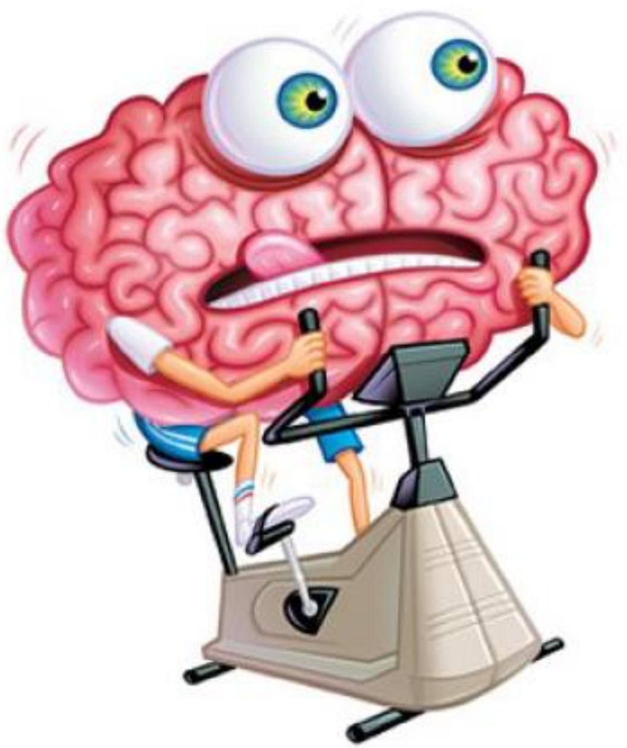
DNN(**D**eep **N**eural **N**etwork) for Natural Language Processing



Shujie Liu

Microsoft Research Asia

Natural Language Processing Group

Outline

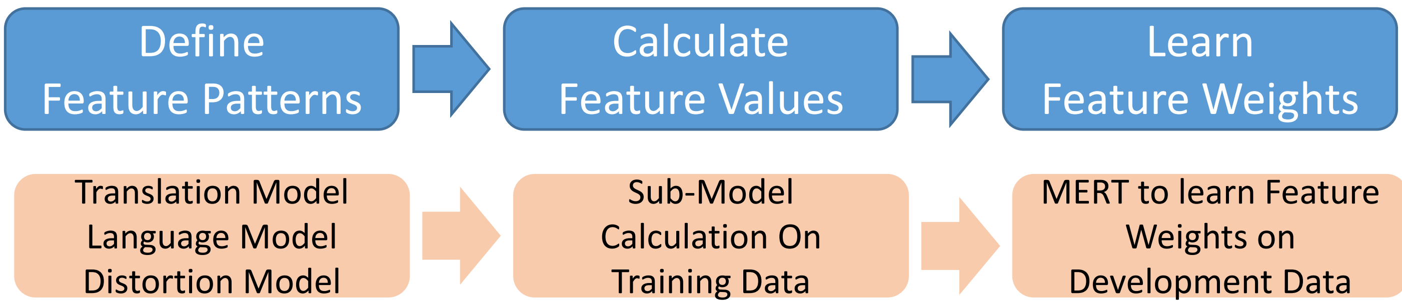
- Representation Learning
- Introduction to DNN
- DNN for Natural Language Processing

Outline

- **Representation Learning**
  - **Why Learning Representation**
  - **How to Learn Representation**
- Introduction to DNN
- DNN for Natural Language Processing

Why Learning Representation

- A popular way to build a ML model (crf, svm)



  - Feature patterns/ Sub-models are defined by human knowledge.
  - Feature engineering is important (most papers)
- Representation/Feature Learning
  - Learning information of the data that make it easier to extract useful information when building classifiers (Yoshua Bengio, et al., 2012)

How to Learn Representation

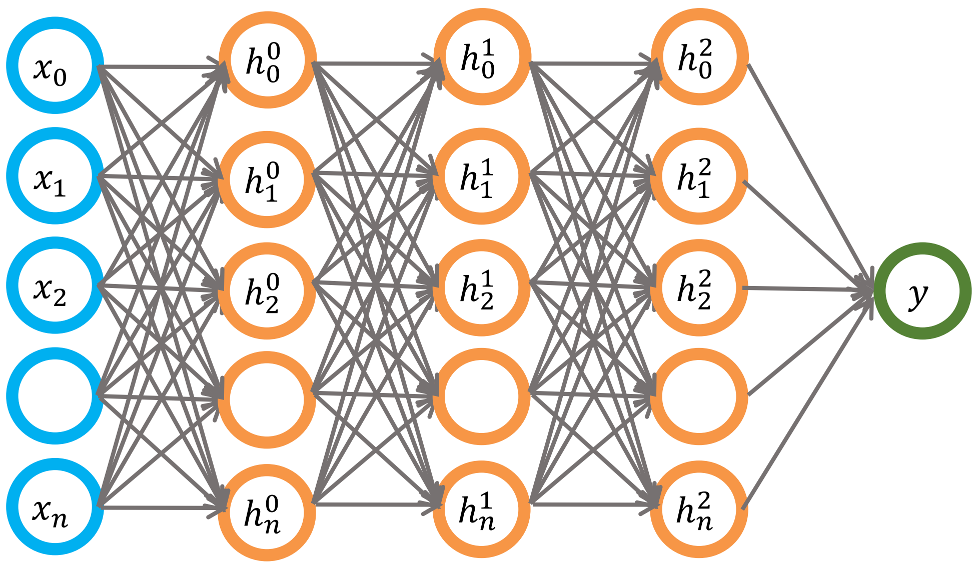
- PCA (Principal Components Analysis)
- Cluster Analysis
- Sparse Coding
- Deep Learning
- …….

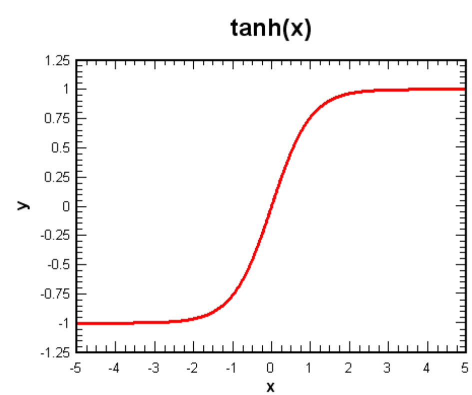
Outline

- Representation Learning First
- **Introduction to DNN**
  - **What is DNN**
  - **DNN vs. other models(perceptron, classifiers, etc.)**
  - **Why DNN now**
  - **Our understanding of DNN**
- DNN for Natural Language Processing

Deep Neural Network

- **D**eep **N**eural **N**etwork :
  - Involve multiple level neural networks
  - Non-Linear Learner
  - Automatically learn hierarchical representations from raw signals





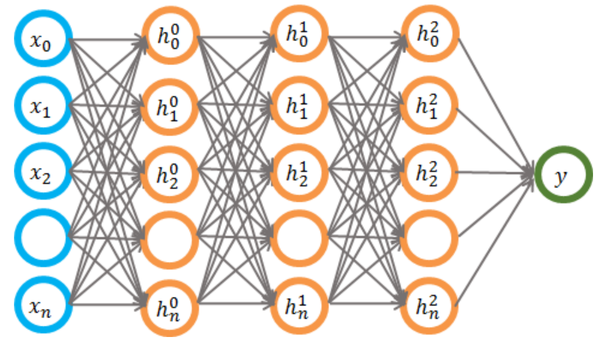
DNN vs. Perceptron/SVM/MaxEnt

- Input Feature
  - DNN consumes raw signals (or with minimal feature engineering)
  - Others requires task specific, hand-crafted features
- Classifier Type
  - DNN can handle highly non-linear space
  - Others are essentially linear classifiers (SVM uses kernels to transform input space)
- What is learned
  - DNN learns abstract representations (that can be shared with related tasks)
  - Others learn feature weights (which is task-specific)

Why Now?

Big Data

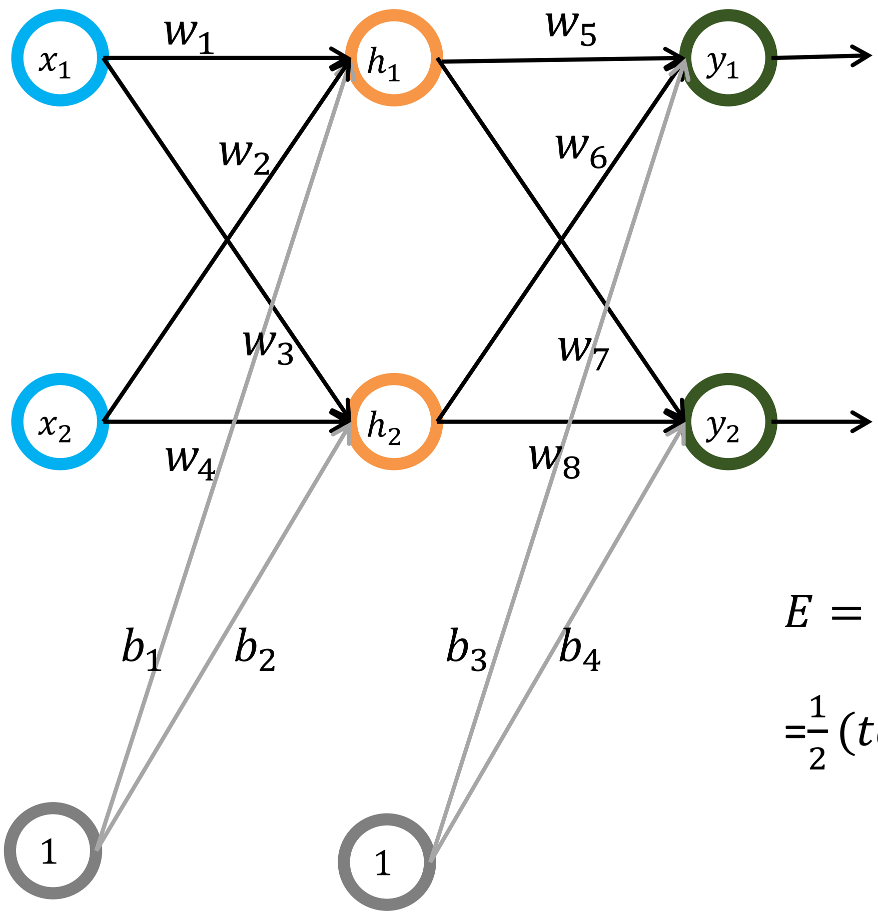
Computing Power



DNN

- Training methods: SGD, Momentum, AdaGrad, AdaDelta, Adam
- Tricks: Parameter initialization, Early stopping, Dropout, Regularization, Active functions, Hyper-parameter optimization, Weight and Gradient normalization….

Back Propagation Training (by Example)



$$net_{h1} = w_1 * x_1 + w_2 * x_2 + b_1 * 1$$

$$out_{h1} = \frac{1}{1 + e^{-net_{h1}}}$$

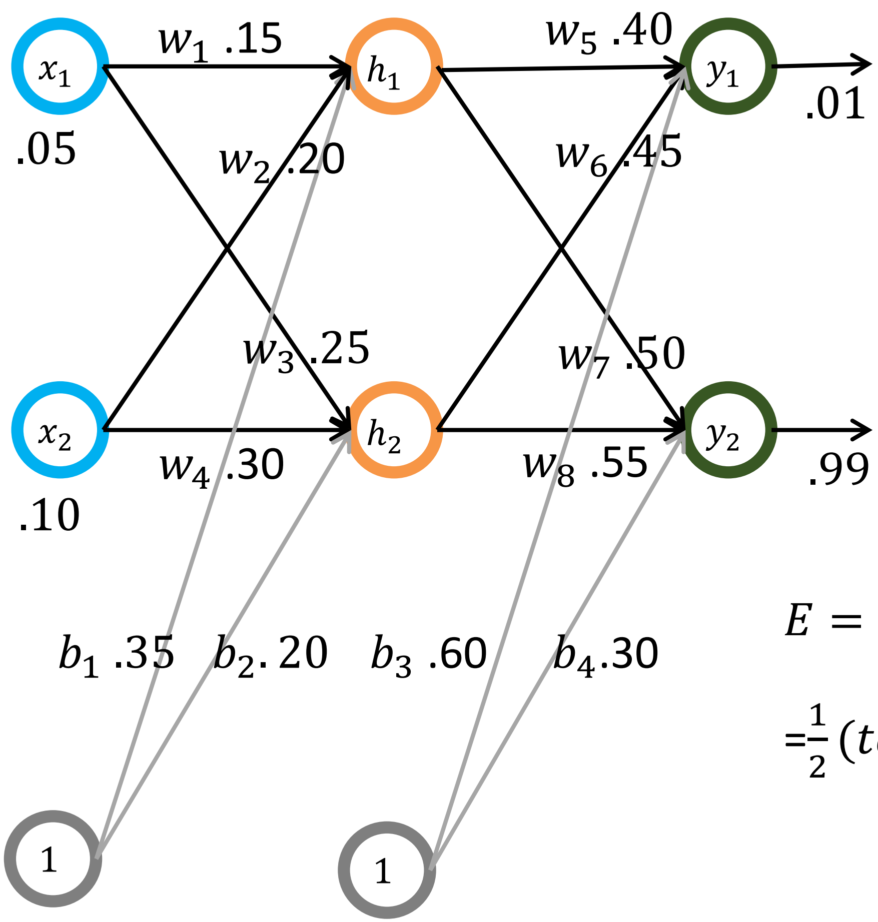
$$net_{y1} = w_5 * net_{h1} + w_6 * net_{h2} + b_3 * 1$$

$$out_{y1} = \frac{1}{1 + e^{-net_{y1}}}$$

$$E = \sum \frac{1}{2}(target - output)^2$$

$$= \frac{1}{2}(target_{y1} - out_{y1})^2 + \frac{1}{2}(target_{y2} - out_{y2})^2$$

Back Propagation Training (by Example)



$$net_{h1} = w_1 * x_1 + w_2 * x_2 + b_1 * 1$$

$$out_{h1} = \frac{1}{1 + e^{-net_{h1}}}$$

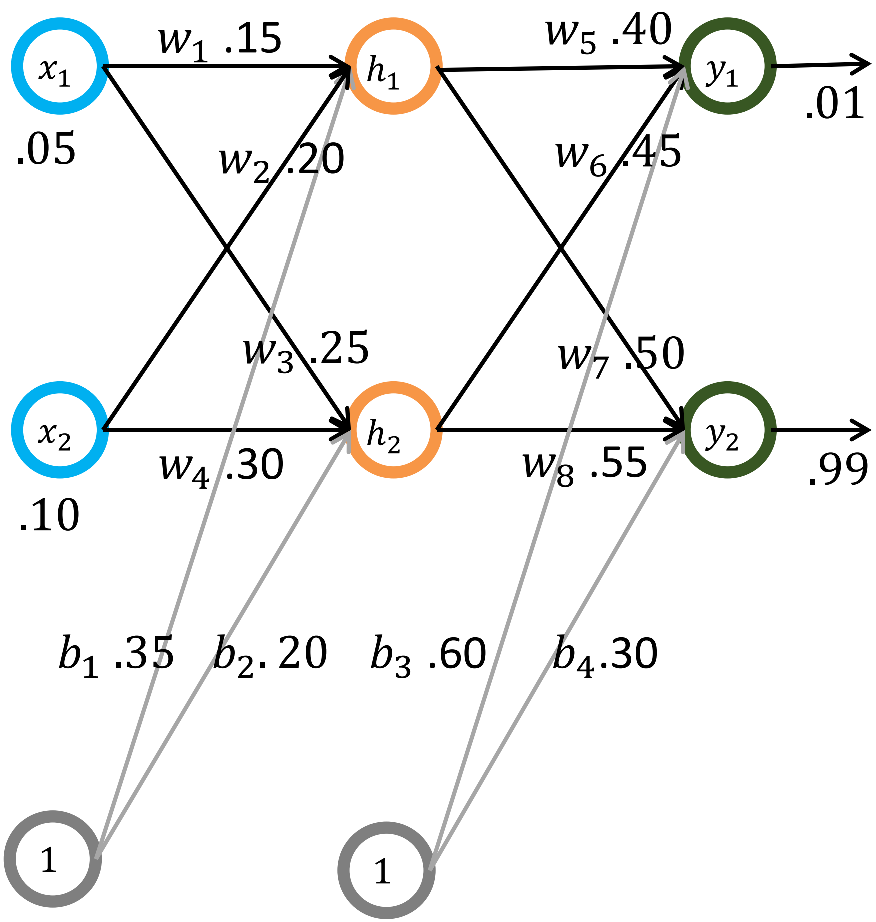
$$net_{y1} = w_5 * net_{h1} + w_6 * net_{h2} + b_3 * 1$$

$$out_{y1} = \frac{1}{1 + e^{-net_{y1}}}$$

$$E = \sum \frac{1}{2}(target - output)^2$$

$$= \frac{1}{2}(target_{y1} - out_{y1})^2 + \frac{1}{2}(target_{y2} - out_{y2})^2$$

Back Propagation Training (by Example)



- Forward process:

$net_{h1} = w_1 * x_1 + w_2 * x_2 + b_1 * 1$

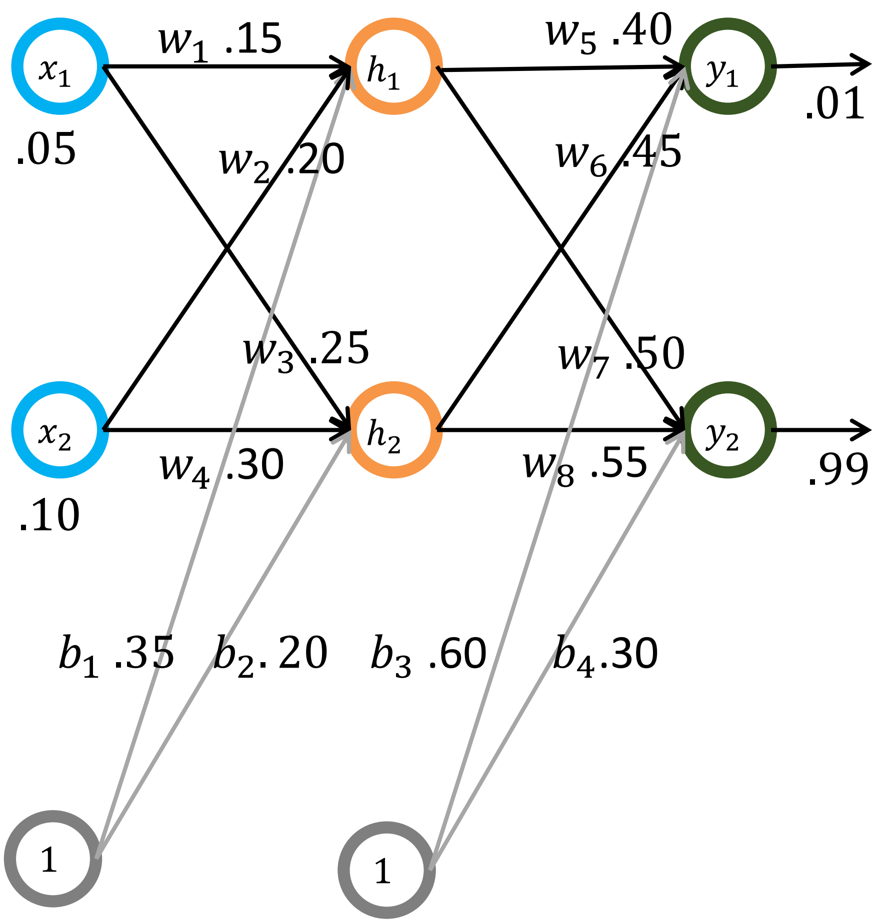
=0.15*0.05+0.2*0.1+0.35*1=0.3775

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}}$$

=0.593269992

$out_{h2} = 0.596884378$

Back Propagation Training (by Example)



- Forward process:

$$net_{h1} = w_1 * x_1 + w_2 * x_2 + b_1 * 1$$
=0.15*0.05+0.2*0.1+0.35*1=0.3775

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}}$$
=0.593269992

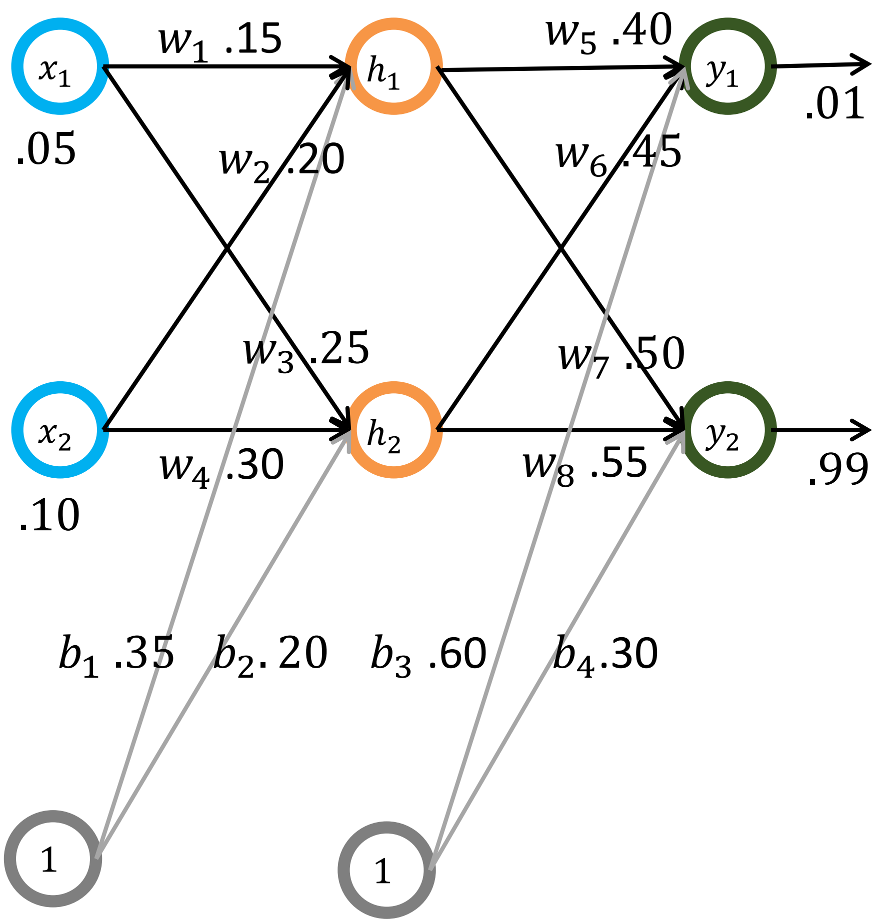
$$out_{h2} = 0.596884378$$

$$net_{y1} = w_5 * net_{h1} + w_6 * net_{h2} + b_3 * 1$$
=0.4*0.593269992+0.45*0.596884378+0.6*1
=1.105905967

$$out_{y1} = \frac{1}{1+e^{-net_{y1}}} = \frac{1}{1+e^{-1.105905967}}$$
=0.75136507

$$out_{y2} = 0.772928465$$

Back Propagation Training (by Example)



- Calculate the error:

$$E = \sum \frac{1}{2}(target - output)^2$$

$$= \frac{1}{2}(target_{y1} - out_{y1})^2$$

$$+ \frac{1}{2}(target_{y2} - out_{y2})^2$$

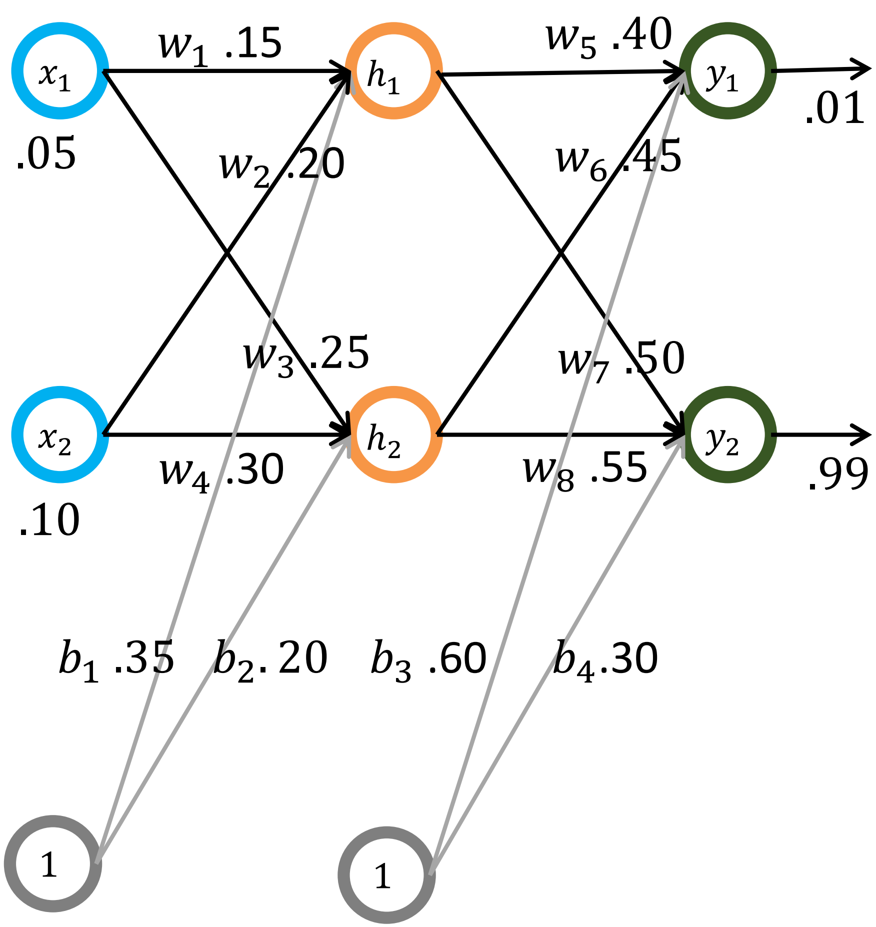
$$= \frac{1}{2}(0.01 - 0.75136507)^2$$

$$+ \frac{1}{2}(0.99 - 0.772928465\ )^2$$

$$= 0.274811083 + 0.023560026$$

$$= 0.298371109$$

Back Propagation Training (by Example)



