Deep learning for Natural Language Processing and Machine Translation

2015.10.16

Seung-Hoon Na

Contents

- **Introduction: Neural network, deep learning**
- **Deep learning for Natural language processing**
  - Neural network for classification
  - Word embedding
  - General architecture for sequential labeling
  - Recent advances
- **Neural machine translation**
- **Future plan**

Deep learning: Motivation

- 기계 학습 방법에서 자질 추출 (Feature extraction)
  - Handcrafted features 사용
    - 자질 추출 단계가 자동화되지는 않음
  - 지속적으로 자질 개선 필요
    - 성능 개선 및 튜닝 요구

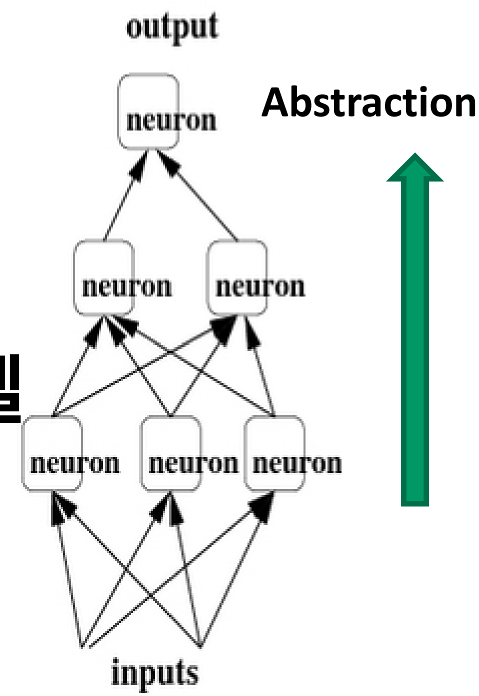
Feature vector

$$\boldsymbol{f}(\mathbf{x};\mathbf{w}) = \mathrm{argmax}_{\mathrm{y}}\ \mathbf{w} \cdot \boldsymbol{\Psi}(\mathbf{x},\mathbf{y})$$

Deep learning

- 자질 추출 단계의 제거 또는 간소화
  - 정교한 자질을 비선형 학습 과정에 내재시킴

Deep learning



- **Multi-layer perceptron**
  - 상위 은닉층은 하위 은닉층의 출력에 대한 **추상화 ➔ 비선형성 모델**
  - 다층 NN 구조로 추상 자질 내재가능 ➔ <u>자질 튜닝 절차를 단순화 시킴</u>

일반적인 기계 학습

Raw data ➔ Hand-crafted Features ➔ Trainable Classifier

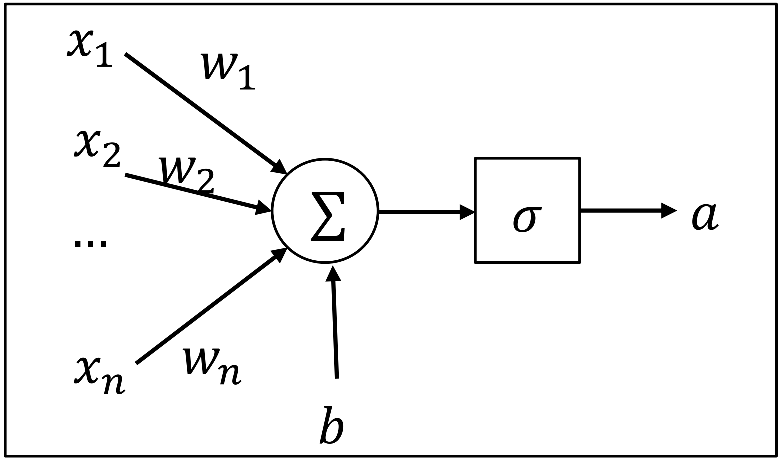
딥 러닝

Raw data ➔ Trainable Features ➔ Trainiable Classifier

Neural Network

- **A neuron**
  - A general computational unit that takes n inputs and produces a single output
  - Sigmoid & binary logistic regression unit
    - The most popular neurons
    - Takes an n-dimensional input vector x and produces the scalar activation (output) a

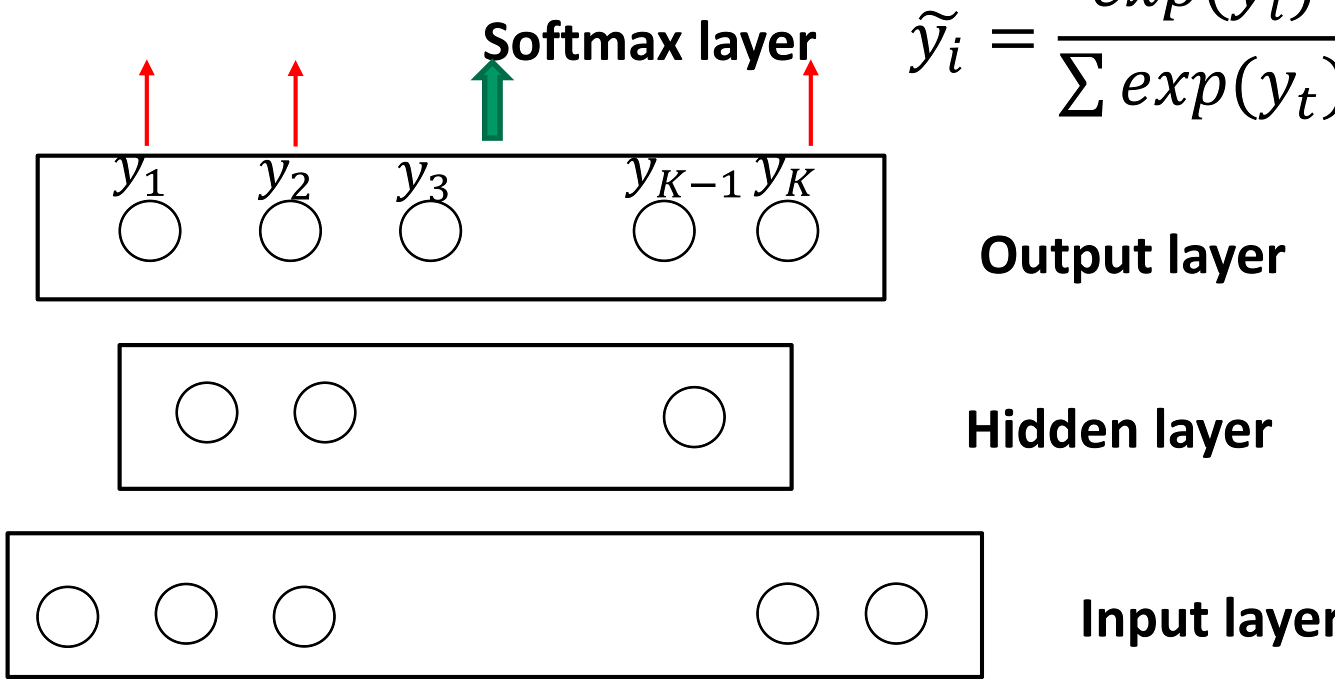
$$a = \frac{1}{1 + \exp(w^T x + b)}$$



Neural network: Setting for Classification

- Input layer: Feature values
- Output layer: Scores of labels
- Softmax layer: Normalization of output values
  - To get probabilities of output labels given input
  - K: the number of labels

$$\tilde{y}_i = \frac{exp(y_i)}{\sum exp(y_t)}$$



Neural network: Training

- **Training data:** $\mathrm{Tr} = (\boldsymbol{x}_1, g_1), \dots, (\boldsymbol{x}_N, g_N)$
  - $\boldsymbol{x}_i$: i-th input feature vector
  - $g_i \in \{1, \dots, K\}$: i-th target label

- **Objective function**
  - Negative Log-likelihood (NLL)
  - $L = -\sum_{(\boldsymbol{x},g)\in T} \log P(g|\boldsymbol{x})$

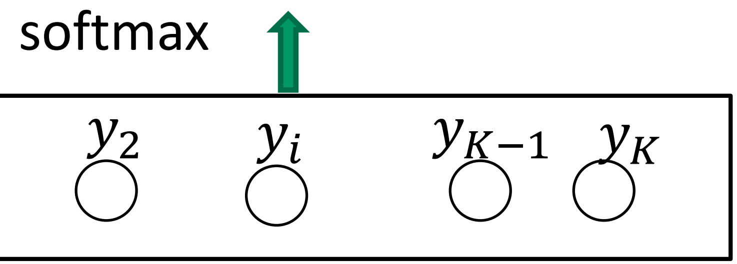
Neural network: Training

- **Stochastic gradient method**
  - 1. Randomly sample $(\boldsymbol{x}, g)$ from training data
  - 2. Define NLL for $(\boldsymbol{x}, g)$
    - $\mathrm{L} = \log P(g|\boldsymbol{x})$
  - for each weight matrix $\boldsymbol{W} \in \boldsymbol{\theta}$
  - 3. Compute gradients : $\frac{\partial L}{\partial \boldsymbol{W}}$
  - 4. Update weight matrix W: $W \leftarrow W - \eta \frac{\partial L}{\partial W}$
  - Iterate the above procedure

Neural network: Backpropagation

Output layer

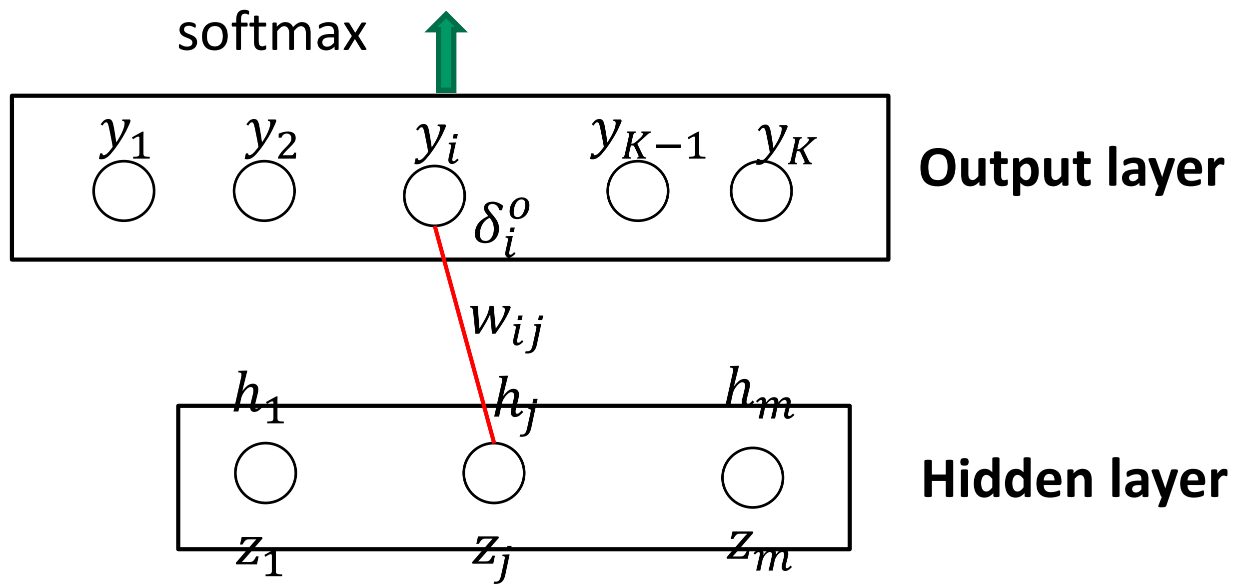
$$\mathrm{L} = \log P(g|x) = log \frac{exp(y_g)}{\sum \exp(y_i)} = \boxed{y_g - log \sum \exp(y_i)}$$



- Compute **delta** for i-th output node
  - $\delta_i^o = \frac{\partial L}{\partial y_i} = \delta(i,g) - \frac{\exp(y_i)}{\exp \sum(y_j)} = \delta(i,g) - P(i|x)$
- Vector form: $\boldsymbol{\delta}^o = \mathbf{1}_g - \begin{bmatrix} P(1|x) \\ \vdots \\ P(K|x) \end{bmatrix}$

Neural network: Backpropagation

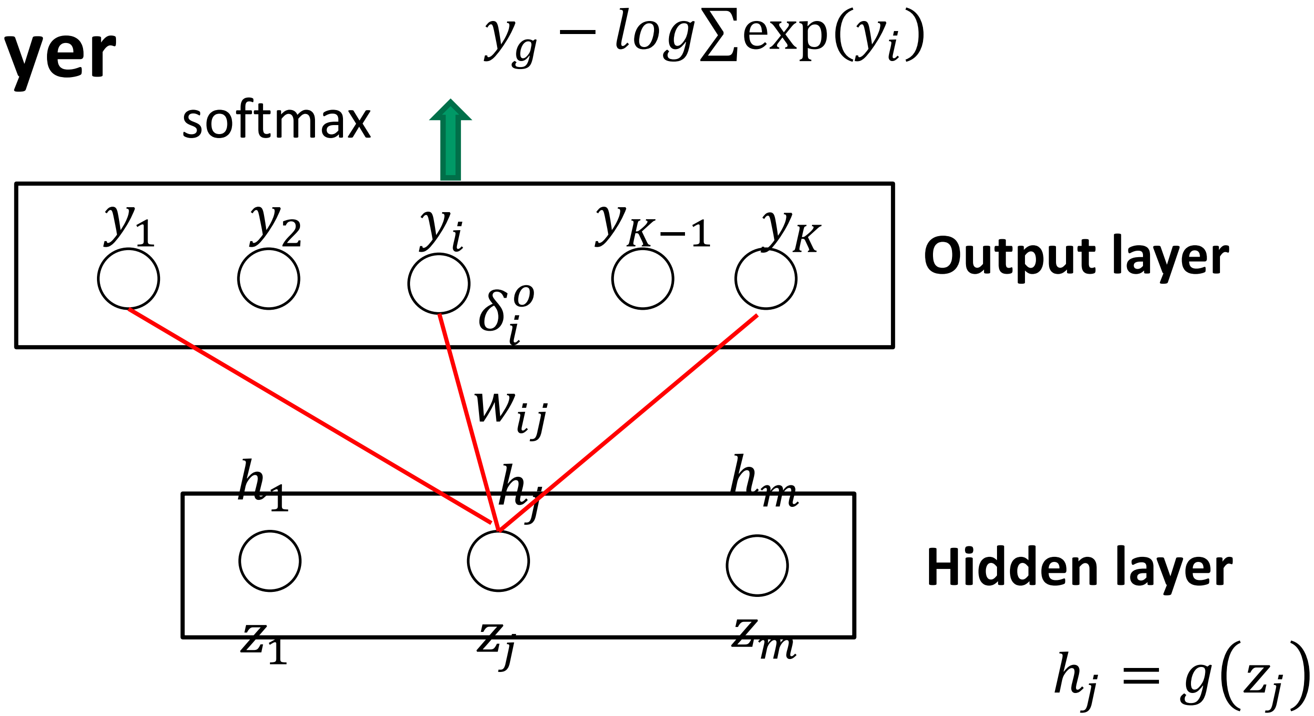
- **Output weight matrix W** $y_g - log\sum\exp(y_i)$



- Compute gradient of w_{ij} $\qquad h_j = g(z_j)$
  - $\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial y_i}\frac{\partial y_i}{\partial w_{ij}} = \delta_i^o h_j$
  - $\frac{\partial L}{\partial \boldsymbol{W}} = \boldsymbol{\delta}^o \mathbf{h}^{\mathrm{T}}$

Neural network: Backpropagation

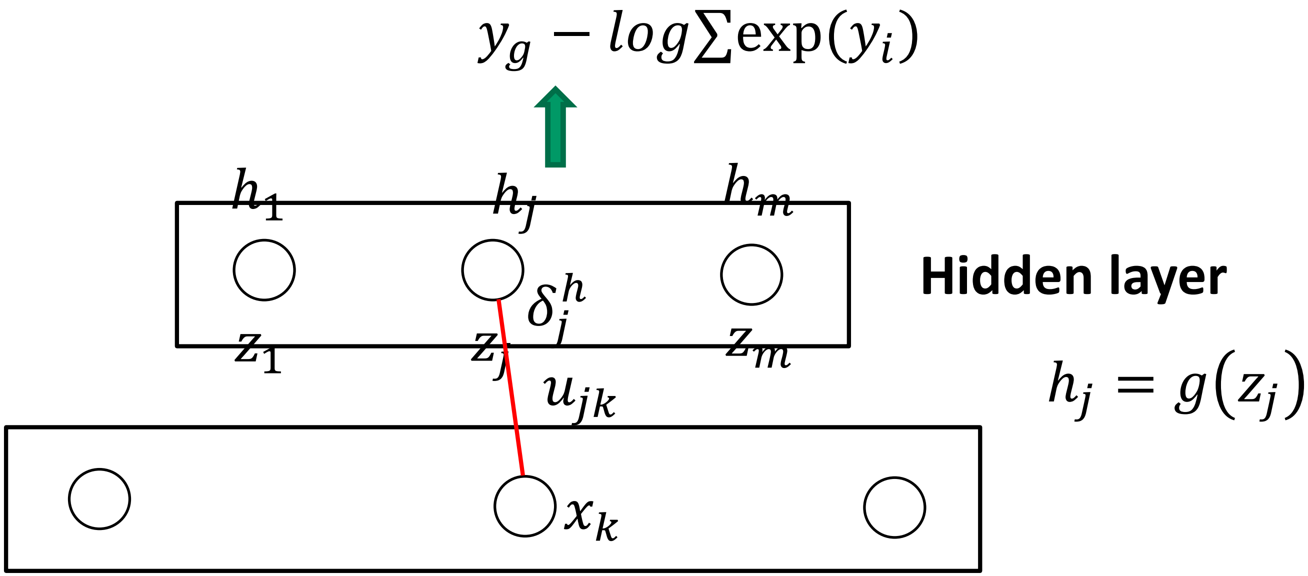
- **Hidden layer**



- Compute **delta** for j-th hidden node
  - $\delta_j^h = \frac{\partial L}{\partial z_j} = \frac{\partial L}{\partial h_j}\frac{\partial h_j}{\partial z_j} = \frac{\partial h_j}{\partial z_j}\sum_i \frac{\partial L}{\partial y_i}\frac{\partial y_i}{\partial h_j} = g'(z_j)\sum_i \delta_i^O w_{ij}$
  - $\boldsymbol{\delta}^h = g'(\mathbf{z})\circ\mathbf{W}^{\mathrm{T}}\boldsymbol{\delta}^{\mathrm{o}}$

