CSCI 5541: Natural Language Processing

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

computer science
& engineering

MINNESOTA · NLP · EST. 2021

University of Minnesota
Driven to Discover®

Announcement (0213)

- Minor HW2 Revisions --> See slack announcement
- HW3 is released. The due date is due Tue, Feb 25.
- Project
  - Brainstorming is due next Tuesday, Feb 18
  - Groups have been assigned in slack
  - There are a couple of students not yet in groups. If you have a fully formed group and are willing to take on someone else, let me know.

Ngram LM

Uni-gram

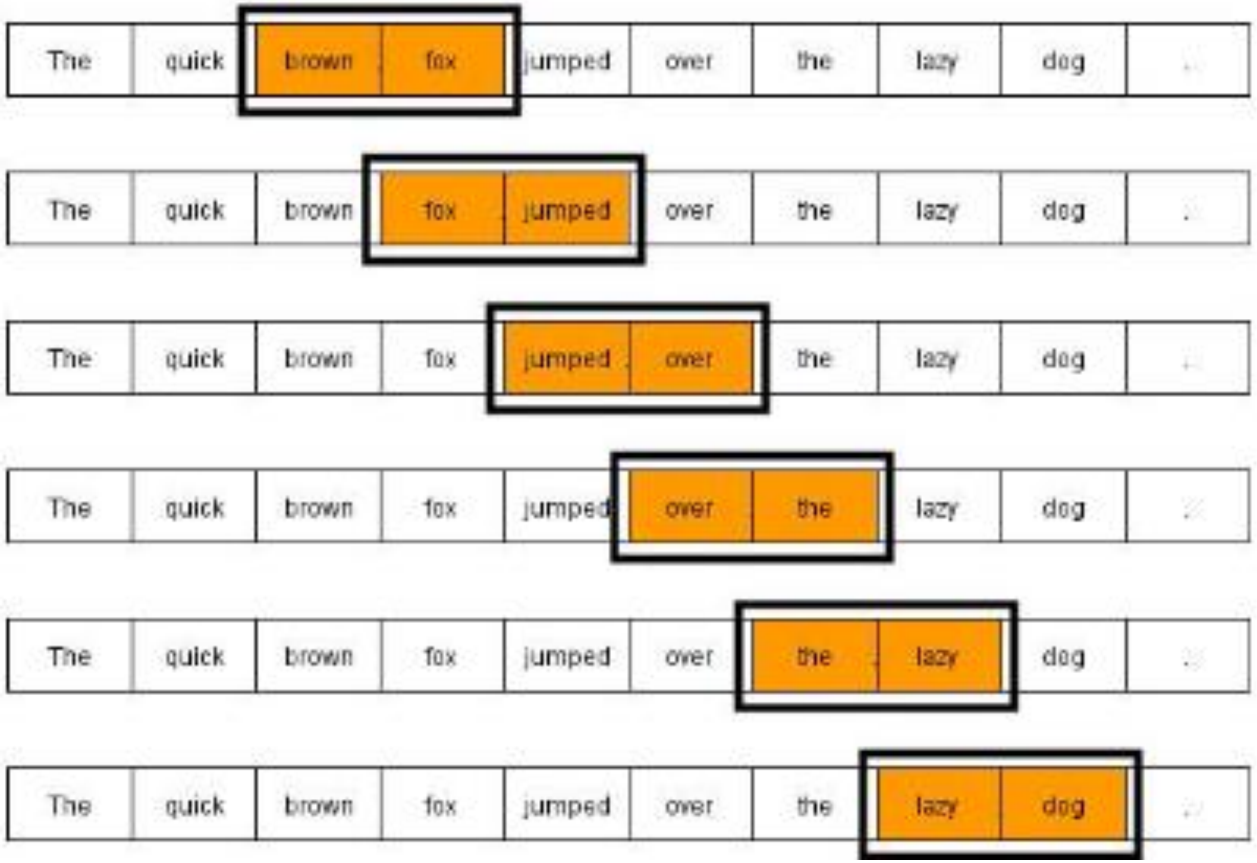
$$\prod_{i=1}^{n} P(w_i) \times P(STOP)$$

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

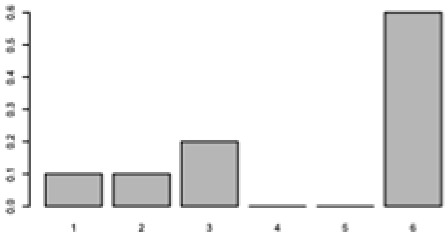
Bi-gram

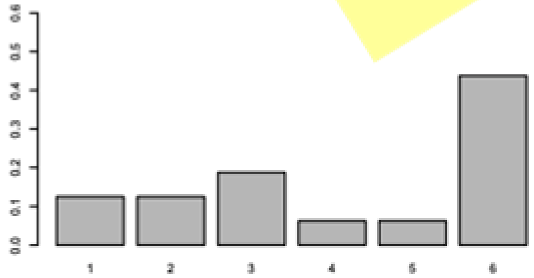
$$\prod_{i=1}^{n} P(w_i | w_{i-1}) \times P(STOP \mid w_n)$$

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



Sparsity in Ngram LM





	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

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.

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

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

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

Ngram LM vs Neural LM

To avoid the data sparsity problem from the ngram LM

Neural LM

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

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

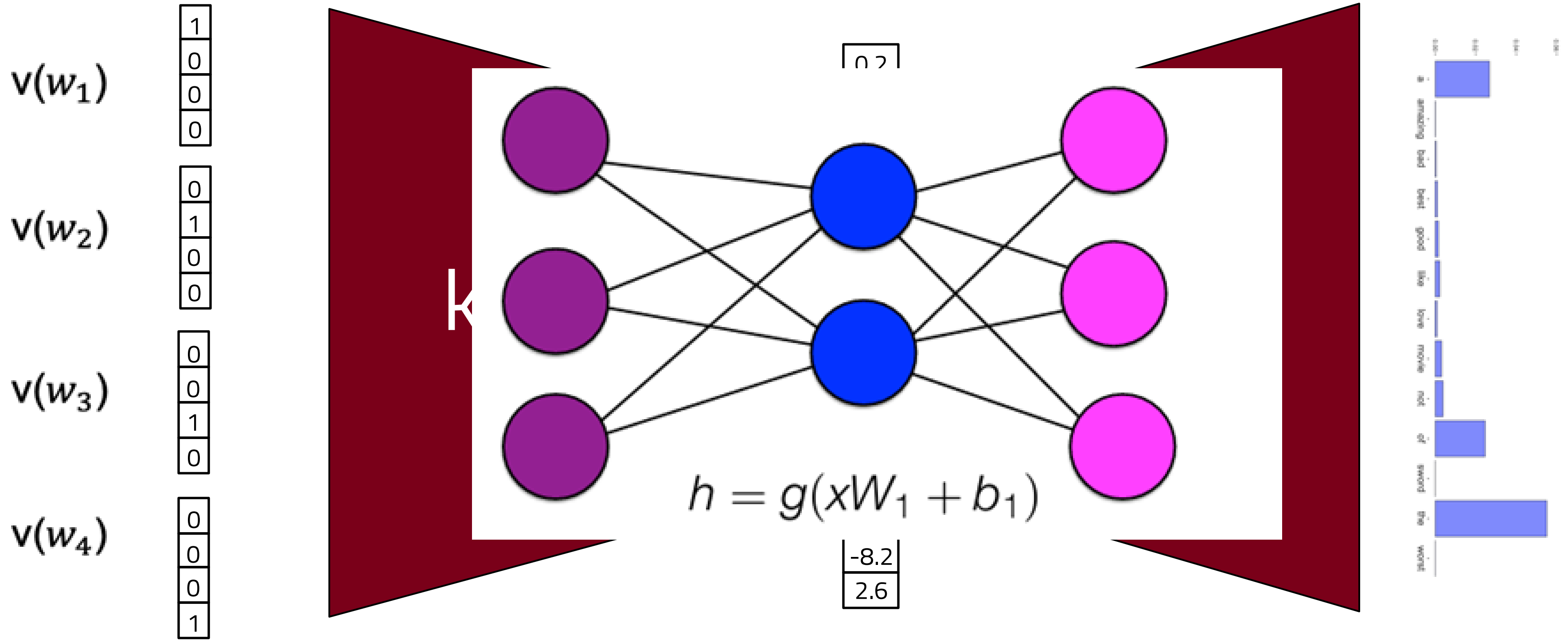
Concatenation (k x V)

w_1 = tried

w_2 = to

w_3 = prepare

w_4 = midterms



One-hot encoding

Distributed representation

Multi-class (Vocab) classification

Bengio et al. 2003, A Neural Probabilistic Language Model

Neural LM

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

Recap

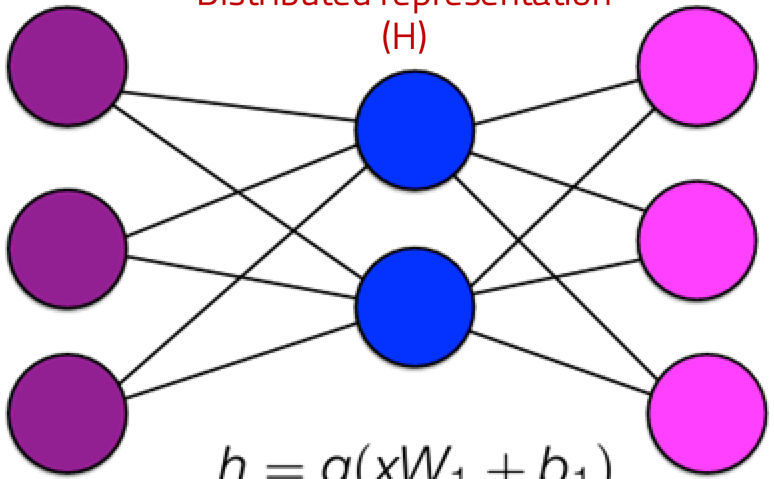
$$W_1 \in \mathbb{R}^{kV \times H} \qquad W_2 \in \mathbb{R}^{H \times V}$$

$$b_1 \in \mathbb{R}^{H} \qquad b_2 \in \mathbb{R}^{V}$$

One-hot encoding
(|x| = V)

Output space: |y| = V

Distributed representation
(H)



$$h = g(xW_1 + b_1)$$

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

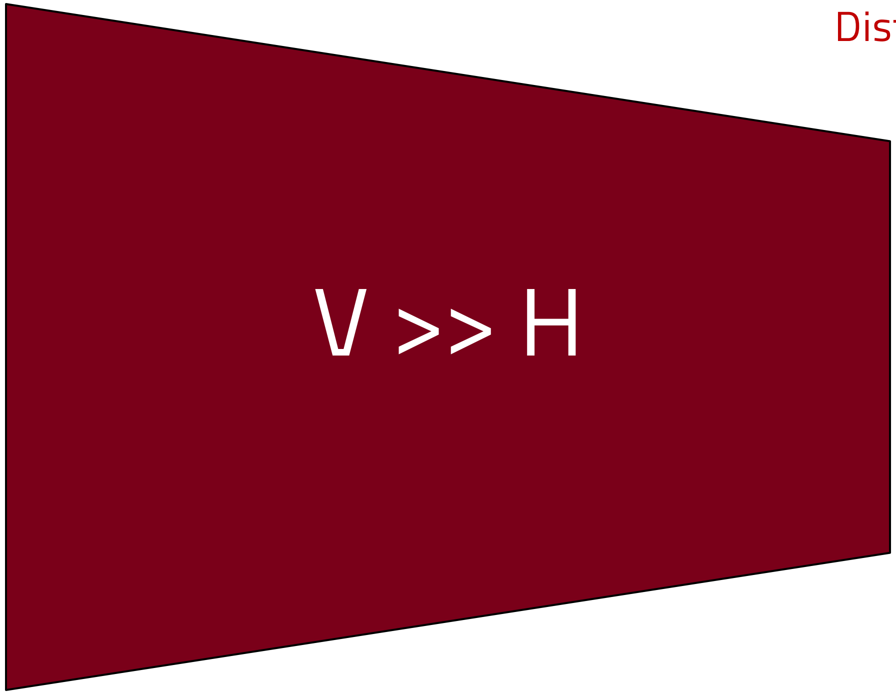
$$\hat{y} = \text{softmax}(hW_2 + b_2)$$

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)

Bengio et al. 2003, A Neural Probabilistic Language Model

Conditioning context (X [k x V])

tried to prepare midterm | but | I was too tired of…

Next word to predict (Y)

Context window size: k=4

Recap

Conditioning context (X [k x 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 x 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
- Teaser: Transformer-based LMs
- Why language models are useful?

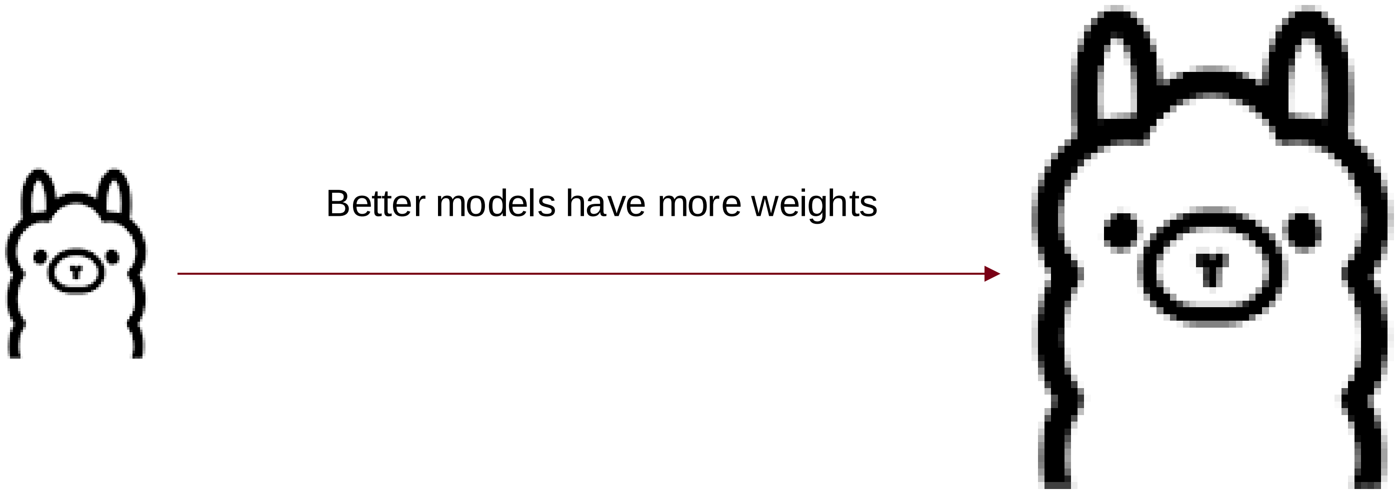
Outline

- ❑ **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?

How do we make a better model?

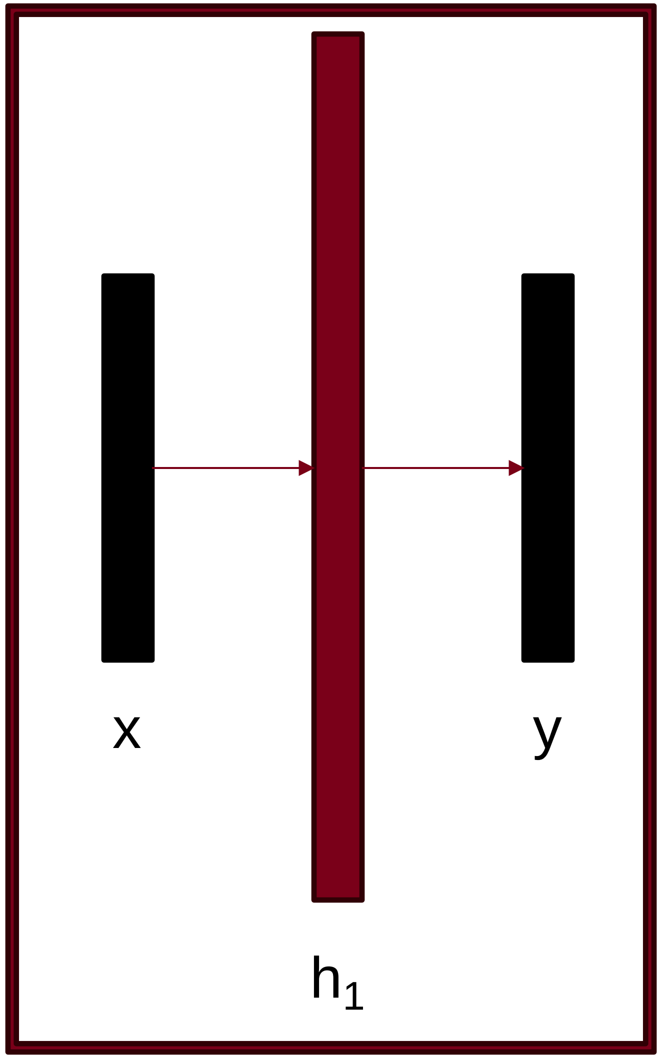
?

