k^{task}]$
  - Input & Output

Task	Input Only	Input & Output	Output Only
SQuAD	85.1	**85.6**	84.8
SNLI	88.9	**89.5**	88.7
SRL	**84.7**	84.3	80.9

Transformer

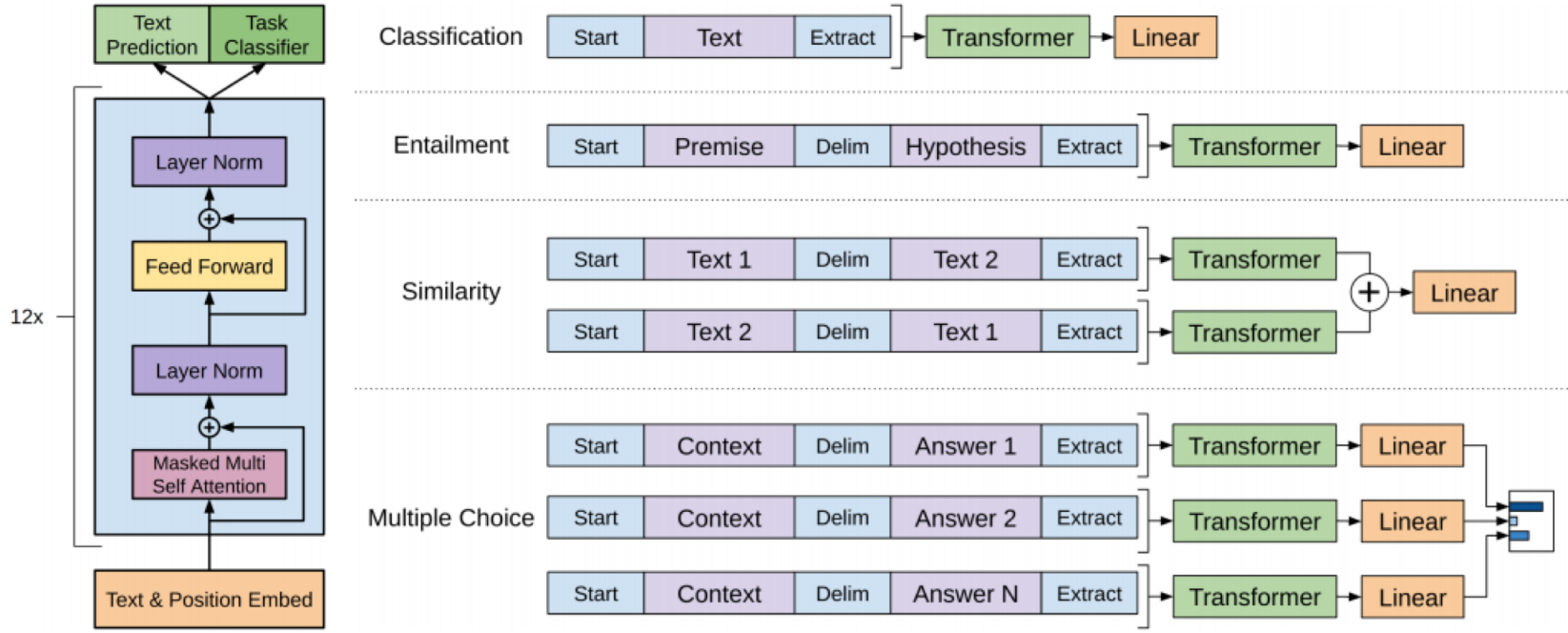
- Left: encoder; Right: decoder
- N: stack size
- Input/Output: all words in sequence
- Add&Norm: LayerNorm(x+sublayer(x))
- Multi-Head Attention
  - Encoder: self attention
  - Decoder: query over encoded hidden state
- Masked Multi-Head Attention
  - Ensure that the predictions for position i can only on the known outputs at positions less than i (recall LM)
- Word order only comes from positional embedding



Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[C]//Advances in Neural Information Processing Systems. 2017: 5998-6008.

GPT: Generative Pre-Training language model

- Transformer decoder
  - Language model pre-training objective
  - Task-specific layer for finetuning



Improving Language Understanding by Generative Pre-Training

GPT: Generative Pre-Training language model

- Language model pre-training objective

$$L_1(\mathcal{U}) = \sum_i \log P(u_i|u_{i-k}, \ldots, u_{i-1}; \Theta)$$

- Given an unsupervised corpus of tokens $\mathcal{U} = \{u_1, \ldots, u_n\}$, GPT maximize the above likelihood for unidirectional language modeling.
- Supervised fine-tuning objective

$$L_2(\mathcal{C}) = \sum_{(x,y)} \log P(y|x^1, \ldots, x^m).$$

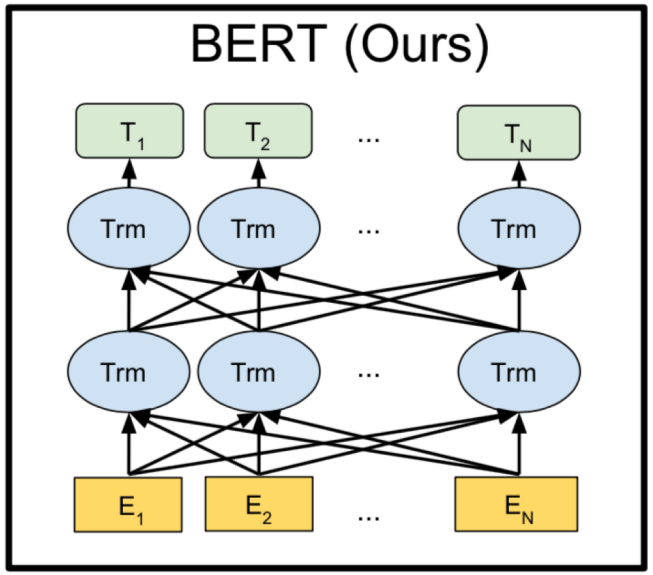
- Include language modeling as an auxiliary objective to the fine-tuning stage.

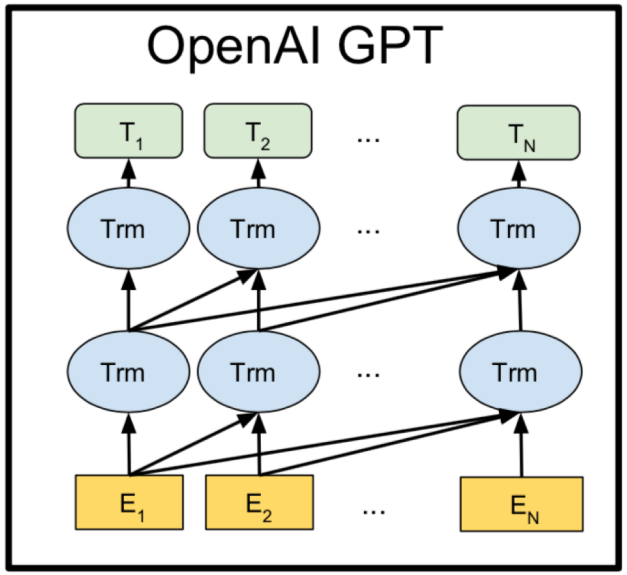
$$L_3(\mathcal{C}) = L_2(\mathcal{C}) + \lambda * L_1(\mathcal{C})$$

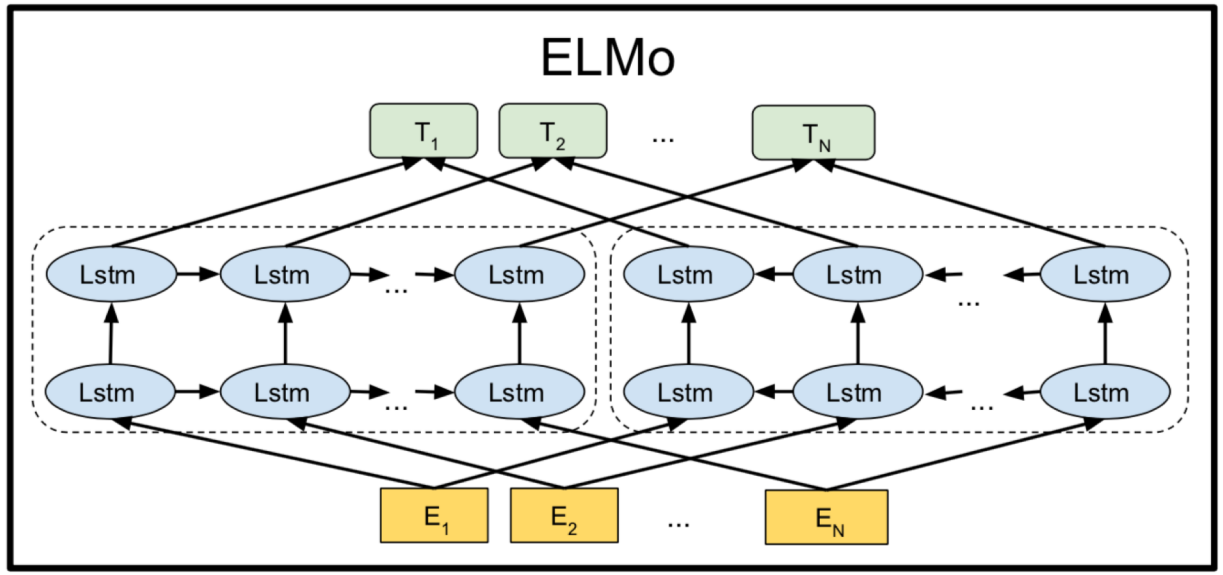
Improving Language Understanding by Generative Pre-Training

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

- Pre-training language model for downstream task(fine-tune)
  - OpenAI GPT(**unidirectional** transformer)/ELMo(shallow concatenation of independent **unidirectional** LSTM) → BERT(**bidirectional** transformer)
  - Objective: language model → **masked** language model



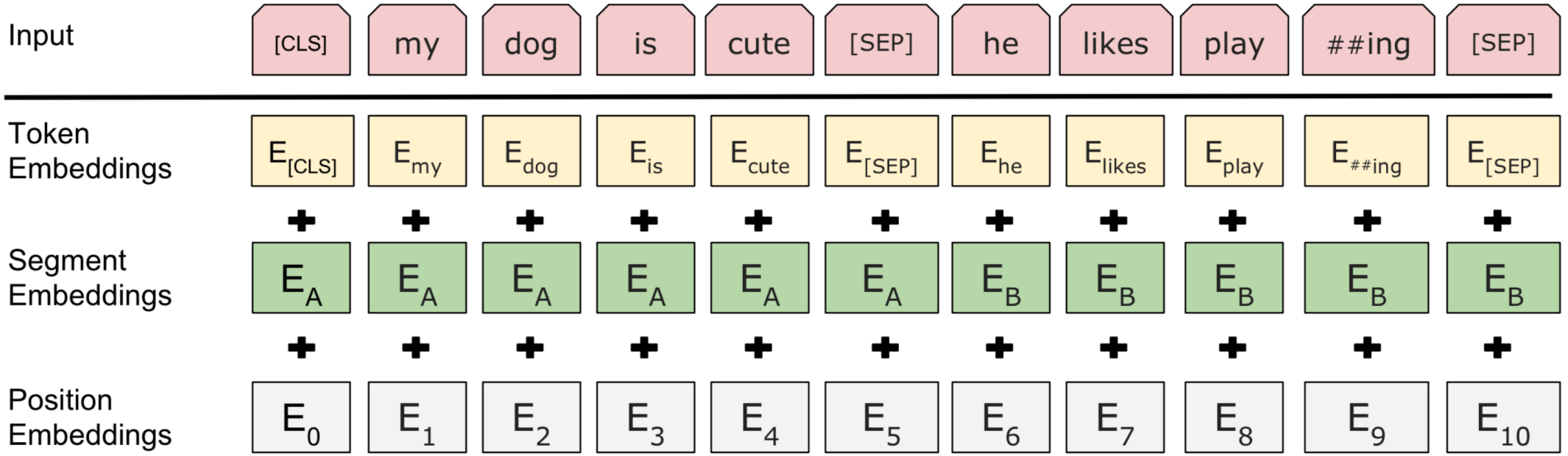




Devlin J, Chang M W, Lee K, et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding[J]. arXiv preprint arXiv:1810.04805, 2018.