Neural network: Backpropagation

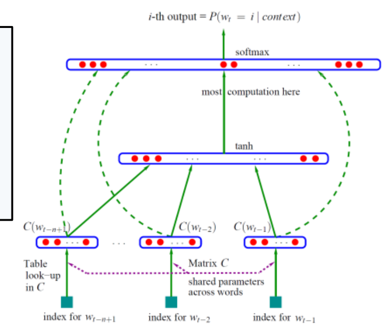
❖ Hidden weight matrix U



- Compute gradient of u_{ij}
  - $\frac{\partial L}{\partial u_{jk}} = \frac{\partial L}{\partial z_j}\frac{\partial z_j}{\partial u_{jk}} = \delta_j^h x_k$
  - $\frac{\partial L}{\partial \boldsymbol{U}} = \boldsymbol{\delta}^h \boldsymbol{x}^{\mathrm{T}}$

Deep learning for NLP

NN for Word embedding



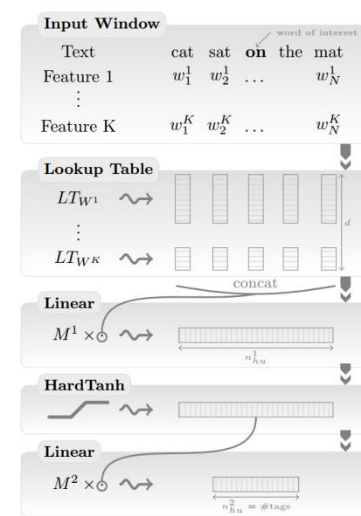
Raw corpus

Learning word embedding matrix

Unsupervised

L=

Application-specific NN



Initialize lookup table

Supervised

Annotated corpus

Application-specific neural network

➔ Lookup table is further fine-tuned

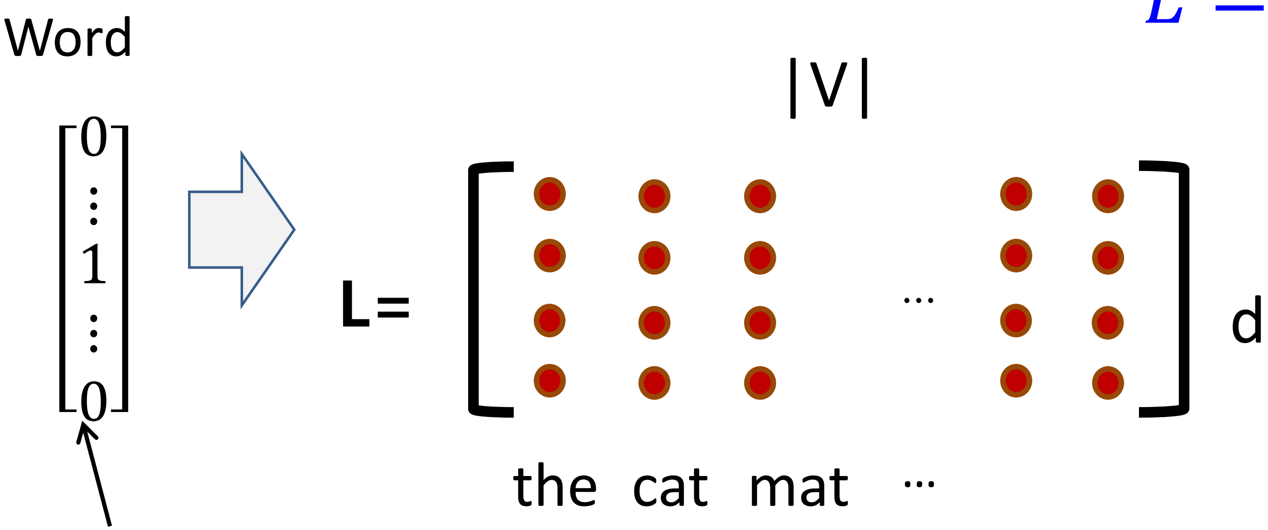
Word embedding: Distributed representation

- **Distributed representation**
  - **n-dimensional latent vector for a word**
  - Semantically similar words are closely located in vector space

$$\textit{linguistics} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

Word embedding matrix: Lookup Table

$$L = R^{d \times |V|}$$



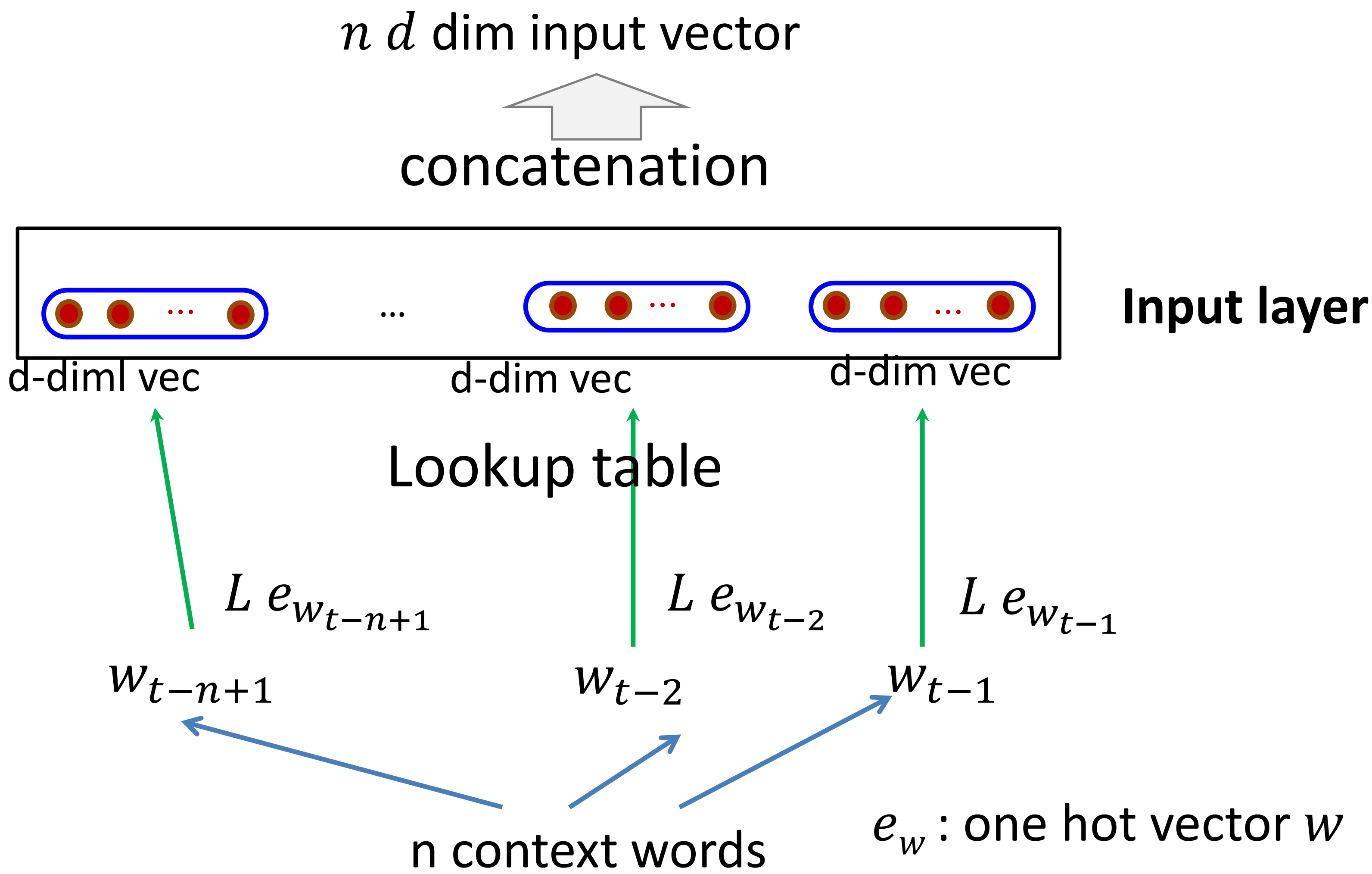
One-hot vector **e**
(|V|-dimensional vector)

$$\boldsymbol{x} = \boldsymbol{L}\ \boldsymbol{e}$$

Word vector ***x*** is obtained from one-hot vector **e** by referring to lookup table

Word embedding matrix: context ➔ input layer

- word seq $w_1 \cdots w_n$ ➔ input layer

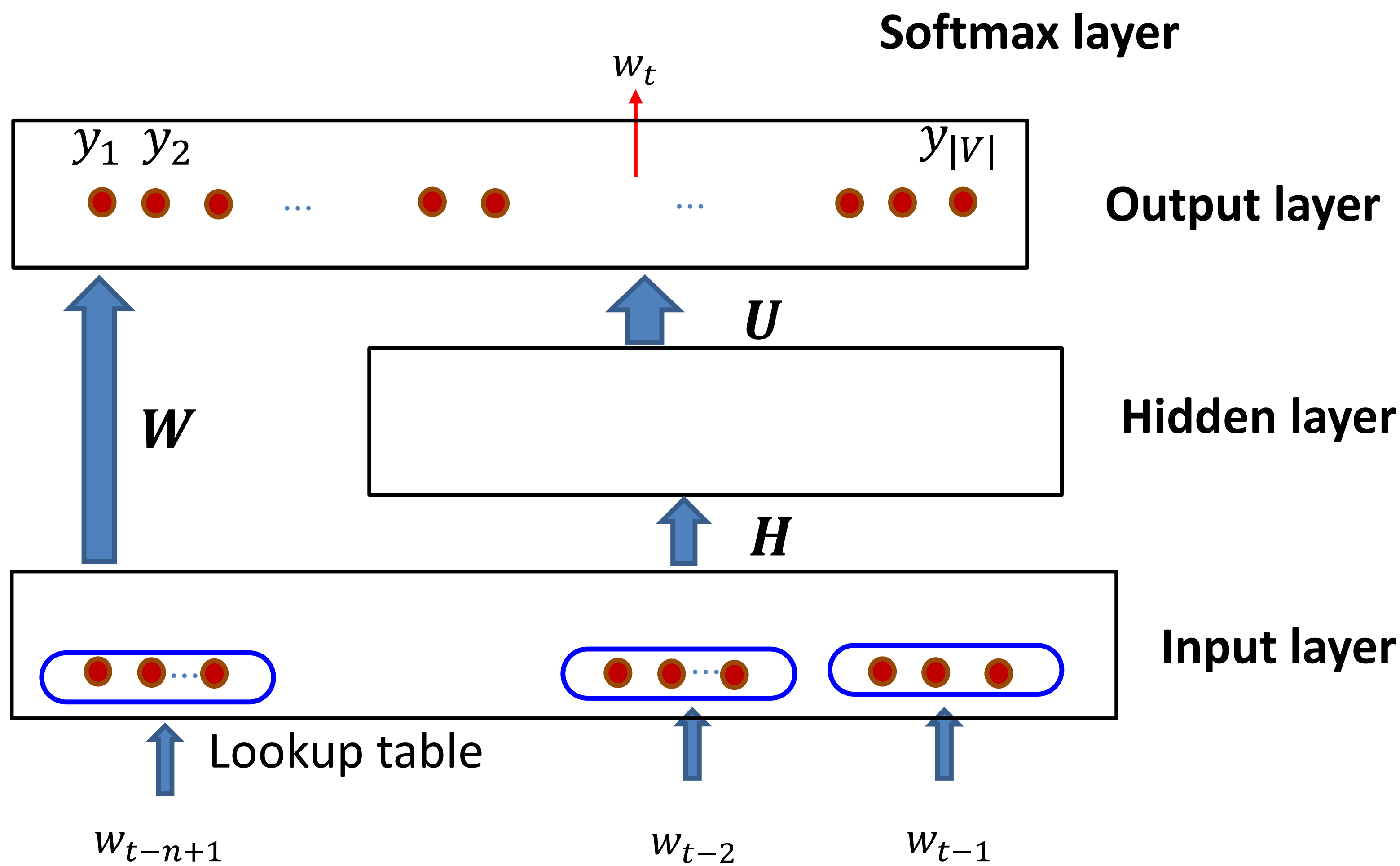


Neural Probabilistic Language Model (Bengio '03)

- **Language models**
  - The probability of a sequence of words
  - N-gram model
    - $P(w_i|w_1, \ldots, w_{i-1}) \approx P(w_i|w_{i-(n-1)}, \cdots, w_{i-1})$
- **Neural probabilistic language model**
  - Estimate $P(w_i|w_{i-(n-1)}, \cdots, w_{i-1})$ by neural networks for classification
  - $\mathbf{x}$: Concatenated input features (input layer)
  - $\boldsymbol{y} = \mathbf{U} \tanh(\mathbf{d} + \mathbf{H}\mathbf{x})$
  - $\boldsymbol{y} = \mathbf{W}\mathbf{x} + \mathbf{b} + \mathbf{U} \tanh(\mathbf{d} + \mathbf{H}\mathbf{x})$

