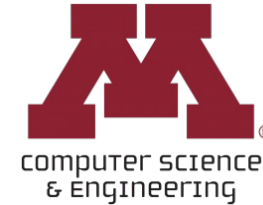


# CSCI 5541: Natural Language Processing

## Lecture 6: Language Models: N-grams, Neural LM, RNN, Seq2Seq



UNIVERSITY OF MINNESOTA  
**Driven to Discover®**

# Three ways of looking at word meaning



## ❑ Decompositional

- What **characteristics/components** of what the word represents

## ❑ Ontological

- How the meaning of the word **relates** to the meanings of other words

## ❑ Distributional

- What **contexts** the word is found in, relative to other words



# Count-based vs Prediction-based Methods

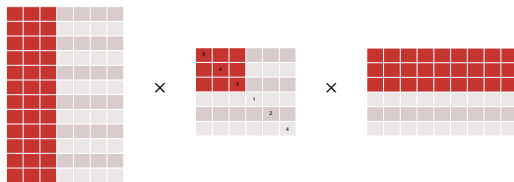
To obtain good dense representations



**LSA, HAL** (Lund & Burgess)

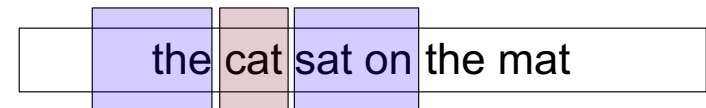
**Hellinger-PCA** (Rohde et al, Lebrete & Collobert)

	Hamlet	Macbeth
knife	1	1
dog		
sword	2	2
love	64	
like	75	38



**Skip-gram/CBOW** (Mikolov et al)

**NLM, HLBL, RNN** (Bengio et al; Collobert & Weston; Huang et al; Mnih & Hinton)



# Different kinds of encoding “context”



## ☒ ~~Count-based~~

- PMI, TF-IDF

## ☒ ~~Distributed prediction-based (type) embeddings~~

- Word2vec, GloVe, Fasttext

## ☐ Distributed contextual (token) embeddings from **language models**

- ELMo, BERT, GPT

## ☐ Many more variants

- Multilingual / multi-sense / syntactic embeddings, etc





# Outline

- ❑ Language modeling
- ❑ Applications of language models
- ❑ How to estimate  $P(w)$  from data? Ngram Language Model (LM)
- ❑ Advanced techniques for ngram LM
- ❑ Ngram LM vs Neural LM
- ❑ Linearization: A general heuristic for model improvement
- ❑ Recurrent Neural Network (RNN)
- ❑ Long Short-term Memory (LSTM)
- ❑ Implementation of RNN and LSTM using PyTorch
- ❑ Sequence-to-Sequence modeling



# Which sentence is more natural?

“man bites dog”

“dog bites man”

“bites man dog”

# Language modeling

- Provide a way to quantify the likelihood of a sequence
  - i.e., **plausible** sentences
- Vocabulary ( $V$ ) is a finite set of discrete symbols (e.g., words, characters);
  - ~170K words for English, ~150K words for Russian, ~1.1M words for Korean, ~85K words for Chinese
- $V^+$  is the infinite set of **sequences** of symbols from  $V$ ; each sequence ends with **STOP**
  - A sentence of  $k$  words:  $V * V .. * V = V^k$  e.g.,  $170,000^{100}$  for English 100-length sentence



sequence

$$P(w) = P(w_1, \dots, w_n)$$

$$\begin{aligned} &P(\text{"Call me DK"}) \\ &= P(w_1 = \text{"Call"}, w_2 = \text{"me"}, w_3 = \text{"DK"}) \times P(\text{"STOP"}) \end{aligned}$$

$$\sum_{w \in V^+} P(w) = 1 \quad 0 \leq P(w) \leq 1$$

over all the possible sequences of words



# Which sentence is more natural?

*"Call me DK"*

$$P(\text{"Call me DK"}) = 10^{-5}$$

*"DK me Call"*

$$P(\text{"DK me call"}) = 10^{-15}$$



# Use Cases of Language Model

- ❑ Provide a way to quantify the likelihood of a sequence i.e., **plausible** sentences

- Probability distributions over sentences (i.e., word sequences)

$$P(w) = P(w_1, \dots, w_n)$$

- ❑ Can use them to generate strings

- $P(w_k \mid w_2 w_3 w_4 \dots w_{k-1})$

- ❑ Rank possible sentences

- $P(\text{"Today is Thursday"}) > P(\text{"Thursday Today is"})$
  - $P(\text{"Today is Thursday"}) > P(\text{"Today is Minneapolis"})$



# Applications of language models



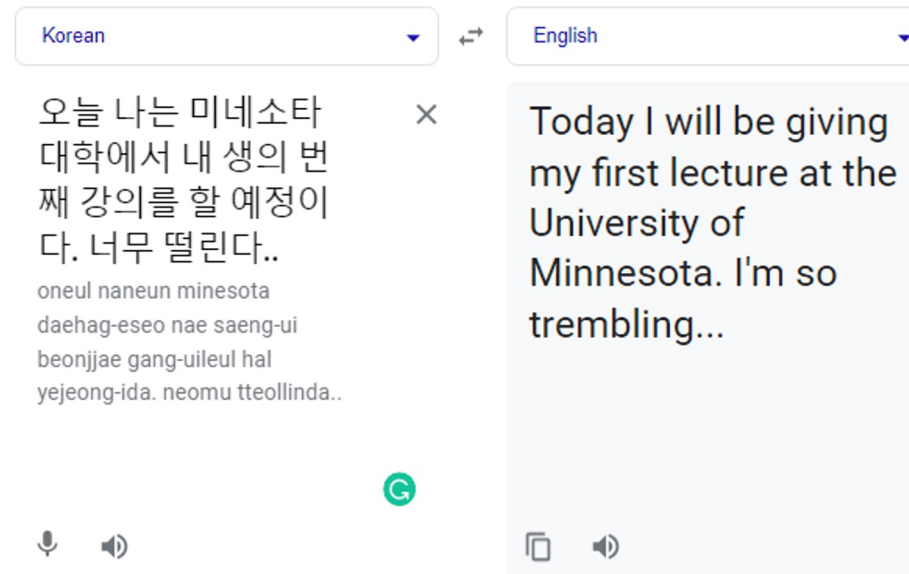
# What is natural language generation?

- ❑ NLP = Natural Language Understanding (NLU) + Natural Language Generation (NLG)
- ❑ NLG focuses on systems that produce **coherent** and **useful** language output for human consumption
- ❑ Deep Learning is powering (some) next-gen NLG systems

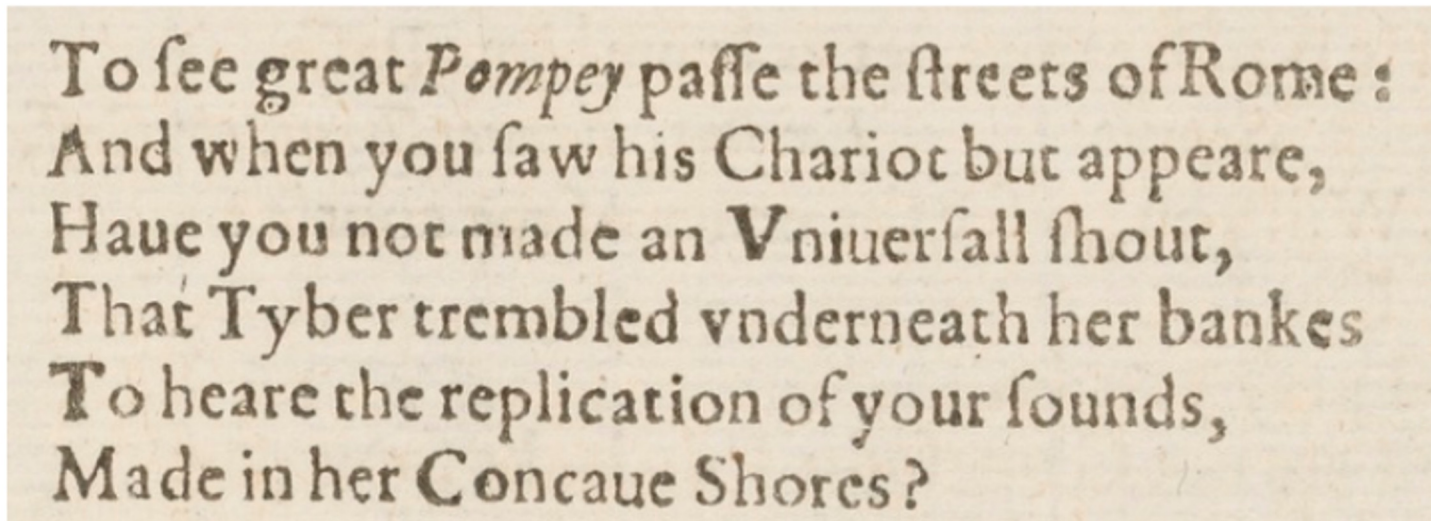




# Machine Translation



# Optical Character Recognition (OCR)



To see great *Pompey* passe the streets of Rome :  
And when you saw his Chariot but appeare,  
Haue you not made an Vniuersall shout,  
That Tyber trembled vnderneath her bankes  
To heare the replication of your sounds,  
Made in her Concaue Shores?

to fee great Pompey paffe the Areets of Rome:

to see great Pompey passe the streets of Rome:

# Speech Recognition



'Scuse me while I kiss this guy

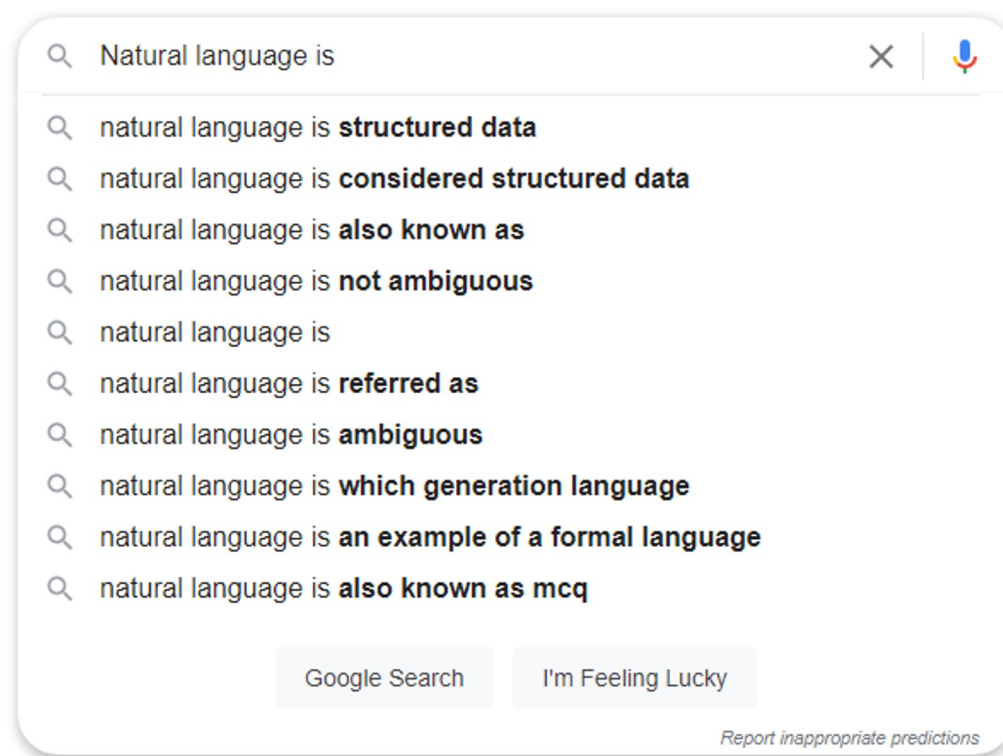
'Scuse me while I kiss the sky

'Scuse me while I kiss this fly

'Scuse me while my biscuits fry



# Automatic Completion



$$P(w_k \mid w_2 w_3 w_4 \dots w_{k-1})$$

# Language Generation

## Router: A Methodology for the Typical Unification of Access Points and Redundancy

Jeremy Stribling, Daniel Aguayo and Maxwell Krohn

### ABSTRACT

Many physicists would agree that, had it not been for congestion control, the evaluation of web browsers might never have occurred. In fact, few hackers worldwide would disagree with the essential unification of voice-over-IP and public-private key pair. In order to solve this riddle, we confirm that SMPs can be made stochastic, cacheable, and interposable.

### I. INTRODUCTION

Many scholars would agree that, had it not been for active networks, the simulation of Lamport clocks might never have occurred. The notion that end-users synchronize with the investigation of Markov models is rarely outdated. A theoretical grand challenge in theory is the important unification of virtual machines and real-time theory. To what extent can web browsers be constructed to achieve this purpose?

Certainly, the usual methods for the emulation of Smalltalk that paved the way for the investigation of rasterization do

The rest of this paper is organized as follows. For starters, we motivate the need for fiber-optic cables. We place our work in context with the prior work in this area. To address this obstacle, we disprove that even though the much-touted autonomous algorithm for the construction of digital-to-analog converters by Jones [10] is NP-complete, object-oriented languages can be made signed, decentralized, and signed. Along these same lines, to accomplish this mission, we concentrate our efforts on showing that the famous ubiquitous algorithm for the exploration of robots by Sato et al. runs in  $\Omega((n + \log n))$  time [22]. In the end, we conclude.

### II. ARCHITECTURE

Our research is principled. Consider the early methodology by Martin and Smith; our model is similar, but will actually overcome this grand challenge. Despite the fact that such a claim at first glance seems unexpected, it is buffeted by previous work in the field. Any significant development of secure theory will clearly require that the acclaimed real



Write a book report on a tale of two cities 300 words



A Tale of Two Cities is a novel by Charles Dickens set during the time of the French Revolution. The novel is set in Paris and London and follows the lives of several characters as they are caught up in the events of the revolution.

The novel opens with the famous line, "it was the best of times, it was the worst of times," setting the stage for the tumultuous events that are to come. The story centers around Charles Darnay, a French aristocrat, and Sydney Carton, a drunken lawyer who bears a striking resemblance to Darnay.

As

<https://pdos.csail.mit.edu/archive/scigen/>



# Dialogue Generation

Q: How many rainbows does it take to jump from Hawaii to seventeen?

A: It takes two rainbows to jump from Hawaii to seventeen.

Q: Which colorless green ideas sleep furiously?

A: Ideas that are colorless, green, and sleep furiously are the ideas of a sleep furiously.

Q: Do you understand these questions?

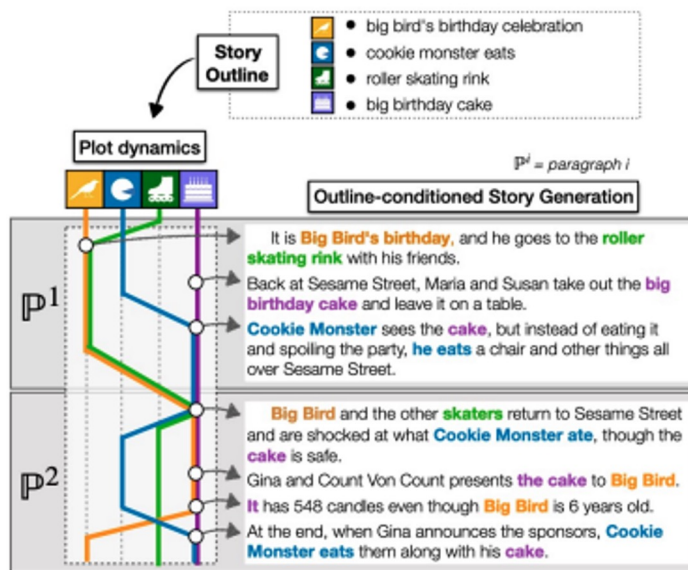
A: I understand these questions.

Giving GPT-3 a Turing Test, Kevin Lacker's blog, <https://lacker.io/ai/2020/07/06/giving-gpt-3-a-turing-test.html>





# More interesting NLG uses



Creative story generation

Table Title: Robert Craig (American football)  
Section Title: National Football League statistics  
Table Description Note:

YEAR	TEAM	ATT	YDS	AVG	LNG	TD	RUSHING					NO.	YDS	AVG	LNG	TD
1983	SF	176	725	4.1	71	8	48	427	8.9	23	4					
1984	SF	155	649	4.2	28	4	71	675	9.5	64	3					
1985	SF	214	1050	4.9	62	9	92	1016	11	73	6					
1986	SF	204	830	4.1	25	7	81	654	7.7	48	0					
1987	SF	215	815	3.8	25	3	66	492	7.5	35	1					
1988	SF	310	1302	4.8	46	9	76	534	7.0	22	1					
1989	SF	271	1054	3.9	27	6	49	473	9.7	44	1					
1990	SF	141	439	3.1	26	1	25	201	8.0	31	0					
1991	RAI	162	590	3.6	15	1	17	136	8.0	20	0					
1992	MIN	105	416	4.0	21	4	22	164	7.5	22	0					
1993	MIN	38	119	3.1	11	1	19	169	8.9	31	1					
Totals	-	1991	8189	4.1	71	56	564	4911	8.7	73	17					

Craig finished his eleven NFL seasons with 8,189 rushing yards and 566 receptions for 4,911 receiving yards.


Data/Table to text



Two children are sitting at a table in a restaurant. The children are one little girl and one little boy. The little girl is eating a pink frosted donut with white icing lines on top of it. The girl has blonde hair and is wearing a green jacket with a black long sleeve shirt underneath. The little boy is wearing a black zip up jacket and is holding his finger to his lip but is not eating. A metal napkin dispenser is in between them at the table. The wall next to them is white brick. Two adults are on the other side of the short white brick wall. The room has white circular lights on the ceiling and a large window in the front of the restaurant. It is daylight outside.

Visual description

ST

Can you write out an Adobe After Effects expression to make a shape layer wiggle when a null object is within 50 pixels of the shape's anchor point. 





Language modeling is the task of estimating  $P(w)$

How to estimate  $P(w)$  from data?