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

- BERT: Transformer encoder (OpenAI GPT: Transformer decoder)
- Token embeddings: output of [CLS] for classification, [SEP] for separating two sequence
- Segment embeddings: for sentence pairs



BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Pre-training task

- Task #1: Masked LM
  - Masking some percentage(15%) of the input tokens at random, and then predicting only those masked tokens.
  - The final hidden vectors corresponding to the mask tokens are fed into an output softmax over the vocabulary, as in a standard LM.
  - Gap between pre-training and fine-tuning(no [MASK] token)?
    - 80% of the time: Replace the word with the [MASK] token.
    - 10% of the time: Replace the word with a random word.
    - 10% of the time: Keep the word unchanged.

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Pre-training task

- Task #2: Next Sentence Prediction
  - Need to understand the relationship between two text sentences, which is not directly captured by language modeling.
  - Pre-train a binarized next sentence prediction task:
    - 50% of the time: A+next sentence
    - 50% of the time: A+random sentence
    - 97%-98% accuracy
- The training loss is the sum of the mean masked LM likelihood and mean next sentence prediction likelihood.

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Fine-tuning

- Sequence-level classification
  - $\mathrm{P} = softmax(CW^T)$, where C denotes the output of the Transformer for [CLS], W is the **only** new parameters.
  - All of the parameters of BERT and W are fine-tuned jointly to maximize the log-probability of the correct label.
- Span prediction task
  - A start vector S and an end vector E, the output of BERT for i^{th} token T_i
  - Probability of word i being the start: $\mathrm{P}_i = softmax(S \cdot T_i)$
  - The maximum scoring span is used as the prediction, the training objective is the log-likelihood of the correct start and end positions.

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

11 tasks: GLUE(8 tasks), SQuAD v1.1, NER, SWAG

- GLUE

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

SWAG →

System	Dev	Test
ESIM+GloVe	51.9	52.7
ESIM+ELMo	59.1	59.2
$\text{BERT}_{\text{BASE}}$	81.6	-
$\text{BERT}_{\text{LARGE}}$	**86.6**	**86.3**
Human (expert)†	-	85.0
Human (5 annotations)†	-	88.0

NER ↓

System	Dev F1	Test F1
ELMo+BiLSTM+CRF	95.7	92.2
CVT+Multi (Clark et al., 2018)	-	92.6
$\text{BERT}_{\text{BASE}}$	96.4	92.4
$\text{BERT}_{\text{LARGE}}$	**96.6**	**92.8**

System	Dev EM	Dev F1	Test EM	Test F1
Leaderboard (Oct 8th, 2018)				
Human	-	-	82.3	91.2
#1 Ensemble - nlnet	-	-	86.0	91.7
#2 Ensemble - QANet	-	-	84.5	90.5
#1 Single - nlnet	-	-	83.5	90.1
#2 Single - QANet	-	-	82.5	89.3
Published				
BiDAF+ELMo (Single)	-	85.8	-	-
R.M. Reader (Single)	78.9	86.3	79.5	86.6
R.M. Reader (Ensemble)	81.2	87.9	82.3	88.5
Ours				
$\text{BERT}_{\text{BASE}}$ (Single)	80.8	88.5	-	-
$\text{BERT}_{\text{LARGE}}$ (Single)	84.1	90.9	-	-
$\text{BERT}_{\text{LARGE}}$ (Ensemble)	85.8	91.8	-	-
$\text{BERT}_{\text{LARGE}}$ (Sgl.+TriviaQA)	**84.2**	**91.1**	**85.1**	**91.8**
$\text{BERT}_{\text{LARGE}}$ (Ens.+TriviaQA)	**86.2**	**92.2**	**87.4**	**93.2**

Table 2: SQuAD results. The BERT ensemble is 7x systems which use different pre-training checkpoints and fine-tuning seeds.

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Effect of Pre-training Tasks

- In order make a good faith attempt at strengthening the LTR system, we tried adding a randomly initialized BiLSTM on top of it for fine-tuning.

Tasks	Dev Set MNLI-m (Acc)	QNLI (Acc)	MRPC (Acc)	SST-2 (Acc)	SQuAD (F1)
$\text{BERT}_{\text{BASE}}$	84.4	88.4	86.7	92.7	88.5
No NSP	83.9	84.9	86.5	92.6	87.9
LTR & No NSP	82.1	84.3	77.5	92.1	77.8
+ BiLSTM	82.1	84.1	75.7	91.6	84.9

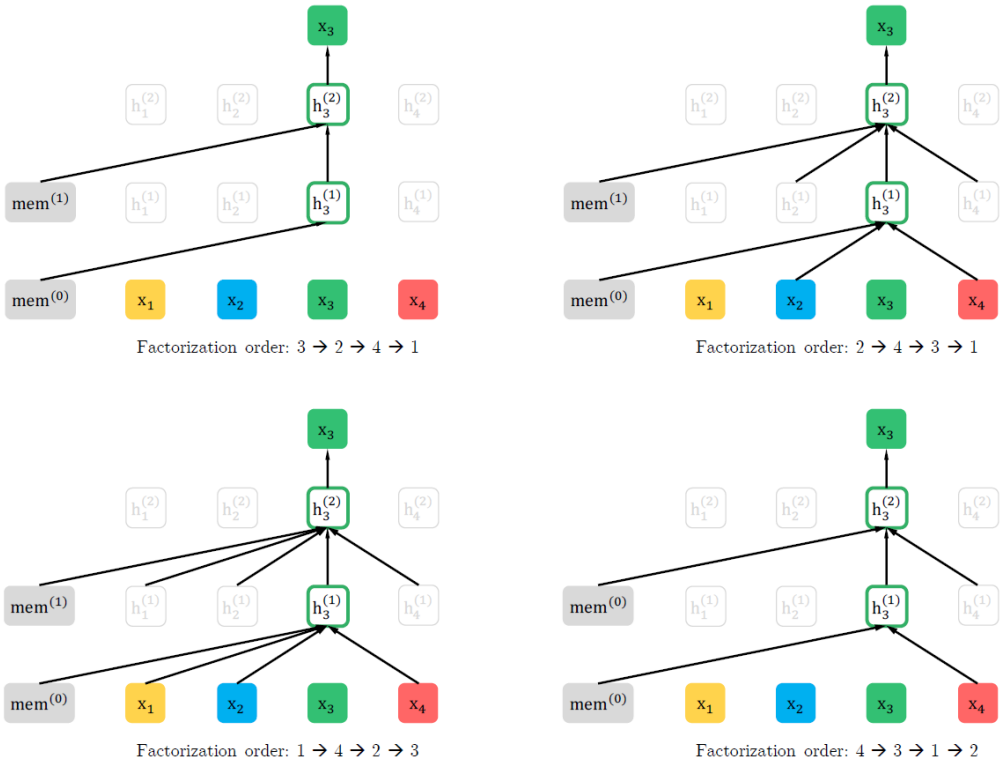
Table 5: Ablation over the pre-training tasks using the $\text{BERT}_{\text{BASE}}$ architecture. "No NSP" is trained without the next sentence prediction task. "LTR & No NSP" is trained as a left-to-right LM without the next sentence prediction, like OpenAI GPT. "+ BiLSTM" adds a randomly initialized BiLSTM on top of the "LTR + No NSP" model during fine-tuning.

XLNet: Generalized Autoregressive Pretraining for Language Understanding

- XLNet: A generalized auto-regressive method benefitting from AR language model and AE
  - Maximize the expected log likelihood of a sequence w.r.t. **all possible permutations of the factorization order**
  - Don’ t rely on data corruption and **eliminate the independence assumption** made in BERT
  - Integrate the **segment recurrence mechanism** and **relative encoding scheme** of Transformer-XL

XLNet: Generalized Autoregressive Pretraining for Language Understanding

- Permutation Language Model



- The objective of permutation language model:

$$\max_{\theta} \quad \mathbb{E}_{\mathbf{z}\sim\mathcal{Z}_T}\left[\sum_{t=1}^{T}\log p_\theta\left(x_{z_t} \mid \mathbf{x}_{\mathbf{z}<t}\right)\right]$$

- Permute the **factorization order** rather than the sequence order

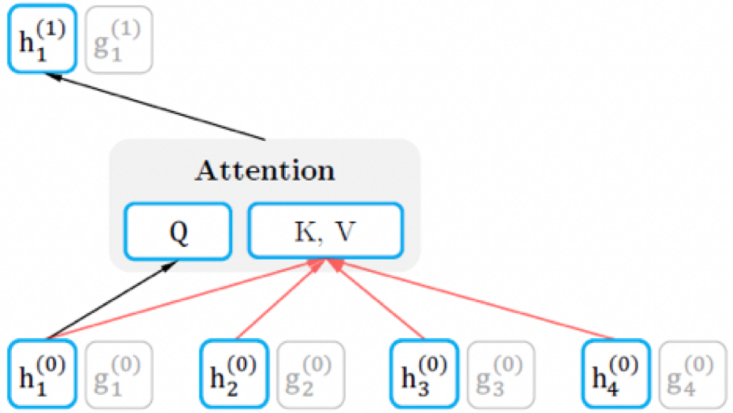
XLNet: Generalized Autoregressive Pretraining for Language Understanding

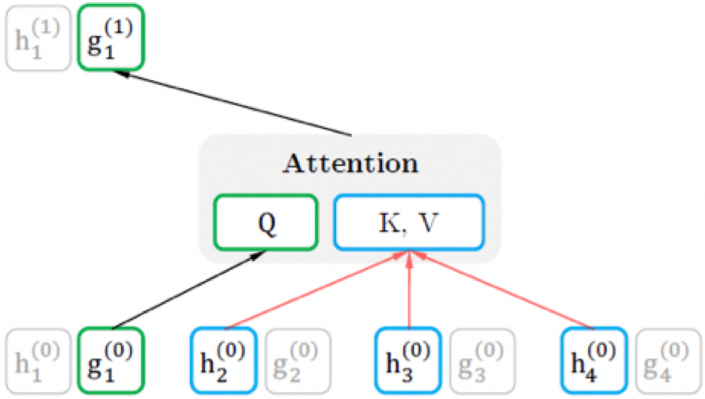
- Two-Stream Self-Attention
  - Content stream

$$h_{z_t}^{(m)} \leftarrow \text{Attention}(\mathrm{Q} = h_{z_t}^{(m-1)}, \mathrm{KV} = \mathbf{h}_{\mathbf{z}_{\leq t}}^{(m-1)}; \theta), \quad \text{(content stream: use both } z_t \text{ and } x_{z_t})$$