Neural probabilistic language model

$$P(w_t|w_{t-n+1},\cdots,w_{t-1}) = \frac{exp(y_{w_t})}{\sum exp(y_t)}$$

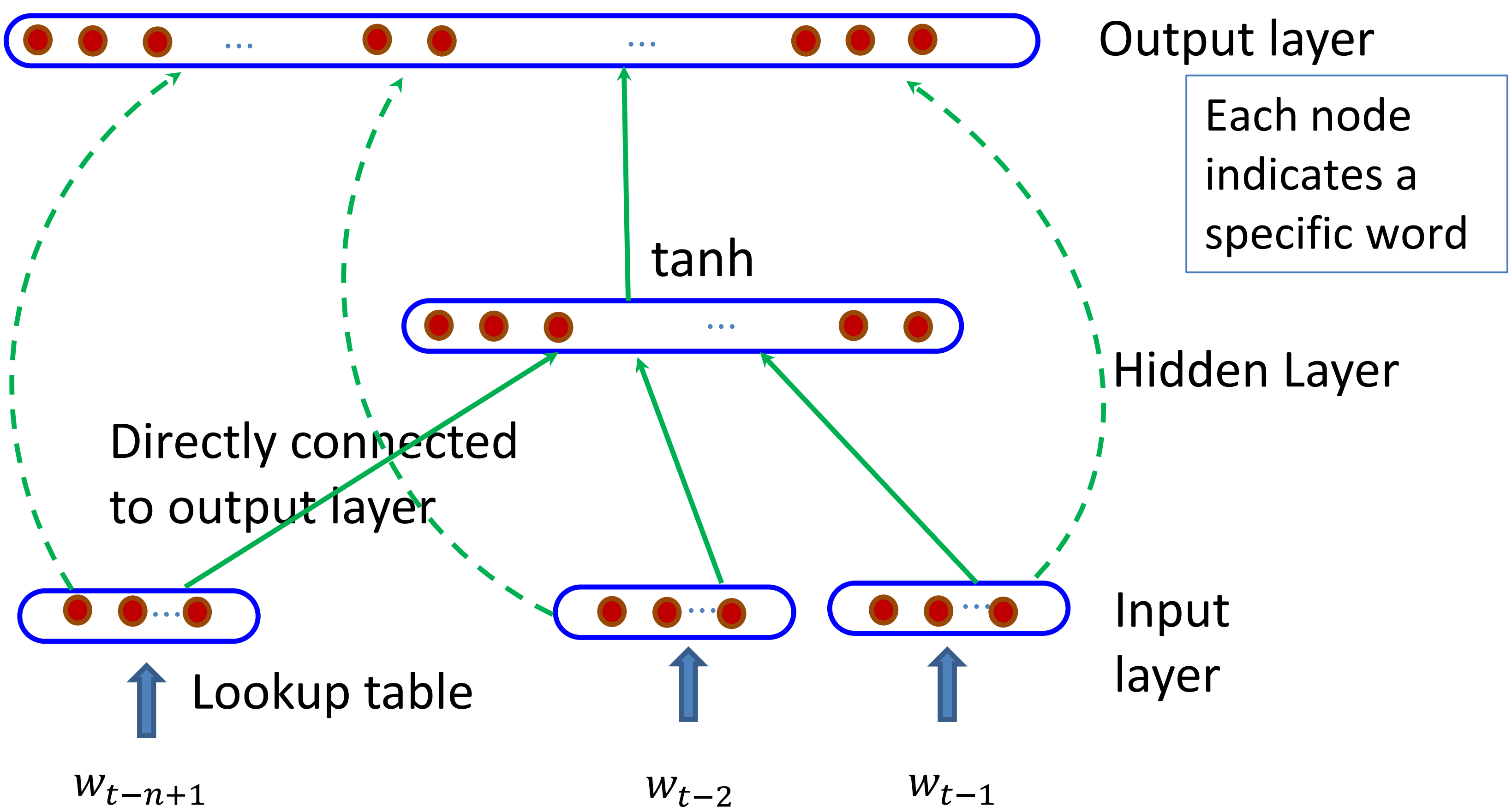


Neural Probabilistic Language Model

$P(w_t | w_{t-n+1}, \cdots, w_{t-1}) =$

$\text{Softmax} = \dfrac{exp(y_{w_t})}{\sum exp(y_t)}$

Normalization for probabilistic value



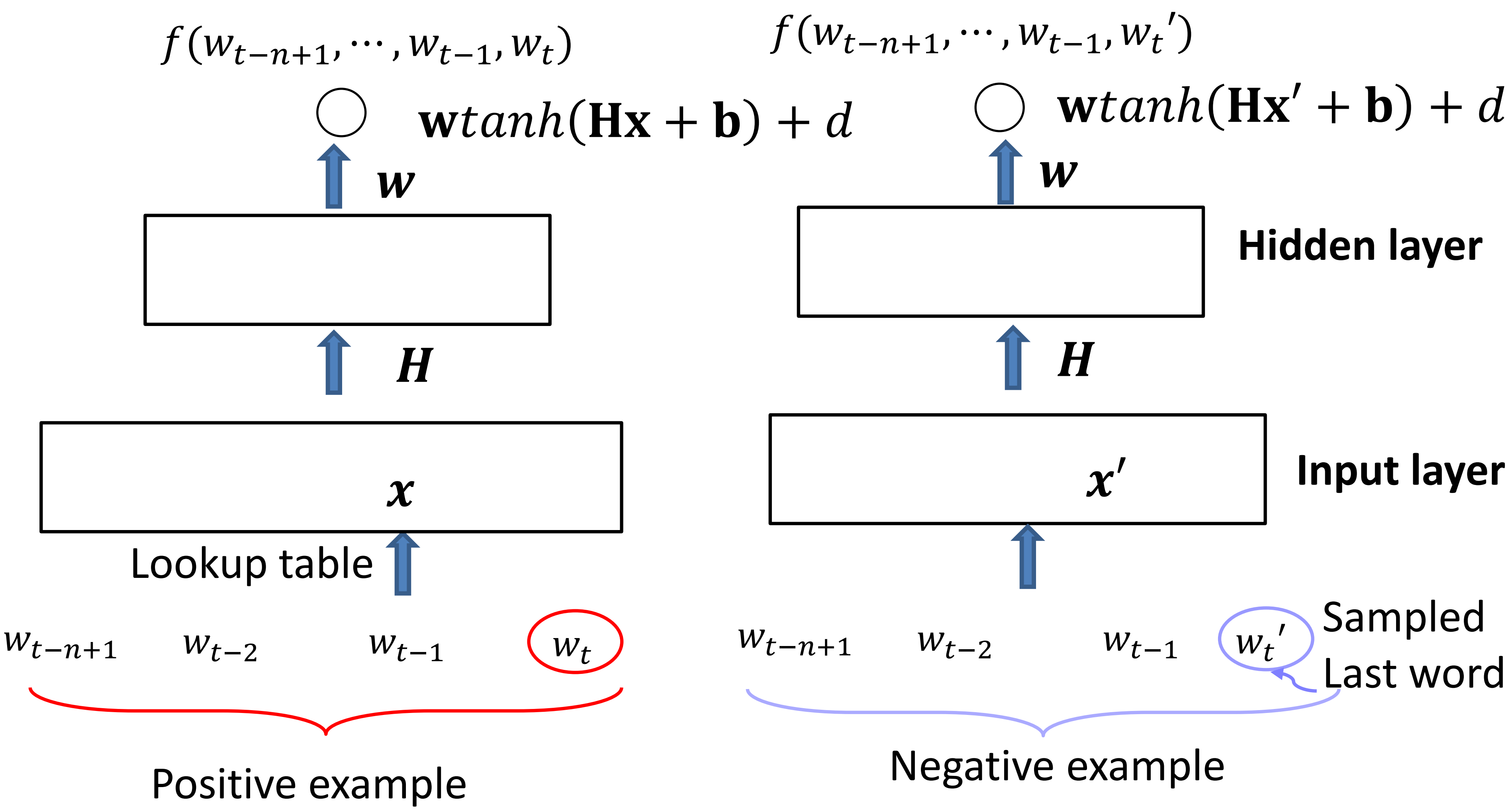
NPLM: Discussion

- **Limitation: Computational complexity**
  - Softmax layer requires computing scores over all vocabulary words
    - Vocabulary size is very large

Ranking Approach for Word Embedding

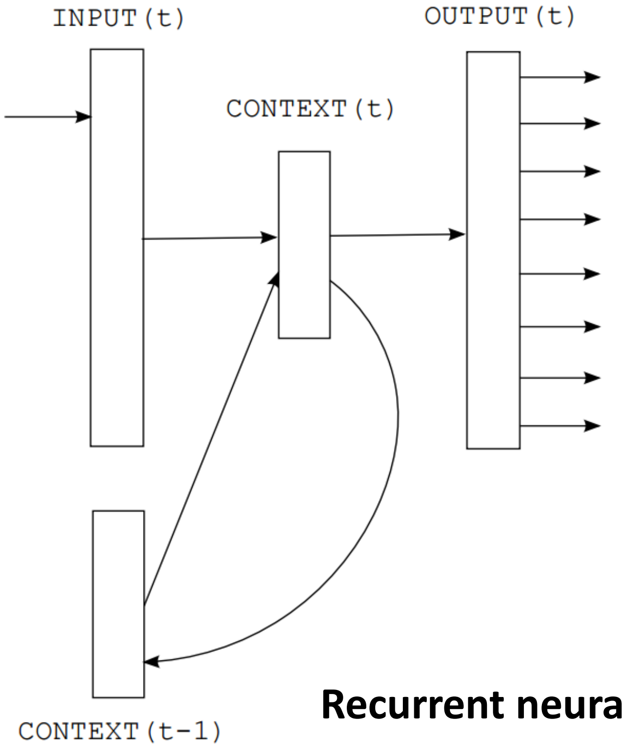
- **Idea: Sampling a negative example (Collobert & Weston '08)**
  - $\boldsymbol{s}$: a given sequence of words (in training data)
  - $\boldsymbol{s}'$: a negative example the last word is replaced with another word
  - $f(\boldsymbol{s})$: score of the sequence $\boldsymbol{s}$
  - Goal: makes the score difference ($f(\boldsymbol{s}) - f(\boldsymbol{s}')$) large
  - Various loss functions are possible
    - Hinge loss: $\max(0, 1 - f(\boldsymbol{s}) + f(\boldsymbol{s}'))$

Ranking Approach for Word Embedding (Collobert and Weston '08)



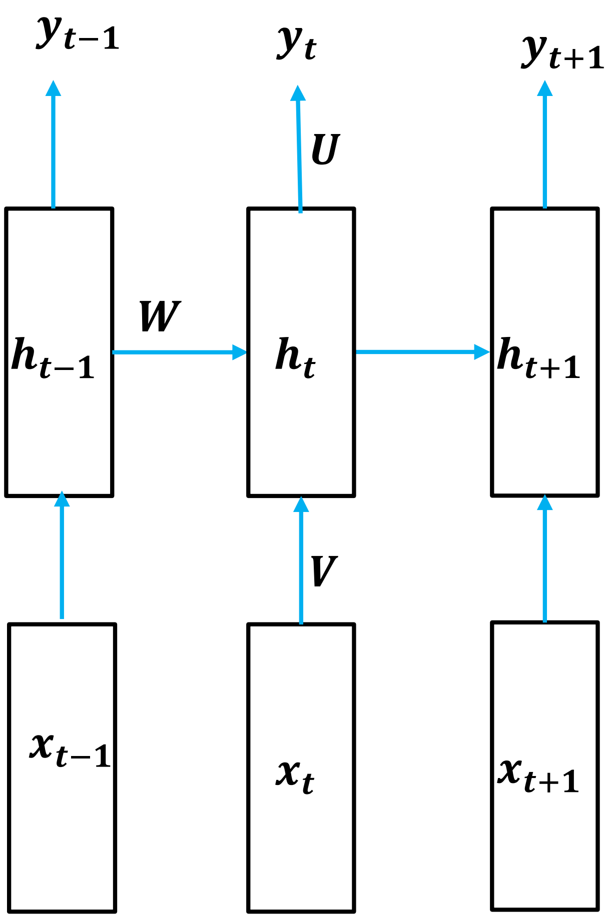
Recurrent Neural Network (RNN) based Language Model

- NPLM: only (n-1) previous words are conditioned
  - $P(w_i | w_{i-(n-1)}, \cdots, w_{i-1})$
- RNNLM: all previous words are conditioned
  - $P(w_i | w_1, \ldots, w_{i-1})$



Recurrent neural network

Recurrent Neural Network



$$\boxed{h_t = f(x_t, h_{t-1})}$$

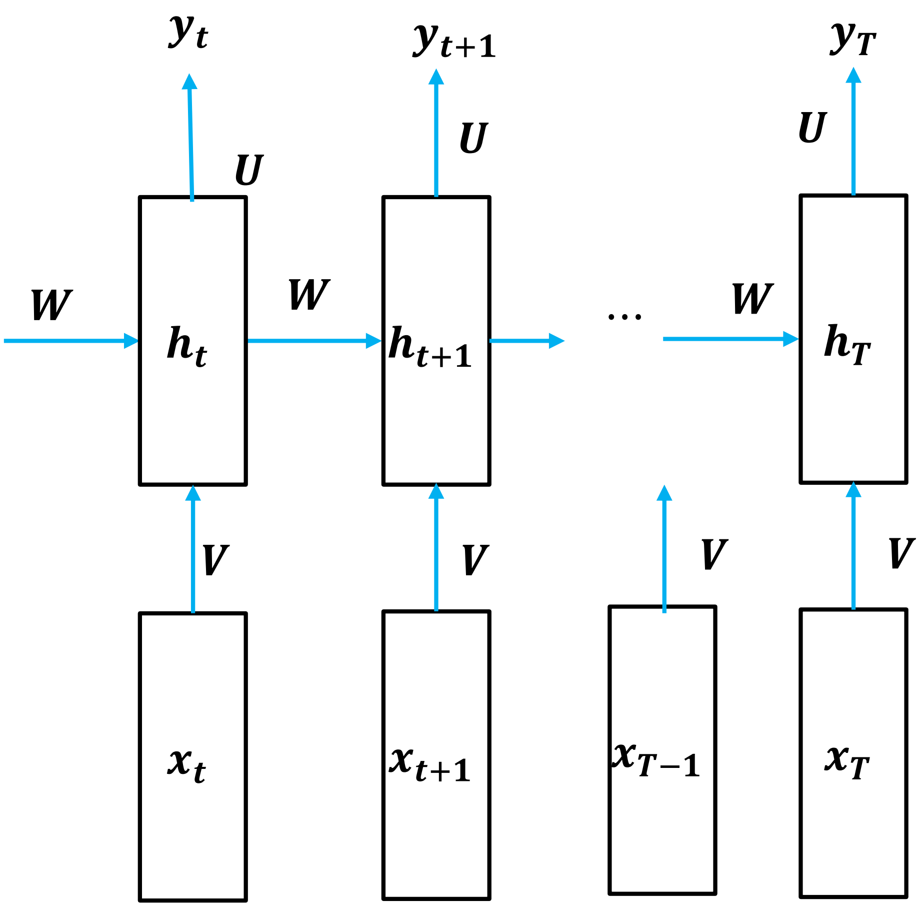
$$z_t = W h_{t-1} + V x_t$$

$$h_t = g(z_t)$$

$$y_t = U h_t$$

Recurrent Neural Network: Backpropagation Through Time

- Objective function: $\mathrm{L} = \sum_{\mathrm{t}} L(t)$



$$\frac{\partial L}{\partial \boldsymbol{U}} = \sum_t \boldsymbol{\delta}_t^o \boldsymbol{h}_t^T$$

$$\frac{\partial L}{\partial \boldsymbol{W}} = \sum_t \boldsymbol{\delta}_t^h \boldsymbol{h}_{t-1}^T$$

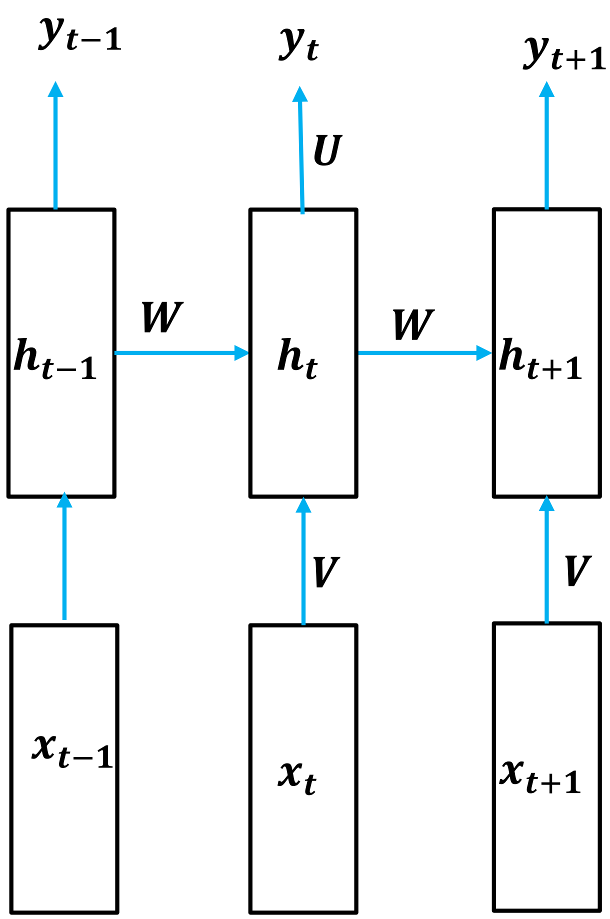
$$\frac{\partial L}{\partial \boldsymbol{V}} = \sum_t \boldsymbol{\delta}_t^h \boldsymbol{x}_t^T$$

$$\boldsymbol{\delta}_t^h = g'(\boldsymbol{z}_{t+1}) \circ \boldsymbol{W}^T \boldsymbol{\delta}_{t+1}^h + \boldsymbol{U}^T \boldsymbol{\delta}_t^o$$

Recurrent Neural Network: BPTT

- Objective function: $\mathrm{L} = \sum_{\mathrm{t}} L(t)$

$$\boldsymbol{\delta}_{t,t}^{h} = \boldsymbol{U}^{\mathrm{T}} \boldsymbol{\delta}_{\boldsymbol{t}}^{\boldsymbol{o}}$$
$$\boldsymbol{\delta}_{t-1,t}^{h} = g'(\boldsymbol{z}_t) \circ \mathbf{W}^{\mathrm{T}} \boldsymbol{\delta}_{t,t}^{\boldsymbol{h}}$$
$$\boldsymbol{\delta}_{t-2,t}^{h} = g'(\boldsymbol{z}_{t-1}) \circ \mathbf{W}^{\mathrm{T}} \boldsymbol{\delta}_{t-1,t}^{\boldsymbol{h}}$$



Compute gradient of L(t) w.r.t. params

Given specific time t

$$\frac{\partial L(t)}{\partial \boldsymbol{U}} = \boldsymbol{\delta}_t^{o} \boldsymbol{h}_t^{T}$$

$$\frac{\partial L(t)}{\partial \boldsymbol{W}} = \boldsymbol{\delta}_{\mathrm{t,t}}^{h} \boldsymbol{h}_{t-1}^{T} + \boldsymbol{\delta}_{\mathrm{t-1,t}}^{h} \boldsymbol{h}_{t-2}^{T} + \cdots + \boldsymbol{\delta}_{1,\mathrm{t}}^{h} \boldsymbol{h}_{0}^{T}$$

$$\frac{\partial L(t)}{\partial \boldsymbol{V}} = \boldsymbol{\delta}_{\mathrm{t,t}}^{h} \boldsymbol{x}_{t}^{T} + \cdots + \boldsymbol{\delta}_{1,\mathrm{t}}^{h} \boldsymbol{x}_{0}^{T}$$

$$\frac{\partial L(t)}{\partial C(w)} = \sum_{t' \leq t : w_t = w} \boldsymbol{\delta}_{\mathrm{t',t}}^{x}$$

Lookup table

Recurrent Neural Network: BPTT

- Vanishing gradient & gradient explosion problems

$$\frac{\partial L(t)}{\partial \boldsymbol{W}} = \boldsymbol{\delta}_{\mathrm{t,t}}^{h}\boldsymbol{h}_{t-1}^{T} + \boldsymbol{\delta}_{\mathrm{t-1,t}}^{h}\boldsymbol{h}_{t-2}^{T} + \cdots + \boldsymbol{\delta}_{\mathrm{1,t}}^{h}\boldsymbol{h}_{0}^{T}$$

$$\boldsymbol{\delta}_{t-1,t}^{h} = g'(\mathbf{z}_t)\circ \mathbf{W}^{\mathbf{T}}\boldsymbol{\delta}_{t,t}^{h}$$

$$\begin{aligned}\boldsymbol{\delta}_{t-2,t}^{h} &= g'(\mathbf{z}_{t-\mathbf{1}})\circ\mathbf{W}^{\mathbf{T}}\boldsymbol{\delta}_{t-1,t}^{h} \\ &= g'(\mathbf{z}_{t-\mathbf{1}})\circ g'(\mathbf{z}_t)\circ\mathbf{W}^{T}\mathbf{W}^{T}\delta_{t,t}^{h}\end{aligned}$$

$$\boxed{\begin{array}{l} g'(z_k)\cdots\circ g'(\mathbf{z}_{t-\mathbf{1}})\circ g'(\mathbf{z}_t) \\ \quad \mathbf{W}^{T}\ldots\mathbf{W}^{T}\mathbf{W}^{T}\end{array}} \longrightarrow$$