# Chain rule (of probability)

$$\begin{aligned} P(x_1, x_2, x_3, x_4, x_5) = & P(x_1) \\ & \times P(x_2|x_1) \\ & \times P(x_3|x_1, x_2) \\ & \times P(x_4|x_1, x_2, x_3) \\ & \times P(x_5|x_1, x_2, x_3, x_4) \end{aligned}$$



# Chain rule (of probability)

Repeatedly apply  
definition of  
conditional probability

$$\begin{aligned} P(x_1, x_2, x_3, x_4, x_5) &= P(x_1) \\ &\times P(x_2|x_1) \\ &\times P(x_3|x_1, x_2) \\ &\times P(x_4|x_1, x_2, x_3) \\ &\times P(x_5|x_1, x_2, x_3, x_4) \end{aligned}$$

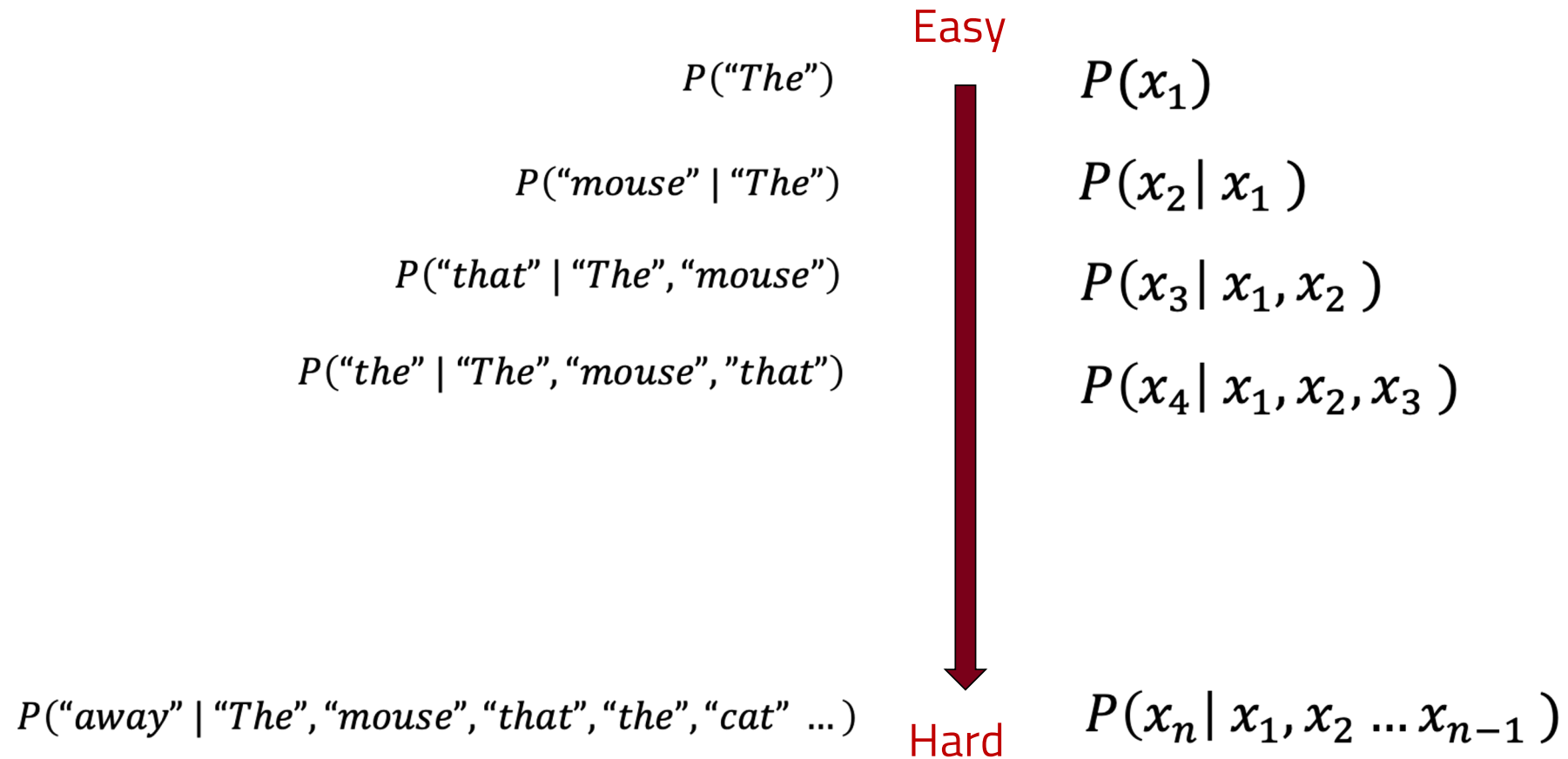
$$P(x_1, x_2) = P(x_2|x_1)P(x_1)$$




*"The mouse that the cat that the  
dog that the man frightened and  
chased ran away."*



*"The mouse that the cat that the dog that the man frightened and chased ran away."*



# Markov assumption

$$\begin{aligned}
 &= P(\cancel{x_1}) \\
 &\times P(x_2 | x_1) \\
 &\times P(x_3 | \cancel{x_1} x_2) \\
 &\times P(x_4 | \cancel{x_1} \cancel{x_2} x_3) \\
 &\times P(x_5 | \cancel{x_1} \cancel{x_2} \cancel{x_3} x_4) \\
 &= P(\cancel{x_1}) \\
 &\times P(\cancel{x_2} | x_1) \\
 &\times P(x_3 | x_1, x_2) \\
 &\times P(x_4 | \cancel{x_1} x_2, x_3) \\
 &\times P(x_5 | \cancel{x_1} \cancel{x_2} x_3, x_4)
 \end{aligned}$$

first-order

$$P(x_i | x_1, x_2 \dots x_{i-1}) \approx P(x_i | x_{i-1})$$

second-order

$$P(x_i | x_1, x_2 \dots x_{i-1}) \approx P(x_i | x_{i-2}, x_{i-1})$$



# Markov assumption

$$\begin{aligned}
 &= P(x_1) \\
 &\times P(x_2|x_1) \\
 &\times P(x_3|x_2) \\
 &\times P(x_4|x_3) \\
 &\times P(x_5|x_4)
 \end{aligned}
 \quad
 \begin{aligned}
 &= P(x_1) \\
 &\times P(x_2|x_1) \\
 &\times P(x_3|x_1, x_2) \\
 &\times P(x_4|x_1, x_2, x_3) \\
 &\times P(x_5|x_1, x_2, x_3, x_4)
 \end{aligned}$$

first-order

$$P(x_i | x_1, x_2 \dots x_{i-1}) \approx P(x_i | x_{i-1})$$

second-order

$$P(x_i | x_1, x_2 \dots x_{i-1}) \approx P(x_i | x_{i-2}, x_{i-1})$$



# Markov assumption

Bi-gram model  
(first-order markov)

$$P(w) = \prod_{i=1}^n P(w_i | w_{i-1}) \times P(\text{STOP} | w_n)$$

Tri-gram model  
(second-order markov)

$$P(w) = \prod_{i=1}^n P(w_i | w_{i-2}, w_{i-1}) \times P(\text{STOP} | w_{n-1}, w_n)$$





Tri-gram model  
(second-order markov)

$P(\text{"The"} \mid \text{START}_1, \text{START}_2)$

$P(\text{"mouse"} \mid \text{START}_2, \text{"The"})$

$P(\text{"that"} \mid \text{"The"}, \text{"mouse"})$

$P(\text{"the"} \mid \text{"mouse"}, \text{"that"})$

...

$P(\text{"away"} \mid \text{"chased"}, \text{"ran"})$

$P(\text{STOP} \mid \text{"ran"}, \text{"away"})$

*"The mouse that the cat  
that the dog that the man  
frightened and chased ran  
away."*



# Estimation from data

Uni-gram

$$\prod_{i=1}^n P(w_i) \\ \times P(STOP)$$

Bi-gram

$$\prod_{i=1}^n P(w_i | w_{i-1}) \\ \times P(STOP | w_n)$$

Tri-gram

$$\prod_{i=1}^n P(w_i | w_{i-2}, w_{i-1}) \\ \times P(STOP | w_{n-1} w_n)$$



# Estimation from data

Uni-gram

$$\prod_{i=1}^n \boxed{P(w_i)} \\ \times P(STOP)$$

Bi-gram

$$\prod_{i=1}^n \boxed{P(w_i | w_{i-1})} \\ \times P(STOP | w_n)$$

Tri-gram

$$\prod_{i=1}^n \boxed{P(w_i | w_{i-2}, w_{i-1})} \\ \times P(STOP | w_{n-1} w_n)$$

How do we calculate  
each of these  
probabilities?



# Estimation from data

Uni-gram

$$\prod_{i=1}^n P(w_i) \times P(STOP)$$

Bi-gram

$$\prod_{i=1}^n P(w_i | w_{i-1}) \times P(STOP | w_n)$$

Tri-gram

$$\prod_{i=1}^n P(w_i | w_{i-2}, w_{i-1}) \times P(STOP | w_{n-1} w_n)$$

Use the counts of words, pairs of words and groups of three words

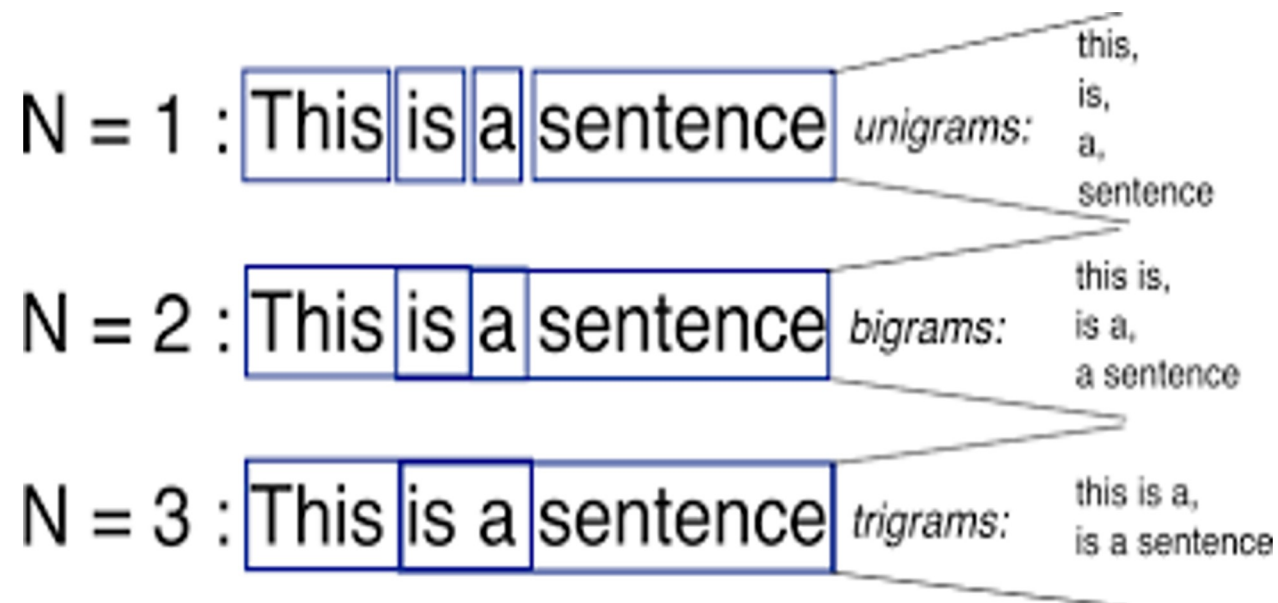
$$\frac{c(w_i)}{N}$$

$$\frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

$$\frac{c(w_{i-2}, w_{i-1}, w_i)}{c(w_{i-2}, w_{i-1})}$$

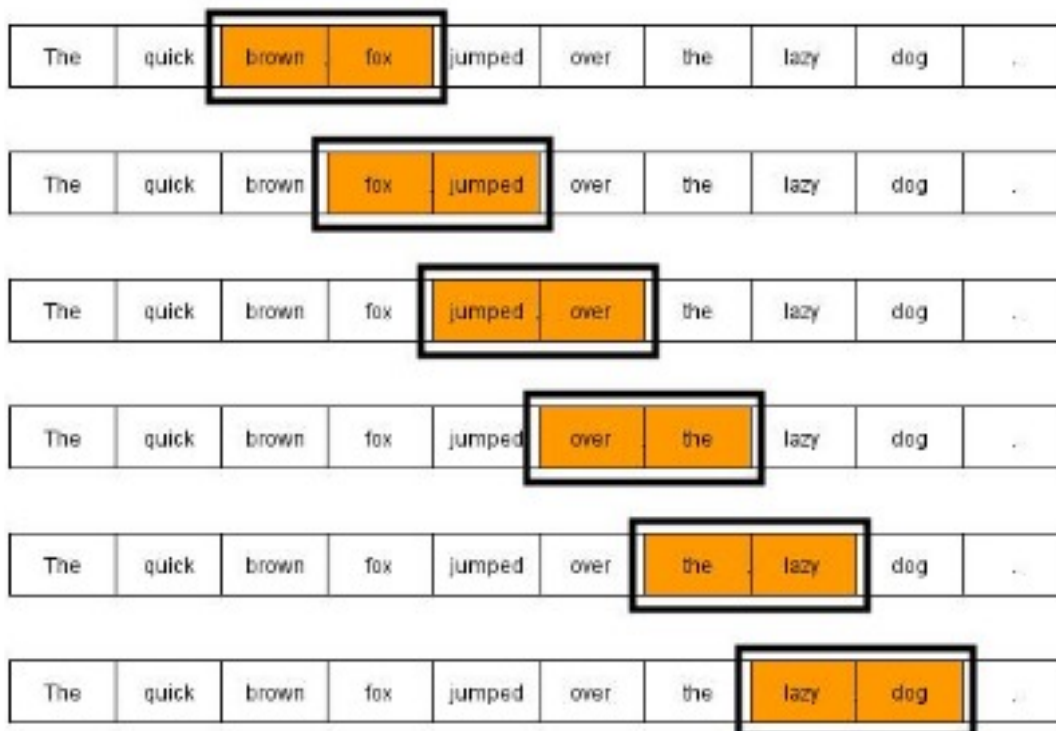


# Estimation from data



# Estimation from data

$$c(w_{i-1}, w_i)$$



## Part of A Unigram Distribution trained on academic papers

[rank 1]

$p(\text{the}) = 0.038$

$p(\text{of}) = 0.023$

$p(\text{and}) = 0.021$

$p(\text{to}) = 0.017$

$p(\text{is}) = 0.013$

$p(\text{a}) = 0.012$

$p(\text{in}) = 0.012$

$p(\text{for}) = 0.009$

...

...

[rank 1001]

$p(\text{joint}) = 0.00014$

$p(\text{relatively}) = 0.00014$

$p(\text{plot}) = 0.00014$

$p(\text{DEL1SUBSEQ}) = 0.00014$

$p(\text{rule}) = 0.00014$

$p(62.0) = 0.00014$

$p(9.1) = 0.00014$

$p(\text{evaluated}) = 0.00014$

...



# Generated text from a uni-gram model

first, from less the This different 2004), out which goal 19.2  
Model their It ~(i?1), given 0.62 these (x0; match 1 schedule. x 60  
1998. under by Notice we of stated CFG 120 be 100 a location  
accuracy If models note 21.8 each 0 WP that the that Nov?ak. to  
function; to [0, to different values, model 65 cases. said - 24.94  
sentences not that 2 In to clustering each K&M 100 Boldface X))]  
applied; In 104 S. grammar was (Section contrastive thesis, the  
machines table -5.66 trials: An the textual (family  
applications. We have for models 40.1 no 156 expected are  
neighborhood





# Generated text from a bi-gram model

e. (A.33) (A.34) A.5 Models are also been completely surpassed in performance on drafts of **online algorithms** can achieve **far more** so while substantially improved using CE. 4.4.1 MLEasaCaseofCE 71 26.34 23.1 57.8 K&M 42.4 62.7 40.9 44 43 90.7 100.0 100.0 100.0 15.1 30.9 18.0 21.2 60.1 undirected evaluations directed DEL1 TRANS1 neighborhood. **This continues**, with supervised init., semisupervised MLE with the METU- SabanciTreebank 195 ADJA ADJD ADV APPR APPRART APPO APZR ART CARD FM ITJ KOU1 KOUS KON KOKOM NN NN NN IN JJ NNTheir problem is  $y \propto x$ . The evaluation offers the hypothesized link grammar with a Gaussian



# Generated text from a tri-gram model

top(xl ,right,B). (A.39) vine0(X, l) rconstit0(l 1, l). (A.40) vine(n). (A.41) These equations **were presented in** both cases; these scores u<AC>into a probability distribution is even smaller(r =0.05). This is exactly fEM. During DA, is gradually relaxed. This approach could be efficiently used in previous chapters) before training (test) K&MZeroLocalrandom models Figure4.12: Directed accuracy on all six languages. Importantly, these papers **achieved state- of-the-art results on their tasks** and unlabeled data and the verbs are allowed (for instance) to select the cardinality of discrete structures, like matchings on weighted graphs (**McDonald et al., 1993**) (35 tag types, 3.39 bits). The Bulgarian,



# Evaluation for Language Models

- ❑ The best evaluation metrics are **external**
  - How does a better language model influence the application you care about?
  - E.g.,
    - machine translation (BLEU score)
    - sentiment classification (F1 score)
    - speech recognition (word error rate)