- Backward Process:

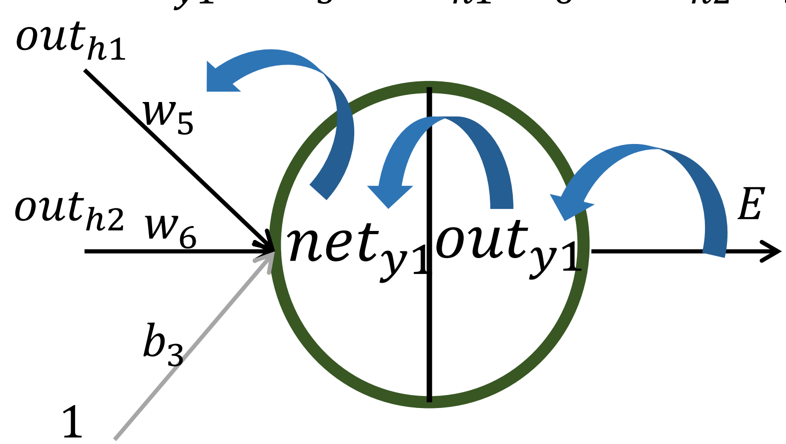
$$\frac{\partial E}{\partial w_5} = \frac{\partial E}{\partial out_{y1}} * \frac{\partial out_{y1}}{\partial net_{y1}} * \frac{\partial net_{y1}}{\partial w_5}$$

$$E = E_{y1} + E_{y2}$$

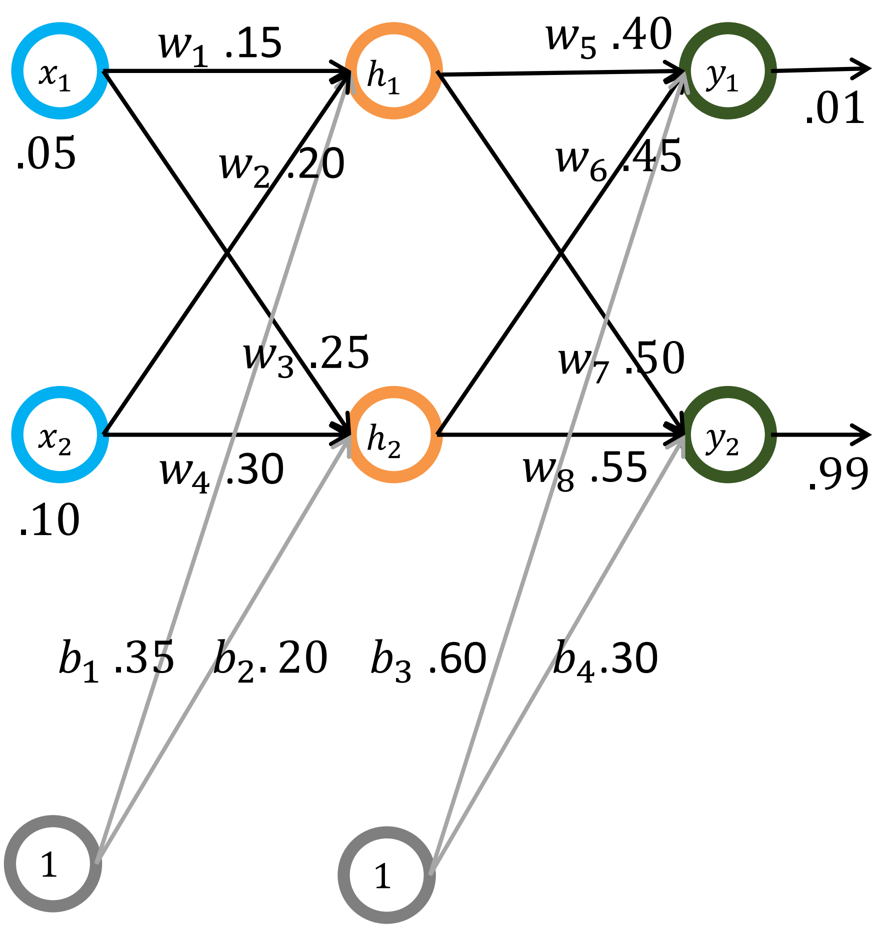
$$E_{y1} = \frac{1}{2}(target_{y1} - out_{y1})^2$$

$$out_{y1} = \frac{1}{1 + e^{-net_{y1}}}$$

$$net_{y1} = w_5 * net_{h1} + w_6 * net_{h2} + b_3 * 1$$



Back Propagation Training (by Example)



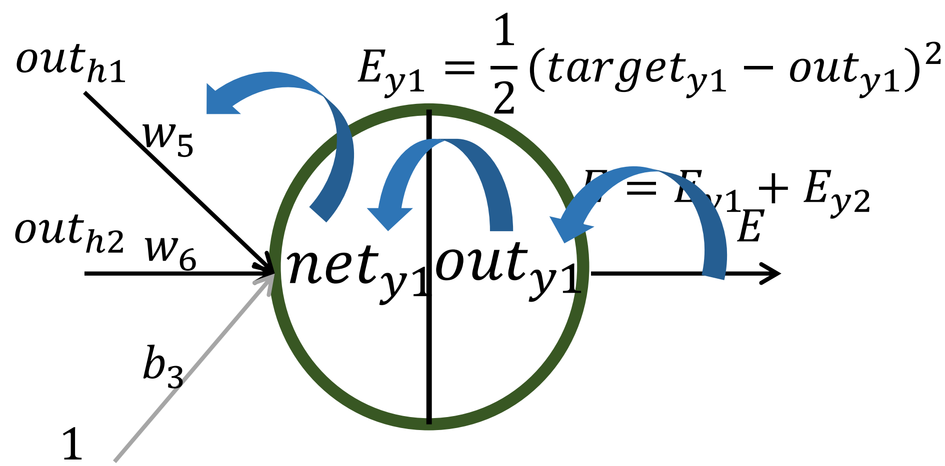
- Backward Process:

$$\frac{\partial E}{\partial w_5} = \frac{\partial E}{\partial out_{y1}} * \frac{\partial out_{y1}}{\partial net_{y1}} * \frac{\partial net_{y1}}{\partial w_1}$$

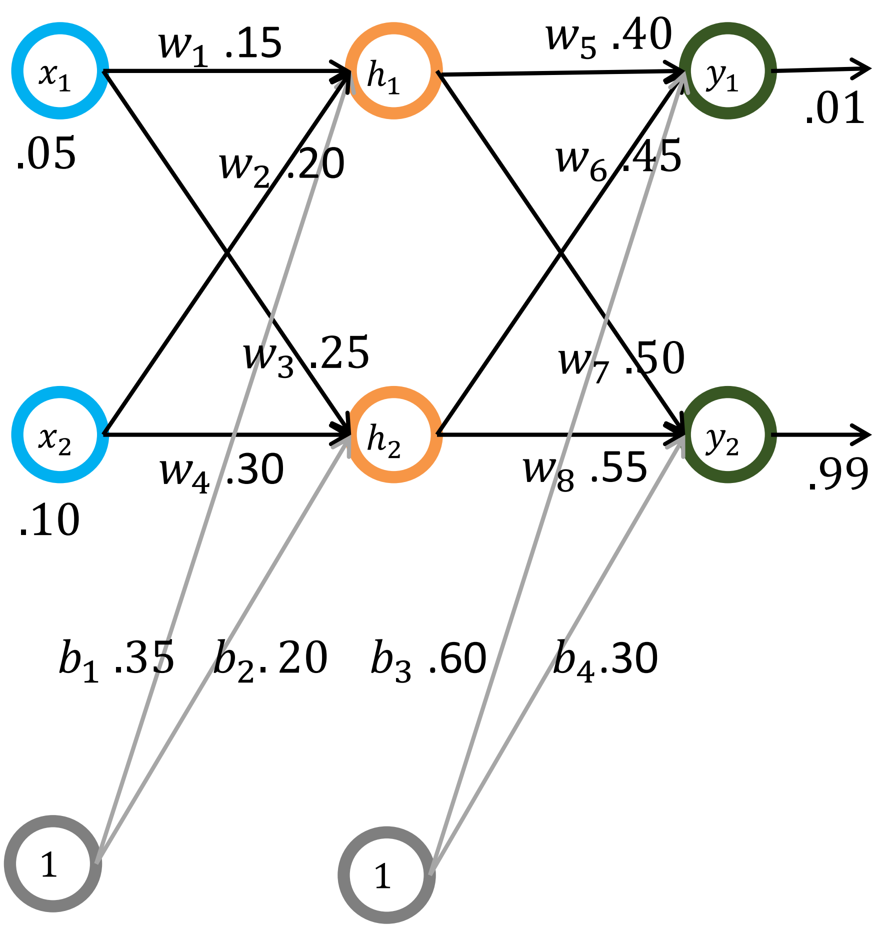
$$\frac{\partial E}{\partial out_{y1}} = -2 * \frac{1}{2}\left(target_{y1} - out_{y1}\right)$$

$$\frac{\partial out_{y1}}{\partial net_{y1}} = out_{y1}\left(1 - out_{y1}\right)$$

$$\frac{\partial net_{y1}}{\partial w_1} = out_{h1}$$



Back Propagation Training (by Example)



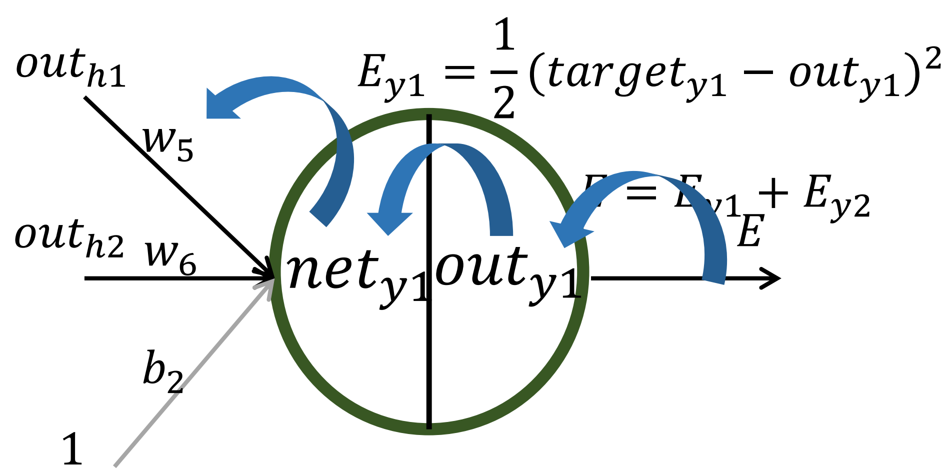
- Backward Process:

$$\frac{\partial E}{\partial w_5} = \frac{\partial E}{\partial out_{y1}} * \frac{\partial out_{y1}}{\partial net_{y1}} * \frac{\partial net_{y1}}{\partial w_5}$$

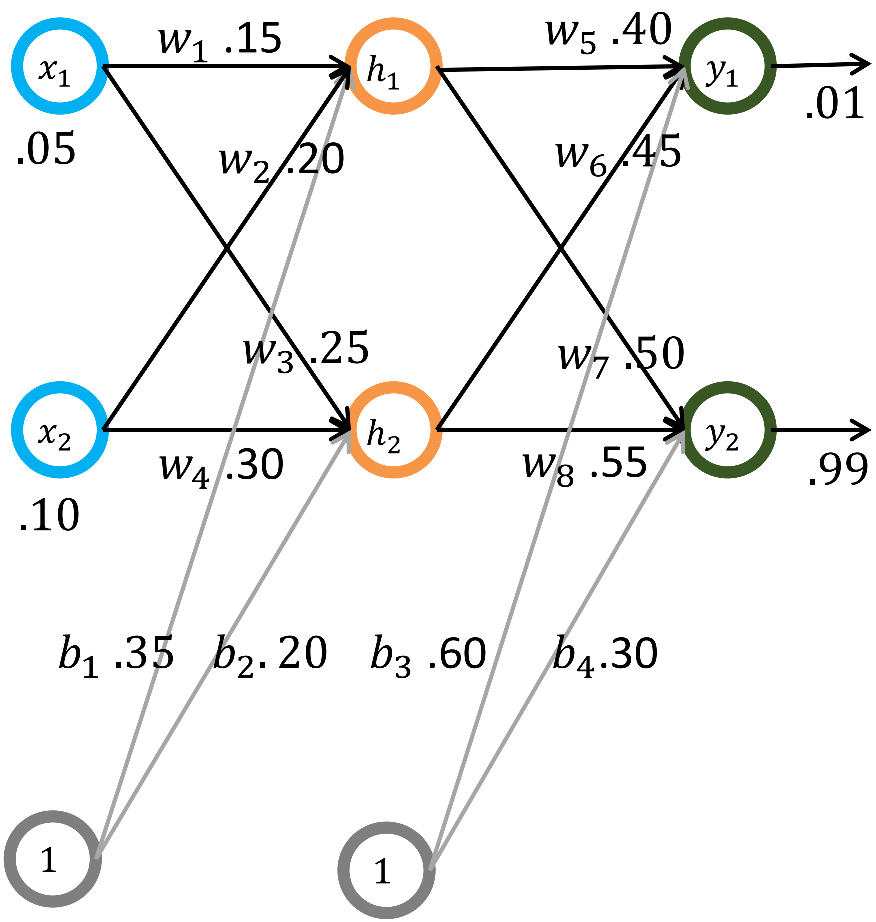
$= -(target_{y1} - out_{y1})$

$*out_{y1}(1 - out_{y1})$

$*out_{h1}$



Back Propagation Training (by Example)



- Backward Process:

$$\frac{\partial E}{\partial w_5} = \frac{\partial E}{\partial out_{y1}} * \frac{\partial out_{y1}}{\partial net_{y1}} * \frac{\partial net_{y1}}{\partial w_5}$$

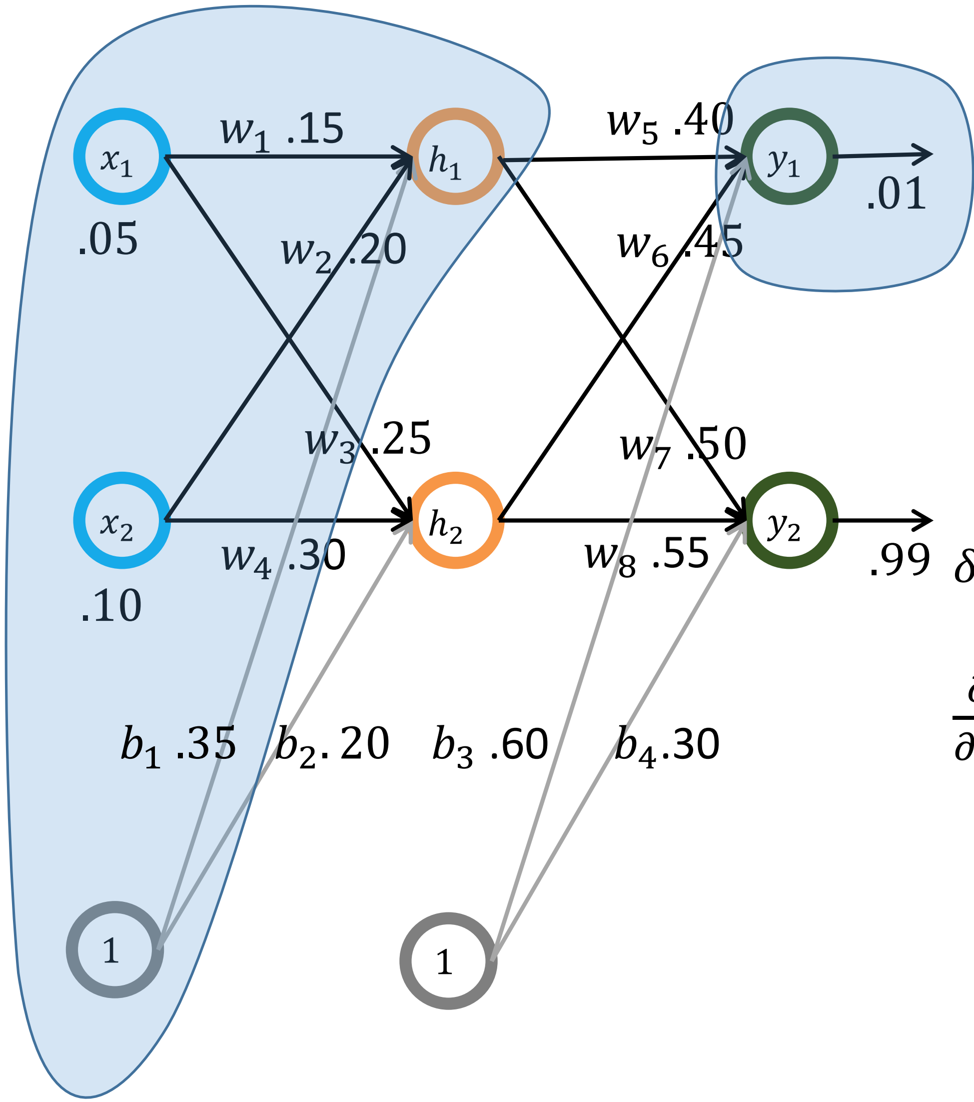
$$\left.\begin{aligned} &= -(target_{y1} - out_{y1}) \\ &*out_{y1}(1 - out_{y1}) \end{aligned}\right\} \delta_{\,y1}$$

$$\left.*out_{h1}\right\} out_{h1}$$

$$\delta_{\,y1} = (target_{y1} - out_{y1}) * out_{y1}(1 - out_{y1})$$

$$\frac{\partial E}{\partial w_5} = \delta_{\,y1} * out_{h1} = 0.082167041$$

Back Propagation Training (by Example)



- Backward Process:

$$\frac{\partial E}{\partial w_5} = \frac{\partial E}{\partial out_{y1}} * \frac{\partial out_{y1}}{\partial net_{y1}} * \frac{\partial net_{y1}}{\partial w_5}$$

$$\left.\begin{matrix} = -(target_{y1} - out_{y1}) \\ * out_{y1}(1 - out_{y1}) \end{matrix}\right\} \delta_{\,y1}$$

$$\left.* out_{h1}\right\} out_{h1}$$

$$\delta_{\,y1} = (target_{y1} - out_{y1}) * out_{y1}(1 - out_{y1})$$

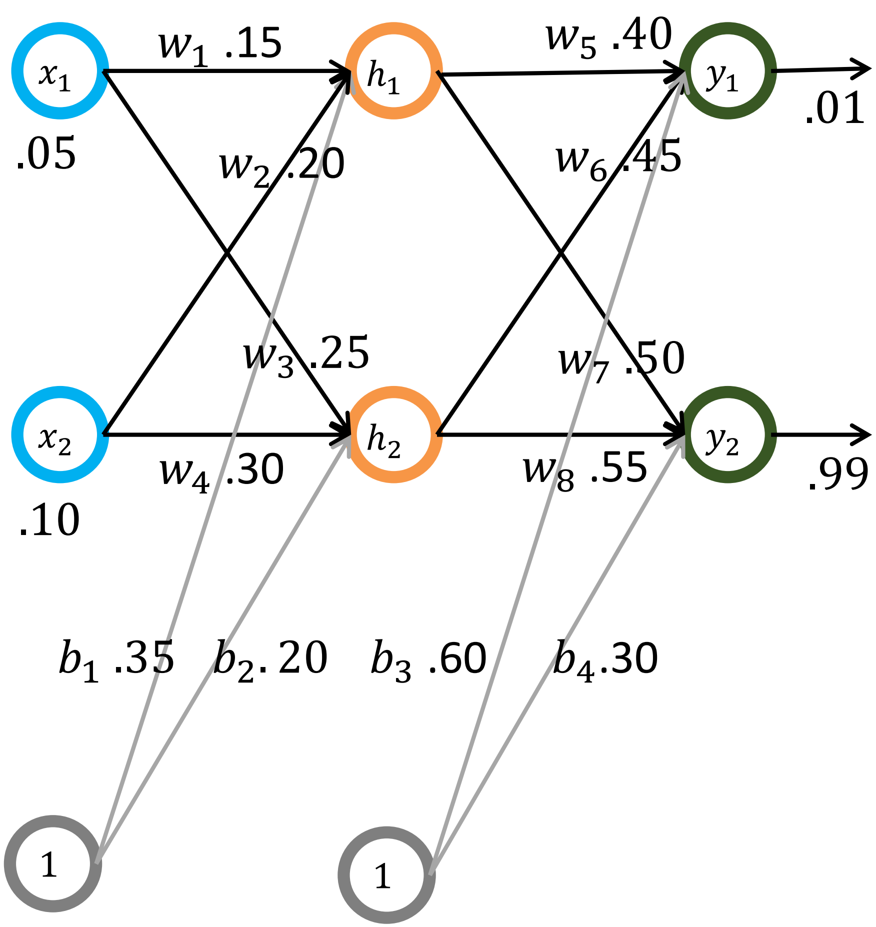
$$\frac{\partial E}{\partial w_5} = \delta_{\,y1} * out_{h1} = 0.082167041$$

- Update Process:

$$w_5^+ = w_5 - \alpha * \frac{\partial E}{\partial w_5} =$$

0.4-0.5*0.082167041=0.35891648

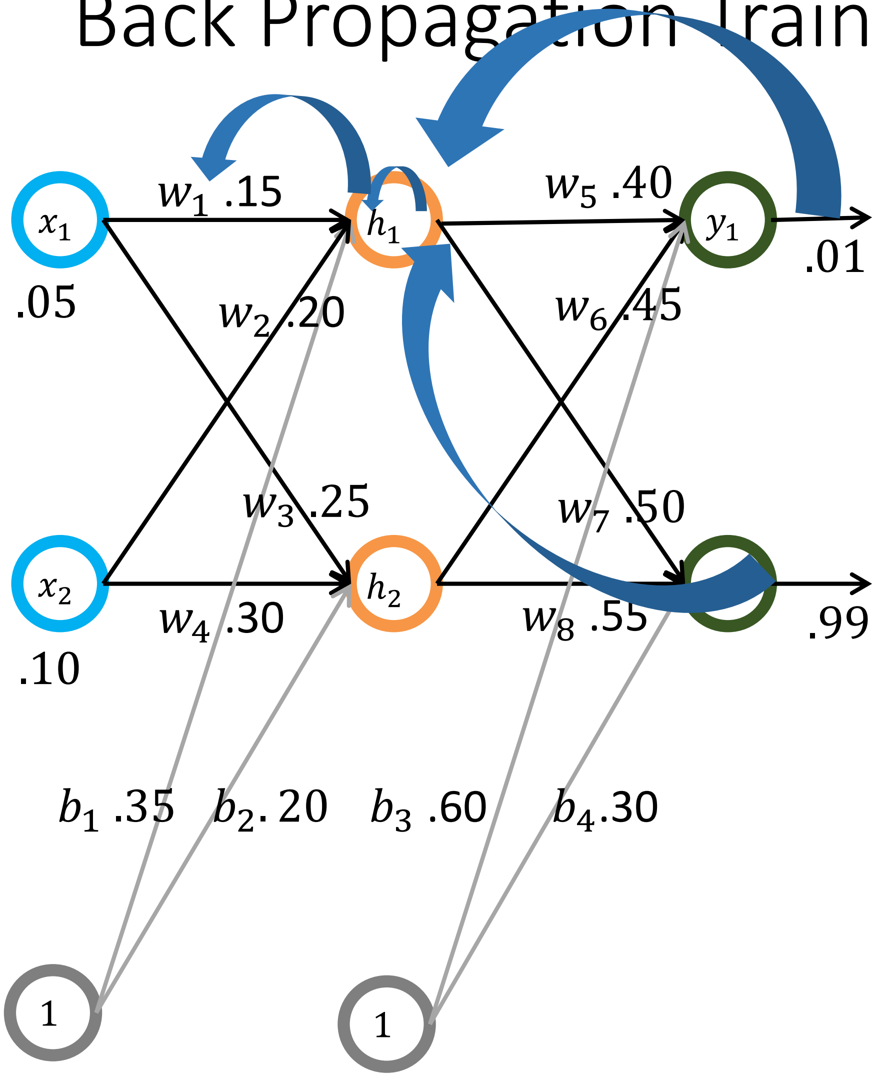
Back Propagation Training (by Example)