Can easily become
a very small or large number

Recurrent Neural Network: BPTT

- **Solutions to the exploding and vanishing gradients**
  - 1. Instead of initializing W randomly, start off from an identify matrix initialization
  - 2. Use the Rectified linear units (ReLU) instead of the sigmoid function
    - The derivative for the ReLU is either 0 or 1

Experiments: NPLM

NPLM

	n	c	h	m	direct	mix	train.	valid.	test.
MLP1	5		50	60	yes	no	182	284	268
MLP2	5		50	60	yes	yes		275	257
MLP3	5		0	60	yes	no	201	327	310
MLP4	5		0	60	yes	yes		286	272
MLP5	5		50	30	yes	no	209	296	279
MLP6	5		50	30	yes	yes		273	259
MLP7	3		50	30	yes	no	210	309	293
MLP8	3		50	30	yes	yes		284	270
MLP9	5		100	30	no	no	175	280	276
MLP10	5		100	30	no	yes		265	**252**
Del. Int.	3						31	352	336
Kneser-Ney back-off	3							334	323
Kneser-Ney back-off	4							332	321
Kneser-Ney back-off	5							332	321
class-based back-off	3	150						348	334
class-based back-off	3	200						354	340
class-based back-off	3	500						326	**312**
class-based back-off	3	1000						335	319
class-based back-off	3	2000						343	326
class-based back-off	4	500						327	312
class-based back-off	5	500						327	312

Experiments: RNN LM

	PPL		WER	
Model	RNN	RNN+KN	RNN	RNN+KN
KN5 - baseline	-	221	-	13.5
RNN 60/20	229	186	13.2	12.6
RNN 90/10	202	173	12.8	12.2
RNN 250/5	173	155	12.3	11.7
RNN 250/2	176	156	12.0	11.9
RNN 400/10	171	152	12.5	12.1
3xRNN static	151	143	11.6	11.3
3xRNN dynamic	128	121	11.3	11.1

Model	DEV WER	EVAL WER
Lattice 1 best	12.9	18.4
Baseline - KN5 (37M)	12.2	17.2
Discriminative LM [8] (37M)	11.5	16.9
Joint LM [9] (70M)	-	16.7
Static 3xRNN + KN5 (37M)	11.0	15.5
Dynamic 3xRNN + KN5 (37M)	10.7	16.3[4]

Long Short Term Memory (LSTM)

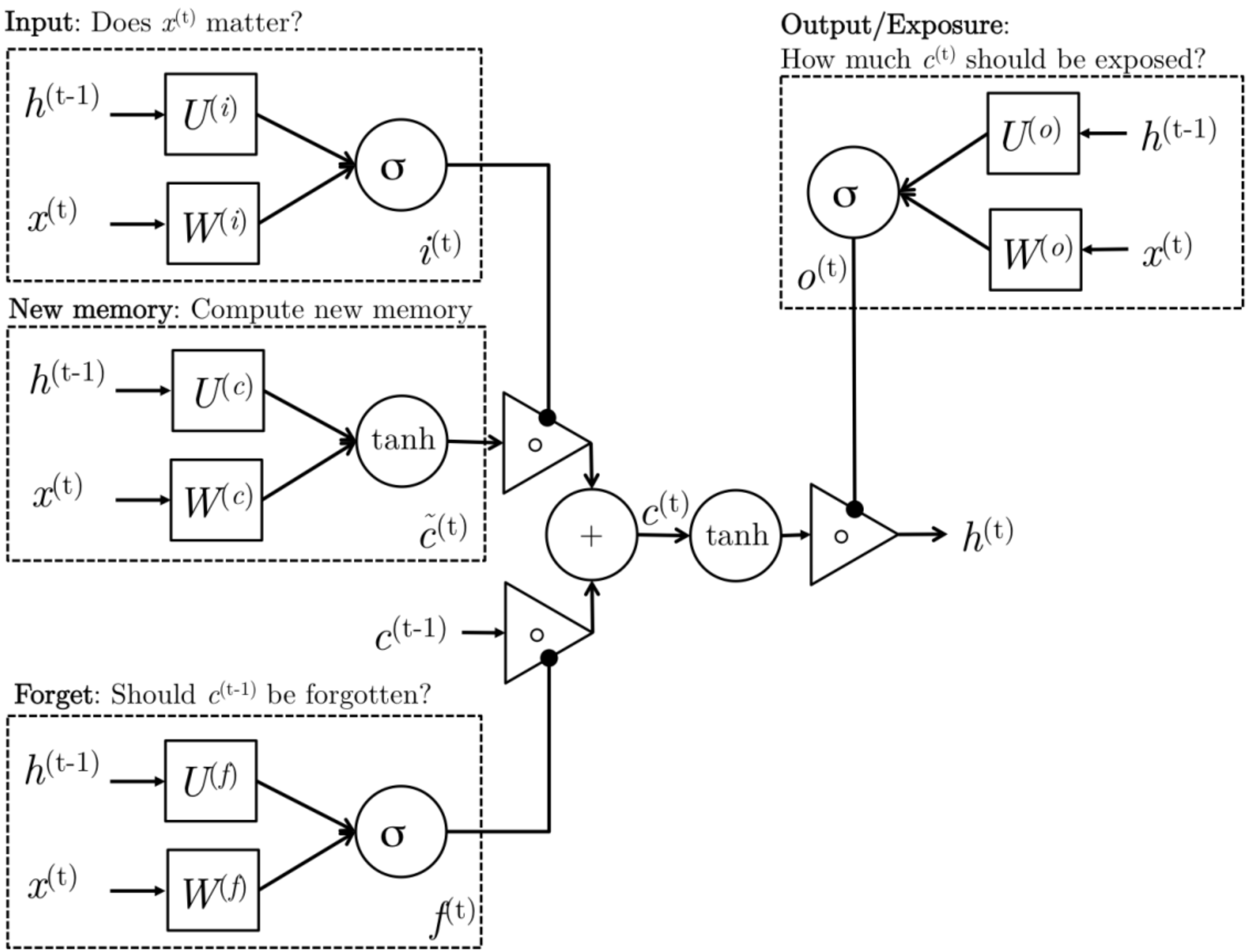
- **RNN**: Very hard to actually train long-term dependencies ← exploding & vanishing gradients
- **LSTM**: makes it easier for RNNs to capture long-term dependencies → Using **gated units**
  - Traditional LSTM (Hochreiter and Schmidhuer, 98)
    - Introduces input gate & output gate
    - Limitation: The output is close to zero as long as the output gate is closed.
  - Modern LSTM: Uses **forget** gate (Gers et al '00)
  - Variants of LSTM
    - Add **peephole connections** (Gers et al '02)
      - Allow all gates to inspect the current cell state even when the output gate is closed.

Long Short Term Memory (LSTM)

$$\boldsymbol{h}^{(t)} = \boldsymbol{f}(\boldsymbol{x}^{(t)}, \boldsymbol{h}^{(t-1)})$$

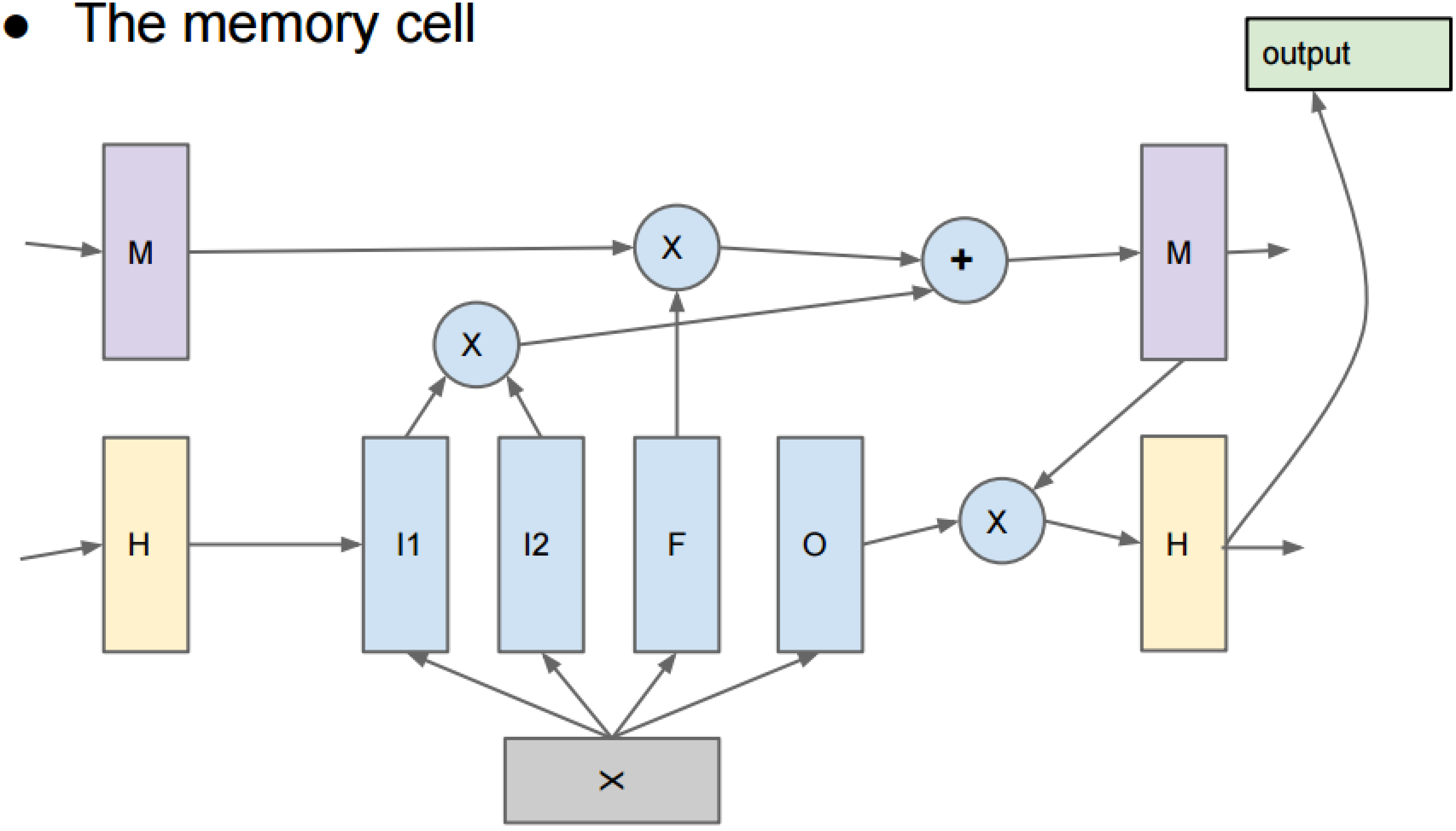
- $i^{(t)} = \sigma\left(W^{(i)}x^{(t)} + U^{(i)}h^{(t-1)}\right)$ (Input gate)
- $f^{(t)} = \sigma\left(W^{(f)}x^{(t)} + U^{(f)}h^{(t-1)}\right)$ (Forget gate)
- $o^{(t)} = \sigma\left(W^{(o)}x^{(t)} + U^{(o)}h^{(t-1)}\right)$ (Output/Exposure gate)
- $\tilde{c}^{(t)} = tanh\left(W^{(c)}x^{(t)} + U^{(c)}h^{(t-1)}\right)$ (New memory cell)
- $c^{(t)} = f^{(t)} \circ \tilde{c}^{(t-1)} + i^{(t)} \circ \tilde{c}^{(t)}$ (Final memory cell)
- $h^{(t)} = o^{(t)} \circ \tanh(c^{(t)})$

Long Short Term Memory (LSTM)



LSTM: Memory cell

- The memory cell



Long Short Term Memory (LSTM)

- **Input gate**: Whether or not the input is worth preserving and thus is used to gate the new memory
- **Forget gate**: Whether the past memory call is useful for the computation of the current memory cell
- **Final memory generation**: Takes the advices of the forget and input gates to produce the final memory
- **Output/Exposure gate**: What parts of the memory needs to be explored in the hidden state
  - The purpose is to separate the final memory from the hidden state. The final memory contains a lot of information that is not necessarily required to be saved in the hidden state

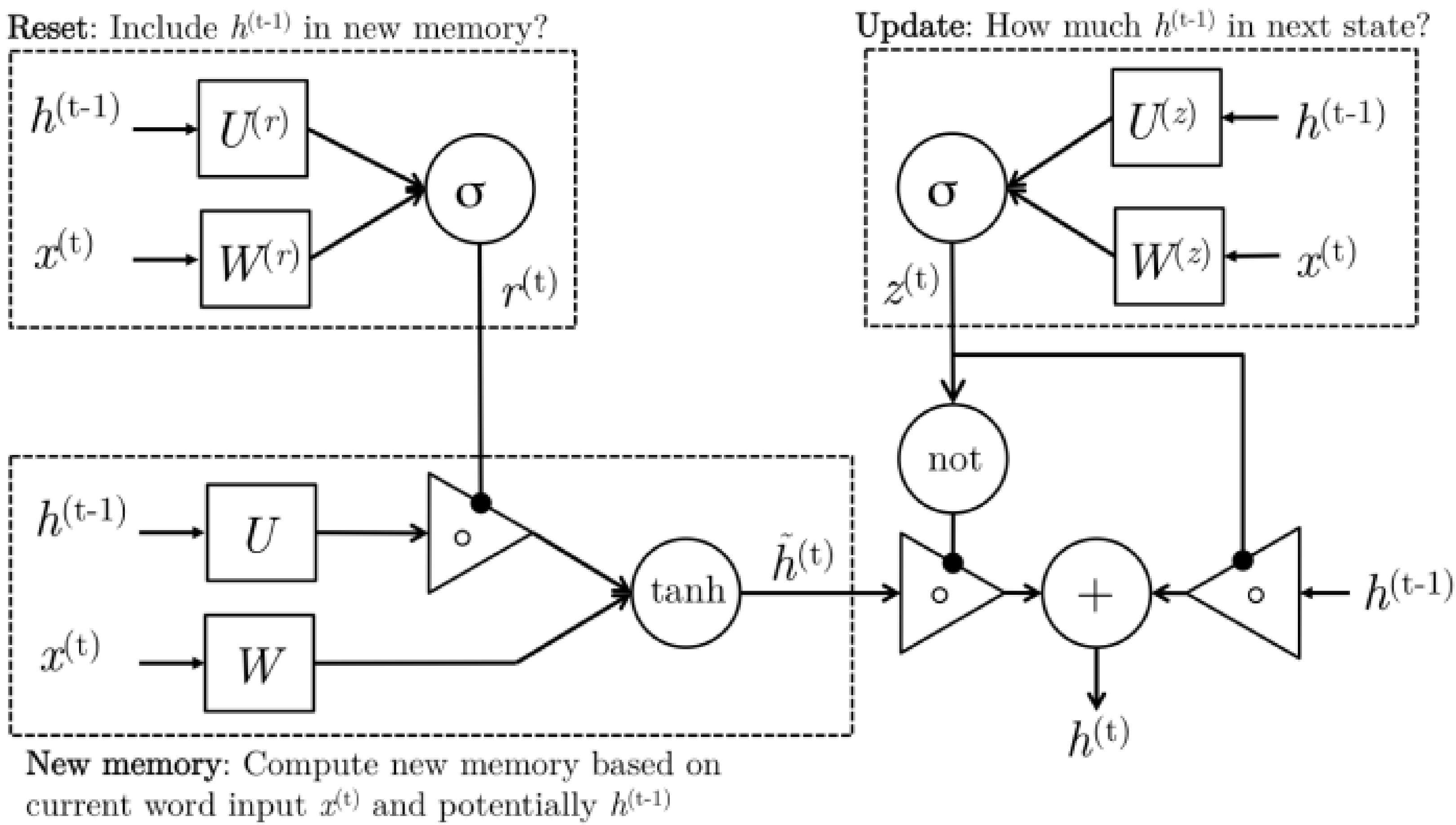
Gated Recurrent Units (Cho et al '14)

$$\boldsymbol{h}^{(t)} = \boldsymbol{f}(\boldsymbol{x}^{(t)}, \boldsymbol{h}^{(t-1)})$$

- Alternative architecture to handle long-term dependencies

- $z^{(t)} = \sigma\left(W^{(z)}x^{(t)} + U^{(z)}h^{(t-1)}\right)$ (Update gate)
- $r^{(t)} = \sigma\left(W^{(r)}x^{(t)} + U^{(r)}h^{(t-1)}\right)$ (Reset gate)
- $\tilde{h}^{(t)} = tanh\left(r^{(t)} \circ Uh^{(t-1)} + Wx^{(t)}\right)$ (New memory)
- $h^{(t)} = (1 - z^{(t)}) \circ \tilde{h}^{(t)} + z^{(t)} \circ h^{(t-1)}$ (Hidden state)