# (Intrinsic) Evaluation

- ❑ A good language model should judge **unseen real language** to have high probability
- ❑ **Perplexity** = inverse probability of test data, averaged by word
  - Better models have lower perplexity
- ❑ To be reliable, the test data must be truly unseen (including knowledge of its vocabulary)

$$\text{Perplexity} = \sqrt[N]{\frac{1}{P(w_1, \dots, w_n)}}$$



$$\sqrt[N]{\frac{1}{\prod_i^N P(w_i)}} = \left( \prod_i^N P(w_i) \right)^{-\frac{1}{N}}$$



$$\begin{aligned}
\sqrt[N]{\frac{1}{\prod_i^N P(w_i)}} &= \left( \prod_i^N P(w_i) \right)^{-\frac{1}{N}} \\
&= \text{exp log} \left( \prod_i^N P(w_i) \right)^{-\frac{1}{N}} \\
&= \exp \left( -\frac{1}{N} \log \prod_i^N P(w_i) \right) \\
\text{Perplexity} &= \exp \left( -\frac{1}{N} \sum_i^N \log P(w_i) \right)
\end{aligned}$$



$$\sqrt[N]{\frac{1}{\prod_i^N P(w_i)}} = \left( \prod_i^N P(w_i) \right)^{-\frac{1}{N}}$$

$$= \text{exp log} \left( \prod_i^N P(w_i) \right)^{-\frac{1}{N}}$$

$$= \exp \left( -\frac{1}{N} \log \prod_i^N P(w_i) \right)$$

Perplexity

$$= \exp \left( -\frac{1}{N} \sum_i^N \log P(w_i) \right)$$

Bi-gram

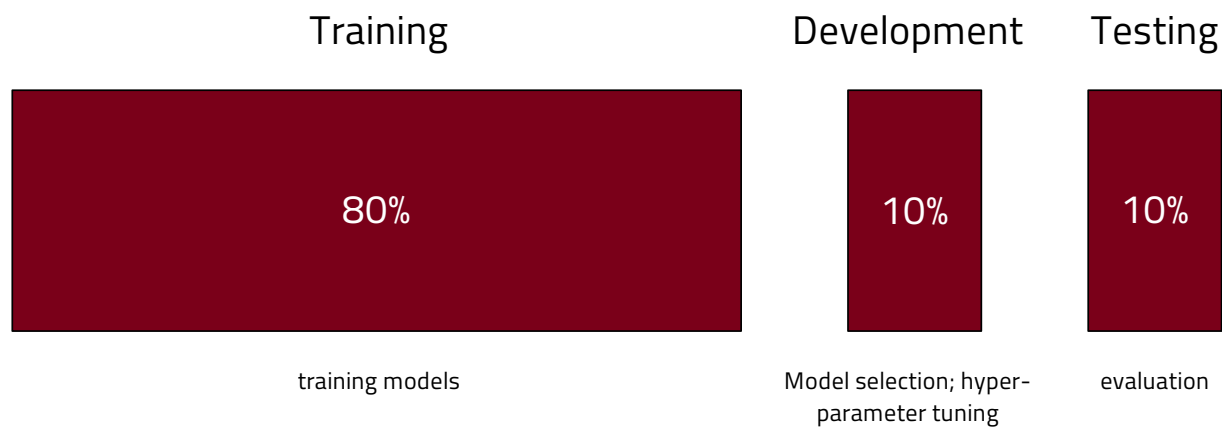
$$P(w_i | w_{i-1})$$

Tri-gram

$$P(w_i | w_{i-2}, w_{i-1})$$



# Intrinsic Evaluation





# Perplexity

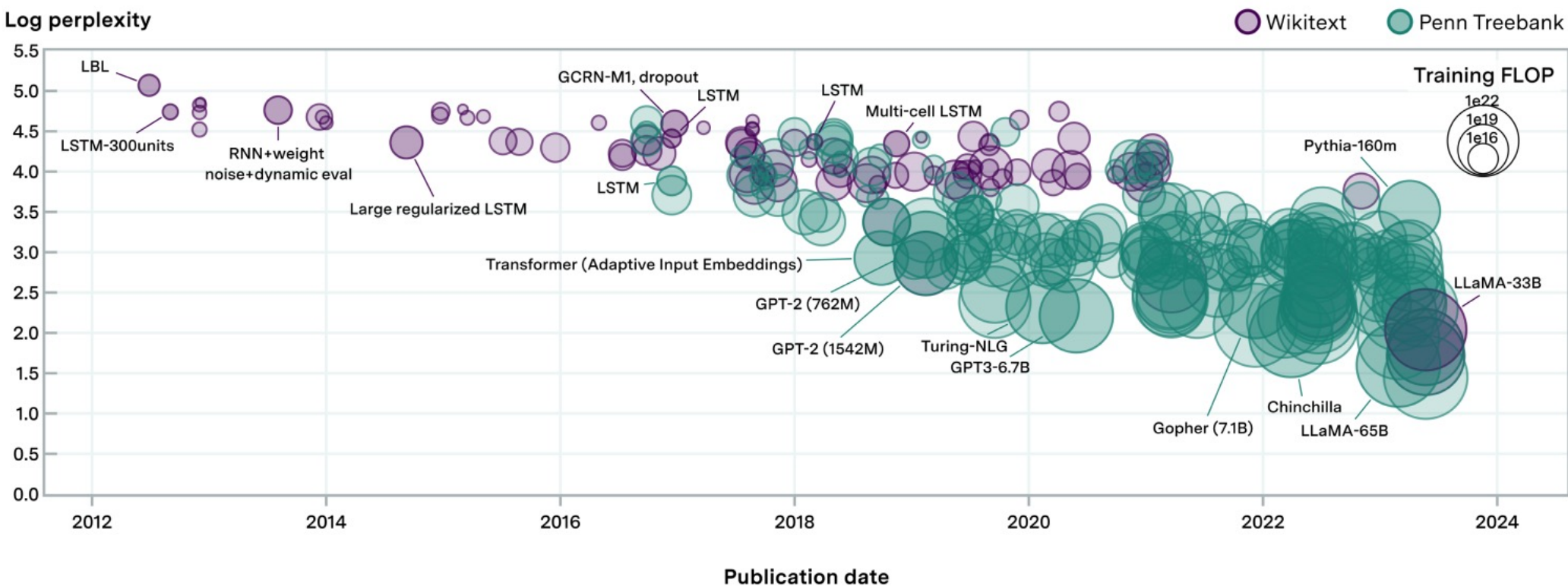
Model	Unigram	Bigram	Trigram
Perplexity	962	170	109

On PennTreeBank test set



# Performance and scale of language models over time

Log perplexity



# Advanced techniques for ngram LM



# Data sparsity

- Training data is a small (and biased) sample of the **creativity** of language.

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

$$\frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

**Figure 4.1** Bigram counts for eight of the words (out of  $V = 1446$ ) in the Berkeley Restaurant Project corpus of 9332 sentences. Zero counts are in gray.

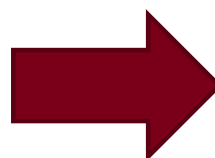
SLP3 4.1



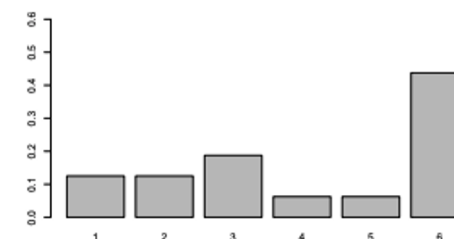
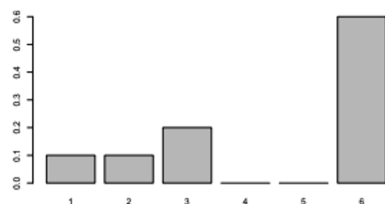
# Additive Smoothing

Uni-gram

$$\frac{c(w_i)}{N}$$



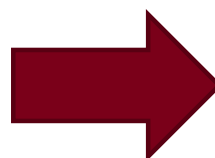
$$\frac{c(w_i) + \alpha}{N + V\alpha}$$



smoothing with  $\alpha = 1$

Bi-gram

$$\frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$



$$\frac{c(w_{i-1}, w_i) + \alpha}{c(w_{i-1}) + V\alpha}$$

## Kneser-ney smoothing

Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. Technical Report TR-10-98, Center for Research in Computing Technology, Harvard University, 1998.



# Interpolation over different LMs

- ❑ As ngram order rises, we have the potential for higher **precision** but also higher **variability** in our estimates.
- ❑ A linear interpolation of any two language models  $p$  and  $q$  (with  $\lambda \in [0,1]$ ) is also a valid language model, to reduce the variability

$$\lambda p + (1 - \lambda)q$$

$p$  = LM of web

$q$  = LM of political speeches



# Interpolation over higher-order LMs

□ How do we pick the best values of  $\lambda$ ?

- Grid search over Dev set

$$\begin{aligned} P(w_i \mid w_{i-2}, w_{i-1}) &= \lambda_1 P(w_i \mid w_{i-2}, w_{i-1}) \\ &\quad + \lambda_2 P(w_i \mid w_{i-1}) \\ &\quad + \lambda_3 P(w_i) \end{aligned}$$

$$\lambda_1 + \lambda_2 + \lambda_3 = 1$$



# Stupid backoff

back off to lower order ngram if the higher order is not observed.

if full sequence observed

$$S(w_i \mid w_{i-k+1}, \dots, w_{i-1}) = \frac{c(w_{i-k+1}, \dots, w_i)}{c(w_{i-k+1}, \dots, w_{i-1})}$$

Otherwise

$$= \lambda S(w_i \mid w_{i-k+2}, \dots, w_{i-1})$$

Cheap to calculate; works well when there is a **lot of data**

Brants et al. (2007), "Large Language Models in Machine Translation"





# Ngram LM vs Neural LM

To avoid the data sparsity  
problem from the ngram LM



# Neural LM

$$x = [v(w_1); \dots v(w_k)]$$

Concatenation ( $k \times V$ )

$w_1 = \text{tried}$

$w_2 = \text{to}$

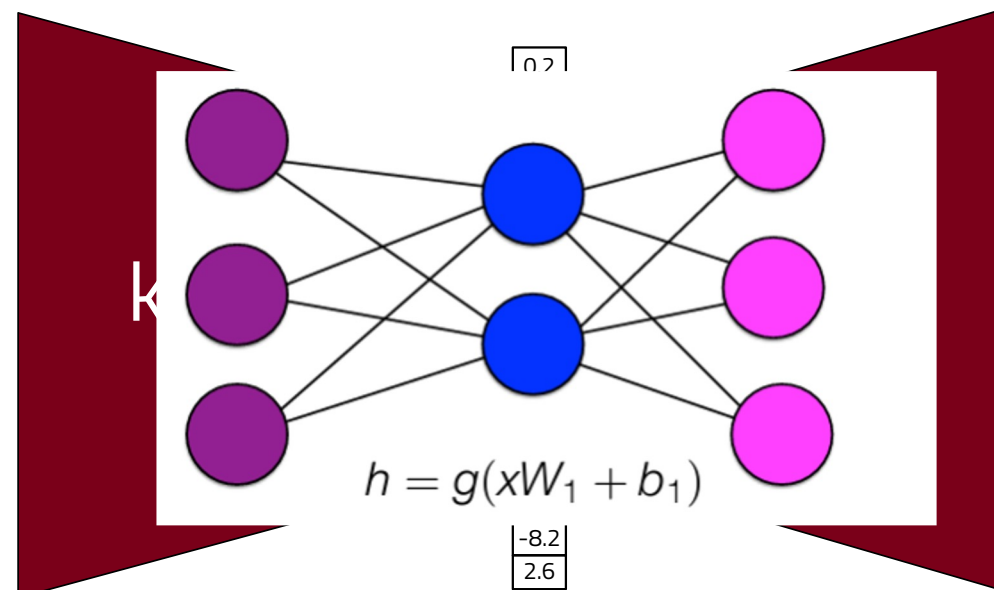
$w_3 = \text{prepare}$

$w_4 = \text{midterms}$

$v(w_1)$	1	0	0	0
$v(w_2)$	0	1	0	0
$v(w_3)$	0	0	1	0
$v(w_4)$	0	0	0	1

One-hot encoding

Simple feed-forward multilayer perceptron  
(e.g., one hidden layer)



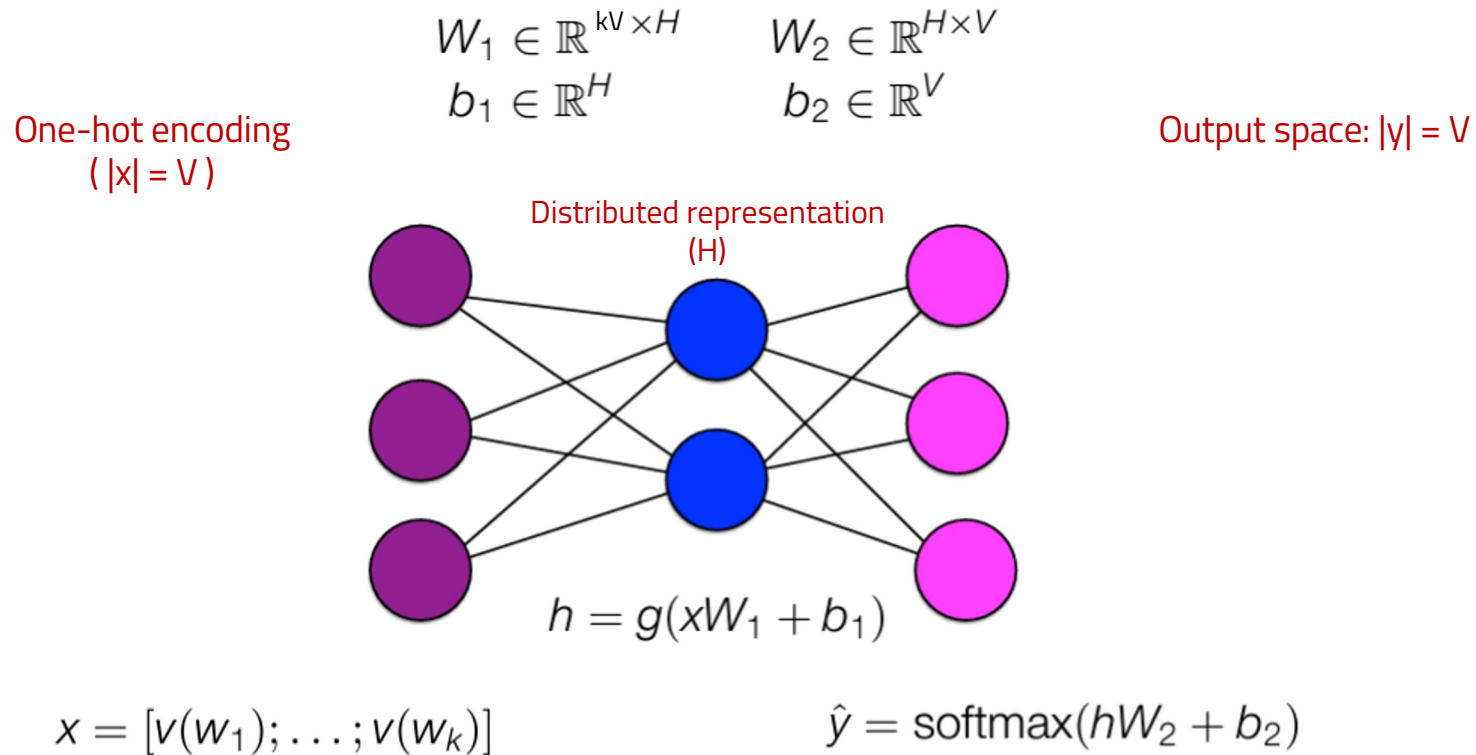
Distributed representation

Multi-class (Vocab)  
classification

Bengio et al. 2003, A Neural Probabilistic Language Model

# Neural LM

$$P(w) = P(w_i | w_{i-k} \dots w_{i-1}) = \text{softmax}(W \cdot h)$$



Bengio et al. 2003, A Neural Probabilistic Language Model

# Neural LM

Represent high-dimensional words (and contexts) as low-dimensional vectors

One-hot encoding  
( $|x| = V$ )

Distributed representation  
( $|y| = H$ )



$V \gg H$

Bengio et al. 2003, A Neural Probabilistic Language Model



Conditioning context ( $X [k \times V]$ )

tried to prepare midterm but I was too tired of...

Next word to predict ( $Y$ )

Context window size:  $k=4$



Conditioning context ( $X [k \times V]$ )

tried to prepare midterm but I was too tired of...

Next word to predict ( $Y$ )

Context window size:  $k=4$



Conditioning context ( $X [k \times V]$ )

tried to prepare midterm but I was too tired of...

Next word to predict ( $Y$ )

Context window size:  $k=4$



# Neural LM against Ngram LM

## Pros

- ❑ No sparsity problem
- ❑ Don't need to store all observed n-gram counts

## Cons

- ❑ Fixed context window is too small (larger window, larger  $W$ )
  - Windows can never be large enough
- ❑ Different words are multiplied by completely different weights ( $W$ ); no **symmetry** in how the inputs are processed.





# Outline

- ❑ Linearization: A general heuristic for model improvement
- ❑ Recurrent Neural Network (RNN)
- ❑ Long Short-term Memory (LSTM)
- ❑ Implementation of RNN and LSTM using PyTorch
- ❑ Sequence-to-Sequence modeling

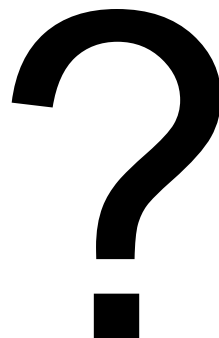


# Outline

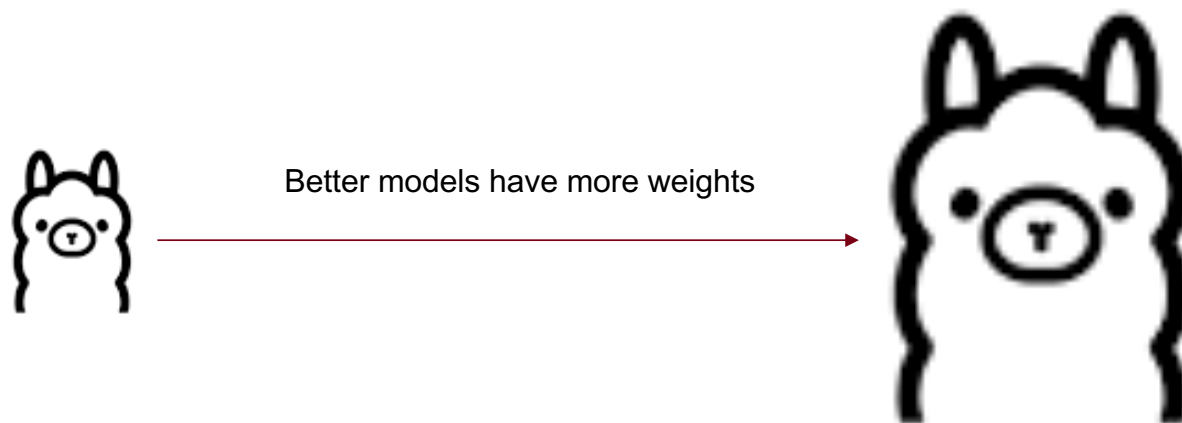
- ❑ **Linearization: A general heuristic for model improvement**
- ❑ Recurrent Neural Network (RNN)
- ❑ Long Short-term Memory (LSTM)
- ❑ Implementation of RNN and LSTM using PyTorch
- ❑ Sequence-to-Sequence modeling