  - Query stream

$$g_{z_t}^{(m)} \leftarrow \text{Attention}(\mathrm{Q} = g_{z_t}^{(m-1)}, \mathrm{KV} = \mathbf{h}_{\mathbf{z}_{<t}}^{(m-1)}; \theta), \quad \text{(query stream: use } z_t \text{ but cannot see } x_{z_t})$$





XLNet: Generalized Autoregressive Pretraining for Language Understanding

- Pre-training dataset
  - BookCorpus (1.09B)
  - English Wikipedia (2.78B)
  - Giga5 (4.75B)
  - ClueWeb (4.30B)
  - Common Crawl (19.97B)

Model	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B	WNLI
Single-task single models on dev									
BERT [2]	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-
XLNet	**89.8/-**	**93.9**	**91.8**	**83.8**	**95.6**	**89.2**	**63.6**	**91.8**	-
Single-task single models on test									
BERT [10]	86.7/85.9	91.1	89.3	70.1	94.9	89.3	60.5	87.6	65.1
Multi-task ensembles on test (from leaderboard as of June 19, 2019)									
Snorkel* [29]	87.6/87.2	93.9	89.9	80.9	96.2	91.5	63.8	90.1	65.1
ALICE*	88.2/87.9	95.7	**90.7**	83.5	95.2	92.6	**68.6**	91.1	80.8
MT-DNN* [18]	87.9/87.4	96.0	89.9	**86.3**	96.5	92.7	68.4	91.1	89.0
XLNet*	**90.2/89.7**†	**98.6**†	90.3†	**86.3**	**96.8**†	**93.0**	67.8	**91.6**	**90.4**

Model	IMDB	Yelp-2	Yelp-5	DBpedia	AG	Amazon-2	Amazon-5
CNN [14]	-	2.90	32.39	0.84	6.57	3.79	36.24
DPCNN [14]	-	2.64	30.58	0.88	6.87	3.32	34.81
Mixed VAT [30, 20]	4.32	-	-	0.70	4.95	-	-
ULMFiT [13]	4.6	2.16	29.98	0.80	5.01	-	-
BERT [35]	4.51	1.89	29.32	0.64	-	2.63	34.17
XLNet	**3.79**	**1.55**	**27.80**	**0.62**	**4.49**	**2.40**	**32.26**

QA

← Classification

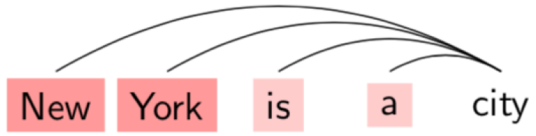
XLNet: Generalized Autoregressive Pretraining for Language Understanding

GLUE

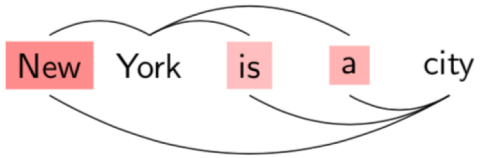
Model	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B	WNLI
Single-task single models on dev									
BERT [2]	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-
XLNet	**89.8/-**	**93.9**	**91.8**	**83.8**	**95.6**	**89.2**	**63.6**	**91.8**	-
Single-task single models on test									
BERT [10]	86.7/85.9	91.1	89.3	70.1	94.9	89.3	60.5	87.6	65.1
Multi-task ensembles on test (from leaderboard as of June 19, 2019)									
Snorkel* [29]	87.6/87.2	93.9	89.9	80.9	96.2	91.5	63.8	90.1	65.1
ALICE*	88.2/87.9	95.7	**90.7**	83.5	95.2	92.6	**68.6**	91.1	80.8
MT-DNN* [18]	87.9/87.4	96.0	89.9	**86.3**	96.5	92.7	68.4	91.1	89.0
XLNet*	**90.2/89.7**†	**98.6**†	90.3†	**86.3**	**96.8**†	**93.0**	67.8	**91.6**	**90.4**

Comparison

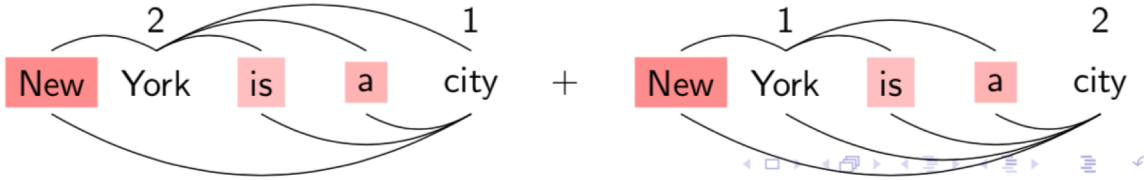
Model	Unit	Context modeling	Objective	Attention sketch
ELMo	LSTM	Unidirectional	Autoregressive	(a)
GPT	Transformer	Unidirectional	Autoregressive	(a)
BERT	Transformer	Bidirectional	Autoencoder	(b)
XLNet	Transformer	Bidirectional	Autoregressive +Autoencoder	(c)



(a)



(b)



(c)

Roadmap of this Lecture

Neural language model
Word2vec
GloVe

Recursive Neural Network
Recurrent Neural Network
Convolutional Neural Network

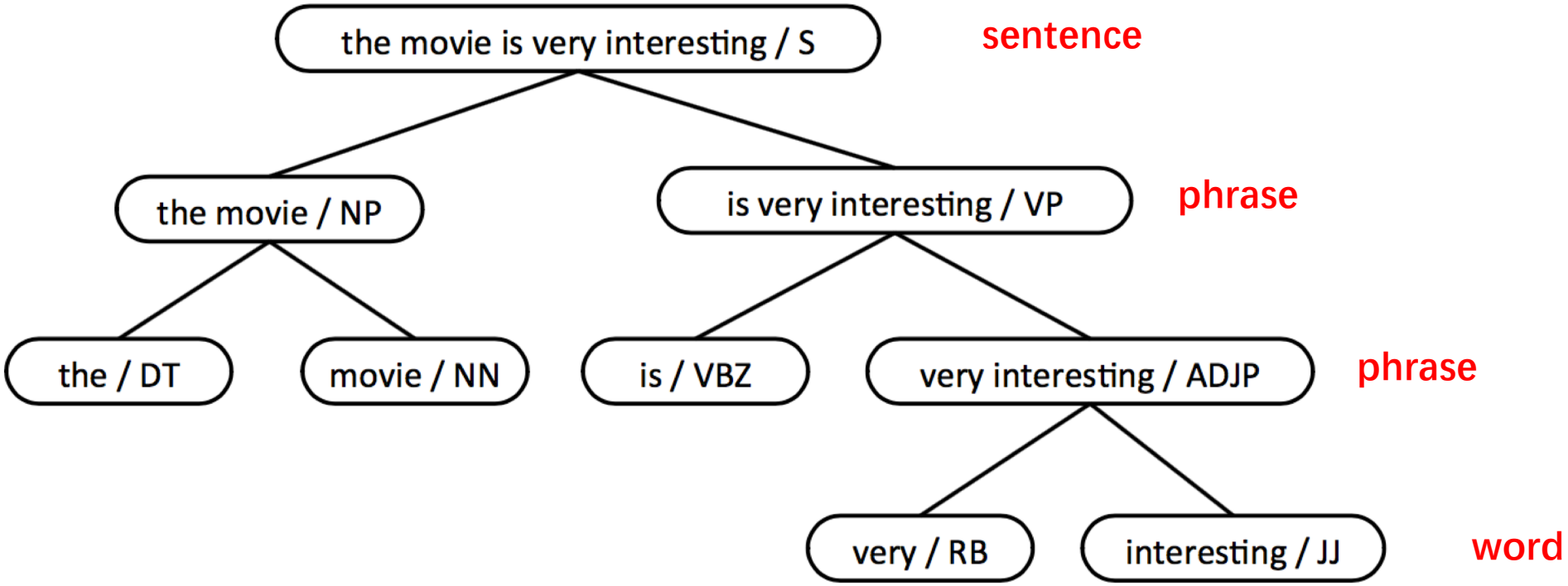
Convolutional Neural Network

Q: How many ways can you imagine to represent a sentence?

Recursive Network for NLP

Recursive Autoencoders

Sentences/phrases have composition structures!