- Backward Process and Update Process for w_6 w_7 w_8 :

$$w_6^+ = 0.408666186$$
$$w_7^+ = 0.511301270$$
$$w_8^+ = 0.561370121$$

Back Propagation Training (by Example)



- Backward Process:

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E}{\partial out_{h1}} = \frac{\partial E_{y1}}{\partial out_{h1}} + \frac{\partial E_{y2}}{\partial out_{h1}}$$

$$\frac{\partial E_{y1}}{\partial out_{h1}} = \frac{\partial E_{y1}}{\partial net_{y1}} * \frac{\partial net_{y1}}{\partial out_{h1}}$$

$$\frac{\partial E_{y1}}{\partial net_{y1}} = 0.138498562$$

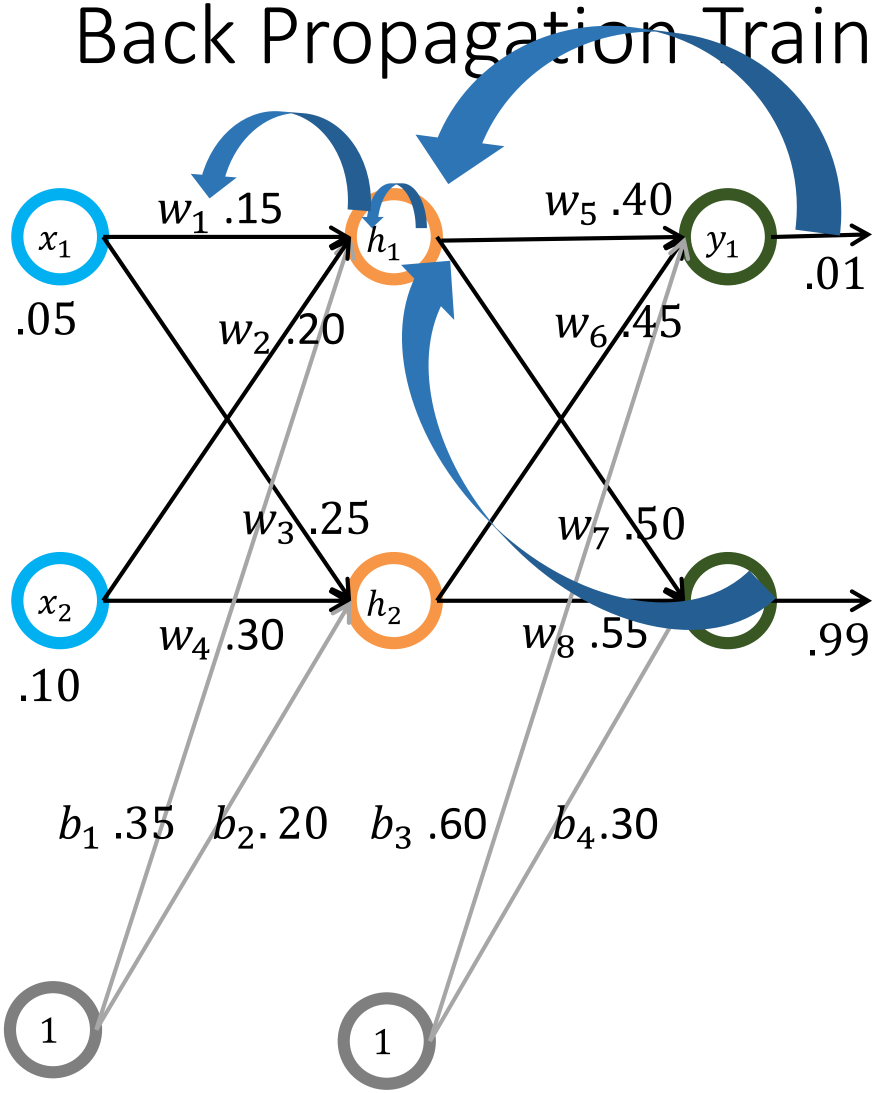
$$\frac{\partial net_{y1}}{\partial out_{h1}} = w_5$$

$$\frac{\partial E_{y1}}{\partial out_{h1}} = 0.138498562*0.4$$

=0.055399425

$$\frac{\partial E_{y2}}{\partial out_{h1}} = -0.019049119$$

Back Propagation Training (by Example)



- Backward Process:

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E}{\partial out_{h1}} = \frac{\partial E_{y1}}{\partial out_{h1}} + \frac{\partial E_{y2}}{\partial out_{h1}}$$

= 0.055399425-0.019049119

=0.036350306

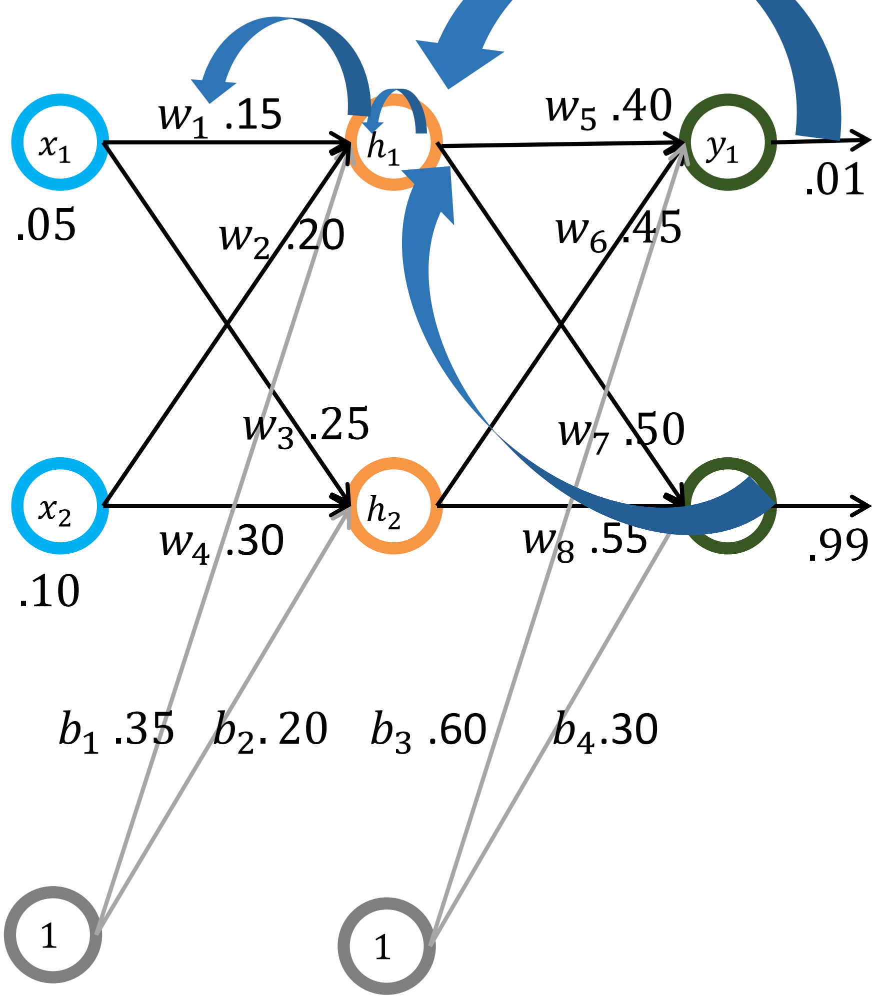
$$out_{h1} = \frac{1}{1 + e^{-net_{h1}}}$$

$\frac{\partial out_{h1}}{\partial net_{h1}}$=$out_{h1}$(1-$out_{h1}$)=0.241300709

$$net_{h1} = w_1 * x_1 + w_2 * x_2 + b_1 * 1$$

$\frac{\partial net_{h1}}{\partial w_5}$=$x_1$=0.05

Back Propagation Training (by Example)



- Backward Process:

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$= \left(\sum_i \frac{\partial E}{\partial out_{yi}} * \frac{\partial out_{yi}}{\partial net_{yi}} * \frac{\partial net_{yi}}{\partial w_1}\right) * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$= \left(\sum_i \delta_{\,yi} * w_{hi}\right) * out_{h1}(1 - out_{h1}) * x_1$$

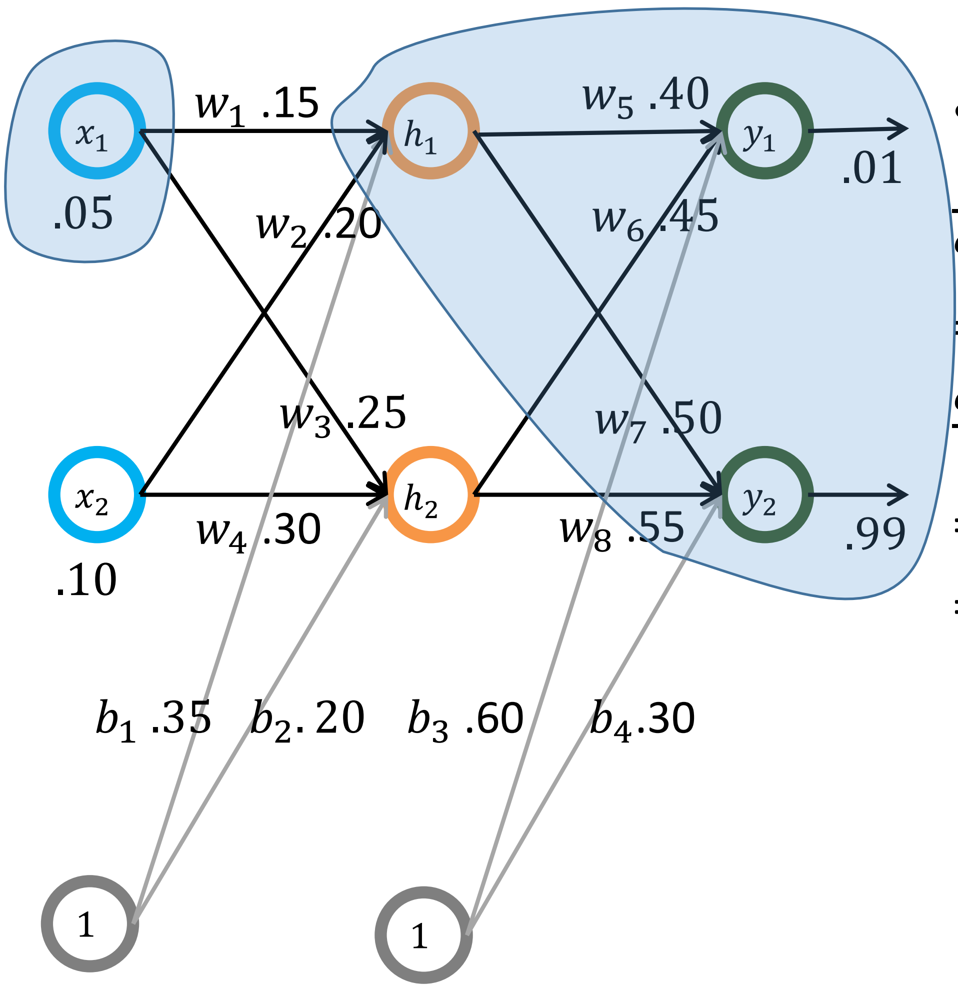
$$= \delta_{\,h1} * x_1 = 0.000438568$$

- Update Process:

$$w_1^+ = w_1 - \alpha * \frac{\partial E}{\partial w_1} =$$

0.15-0.5*0.000438568=0.149780716

Back Propagation Training (by Example)



- Backward Process:

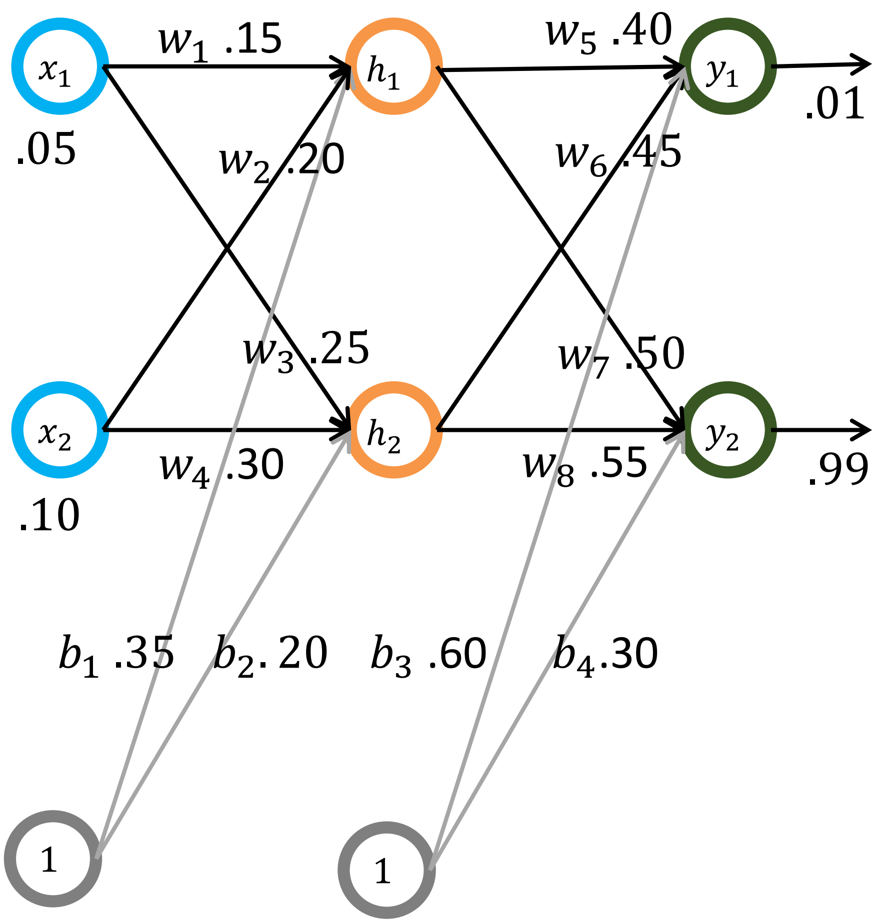
$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$= \left(\sum_i \frac{\partial E}{\partial out_{yi}} * \frac{\partial out_{yi}}{\partial net_{yi}} * \frac{\partial net_{yi}}{\partial w_1}\right) * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$= \left(\sum_i \delta_{yi} * w_{hi}\right) * out_{h1}(1 - out_{h1}) * x_1$$

$$= \delta_{h1} * x_1 = 0.000438568$$

Back Propagation Training (by Example)



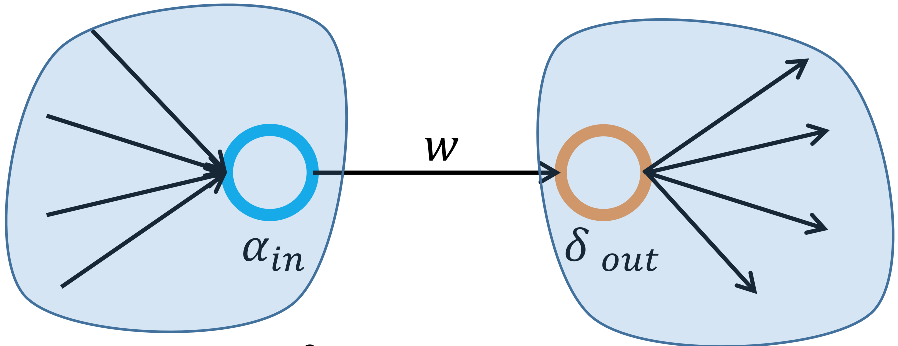
- Backward Process and Update Process for w_2 w_3 w_4 :

$$w_2^+ = 0.19956143$$
$$w_3^+ = 0.24975114$$
$$w_4^+ = 0.29950229$$

- Compute the error using new parameters:

E =0.298371109 → 0.291027

Back Propagation Training (the Rule)

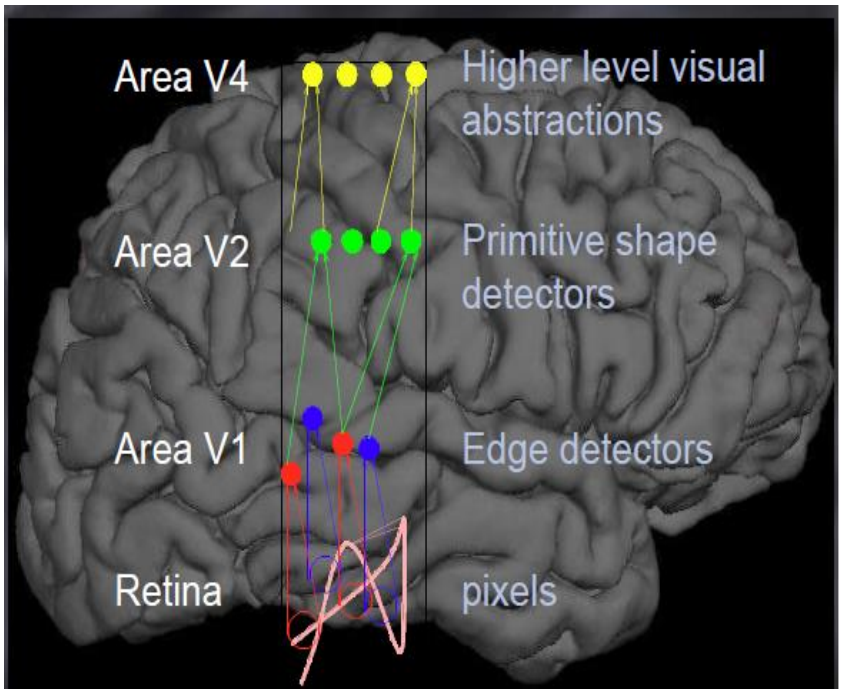


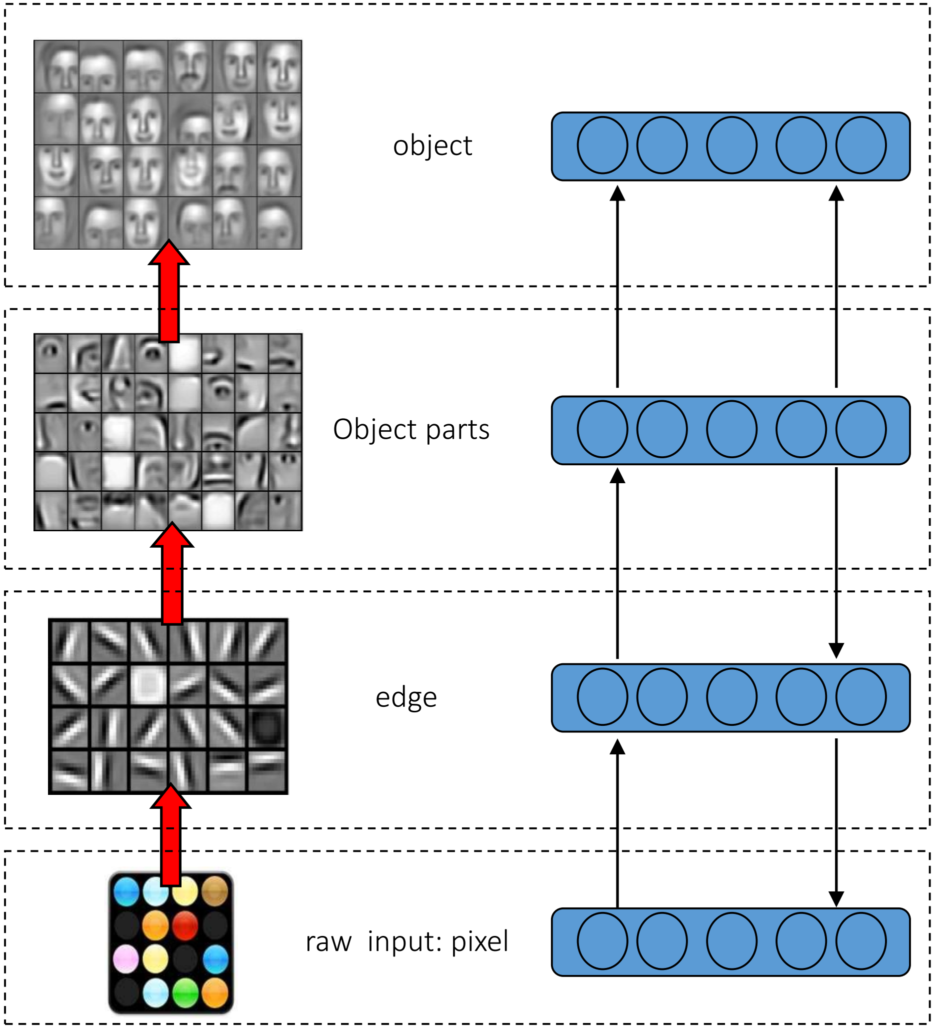
$$\frac{\partial E}{\partial w} = \alpha_{in} * \delta_{\ out}$$

α_{in} is the activation of the neuron input to the weight w

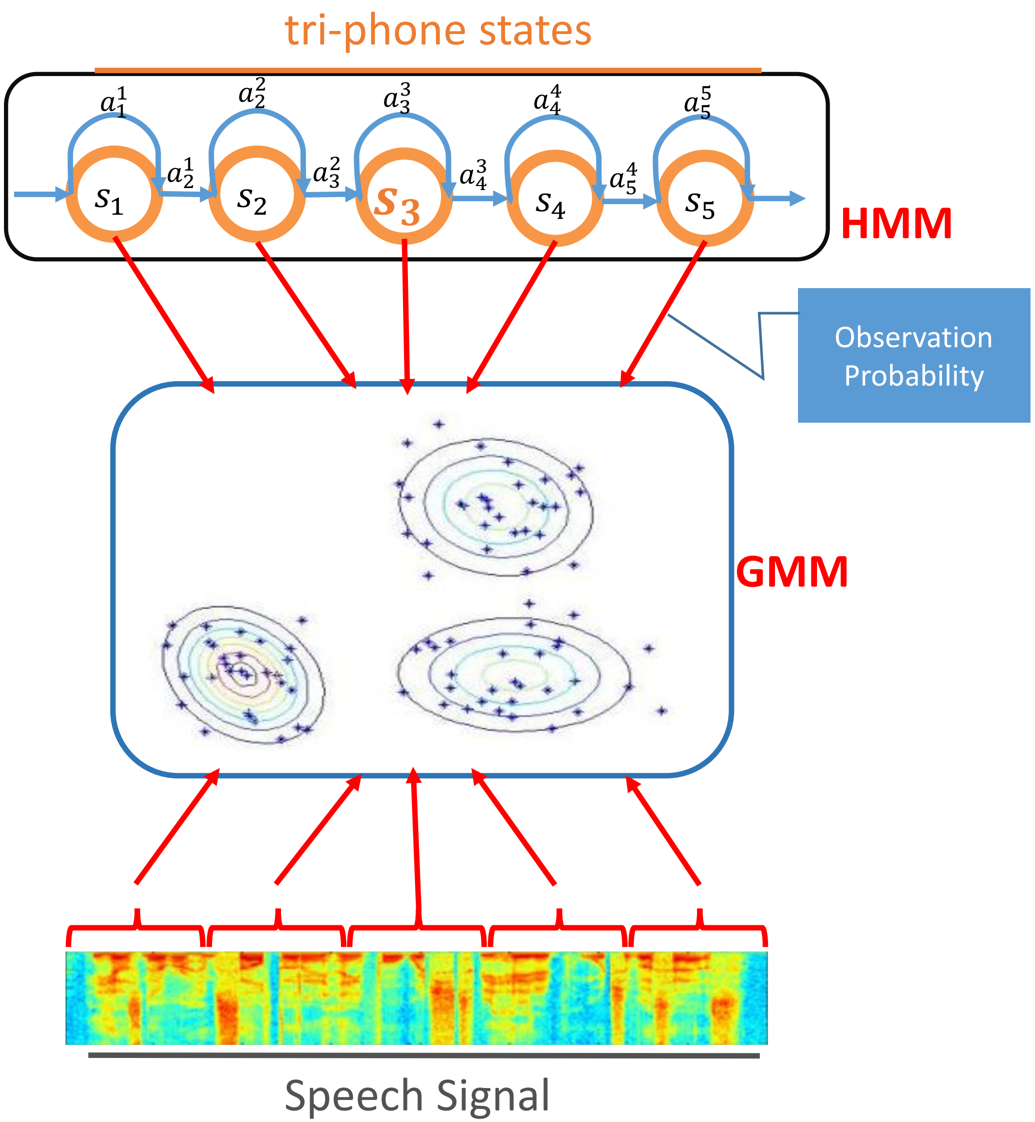
$\delta_{\ out}$ is the error of the neuron output from the weight w