More Params are Better

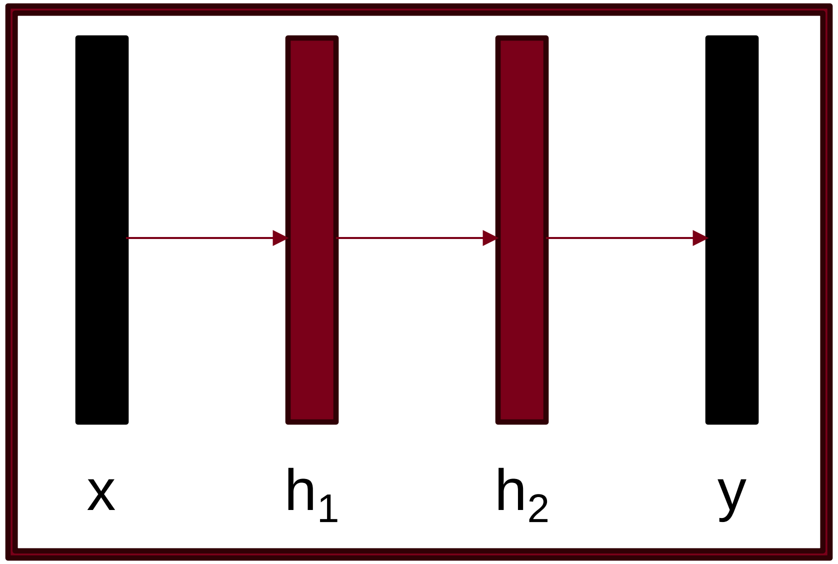


Increasing depth is more efficient than width

Model 1: ❌



Model 2: ✅



…but very deep models are harder to train

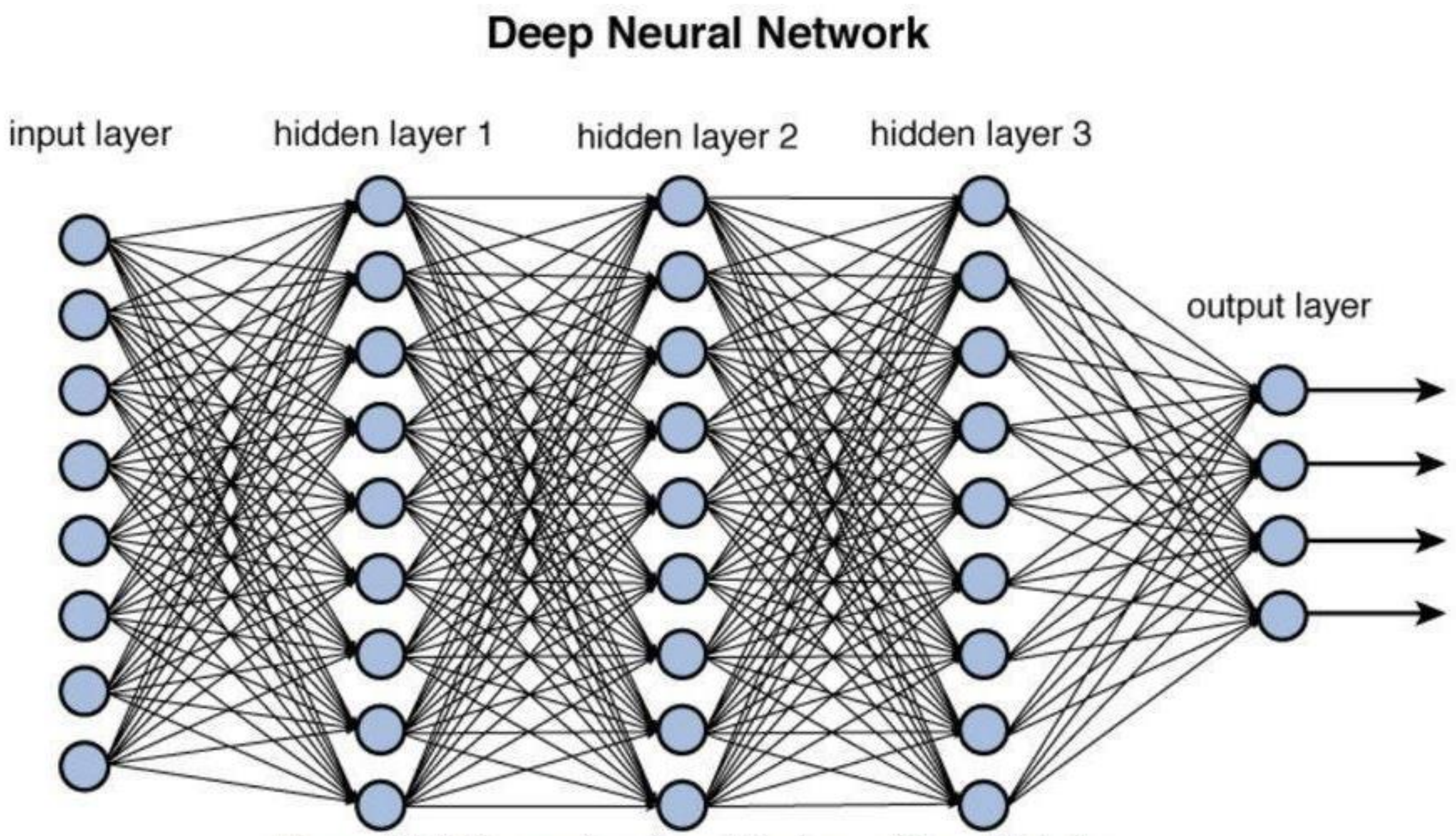


Figure 12.2 Deep network architecture with multiple layers.

Why is this so challenging?

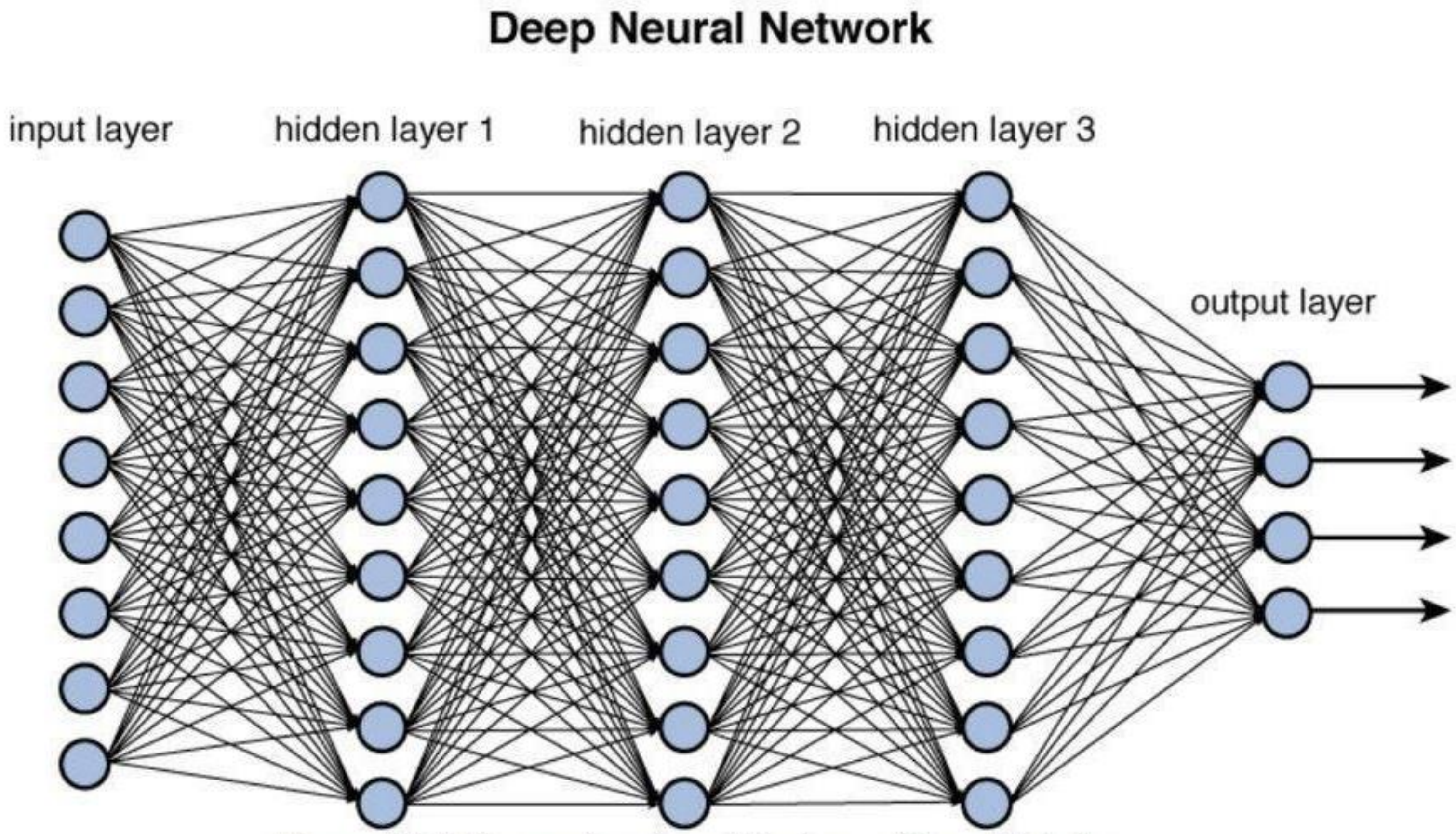


Figure 12.2 Deep network architecture with multiple layers.

Backprop Revisited

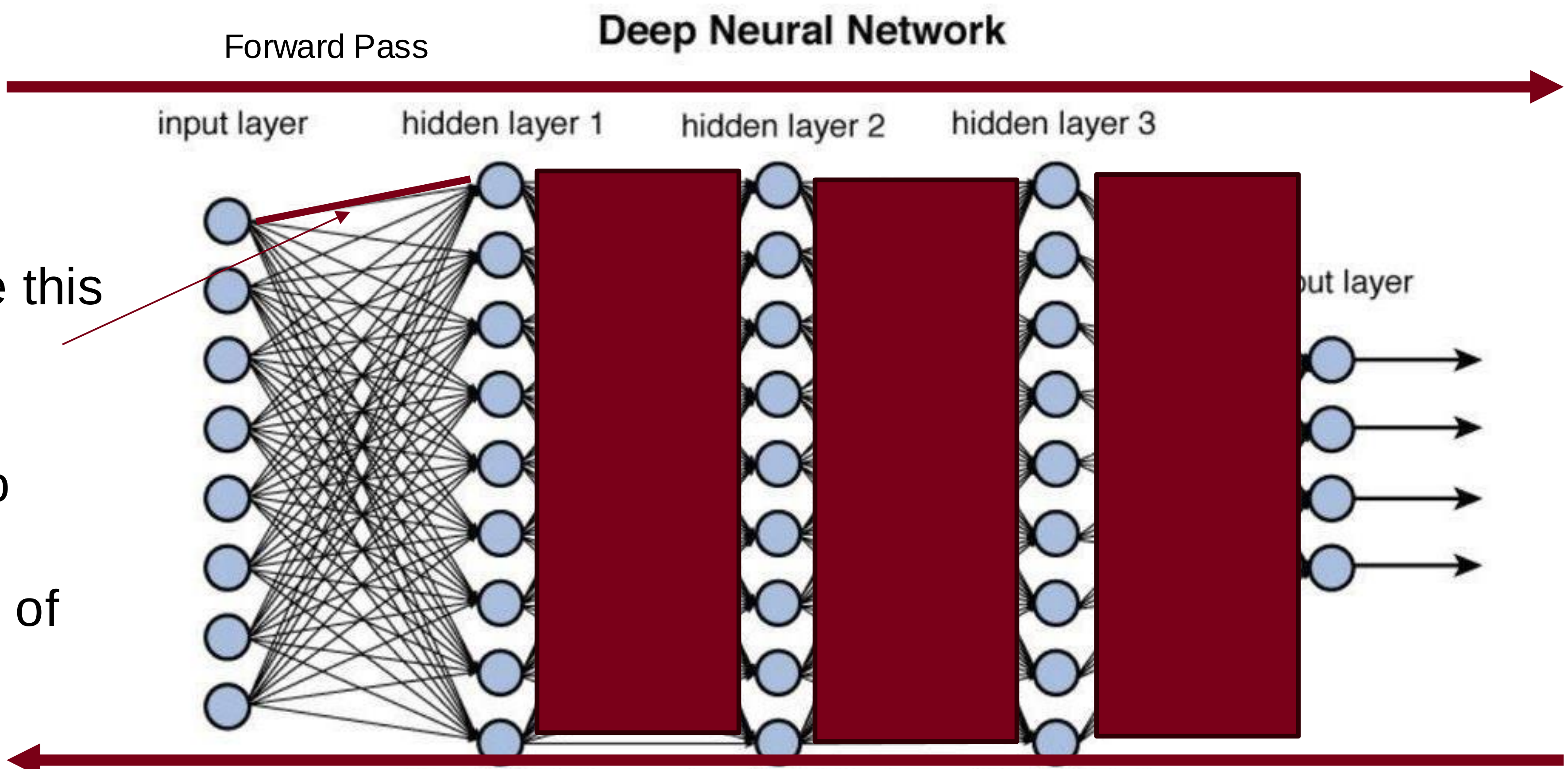


Figure 12.2 Deep network architecture with multiple layers.

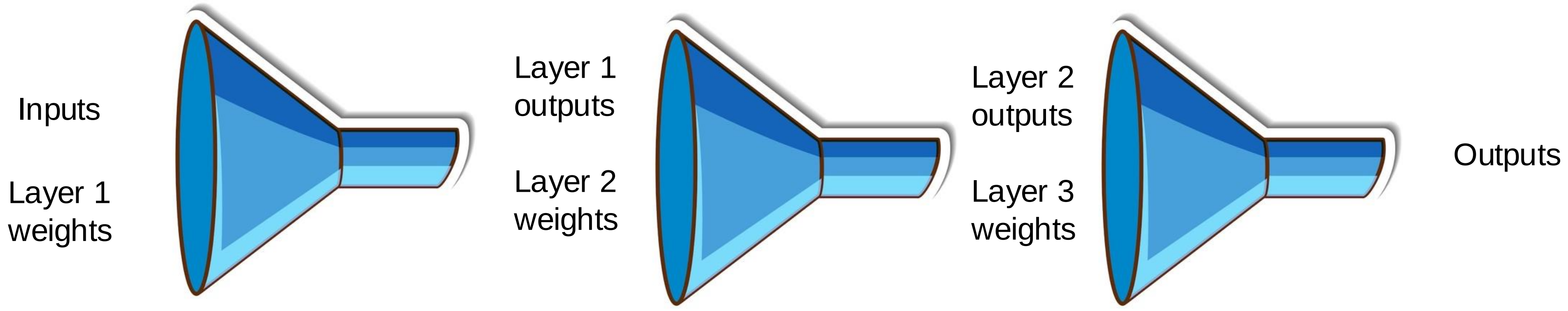
To update this weight…

we have to propagate through all of this

$L(y, \hat{y})$

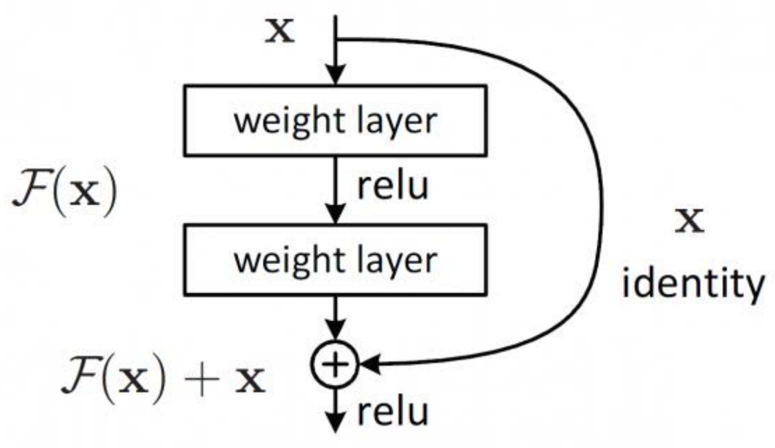
Analogy #1: A Game of Telephone

Analogy #2: A funnel of information



Linearization Solves This

- We need a better way to reduce the number of operations performed between our weights and our loss function (Residual connections)
- 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)

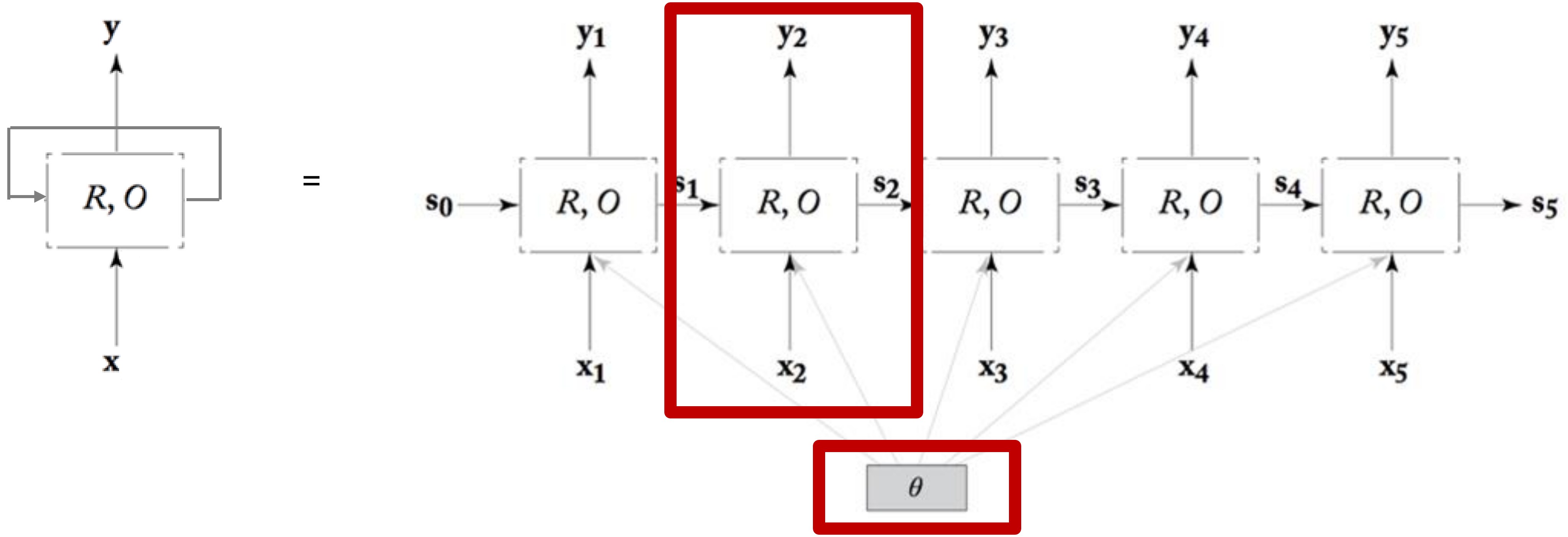


Outline

- ❑ 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
- ❑ Teaser: Transformer-based LMs
- ❑ Why language models are useful?