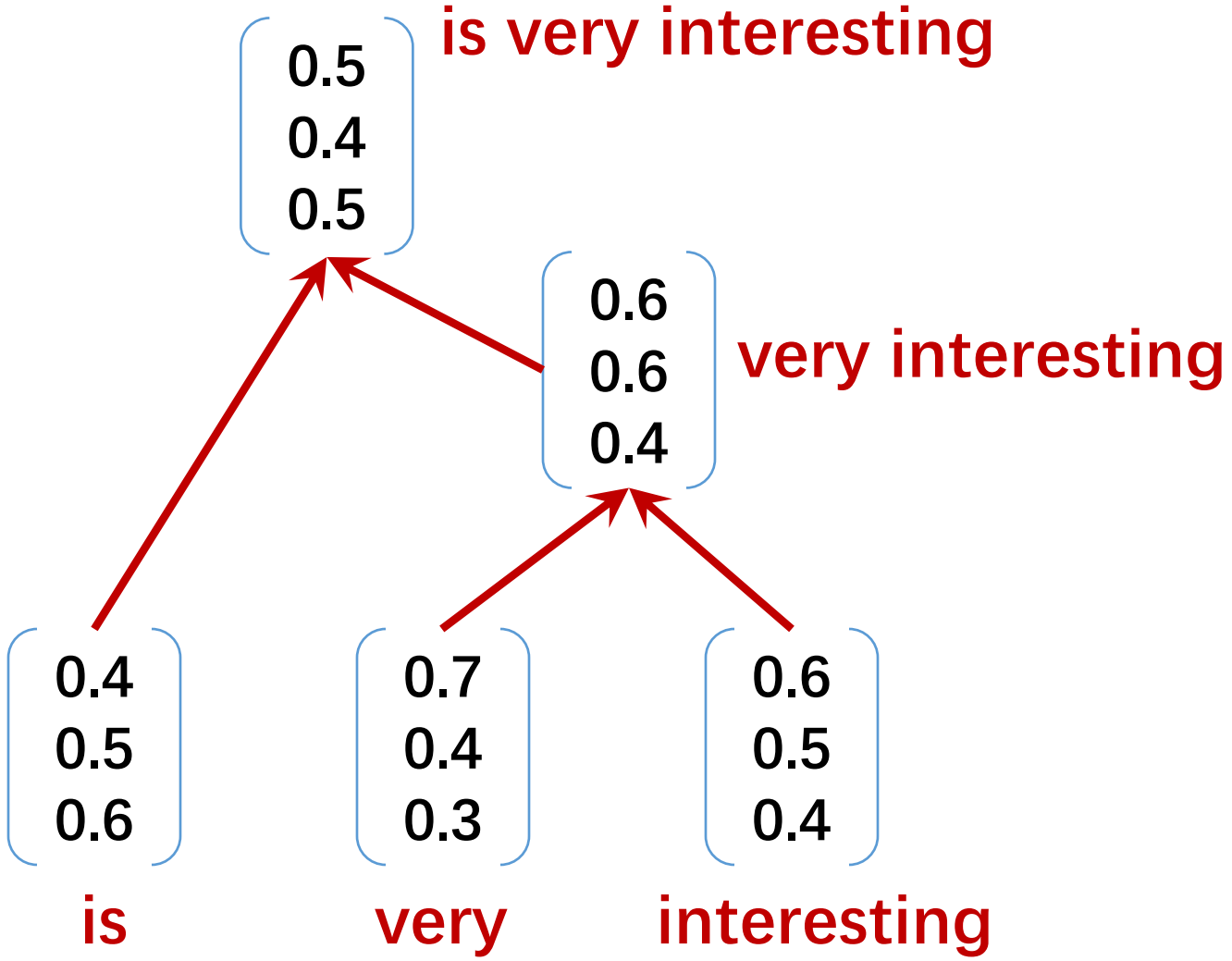
Recursive Autoencoders

Rules of Compositionality

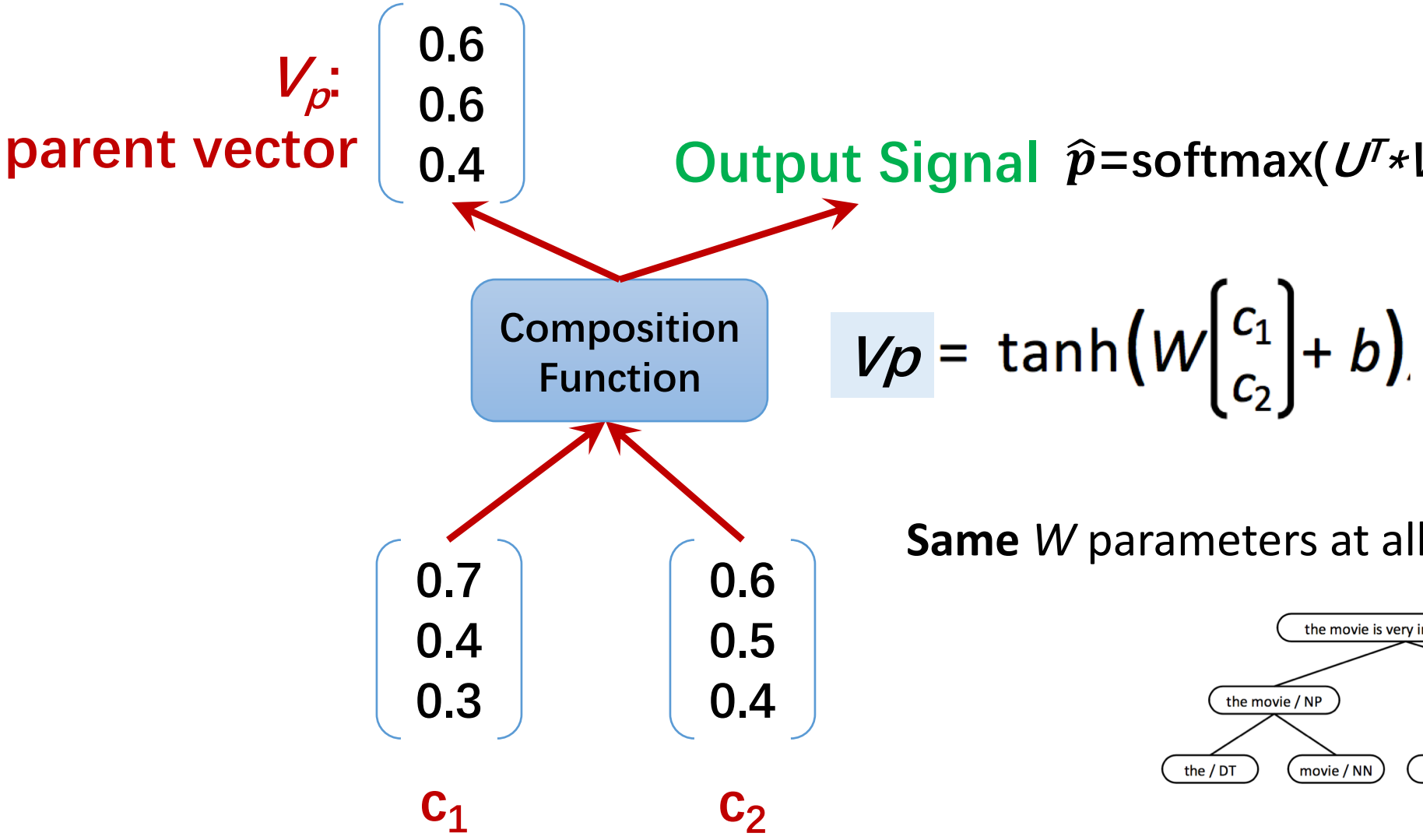
The meaning (vector) of a sentence is determined by

