CSCI 5541: Natural Language Processing

Lecture 7: Langage Models: RNN, LSTM, and Seq2Seq

Dongyeop Kang (DK), University of Minnesota

dongyeop@umn.edu | twitter.com/dongyeopkang | dykang.github.io

computer science & engineering

MINNESOTA NLP EST. 2021

University of Minnesota
Driven to Discover®

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.

Recap

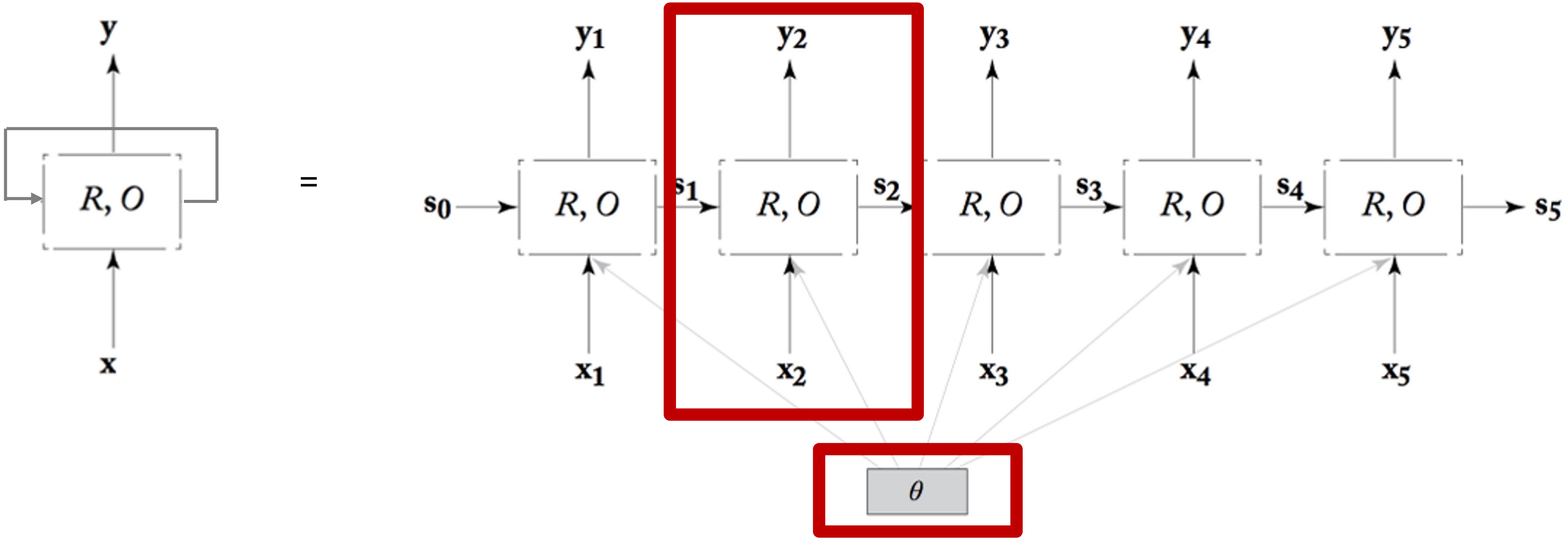
- ❑ Ngram LM $\longrightarrow$ Neural LM : sparsity
- ❑ Neural LM $\longrightarrow$ RNN LM : input size is not scalable
- ❑ RNN LM $\longrightarrow$ LSTM LM:
- ❑ LSTM LM $\longrightarrow$ Transformer :

Outline

- ❑ 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?

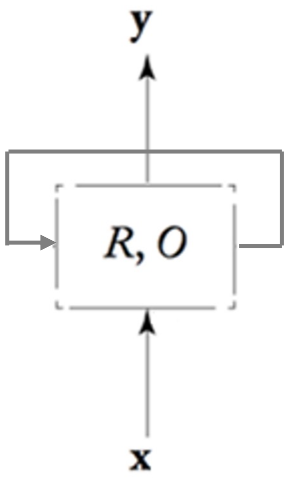
Recurrent Neural Network (RNN)

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



Goldberg 2017

Recurrent Neural Network



Neural-LM: $P(w) = P(w_i | w_{i-k} \ldots w_{i-1}) = softmax\,(W \cdot \boldsymbol{h})$

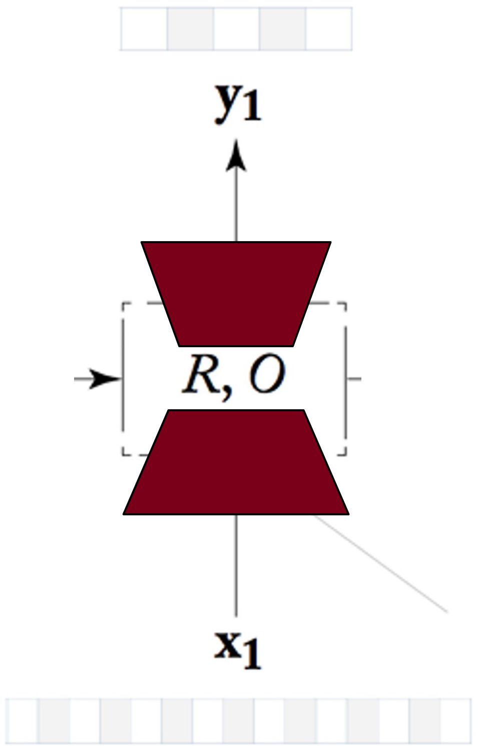
RNN: $P(w) = P(w_i | context)$
$= softmax\,(W \cdot \boldsymbol{h_i})$

Goldberg 2017

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

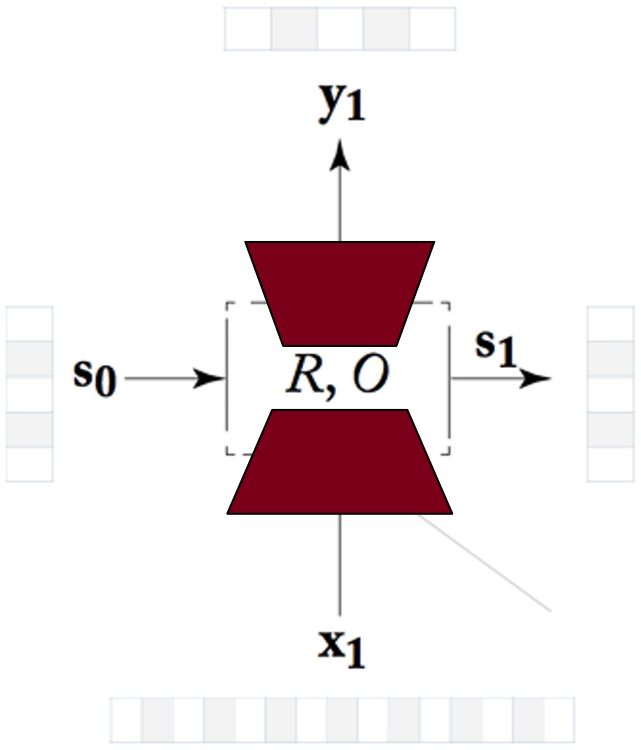


Recurrent Neural Network

- Each time set has two inputs:

- X_i (the observation at time step i):
  - One-hot vector, feature vector, or distribute representation of input token at i step

- S_{i-1} (the output of the previous state):
  - Base case: $S_0 = 0$ vector

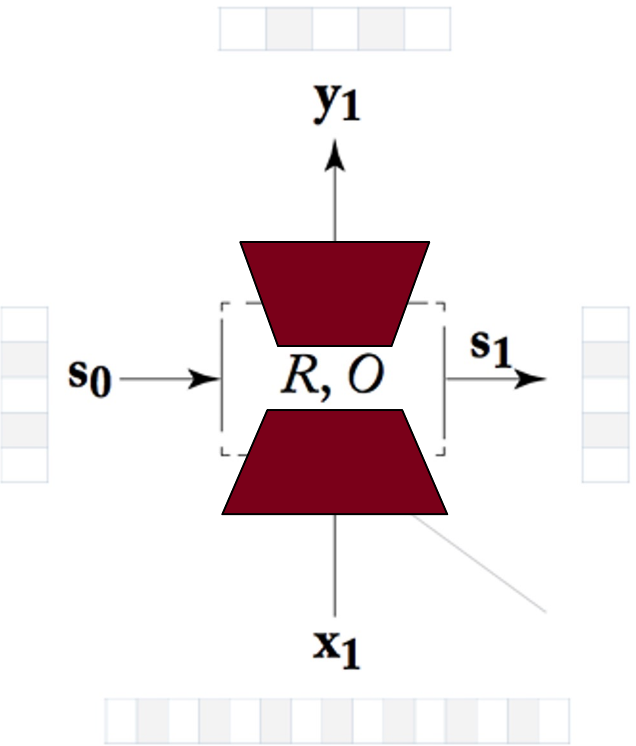


Recurrent Neural Network

- Each time set has two outputs:

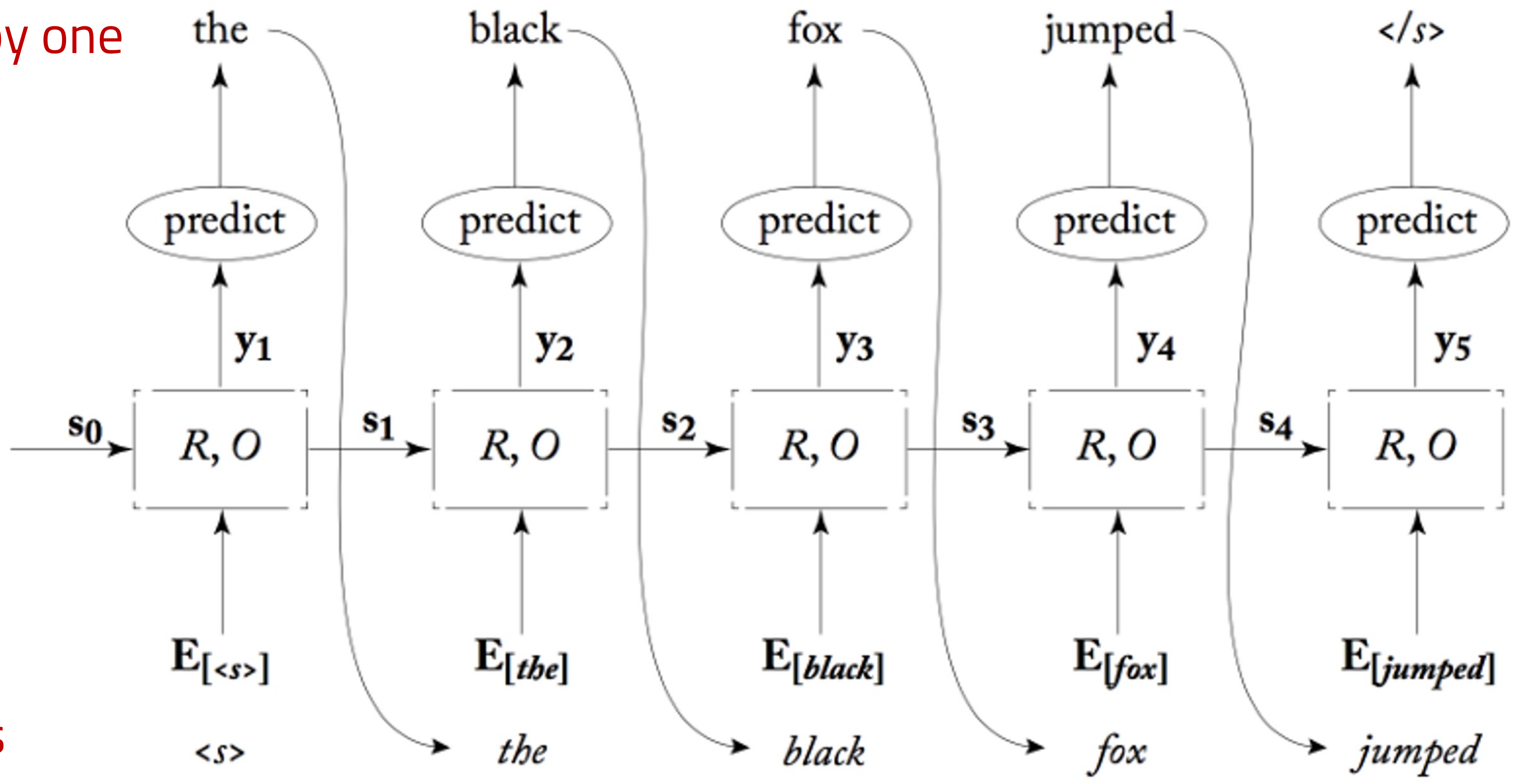
- $S_i = R\ (X_i, S_{i-1})$
  - R computes the output state as a function of the *current input* and *previous state*