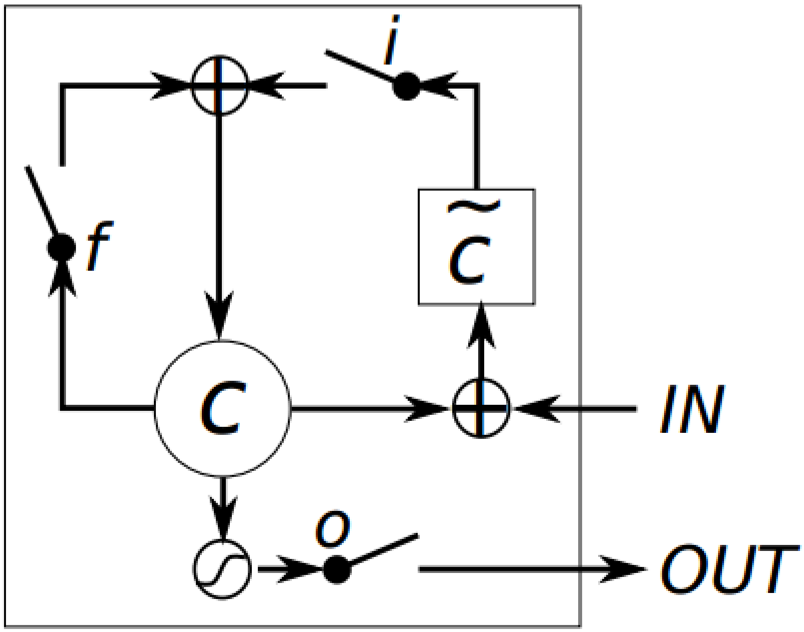
Gated Recurrent Units



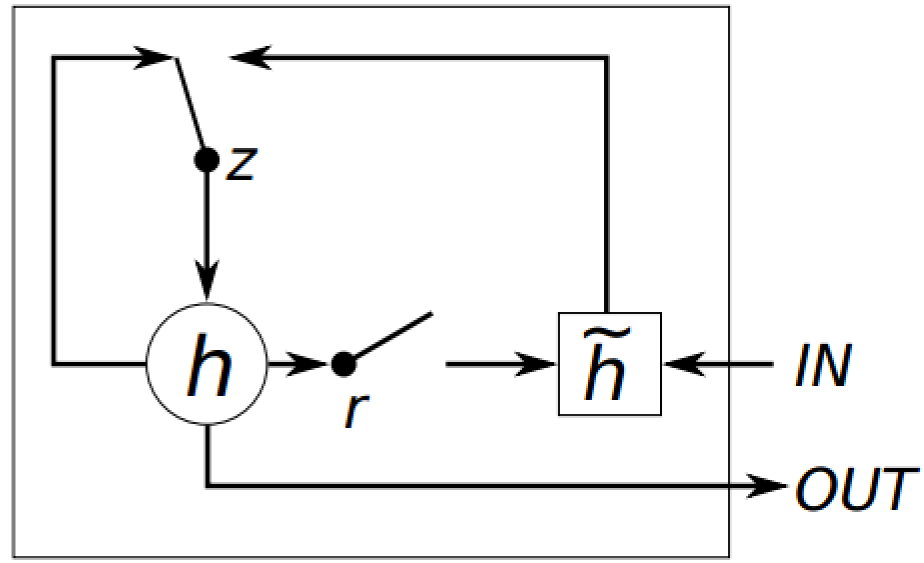
Gated Recurrent Units

- **New memory generation:** A new memory $\tilde{h}^{(t)}$ is the consolidation of a new input word $x^{(t)}$ with the past hidden state $h^{(t-1)}$
- **Reset gate:** Determining how important $h^{(t-1)}$ is to the summarization $\tilde{h}^{(t)}$. It can completely diminish past hidden state if it is irrelevant to the computation of the new memory
- **Update gate:** Determining how much of $h^{(t-1)}$ should be carried forward to the next stage
- **Hidden state:** The hidden state $h^{(t)}$ is generated using the past hidden input $h^{(t-1)}$ and the new memory generated $\tilde{h}^{(t)}$ with the advice of the update gate

LSTM vs. GRU



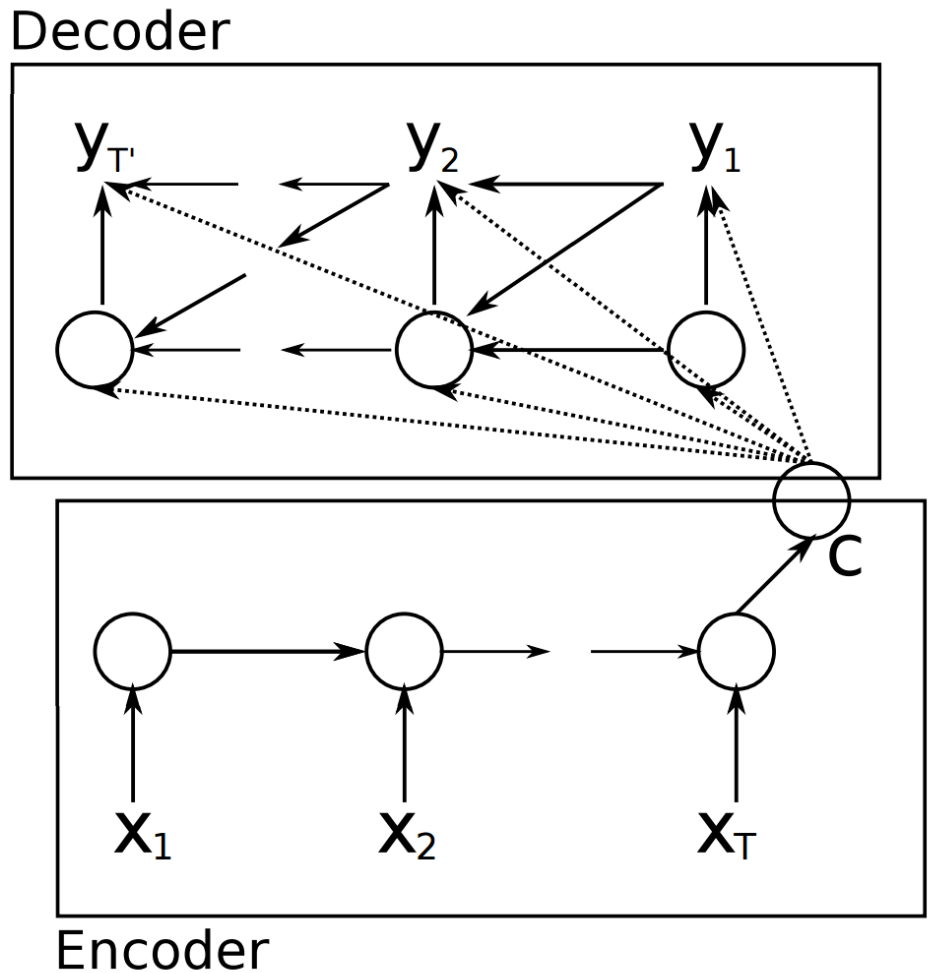
(a) Long Short-Term Memory



(b) Gated Recurrent Unit

Neural Machine Translation: RNN Encoder-Decoder (Cho et al '14)

- Computing the log of translation probability Log P(y|x) by two RNNs



Neural Machine Translation

- **RNN Encoder Decoder**
  - RNN Search
    - RNN Encoder Decoder with Attention
  - RNN Search Large Vocabulary

- **Sequence-to-Sequence Model**
  - LSTM
  - Rare word model

Neural Machine Translation: RNN Encoder-Decoder (Cho et al '14)

- RNN for the encoder: input $\boldsymbol{x}$ ➔ hidden state $\boldsymbol{c}$
  - $\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, x_t)$
    - **Encode** a variable-length sequence into a fixed-length vector representation
    - Reads each symbol of an input sequence **x** sequentially
    - After reading the end of the sequence, we obtain a summary **c** of the whole input sequence: $\boldsymbol{c} = \boldsymbol{h}_T$

RNN Encoder-Decoder (Cho et al '14)

- RNN for the decoder: Hidden state $\boldsymbol{c}$ ➔ Output $\boldsymbol{y}$
  - $\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, y_{t-1}, \boldsymbol{c})$
  - $P(y_t | y_{t-1}, \cdots, y_1, \boldsymbol{c}) = g(\boldsymbol{h}_t, y_{t-1}, \boldsymbol{c})$
    - **Decode** a given fixed-length vector representation back into a variable-length sequence
    - Generate the output sequence by predicting the next symbol y_t given the previous hidden state
    - Both y_t and $\boldsymbol{h}_t$ are conditioned on y_{t-1}

- How to define f?
  - Gated recurrent unit (**GRU**) is used to capture long-term dependencies

Using RNN Encoder-Decoder for Statistical Machine Translation

- Statistical machine translation
  - Generative model: $P(\boldsymbol{f}|\boldsymbol{e}) \propto p(\boldsymbol{e}|\boldsymbol{f})P(\boldsymbol{f})$
    - $p(\boldsymbol{e}|\boldsymbol{f})$: Translation model, $p(\boldsymbol{f})$: Language model
  - Log-linear model: $\log P(\boldsymbol{f}|\boldsymbol{e}) = \sum w_n f_n(\boldsymbol{f},\boldsymbol{e}) + logZ(\boldsymbol{e})$
    - In Phrase-based SMT
      - $\log P(\boldsymbol{e}|\boldsymbol{f})$ is factorized into the translation probabilities of matching phrases in the source and target sents
- **Scoring phrase pairs with RNN Encoder-Decoder**
  - Train the RNN Encoder-Decoder on a table of phrase pairs ➔ Use its scores as additional features
    - Ignore the normalized frequencies of each phrase pair
      - Existing translation prob in the table already reflects the frequencies of the phrase pairs

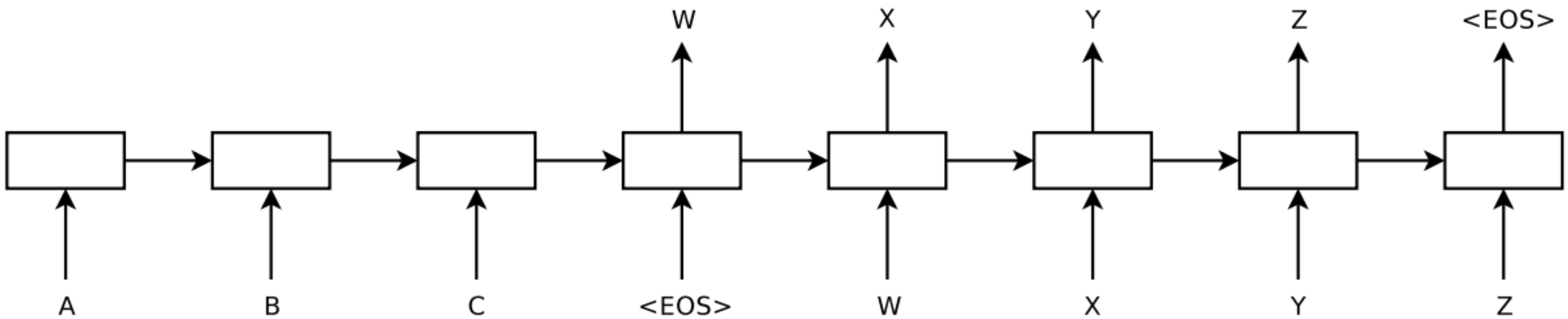
RNN Encoder-Decoder: Experiments

- Training
  - # hidden units: 1,000, # dim for word embedding: 100
  - Adadelta & stochastic gradient descent
  - Use Controlled vocabulary setting
    - Tmost frequent 15,000 words for both source & target languages
  - All the out-of-vocabulary words are mapped to UNK

Models	BLEU dev	BLEU test
Baseline	30.64	33.30
RNN	31.20	33.87
CSLM + RNN	31.48	34.64
CSLM + RNN + WP	31.50	34.54

English-to-French translation

Neural Machine Translation: Sequence-to-sequence (Sutskever et al 14)

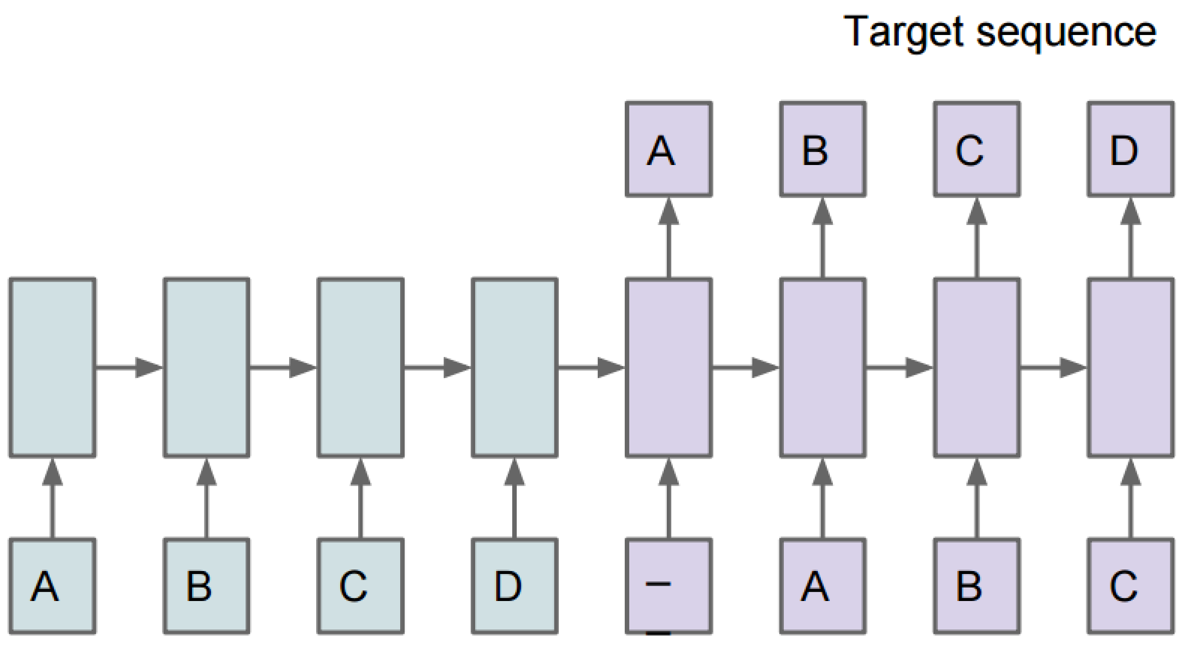


- <EOS>: the end-of-sentence token

- 1) a LSTM reads input sentence $\boldsymbol{x}$ which generates the last hidden state $\boldsymbol{v}$
- 2) a standard LSTM-LM computes the probability of $y_1, \cdots, y_{T'}$ given the initial hidden state $\boldsymbol{v}$
  - $P(y_1, \cdots, y_{T'} | x_1, \cdots, x_T) = \prod_t \; P(y_t | v, y_1, \cdots, y_{t-1})$
  - $P(y_t | v, y_1, \cdots, y_{t-1})$: Taking Softmax over all words

Can the LSTM reconstruct the input sentence?

- Can this scheme learn the identity function?



- Answer: it can, and it can do it very easily. It just does it effortlessly and perfectly.

Slide from http://www.cs.tau.ac.il/~wolf/deeplearningmeeting/pdfs/Sup+LSTM.pdf

Sequence-to-Sequence Model (Sutskever et al 14)

- Two different LSTMs
  - one for the input sentence
  - Another for the output sentence
- Deep LSTM (with 4 layer)
  - 1,000 cell at each layer, 1,000 dimensional word embedding
  - Deep LSTM uses 8000 numbers for sentence representation
- Revising source sentence
  - …, C’, B’, A’ → A, B, C,…
  - Backprop can notice the short-term dependencies first, and slowly extend them to long range dependencies
- Ensemble of deep LSTMs
  - LSTMs trained from Different initialization

Sequence-to-Sequence Model : End-to-end Translation Experiments

- English-to-French translation
- No SMT is used for reference model
- Input/Output vocabulary size: 160,000/80,000
- Training objective: maximizing $\sum_{(S,T)} logP(T|S)$
- Decoding: Beam search

Method	test BLEU score (ntst14)
Bahdanau et al. [2]	28.45
Baseline System [29]	33.30
Single forward LSTM, beam size 12	26.17
Single reversed LSTM, beam size 12	30.59
Ensemble of 5 reversed LSTMs, beam size 1	33.00
Ensemble of 2 reversed LSTMs, beam size 12	33.27
Ensemble of 5 reversed LSTMs, beam size 2	34.50
Ensemble of 5 reversed LSTMs, beam size 12	**34.81**

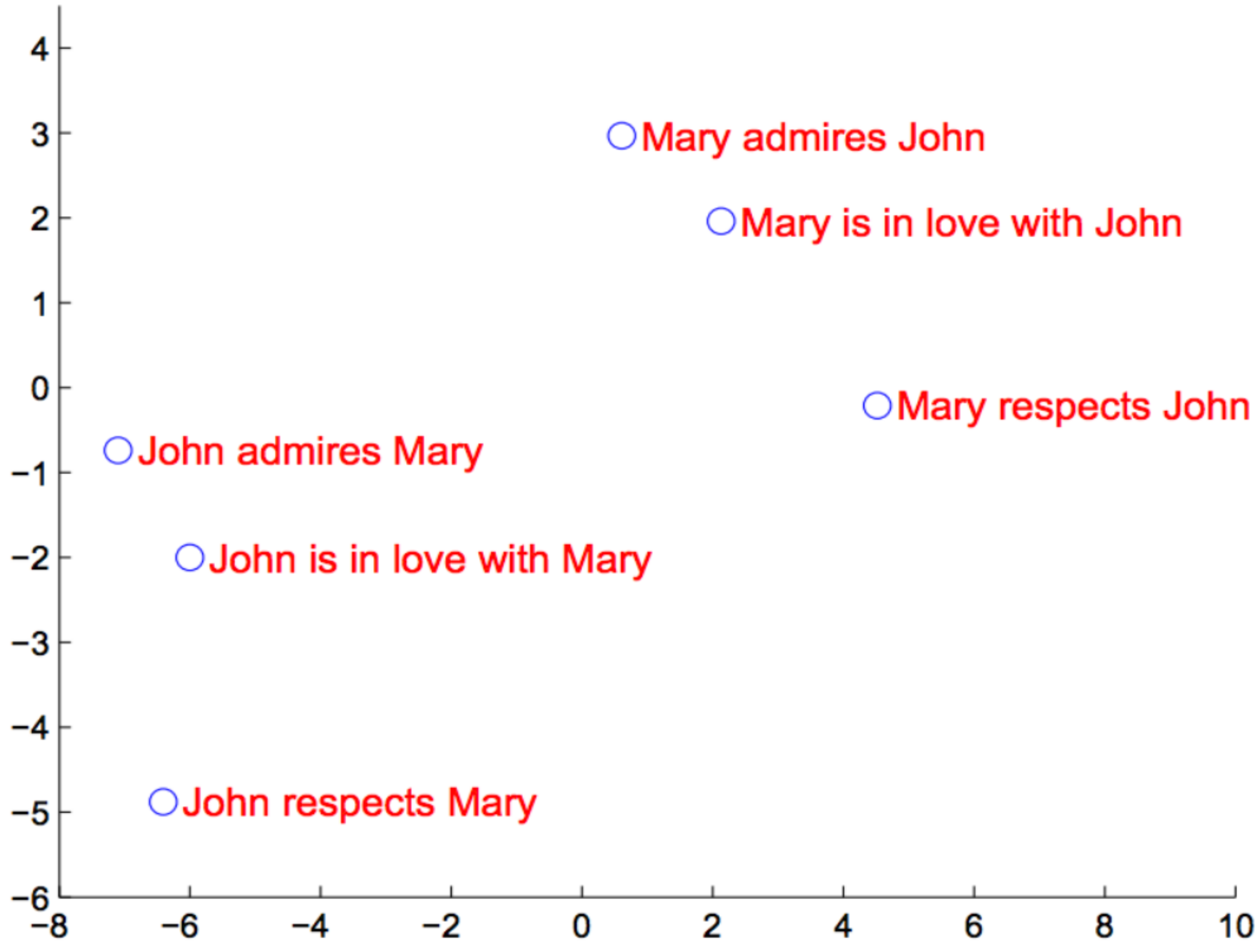
Sequence-to-Sequence Model : Re-scoring Experiments