(1) **The meanings of its words**

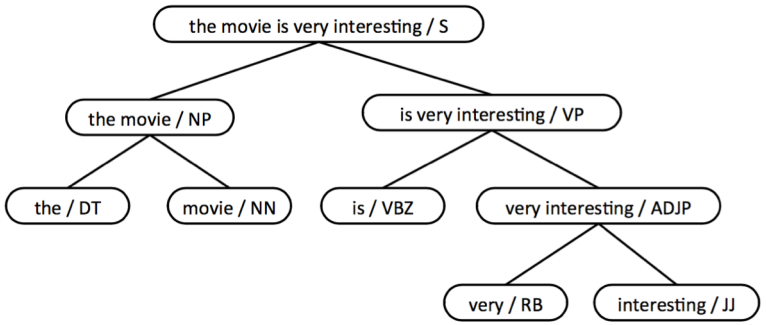
(2) **The rules that combine them**



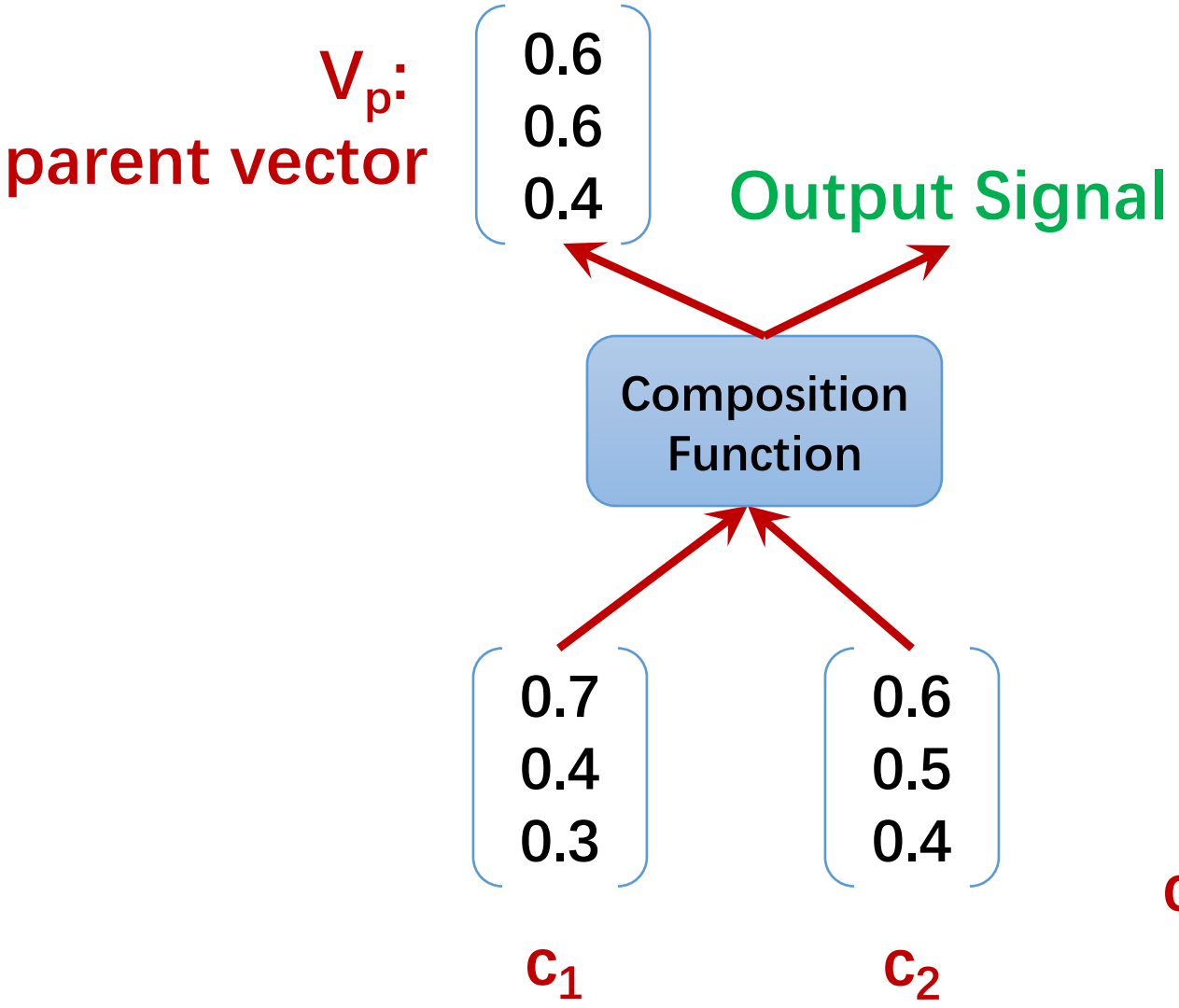
Recursive Autoencoders



Same *W* parameters at all nodes of the tree



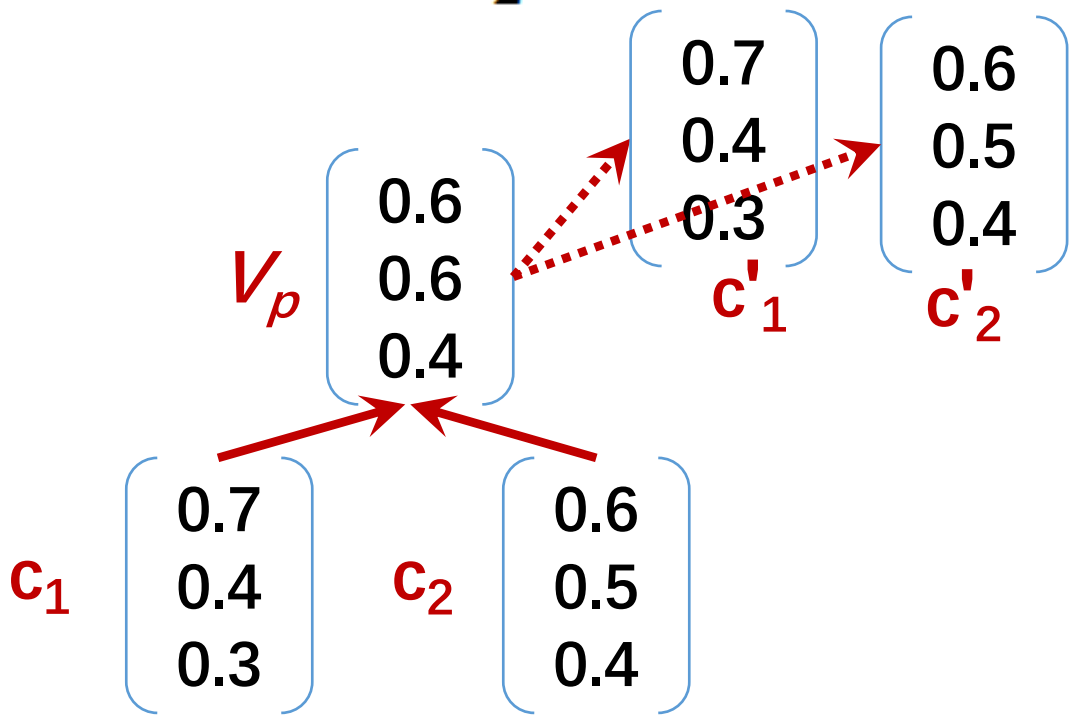
Recursive Autoencoders



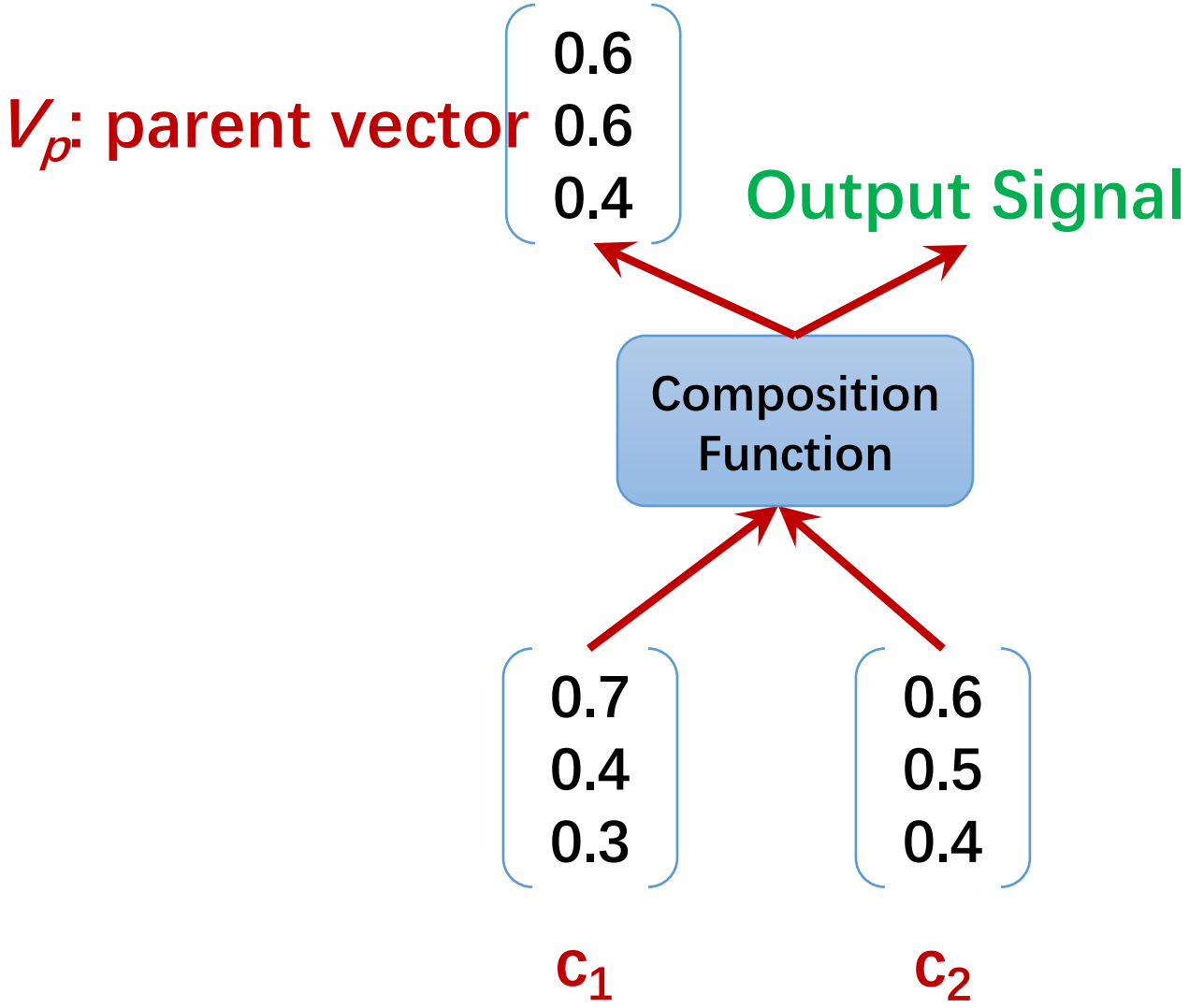
Train the model

- ✓ No supervision: minimizing reconstruction error

$$E_{rec}([c_1; c_2]) = \frac{1}{2} \left|\left|[c_1; c_2] - [c_1'; c_2']\right|\right|^2$$



Recursive Autoencoders



Train the model

✓ With supervision:
minimizing cross entropy error

$$E = \sum_{k=1}^{K} -p(k) \log \hat{p}(k)$$

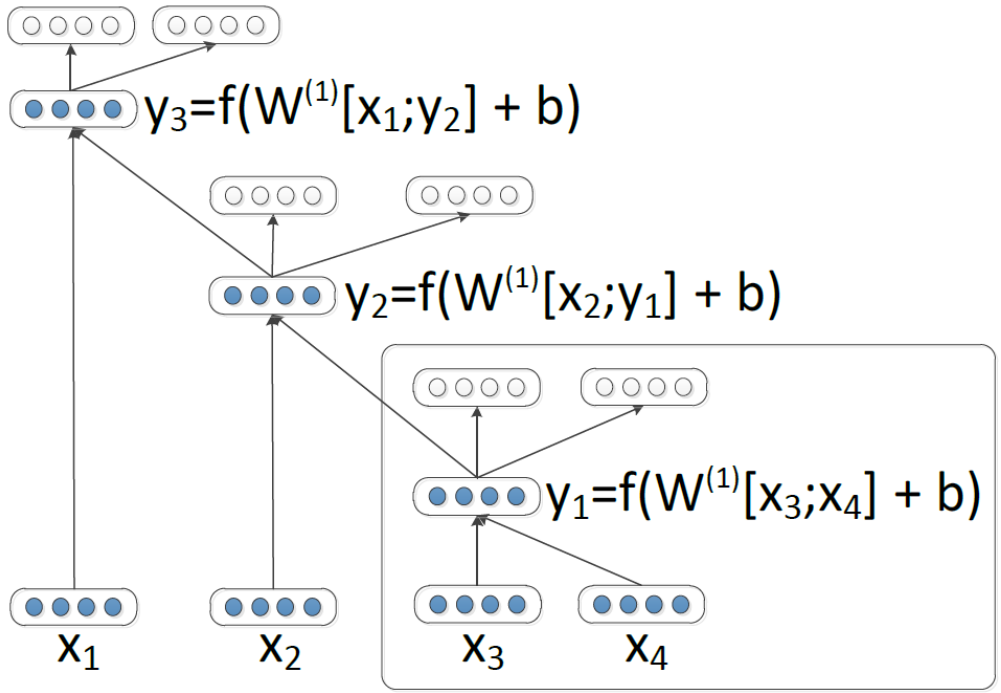
The gold distribution over class labels k

The predicted distribution based on the parent vector V_p

$$\hat{p} = \text{softmax}(U^T * V_p)$$

Recursive Autoencoder

- Given the tree structure, we can compute all the node vectors from bottom to top.
- Train by minimizing reconstruction error



$$E_{rec}([c_1; c_2]) = \frac{1}{2} \left|\left|[c_1; c_2] - [c_1'; c_2']\right|\right|^2$$

Socher et al. Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions

Semi-Supervised Recursive Autoencoders

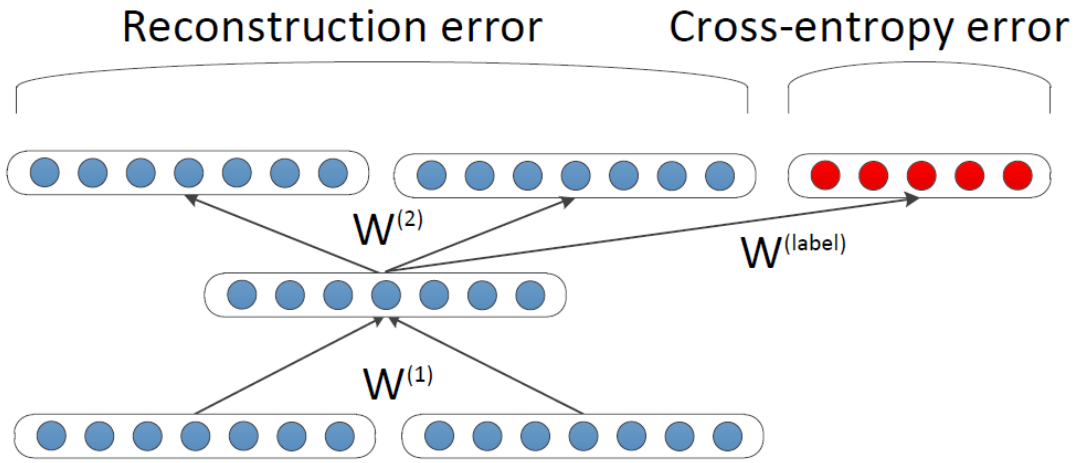
- The task label can be introduced in all nodes of the tree.

$$d(p;\theta) = \mathrm{softmax}(W^{label}p)$$

$$E_{cE}(p,t;\theta) = -\sum_{k=1}^{K} t_k \log d_k(p;\theta)$$

$$E([c_1;c_2]_s, p_s, t, \theta) =$$

$$\alpha E_{rec}([c_1;c_2]_s;\theta) + (1-\alpha)E_{cE}(p_s,t;\theta)$$



Socher et al. Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions

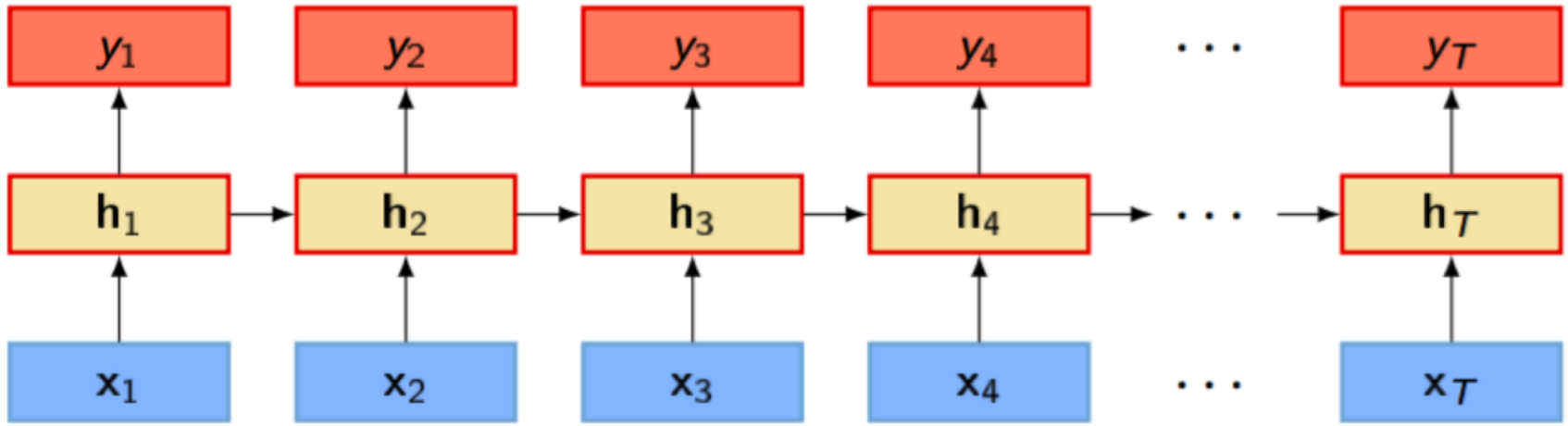
Comments on Recursive Autoencoders

- Dependent on a tree structure
  - Parser is required
- Deep structures (***h=logN***)
  - More supervision is required (at the internal nodes)