- $y_i = O\ (S_i)$
  - O computes the output as a function of the *current output state*



RNN Training

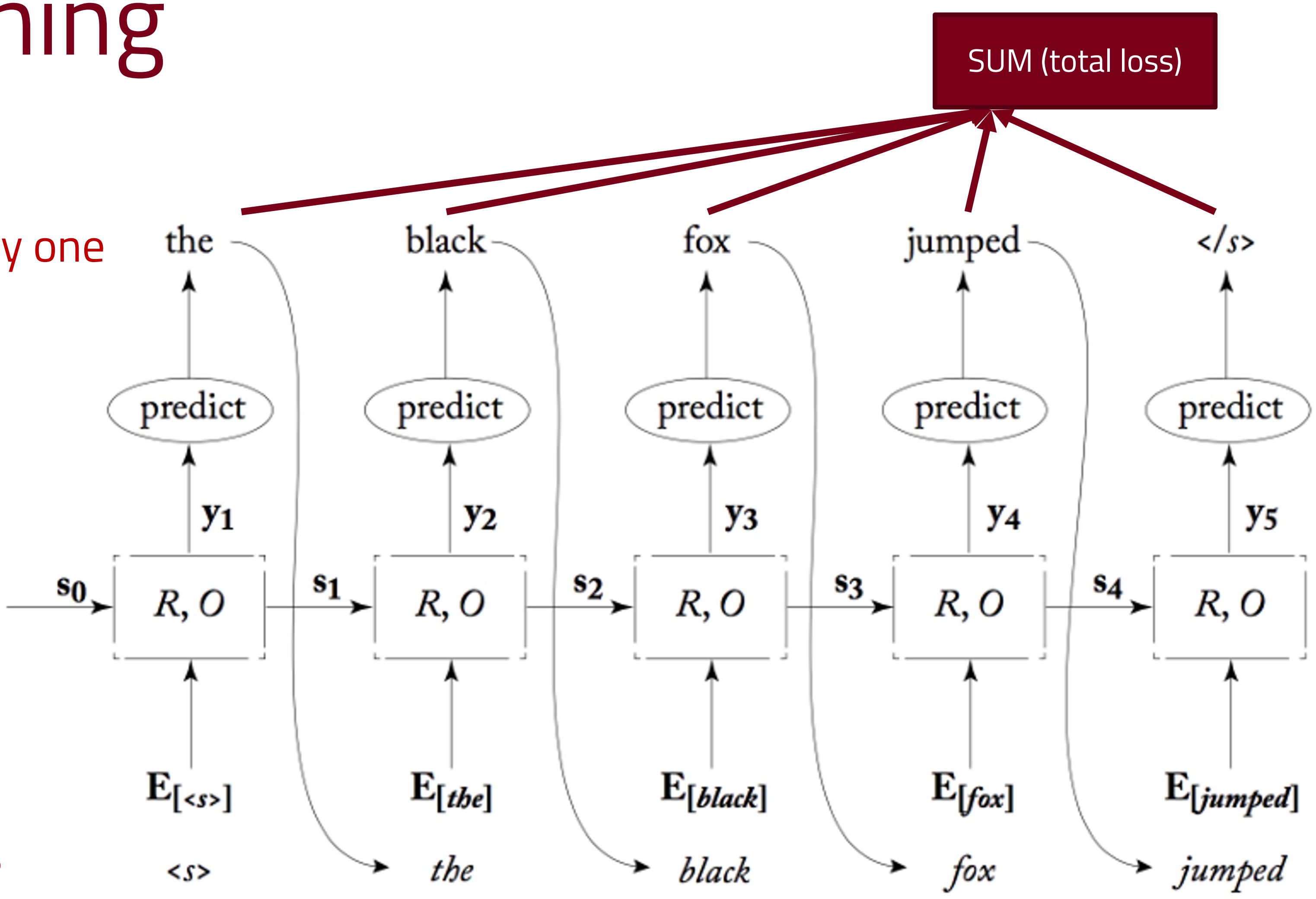
output as shifted by one



sequence of words

RNN Training

output as shifted by one

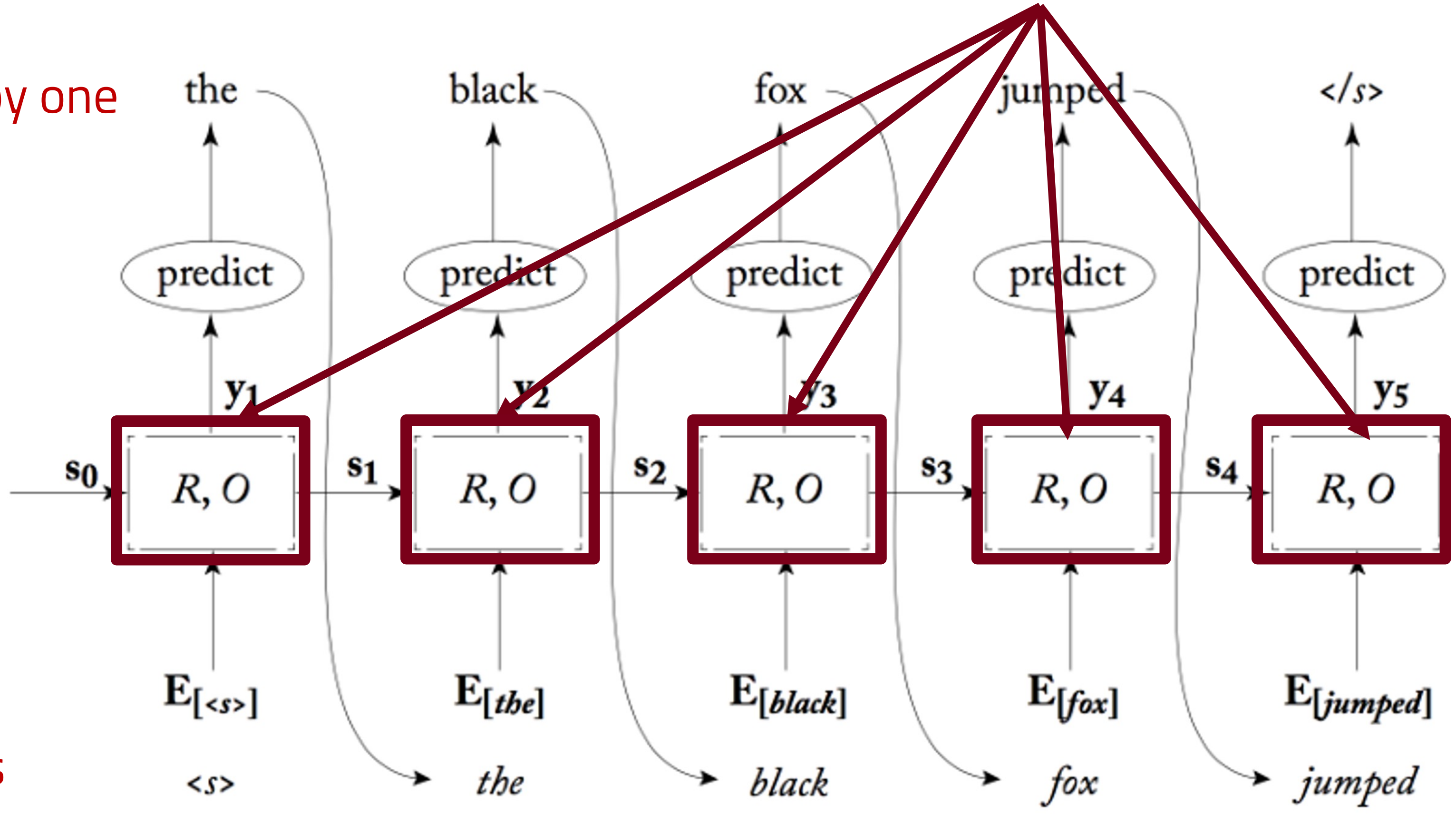


sequence of words

RNN Training

Parameters are shared!
Derivatives are accumulated.

output as shifted by one



sequence of words

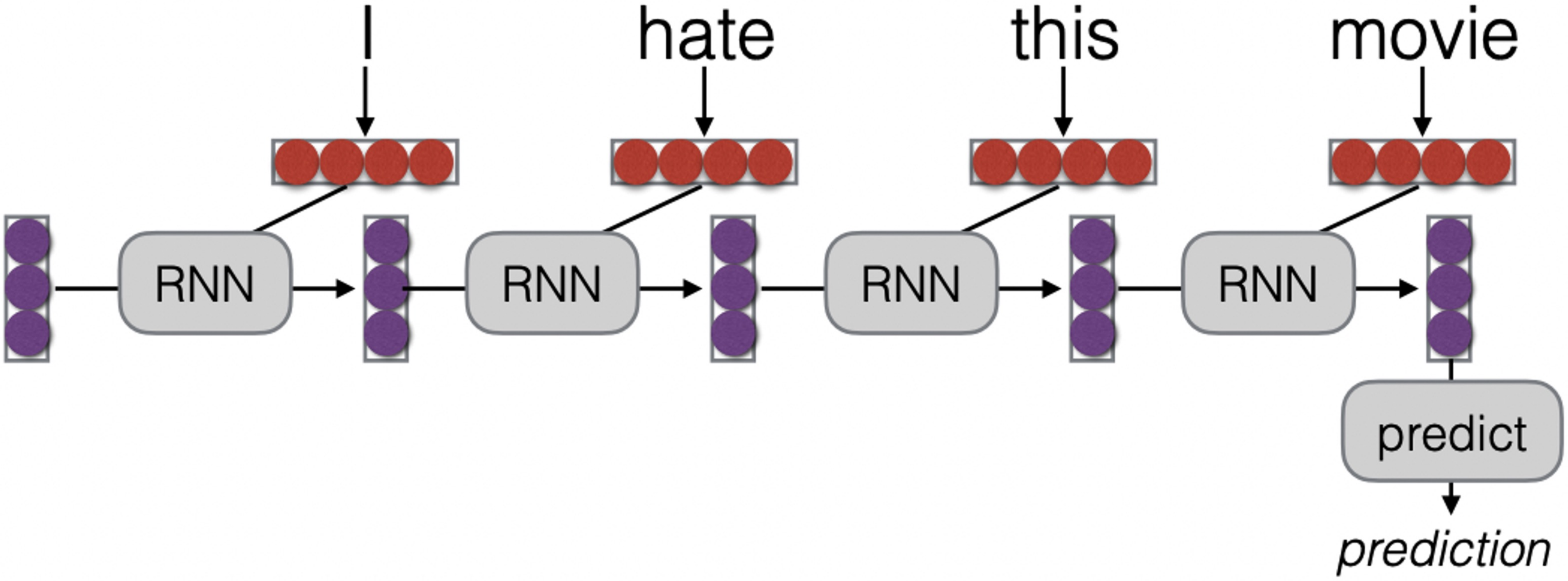
What can RNNs do?