# How do we make a better model?

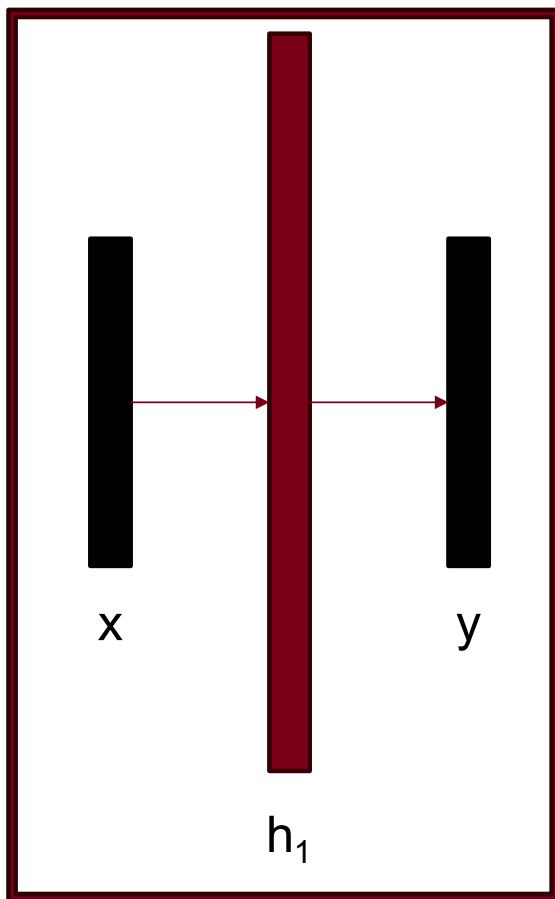


# More Params are Better

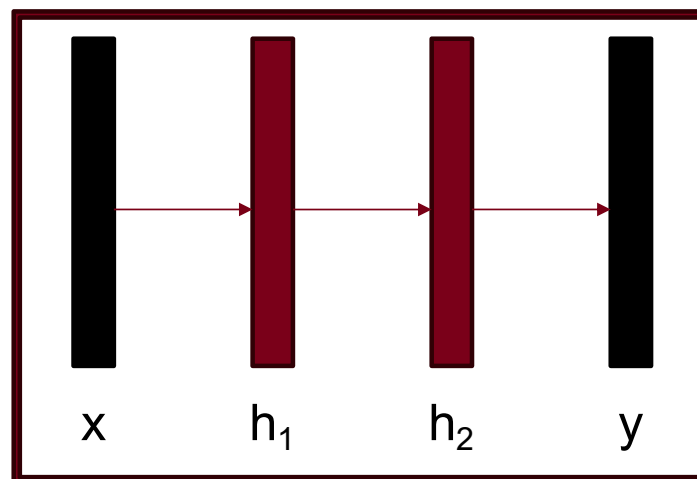


# Increasing depth is more efficient than width

Model 1: ❌



Model 2: ✅



...but very deep models are harder to train

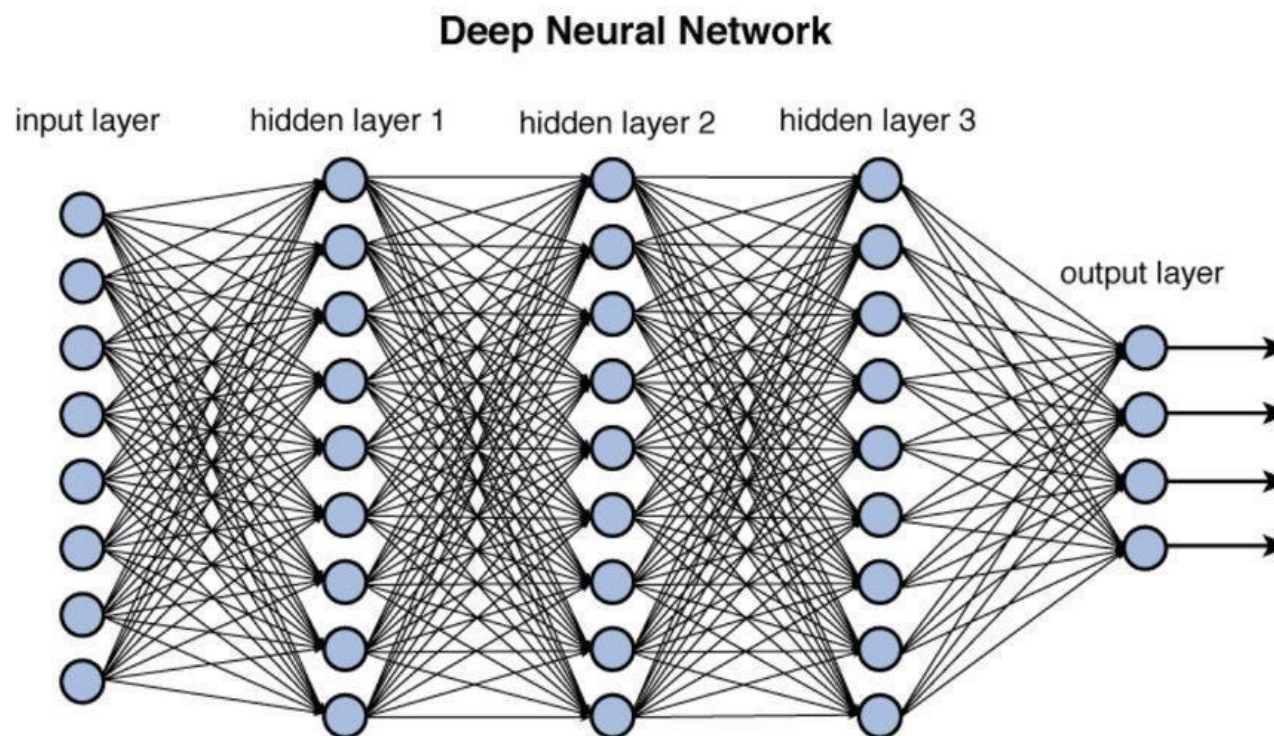


Figure 12.2 Deep network architecture with multiple layers.

# Why is this so challenging?

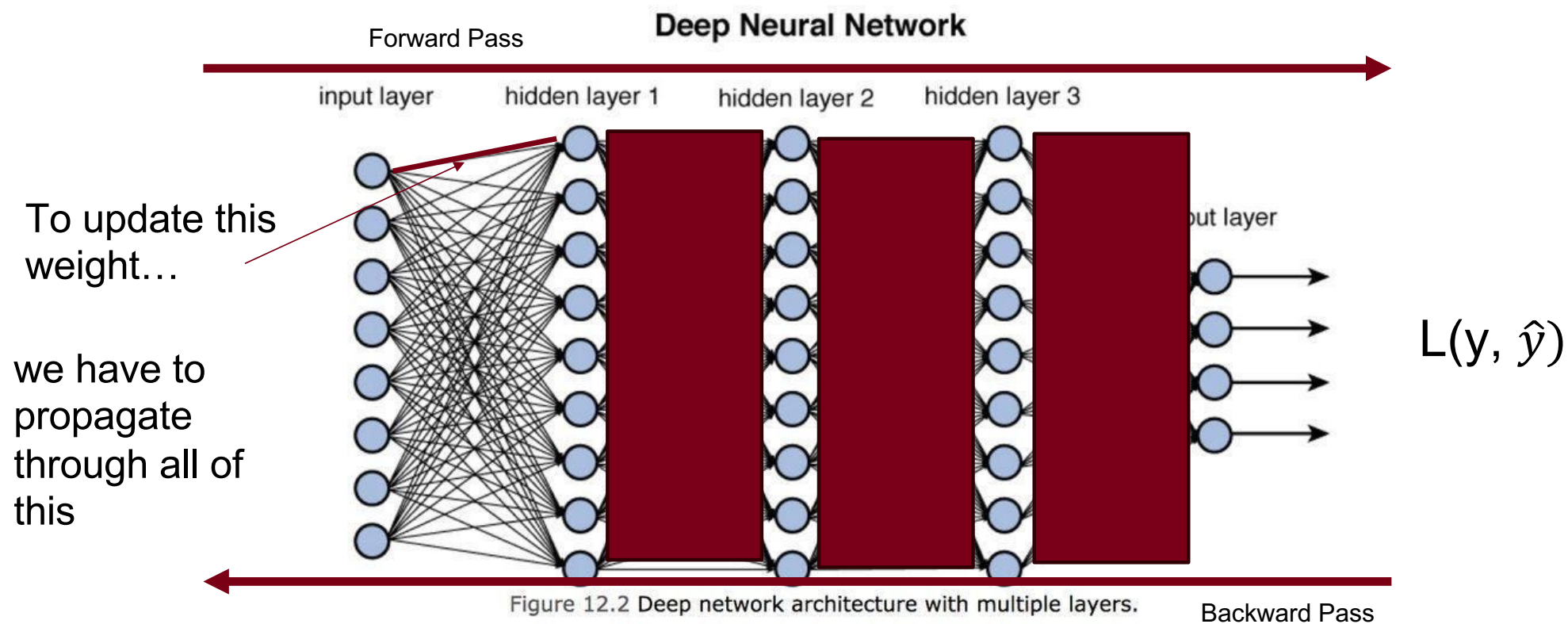


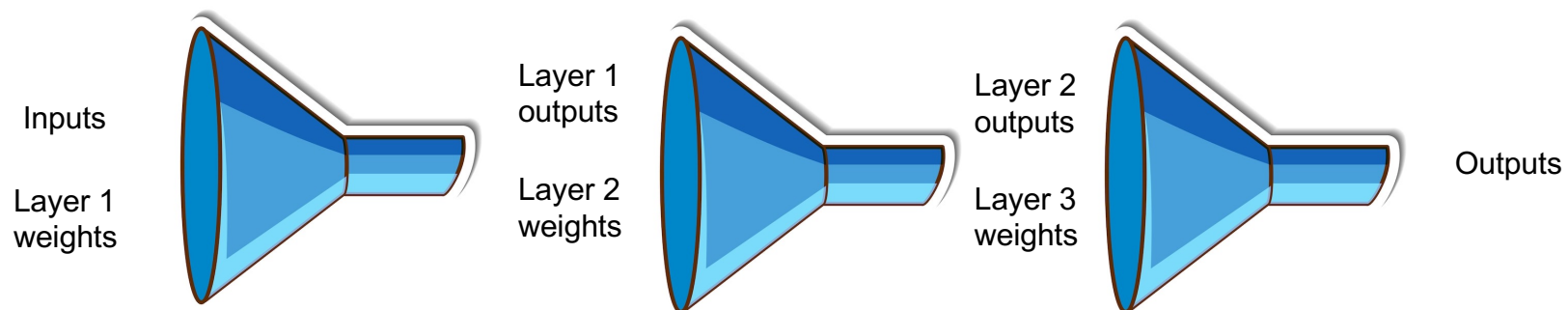
Figure 12.2 Deep network architecture with multiple layers.

# Analogy #1: A Game of Telephone



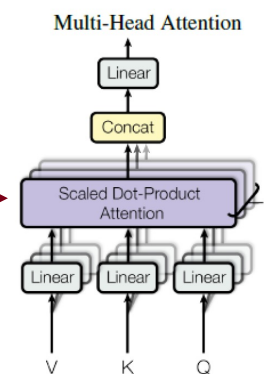
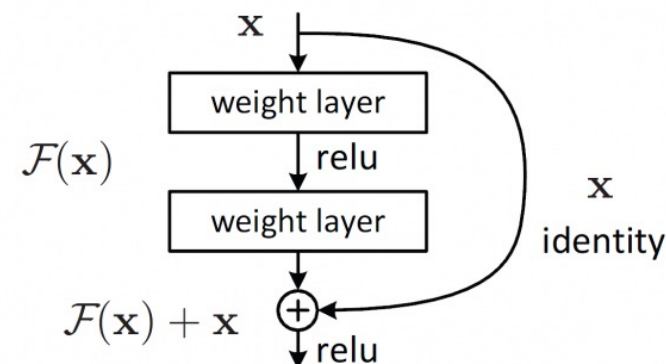


# Analogy #2: A funnel of information



# Linearization and Det-Bottlenecking

- ❑ Linearization → We need a better way to reduce the number of operations performed between our weights and our loss function (Residual connections)
- ❑ De-Bottlenecking → We need a better way to ensure we are not bottlenecking any representations into some channel which is too small to contain all the information we need (Attention mechanism → later)



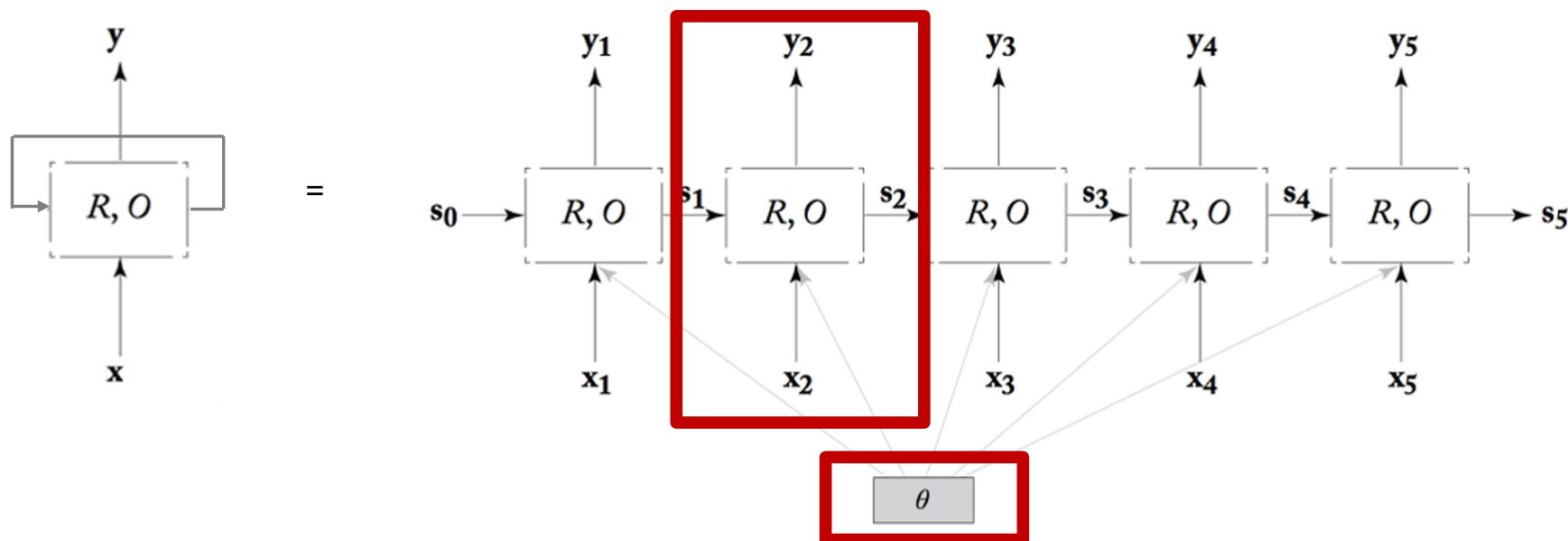
# Outline

- ❑ Linearization: A general heuristic for model improvement
- ❑ **Recurrent Neural Network (RNN)**
- ❑ Long Short-term Memory (LSTM)
- ❑ Implementation of RNN and LSTM using PyTorch
- ❑ Sequence-to-Sequence modeling

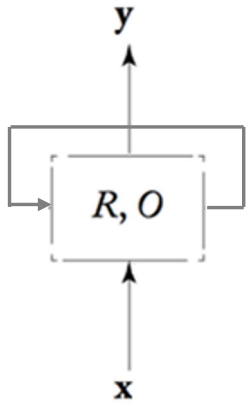


# Recurrent Neural Network (RNN)

RNN allow arbitrarily-sized conditioning contexts;  
condition on the **entire sequence history**.



# Recurrent Neural Network



Neural-LM:

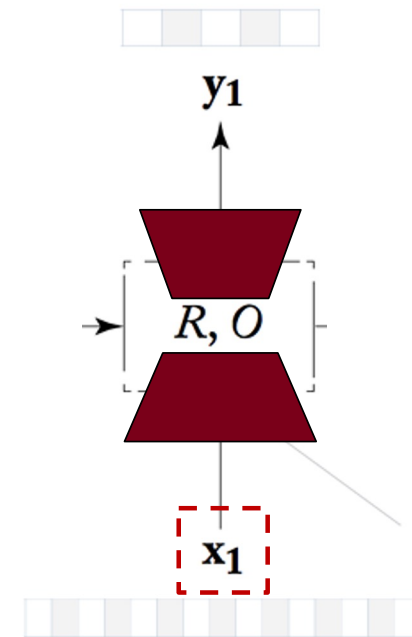
$$P(w) = P(w_i | w_{i-k} \dots w_{i-1}) = \text{softmax}(W \cdot \mathbf{h})$$

RNN:

$$P(w) = P(w_i | \text{context}) \\ = \text{softmax}(W \cdot \mathbf{h}_i)$$

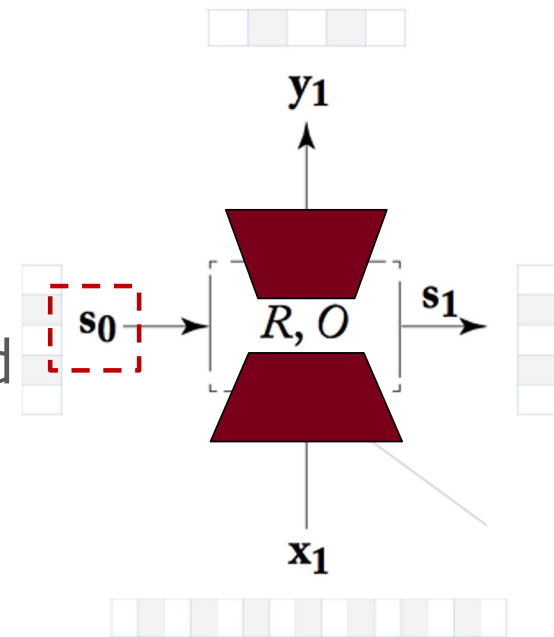
# Recurrent Neural Network

- Each time set has two inputs:
- $X_i$  (the observation at time step  $i$ ):
  - One-hot vector, feature vector, or distributed representation of input token at  $i$  step



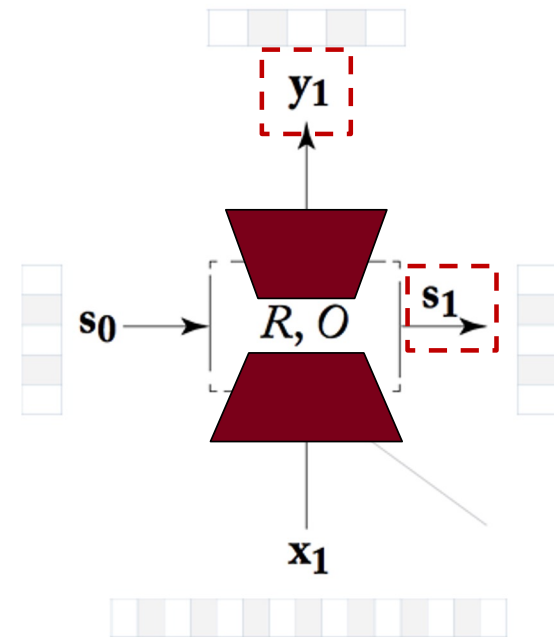
# Recurrent Neural Network

- Each time set has two inputs:
- $X_i$  (the observation at time step  $i$ ):
  - One-hot vector, feature vector, or distributed representation of input token at  $i$  step
- $S_{i-1}$  (the output of the previous state):
  - Base case:  $S_0 = 0$  vector