Recurrent Networks for NLP

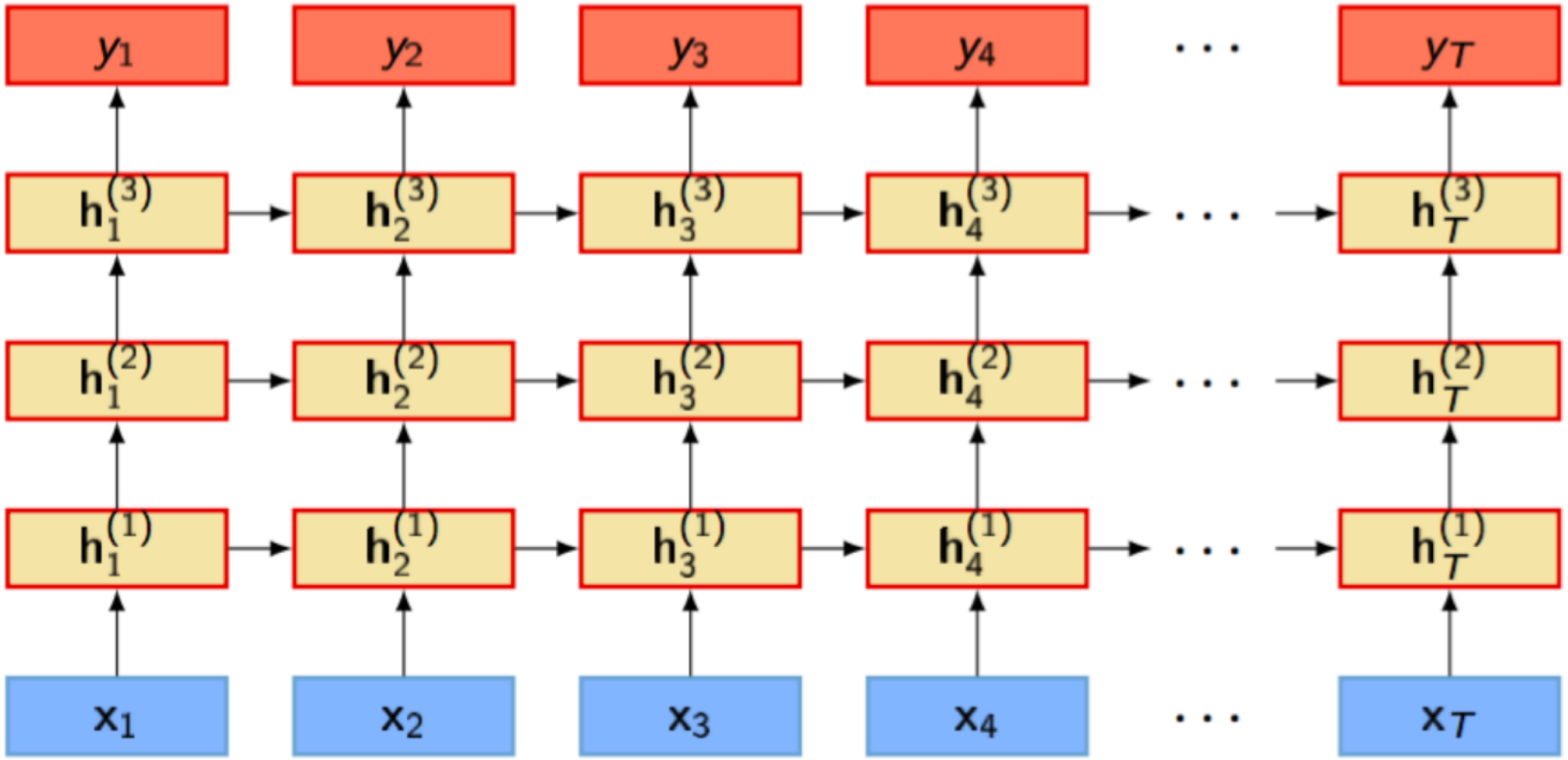
Typical Recurrent Models for NLP

- Input Sequence to **Output Sequence**
- Sequential labeling problems
  - Part-of-speech tagging
  - Semantic role labeling
  - Relation extraction
  - Named entity recognition



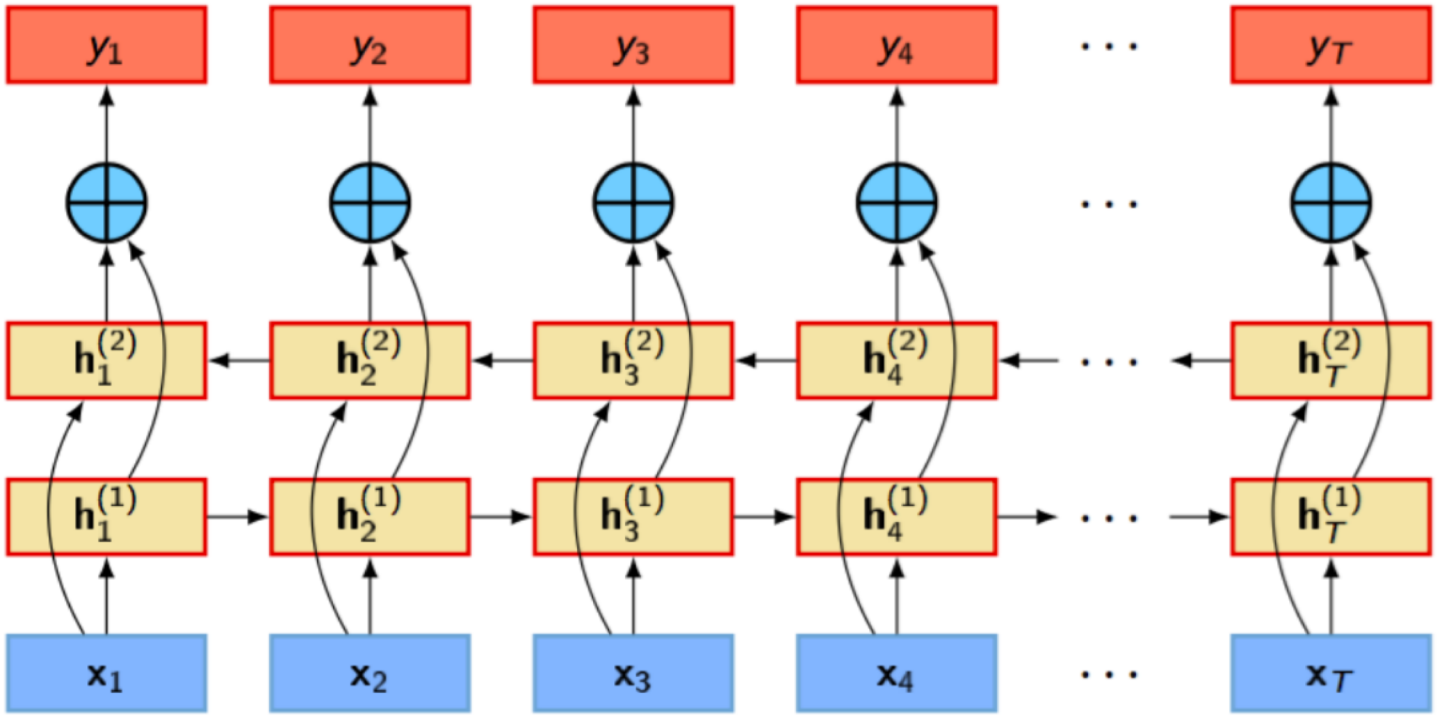
Typical Recurrent Models for NLP

- Input Sequence to **Output Sequence (stacked)**



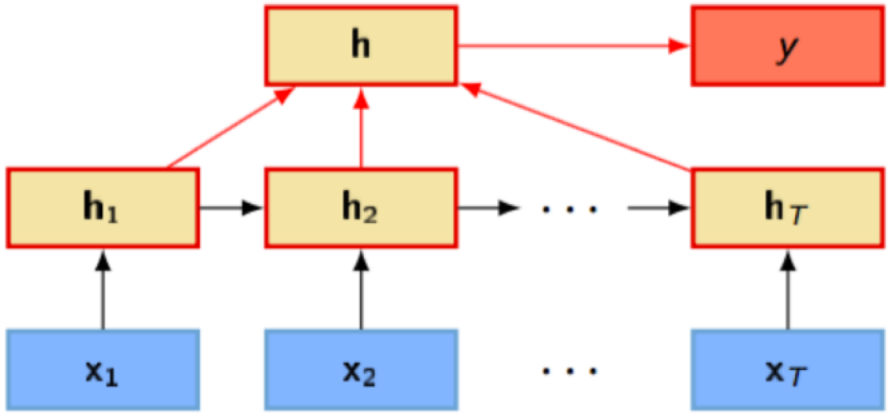
Typical Recurrent Models for NLP

- Input Sequence to **Output Sequence (Bi-directional)**

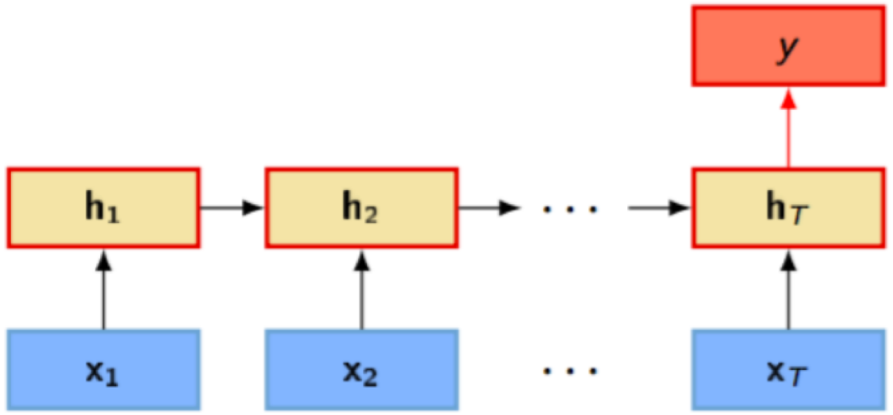


Typical Recurrent Models for NLP

- Input Sequence to **One Output Label**
  - Text classification (topic, sentiment, relation, etc.)