DNN in Image Processing



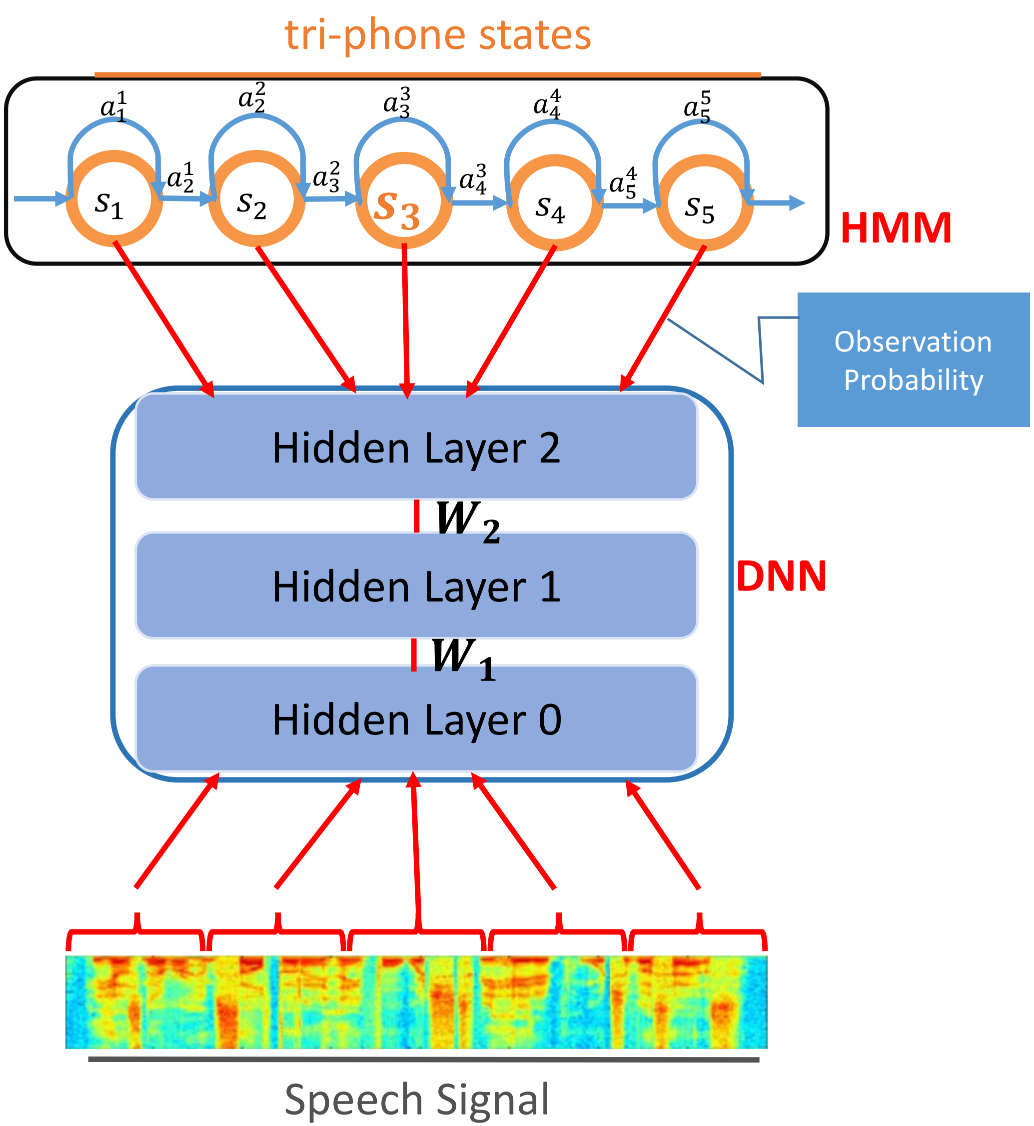


CD-GMM-HMM for Speech



- GMM: Gaussian Mixture Model, used to model observation probability

CD-DNN-HMM for Speech



- GMM: Gaussian Mixture Model, used to model observation probability
- DNN is used to replace GMM
- Forced Alignment with CD-GMM-HMM is used to get training data for DNN
- Training data:

  tri-phone/signal-frame pairs

What DNN can do: Representation/Feature Learning

- Finding multiple levels of representations
  - Dimensionality reduction: Denoising
  - Dimensionality ascension: Sparseness
  - Linear or none-linear mapping: (Kernel Function?)
- Minimal feature engineering
  - Representation(feature) varies for different tasks.
  - Transform 'factual knowledge into usable knowledge'
  - Learn good representations shared across multi tasks.

Advantage and Disadvantage of DNN

- Advantage
  - Usable features are learned automatically.
  - Multiple levels (coarse to fine) of representation.
- Disadvantage
  - Configuration and architecture is art: too many hyper-parameters and variants.
  - Features learned are not understandable.
  - Computational Complexity: time consuming

DNN in NLP

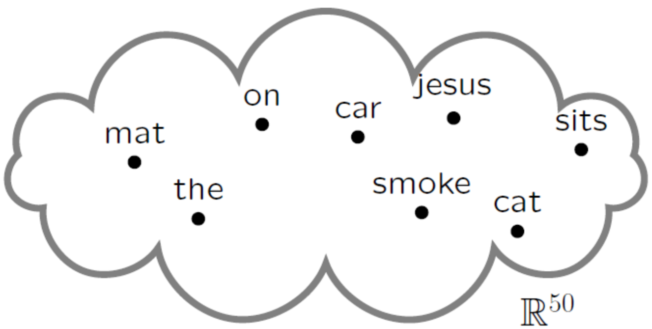
- NLP specific challenge:
  - Extremely high-dimensional space
  - Raw features (words) with strong discriminative power
  - Often highly structured output (Parsing, SMT)

Outline

- Representation Learning First
- Introduction to DNN
- DNN for Natural Language Processing
  - **DNN for Word Embedding**
  - DNN for Language Modeling
  - DNN for Machine Translation

Words into Vectors (Word Embedding)

- Words are embedded in a vector space



- Embeddings are **trained**
- Training Principle: ”You shall know a word by the company it keeps” (J. R. Firth 1957)

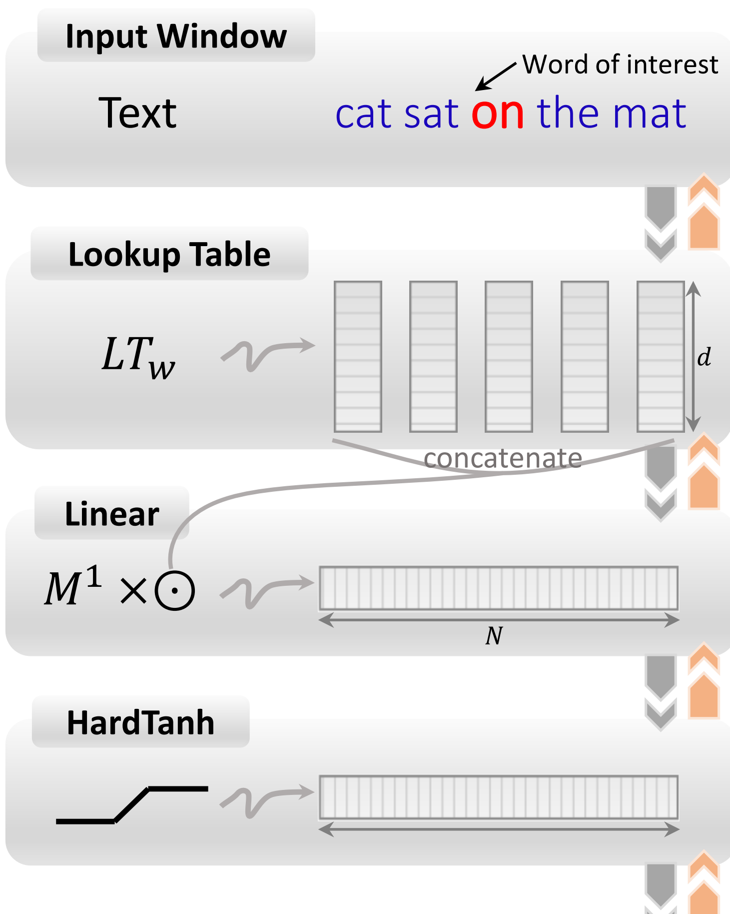
government debt problems turning into **banking** crises as has happened in

saying that Europe needs unified **banking** regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

You can vary whether you use local or large context to get a more syntactic or semantic clustering

JMLR11



$$htanh(x) = \begin{cases} 1 & \text{if } x \text{ is greater than 1} \\ -1 & \text{if } x \text{ is less than -1} \\ x & \text{otherwise} \end{cases}$$

- Ranking margin loss

$$\max\left(0, 1 - f(s, w_t^*) + f(s, w_t)\right)$$

s :sentence window
w_t^*:true middle word in s
w_t:a random word to replace w_t^*
$f(s, w)$:network score for s and w

- Stochastic training and back propagation are used to train the parameters including the word embedding

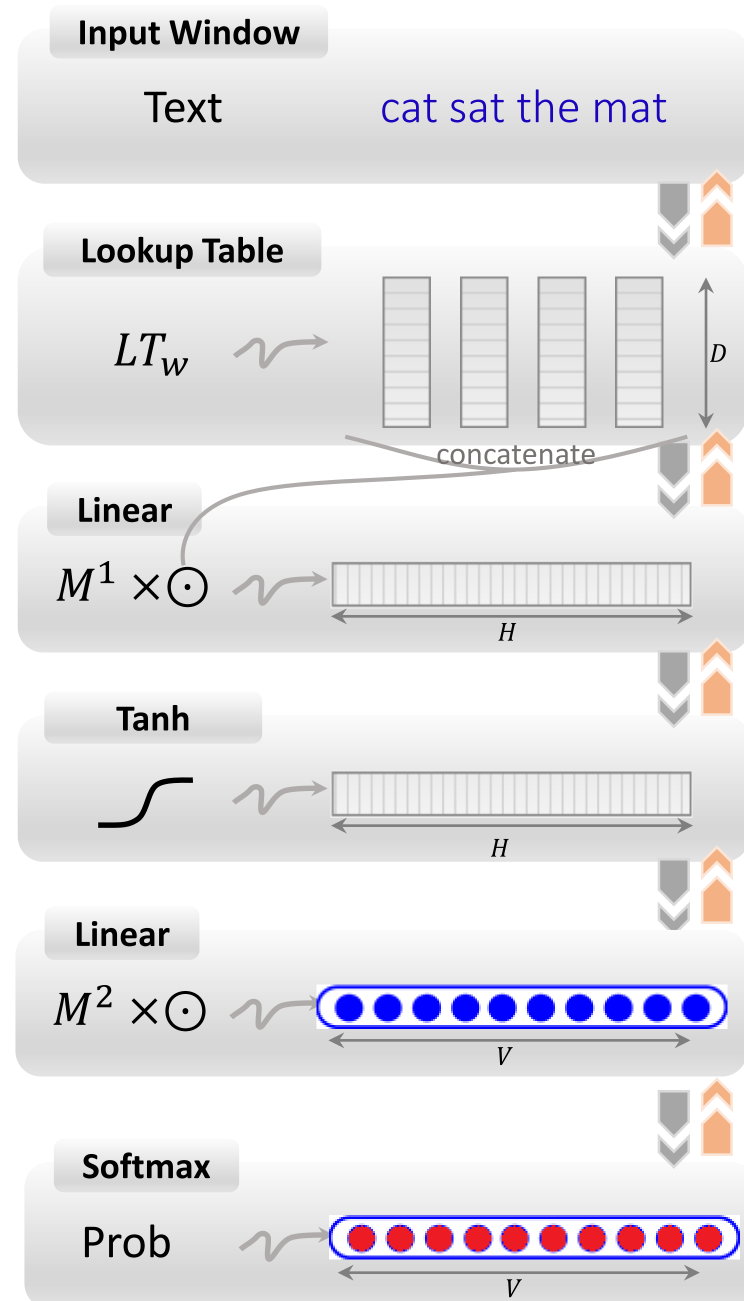
CW08:Feed Forward NNLM

- Computational Complexity

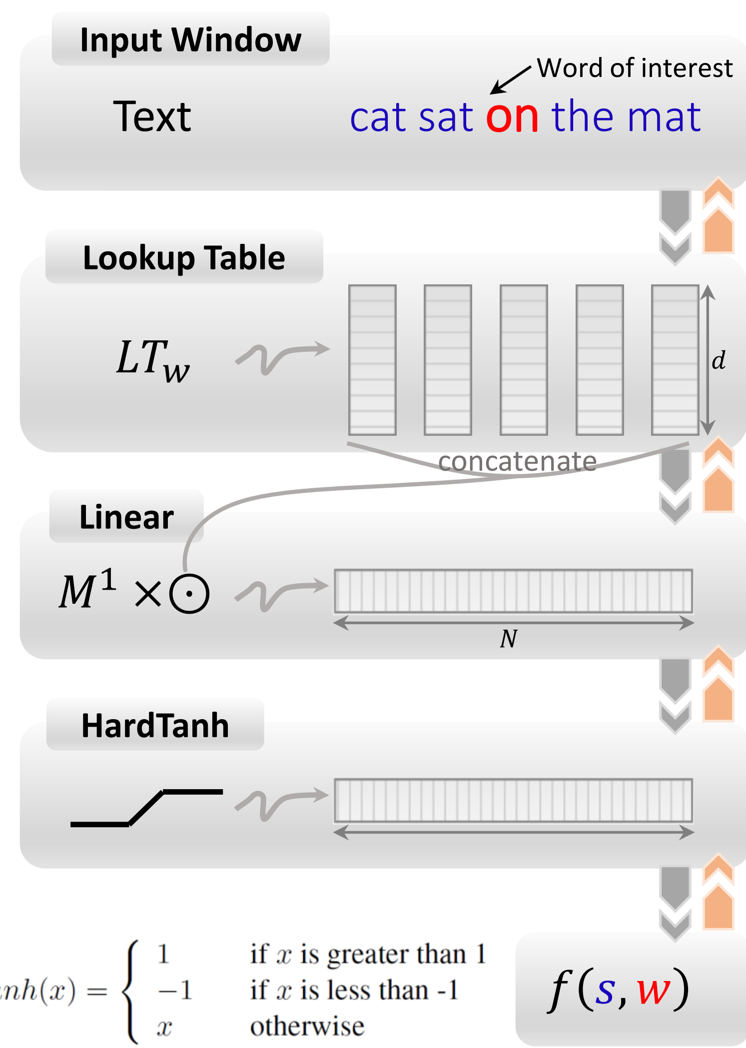
$$Q = N \times D + N \times D \times H + H \times V$$

- The output is a probability

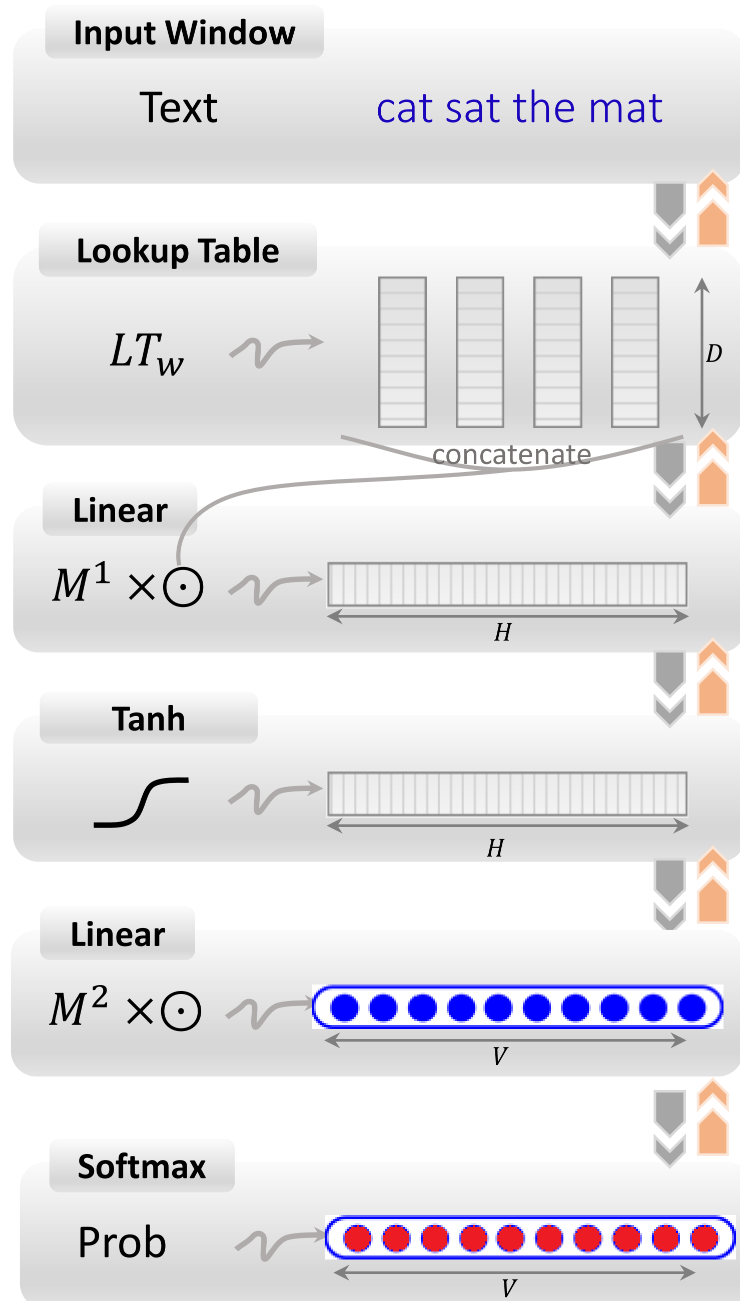
- Back propagation can also be used to update parameters



CW08 VS JMLR11



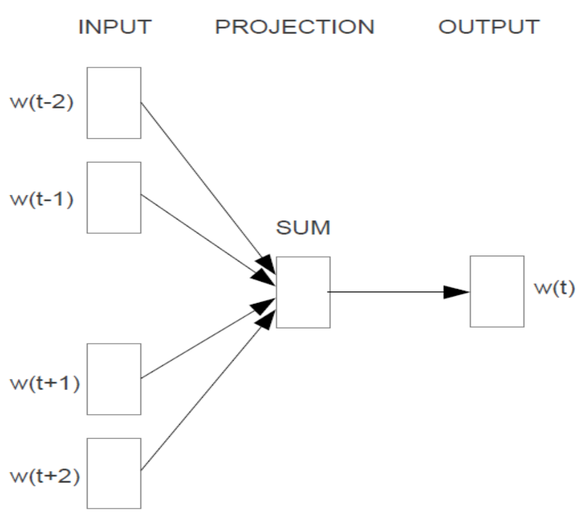
$$htanh(x) = \begin{cases} 1 & \text{if } x \text{ is greater than 1} \\ -1 & \text{if } x \text{ is less than -1} \\ x & \text{otherwise} \end{cases}$$



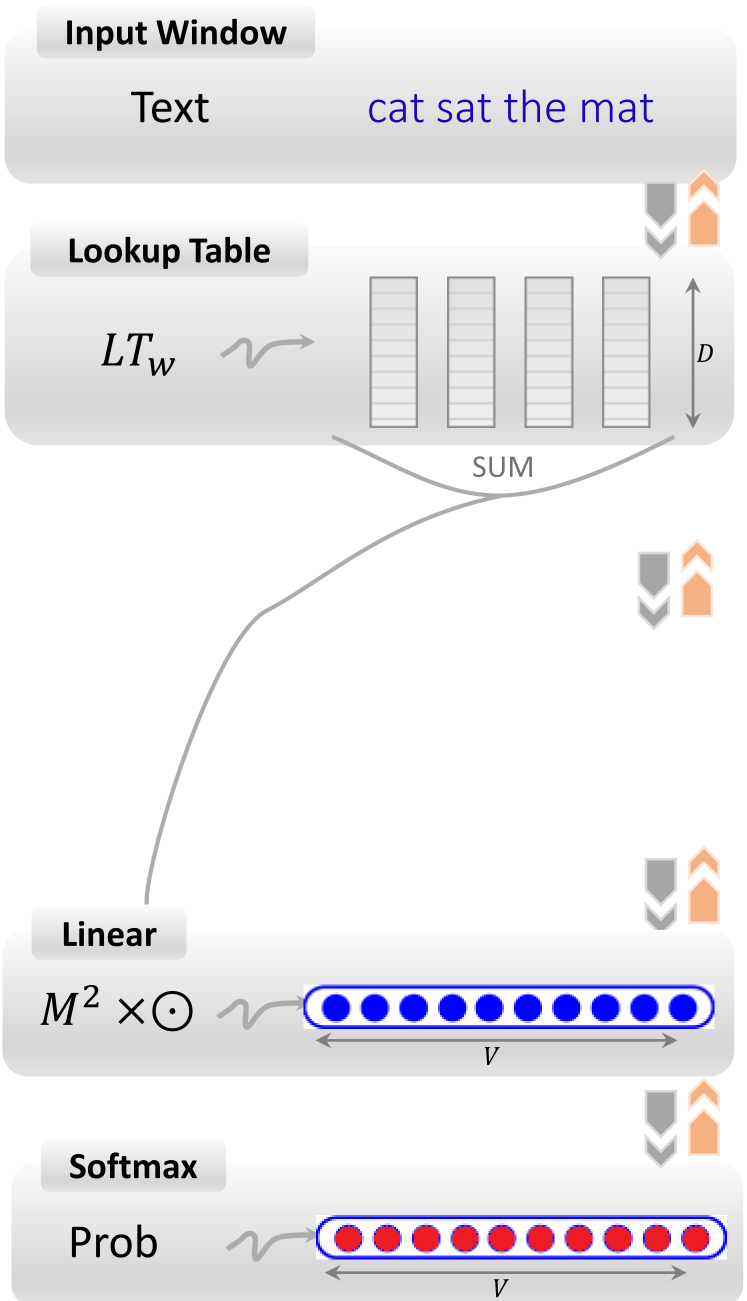
Google's Word2vec

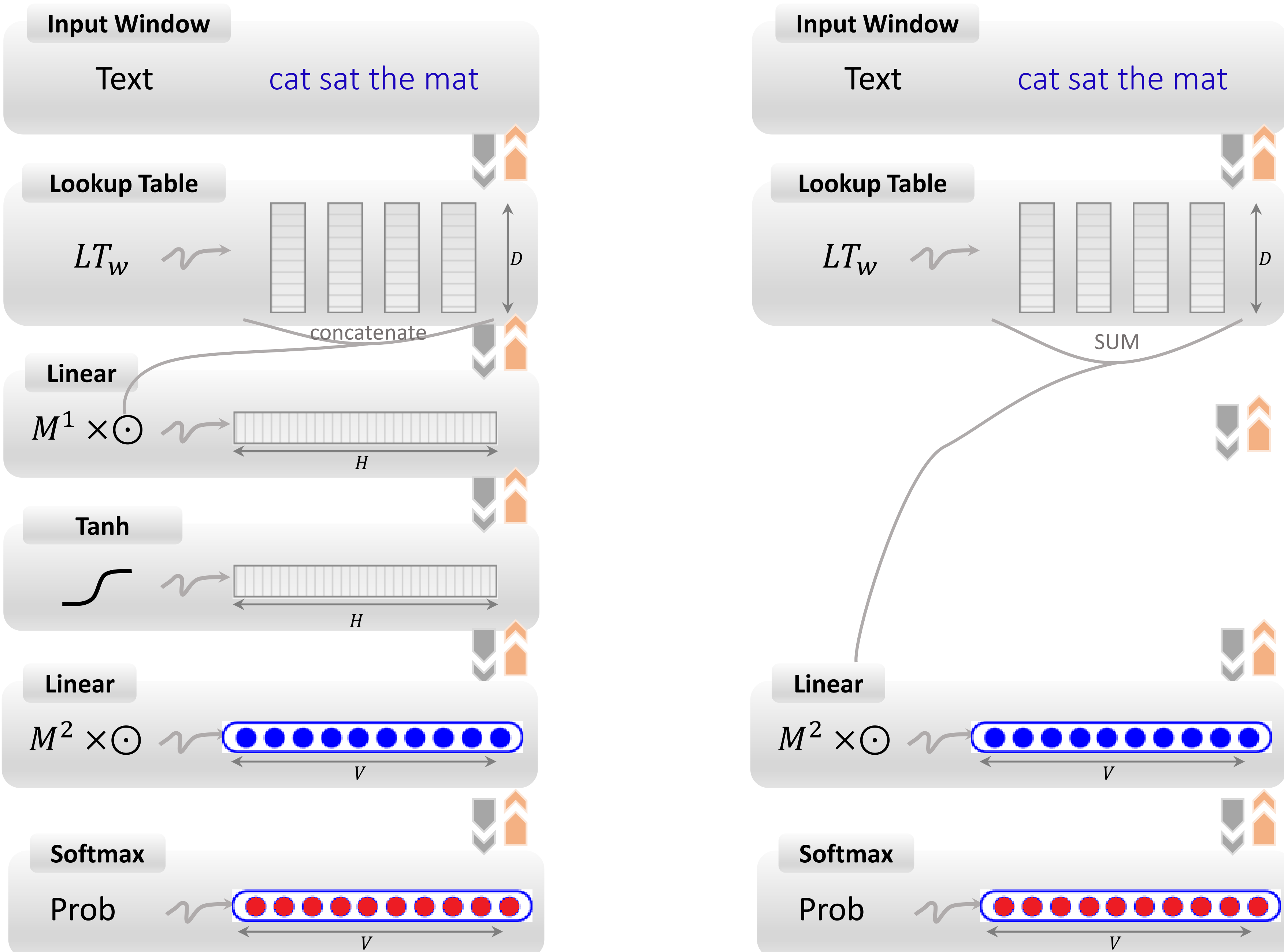
- CBOW: Continuous Bag-of-Word
  - The order of words in the history does not influence the projection
- Computational Complexity

$$\cancel{Q = N \times D + N \times D \times H + H \times V}$$
$$Q = N \times D + D \times V$$