- Represent a sentence
  - Read whole sentence, make a prediction

- Represent a context within a sentence
  - Read context up until that point

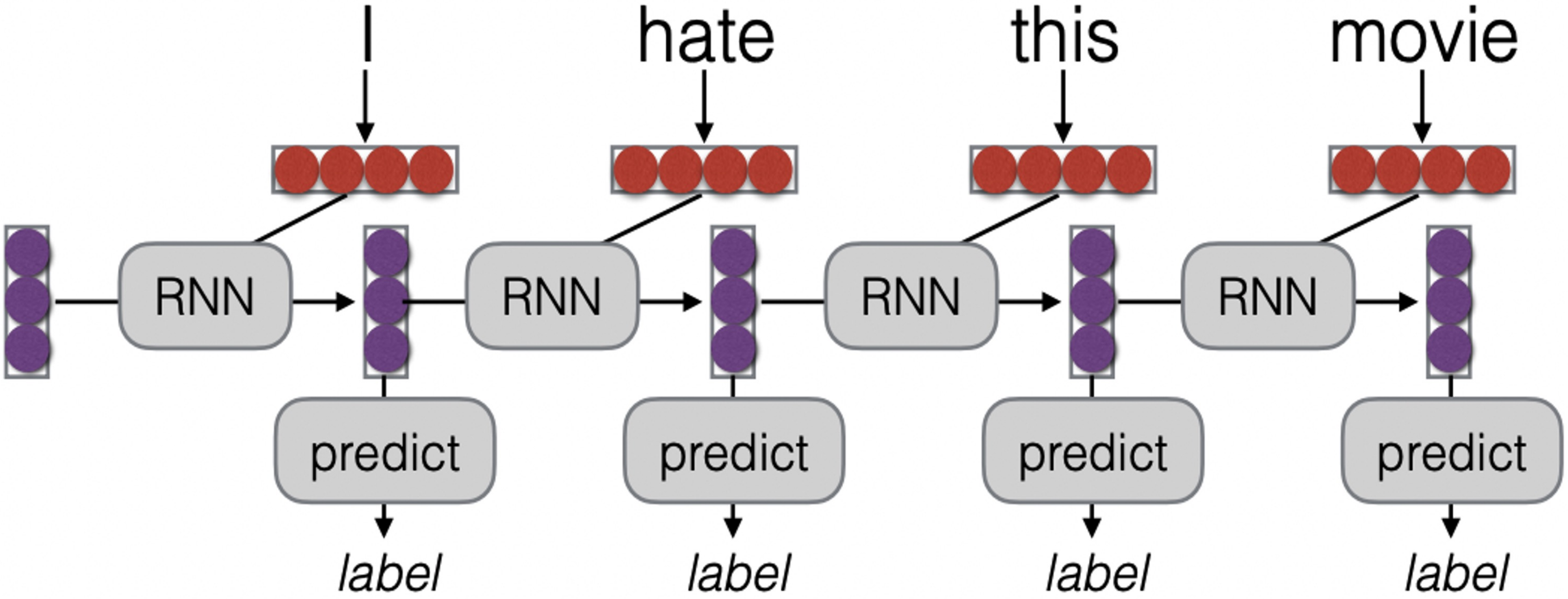
Representing Sentences

- Sentence classification
- Conditioned generation



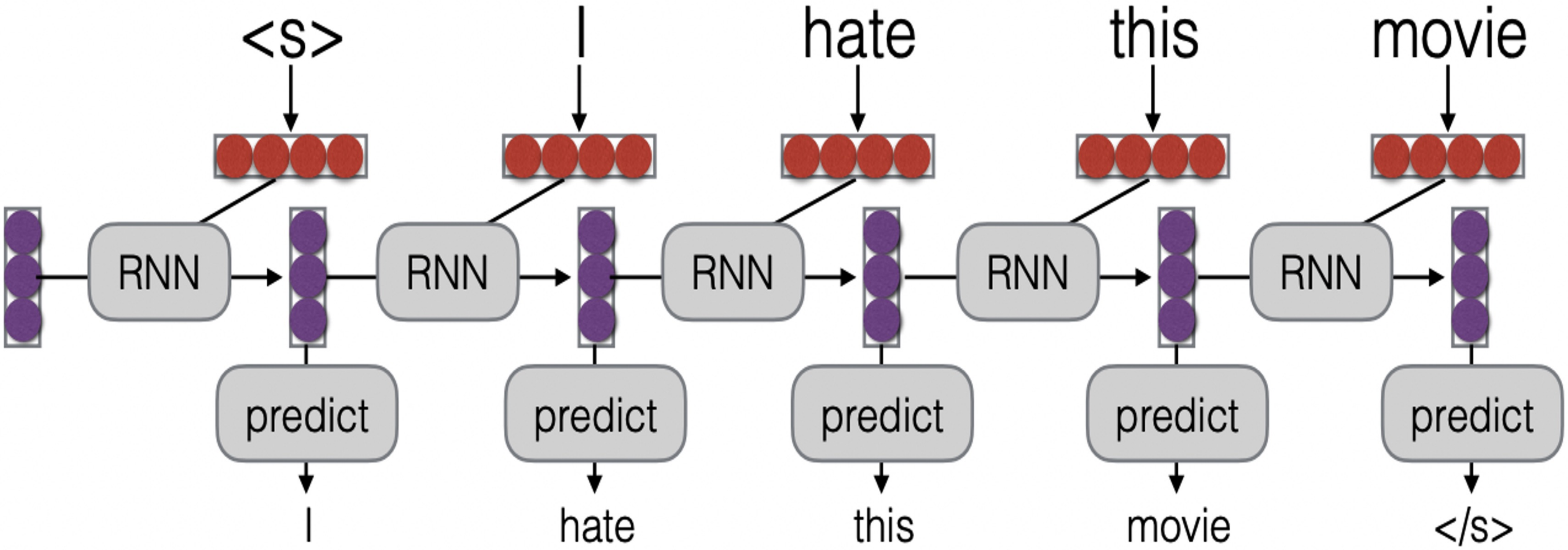
Representing Context within Sentence

- Tagging
- Language modeling

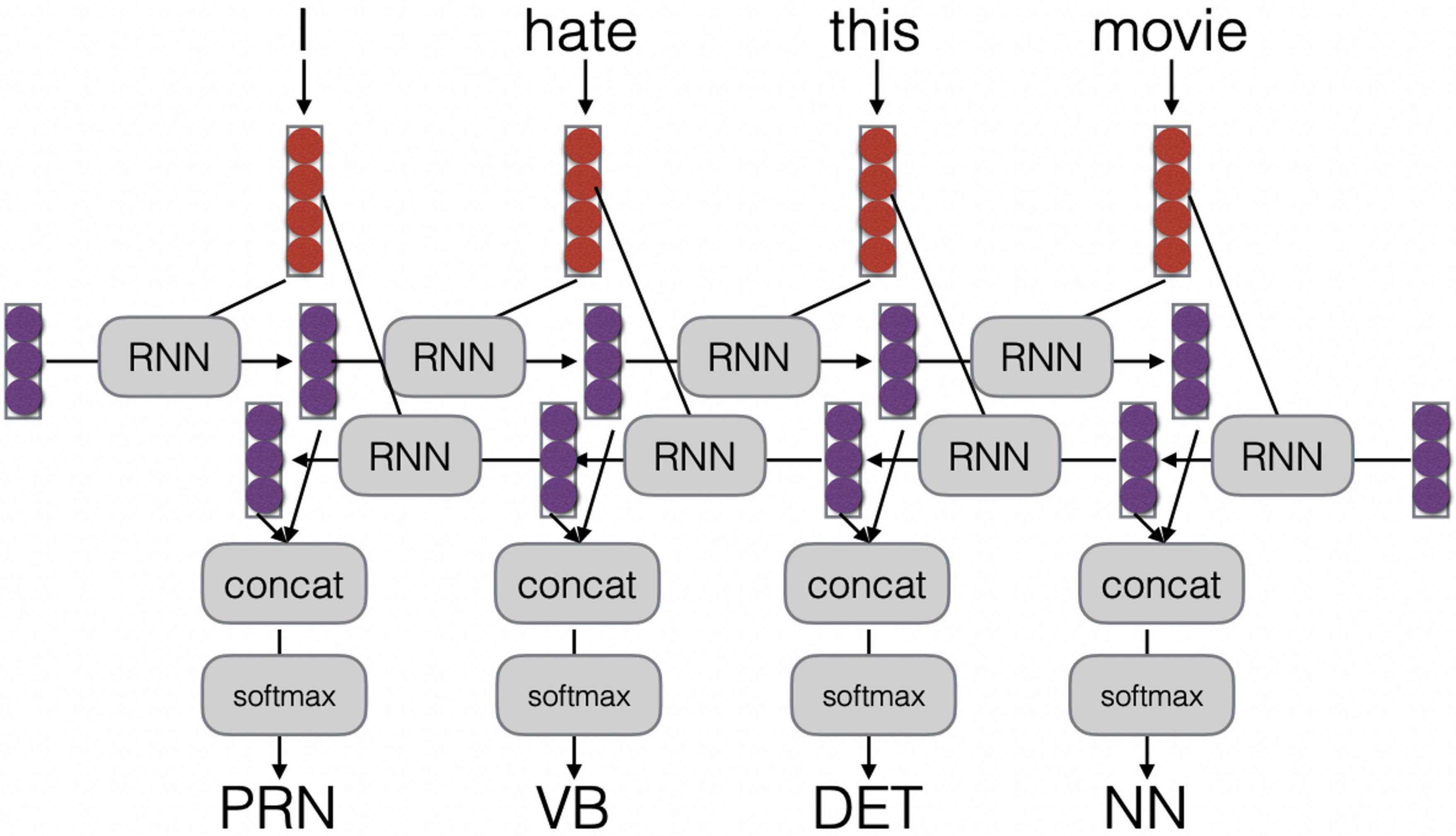


e.g., Language Modeling

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



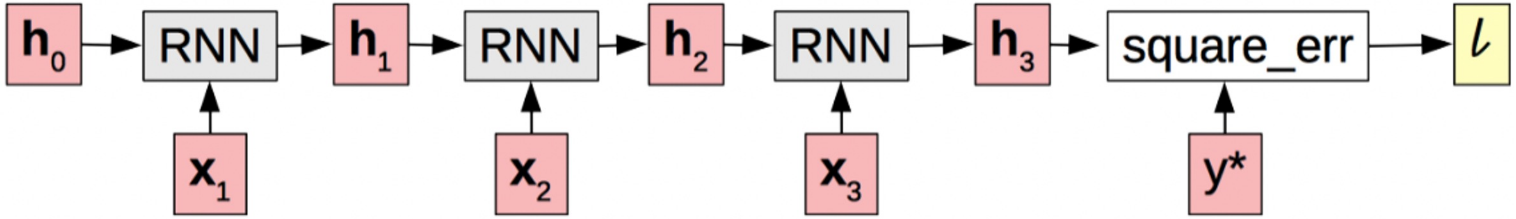
e.g., POS Tagging with Bi-RNNs



Vanishing Gradient

- Gradients decrease as they get pushed back

$$\frac{dl}{d_{h_0}} = \text{tiny} \qquad \frac{dl}{d_{h_1}} = \text{small} \qquad \frac{dl}{d_{h_2}} = \text{med.} \qquad \frac{dl}{d_{h_3}} = \text{large}$$