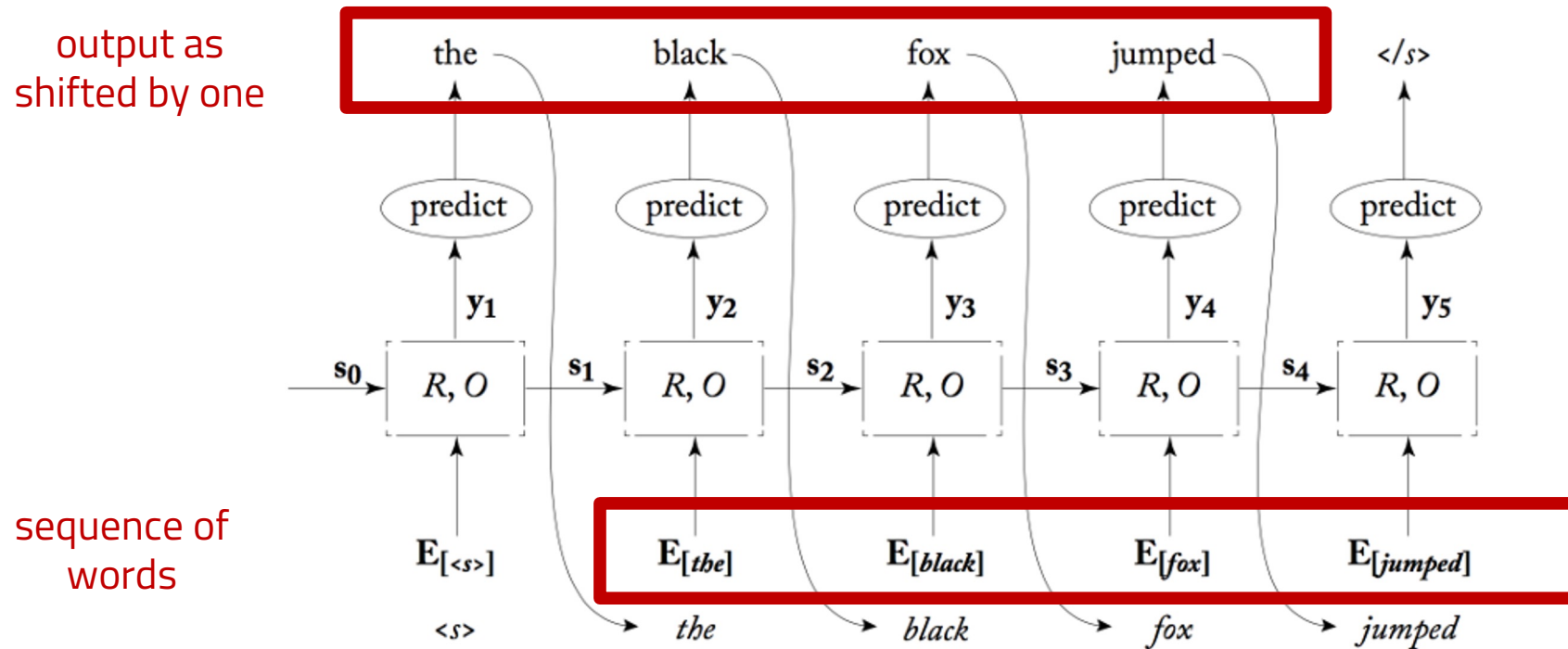
# Recurrent Neural Network

- Each time set has two outputs:
- $S_i = R(X_i, S_{i-1})$ 
  - R computes the **output state** as a function of the *current input* and *previous state*
- $y_i = O(S_i)$ 
  - O computes the **output** as a function of the *current output state*

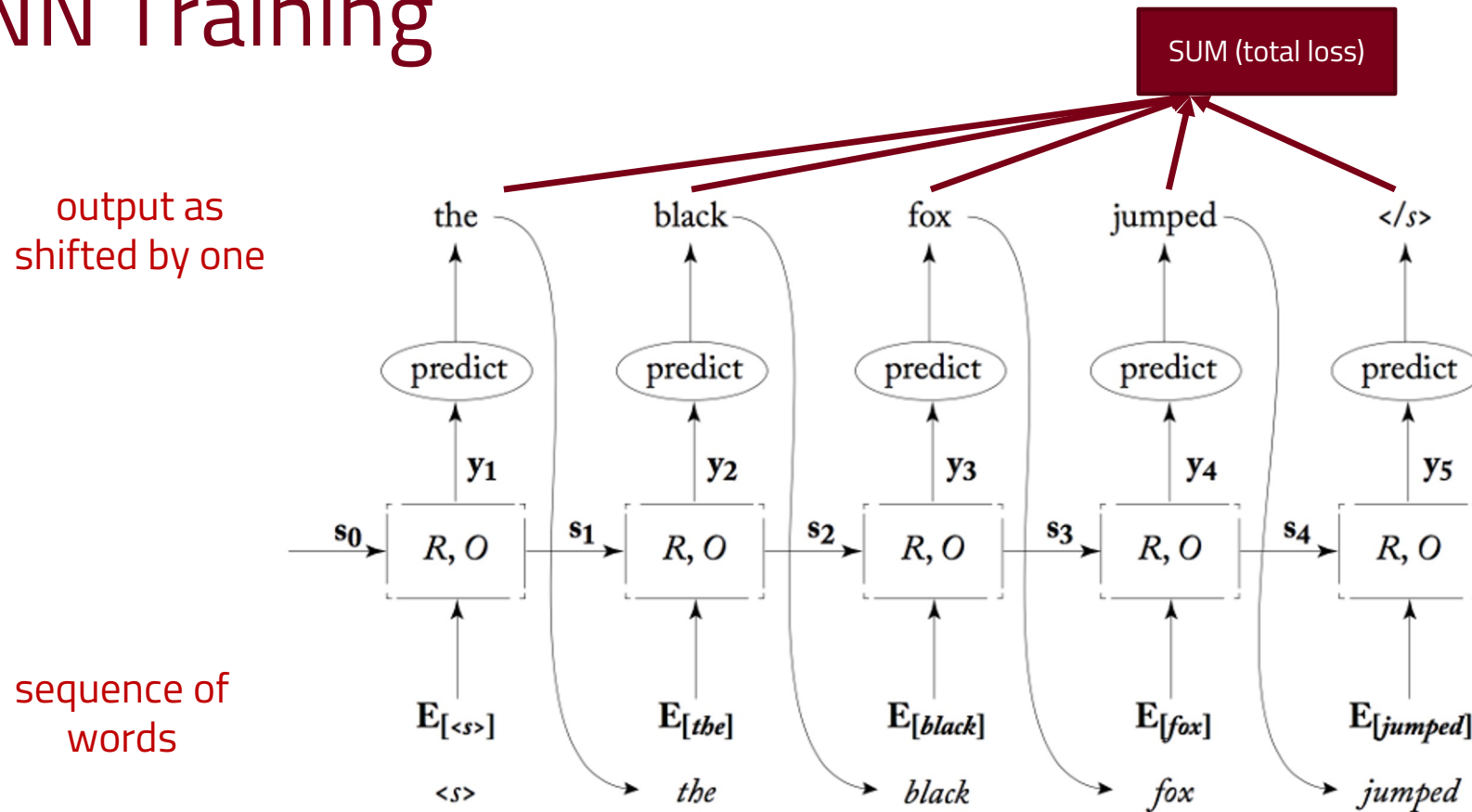




# RNN Training



# RNN Training

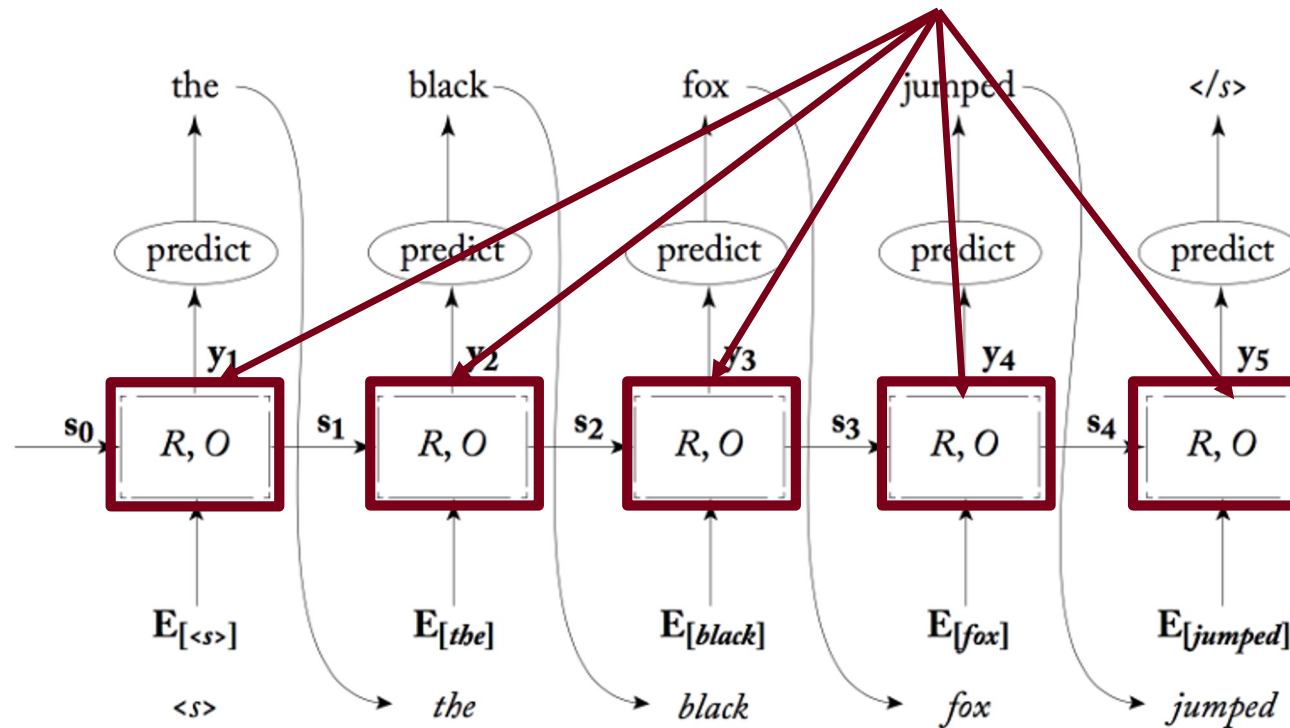


# RNN Training

Parameters are shared!  
Derivatives are accumulated.

output as  
shifted by one

sequence of  
words

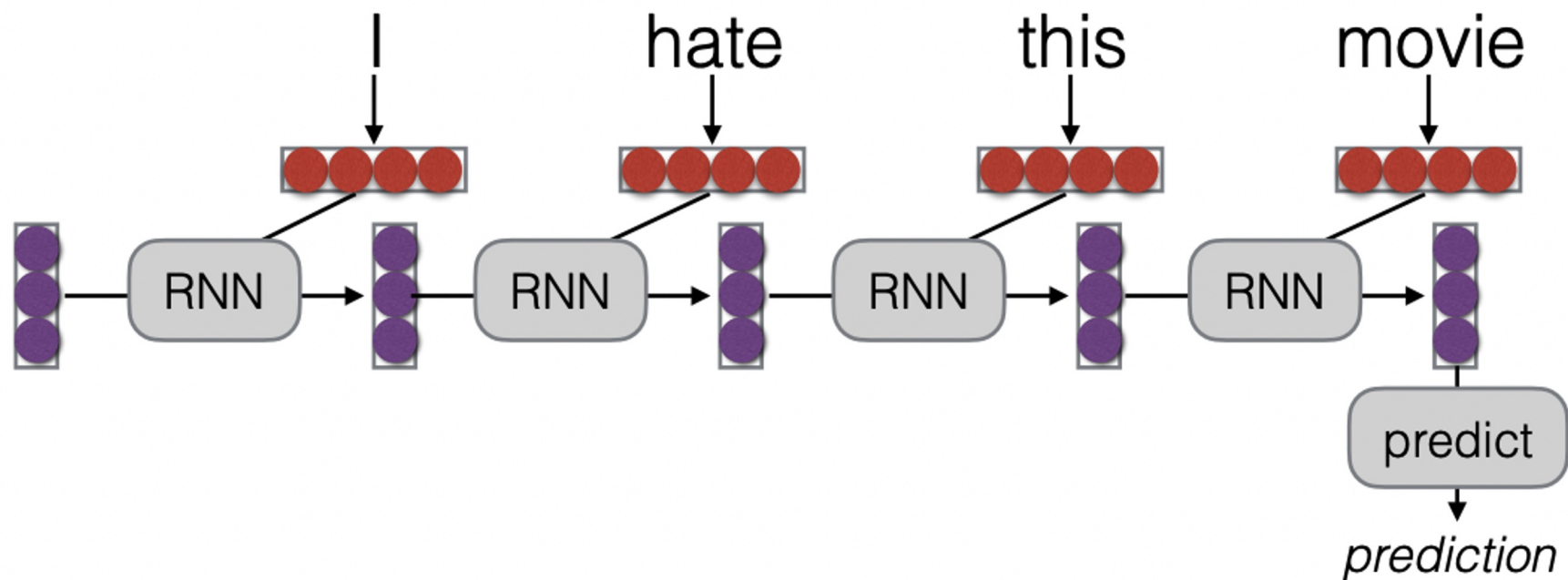


# What can RNNs do?

- ❑ Represent a sentence
  - Read whole sentence, make a prediction
- ❑ Represent a context within a sentence
  - Read context up until that point