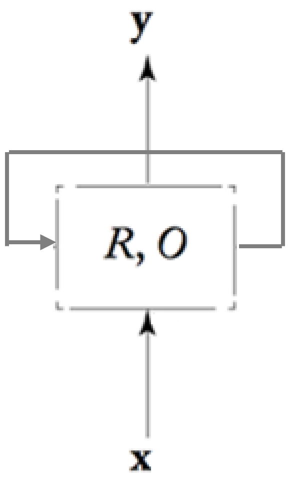
Recurrent Neural Network (RNN)

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



Goldberg 2017

Recurrent Neural Network



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

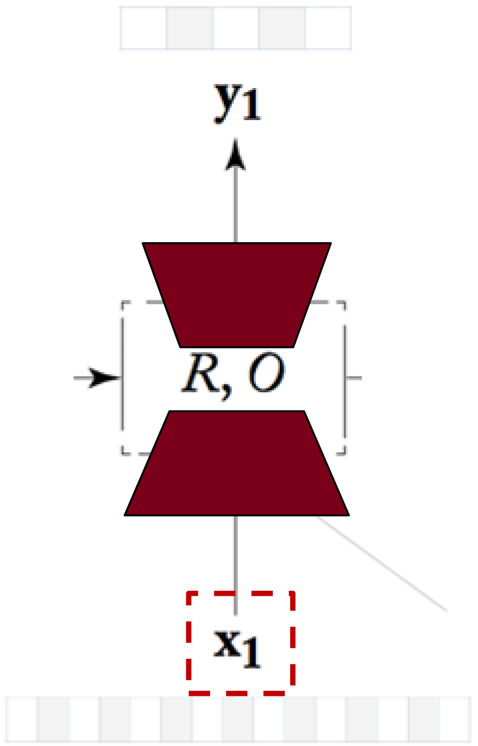
RNN: $P(w) = P(w_i | context)$
$= softmax\,(W \cdot \boldsymbol{h_i})$

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

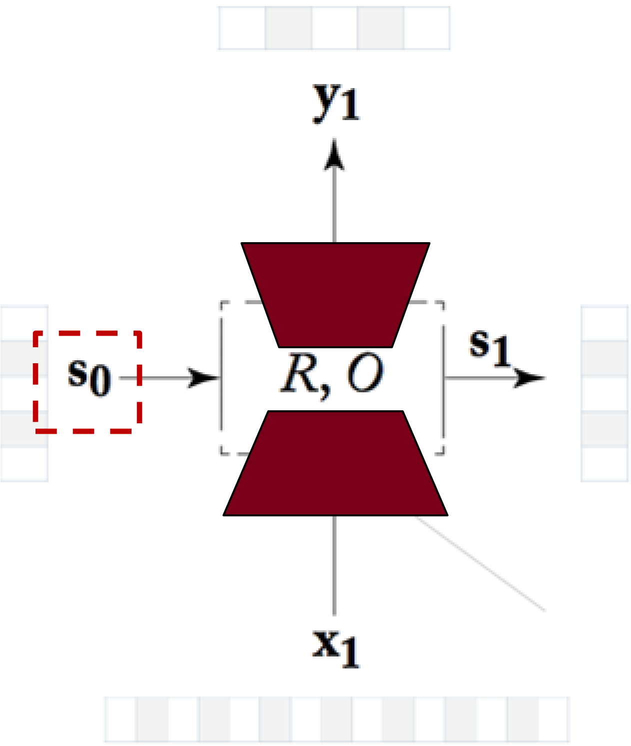


Recurrent Neural Network

- Each time set has two inputs:

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

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

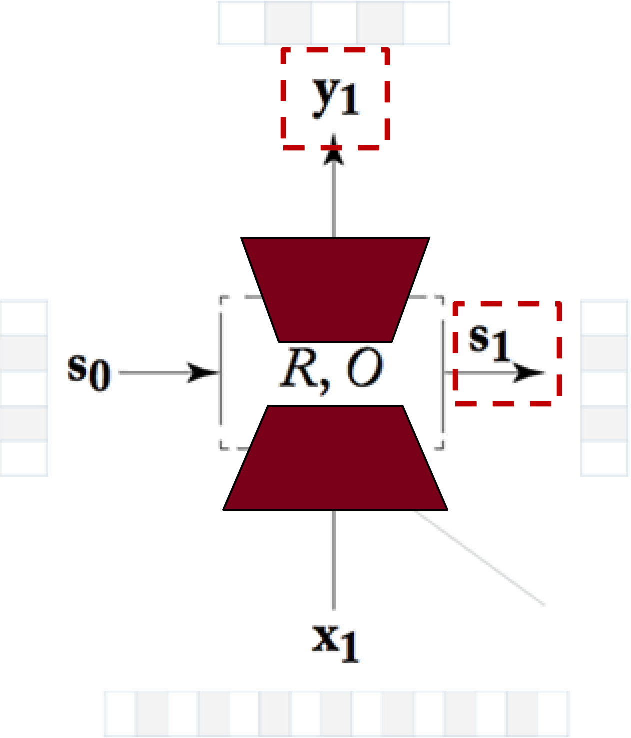


Recurrent Neural Network

- Each time set has two outputs:

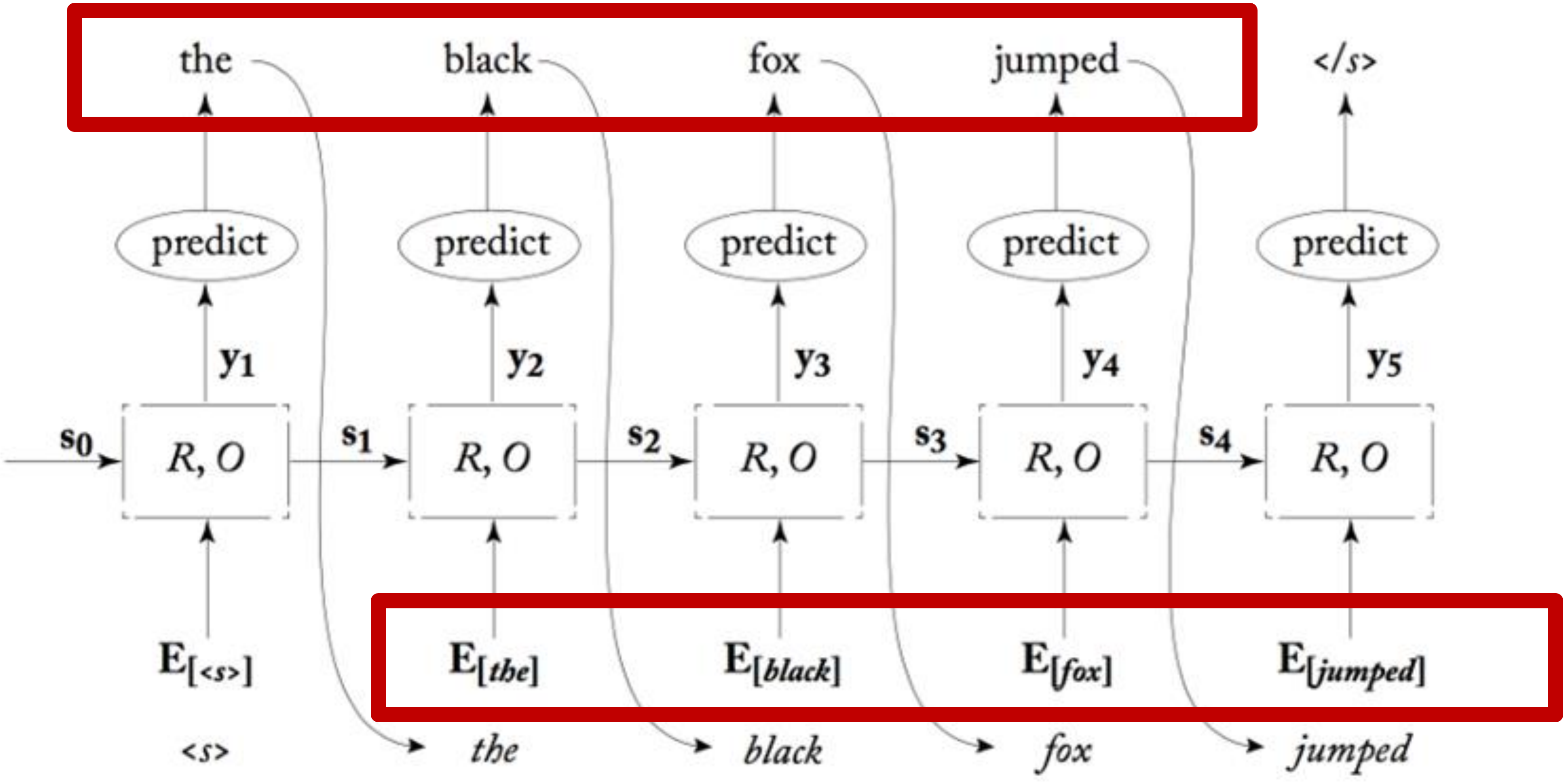
- $S_i = R\ (X_i, S_{i-1})$
  - R computes the output state as a function of the *current input* and *previous state*

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



RNN Training

output as
shifted by one

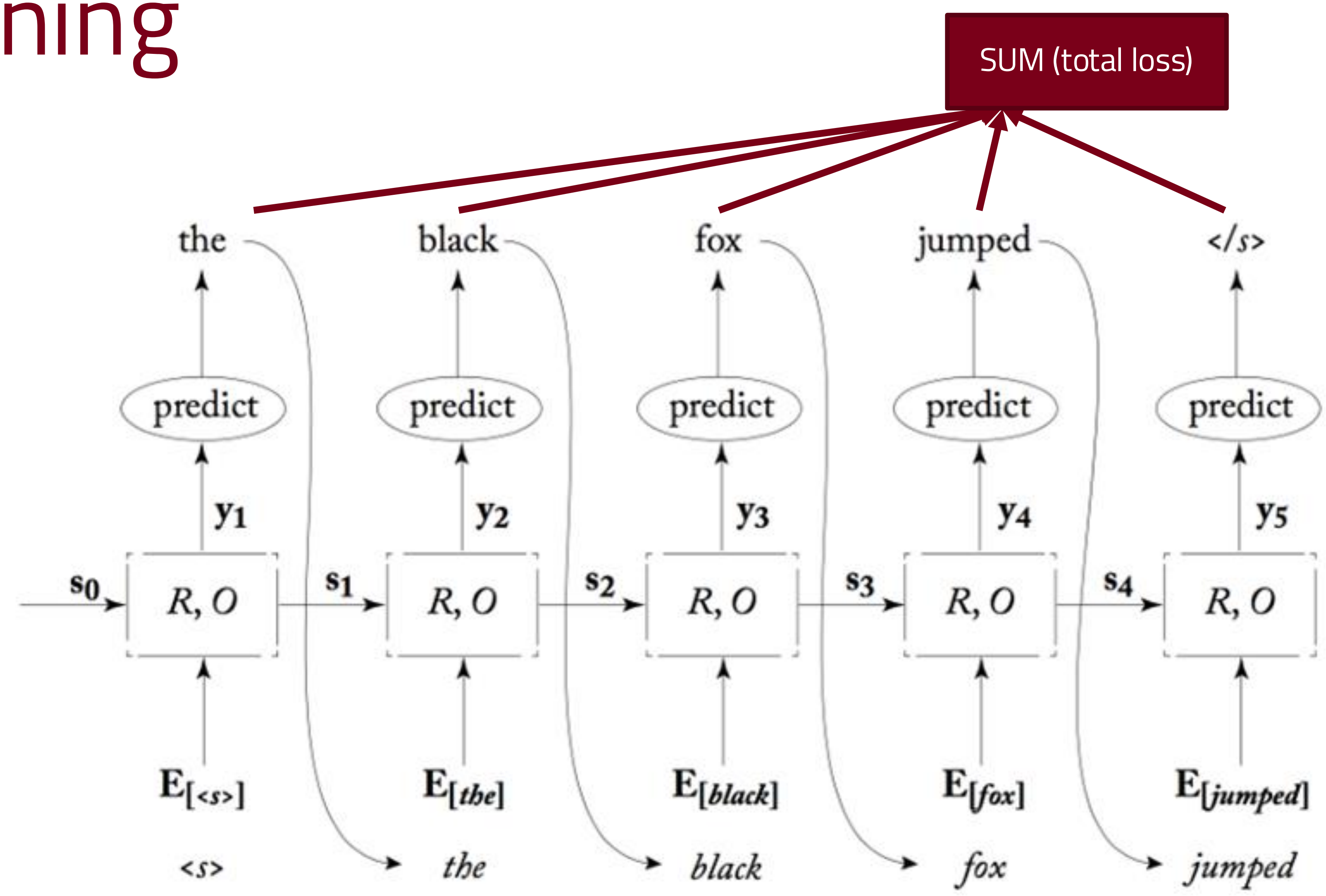


sequence of
words

RNN Training

output as shifted by one

sequence of words

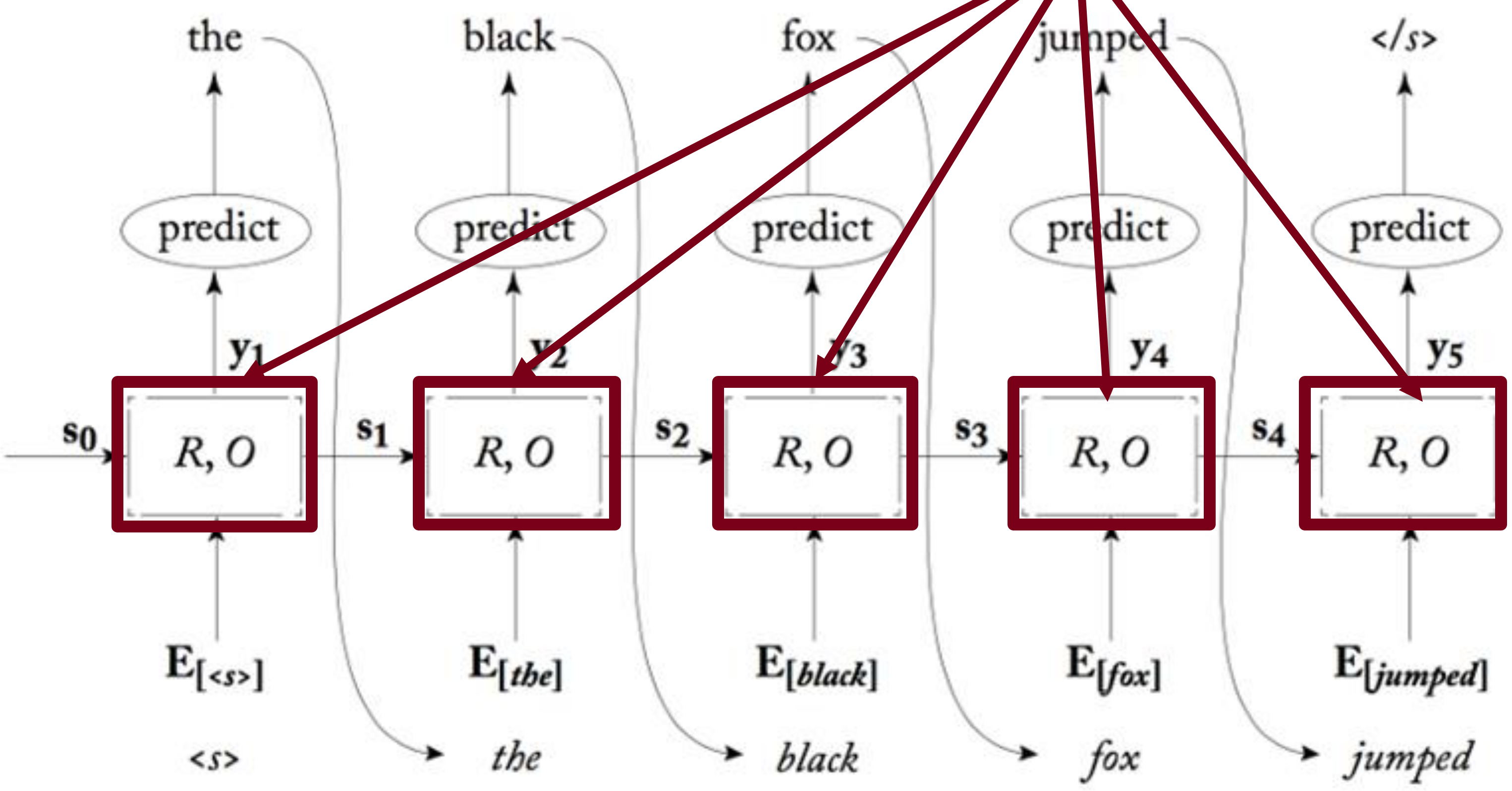


RNN Training

Parameters are shared!
Derivatives are accumulated.