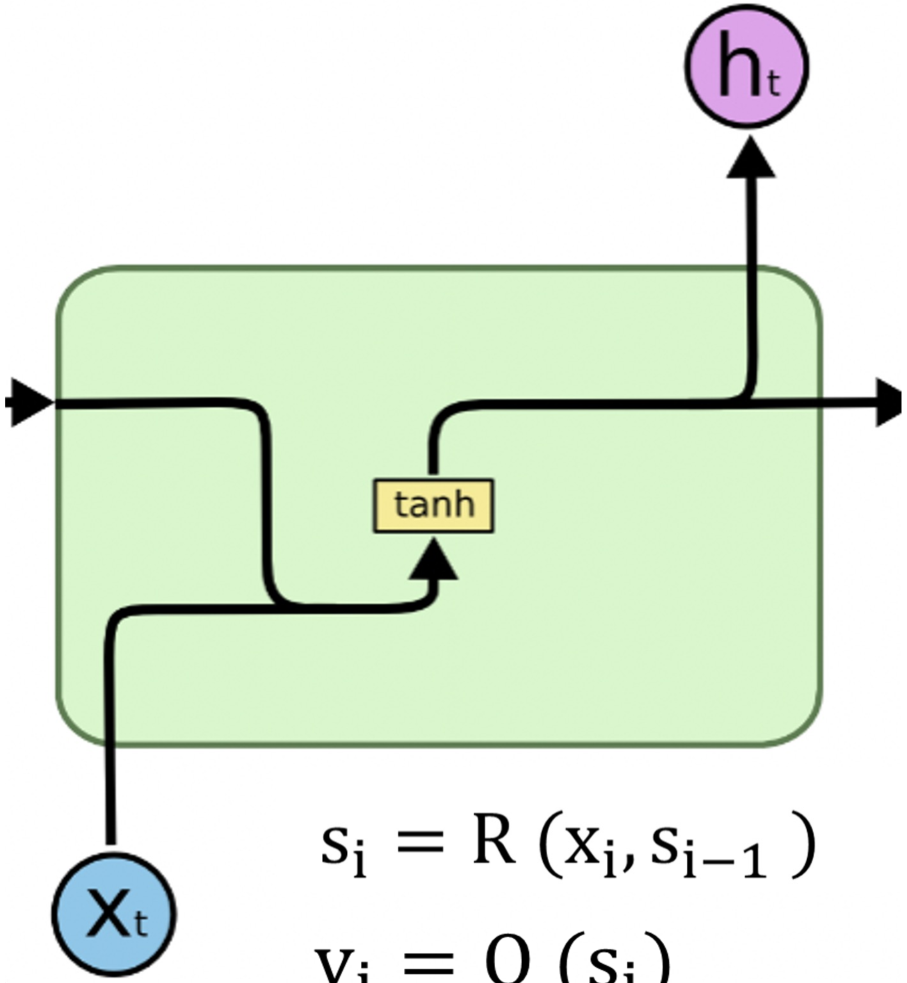
- Why? “Squashed” by non-linearities or small weights in matrices

A Solution: Long Short-term Memory (LSTM)

(Hochreiter and Schmidhuber 1997)

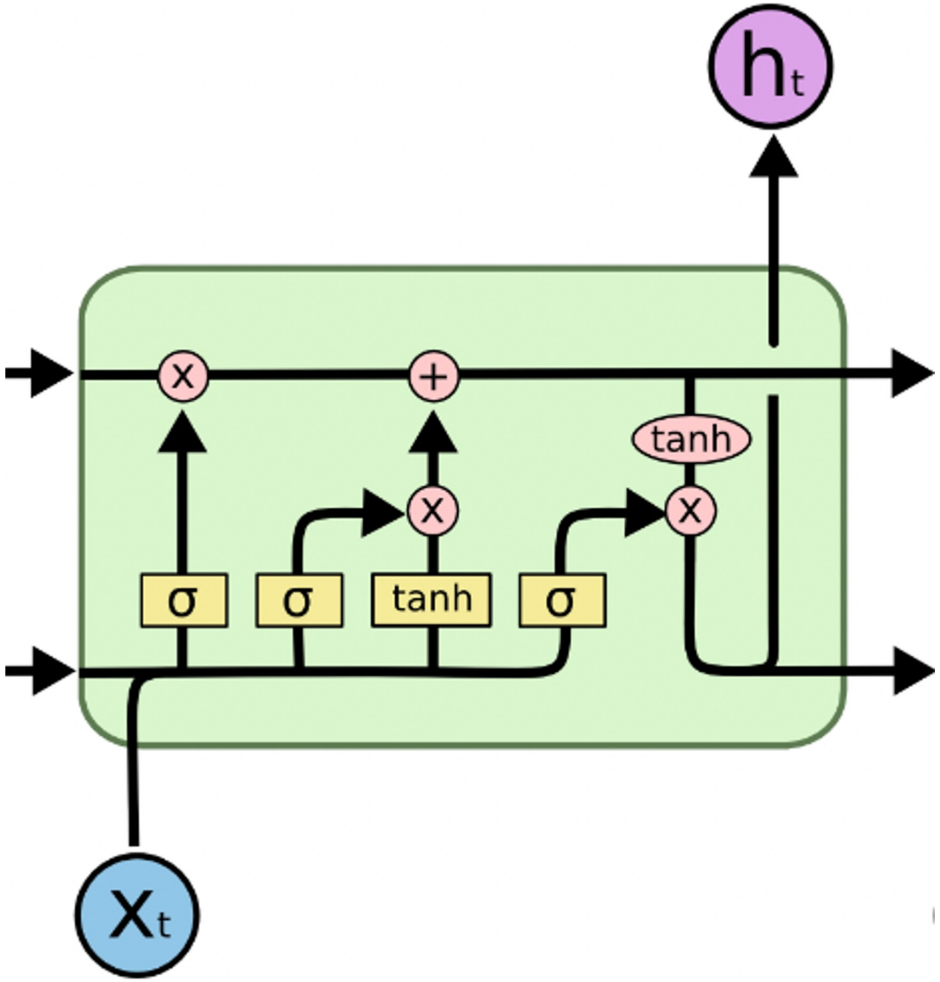
- Make **additive connections** between time steps
- Addition does not modify the gradient, no vanishing
- **Gates** to control the information flow

RNN vs LSTM Structure



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

$$y_i = O(s_i)$$

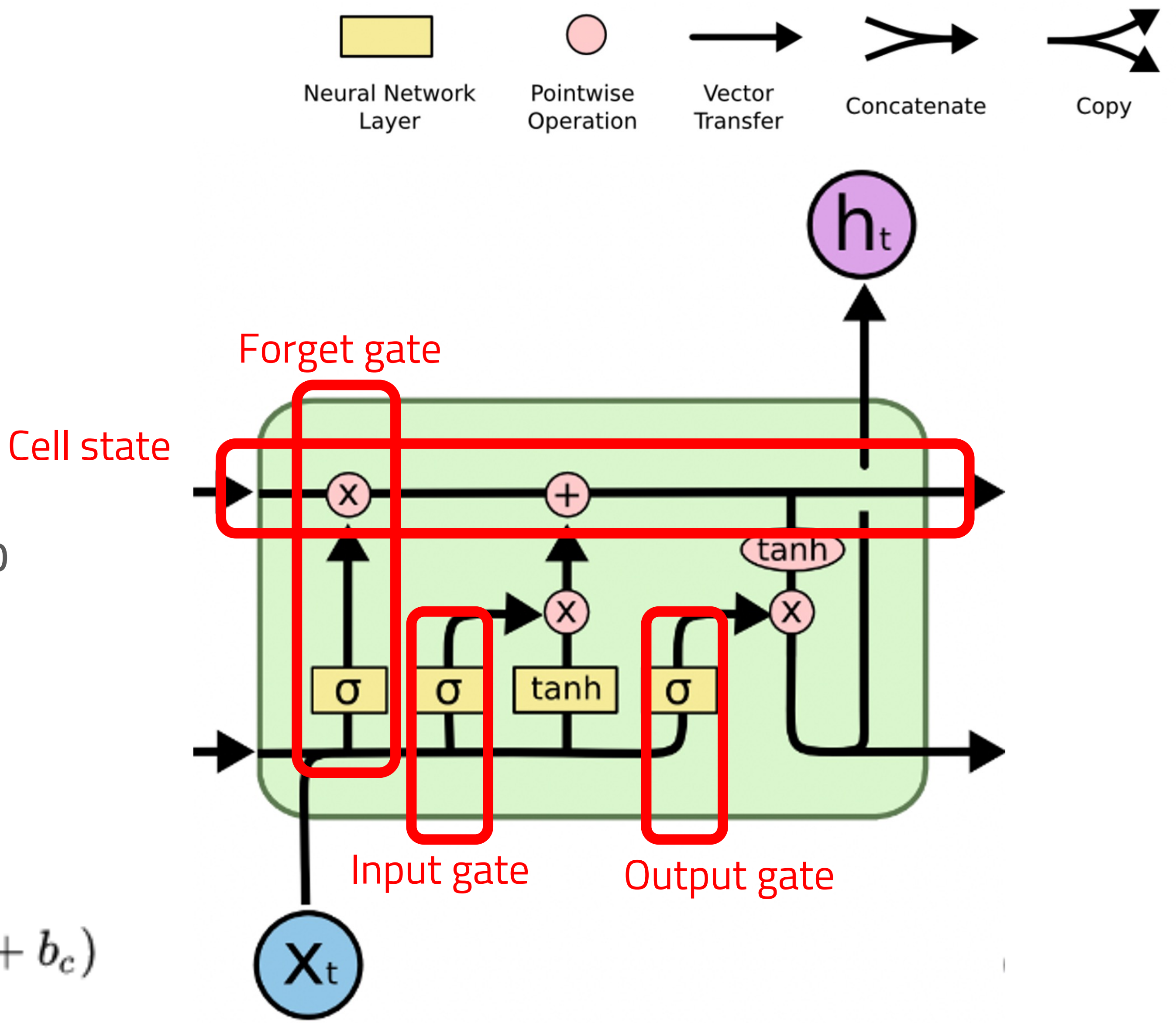


http://colah.github.io/posts/2015-08-Understanding-LSTMs/

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?