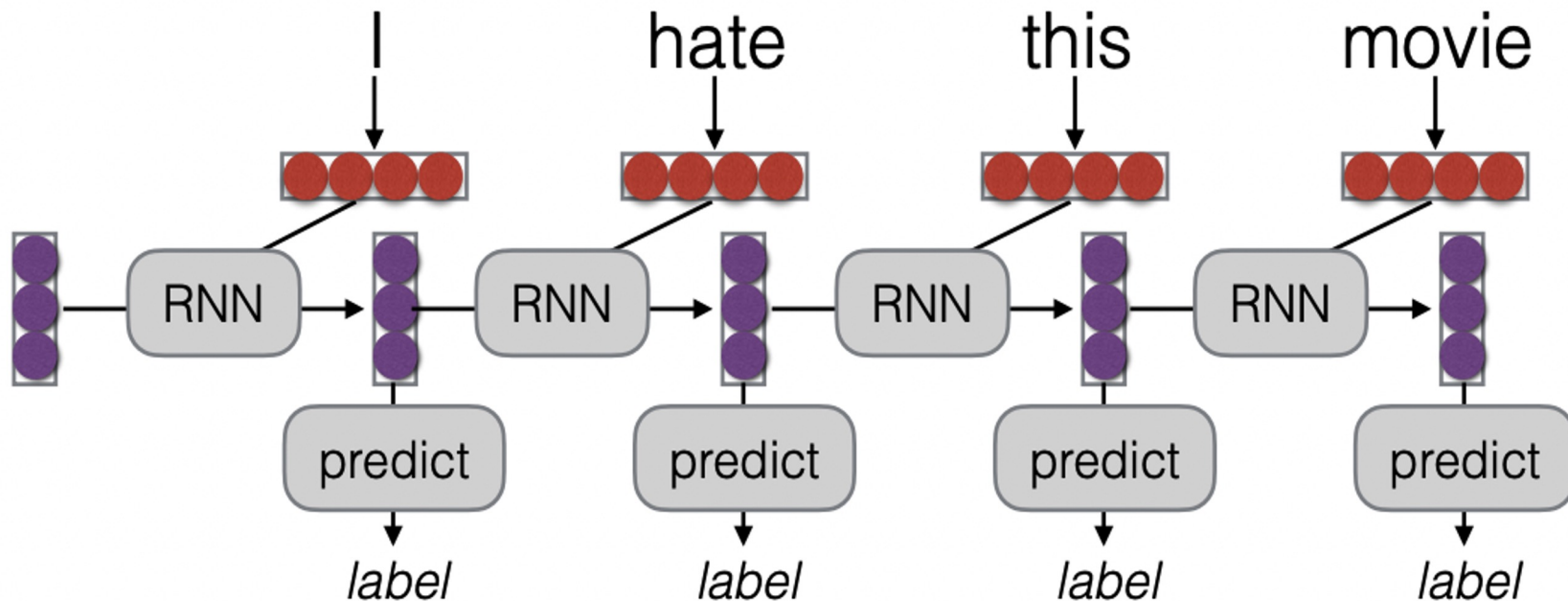
# Representing Sentences

- ❑ Sentence classification
- ❑ Conditioned generation



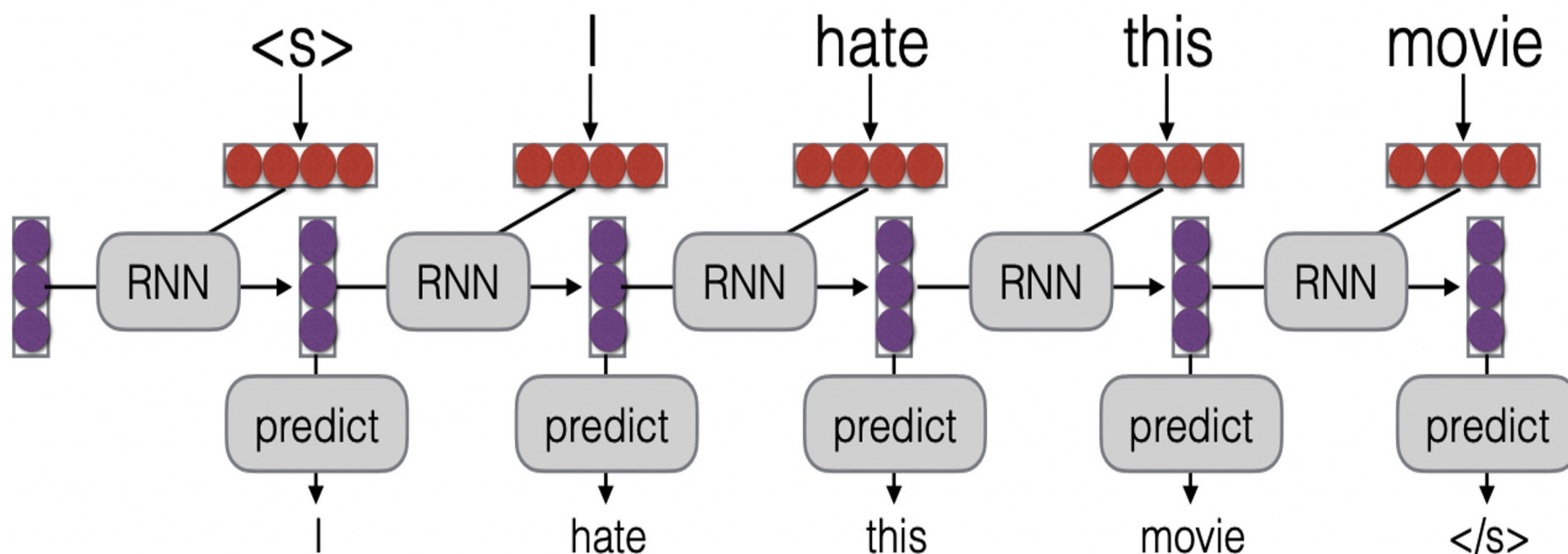
# Representing Context within Sentence

- Tagging
- Language modeling

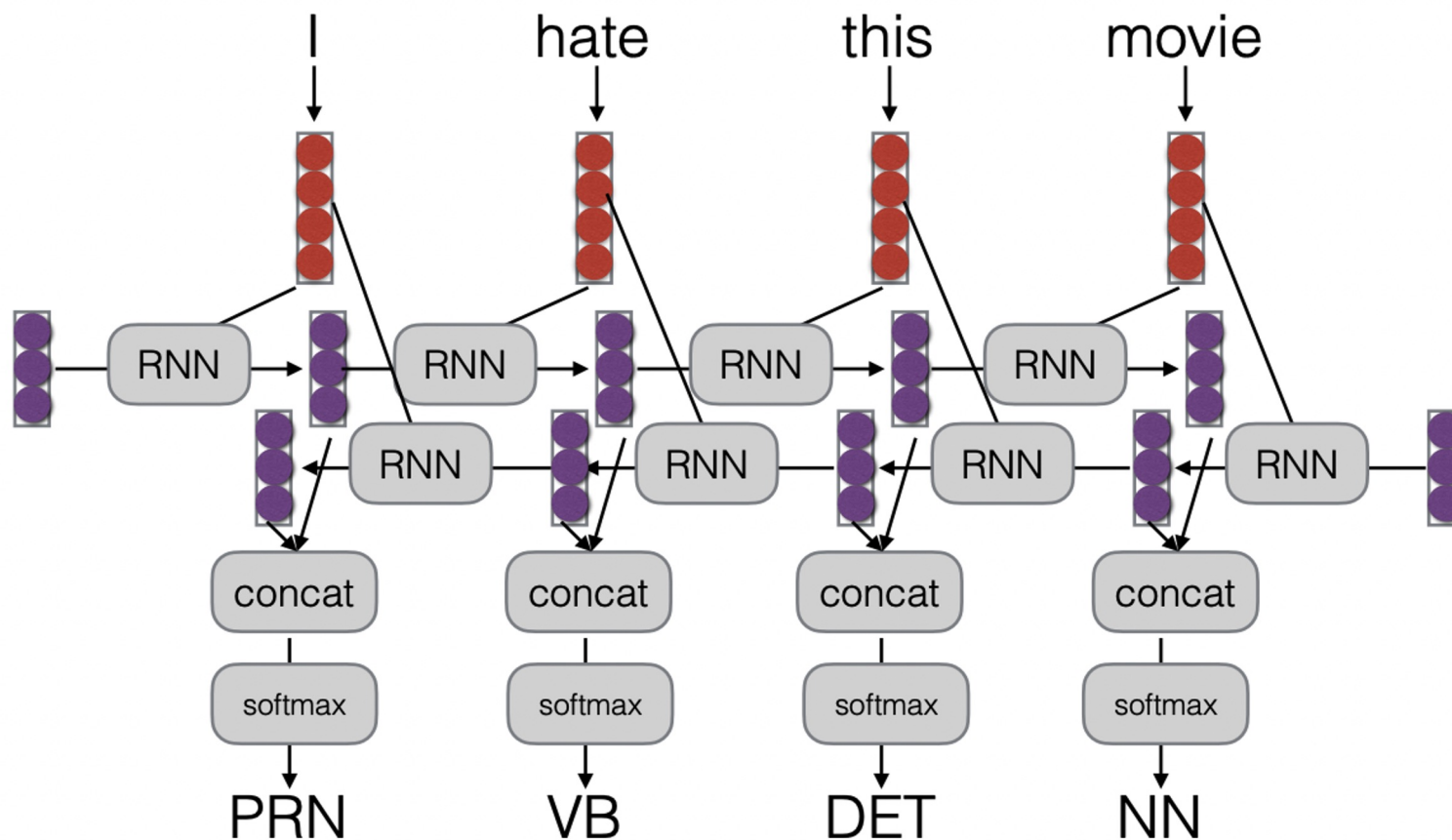


# e.g., Language Modeling

□ Language modeling is like a tagging task, where each tag is the next word!



## e.g., POS Tagging with Bi-RNNs





# Outline

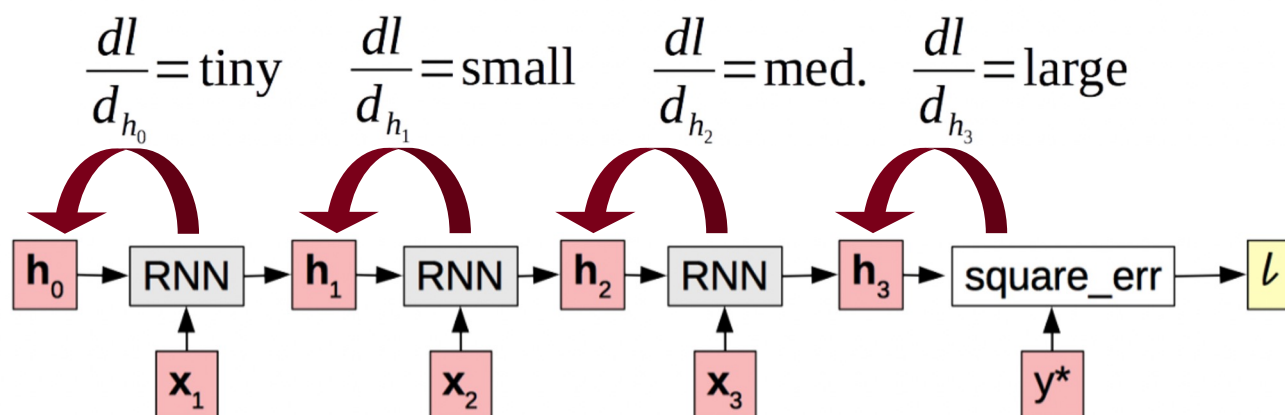
- ❑ Linearization: A general heuristic for model improvement
- ❑ Recurrent Neural Network (RNN)
- ❑ **Long Short-term Memory (LSTM)**
- ❑ Implementation of RNN and LSTM using PyTorch
- ❑ Sequence-to-Sequence modeling



# Vanishing Gradient



- Gradients decrease as they get pushed back



- Why? "Squashed" by non-linearities or small weights in matrices

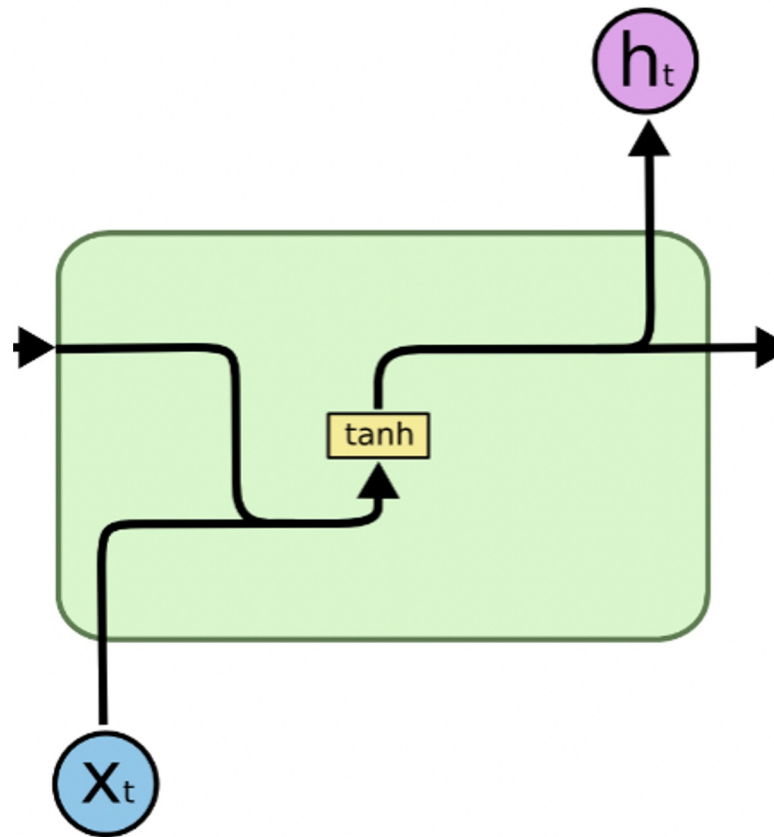
# A Solution: Long Short-term Memory (LSTM)

(Hochreiter and Schmidhuber 1997)

- ❑ Make **additive connections** between time steps
- ❑ Addition does not modify the gradient, no vanishing
- ❑ **Gates** to control the information flow

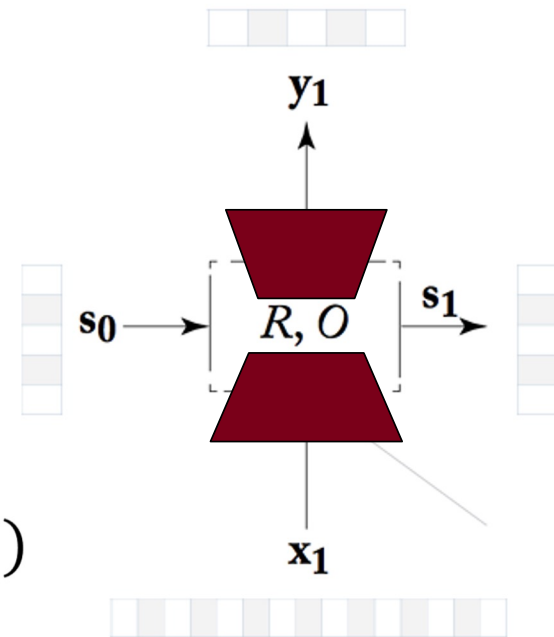


# RNN Structure



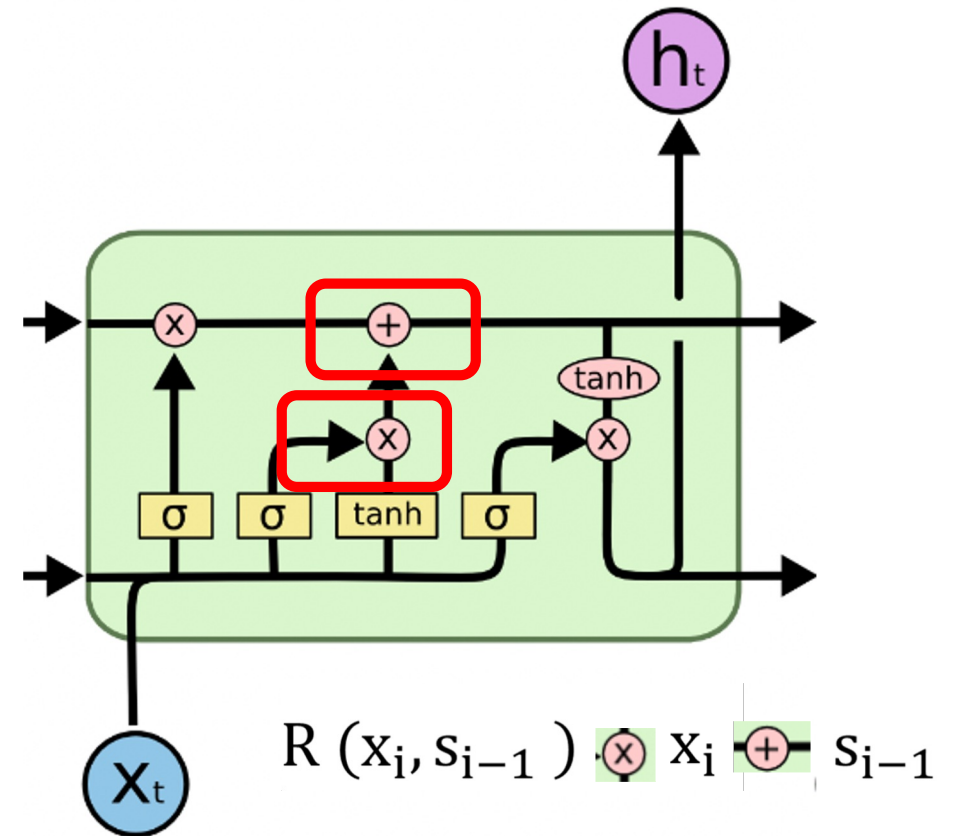
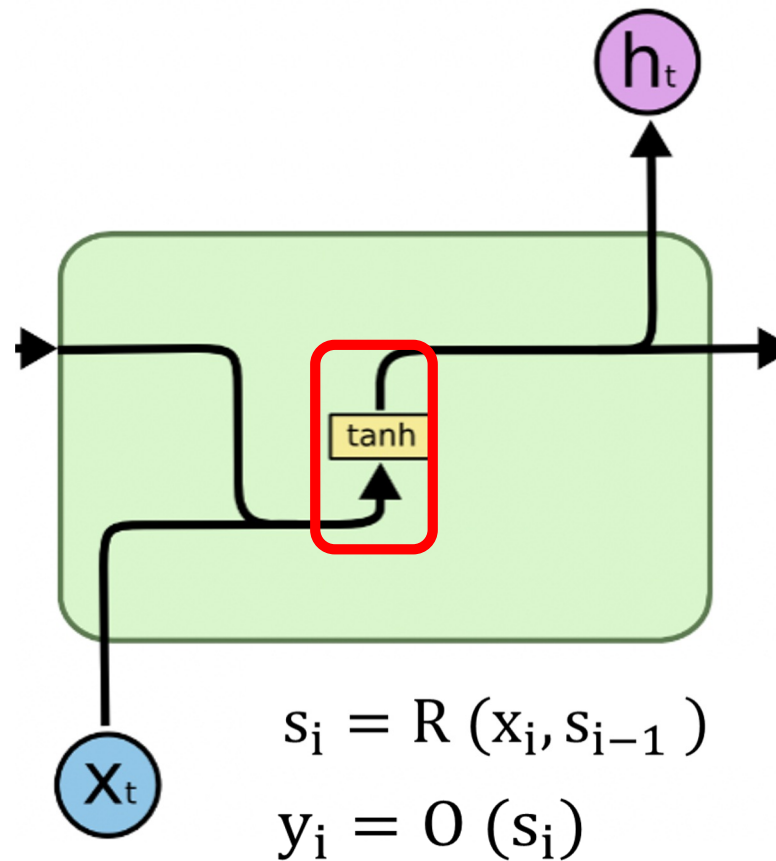
$$s_i = R(x_i, s_{i-1})$$

$$y_i = O(s_i)$$



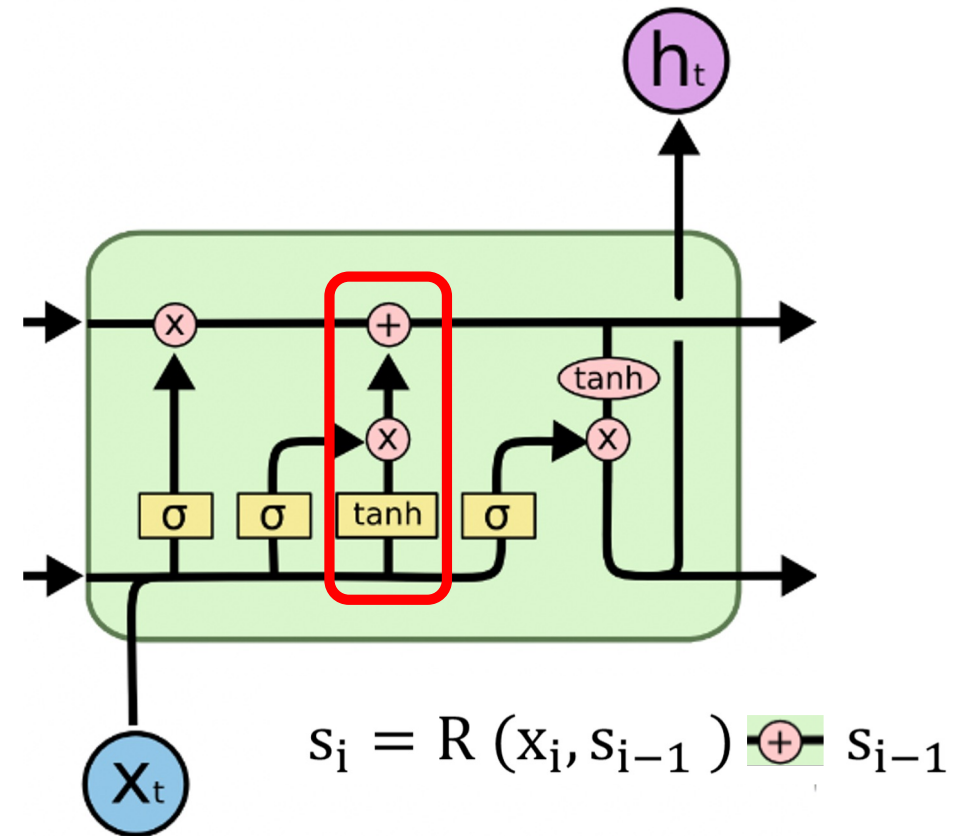
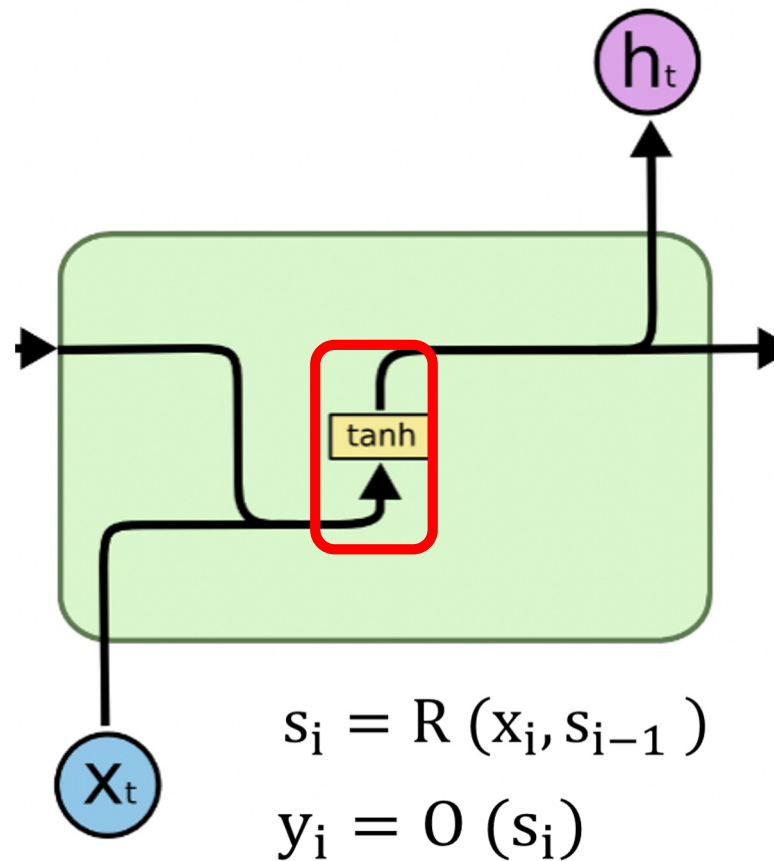
<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

# RNN vs LSTM Structure



<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

# RNN vs LSTM Structure



<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

# LSTM Structure

- ❑ **Forget gate:** what value do we try to add/forget to the memory cell?
- ❑ **Input gate:** how much of the update do we allow to go through?
- ❑ **Output gate:** how much of the cell do we reflect in the next state?