output as shifted by one

sequence of words

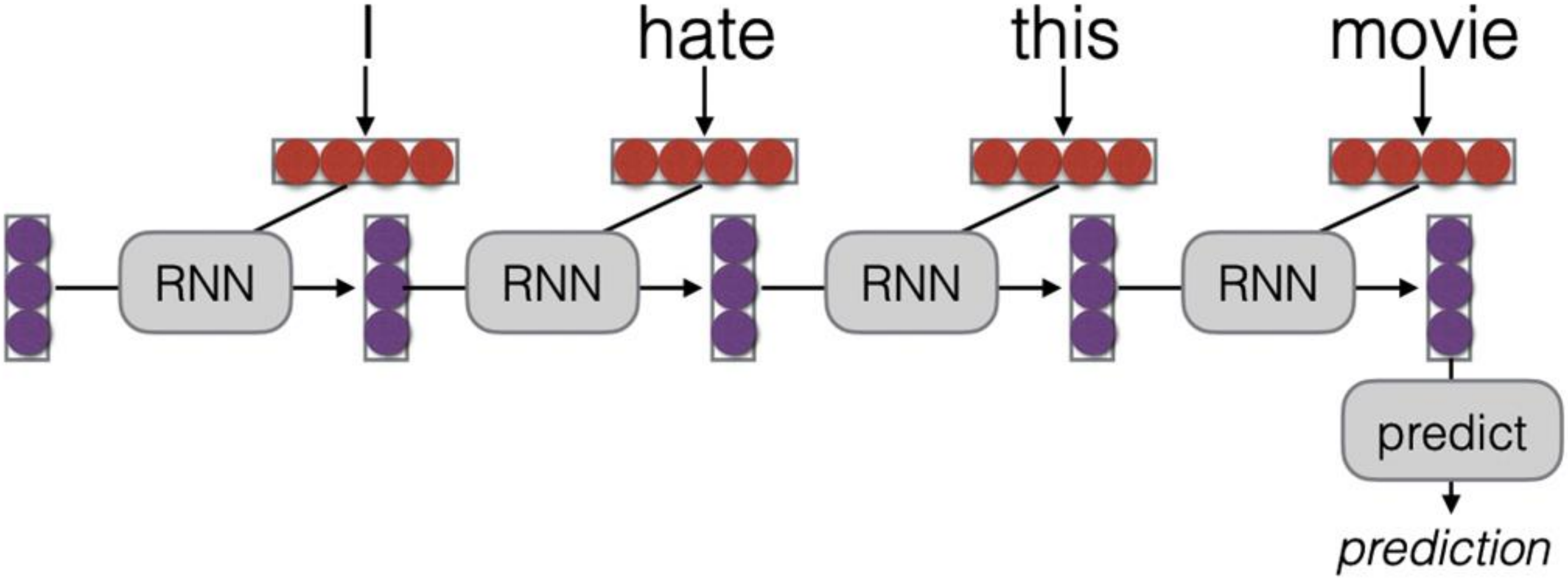


What can RNNs do?

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

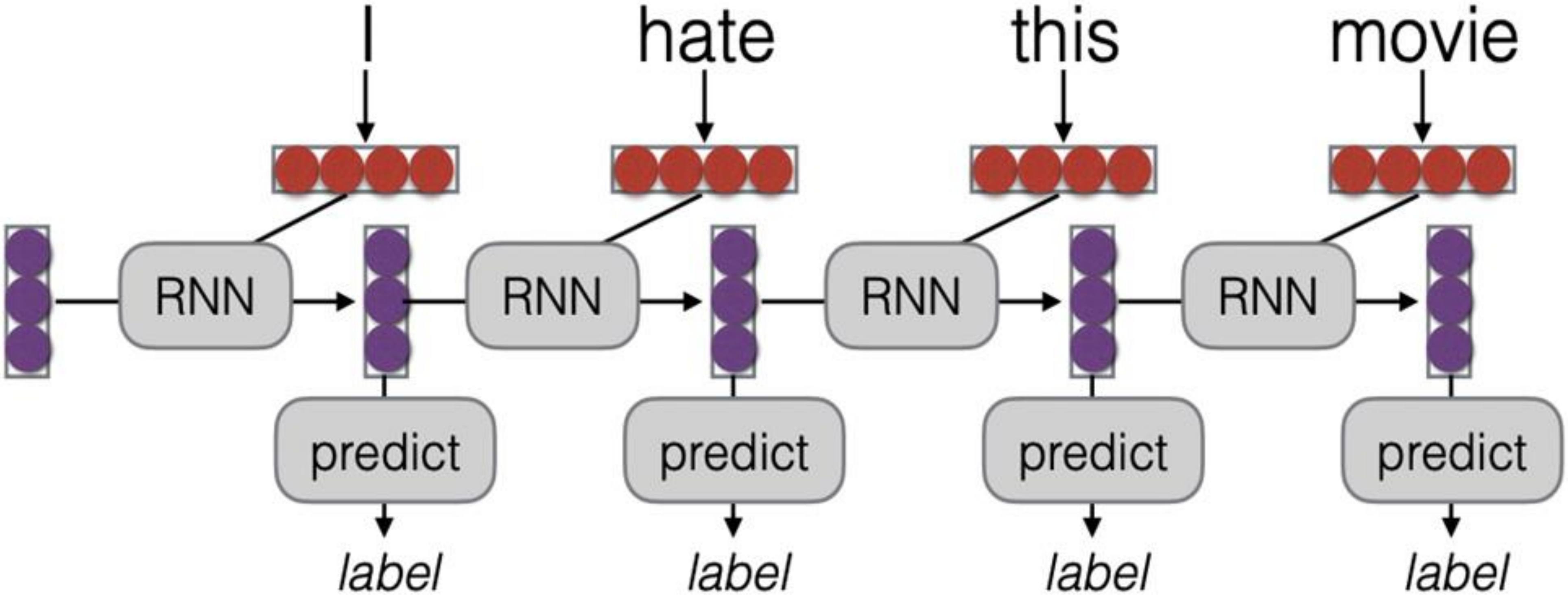
Representing Sentences

- ❑ Sentence classification
- ❑ Conditioned generation



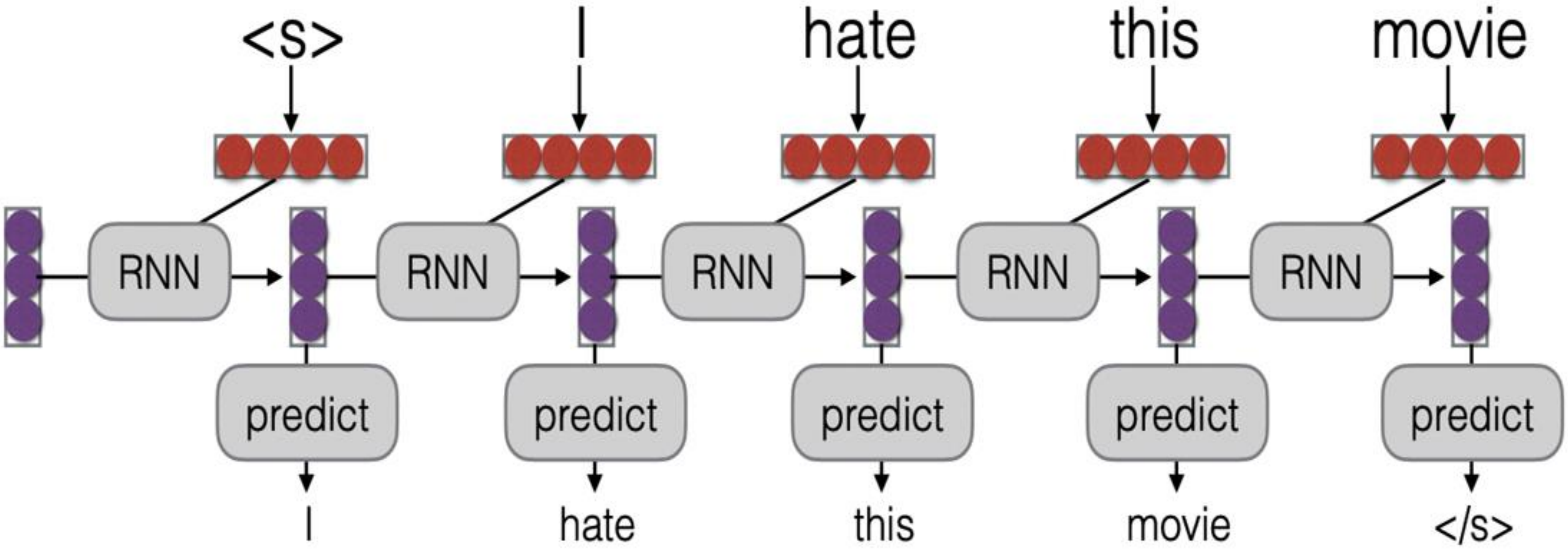
Representing Context within Sentence

- ❑ Tagging
- ❑ Language modeling

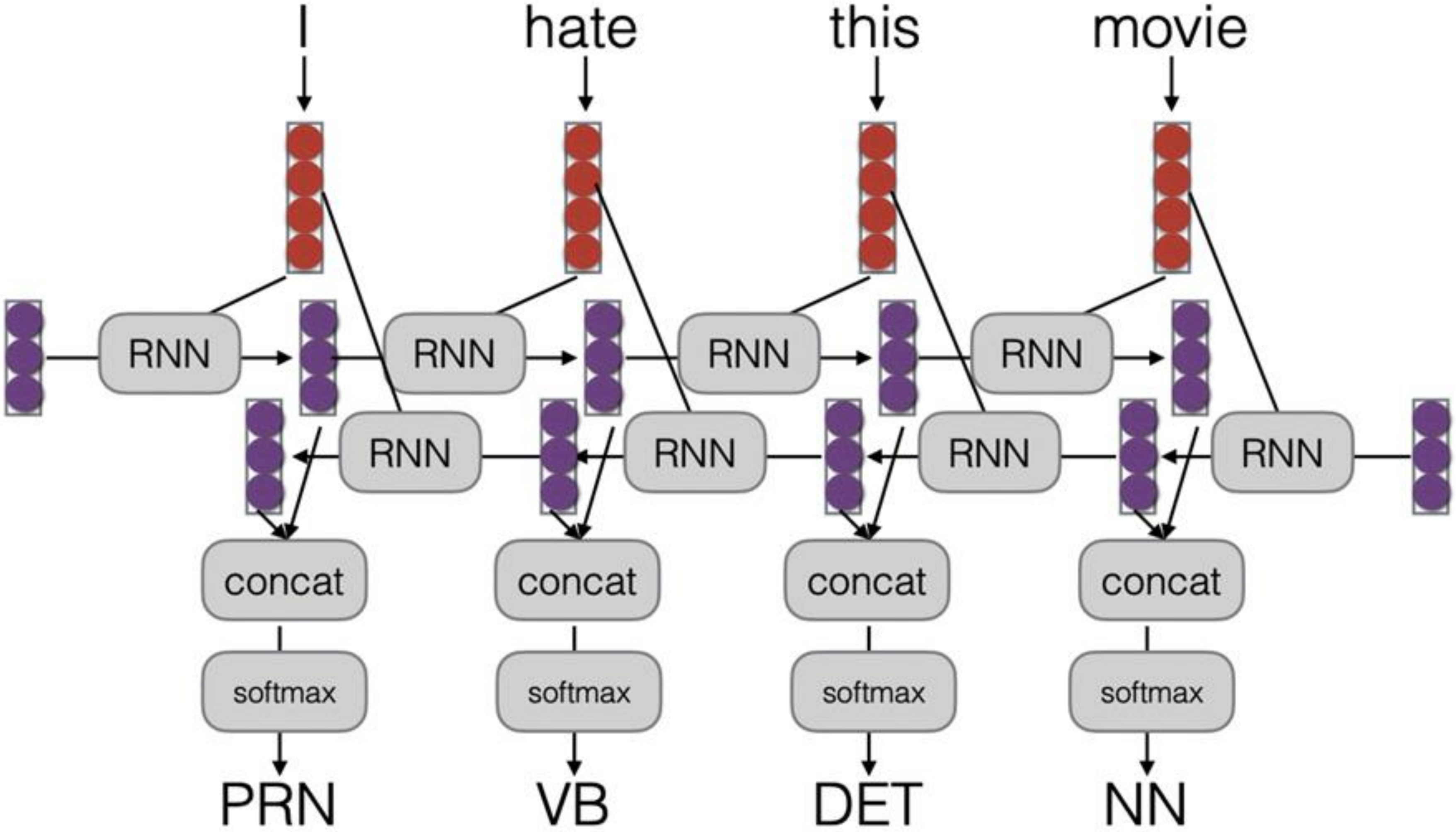


e.g., Language Modeling

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



e.g., POS Tagging with Bi-RNNs

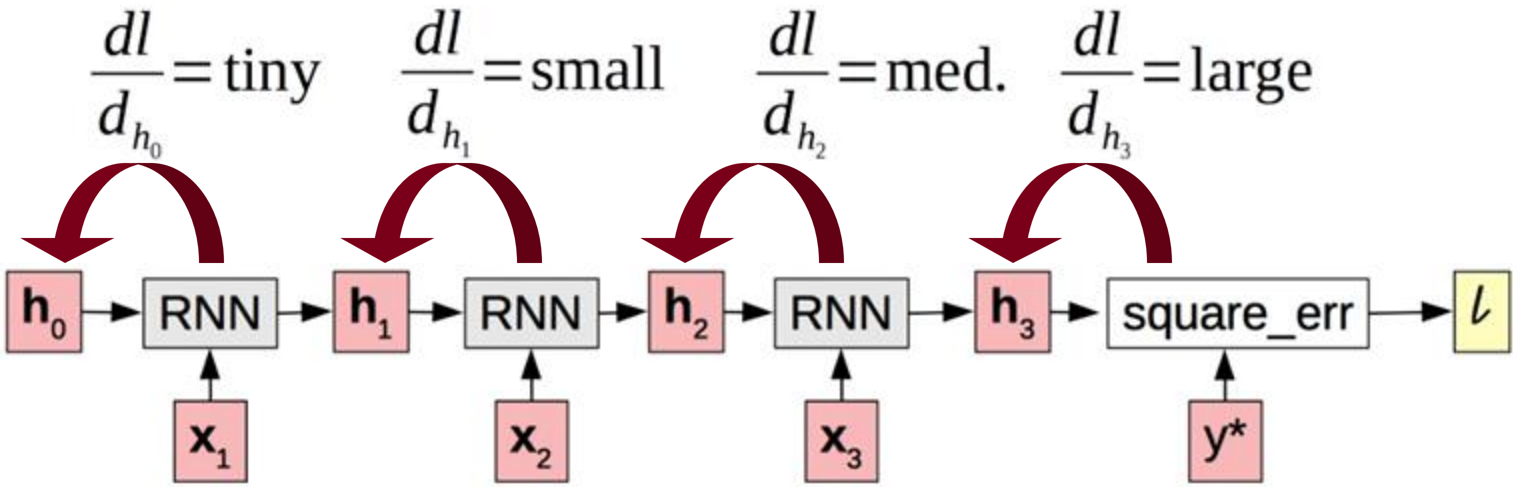


Outline

- ❑ 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?