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$



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

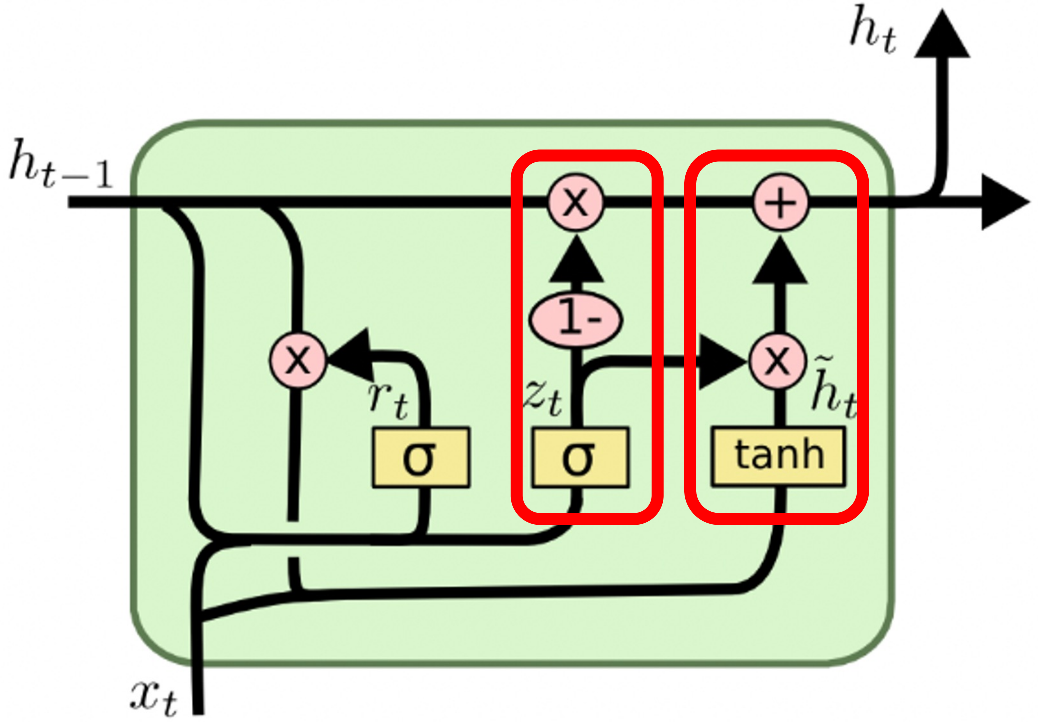
LSTM variant: Gated Recurrent Unit (GRU)

(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

$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$
$$h_t = \boxed{(1 - z_t)} \circ h_{t-1} + \boxed{z_t} \circ \sigma_h(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h)$$

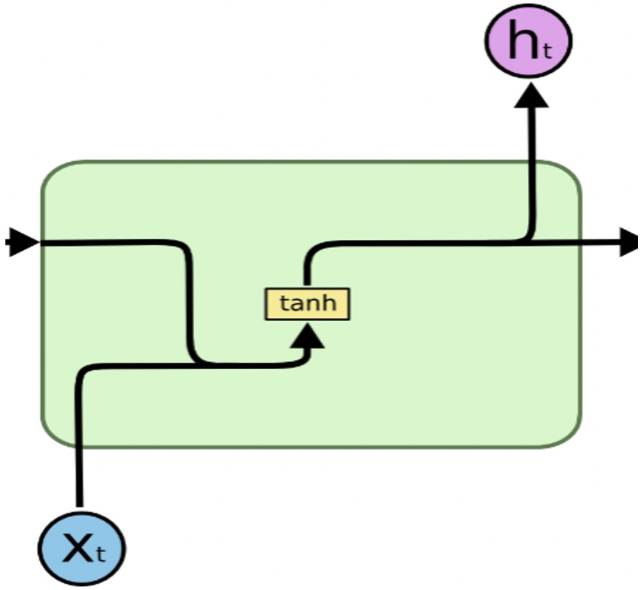
Additive or Non-linear



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

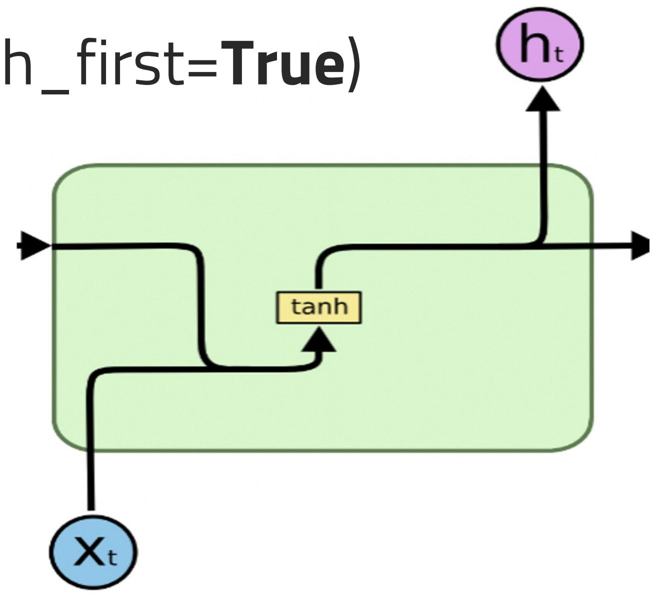
PyTorch

```
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

    def init_zero_hidden(self, batch_size=1) -> torch.Tensor:
        return torch.zeros(batch_size, self.hidden_size, requires_grad=False)
```



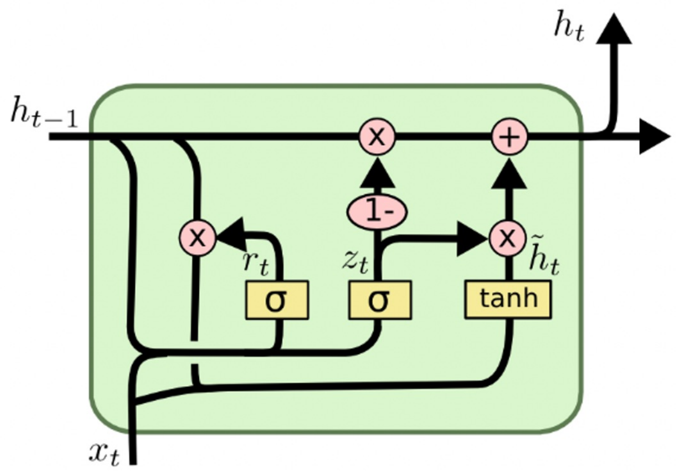
PyTorch

```
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
```




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

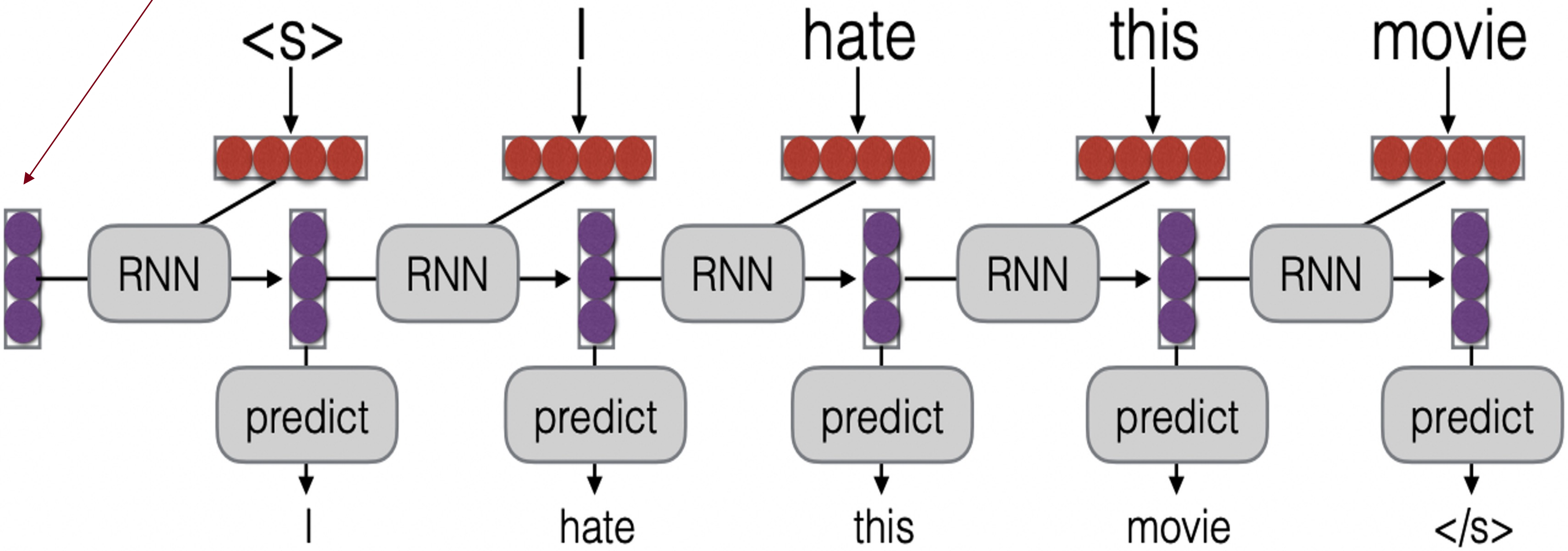
```
class LSTM (nn.Module):
    def __init__(self, num_classes, input_size, hidden_size, num_layers,
seq_length):
        super(LSTM1, self).__init__()
        ...
        self.lstm = nn.LSTM(input_size=input_size, hidden_size=hidden_size,
num_layers=num_layers, batch_first=True)
        self.fc =  nn.Linear(hidden_size, num_classes)
        self.relu = nn.ReLU()

    def forward(self,x):
        h_0 = Variable(torch.zeros(self.num_layers, x.size(0), self.hidden_size)) #hidden
state
        c_0 = Variable(torch.zeros(self.num_layers, x.size(0), self.hidden_size)) #internal
state
        output, (hn, cn) = self.lstm(x, (h_0, c_0))
        hn = hn.view(-1, self.hidden_size)
        return self.fc (self.relu(hn))
```