CBOW

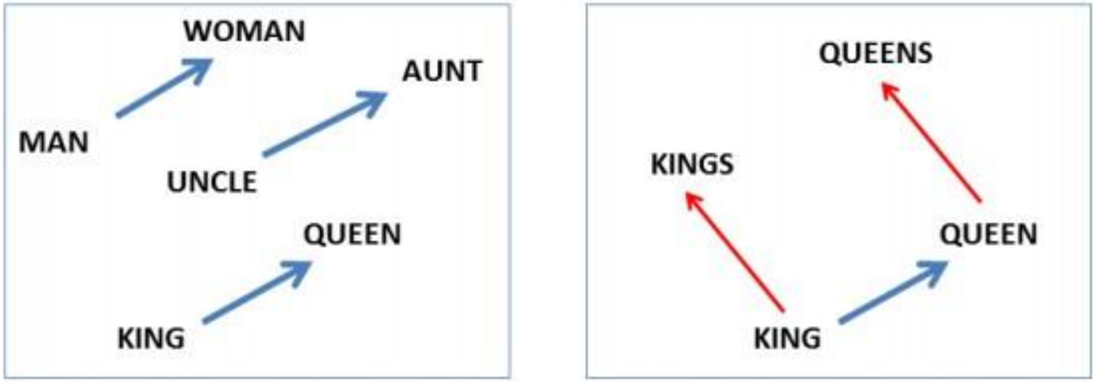




Input Window
Text
cat sat the mat
Lookup Table
LT_w
D
concatenate
Linear
$M^1 \times \odot$
H
Tanh
H
Linear
$M^2 \times \odot$
V
Softmax
Prob
V
Input Window
Text
cat sat the mat
Lookup Table
LT_w
D
SUM
Linear
$M^2 \times \odot$
V
Softmax
Prob
V

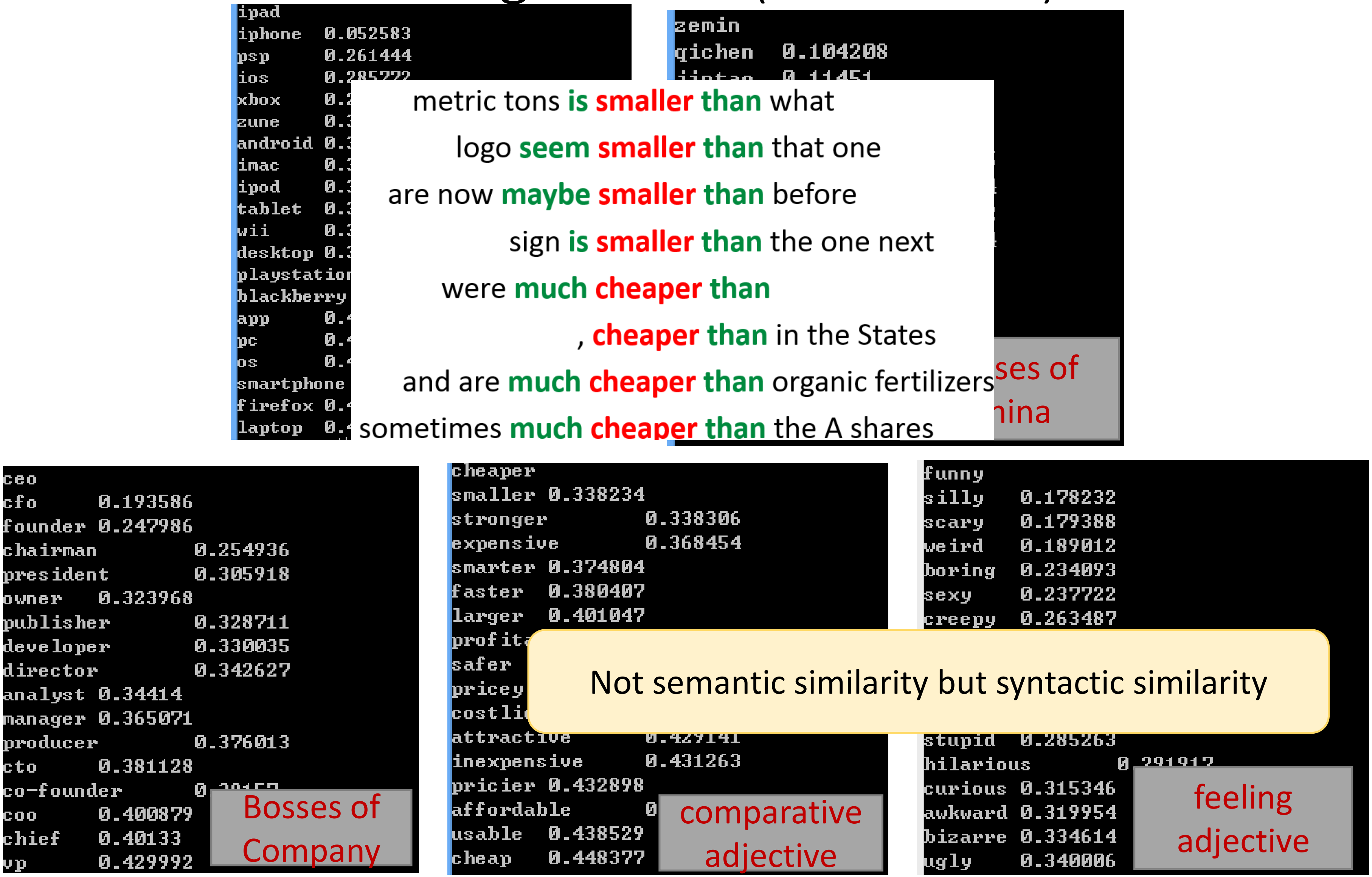
Linguistic Regularities in Continuous Space Word Representations (Mikolov, et al. 2013)

- Measuring Linguistic Regularity
  - Syntactic/Semetic Test



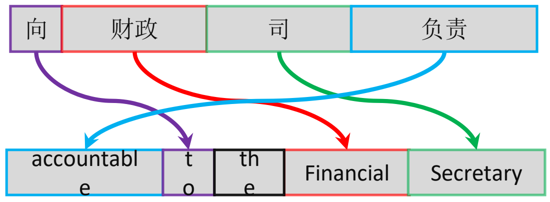
These representations are surprisingly good at capturing syntactic and semantic regularities in language, and that each relationship is characterized by a relation-specific vector offset.

Word Embedding Results (web 130G)



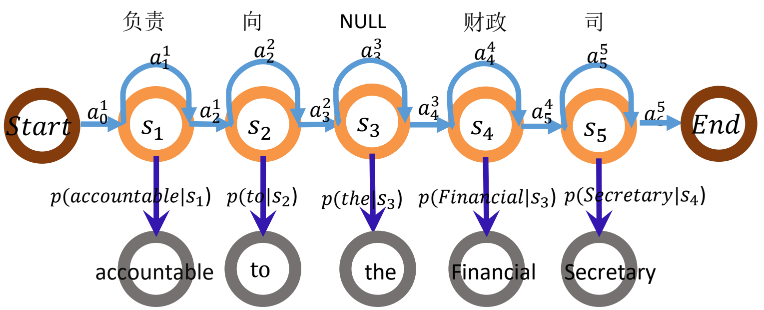
HMM for Word Alignment

- Alignment Problem
  - Finding translation pairs in bitext sentences



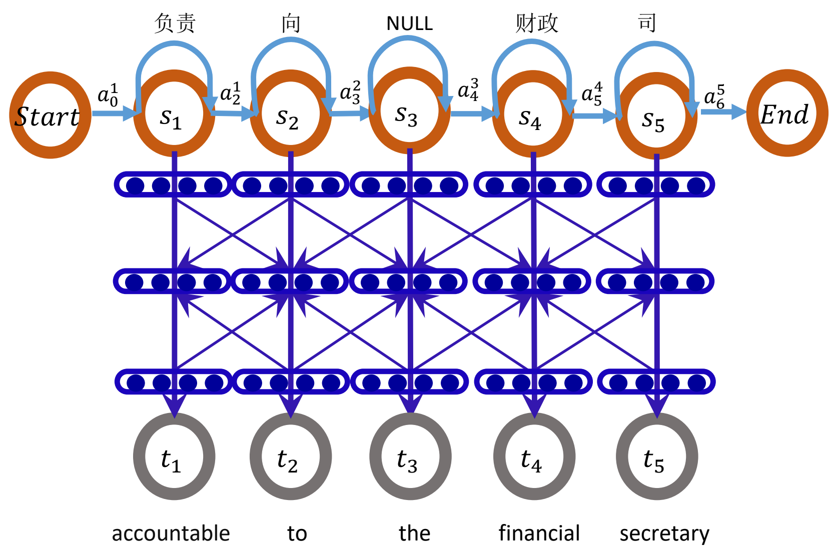
- HMM for Word Alignment
  - (Vogel et al., 1996)

$$p(a, e|f) = \prod_{i=1}^{|e|} p_{lex}(e_i|f_{a_i}) p_d(a_i - a_{i-1})$$



CD-DNN-HMM for Word Alignment

- Discover bilingual lexical similarity from mono and bi-text
- Improve language structure projections with rich context representation
- Cascaded training of neural latent variable models based on a initial alignment result

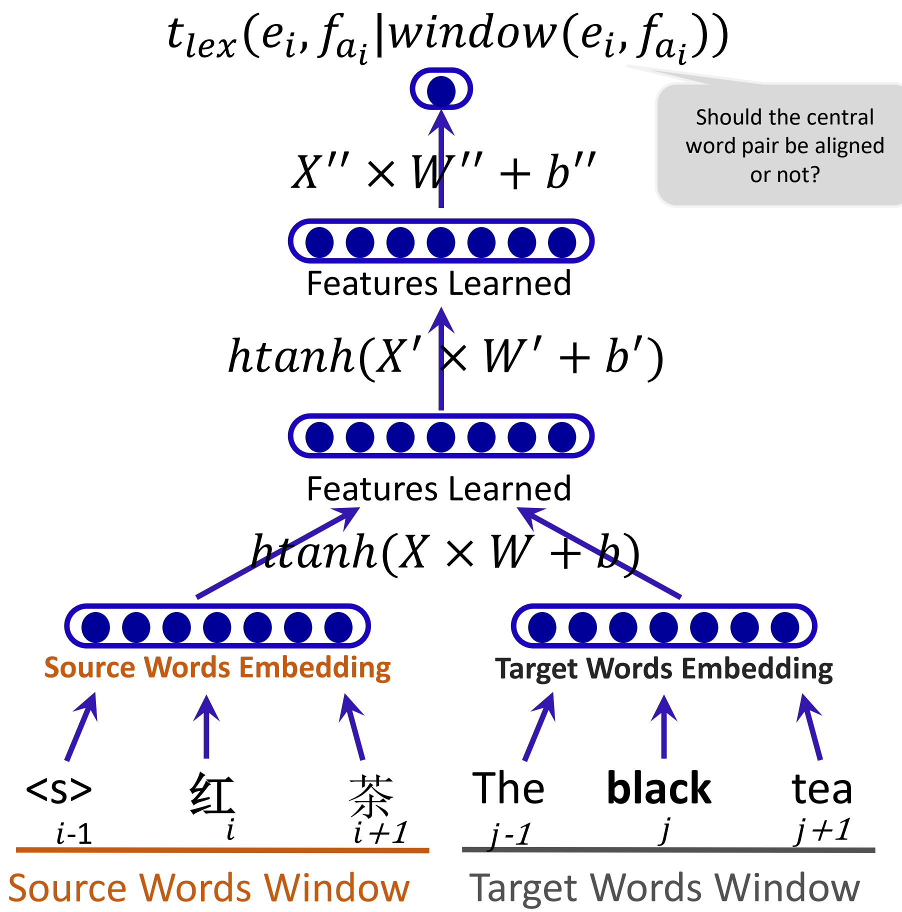


Model Observation Probability with DNN

- Source and target word embedding is initialized with word embedding trained with mono-lingual corpus

- Content information is used for training

- Embedding for both source and target words will be updated



Monolingual vs Bilingual Embedding

word	good	history	british	served	labs	zetian	laggards
LM	bad	tradition	russian	worked	networks	hongzhang	underperformers
	great	culture	japanese	lived	technologies	yaobang	transferees
	strong	practice	dutch	offered	innovations	keming	megabanks
	true	style	german	delivered	systems	xingzhi	mutuals
	easy	literature	canadian	produced	industries	ruihua	non-starters
WA	nice	historical	uk	offering	lab	hongzhang	underperformers
	great	historic	britain	serving	laboratories	qichao	illiterates
	best	developed	english	serve	laboratory	xueqin	transferees
	pretty	record	classic	delivering	exam	fuhuan	matriculants
	excellent	recording	england	worked	experiments	bingkun	megabanks

- “bad” is no longer in the nearest neighborhood of “good”, as they hold opposite semantic meaning
- Neighbors of proper nouns such as person names are relatively unchanged
- Rare words still remain their monolingual embeddings as they are modified a few times during bilingual training

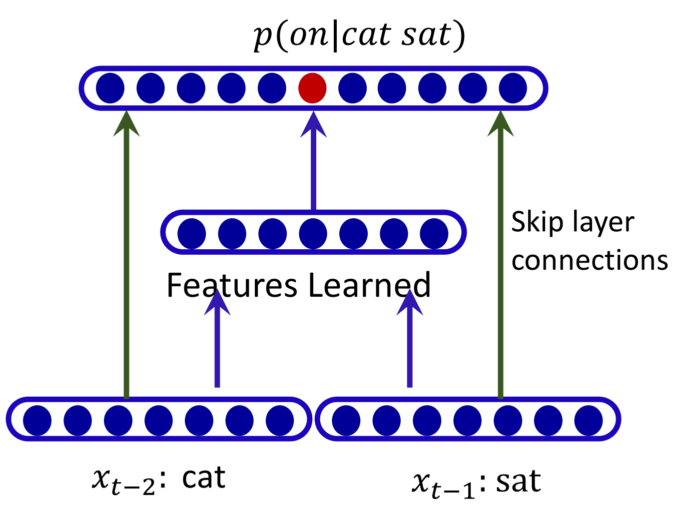
Outline

- Representation Learning First
- Introduction to DNN
- DNN for Natural Language Processing
  - DNN for Word Embedding
  - **DNN for Language Modeling**
  - DNN for Machine Translation

Language Model

- What is Language Model？
  - Language Model is a model to tell whether a sequence of words is a sentence or not.
  - Probabilistic Language Modeling： Score the probability of a sentence generated.
  - $P(w_1,w_2,\ldots,w_n) = p(w_1)p(w_2|w_1)\ldots p(w_n|w_1,\ldots,w_{n-1})$
  - Too much parameters， independence assumption
    - Bigram： $P(w_1,w_2,\ldots,w_n) = p(w_1)p(w_2|w_1)p(w_3|w_2)\ldots p(w_n|w_{n-1})$
    - Trigram： $P(w_1,w_2,\ldots,w_n) = p(w_1)p(w_2|w_1)p(w_3|w_2w_1)\ldots p(w_n|w_{n-1}w_{n-2})$
  - How to compute $p(w_2|w_1)$ and $p(w_3|w_2w_1)$?
    - Maximum Likelihood Estimation（MLE）: $p(w_i|w_{i-1}) = \frac{count(w_{i-1},w_i)}{\sum_{w^*} count(w_{i-1},w^*)}$
  - if $count(w_{i-1},w_i)$ =0？
    - Smoothing： add-1, goodem turing, Katz's back-off....

Feed Forward Language Model

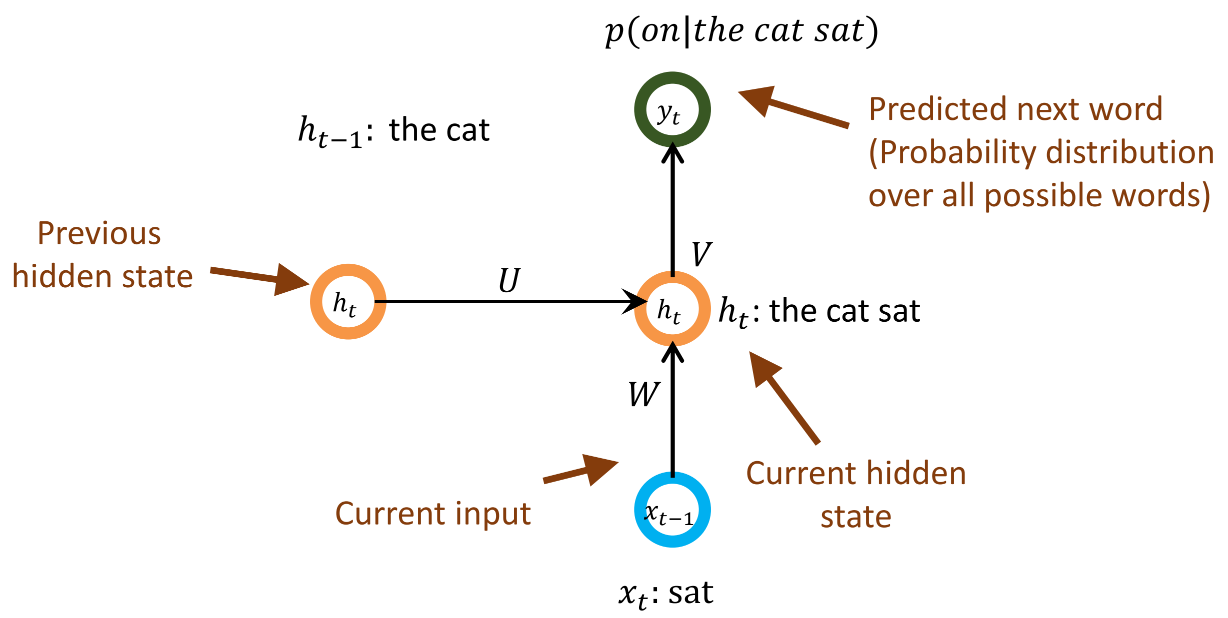


- Discriminative LM
- Assign score for each word given context words with NN
- Produce word embedding (fixed length, real valued vector) which can be used by other NLP tasks
- Softmax Layer to generate the probability:

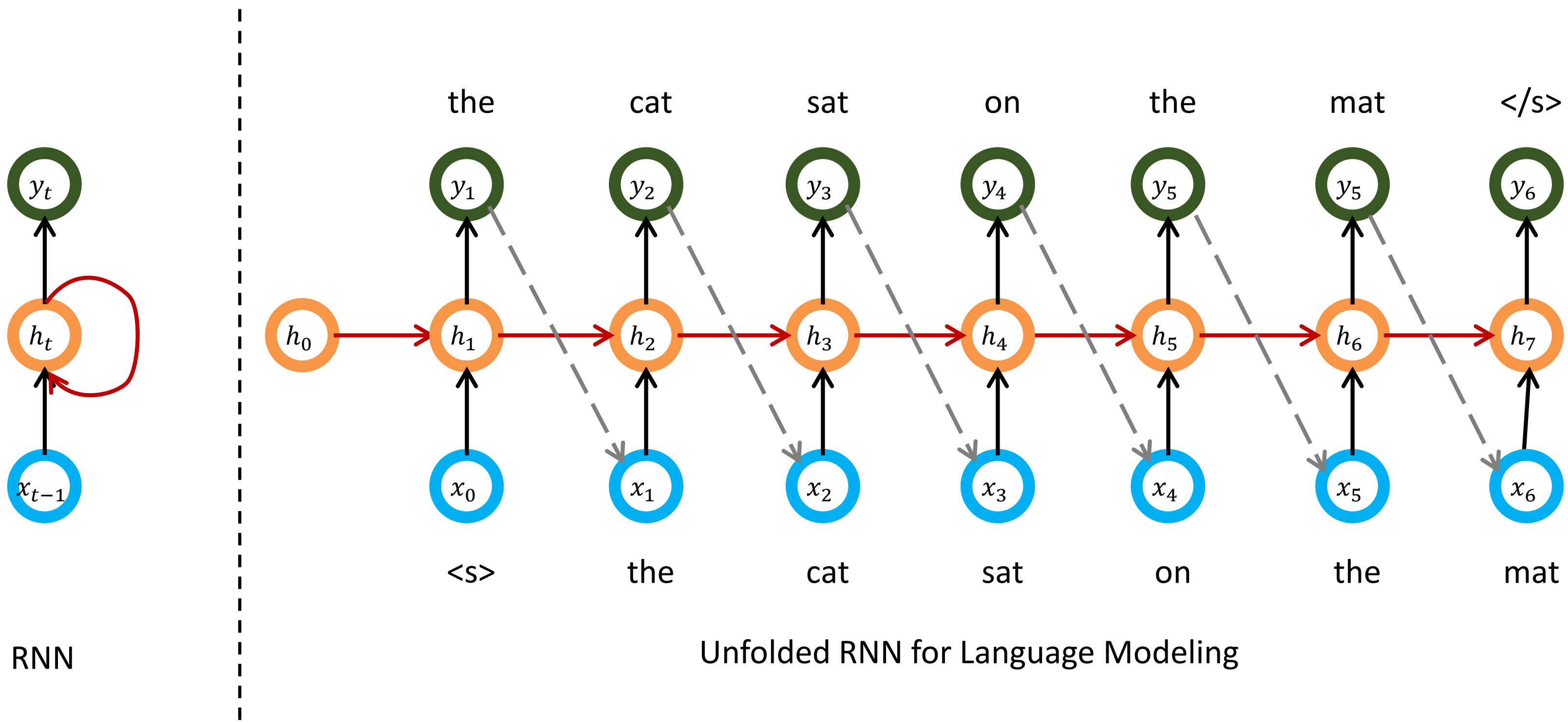
$$P\ (w_i) = \frac{e^{s_i}}{\sum_j e^{s_j}}$$

Recurrent Neural Network

- Inputs: History s_{t-1} at time $t-1$ and input w_t at time t
- Output: History s_t at time t and next input y_t at time t+1



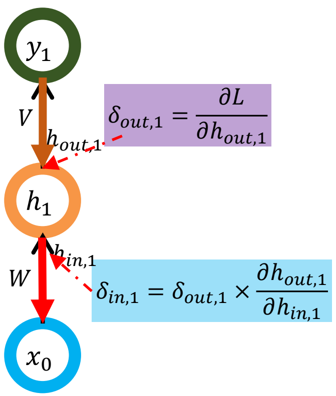
Unfolded RNN



Backward Propagation Through Time

t = 1

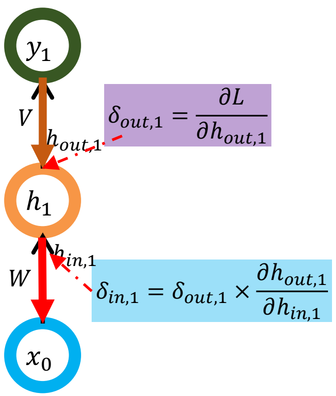
$$\delta_{in,1} = \boxed{\frac{\partial L}{\partial h_{out,1}}} \times \boxed{\frac{\partial h_{out,1}}{\partial h_{in,1}}}$$



Backward Propagation Through Time

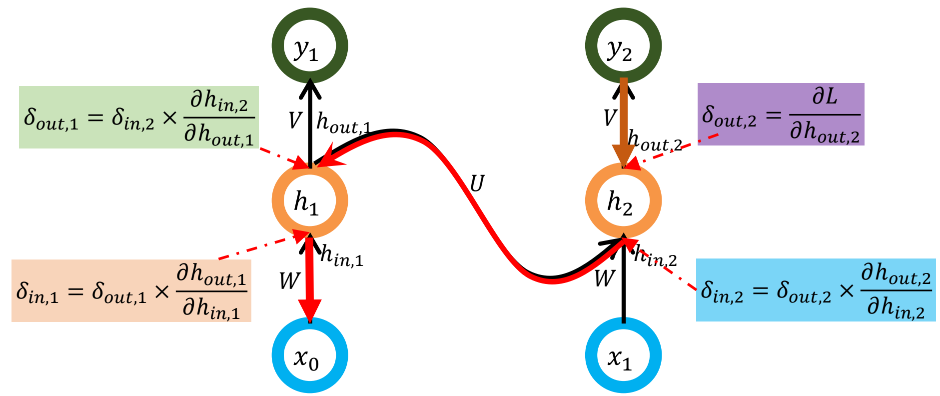
t = 1

$$\delta_{in,1} = \frac{\partial L}{\partial h_{out,1}} \times \frac{\partial h_{out,1}}{\partial h_{in,1}}$$