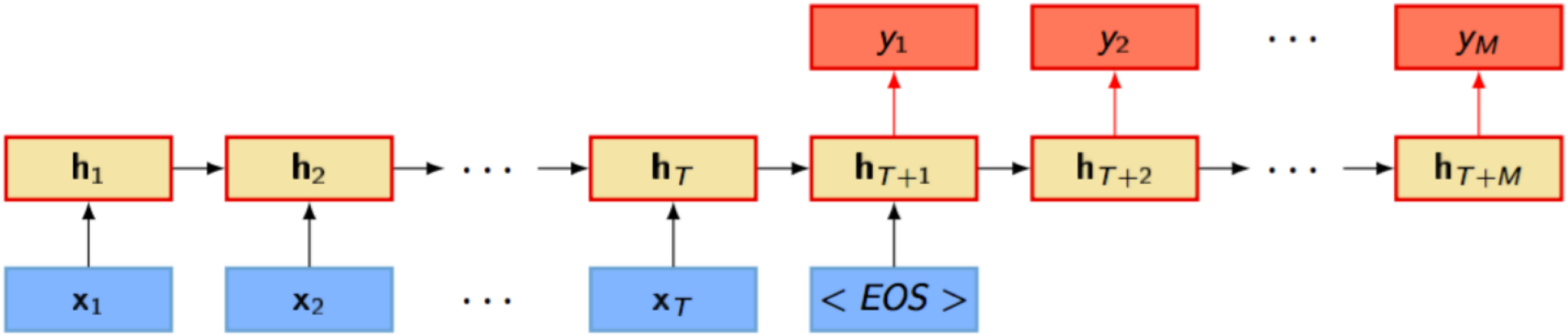
(c) Mean



(d) Last

Typical Recurrent Models for NLP

- Input Sequence to **Asynchronous** **Output Sequence**
  - Machine Translation
  - Dialogue Generation
  - Question Answering
  - Image Captioning

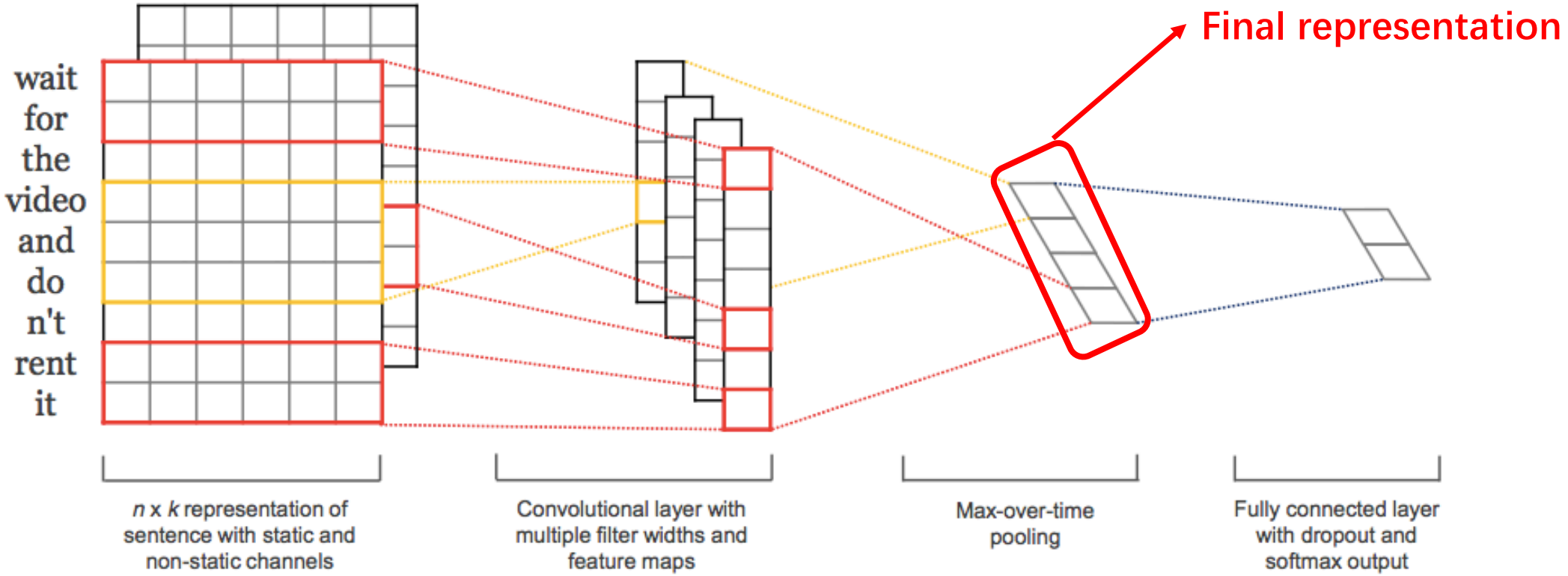


Comments on Recurrent Models

- Become a standard choice of sentence-level representation
- Very suitable for long-context modeling
  - **Attention is usually applied**
- Context Vanishing
  - The beginning context is vanishing with very long texts

Convolutional Network for NLP

Convolutional Network for NLP



Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *EMNLP*. Association for Computational Linguistics, 1746–1751.

Convolution Layer

- Word vectors: $\mathbf{x}_i \in \mathbb{R}^k$

- Sentence (concatenating words sequentially):

$$\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \ldots \oplus \mathbf{x}_n$$

- **Words in range (a vector)**: $\mathbf{x}_{i:i+j}$

- Convolutional filter (**a vector**), h is a window of h words:

$$\mathbf{w} \in \mathbb{R}^{hk}$$

Convolution Layer

- The feature value of position i:

$$c_i = f(\mathbf{w}^T \mathbf{x}_{i:i+h-1} + b)$$

- All possible windows:

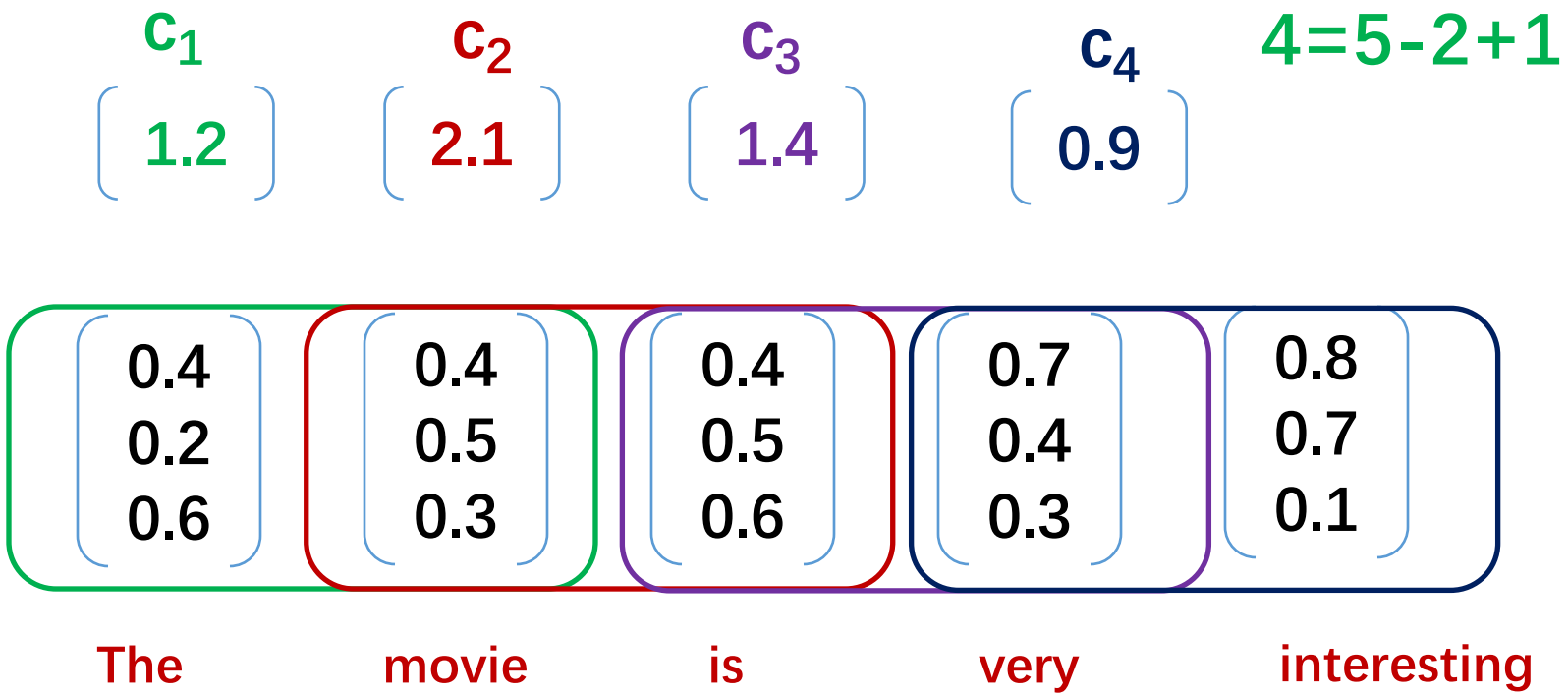
$$\{\mathbf{x}_{1:h}, \mathbf{x}_{2:h+1}, \ldots, \mathbf{x}_{n-h+1:n}\}$$

- The resulting feature map:

$$\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$$

Convolution Layer

When the window width h=2



Pooling Layer

- Feature map with **W**:

$$\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$$

- Pooling this vector to a single scalar (max, min, mean):

$$\hat{c} = \max\{\mathbf{c}\}$$

- But only **ONE** value here? Machine learning generally requires more input values

Final Softmax Layer

- Using **multiple filters W** (~100), and **multiple window h**
  - The final vector is of **N_w*N_h**
- The final feature vector

  $Z=[c_{h\_1,1}, c_{h\_1,2}, \cdots, c_{h\_1,N\_w}, c_{h\_2,1}, c_{h\_2,2}, \cdots, c_{h\_2,N\_w} \cdots]$