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

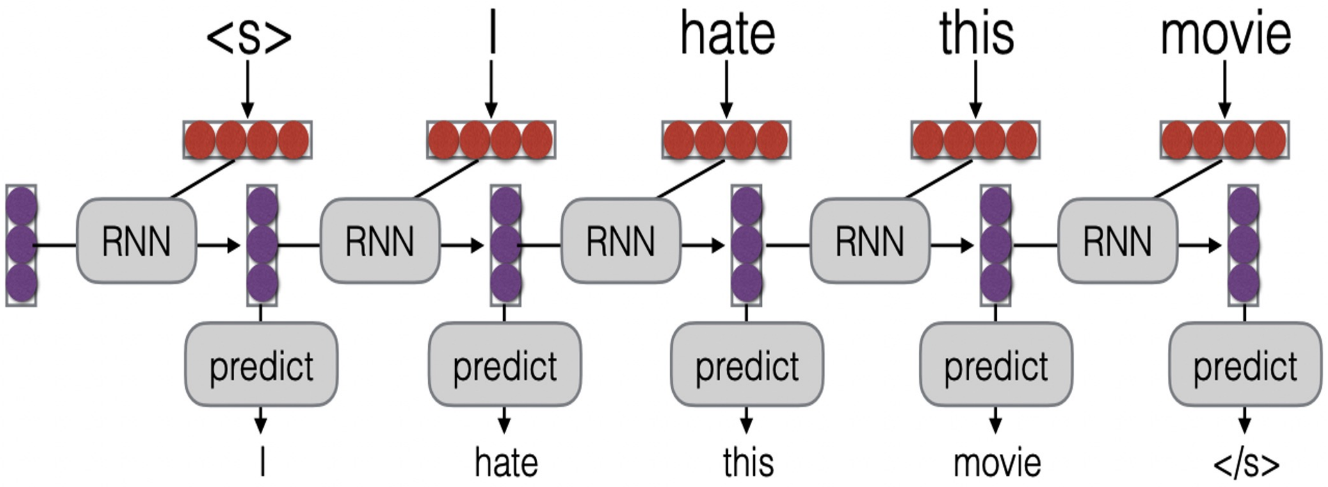
RNN (decoder) for language modeling

Randomly initialized hidden state h_t at time step $t = 0$



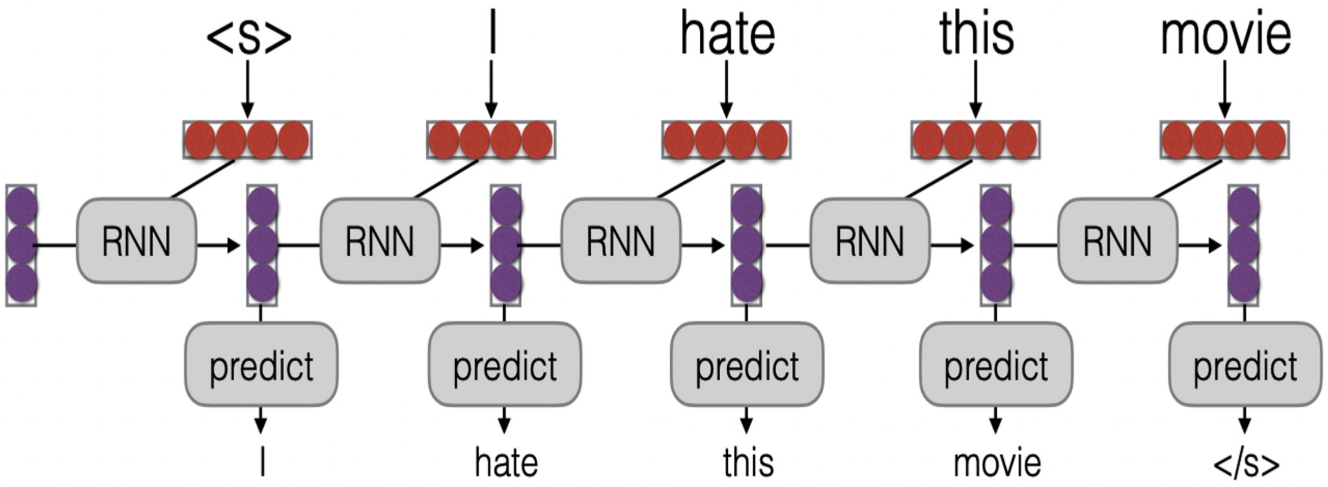
RNN (decoder) for language modeling

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



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

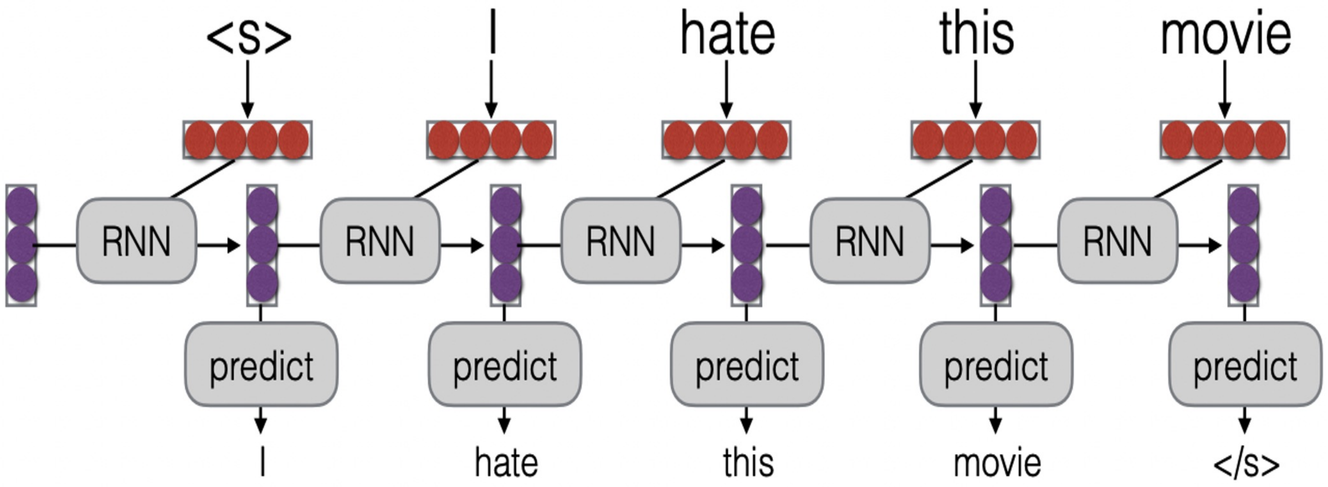
"나는 이 영화가 싫어요"
"Odio esta película"



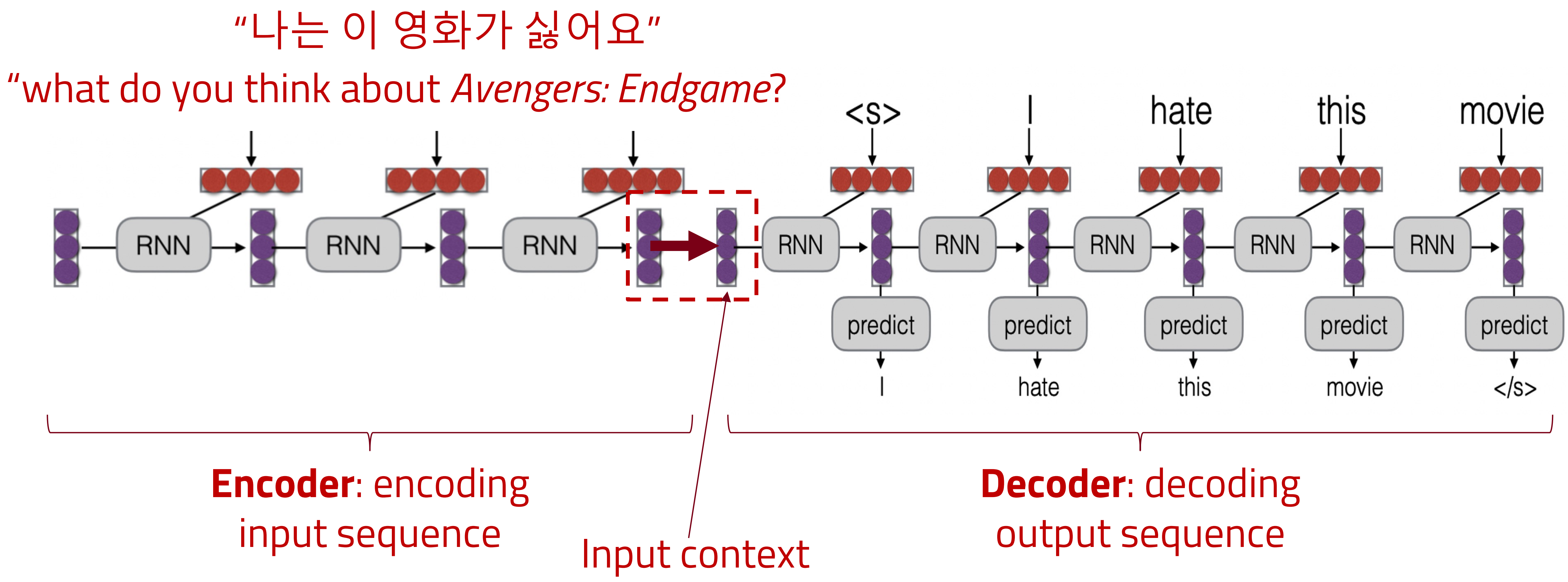
RNN (encoder) - RNN (decoder) for dialogue generation

"나는 이 영화가 싫어요"
"Odio esta película"

"what do you think about *Avengers: Endgame?*"



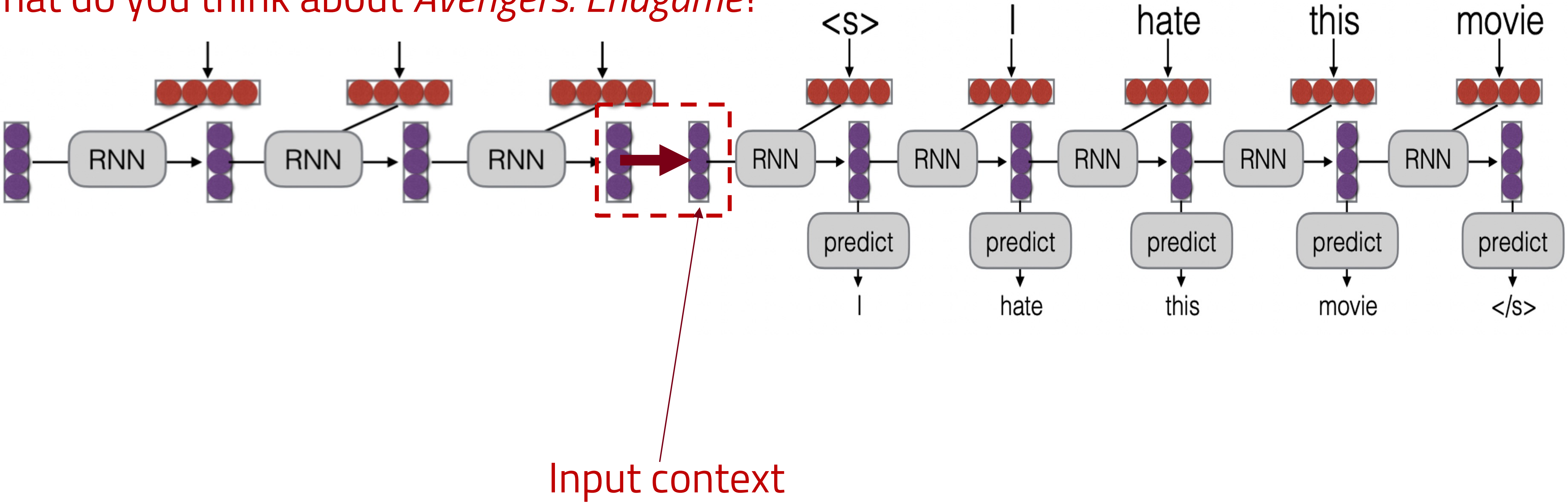
Sequence-to-sequence modeling using RNN (encoder) - RNN (decoder)



Problem: forgetting input context as input gets longer

"나는 이 영화가 싫어요"

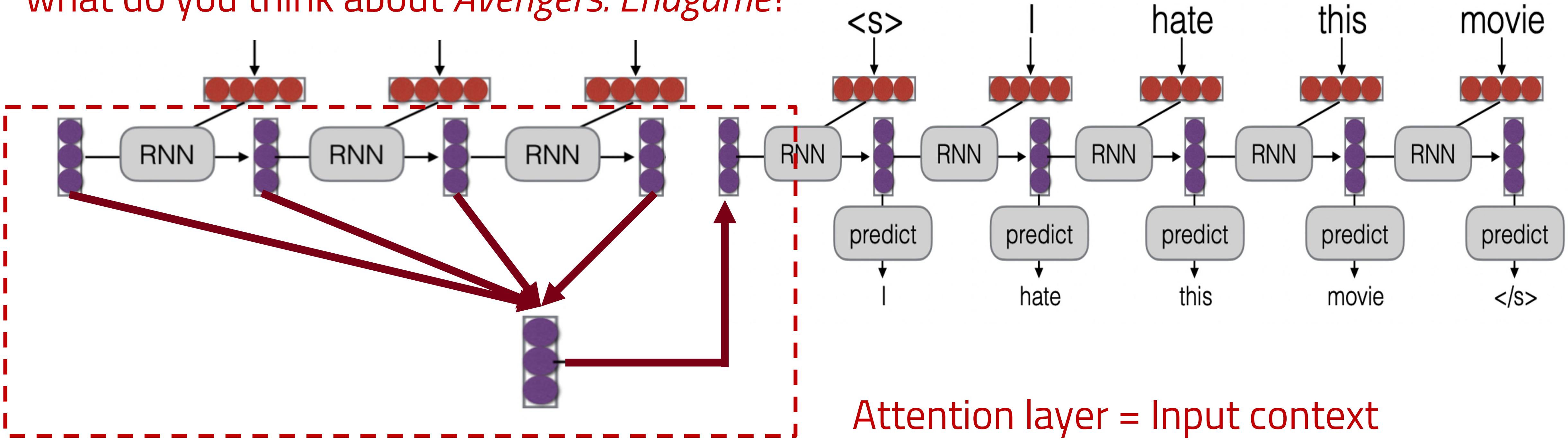
"what do you think about *Avengers: Endgame*?