Vanishing Gradient

- Gradients decrease as they get pushed back



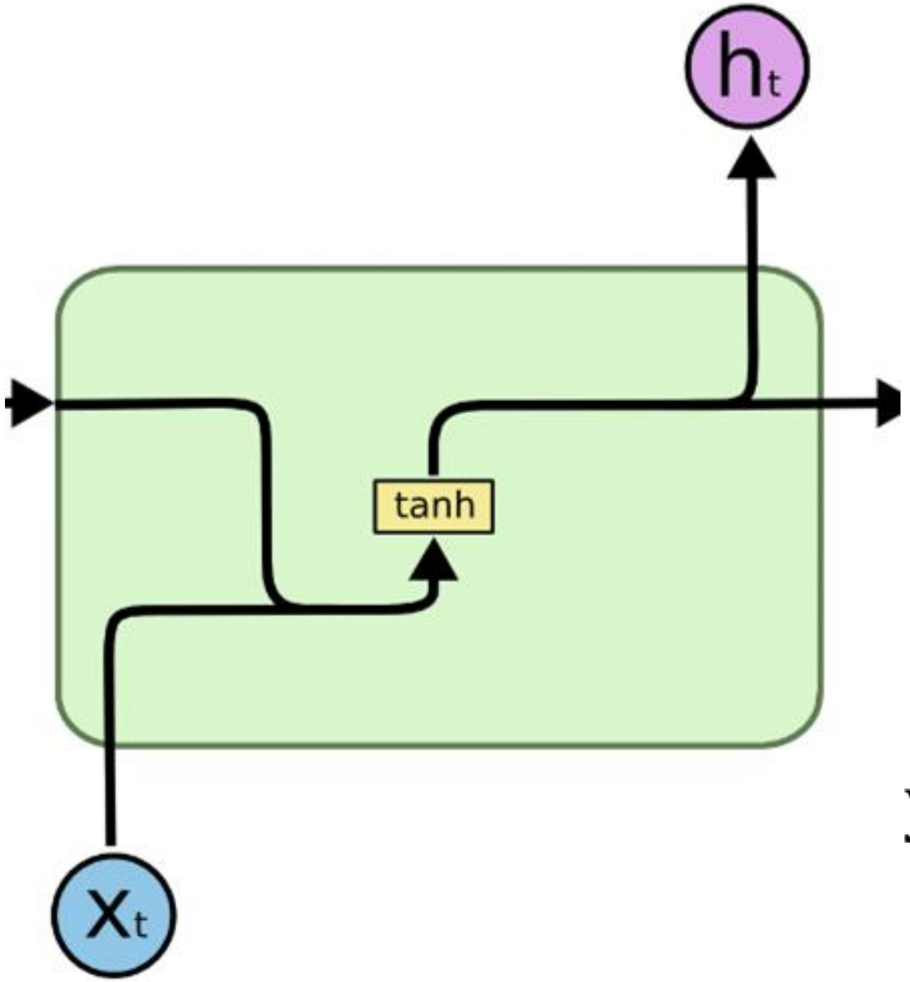
- Why? "Squashed" by non-linearities or small weights in matrices

A Solution: Long Short-term Memory (LSTM)

(Hochreiter and Schmidhuber 1997)

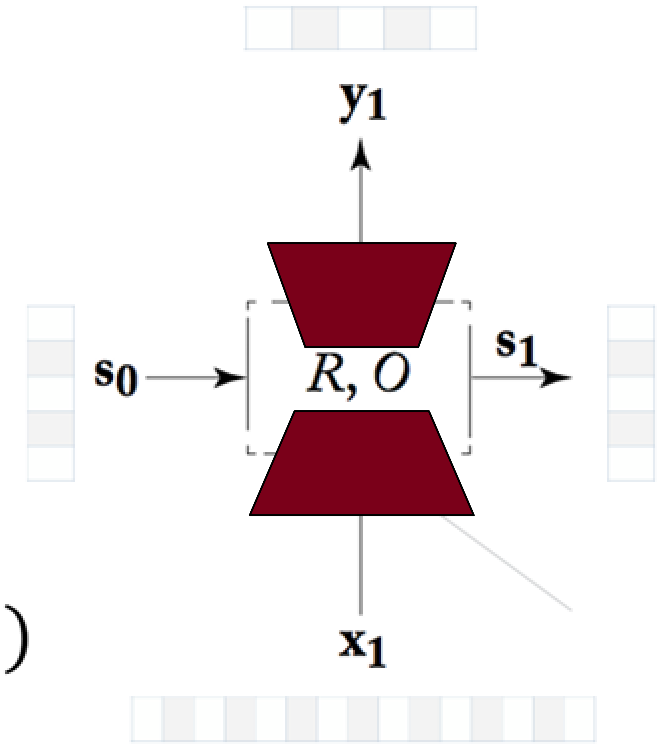
- Make **additive connections** between time steps
- Addition does not modify the gradient, no vanishing
- **Gates** to control the information flow

RNN Structure



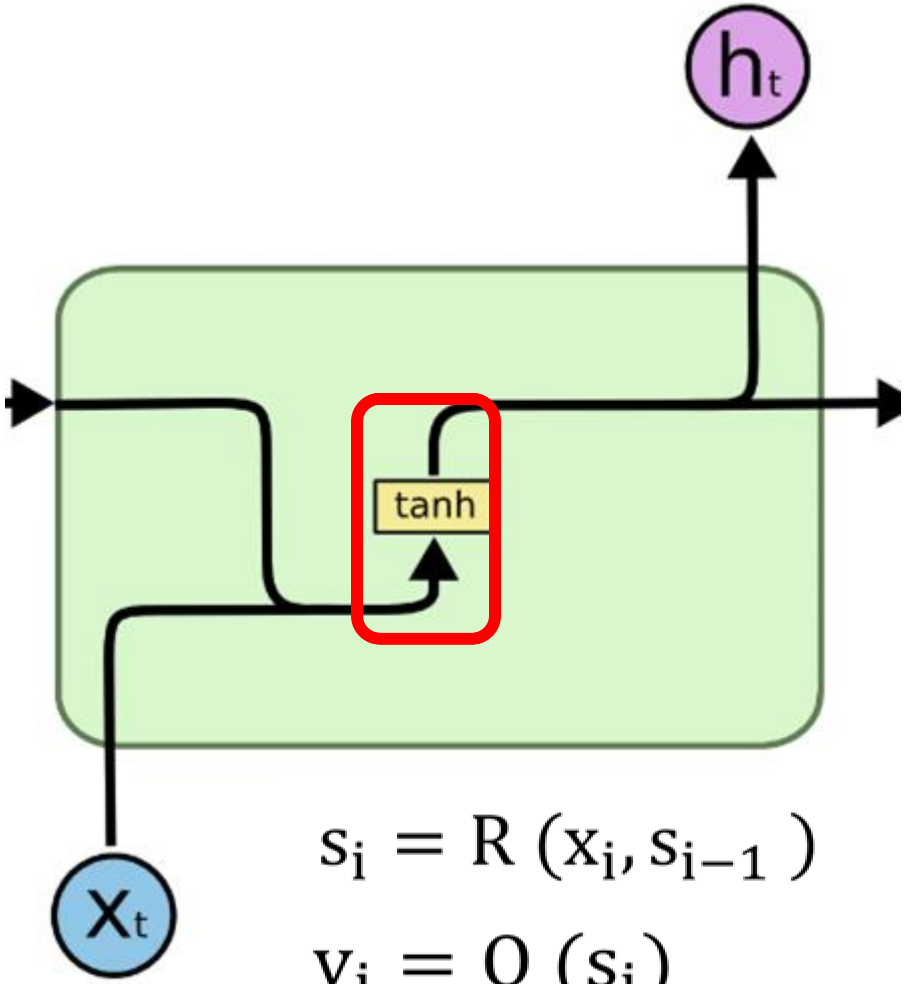
$$s_i = R(x_i, s_{i-1})$$

$$y_i = O(s_i)$$



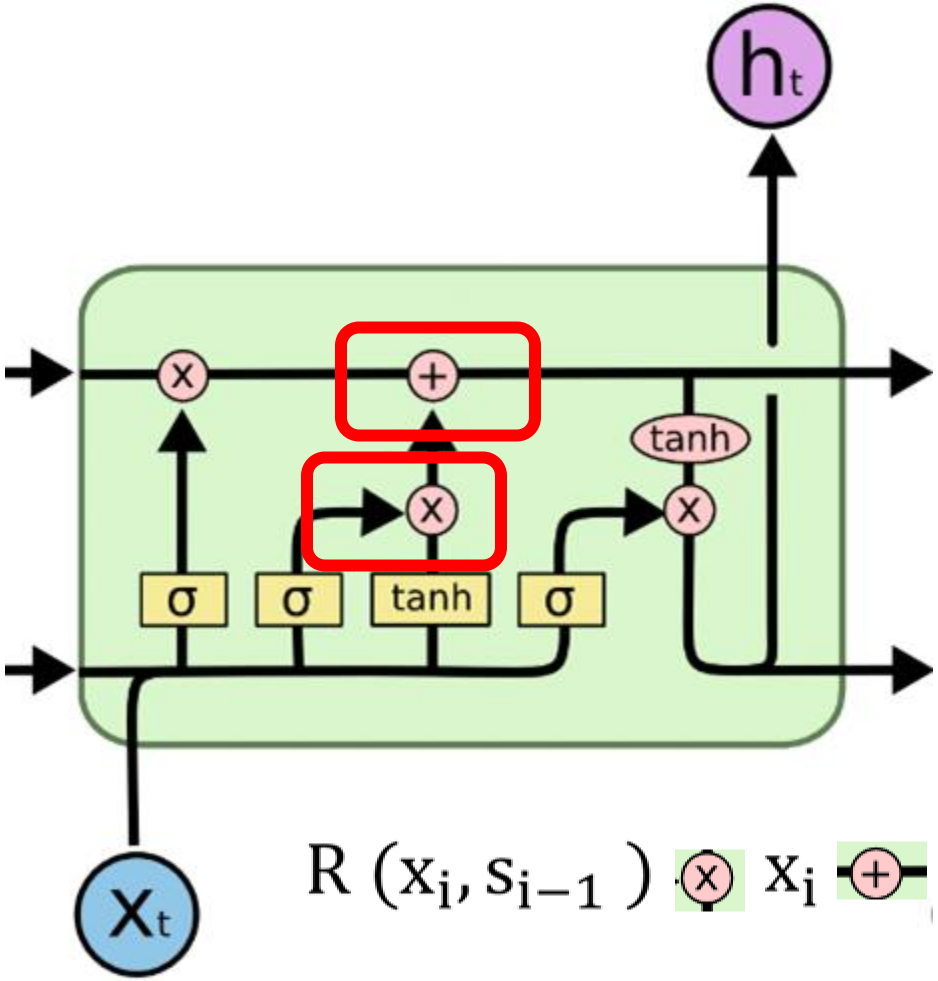
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN vs LSTM Structure



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

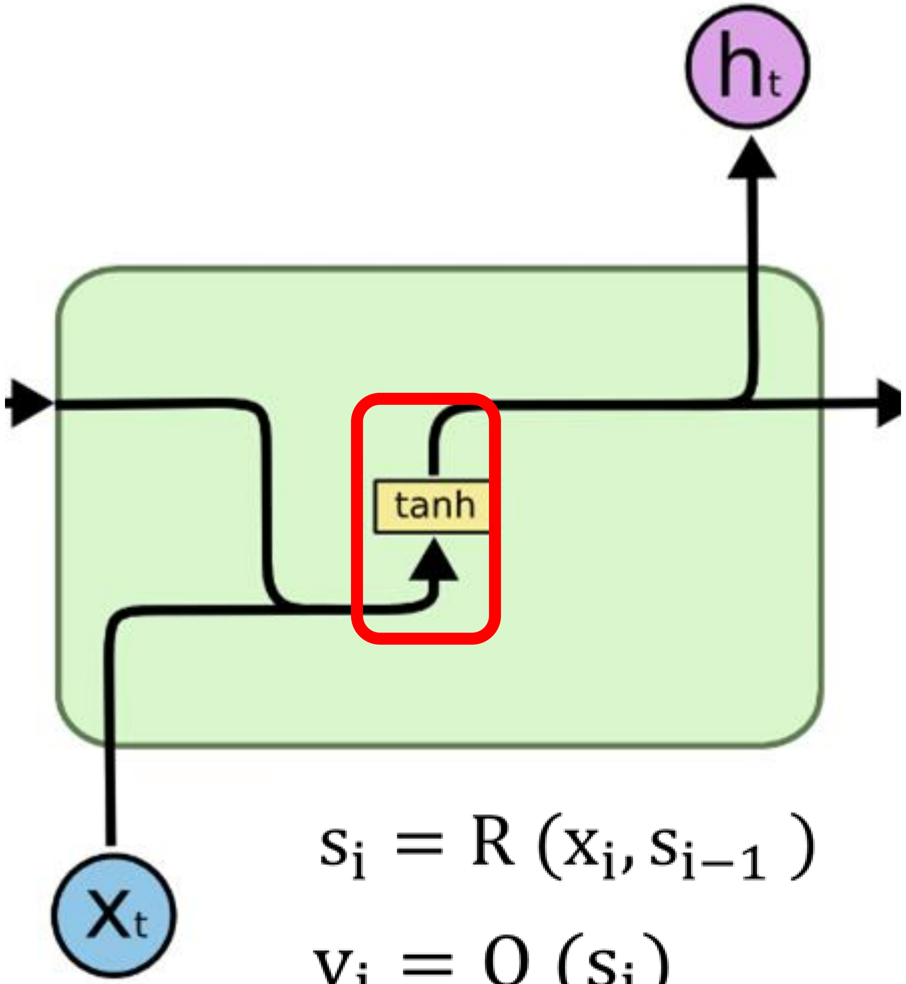
$$y_i = O\,(s_i)$$



$$R\,(x_i, s_{i-1}) \otimes x_i \oplus s_{i-1}$$

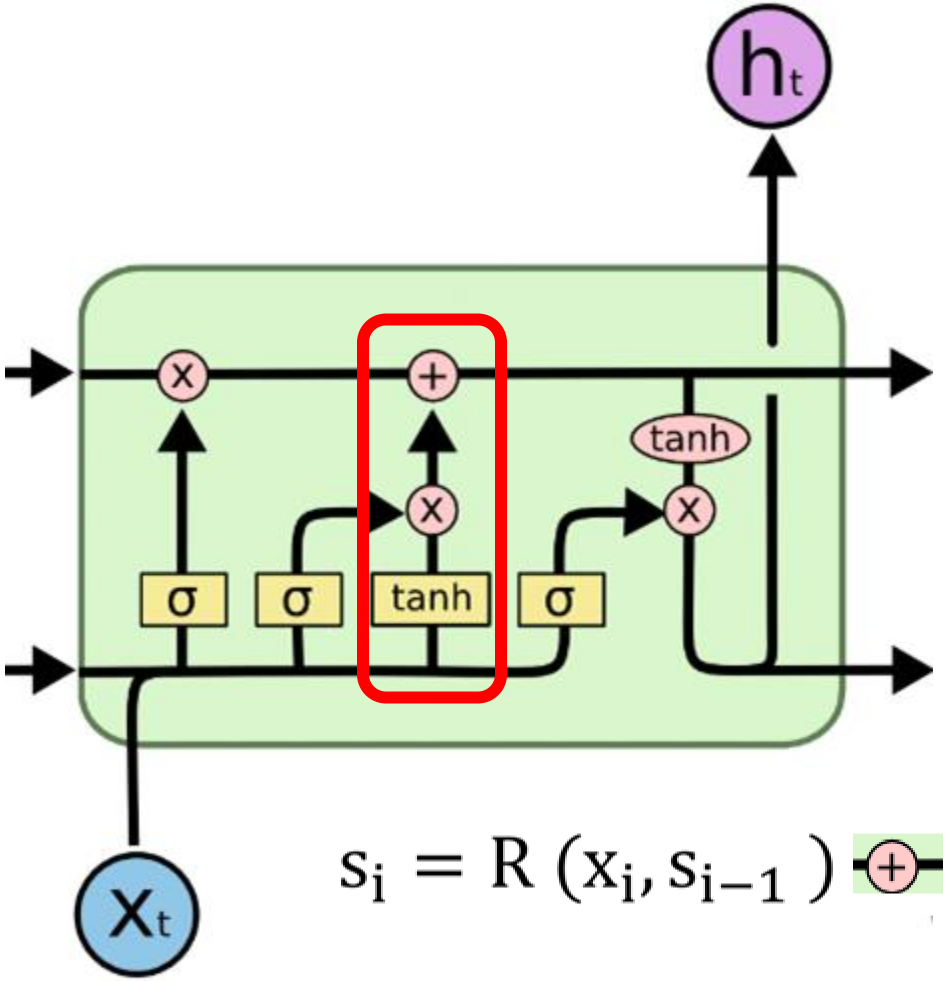
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN vs LSTM Structure



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

$$y_i = O(s_i)$$



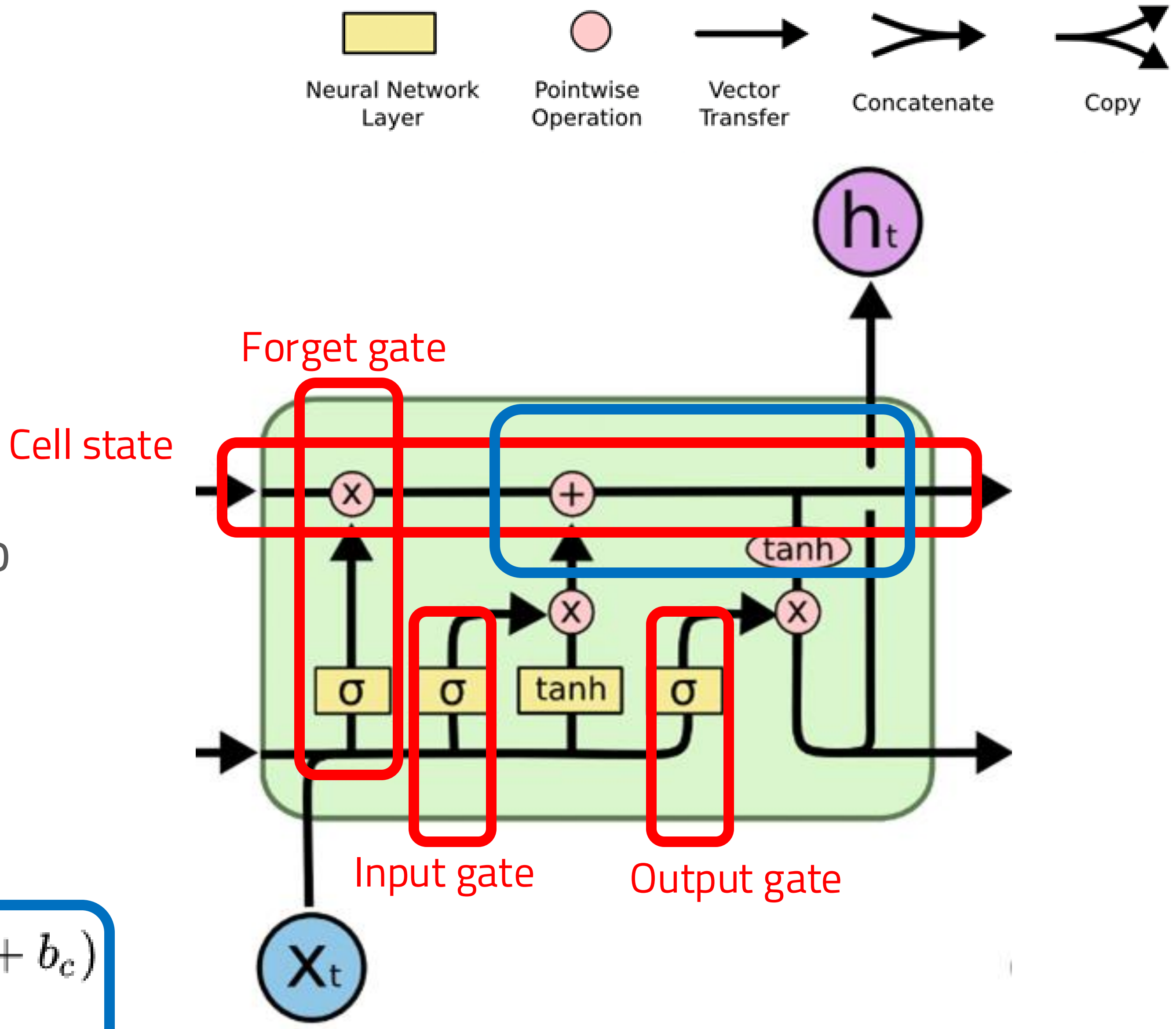
$$s_i = R(x_i, s_{i-1}) \oplus s_{i-1}$$

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?