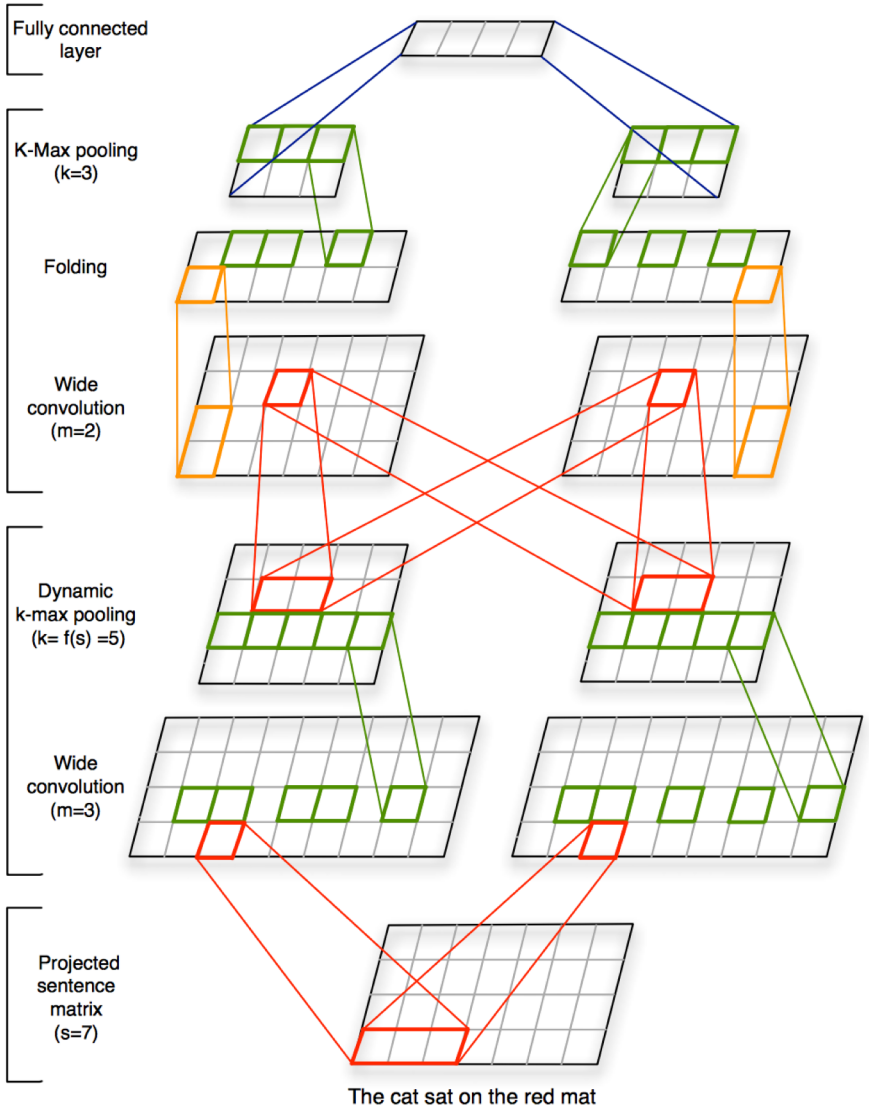
  h_i: how many types of windows

  N_w: how many filters

- The final softmax layer (or another MLP layer) takes z as input:

$$y = softmax\left(W^{(S)}z + b\right)$$

Advanced CNN Models for NLP

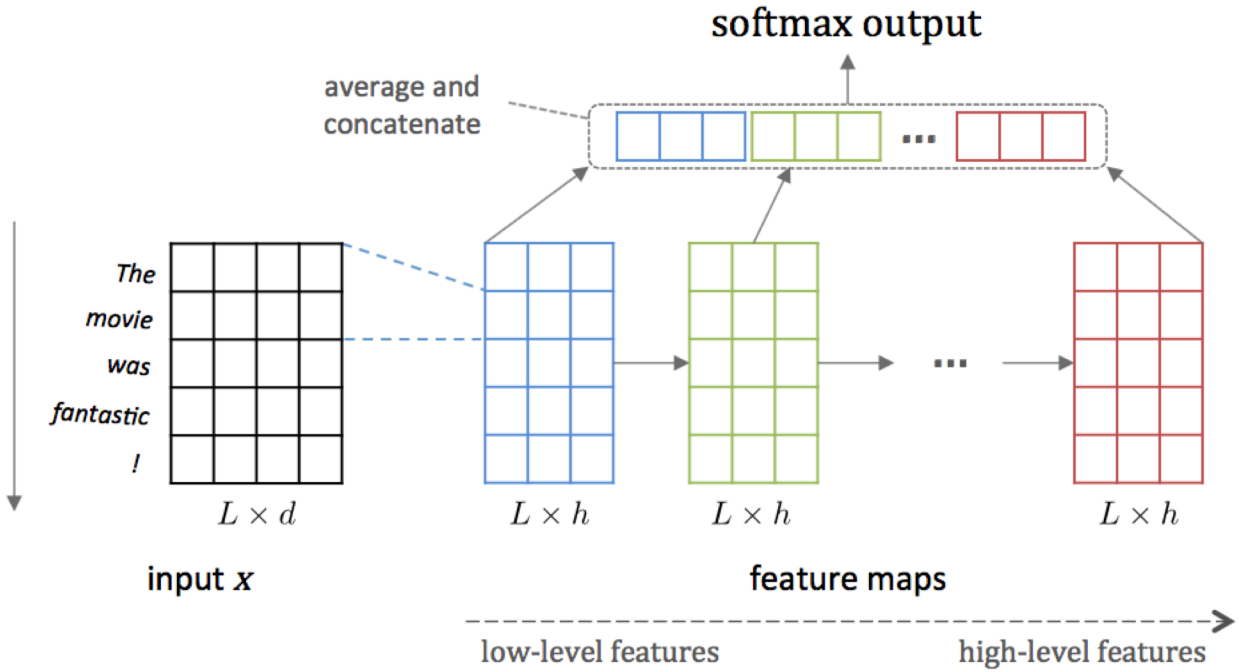


- One-dimensional convolution
- K-max pooling over time
- Multiple filters and multiple channels
- Very similar to image processing

N. Kalchbrenner, E. Grefenstette, and P. Blunsom. A Convolutional Neural Network for Modelling Sentences . In: Proceedings of ACL. 2014.

Advanced CNN Models for NLP

- Convolutional Neural Network



Model	Fine-grained		Binary	
	Dev	Test	Dev	Test
RNN		43.2		82.4
RNTN		45.7		85.4
DRNN		49.8		86.8
RLSTM		51.0		88.0
DCNN		48.5		86.9
CNN-MC		47.4		88.1
CNN	48.8	47.2	85.7	86.2
PVEC		48.7		87.8
DAN		48.2		86.8
SVM	40.1	38.3	78.6	81.3
NBoW	45.1	44.5	80.7	82.0
Ours	49.5	50.6	87.0	87.0
+ phrase labels	53.4	**51.2**	88.9	**88.6**

Lei T, Barzilay R, Jaakkola T. Molding cnns for text: non-linear, non-consecutive convolutions[J]. EMNLP, 2015.

Comments on CNN for NLP

- Suitable for **longer** text
- **Deep structures** (multiple con/pool operations) seem to be NOT that helpful
- **Not as strong as** other models such as recursive/recurrent models
- But has shown **great potential** in recent works

Fan A, Lewis M, Dauphin Y. Hierarchical Neural Story Generation[J]. arXiv preprint arXiv:1805.04833, 2018.

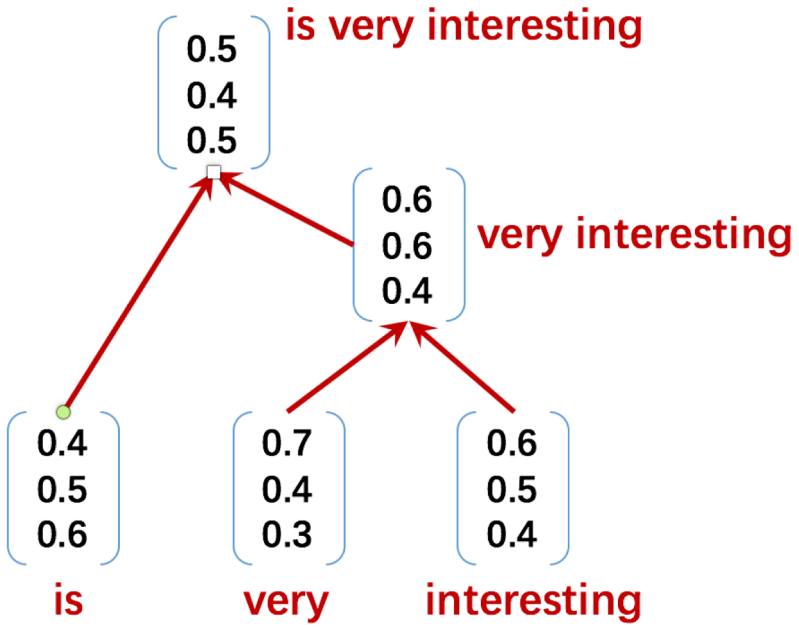
Summary

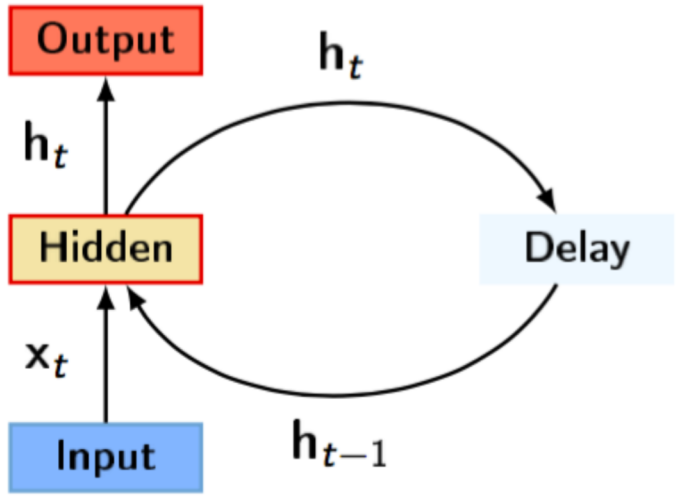
- Word representation
  - Word vectors
- Phrase/sentence representation
  - Recursive models
  - Recurrent models
  - CNN
- Document representation
  - CNN
  - Hierarchical models (RNN)
  - Hybrid models

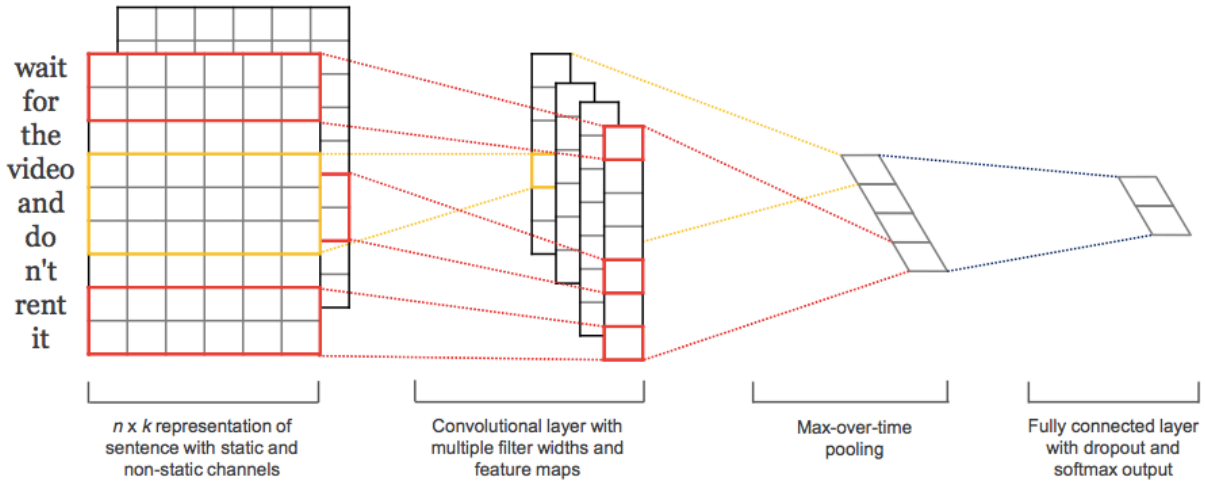
Summary

0.123
0.243
0.789
-0.150
0.893
0.163
-0.876

Deep







Advance Topics of Deep Learning for NLP

- Memory network
- Deep reinforcement learning
- Neural symbolic machine

Thanks for Attention

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