Solution (teaser): Seq2seq with attention

"나는 이 영화가 싫어요"

"what do you think about *Avengers: Endgame*?



Attention layer = Input context attended on all previous context

State-of-the-art Language Models

Teaser: Transformer-based LMs

- SOTA LMs: GPT-2, Radford et al. 2018; GPT

Trigram	LSTM	GPT2	GPT3
109	58.3	35.8	20.5

Mar 19	Transformers (1) Project proposal due	• Attention is All you Need • Tutorial on Illustrated Transformer • Language Models are Unsupervised Multitask Learners
Mar 24	Transformers (2)	• Language Models are Few-Shot Learners • Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer

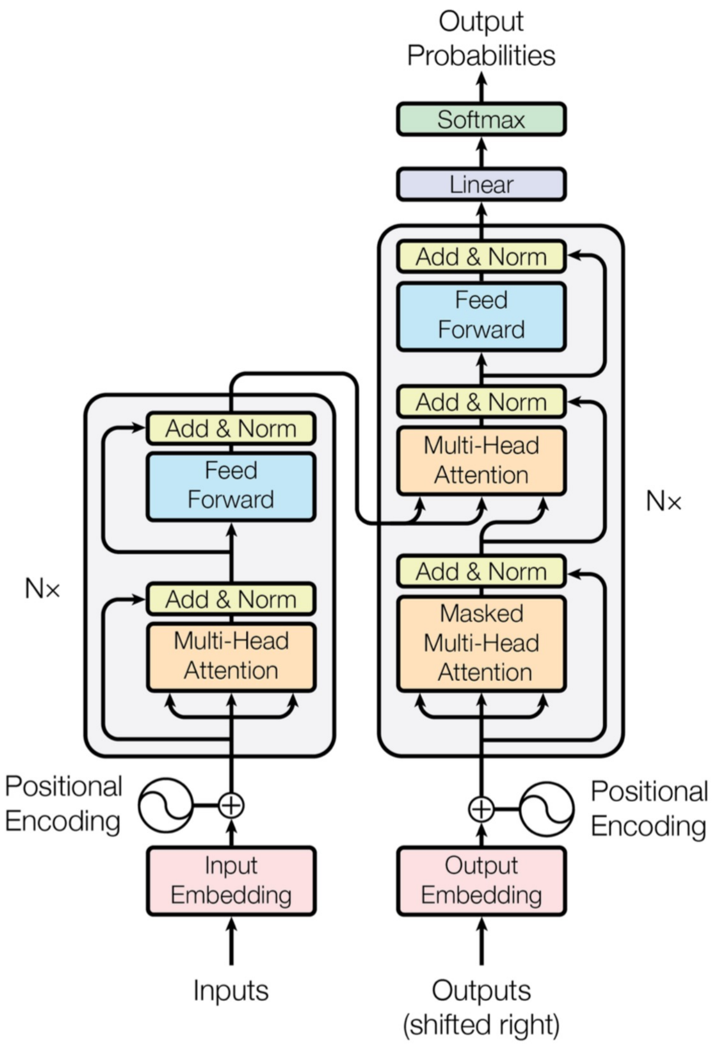
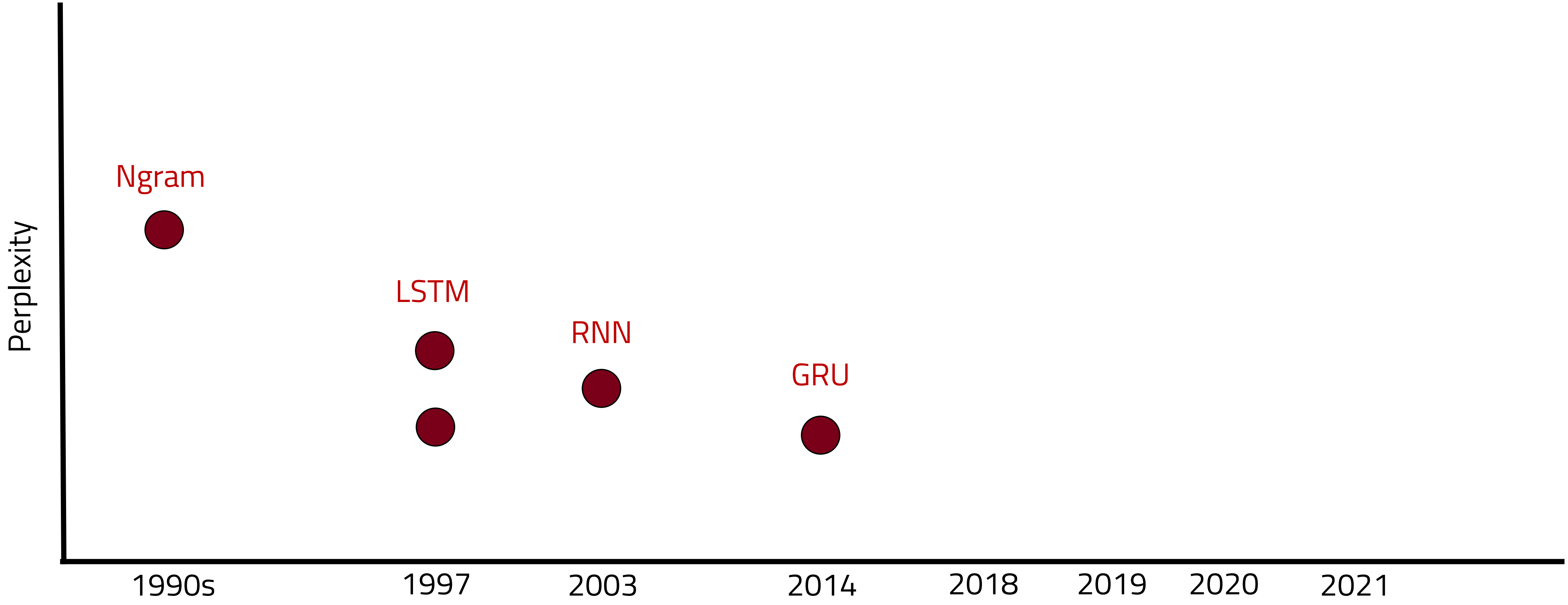
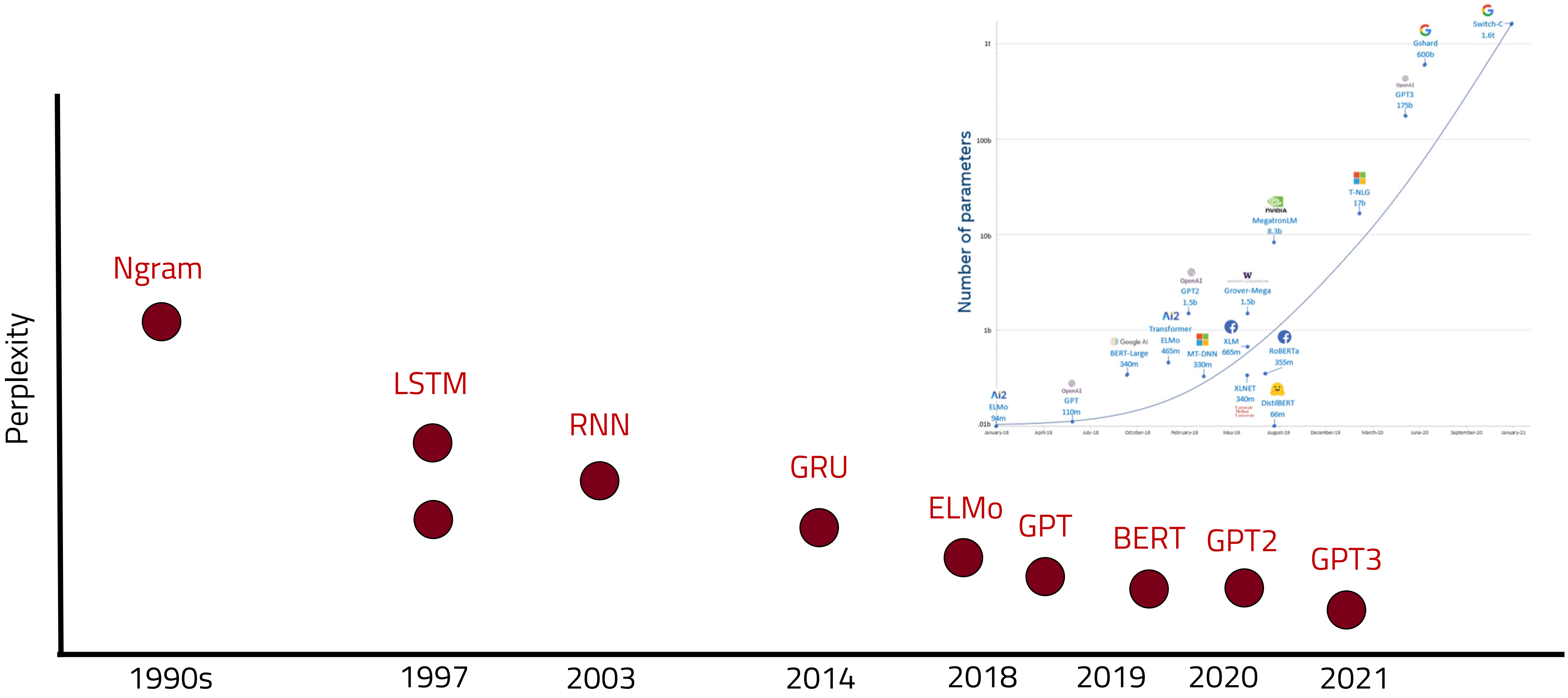


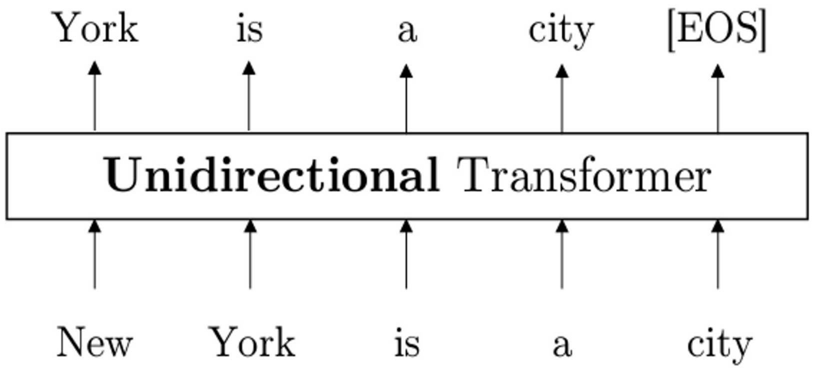
Figure 1: The Transformer - model architecture.