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

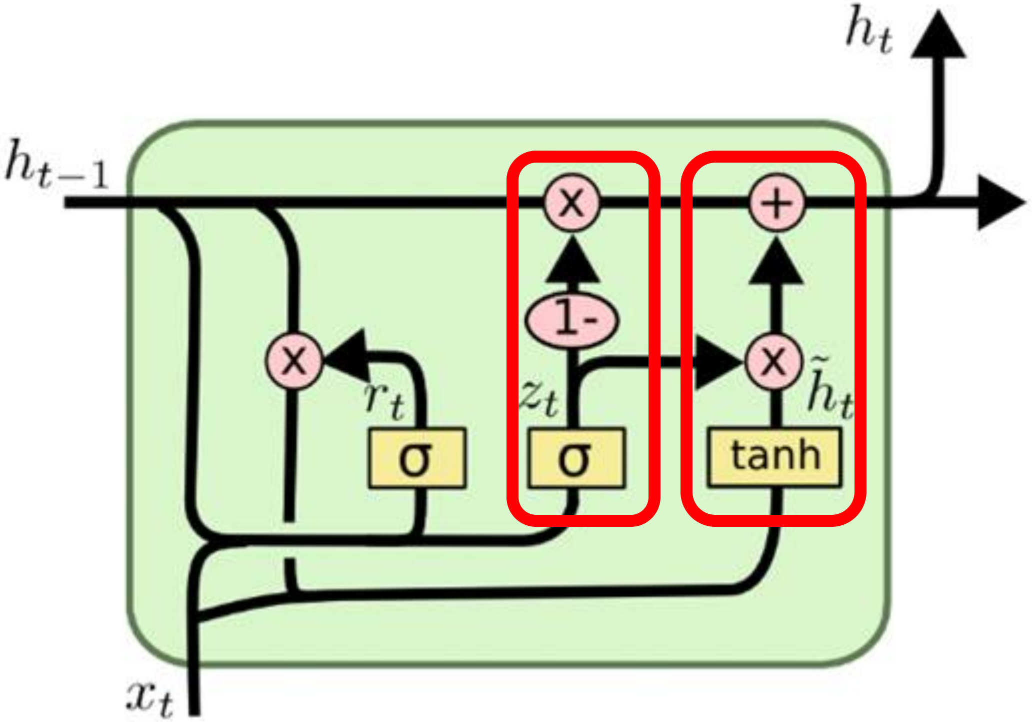


http://colah.github.io/posts/2015-08-Understanding-LSTMs/

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



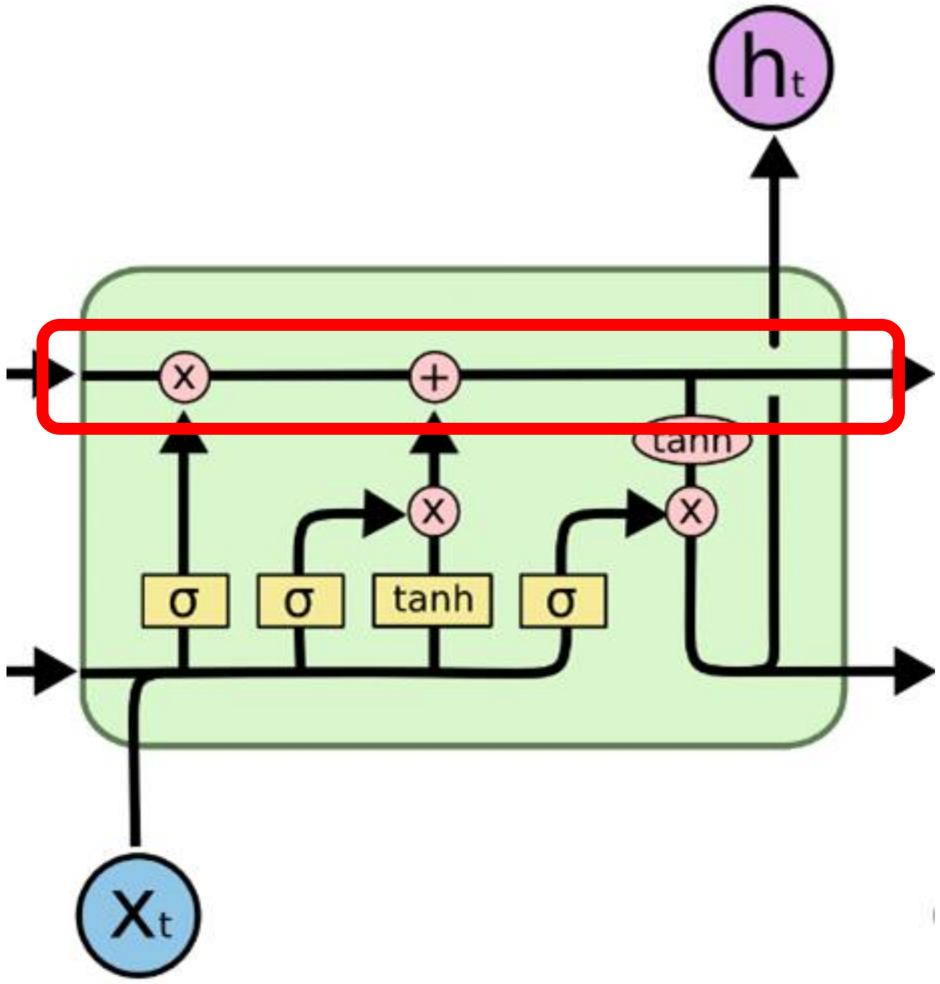
$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$
$$h_t = \boxed{(1 - z_t)} \circ h_{t-1} + \boxed{z_t} \circ \sigma_h(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h)$$

Additive or Non-linear

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

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

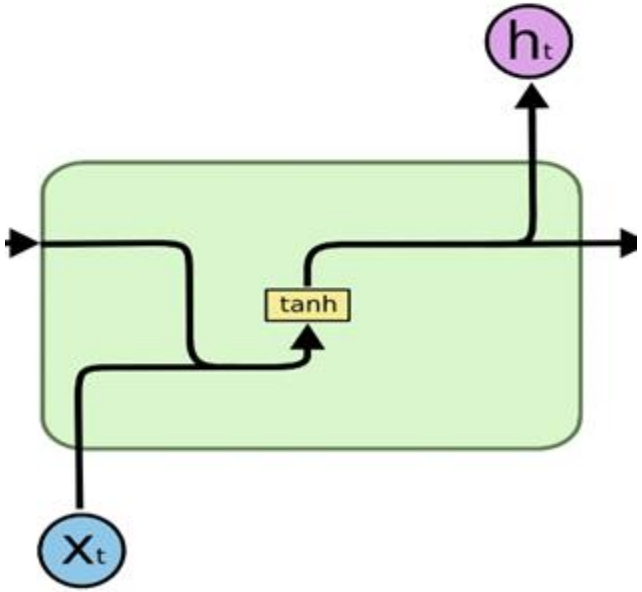
Outline

- ❑ 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?

```python
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
```

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

$$y_i = O\left(s_i\right)$$

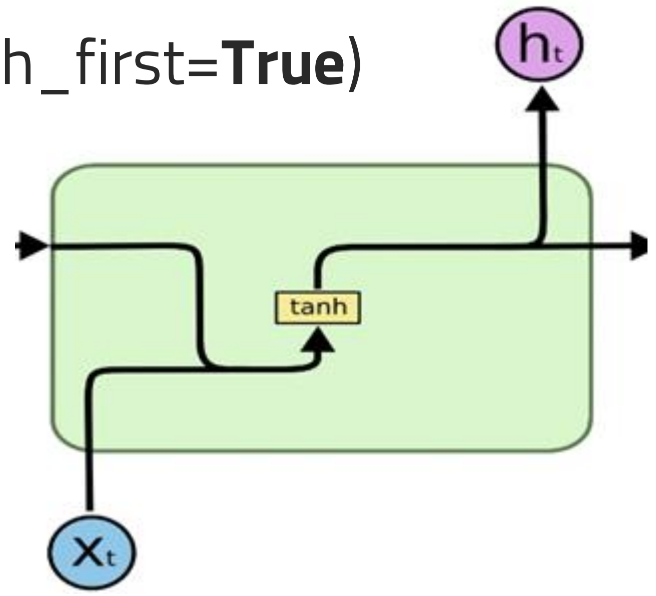


```python
class RNN(nn.Module):
  def __init__(self, input_size, output_size, hidden_dim, n_layers):
        super(RNN, self).__init__()
        ...
        self.rnn = nn.RNN(input_size, hidden_dim, n_layers, batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_size)

  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
```