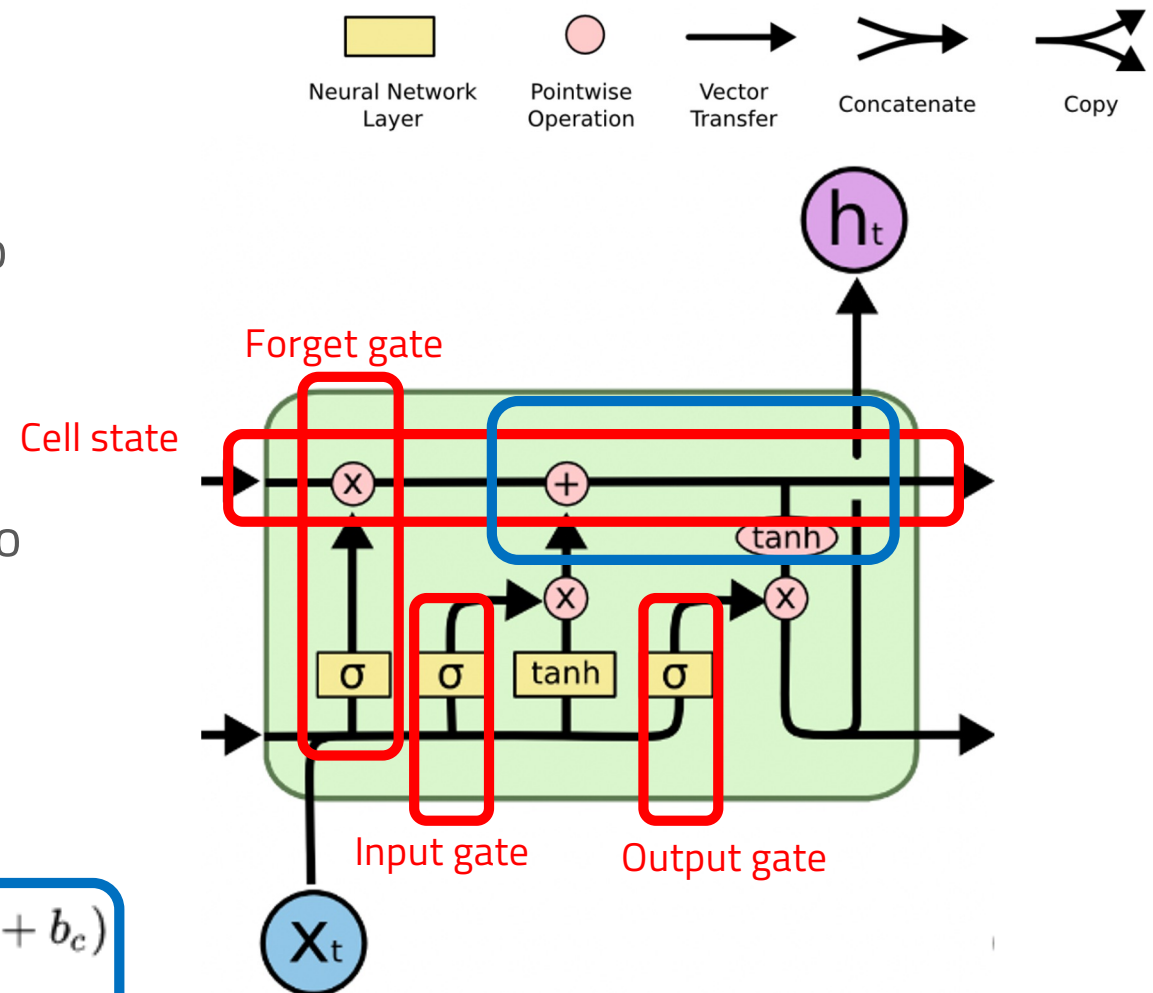
$$f_t = \sigma_a(W_f x_t + U_f h_{t-1} + b_f)$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$$

$$o_t = \sigma_a(W_o x_t + U_o h_{t-1} + b_o)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$$

$$h_t = o_t \circ \sigma_h(c_t)$$



<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

# LSTM variant: Gated Recurrent Unit (GRU)

(Cho et al., 2014)

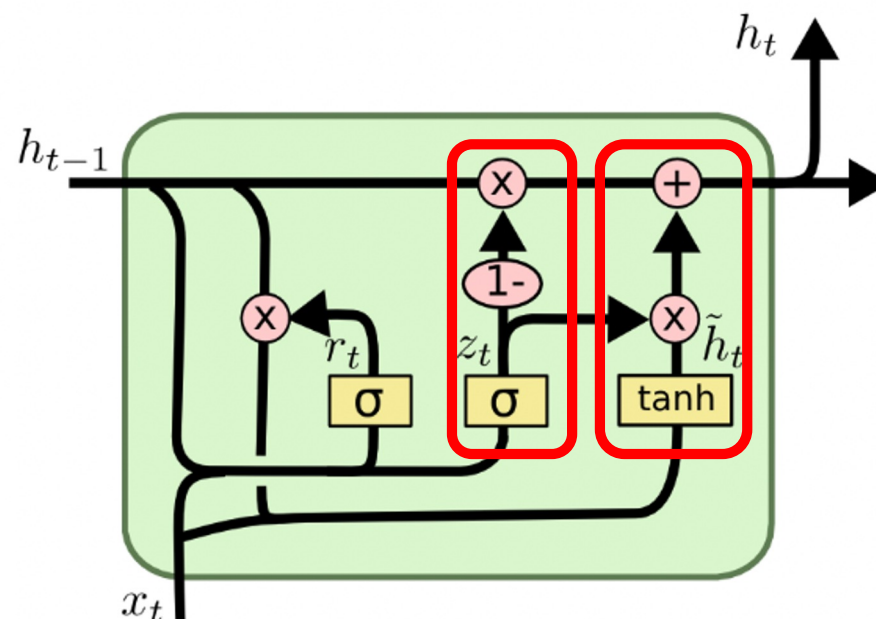
- Combines the forget and input gates into a single “update gate.”
- Merges the cell state and hidden state
- And, other small changes

$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$

$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$

$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \sigma_h(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h)$$

Additive or Non-linear

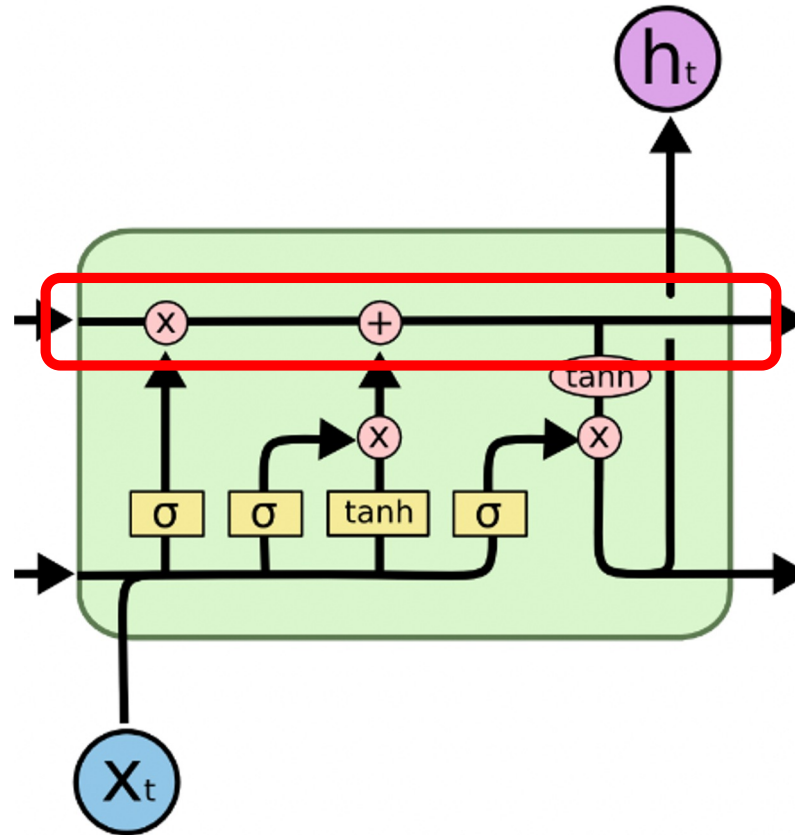


<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>



# Most Important Takeaway

- ❑ The Cell State is an information highway
- ❑ Gradient can flow over this without nearly as many issues of vanishing/exploding gradients that we saw in RNNs
- ❑ We are doing a better job at reducing the 'distance' between our loss function and each individual parameter



# A Solution: Long Short-term Memory (LSTM)

(Hochreiter and Schmidhuber 1997)

- ❑ Make **additive connections** between time steps
- ❑ Addition does not modify the gradient, no vanishing
- ❑ **Gates** to control the information flow



# Outline

- ❑ Linearization: A general heuristic for model improvement
- ❑ Recurrent Neural Network (RNN)
- ❑ Long Short-term Memory (LSTM)
- ❑ **Implementation of RNN and LSTM using PyTorch**
- ❑ Sequence-to-Sequence modeling
- ❑ Teaser: Transformer-based LMs
- ❑ Why language models are useful?



**class RNN(nn.Module):**

```
def __init__(self, input_size: int, hidden_size: int, output_size: int) -> None:
```

```
    super().__init__()
```

```
    ...
```

```
    self.i2h = nn.Linear(input_size, hidden_size, bias=False)
```

```
    self.h2h = nn.Linear(hidden_size, hidden_size)
```

```
    self.h2o = nn.Linear(hidden_size, output_size)
```

```
def forward(self, x, hidden_state):
```

```
    x = self.i2h(x)
```

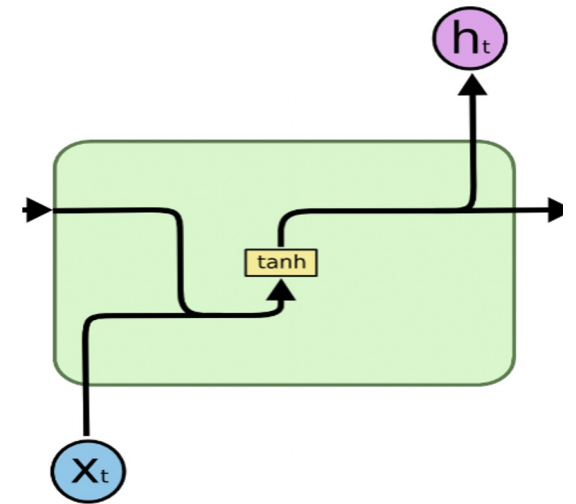
```
    hidden_state = self.h2h(hidden_state)
```

```
    hidden_state = torch.tanh(x + hidden_state)
```

```
    out = self.h2o(hidden_state)
```

```
    return out, hidden_state
```

$$\left. \begin{array}{l} s_i = R(x_i, s_{i-1}) \\ y_i = O(s_i) \end{array} \right\}$$



```
class RNN(nn.Module):
```

```
    def __init__(self, input_size, output_size, hidden_dim, n_layers):
        super(RNN, self).__init__()
```

```
    ...
```

```
    self.rnn = nn.RNN(input_size, hidden_dim, n_layers, batch_first=True)
```

```
    self.fc = nn.Linear(hidden_dim, output_size)
```

```
    def forward(self, x, hidden):
```

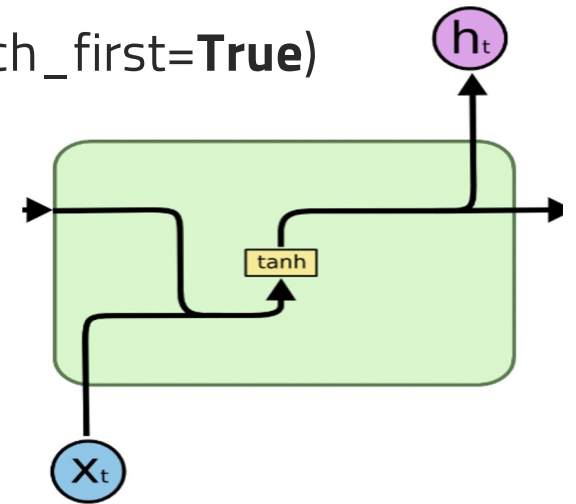
```
        r_out, hidden = self.rnn(x, hidden)
```

```
        r_out = r_out.view(-1, self.hidden_dim)
```

```
        return self.fc(r_out), hidden
```

$$s_i = R(x_i, s_{i-1})$$

$$y_i = O(s_i)$$



```
# x (batch_size, seq_length, input_size)
# hidden (n_layers, batch_size, hidden_dim)
# r_out (batch_size, time_step, hidden_size)
```

```
class LSTM (nn.Module):
```

```
    def __init__(self, num_classes, input_size, hidden_size, num_layers,  
seq_length):
```

```
        super(LSTM1, self).__init__()
```

```
    ...
```

```
        self.lstm = nn.LSTM(input_size=input_size, hidden_size=hidden_size,  
num_layers=num_layers, batch_first=True)
```

```
        self.fc = nn.Linear(hidden_size, num_classes)
```

```
        self.relu = nn.ReLU()
```

```
def forward(self,x):
```

```
    h_0 = Variable(torch.zeros(self.num_layers, x.size(0), self.hidden_size))
```

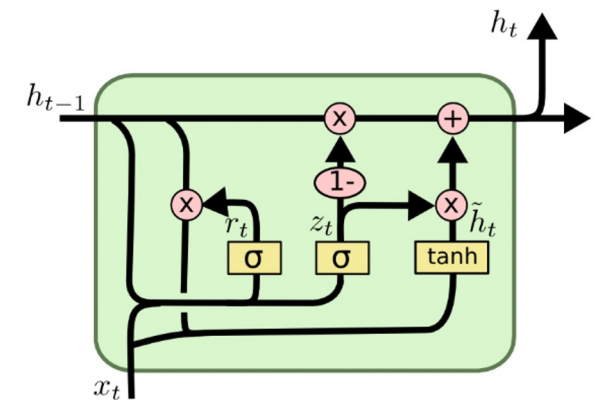
```
    c_0 = Variable(torch.zeros(self.num_layers, x.size(0), self.hidden_size))
```

```
    output, (hn, cn) = self.lstm(x, (h_0, c_0))
```

```
    hn = hn.view(-1, self.hidden_size)
```

```
    return self.fc (self.relu(hn))
```

PyTorch



$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$$

$$h_t = o_t \circ \sigma_h(c_t)$$

# Outline

- ❑ Linearization: A general heuristic for model improvement
- ❑ Recurrent Neural Network (RNN)
- ❑ Long Short-term Memory (LSTM)
- ❑ Implementation of RNN and LSTM using PyTorch
- ❑ **Sequence-to-Sequence modeling**

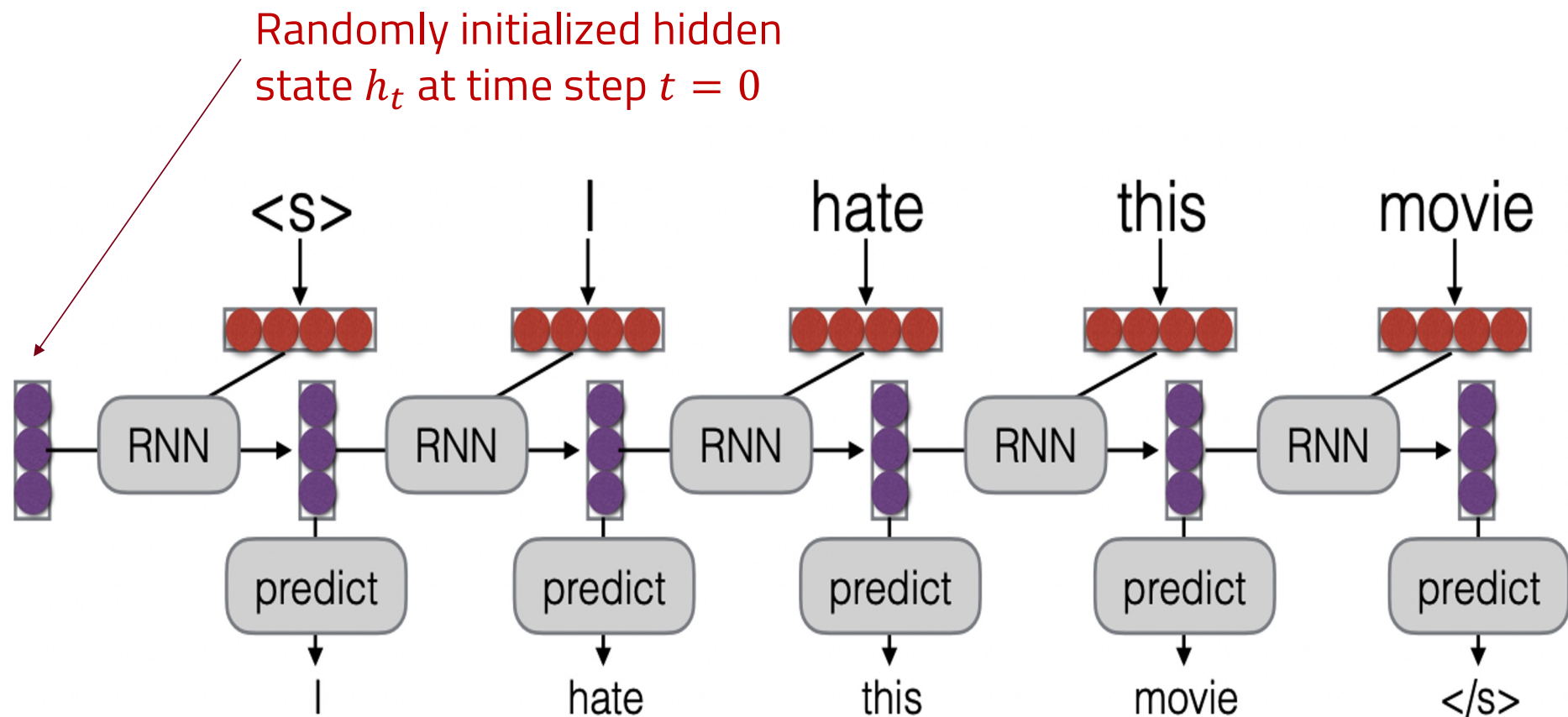


# Connecting RNN to RNN for sequence-to-sequence (seq2seq) modeling



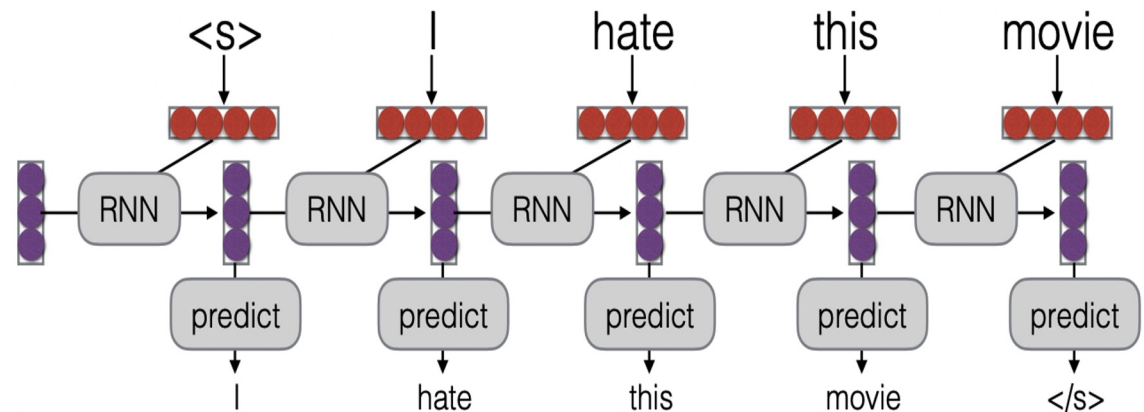


# RNN (decoder) for language modeling



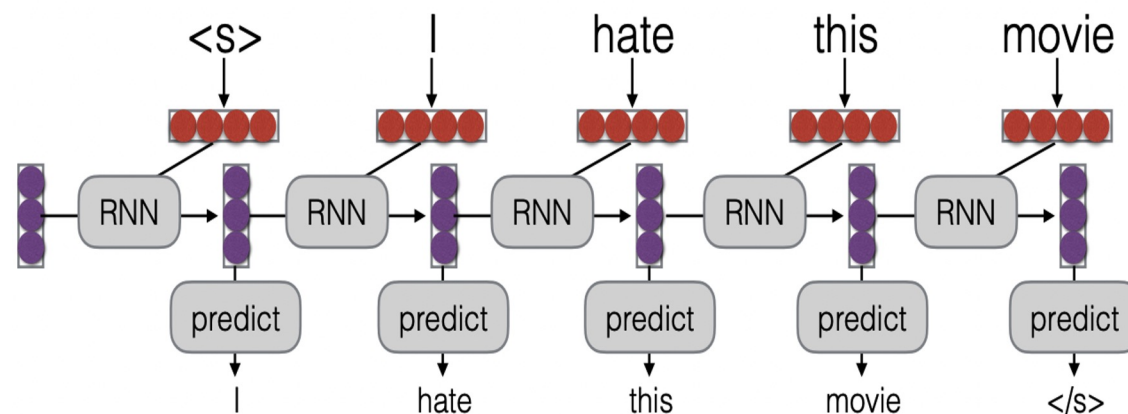
# RNN (decoder) for language modeling

What if we encode some specific context, instead of random state?