- Rescoring the 1000-best list of the baseline system

Method	test BLEU score (ntst14)
Baseline System [29]	33.30
Cho et al. [5]	34.54
Best WMT'14 result [9]	**37.0**
Rescoring the baseline 1000-best with a single forward LSTM	35.61
Rescoring the baseline 1000-best with a single reversed LSTM	35.85
Rescoring the baseline 1000-best with an ensemble of 5 reversed LSTMs	**36.5**
Oracle Rescoring of the Baseline 1000-best lists	$\sim$45

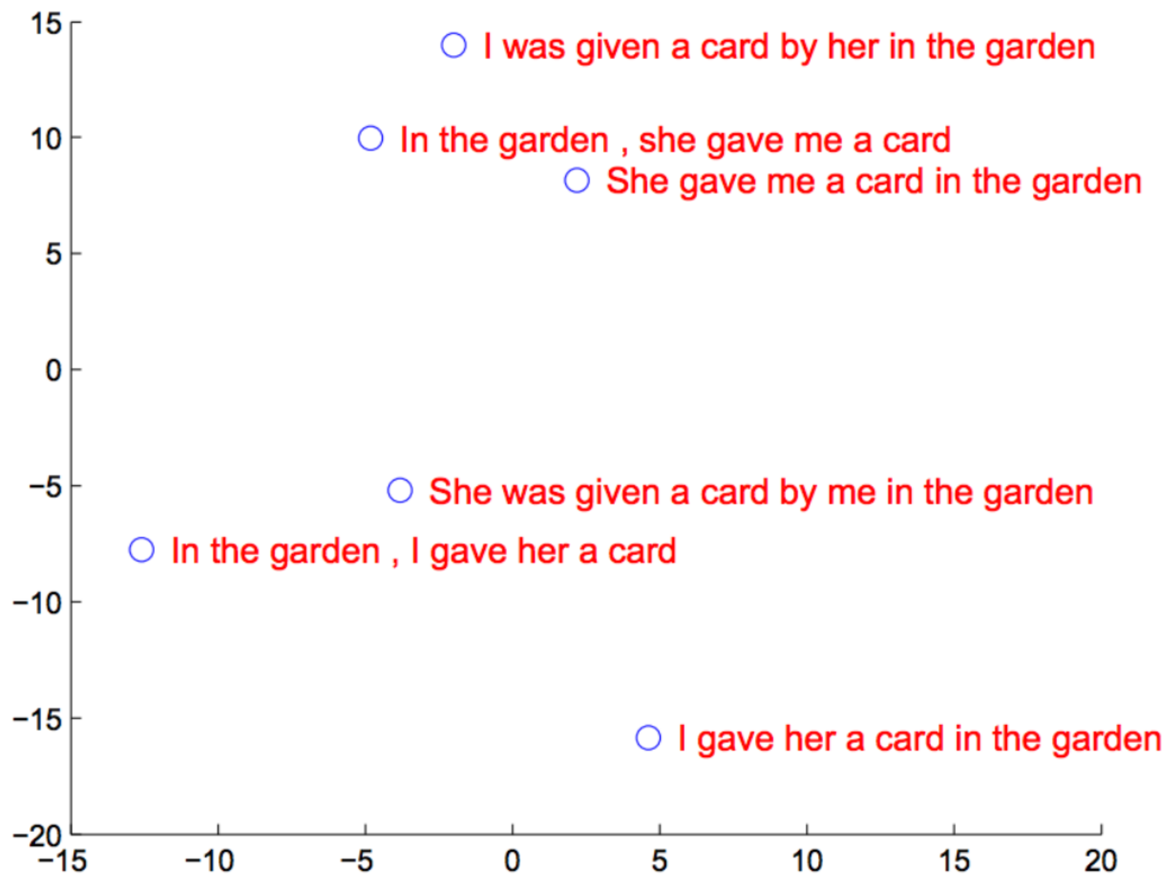
Sequence-to-Sequence Model : Learned representation

- 2-dimensional PCA projection of the LSTM hidden states

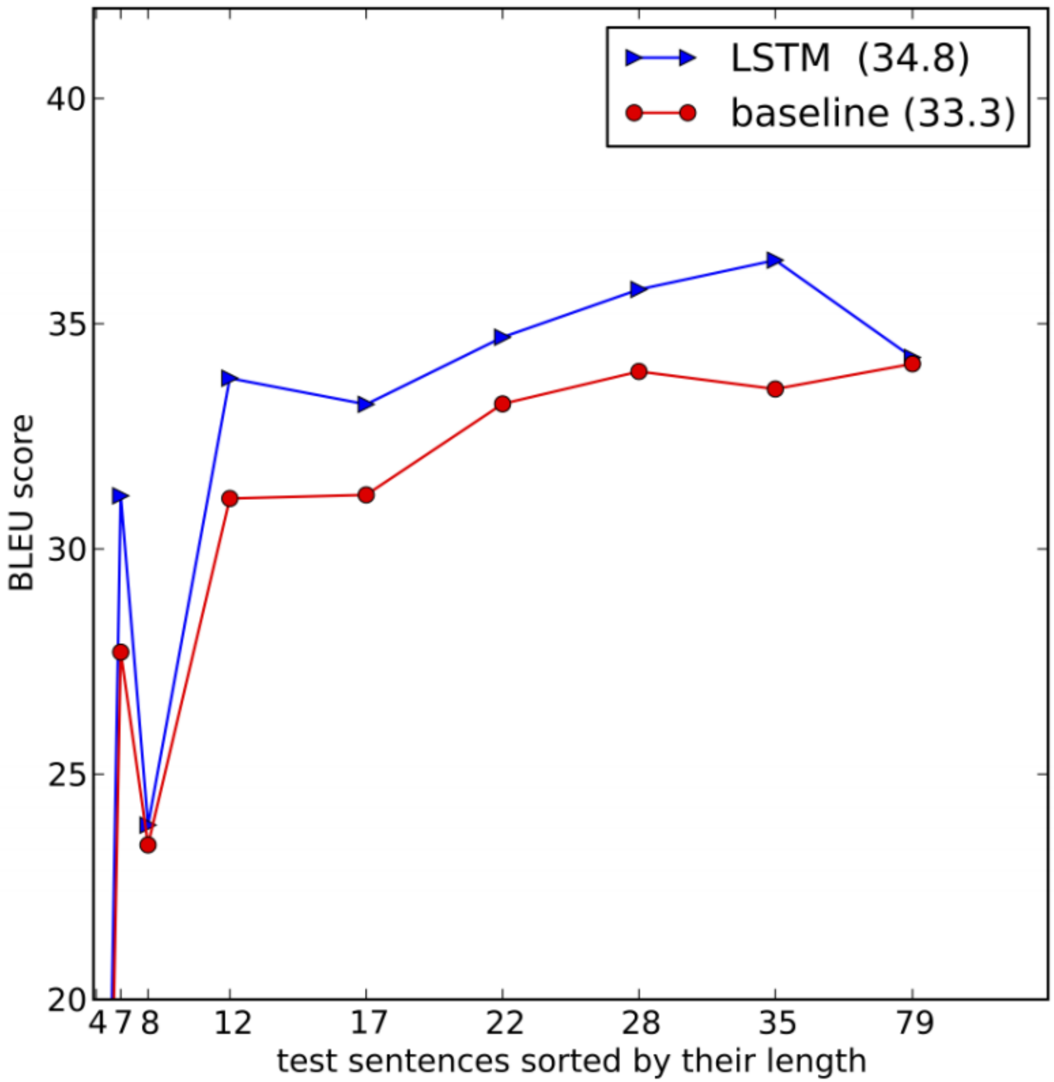


Sequence-to-Sequence Model: Learned representation

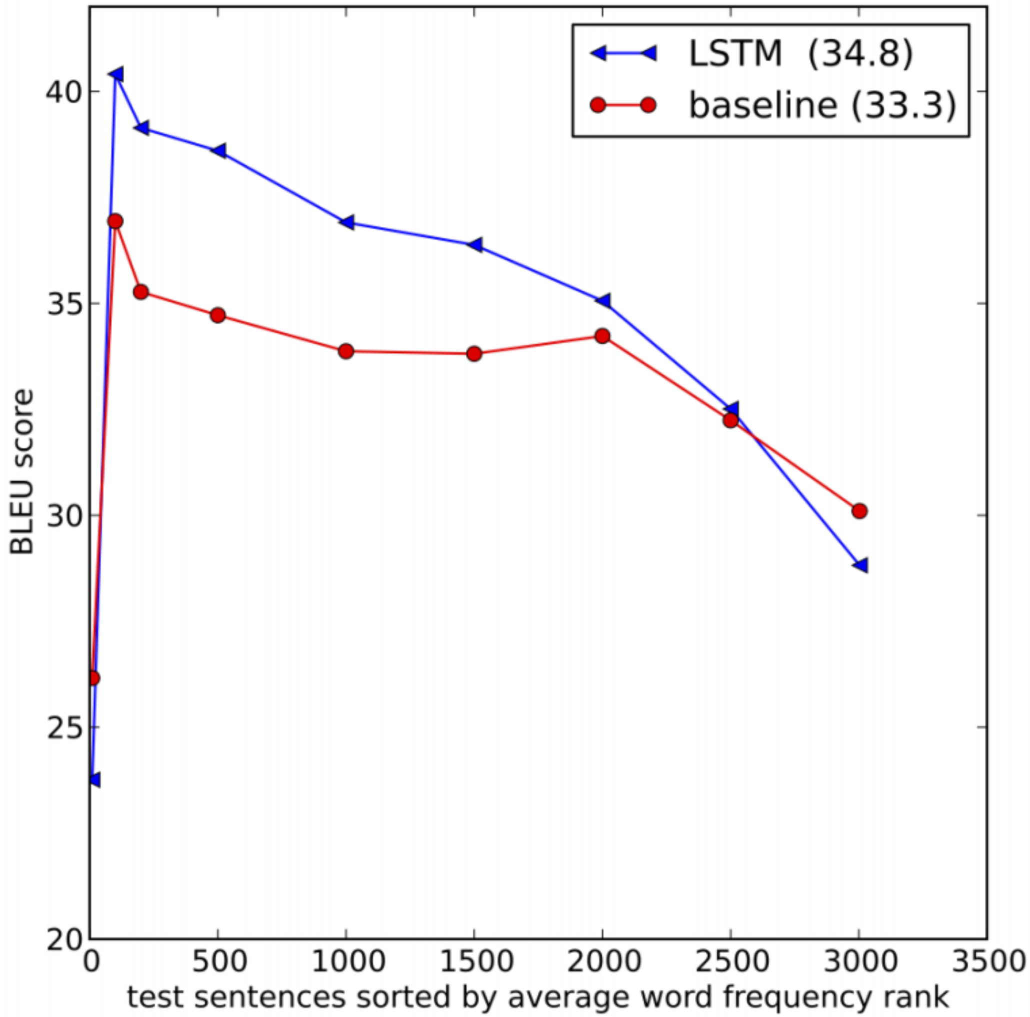
- 2-dimensional PCA projection of the LSTM hidden states



Sequence-to-Sequence Learning: Performance vs Sentence Length



Sequence-to-Sequence Learning: Performance on rare words

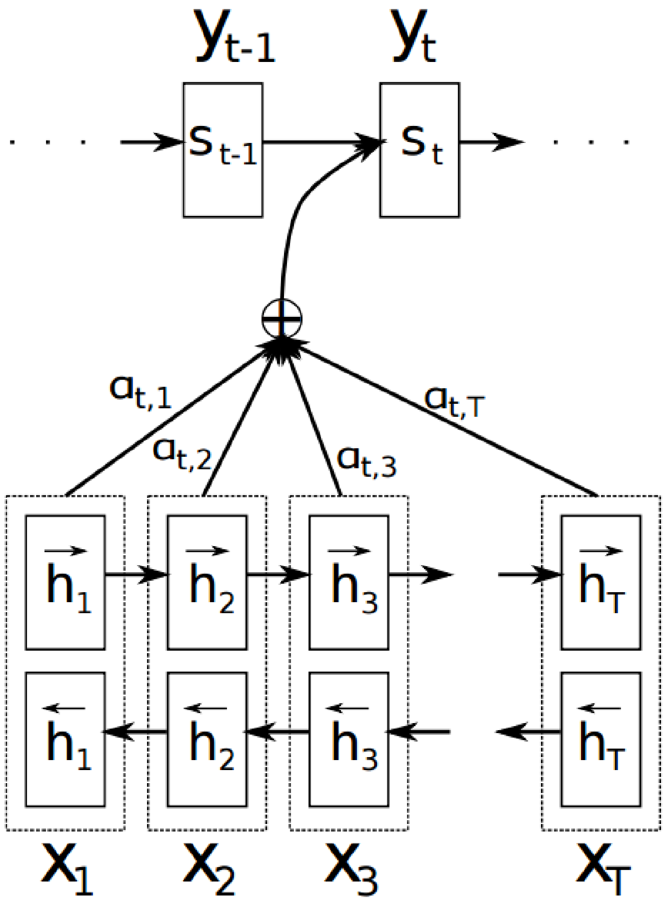


RNN Search: Jointly Learning to Align and Translate (Bahdanau et al '15)

- **RNN Search: Extended architecture of RNN Encoder-Decoder**
  - Encodes the input sentence into a seq of vectors using a bidirectional RNN
  - Context vector **c**
    - Previously, the last hidden state (Cho et al '14)
    - Now, mixture of hidden states of input sentence at generating each target word
      - During decoding, chooses a subset of these vectors adaptively

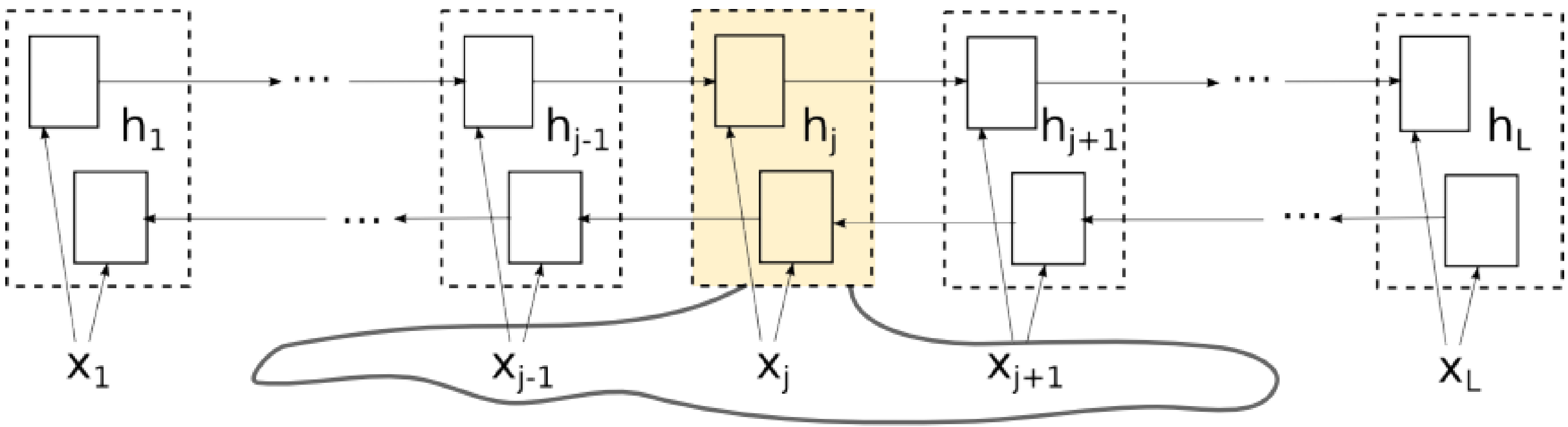
RNN Search: Components

- $p(y_i|y_1, \cdots, y_{i-1}) = g(y_{i-1}, s_i, c_i)$
- $s_i = f(s_{i-1}, y_{i-1}, c_i)$
  - An RNN hidden state
- $c_i = \sum_j \alpha_{ij} h_j$
- $\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_k \exp(e_{ik})}$
- $e_{ij} = a(s_{i-1}, h_j)$
  - Alignment model