PyTorch

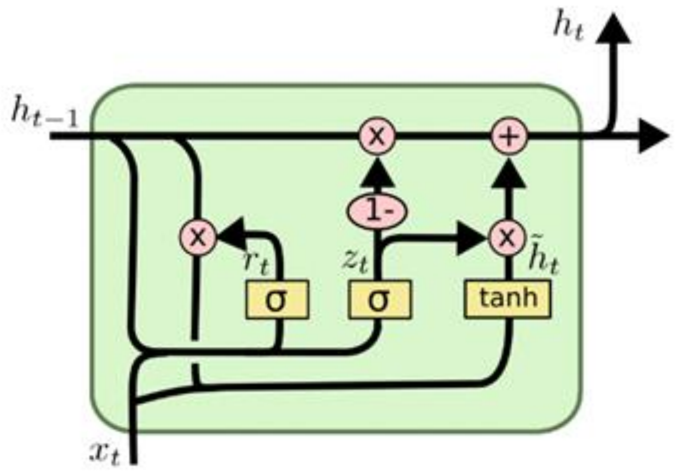
$$s_i = R(x_i, s_{i-1})$$

$$y_i = O(s_i)$$



 # 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))
```



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

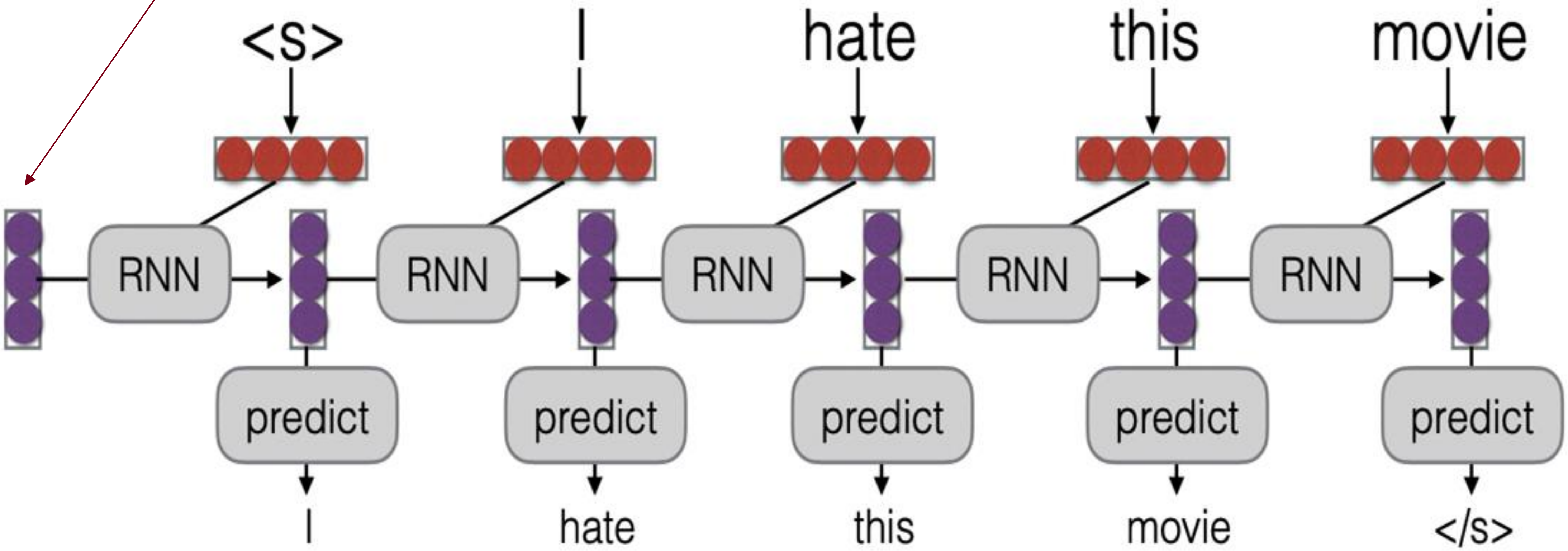
Outline

- ❑ 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?

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

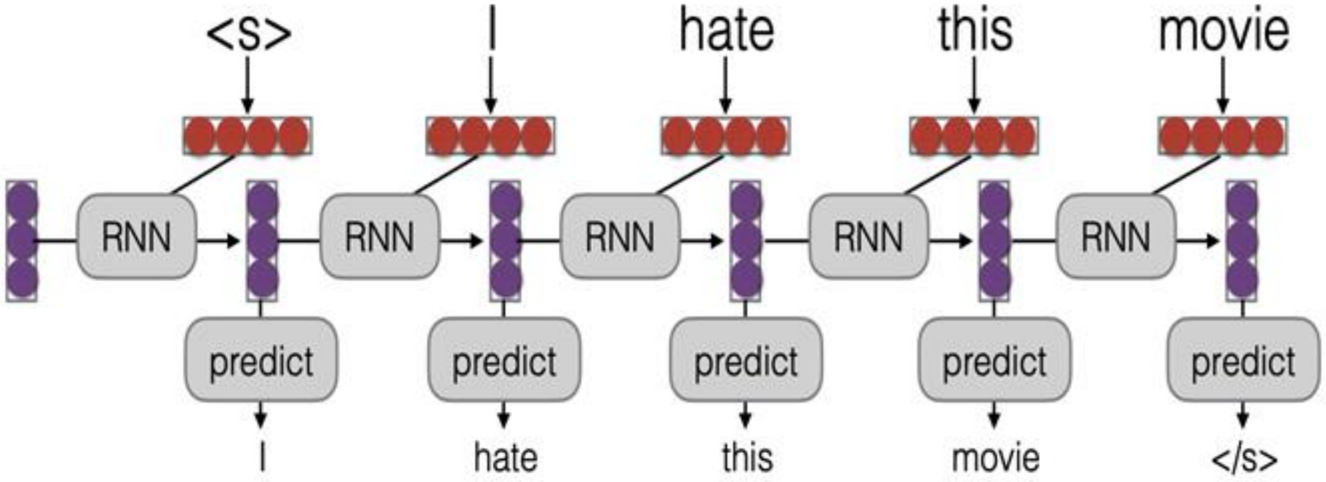
RNN (decoder) for language modeling

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



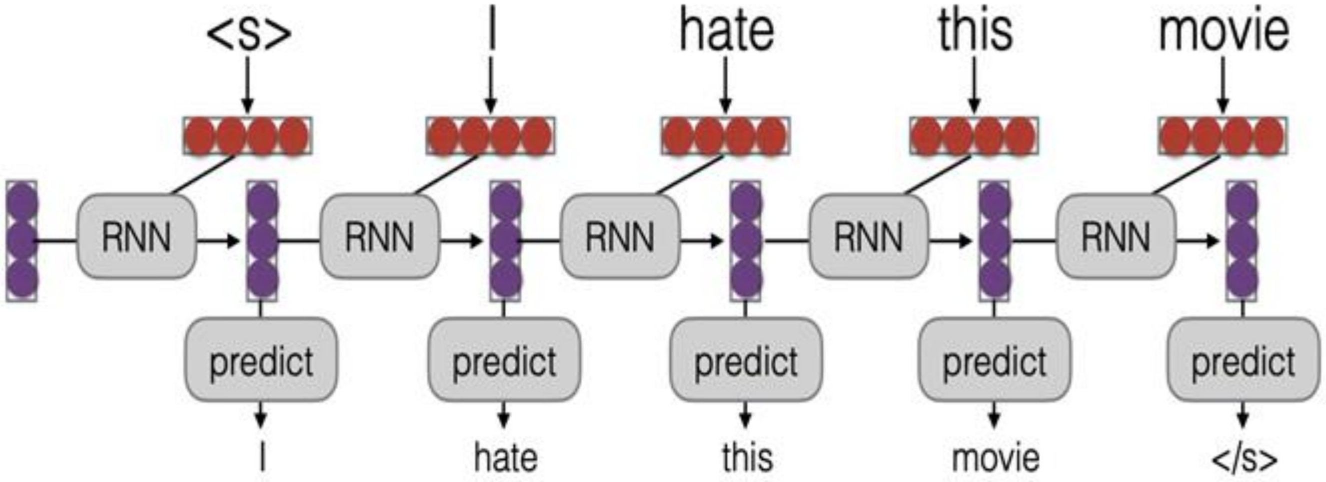
RNN (decoder) for language modeling

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



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

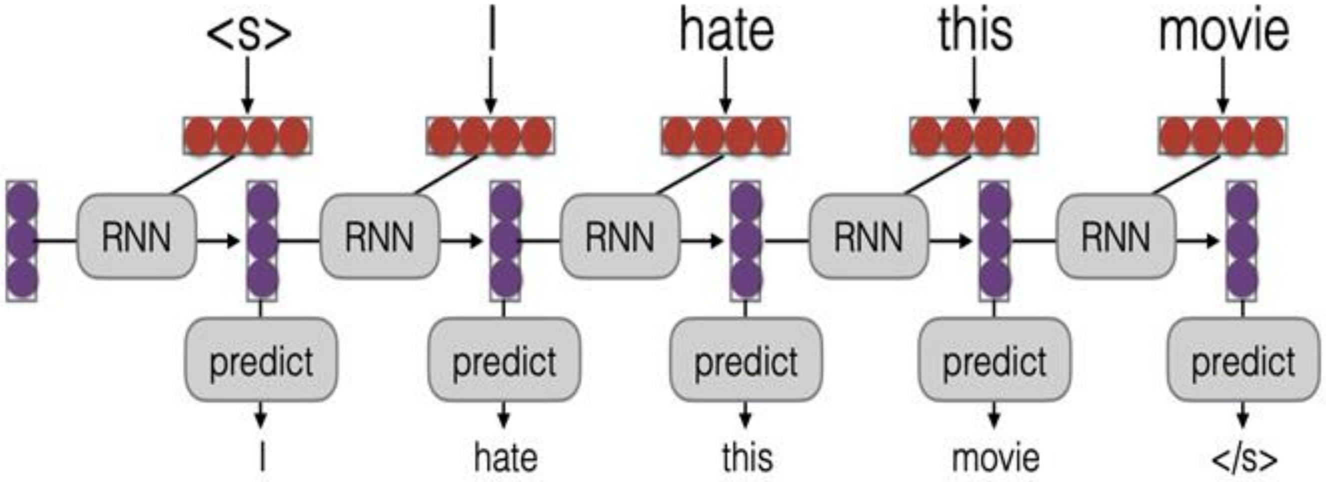
“나는 이 영화가 싫어요”
“Odio esta película”



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

"나는 이 영화가 싫어요"
"Odio esta película"

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



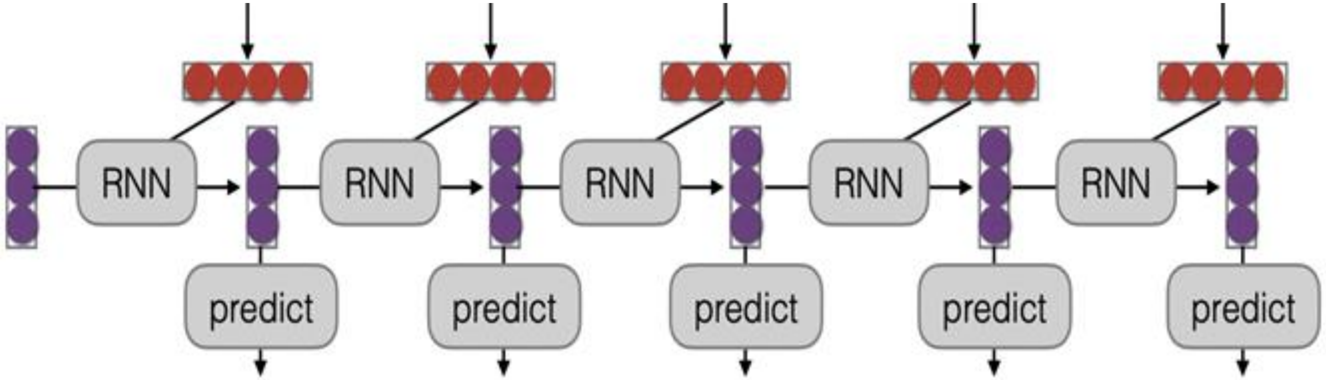
RNN (encoder) - RNN (decoder) for question answering

"나는 이 영화가 싫어요"
"Odio esta película"

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

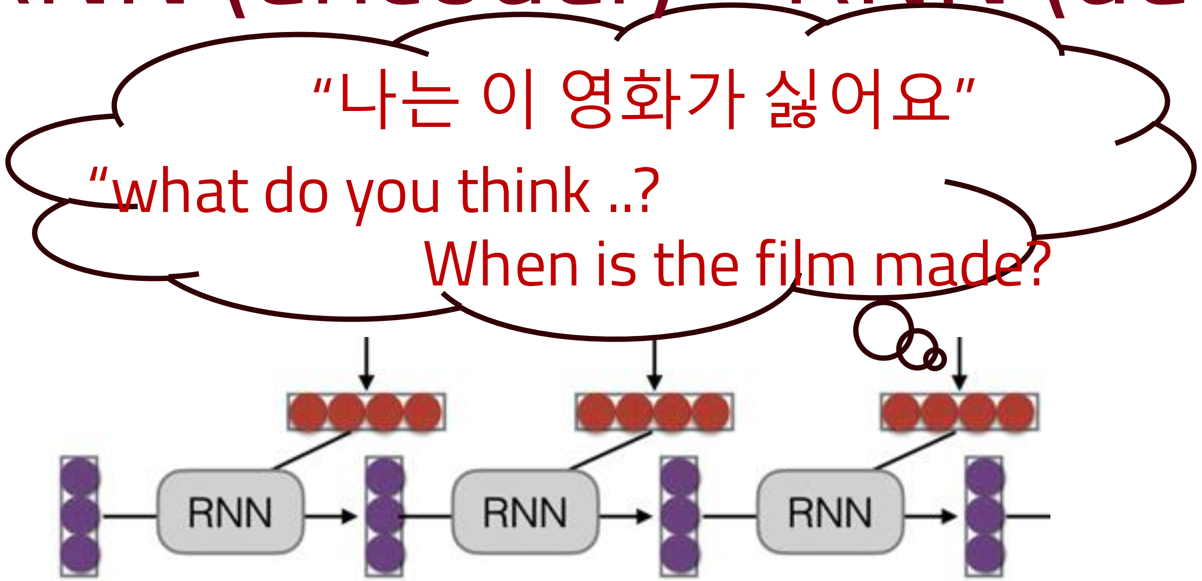
When is the film made?

<s> This film is made in 1997



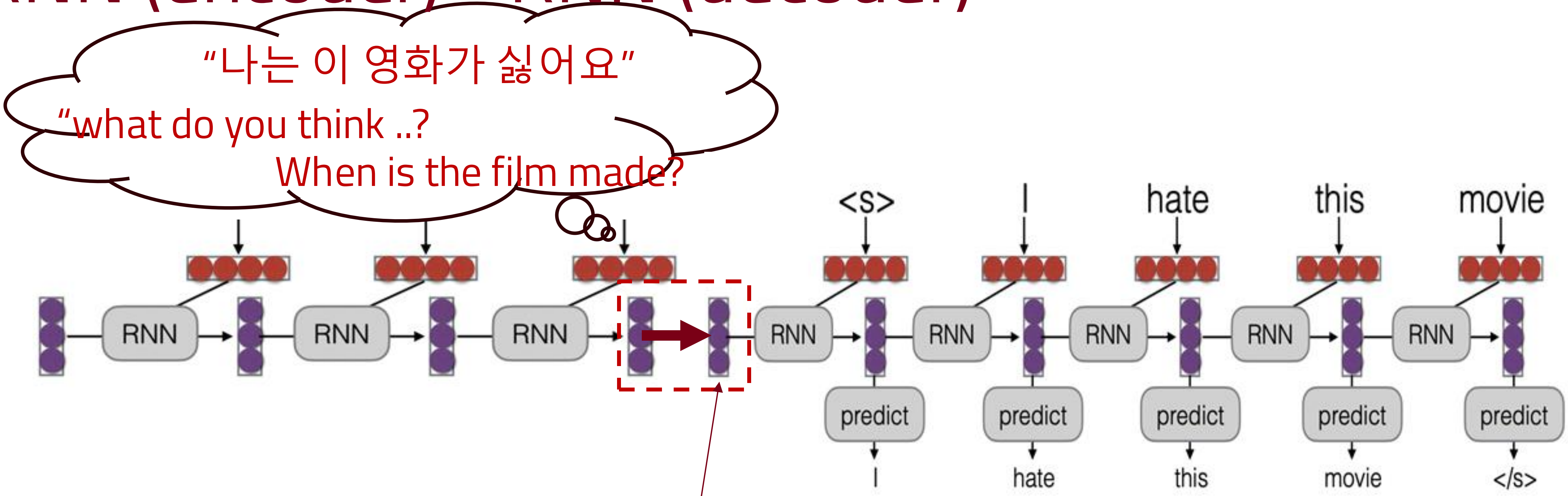
This film is made in 1997

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



Encoder: encoding input sequence

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

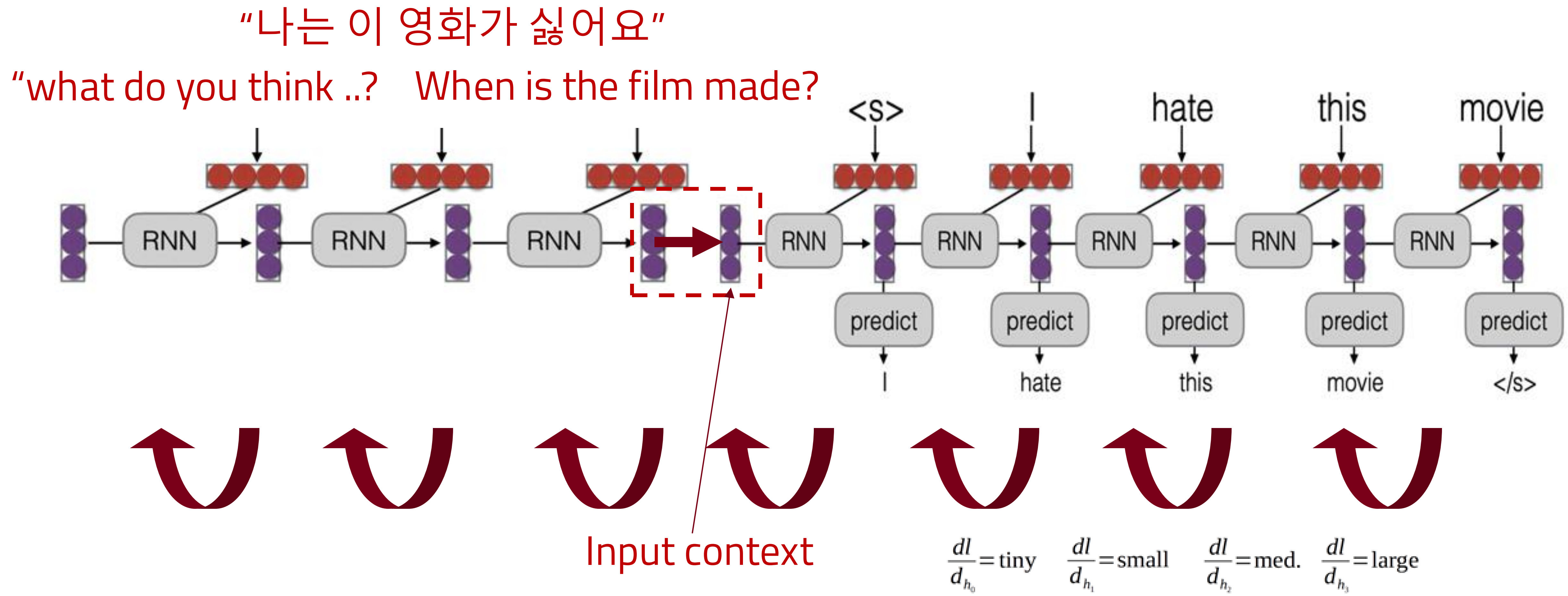


Encoder: encoding input sequence

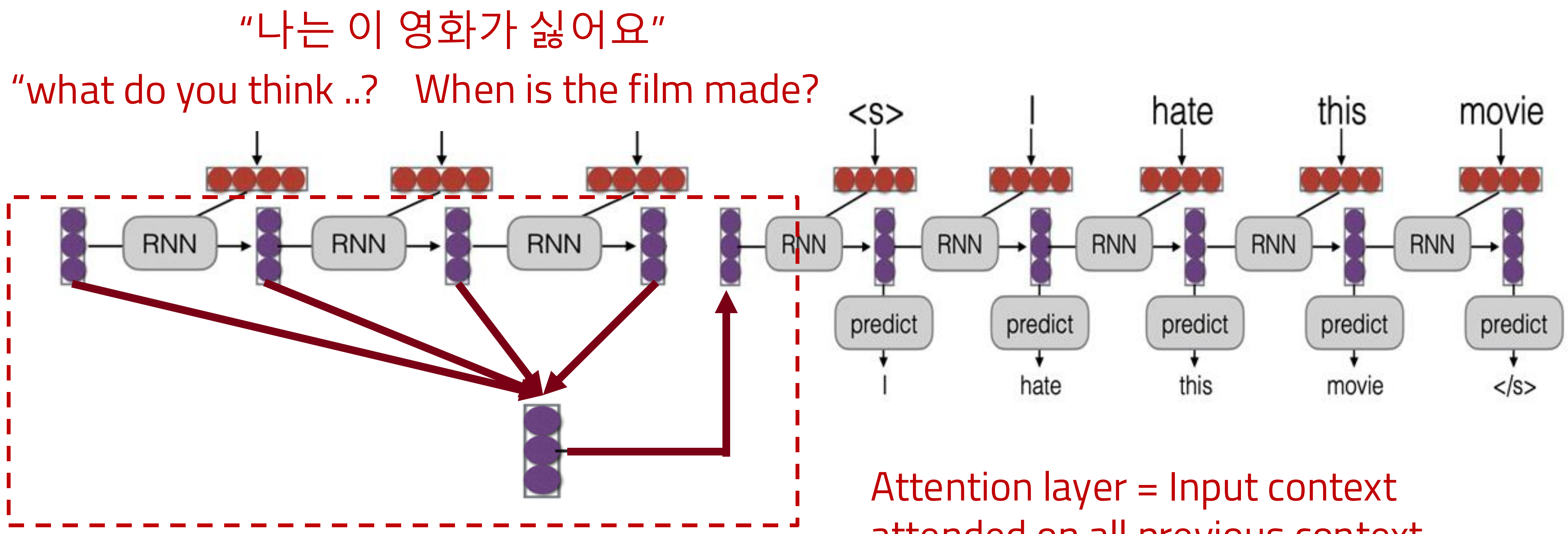
Input context

Decoder: decoding output sequence

Problem: forgetting input context as input gets longer



Solution (teaser): Seq2seq with attention



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

State-of-the-art Language Models

Teaser: Transformer-based LMs

- SOTA LMs: GPT-2, Radford et al. 2018; GPT-3, Brown et al. 2020

Trigram	LSTM
109	58.3

GPT-2	GPT-3
35.8	20.5