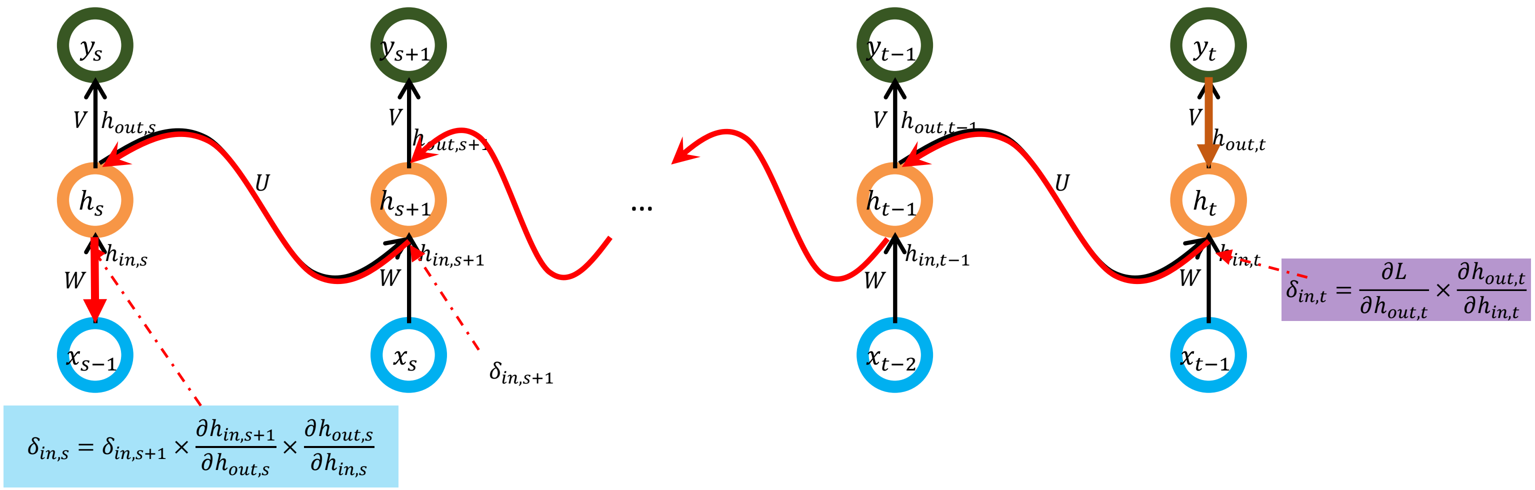
t = 2

$$\delta_{in,1} = \frac{\partial L}{\partial h_{out,2}} \times \frac{\partial h_{out,2}}{\partial h_{in,2}} \times \frac{\partial h_{in,2}}{\partial h_{out,1}} \times \frac{\partial h_{out,1}}{\partial h_{in,1}}$$
$$= \delta_{out,2} \times \frac{\partial h_{out,2}}{\partial h_{in,2}} \times \frac{\partial h_{in,2}}{\partial h_{out,1}} \times \frac{\partial h_{out,1}}{\partial h_{in,1}}$$
$$= \delta_{in,2} \times \frac{\partial h_{in,2}}{\partial h_{out,1}} \times \frac{\partial h_{out,1}}{\partial h_{in,1}} = \delta_{out,1} \times \frac{\partial h_{out,1}}{\partial h_{in,1}}$$

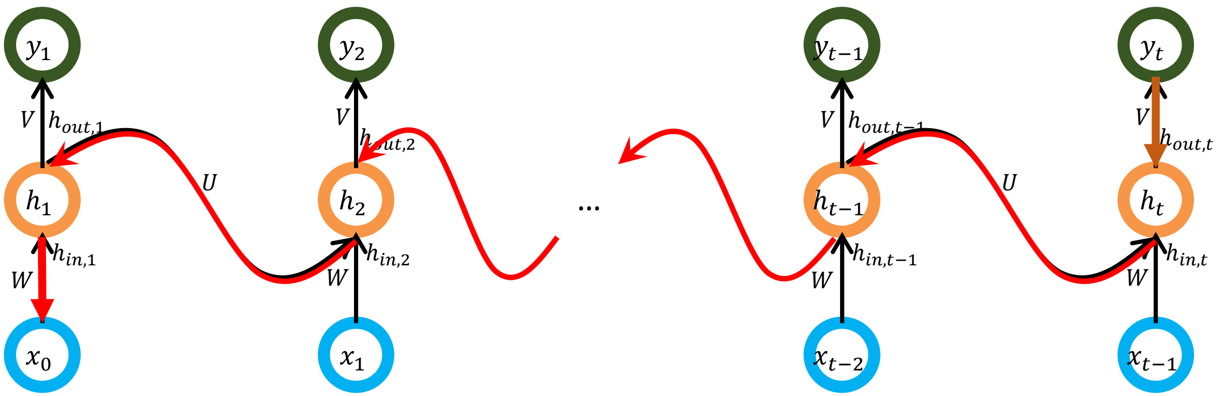


Backward Propagation Through Time

$$\delta_{in,s} = \begin{cases} \dfrac{\partial L}{\partial h_{out,s}} \times \dfrac{\partial h_{out,s}}{\partial h_{in,s}} & if\ s = t \\ \delta_{in,s+1} \times \dfrac{\partial h_{in,s+1}}{\partial h_{out,s}} \times \dfrac{\partial h_{out,s}}{\partial h_{in,s}} & otherwise \end{cases}$$

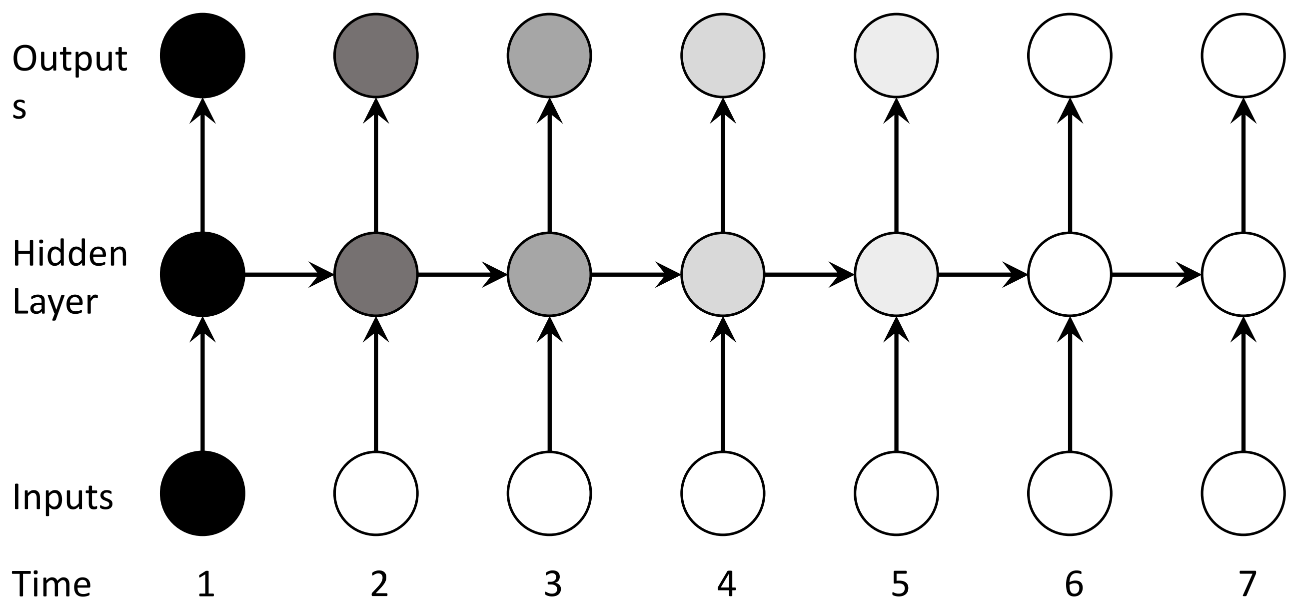


Vanishing Gradient Problem

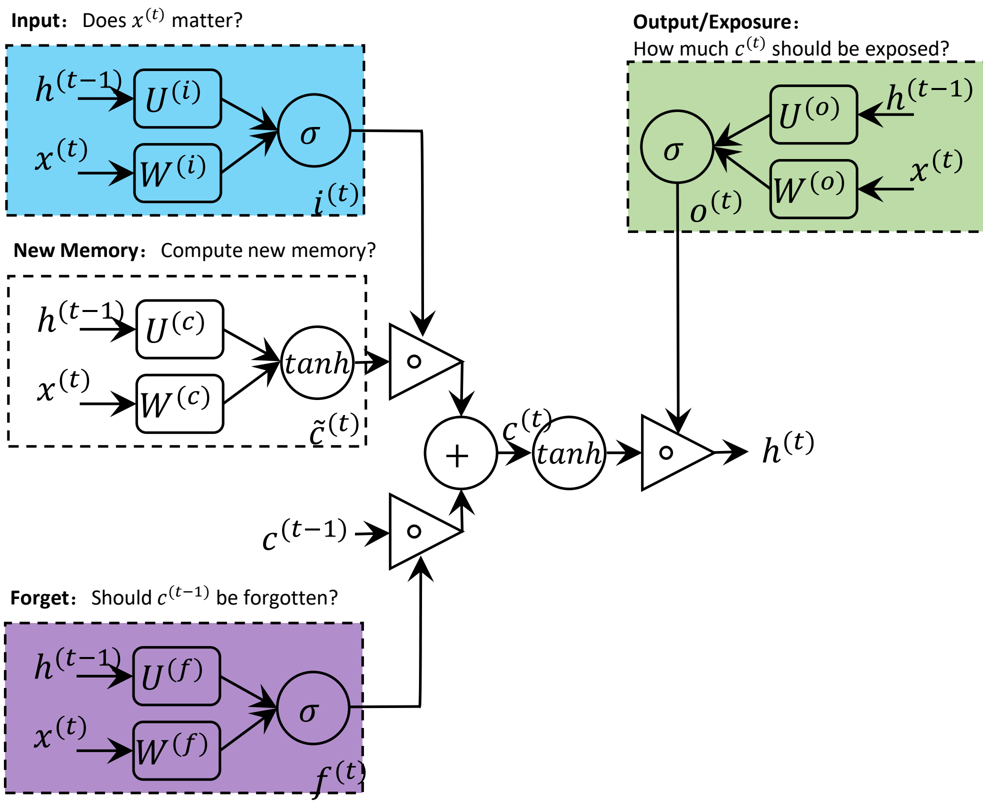


$$\delta_{in,1} = \delta_{out,t} \times \frac{\partial h_{out,t}}{\partial h_{in,t}} \times \frac{\partial h_{in,t}}{\partial h_{out,t-1}} \times \cdots \times \frac{\partial h_{in,2}}{\partial h_{out,1}} \times \frac{\partial h_{out,1}}{\partial h_{in,1}}$$

Vanishing Gradient Problem



LSTM: Long Short Term Memory



$$i^{(t)} = \sigma(W^{(i)}x^{(t)} + U^{(i)}h^{(t-1)})$$

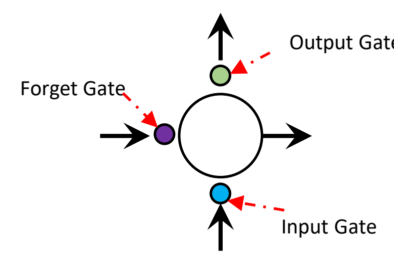
$$f^{(t)} = \sigma(W^{(f)}x^{(t)} + U^{(f)}h^{(t-1)})$$

$$o^{(t)} = \sigma(W^{(o)}x^{(t)} + U^{(o)}h^{(t-1)})$$

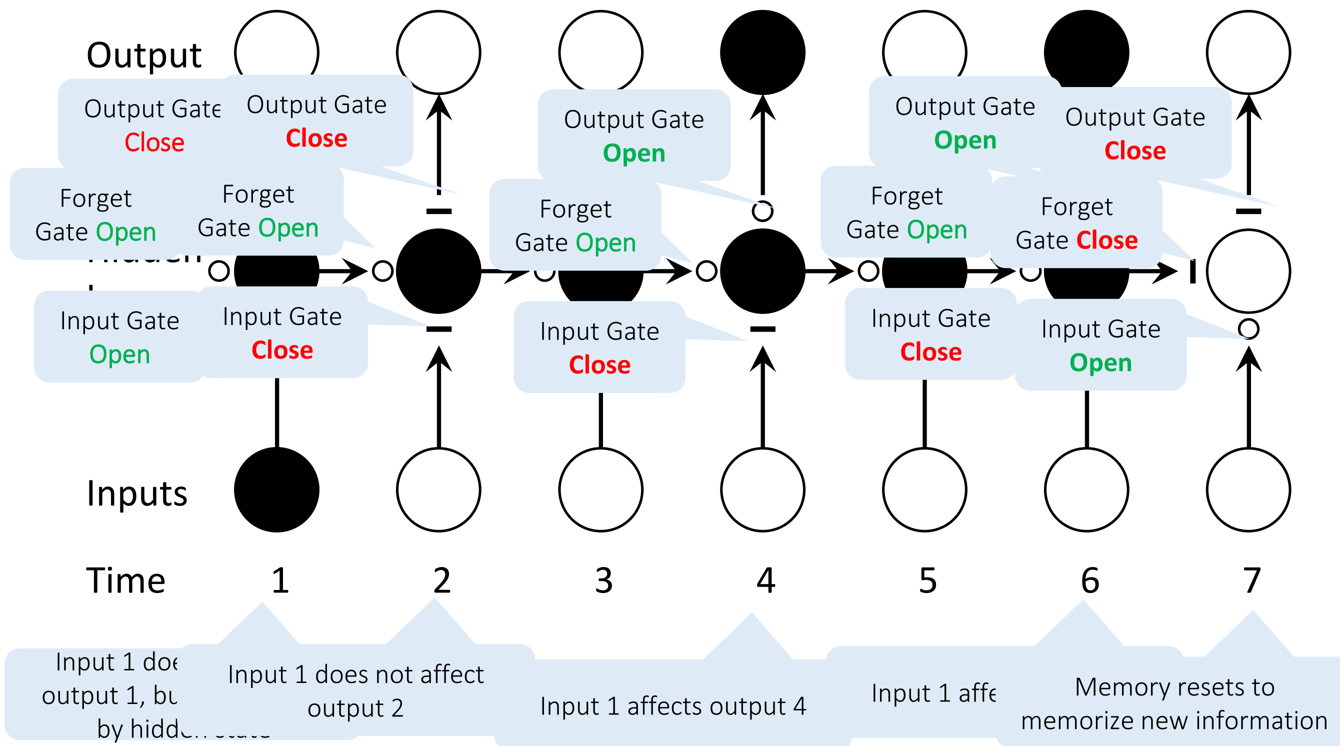
$$\tilde{c}^{(t)} = tanh(W^{(c)}x^{(t)} + U^{(c)}h^{(t-1)})$$

$$c^{(t)} = f^{(t)} \circ \tilde{c}^{(t-1)} + i^{(t)} \circ \tilde{c}^{(t)}$$

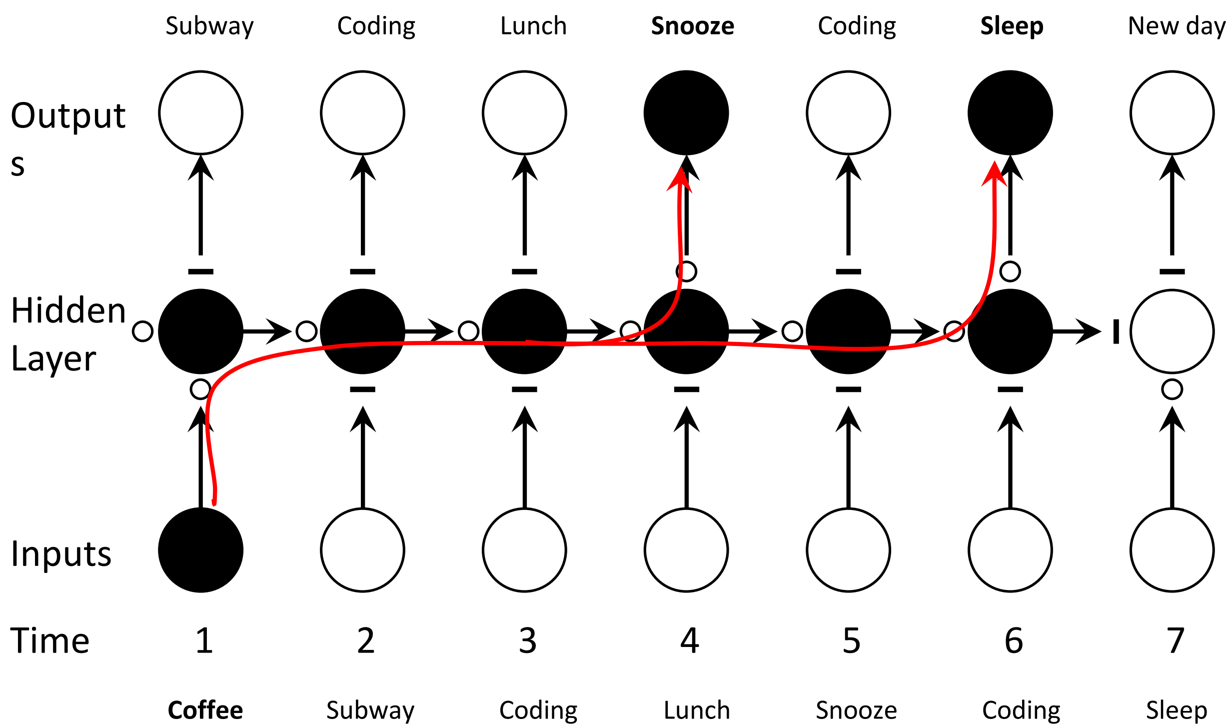
$$h^{(t)} = o^{(t)} \circ tanh(c^{(t)})$$



LSTM: Long Short Term Memory



LSTM: An Example



Feed Forward vs. LSTM LMs

- **Task:** Train FFNN and LSTM LMs on same data
- **Result:** LSTM outperforms FFNN, even if FFNN uses long context

n-gram Order	Num Hidden Layers	Perplexity
5	1	65.8
5	3	58.9
7	3	55.2
10	3	52.8
15	3	51.9
20	3	51.6
LSTM	1	45.1
LSTM	2	41.8

English LM, 100M words, 10k output vocab

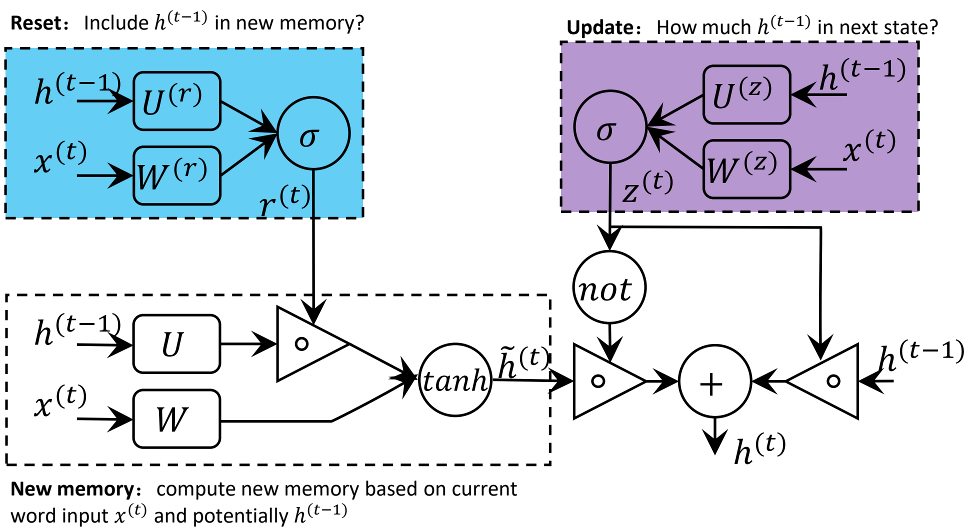
Feed Forward vs. LSTM LMs

- **Qualitative analysis**: LSTM is much better at “parsing” the input

Segment	20-gram FF Log Prob	Recurrent Log Prob
the lawsuit , filed wednesday on behalf of linda and robert lott of birmingham , **alleges**	-8.9	-2.7
the lawsuit **alleges**	-2.9	-2.8
some journalists said the claim that instant news was more incendiary than reports delivered more slowly **was**	-9.3	-1.5
some journalists said the claim **was**	-1.8	-0.8

Word being predicted is in **bold**

GRU: Gated Recurrent Unit

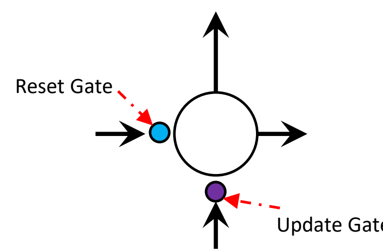


$$z^{(t)} = \sigma(W^{(z)}x^{(t)} + U^{(z)}h^{(t-1)})$$

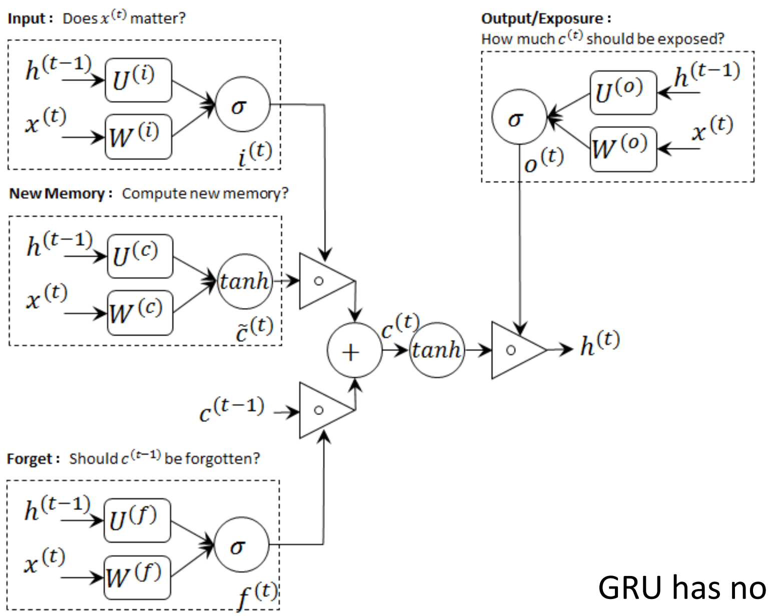
$$r^{(t)} = \sigma(W^{(r)}x^{(t)} + U^{(r)}h^{(t-1)})$$

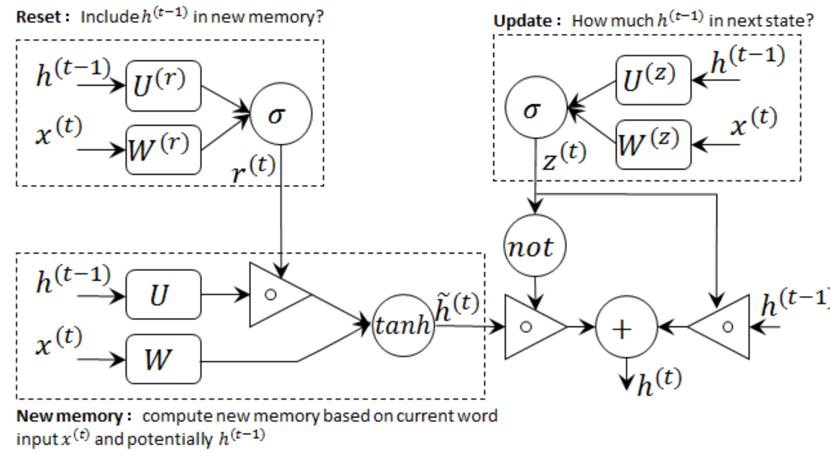
$$\tilde{h}^{(t)} = tanh(r^{(t)} \circ U h^{(t-1)} + W x^{(t)})$$

$$h^{(t)} = \left(1 - z^{(t)}\right) \circ \tilde{h}^{(t)} + z^{(t)} \circ h^{(t-1)}$$



LSTM vs GRU



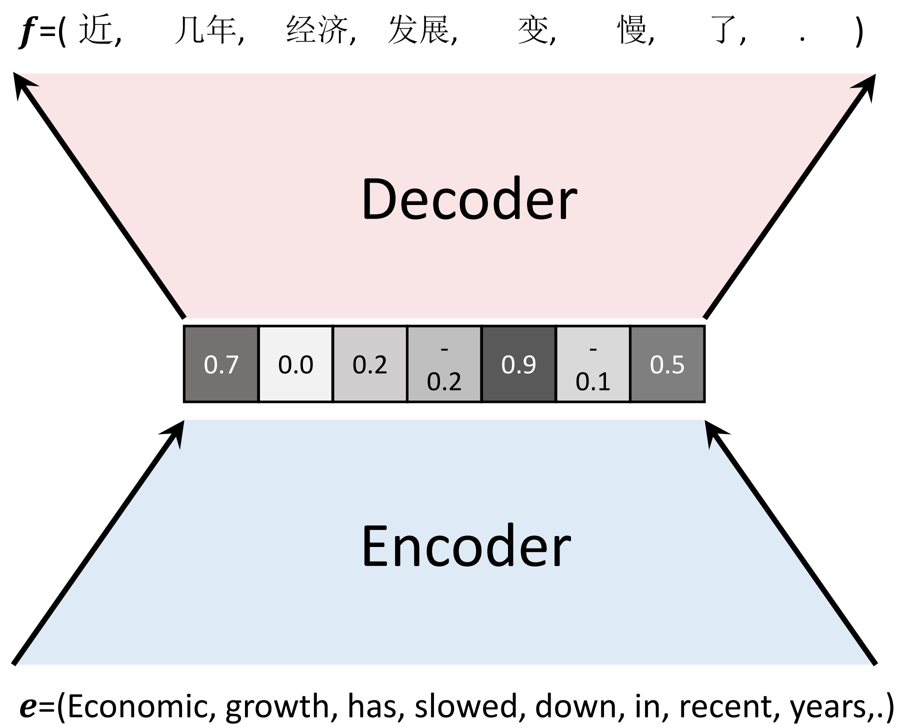


GRU has no output gate. All the information in memory/hidden state will affect the output. The only way to make information inactive is to reset/forget it.