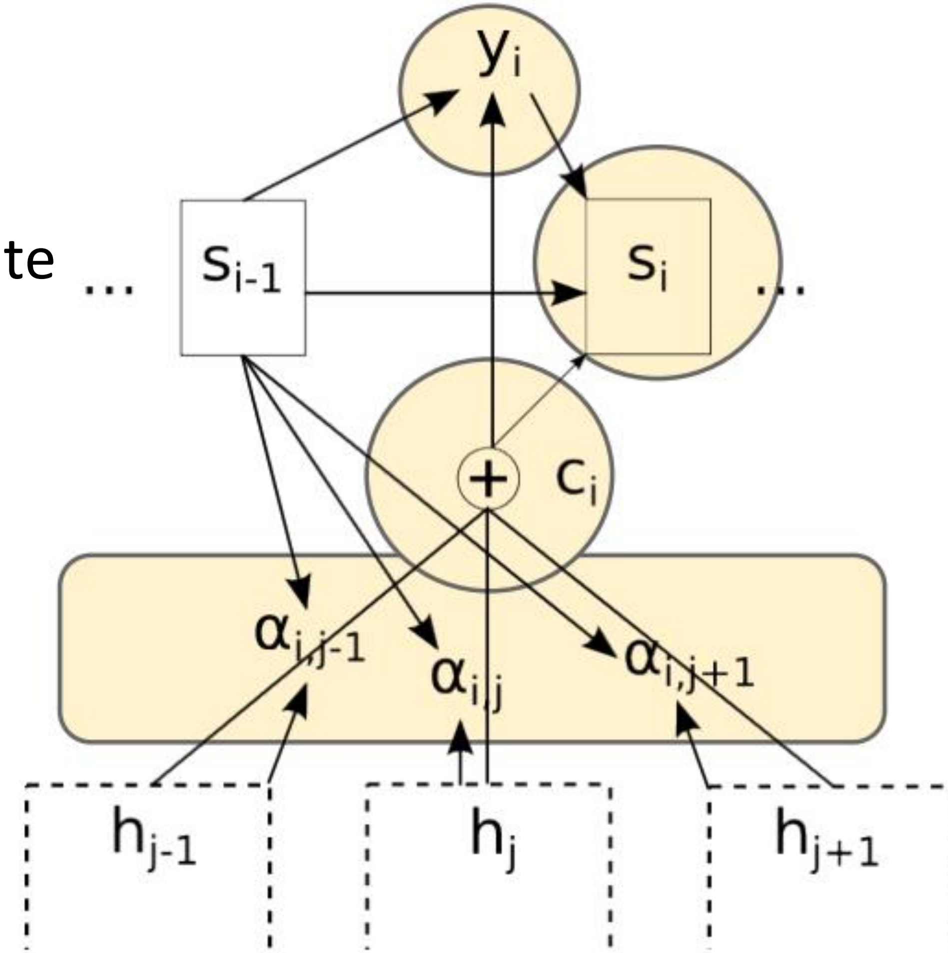
RNN Search: Encoder

- Bidirectional RNN



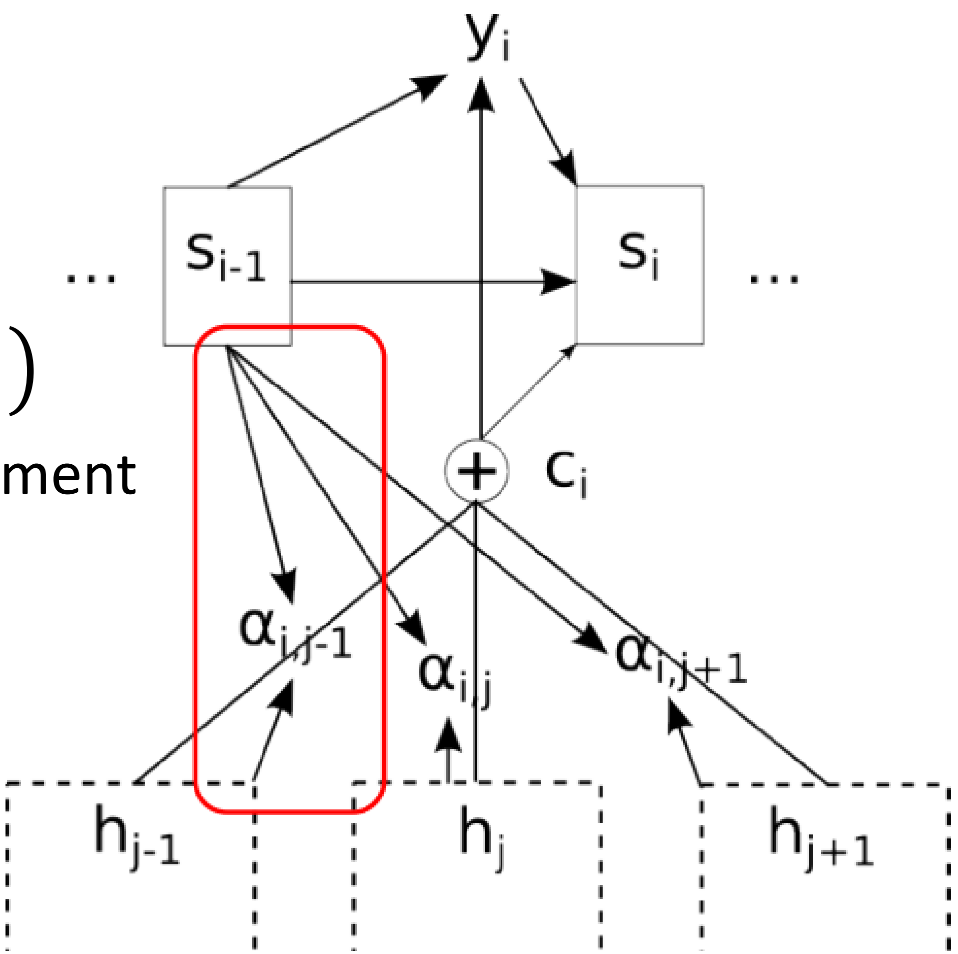
RNN Search: Decoder

- compute alignment
- compute context
- generate new output
- compute new decoder state



RNN Search: Alignment model

- $e_{ij} = v^T \tanh\left(W s_{i-1} + V h_j\right)$
  - Directly computes a soft alignment
- $\alpha_{ij} = \dfrac{\exp(e_{ij})}{\sum_k \exp(e_{ik})}$
  - Expected annotation



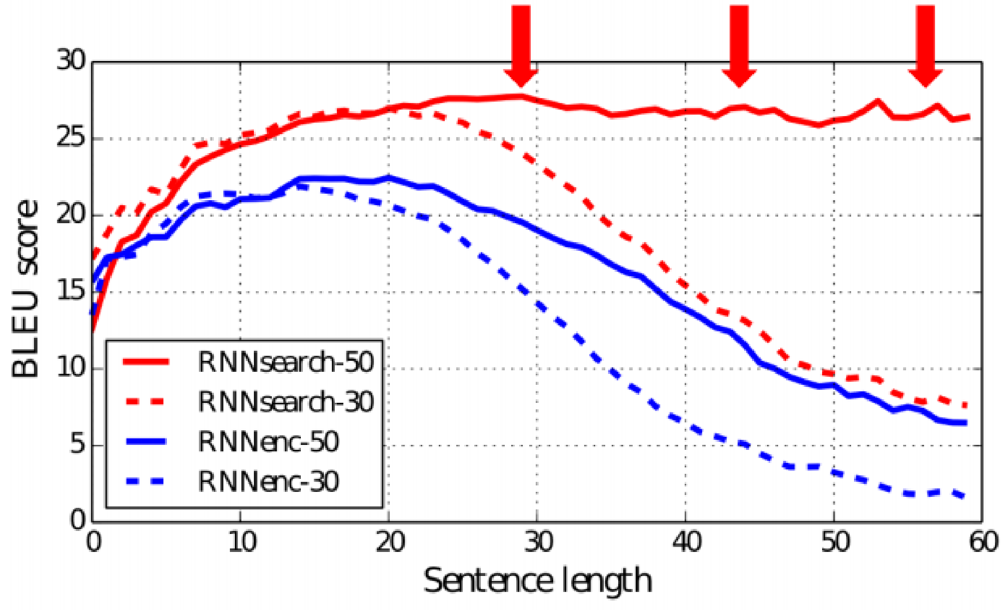
Alignment model is jointly trained
with all other components

RNN Search: Experiment – English-to-French

- Model
  - RNN Search, 1,000 units
- Baseline
  - RNN Encoder-Decoder, 1000 units
- Data
  - English to French translation, 348 million words
  - 30000 words + UNK token for the networks, all words for Moses
- Training
  - Stochastic gradient descent (SGD) with a minibatch of 80
- Decoding
  - Beam search (Sutskever '14)

RNN Search:
Experiment – English-to-French

no performance drop on long sentences



much better than RNN Encoder-Decoder

Model	All	No UNK°
RNNencdec-30	13.93	24.19
RNNsearch-30	21.50	31.44
RNNencdec-50	17.82	26.71
RNNsearch-50	26.75	34.16
RNNsearch-50*	28.45	36.15
Moses	33.30	35.63

without unknown words comparable with the SMT system

RNN Search: Large Target Vocabulary (Jean et al ‘15)

- Decoding in RNN search
  - $P(y_t|y_{<t}, x) = \frac{1}{Z}\exp(w_t^T \phi(y_{t-1}, z_t, c_t) + b_t)$
- Normalization constant
  - $Z = \sum_{k:y_k \in V} \exp(w_t^T \phi(y_{t-1}, z_t, c_t) + b_t)$
    - Computationally inefficient
- Idea: Use only small V’ of target vocabulary to approximate Z
  - Based on importance sampling (Bengio and Senecal, ‘08)

RNN Search: Large Target Vocabulary (Jean et al '15)

$$\nabla \log p(y_t|y_{<t},x) = \nabla \mathcal{E}(y_t) - \boxed{\sum_{k:y_k \in V} P(y_k|y_{<t},x) \nabla \mathcal{E}(y_k)}$$

$$\mathcal{E}(y_j) = w_j^T \phi(y_{j-1}, z_j, c_j) + b_j$$

Energy function

$$E_P\left(\nabla \mathcal{E}(y)\right)$$

Expected gradient of the energy

Approximation by importance sampling

$$\sum_{k:y_k \in V'} \frac{\omega_k}{\sum_{k':y_{k'} \in V'} \omega_{k'}} \nabla \mathcal{E}(y_k)$$

Proposal distribution

$$\omega_k = \exp\{\mathcal{E}(y_k) - \log Q(y_k)\}$$

RNN Search LV: Experiments

- Setting (Vocabulary size)
  - RNN Search: 30,000 for En-Fr, 50,000 for En-Ge
  - RNN Search LV: 500,000 source and target words

	RNNsearch	RNNsearch-LV	Google	Phrase-based SMT	
Basic NMT	29.97 (26.58)	32.68 (28.76)	30.6⋆	33.3*	37.03•
+Candidate List	–	33.36 (29.32)	–		
+UNK Replace	33.08 (29.08)	34.11 (29.98)	33.1°		
+Reshuffle (τ=50k)	–	34.60 (30.53)	–		
+Ensemble	–	37.19 (31.98)	37.5°		

(a) English→French

	RNNsearch	RNNsearch-LV	Phrase-based SMT
Basic NMT	16.46 (17.13)	16.95 (17.85)	20.67⋄
+Candidate List	–	17.46 (18.00)	
+UNK Replace	18.97 (19.16)	18.89 (19.03)	
+Reshuffle	–	19.40 (19.37)	
+Ensemble	–	21.59 (21.06)	

(b) English→German

Sequence-to-Sequence Model: Rare Word Models (Luong et al '15)

- **Extend LSTM for NMT (Suskever '14)**

- **Translation-specific approach**
  - For each OOV word, we make a pointer to its corresponding word in the source sentence
  - The pointer information is later utilized in a post-processing step
    - Directly translates OOV using a original dictionary
    - or with the identify translation, if no translation is found

Rare Word Models (Luong et al ‘15)

- **Replacing rare words with UNK**
  - Positional information is stored in target sentence

en: The *ecotax* portico in *Pont-de-Buis* , . . .

fr: Le *portique* *écotaxe* de *Pont-de-Buis* , .

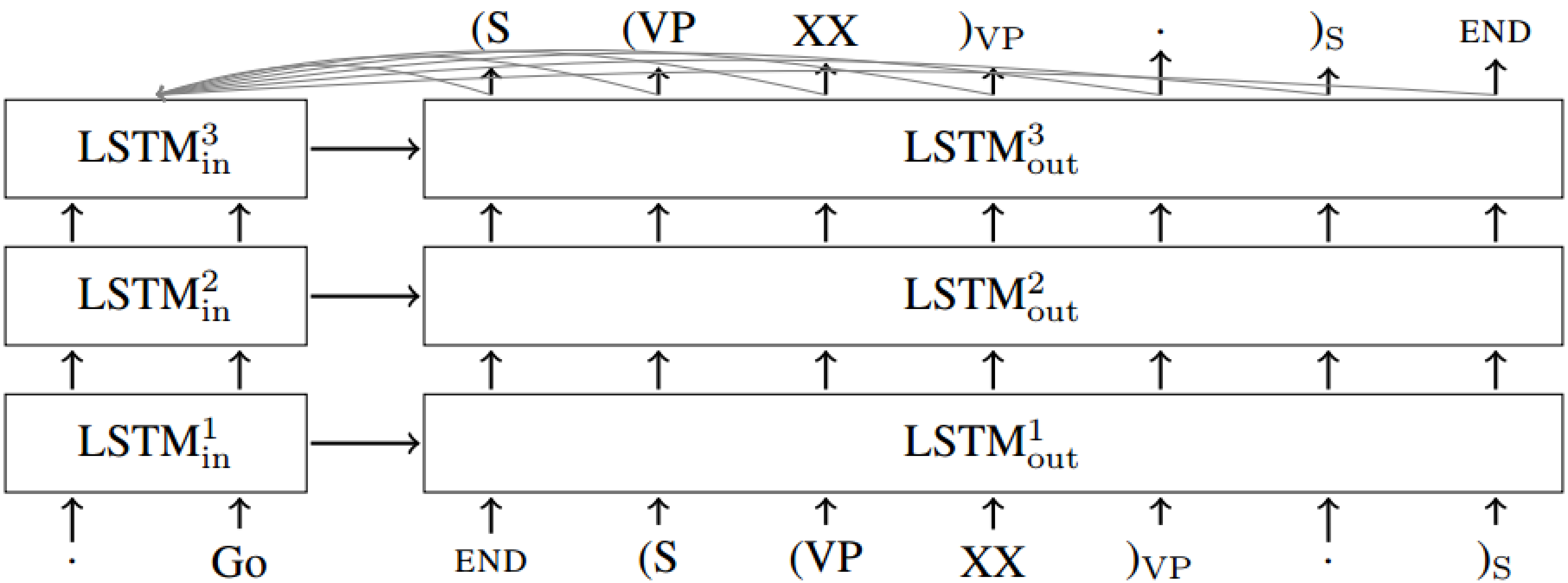
en: The $\underline{unk}$ portico in $\underline{unk}$ . . .

fr: Le $\underline{unkpos_1}$ $\underline{unkpos_{-1}}$ de $\underline{unkpos_1}$ . . .