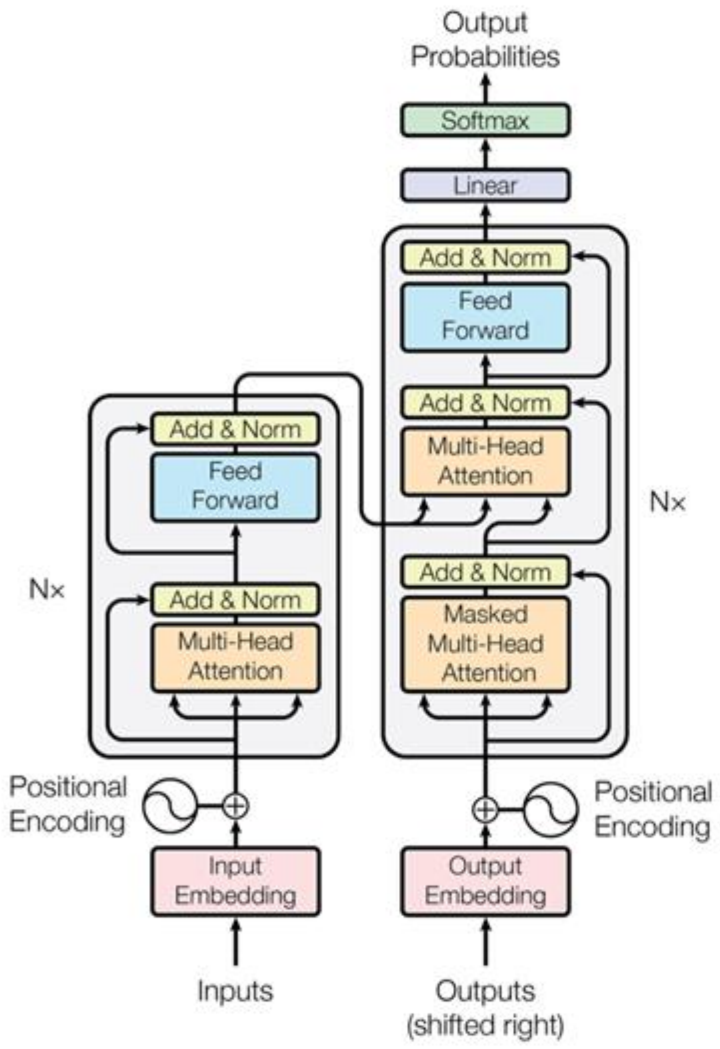
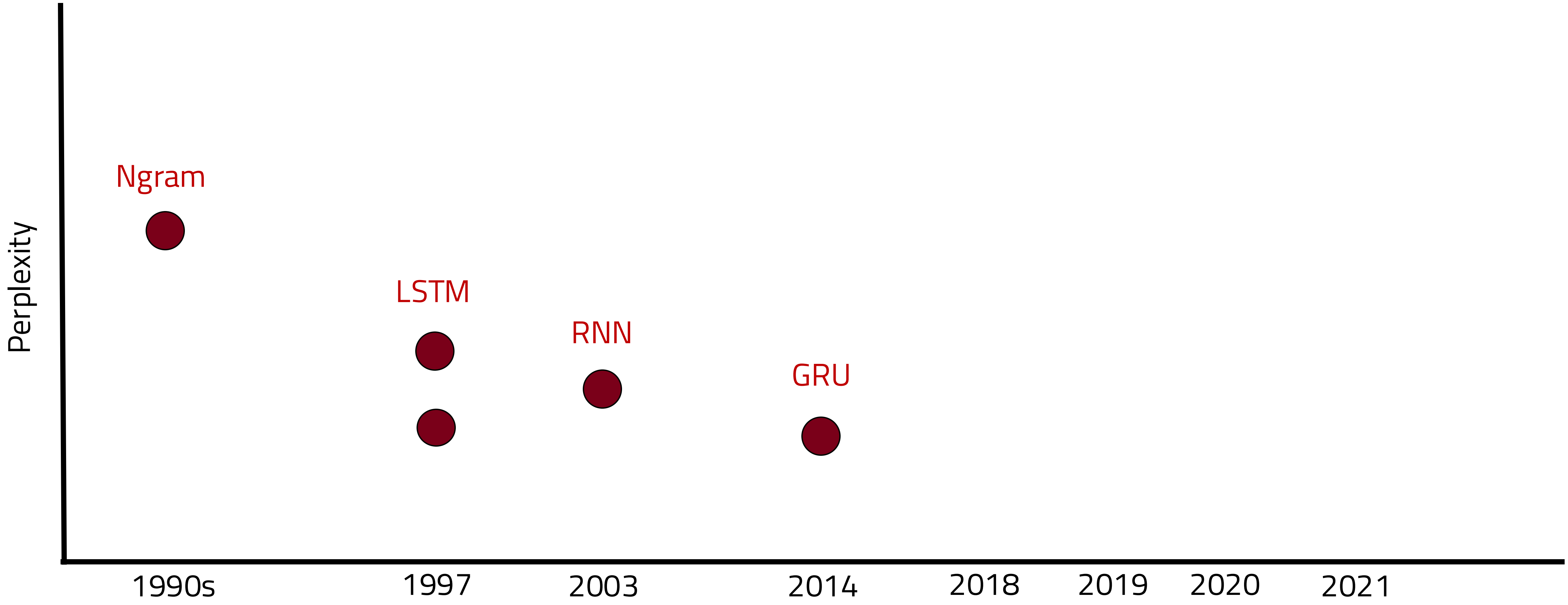
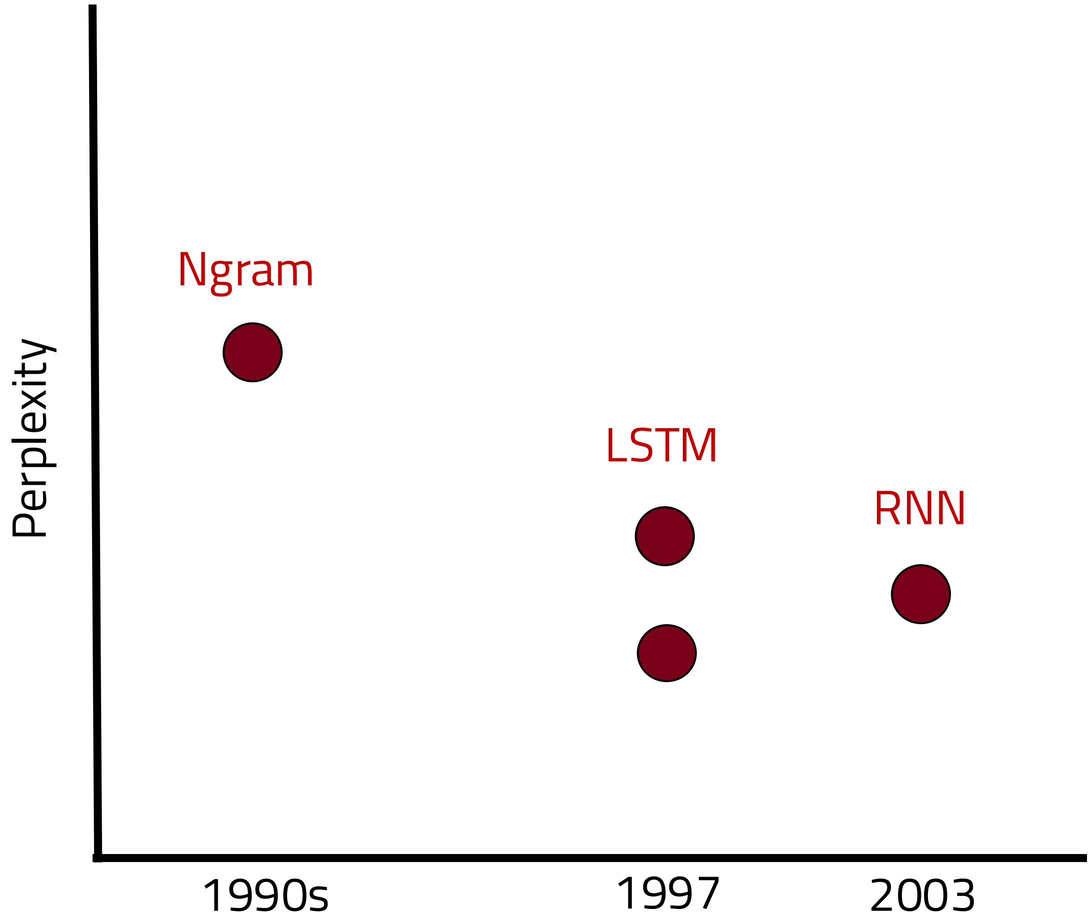


Figure 1: The Transformer - model architecture.



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

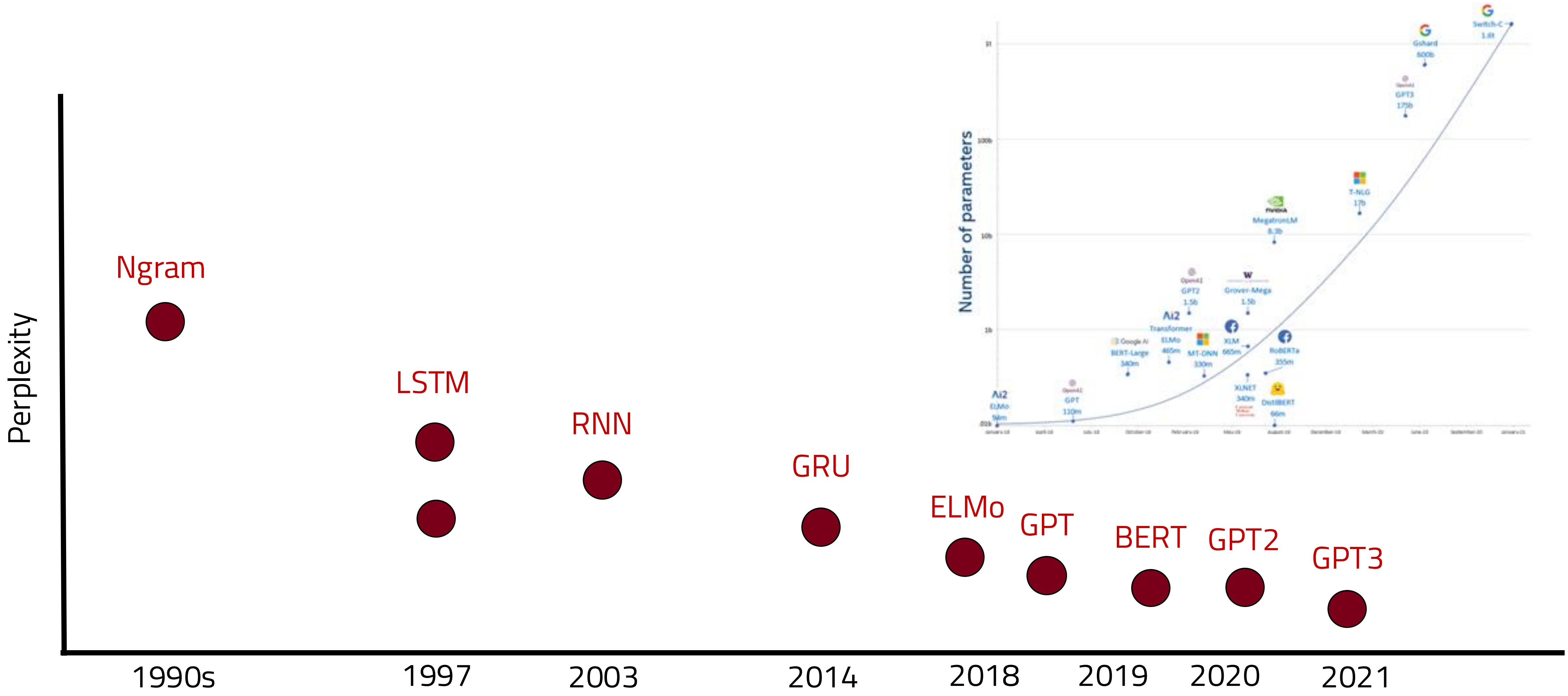


Jürgen Schmidhuber
Pronounce: You_again Shmidhoobuh
Technical Report IDSIA-23-23, IDSIA

AI Blog
Twitter: @SchmidhuberAI
14 December 2023

How 3 Turing Awardees Republished Key Methods and Ideas Whose Creators They Failed to Credit

This write-up is meant to correct an inaccurate history of Artificial Intelligence (AI) propagated by recent uninformed news articles, posts in social media, and a large language model. Most of its statements are taken from a less streamlined report[T22] that has been reviewed on relevant AI mailing lists, profiting from feedback by many experts and well-known AI pioneers. **The piece is aimed at people who are not aware of the numerous AI priority disputes, but are willing to check the facts.**

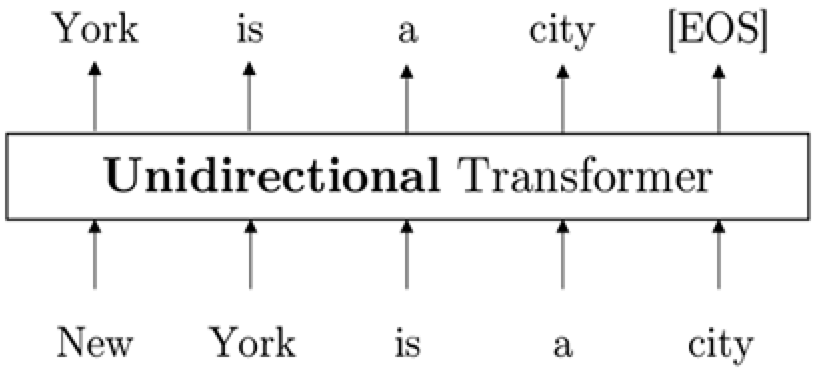


Number of parameters
Switch-C
Gshard
600b
GPT3
175b
T-NLG
17b
MegatronLM
8.3b
GPT2
1.5b
Grover-Mega
1.5b
Ai2
Transformer
ELMo
BERT-Large
340m
MT-DNN
330m
XLM
RoBERTa
355m
XLNET
340m
DistilBERT
66m
Ai2
ELMo
GPT
110m
Perplexity
Ngram
LSTM
RNN
GRU
ELMo
GPT
BERT
GPT2
GPT3
1990s
1997
2003
2014
2018
2019
2020
2021

Teaser: Two Objectives for Language Model Pretraining

GPT GPT2 GPT3

Auto-regressive LM (GPT3)

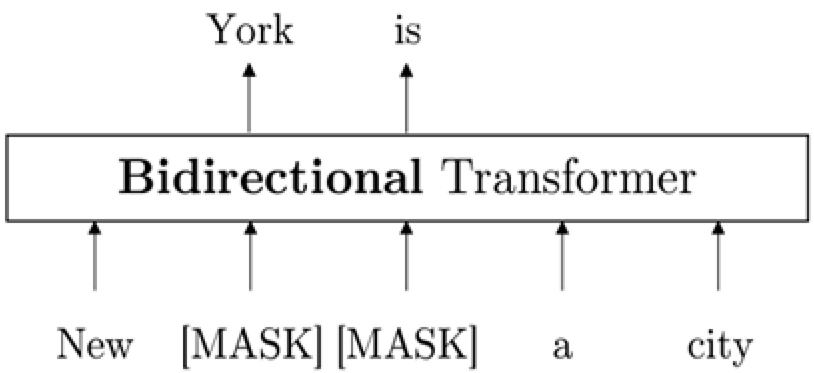


$$\log p(\mathbf{x}) = \sum_{t=1}^{T} \log p(x_t | \mathbf{x}_{<t})$$

Next-token prediction

ELMo BERT

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

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

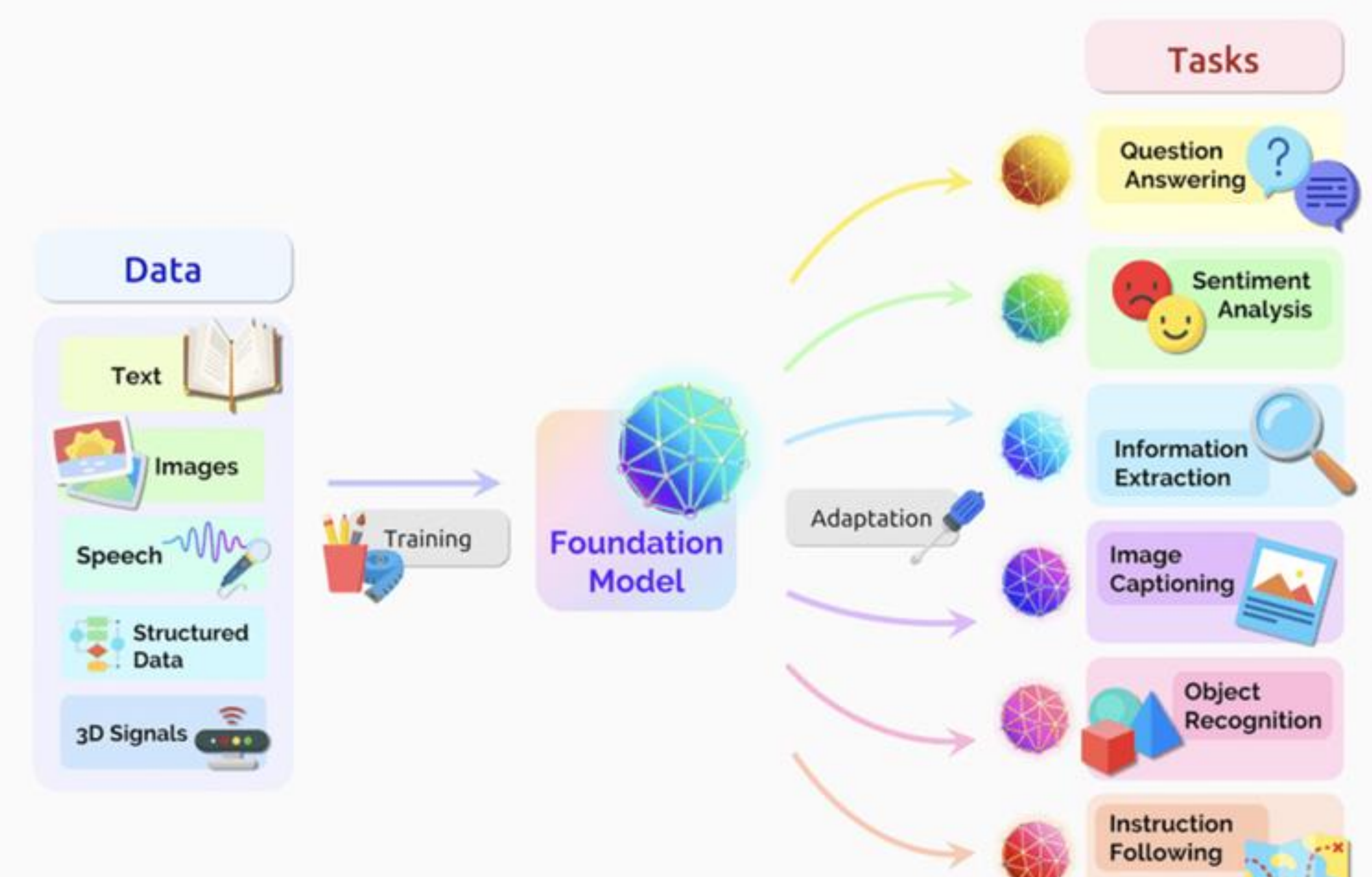
Petroni et al. (2019), "Language Models as Knowledge Bases?" (ACL)

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)

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?"