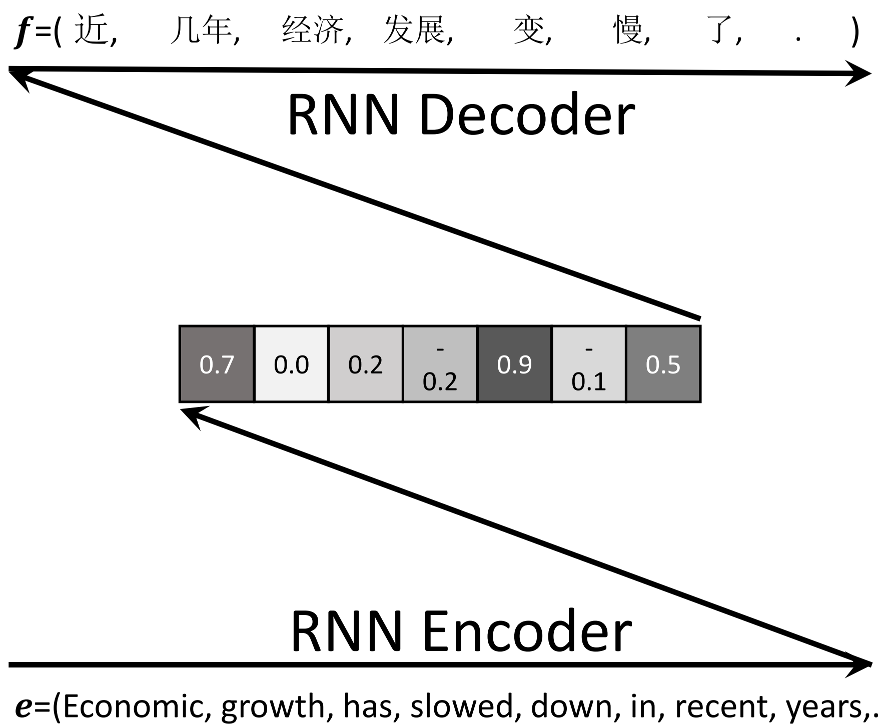
Outline

- Representation Learning First
- Introduction to DNN
- DNN for Natural Language Processing
  - DNN for Word Embedding
  - DNN for Language Modeling
  - **DNN for Machine Translation**

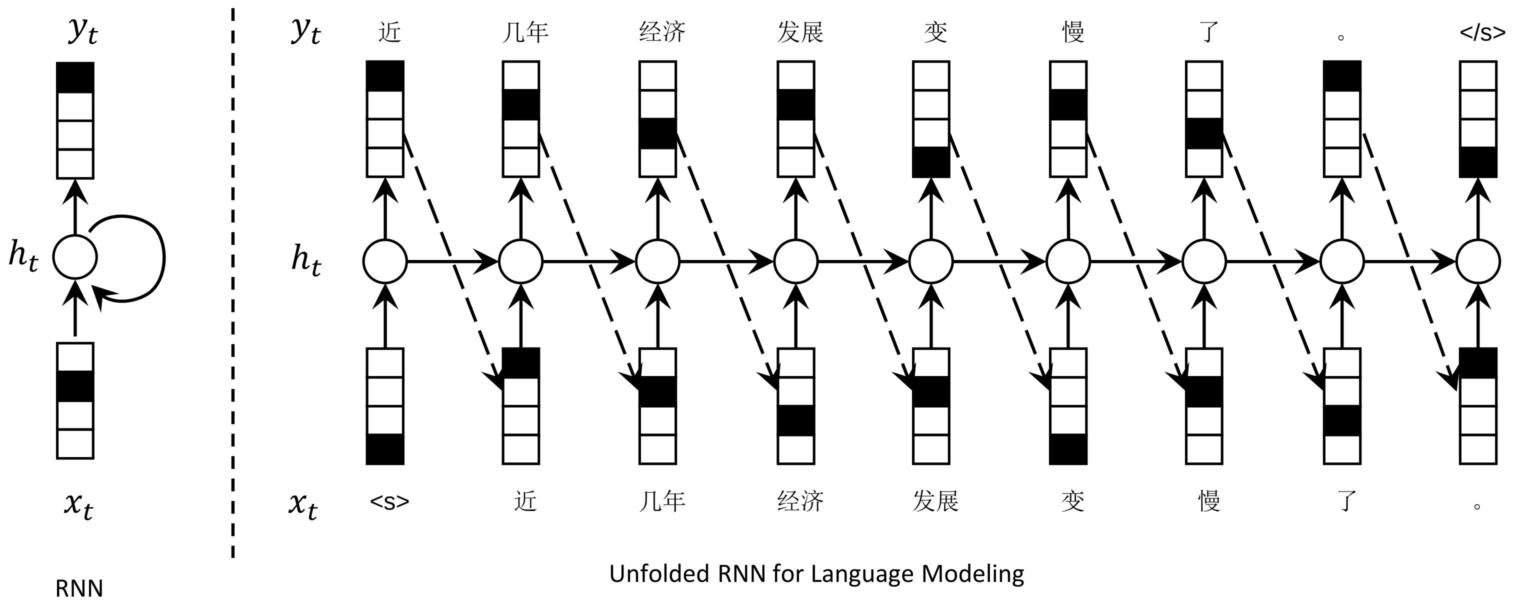
Encoder-Decoder for NMT

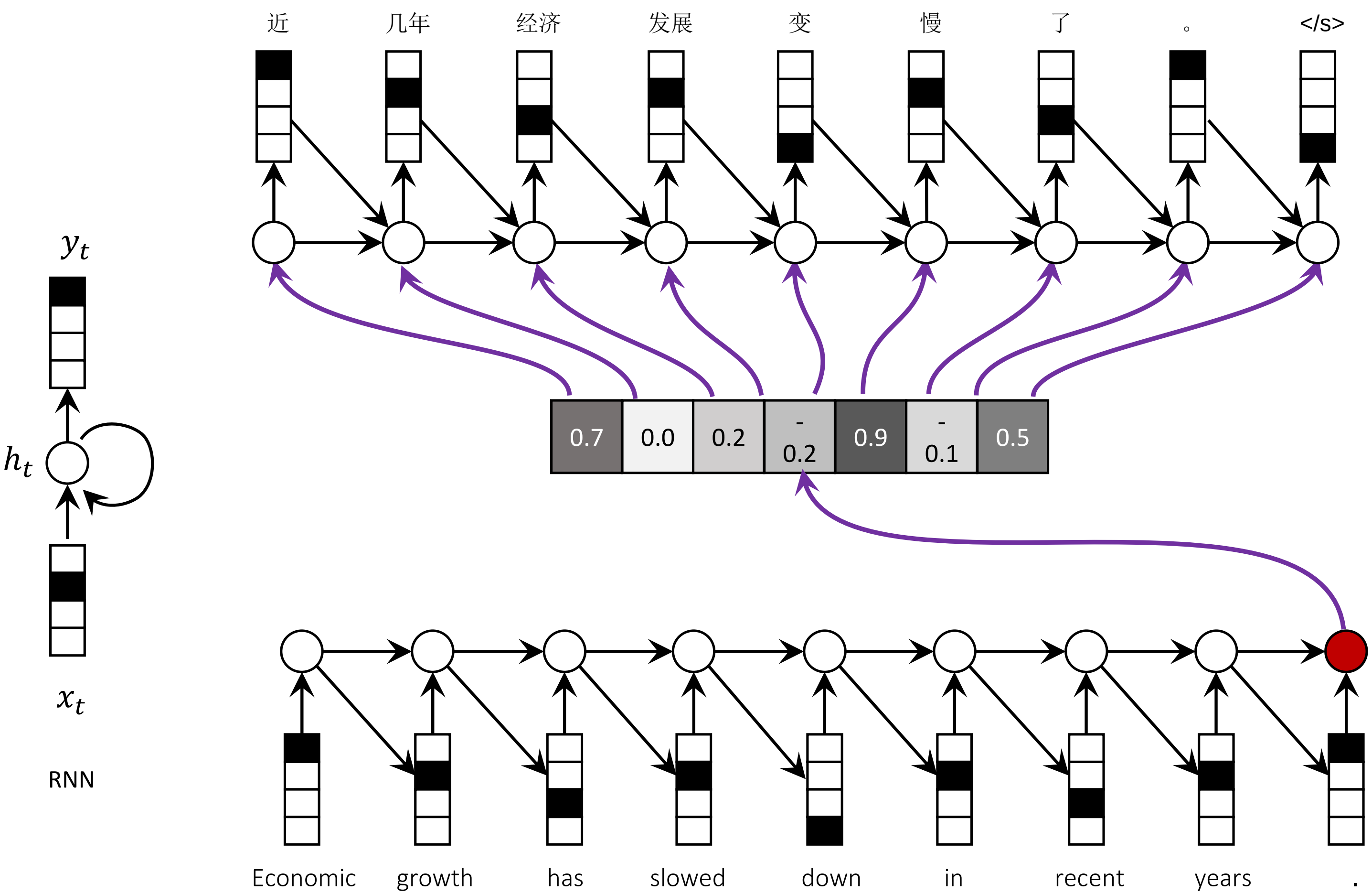


RNN-based Encoder-Decoder for NMT

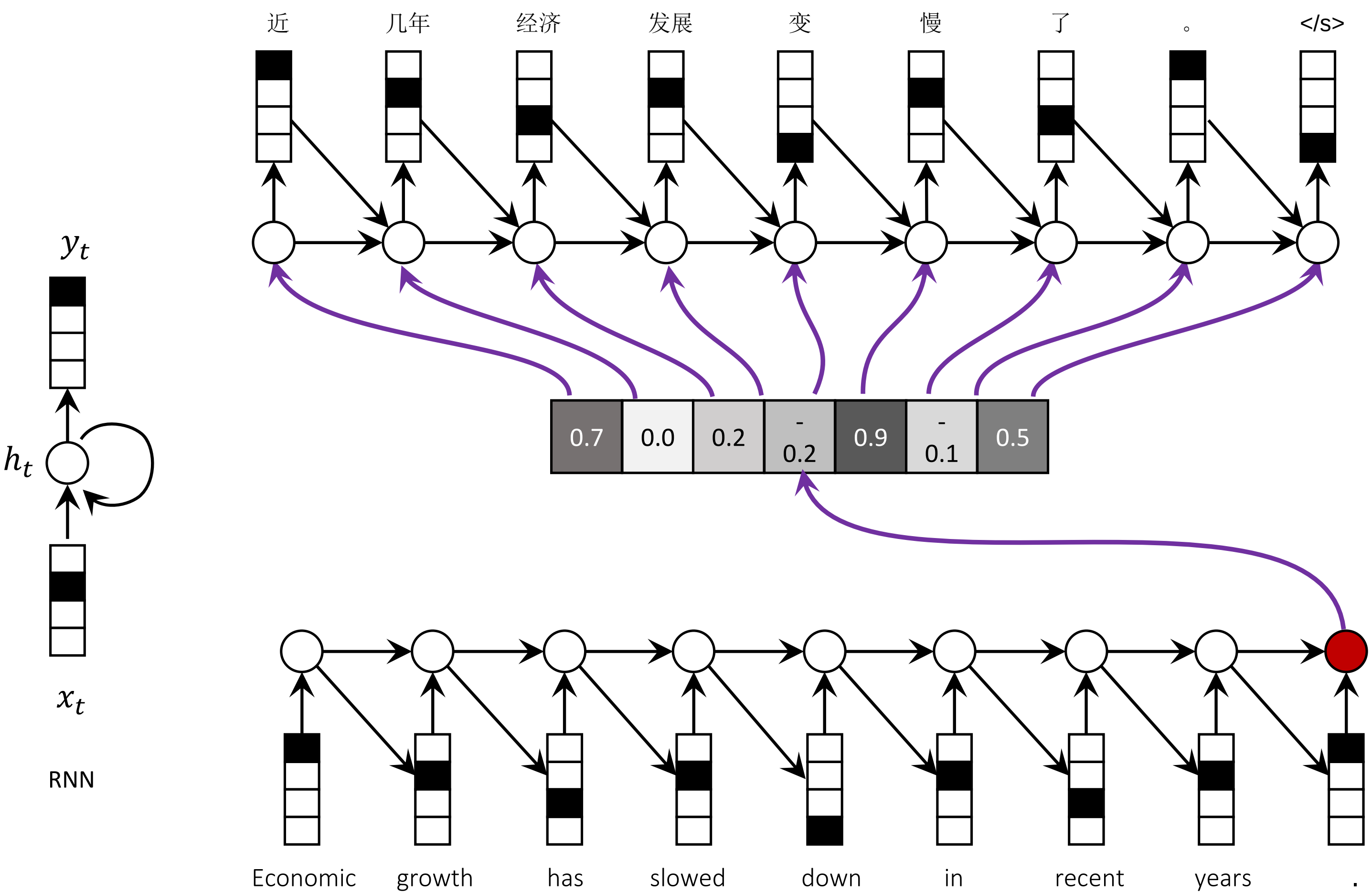


RNN: Recurrent Neural Network



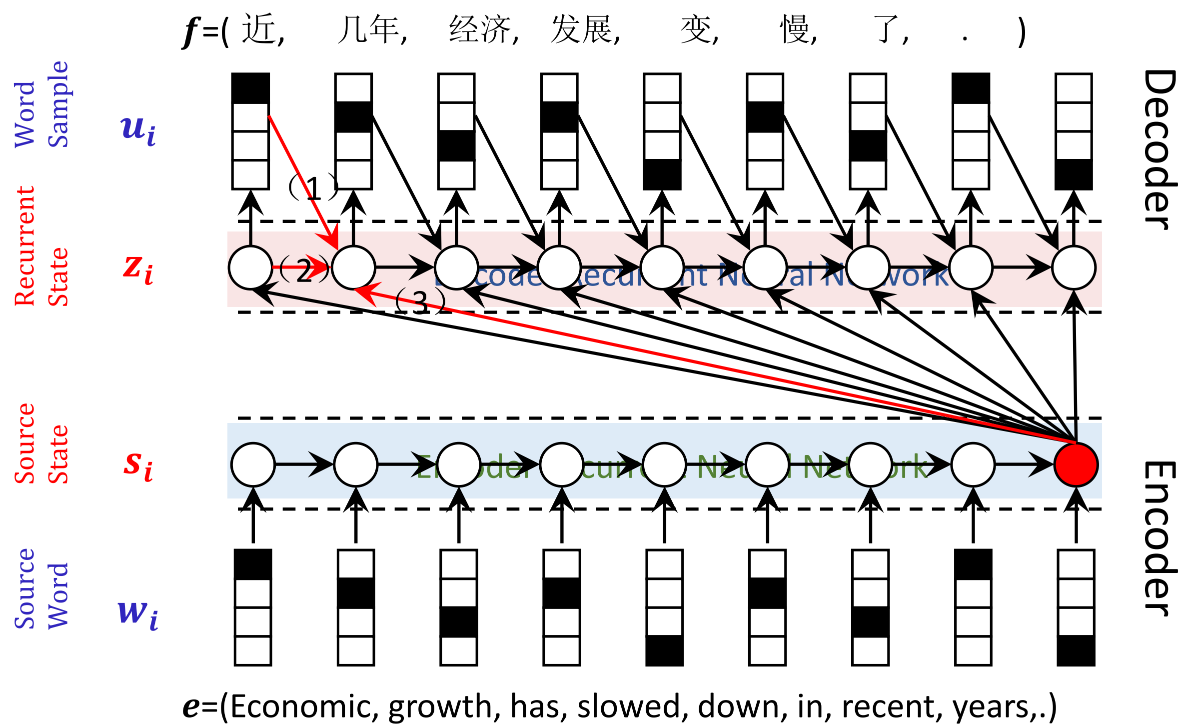


近
几年
经济
发展
变
慢
了
。
</s>
y_t
h_t
x_t
RNN
0.7
0.0
0.2
-0.2
0.9
-0.1
0.5
Economic
growth
has
slowed
down
in
recent
years
.



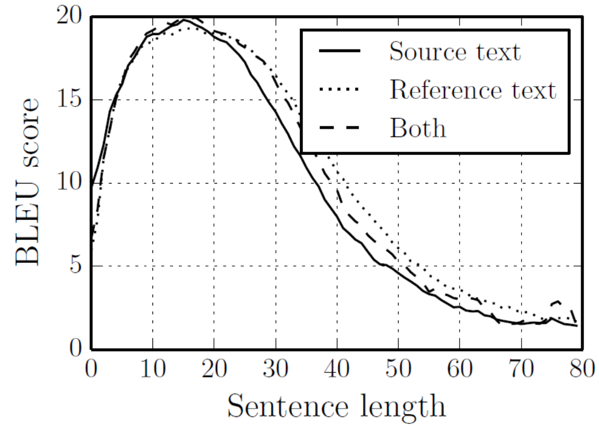
近
几年
经济
发展
变
慢
了
。
</s>
y_t
h_t
x_t
RNN
0.7
0.0
0.2
-0.2
0.9
-0.1
0.5
Economic
growth
has
slowed
down
in
recent
years
.