# RNN (encoder) - RNN (decoder) for machine translation

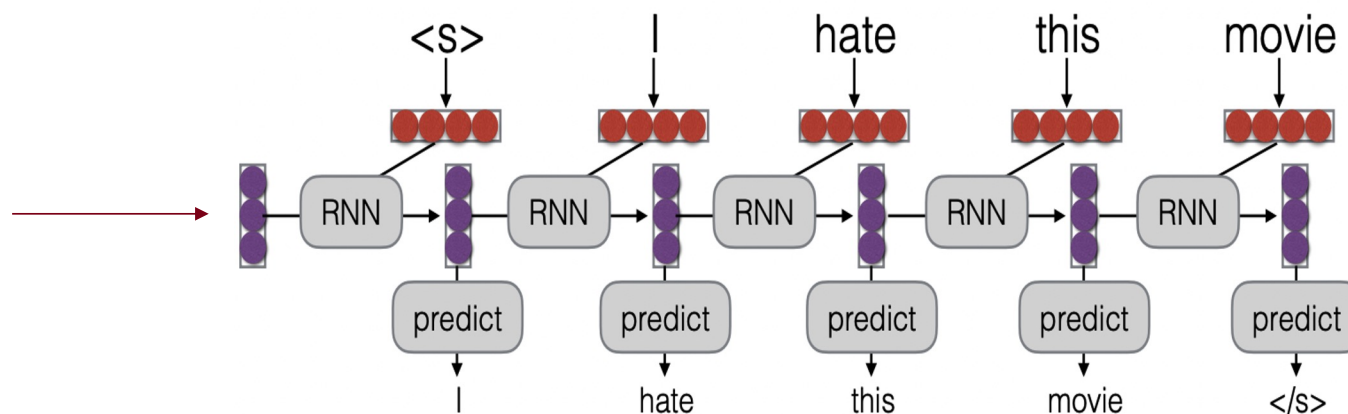
“나는 이 영화가 싫어요”  
“Odio esta película”



# RNN (encoder) - RNN (decoder) for dialogue generation

“나는 이 영화가 싫어요”  
“Odio esta película”

“what do you think about  
*Avengers: Endgame*?”

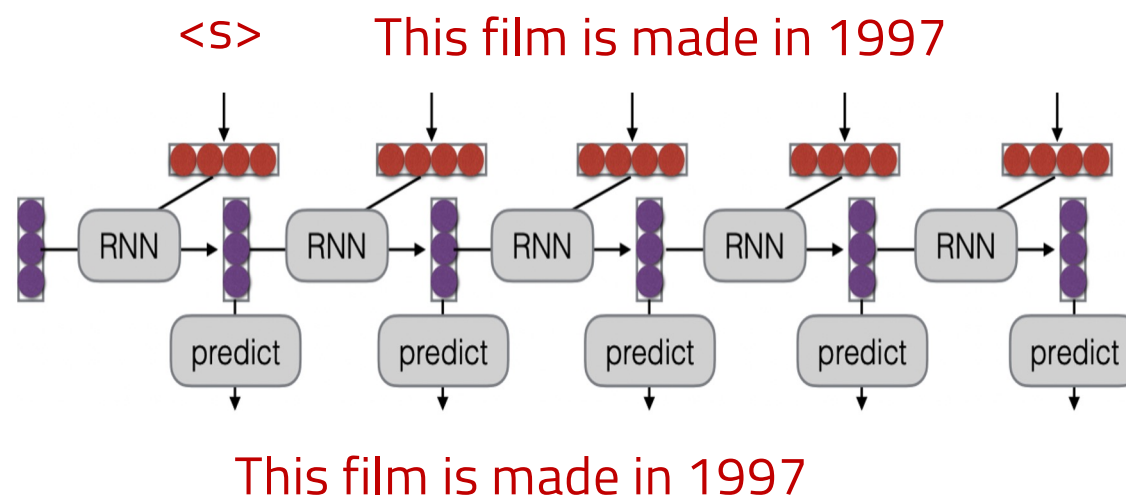


# RNN (encoder) - RNN (decoder) for question answering

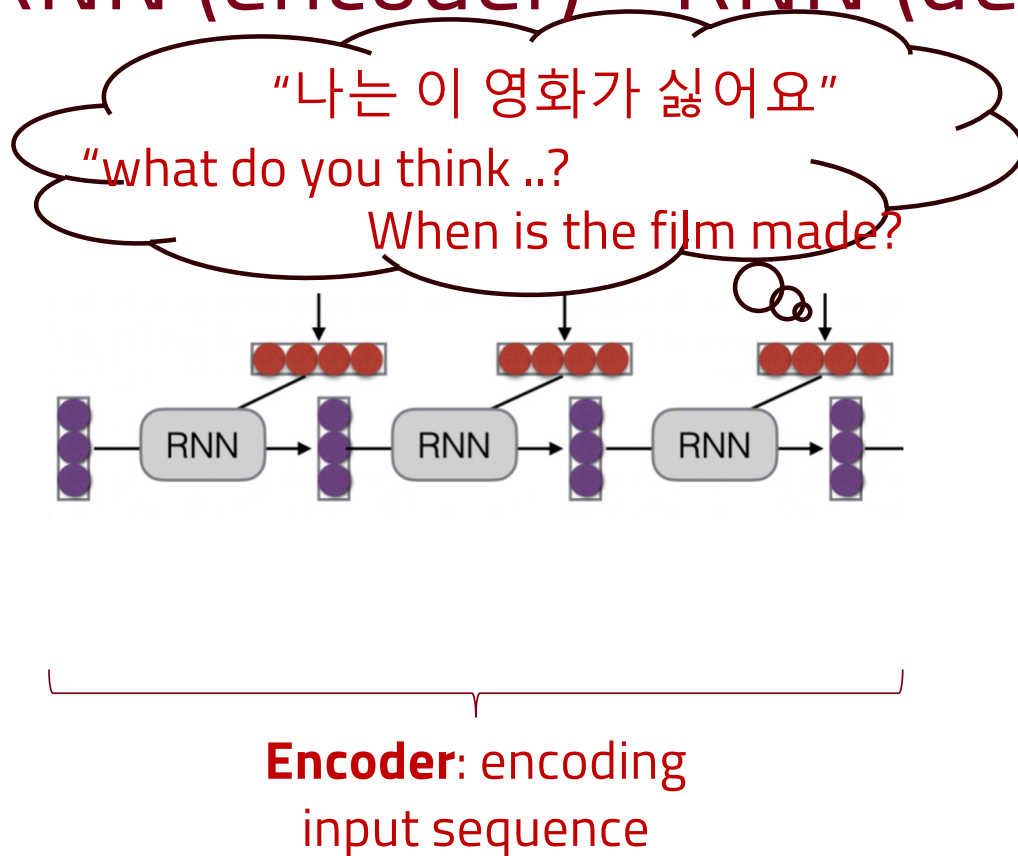
“나는 이 영화가 싫어요”  
“Odio esta película”

“what do you think about  
*Avengers: Endgame*?”

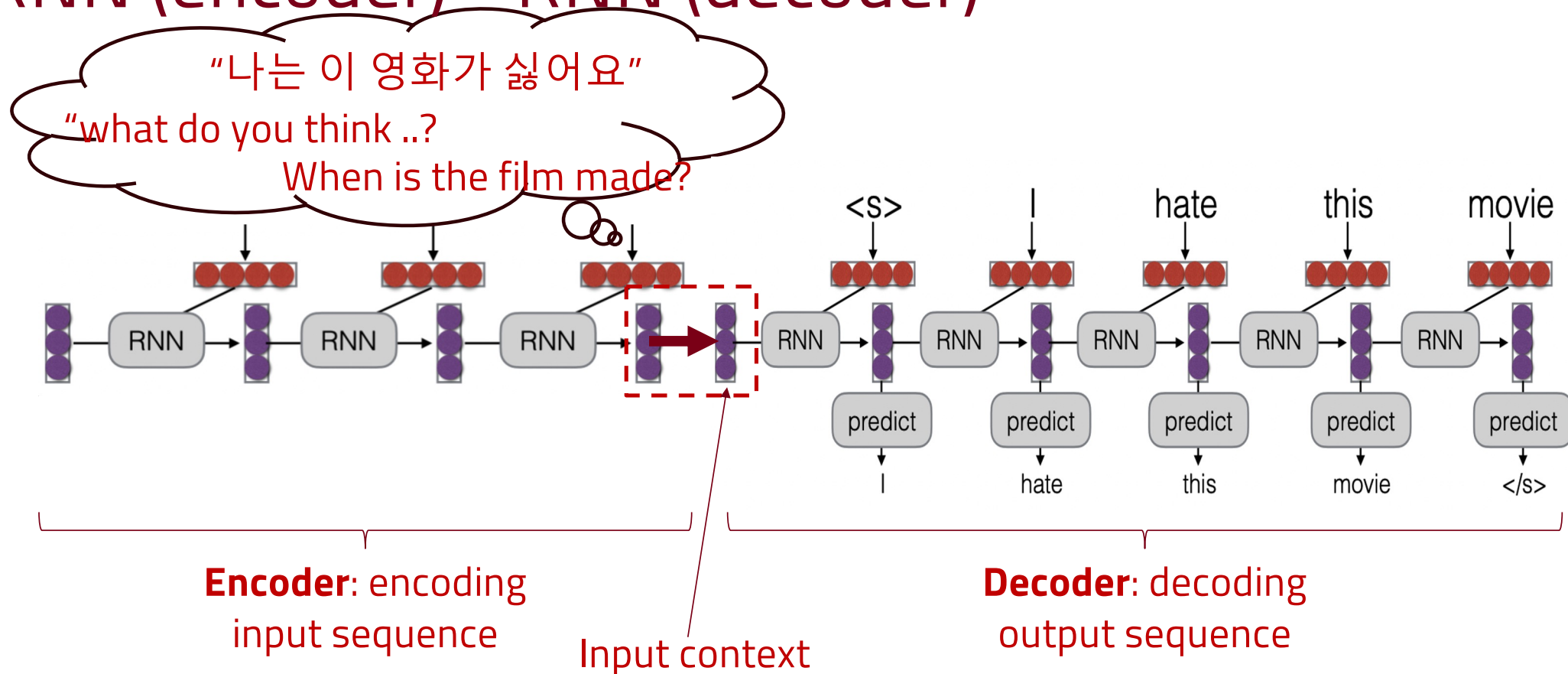
When is the film made?



# Sequence-to-sequence modeling using RNN (encoder) - RNN (decoder)



# Sequence-to-sequence modeling using RNN (encoder) - RNN (decoder)

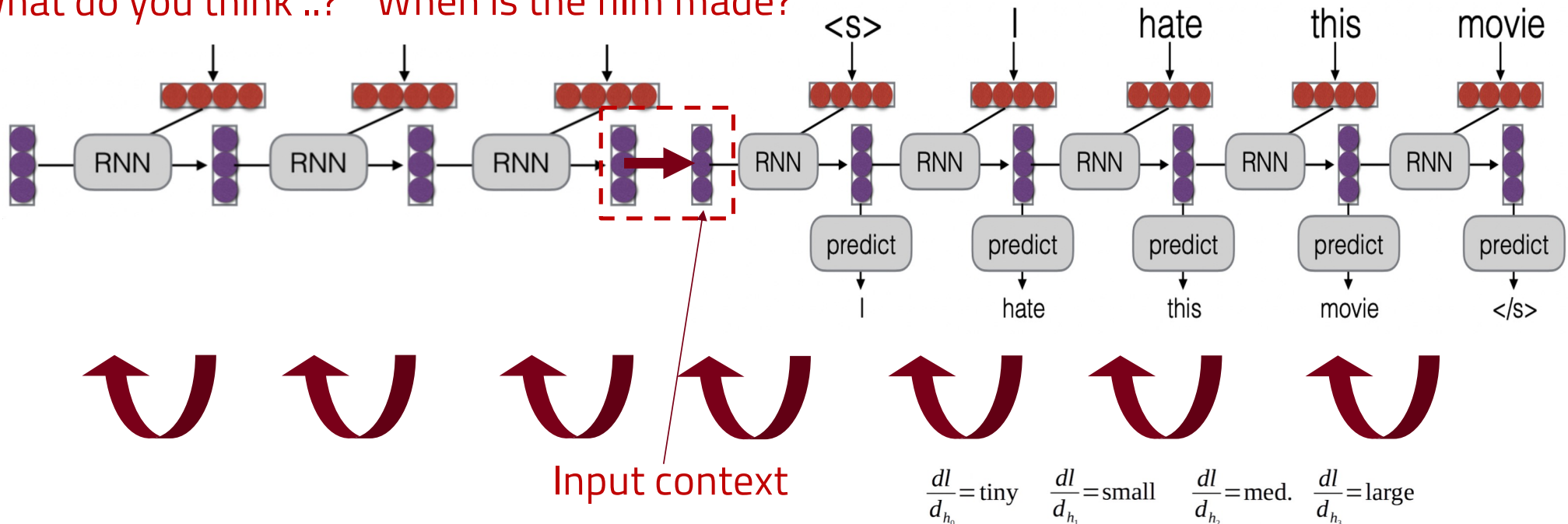


# Problem: forgetting input context as input gets longer



“나는 이 영화가 싫어요”

“what do you think ..? When is the film made?”



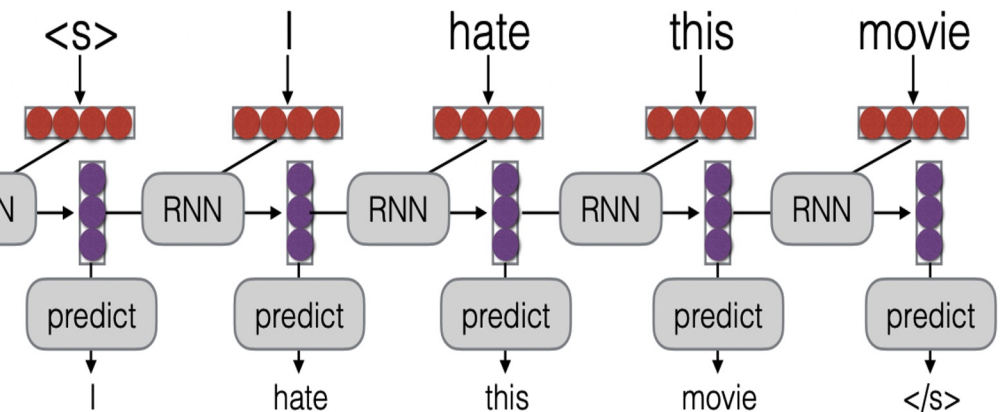
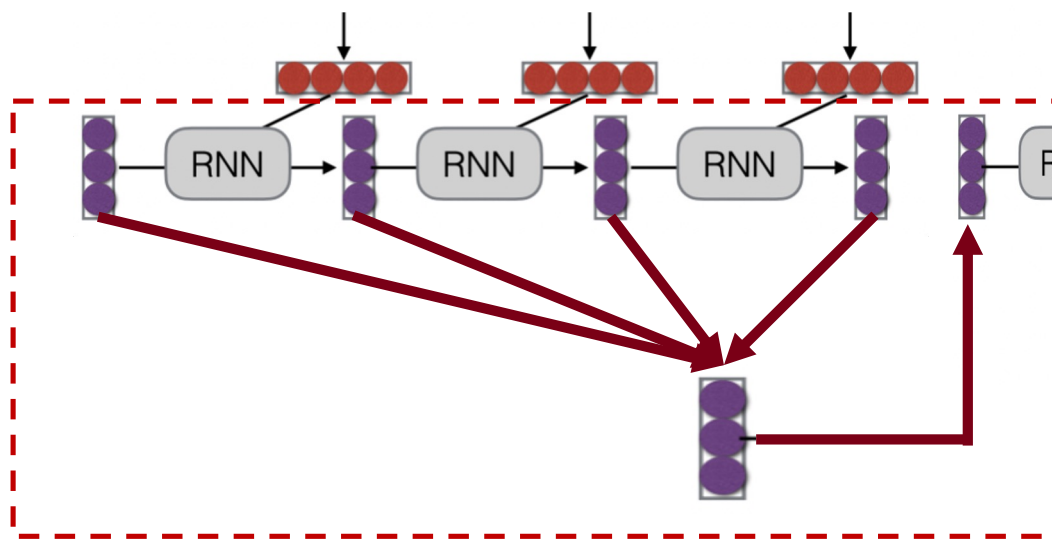


# Solution (teaser): Seq2seq with attention



“나는 이 영화가 싫어요”

“what do you think ..? When is the film made?”



Attention layer = Input context  
attended on all previous context  
(will be covered more in Transformer)