Perplexity
Ngram
LSTM
RNN
GRU
1990s
1997
2003
2014
2018
2019
2020
2021


Perplexity
Ngram
LSTM
RNN
GRU
ELMo
GPT
BERT
GPT2
GPT3
1990s
1997
2003
2014
2018
2019
2020
2021
Number of parameters
Switch-C 1.6t
Gshard 600b
GPT3 175b
T-NLG 17b
MegatronLM 8.3b
GPT2 1.5b
Grover-Mega 1.5b
Transformer ELMo 465m
BERT-Large 340m
MT-DNN 330m
XLM 665m
RoBERTa 355m
XLNET 340m
DistilBERT 66m
ELMo 94m
GPT 110m

Teaser: Two Objectives for Language Model Pretraining

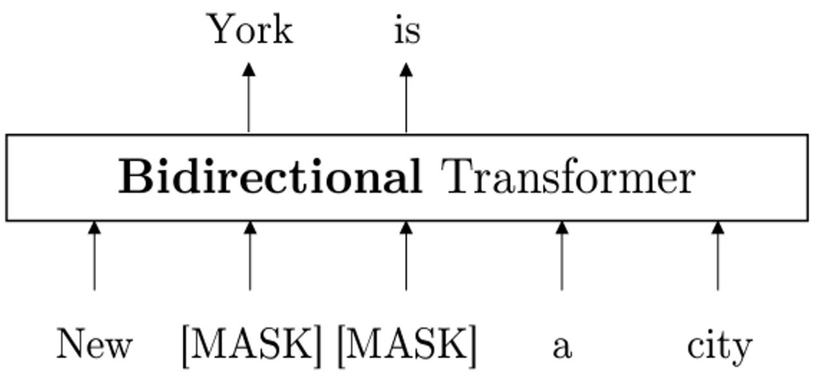
Auto-regressive LM (GPT3)



$$\log p(\mathbf{x}) = \sum_{t=1}^{T} \log p(x_t | \mathbf{x}_{<t})$$

Next-token prediction

Denoising autoencoding (BERT)



$$\log p(\bar{\mathbf{x}} | \hat{\mathbf{x}}) = \sum_{t=1}^{T} \text{mask}_t \log p(x_t | \hat{\mathbf{x}})$$

Reconstruct masked tokens

Slides from Zihang Dai

Recap

- Ngram LM $\longrightarrow$ Neural LM : sparsity
- Neural LM $\longrightarrow$ RNN LM : input size is not scalable
- RNN LM $\longrightarrow$ LSTM LM: vanishing gradients over time steps
- LSTM LM $\longrightarrow$ Transformer : still vanishing gradients
- Transformer $\longrightarrow$ Scaling up Transformer : scaling law!

Why better language models are useful?

Language models can directly encode knowledge present in the training corpus.

The director of 2001: A Space Odyssey is ______________

Language models can directly encode knowledge present in the training corpus.

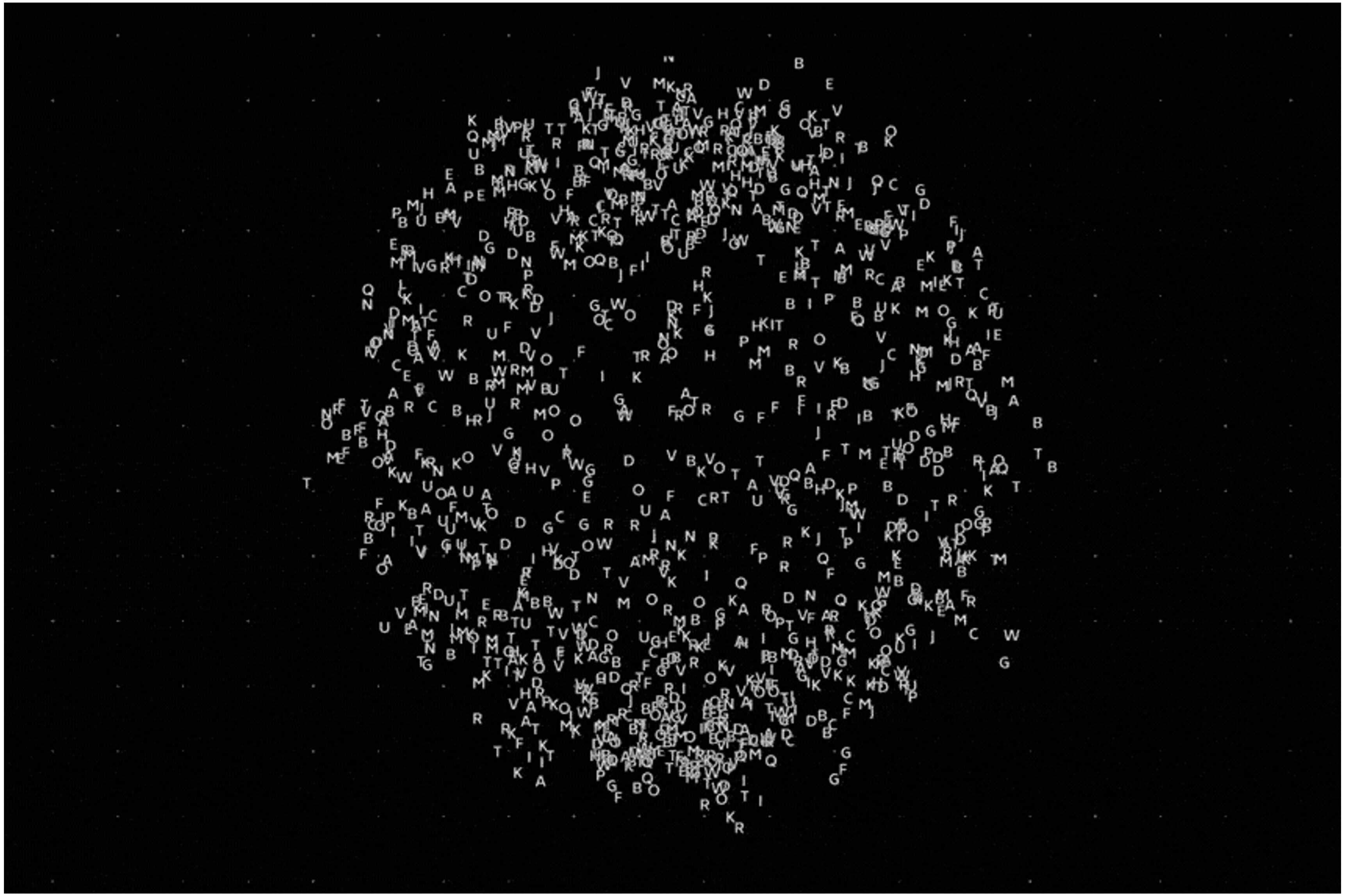
Query	Answer	Generation
Francesco Bartolomeo Conti was born in ____.	Florence	Rome [-1.8] , **Florence** **[-1.8]** , Naples

Petroni et al. (2019), "Language Models as Knowledge Bases?" (ACL)

Language models can directly encode knowledge present in the training corpus.

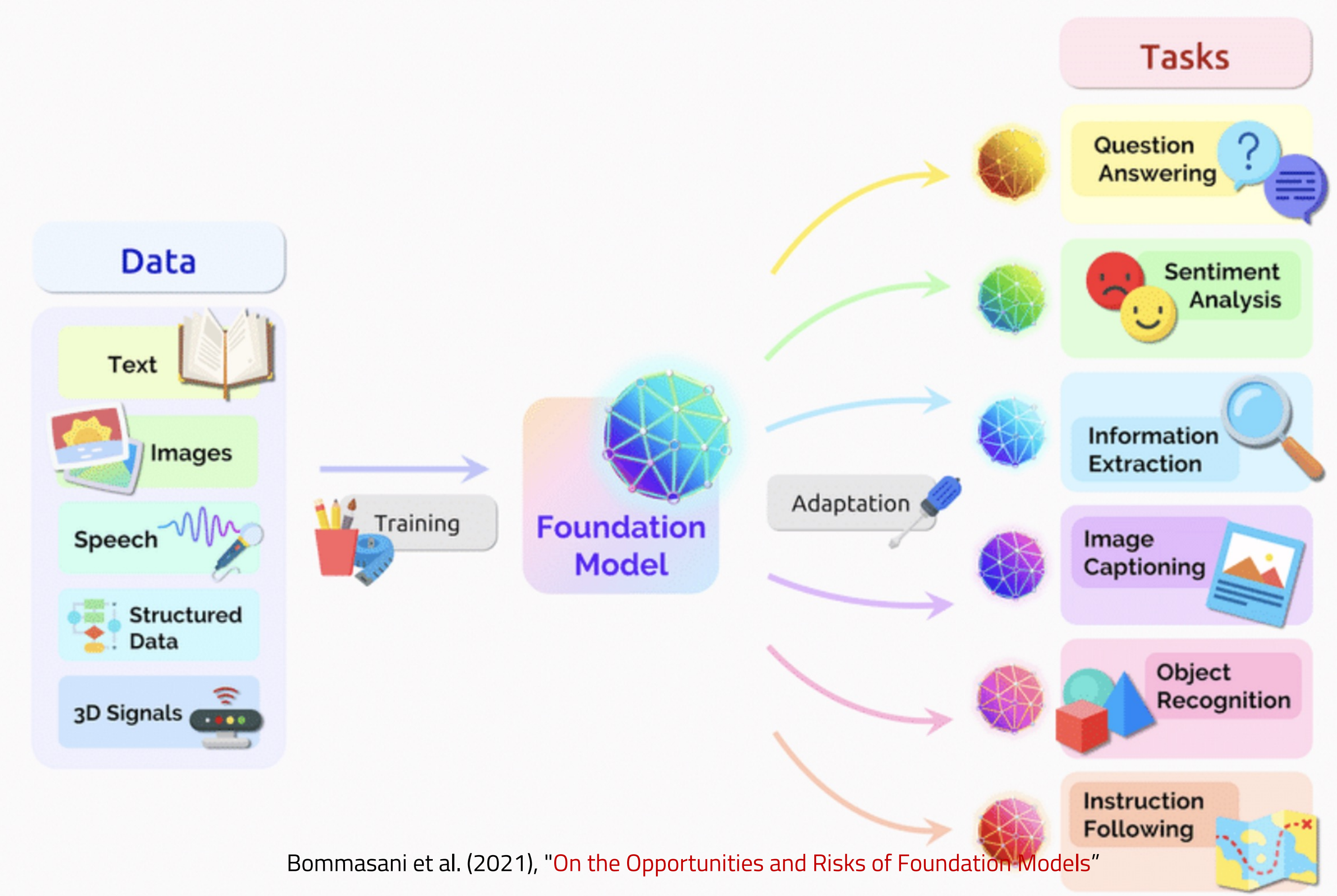
Query	Answer	Generation
Francesco Bartolomeo Conti was born in ____.	Florence	Rome [-1.8] , **Florence [-1.8]** , Naples
Adolphe Adam died in ____.	Paris	**Paris [-0.5]** , London [-3.5] , Vienna
English bulldog is a subclass of ____.	dog	dogs [-0.3] , breeds [-2.2] , **dog**
The official language of Mauritius is ____.	English	**English [-0.6]** , French [-0.9] , Arabic
Patrick Oboya plays in ____ position.	midfielder	centre [-2.0] , center [-2.2] , **midfielder**
Hamburg Airport is named after ____.	Hamburg	Hess [-7.0] , Hermann [-7.1] , Schmidt

Petroni et al. (2019), "Language Models as Knowledge Bases?" (ACL)



ChatGPT Is a Blurry JPEG of the Web, By Ted Chiang February 9, 2023

Language models can be a foundation for various tasks across different modalities



Bommasani et al. (2021), "On the Opportunities and Risks of Foundation Models"

Language models are stochastic parrots

Bender et al. (2021), "On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?"

Questions

- GPT3 is 100x bigger than GPT2. If GPT-K is developed, how can we handle such a large-scale model without industry-level computing powers. Can we compress the models while not sacrificing performance?
- What if those companies can only replicate the results, monopolize their usages, and make them as a paid service? Is it fair?
- Are there different ways of storing the predictive/knowledge power of LMs?
- Can LMs be called as general intelligence or foundational knowledge? If not, what are missing there?