Encoder-Decoder for NMT

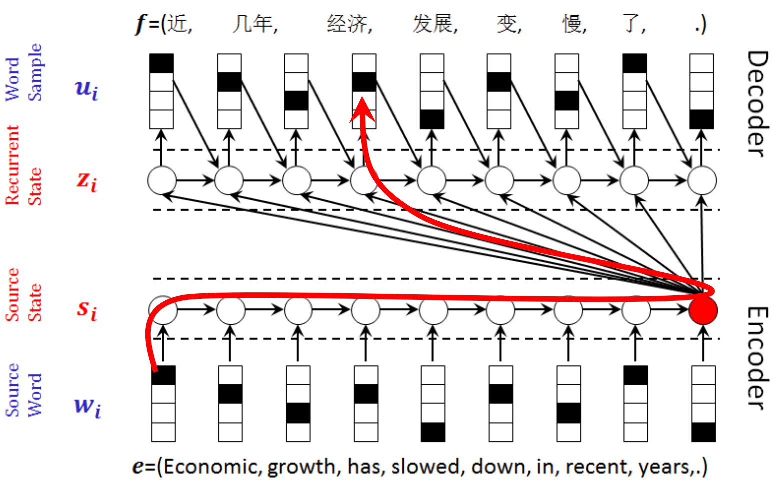


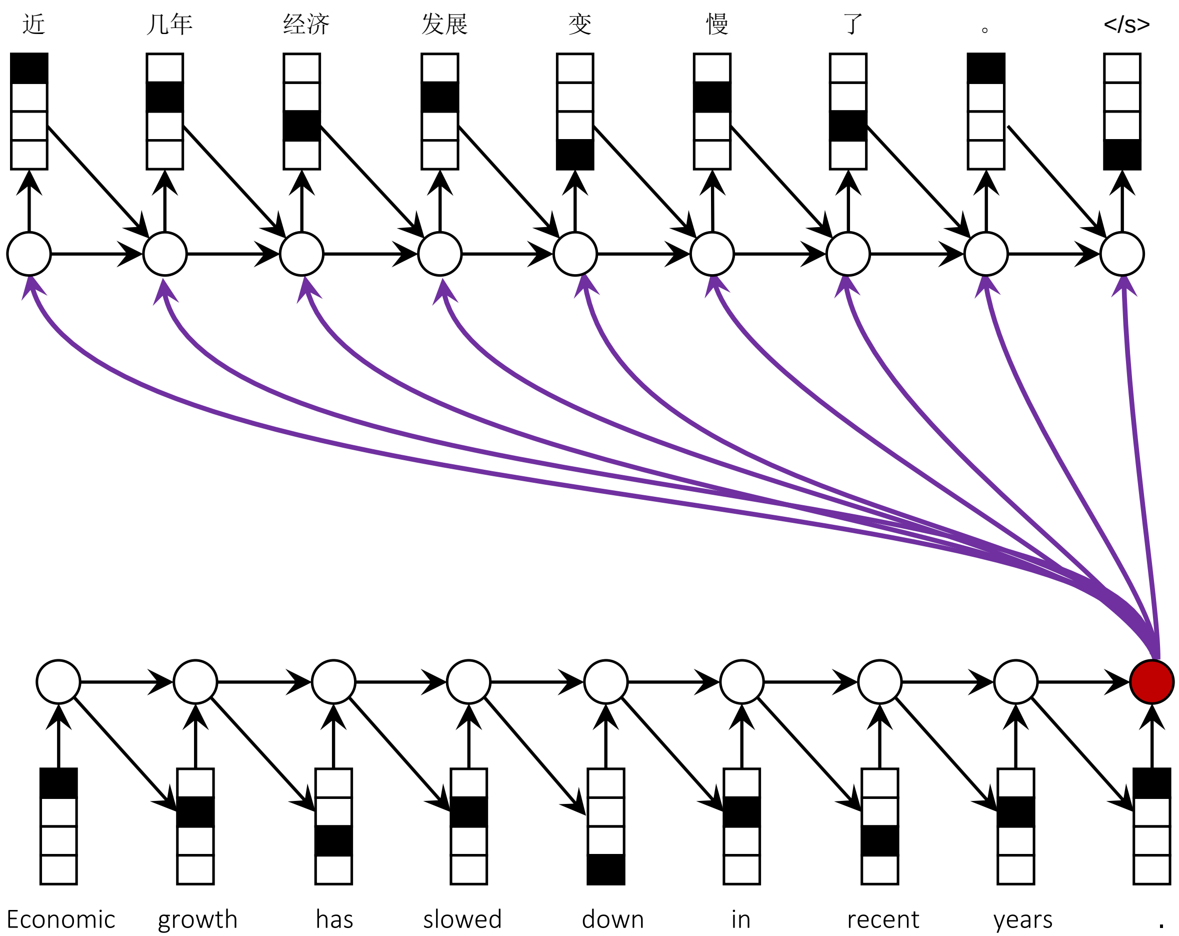
Sutskever, NIPS 2014

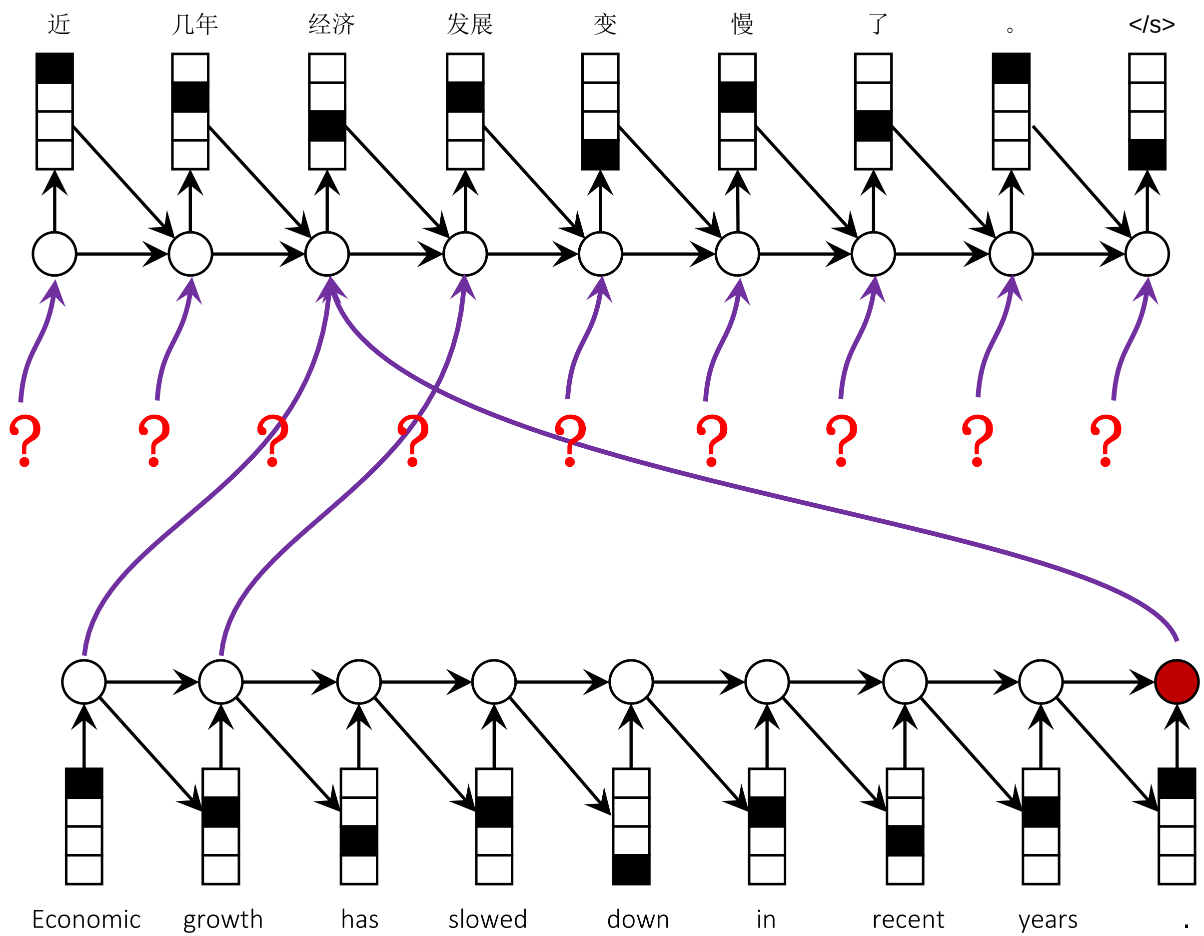
Motivation of Attention

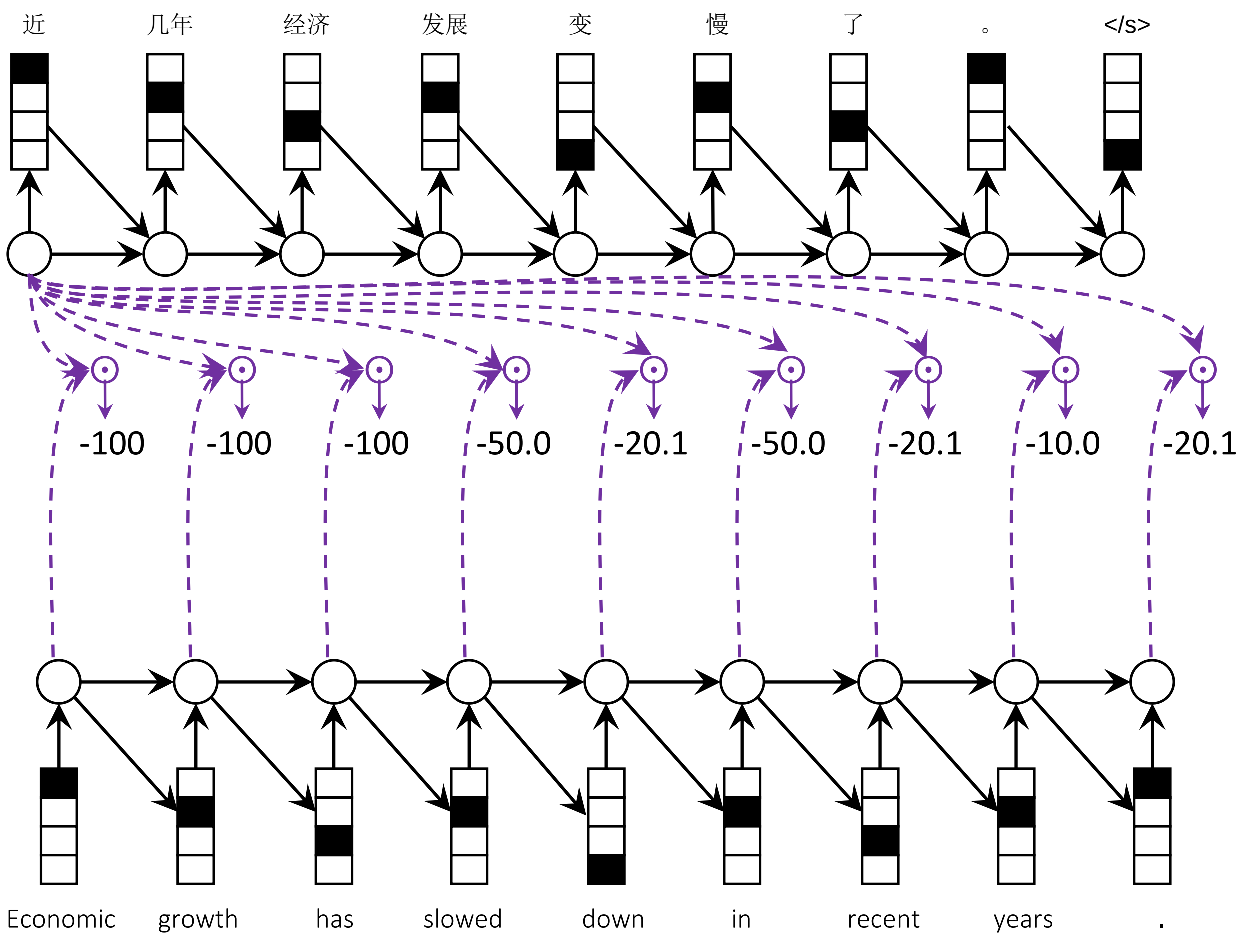


- NMT performs bad for long sentences
  - A long way from source to target
  - Only last hidden vector for decoding
  - Fixed-length hidden state is not enough
- Solution
  - Connect the source and target directly
  - Use all the hidden states for decoding

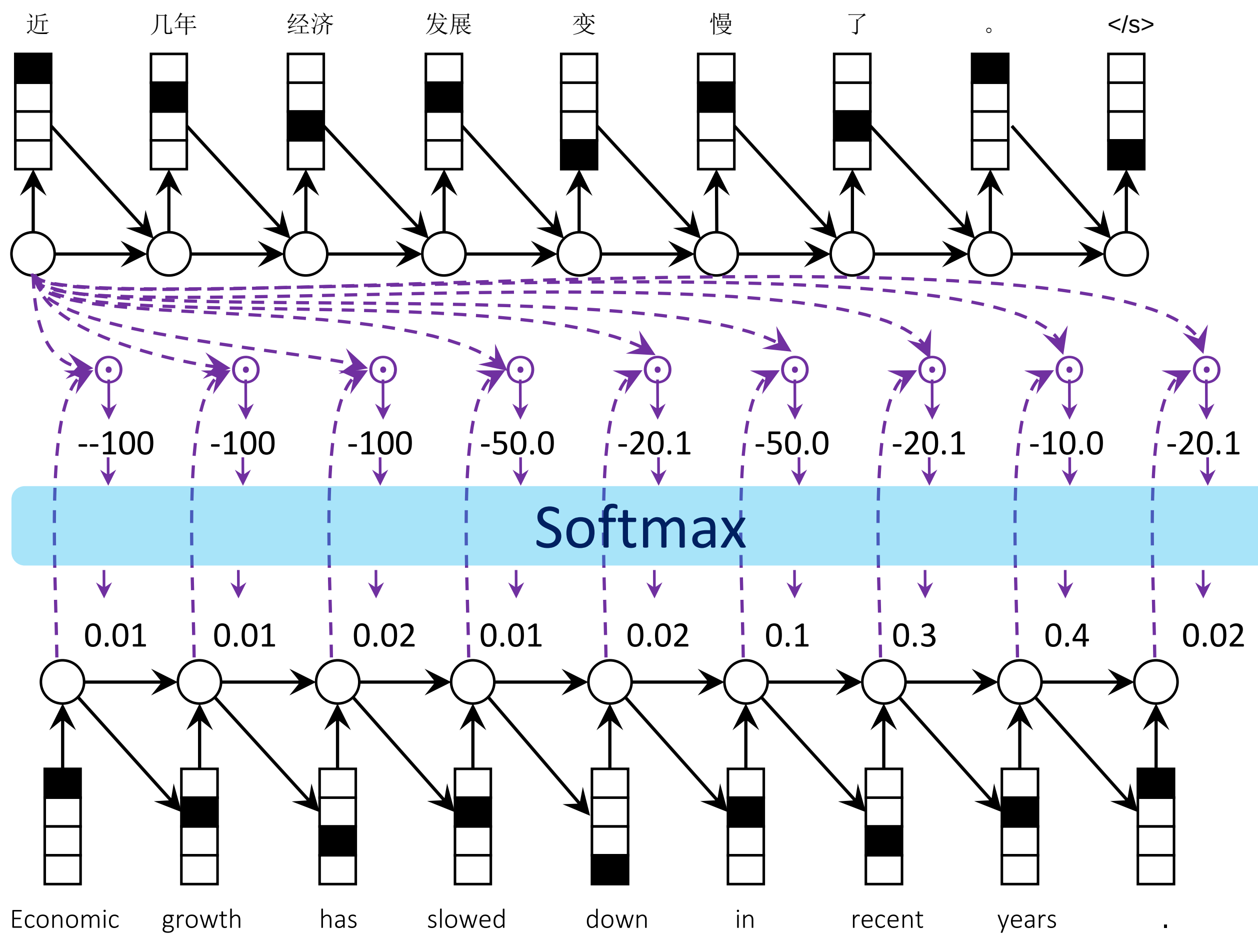



近
几年
经济
发展
变
慢
了
。
</s>
Economic
growth
has
slowed
down
in
recent
years
.

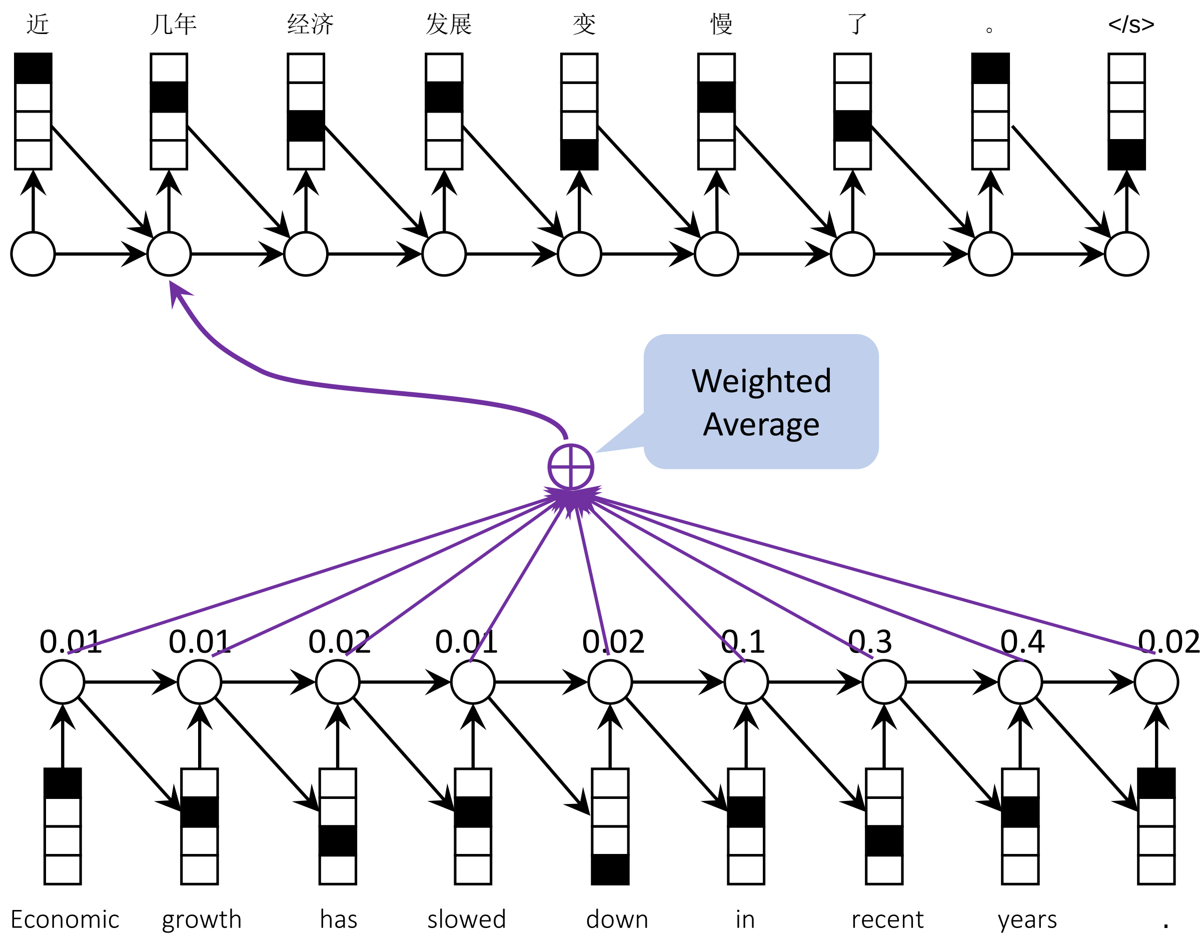

近
几年
经济
发展
变
慢
了
。
</s>
?
?
?
?
?
?
?
?
?
Economic
growth
has
slowed
down
in
recent
years
.



近
几年
经济
发展
变
慢
了
。
</s>
-100
-100
-100
-50.0
-20.1
-50.0
-20.1
-10.0
-20.1
Economic
growth
has
slowed
down
in
recent
years
.

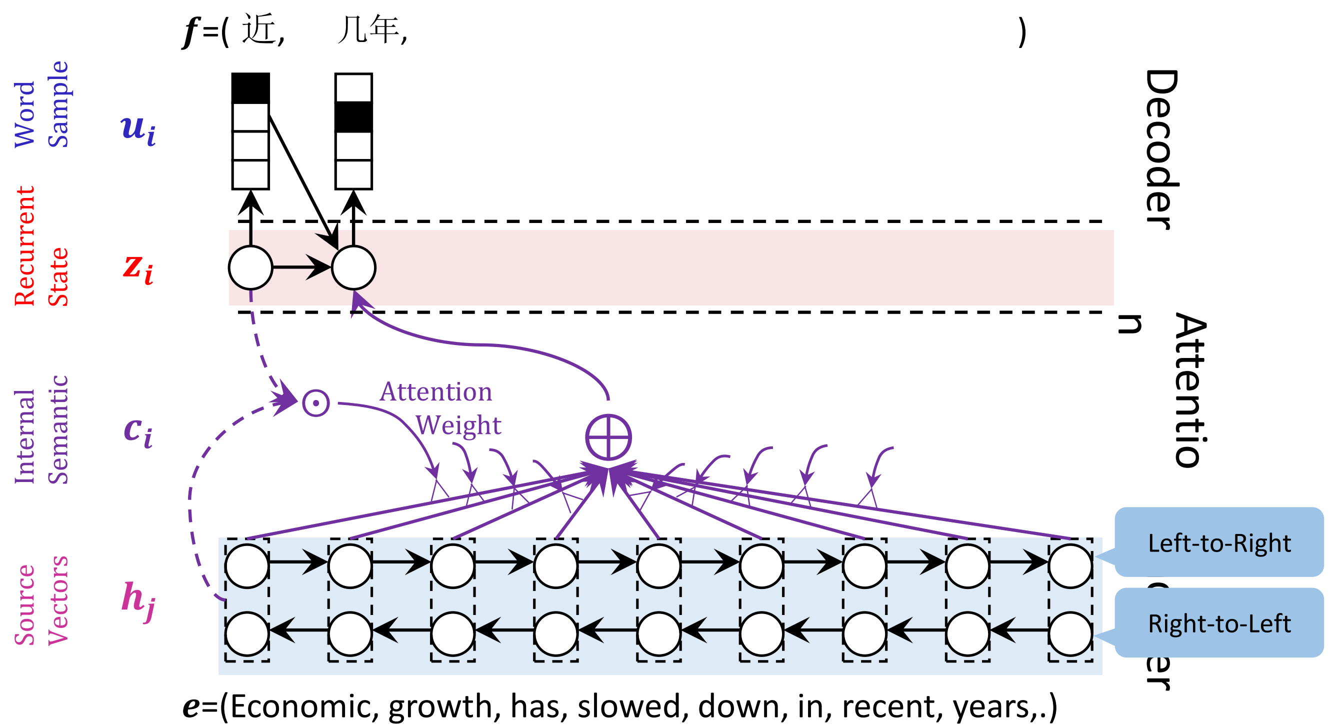


近
几年
经济
发展
变
慢
了
。
</s>
--100
-100
-100
-50.0
-20.1
-50.0
-20.1
-10.0
-20.1
Softmax
0.01
0.01
0.02
0.01
0.02
0.1
0.3
0.4
0.02
Economic
growth
has
slowed
down
in
recent
years
.



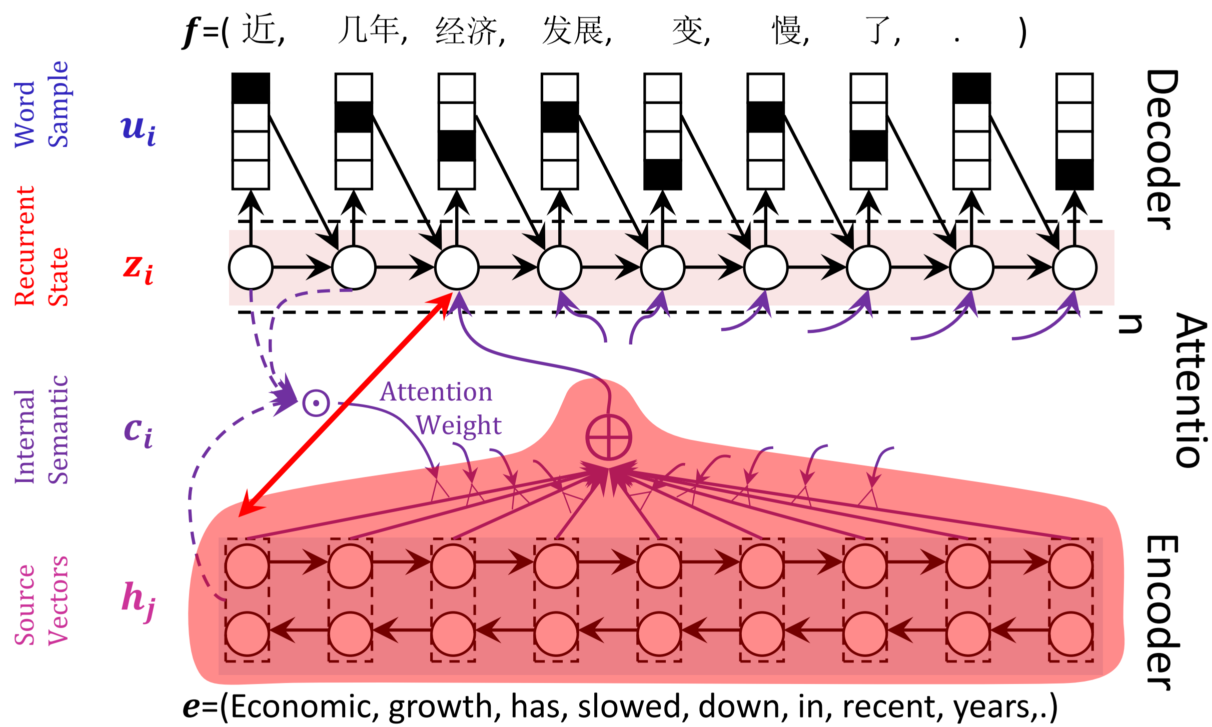
近
几年
经济
发展
变
慢
了
。
</s>
Weighted Average
0.01
0.01
0.02
0.01
0.02
0.1
0.3
0.4
0.02
Economic
growth
has
slowed
down
in
recent
years
.

Attention based Encoder-Decoder



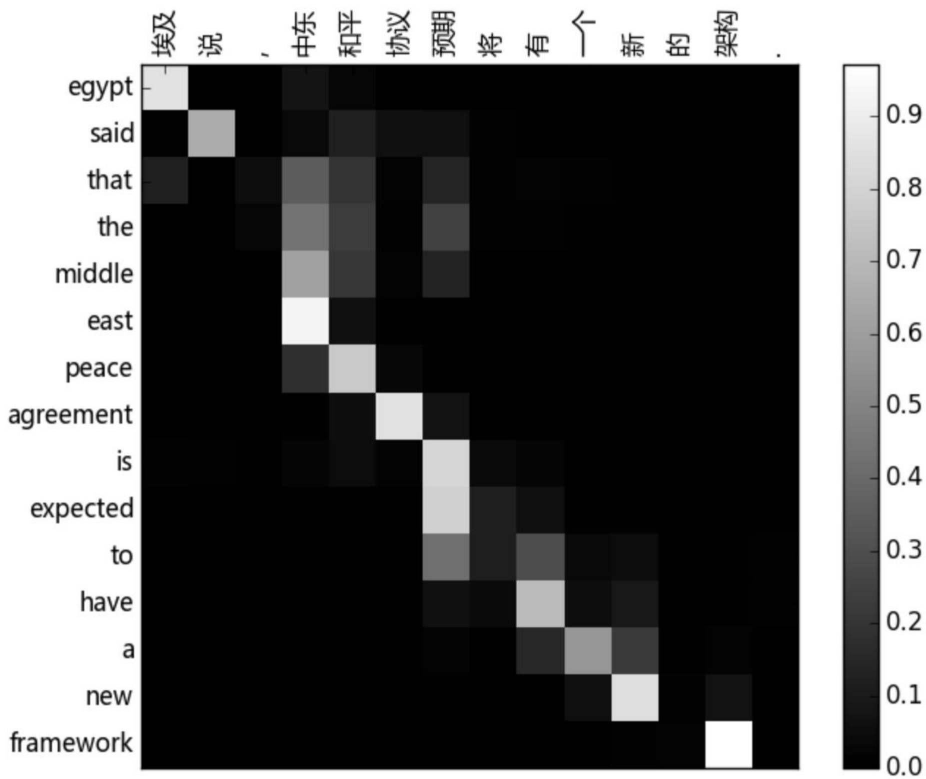
Bengio, ICLR 2015

Attention based Encoder-Decoder



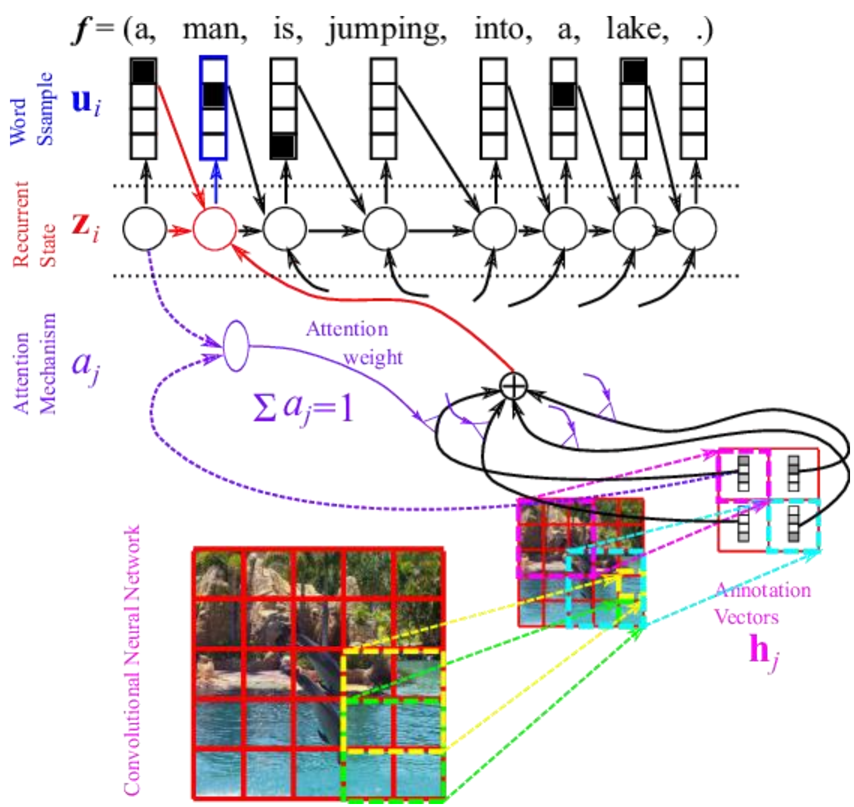
Bengio, ICLR 2015

Case Study of Attention



$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_k \exp(e_{ik})}$$

Image Caption Generation with Attention



not learn good alignments: the *global (location)* model can only obtain a small gain when performing unknown word replacement compared to using other alignment functions.[14] For *content-based* functions, our implementation *concat* does not yield good performances and more analysis should be done to understand the reason.[15] It is interesting to observe that *dot* works well for the global attention and *general* is better for the local attention. Among the different models, the local attention model with predictive alignments (*local-p*) is best, both in terms of perplexities and BLEU.

Case Study of Attention

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_k e_{ik}}$$

Thanks