Rare Word Models: Experiments

System	Vocab	Corpus	BLEU
State of the art in WMT'14 (Durrani et al., 2014)	All	36M	**37.0**
Standard MT + neural components			
Schwenk (2014) – neural language model	All	12M	33.3
Cho et al. (2014)– phrase table neural features	All	12M	34.5
Sutskever et al. (2014) – 5 LSTMs, reranking 1000-best lists	All	12M	36.5
Existing end-to-end NMT systems			
Bahdanau et al. (2015) – single gated RNN with search	30K	12M	28.5
Sutskever et al. (2014) – 5 LSTMs	80K	12M	34.8
Jean et al. (2015) – 8 gated RNNs with search + UNK replacement	500K	12M	37.2
Our end-to-end NMT systems			
Single LSTM with 4 layers	40K	12M	29.5
Single LSTM with 4 layers + PosUnk	40K	12M	31.8 (+2.3)
Single LSTM with 6 layers	40K	12M	30.4
Single LSTM with 6 layers + PosUnk	40K	12M	32.7 (+2.3)
Ensemble of 8 LSTMs	40K	12M	34.1
Ensemble of 8 LSTMs + PosUnk	40K	12M	36.9 (+2.8)
Single LSTM with 6 layers	80K	36M	31.5
Single LSTM with 6 layers + PosUnk	80K	36M	33.1 (+1.6)
Ensemble of 8 LSTMs	80K	36M	35.6
Ensemble of 8 LSTMs + PosUnk	80K	36M	**37.5 (+1.9)**

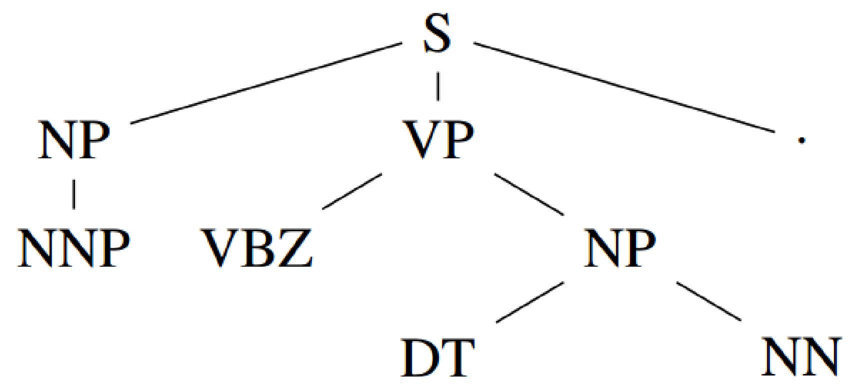
Grammar as Translation (Vinyals et al ‘15)



- Extended Sequence-to-Sequence Model
  - Two LSTMs + Attention (Bahdanau et al ‘15)
- Linearizing parse trees: Depth-first traversal

Grammar as Translation: Linearizing Parse Trees

John has a dog . $\rightarrow$

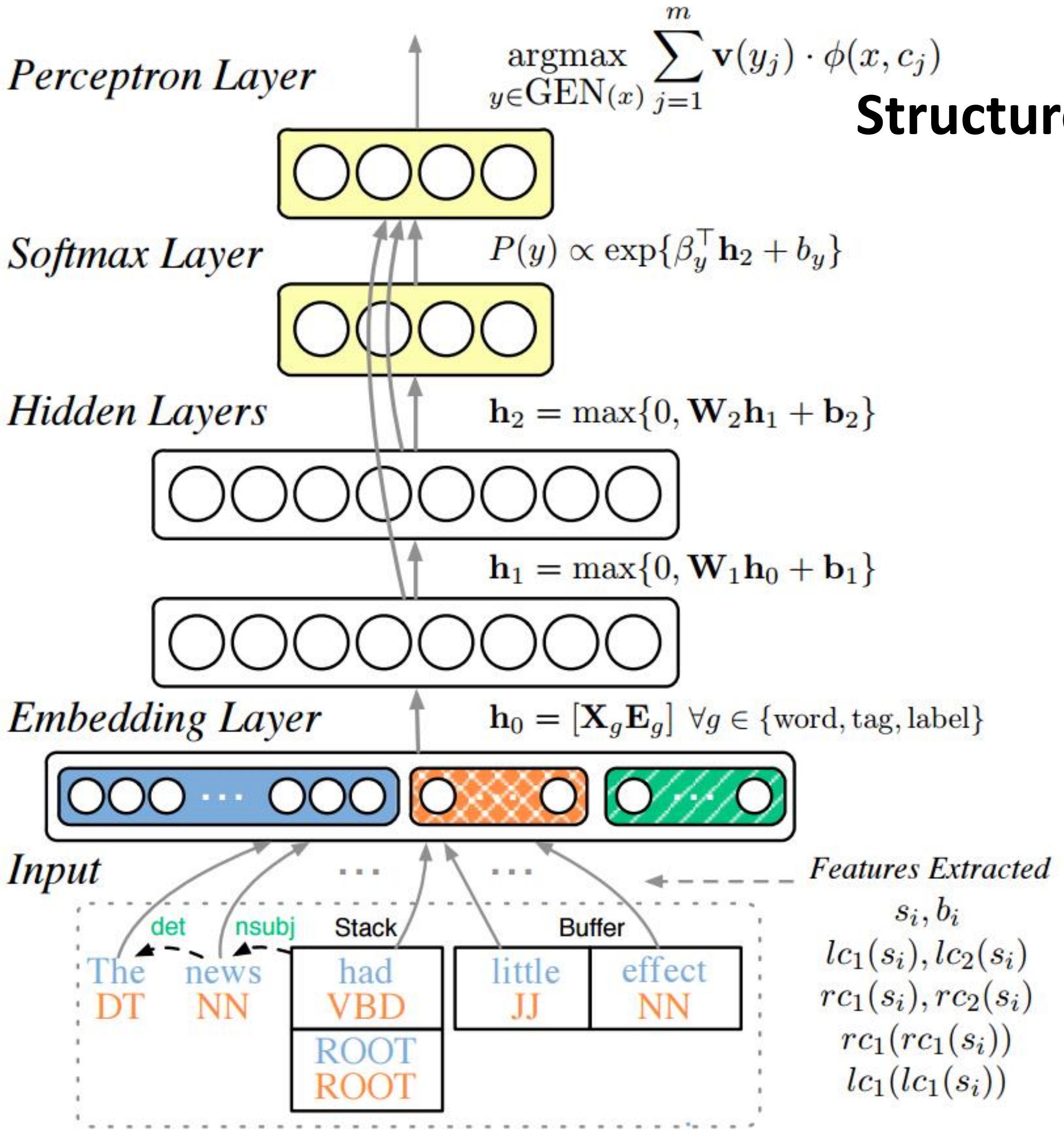


John has a dog . $\rightarrow$ (S (NP NNP $)_{NP}$ (VP VBZ (NP DT NN $)_{NP}$ $)_{VP}$. $)_S$

Sequence-to-Sequence Model: Parsing - Experiment Results

Parser	Training Set	WSJ 22	WSJ 23
baseline LSTM+D	WSJ only	< 70	< 70
LSTM+A+D	WSJ only	88.7	88.3
LSTM+A+D ensemble	WSJ only	90.7	90.5
baseline LSTM	BerkeleyParser corpus	91.0	90.5
LSTM+A	high-confidence corpus	93.3	92.5
LSTM+A ensemble	high-confidence corpus	**93.5**	**92.8**
Petrov et al. (2006) [12]	WSJ only	91.1	90.4
Zhu et al. (2013) [13]	WSJ only	N/A	90.4
Petrov et al. (2010) ensemble [14]	WSJ only	92.5	91.8
Zhu et al. (2013) [13]	semi-supervised	N/A	91.3
Huang & Harper (2009) [15]	semi-supervised	N/A	91.3
McClosky et al. (2006) [16]	semi-supervised	92.4	92.1
Huang & Harper (2010) ensemble [17]	semi-supervised	92.8	92.4

Neural Network Transition-Based Parsing (Weiss et al ‘15)

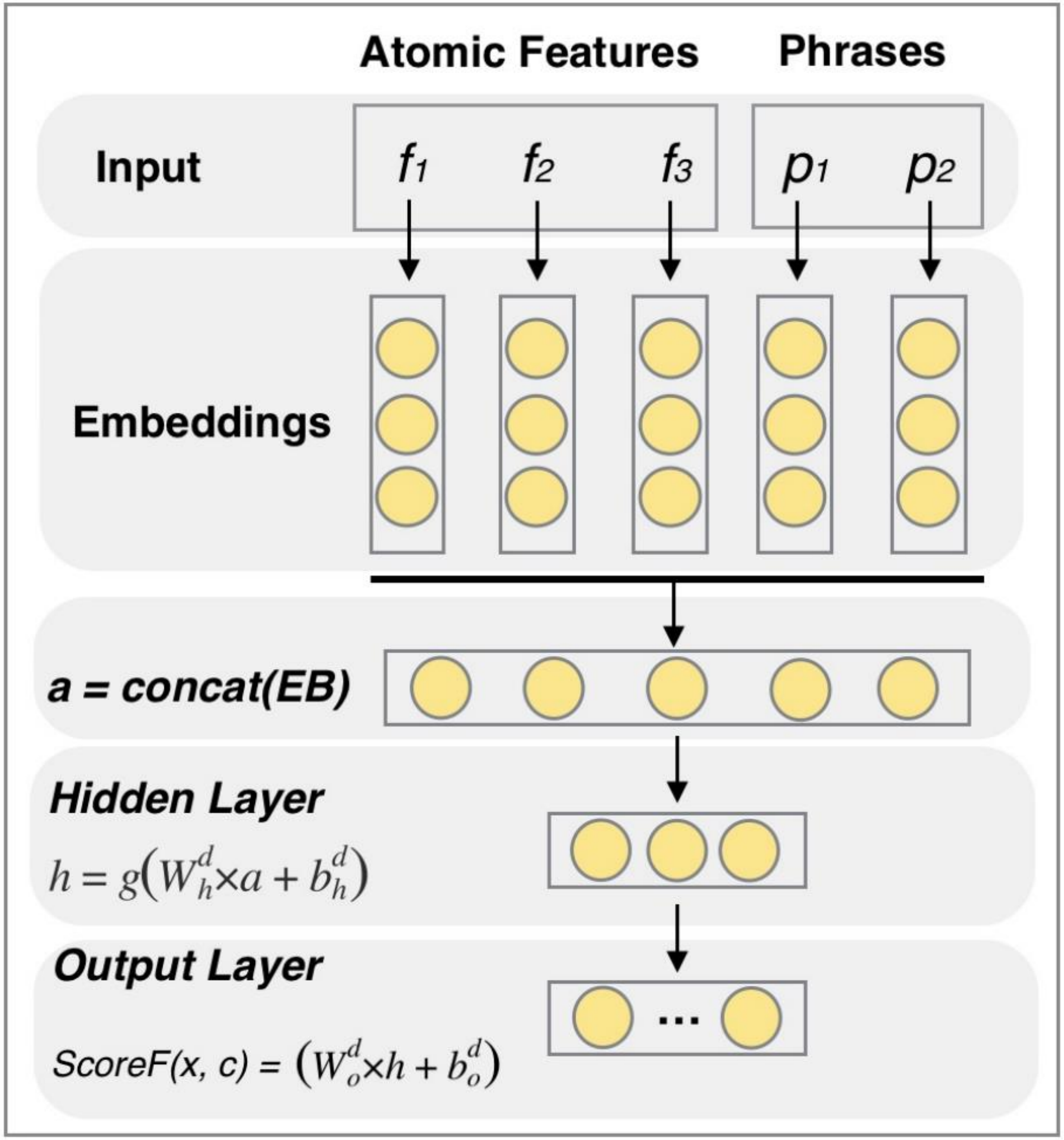


Structured learning!

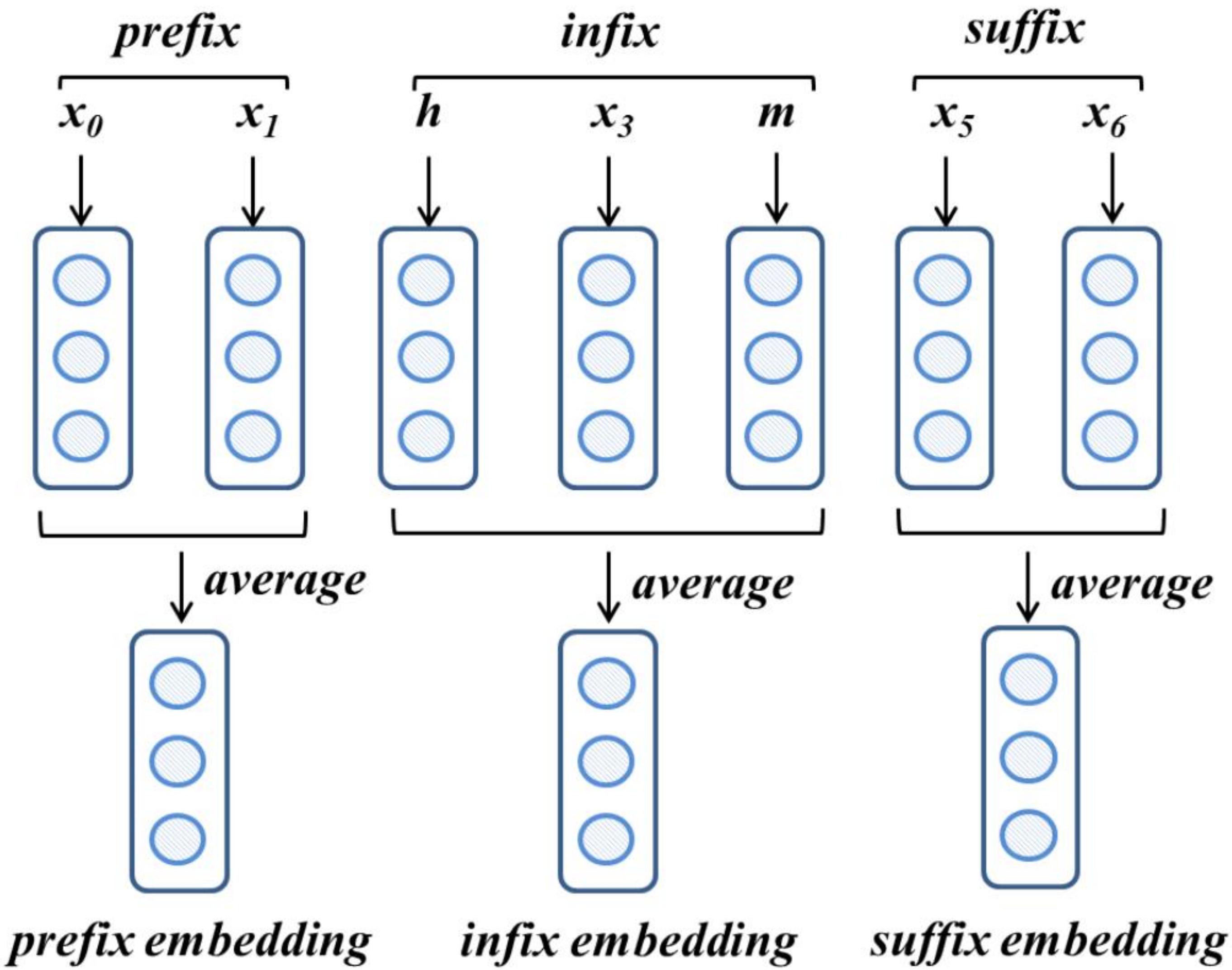
NN Transition-Based Parsing: Experiments

Method	UAS	LAS	Beam
Graph-based			
Bohnet (2010)	92.88	90.71	n/a
Martins et al. (2013)	92.89	90.55	n/a
Zhang and McDonald (2014)	93.22	91.02	n/a
Transition-based			
⋆Zhang and Nivre (2011)	93.00	90.95	32
Bohnet and Kuhn (2012)	93.27	91.19	40
Chen and Manning (2014)	91.80	89.60	1
S-LSTM (Dyer et al., 2015)	93.20	90.90	1
Our Greedy	93.19	91.18	1
Our Perceptron	**93.99**	**92.05**	8
Tri-training			
⋆Zhang and Nivre (2011)	92.92	90.88	32
Our Greedy	93.46	91.49	1
Our Perceptron	**94.26**	**92.41**	8

Neural Network Graph-Based Parsing (Pei et al '15)



Neural Network Graph-Based Parsing (Pei et al ‘15)



NN Graph-Based Parsing: Experiments

	Models	**Dev**		Test		Speed (sent/s)
		UAS	LAS	UAS	LAS	
First-order	MSTParser-1-order	92.01	90.77	91.60	90.39	20
	1-order-atomic-rand	92.00	90.71	91.62	90.41	**55**
	1-order-atomic	92.19	90.94	92.19	92.19	**55**
	1-order-phrase-rand	92.47	91.19	92.25	91.05	26
	1-order-phrase	**92.82**	**91.48**	**92.59**	**91.37**	26
Second-order	MSTParser-2-order	92.70	91.48	92.30	91.06	14
	2-order-phrase-rand	93.39	92.10	92.99	91.79	10
	2-order-phrase	**93.57**	**92.29**	**93.29**	**92.13**	10
Third-order	(Koo and Collins, 2010)	93.49	N/A	93.04	N/A	N/A

Discussion: Research Topics

- **Neural Machine Translation**
  - DNNs represent words, phrases, and sentences in continuous space
  - How to utilize more syntactic knowledge?
    - Recursive recurrent NN (Liu et al ‘14)
  - Deep architecture may approximately represents input structure
- **Neural Dependency Parsing**
- **Neural Language Model**

Reference

- Mikolov Tomá¹, Karafiát Martin, Burget Luká¹, Èernocký Jan, Khudanpur Sanjeev: Recurrent neural network based language model, In: Proceedings of the 11th Annual Conference of the International Speech Communication Association (INTERSPEECH 2010)
- Yoshua Bengio, Jerome Louradour, Ronan Collobert and Jason Weston, Curriculum Learning, ICML '09
- Yoshua Bengio, Réjean Ducharme and Pascal Vincent, A Neural Probabilistic Language Model, NIPS '01
- Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, Yoshua Bengio, Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation , 2014
- S. Hochreiter, J. Schmidhuer, "Long short-term memory," Neural Computation, vol.9, no.8, pp.1735- 1780, 1997.
- F. A. Gers, N. N. Schraudolph, J. Schmidhuer, Learning Precise Timing with LSTM Recurrent Networks, JMLR, 2002
- Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, Yoshua Bengio, Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, 2014
- Ilya Sutskever, Oriol Vinyals, Quoc V. Le, Sequence to Sequence Learning with Neural Networks, NIPS '14
- N. Kalchbrenner and Phil Blunsom, Recurrent continuous translation models, EMNLP ‘13
- Dzmitry Bahdanau, KyungHyun Cho, Yoshua Bengio, Neural Machine Translation by Jointly Learning to Align and Translate, ICLR ‘15
- Minh-Thang Luong, Ilya Sutskever, Quoc V. Le, Oriol Vinyals, Wojciech Zaremba, Addressing the Rare Word Problem in Neural Machine Translation, ALC ’15
- S´ebastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2015. On using very large target vocabulary for neural machine translation. In ACL.
- Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, Geoffrey Hinton, Grammar as a Foreign Language, 15
- David Weiss, Chris Alberti, Michael Collins, Slav Petrov, Structured Training for Neural Network Transition-Based Parsing, ACL '15
- Wenzhe Pei, Tao Ge, Baobao Chang, An Effective Neural Network Model for Graph-based Dependency Parsing, EMNLP '15
- Shujie Liu, Nan Yang, Mu Li, and Ming Zhou, A Recursive Recurrent Neural Network for Statistical Machine Translation, ACL '14
- Jiajun Zhang and Chengqing Zong, Deep Neural Networks in Machine Translation: An Overview, IEEE Intelligent Systems ‘15