

CS5740: Natural Language Processing
Spring 2018

Recurrent Neural Networks

Instructor: Yoav Artzi

Overview

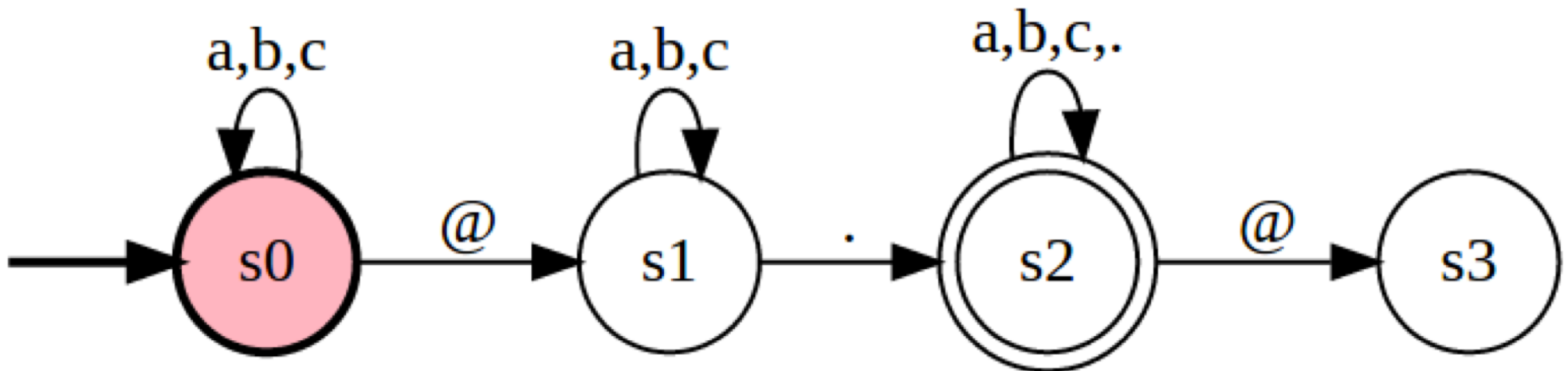
- Finite state models
- Recurrent neural networks (RNNs)
- Training RNNs
- RNN Models
- Long short-term memory (LSTM)
- Attention

Text Classification

- Consider the example:
 - Goal: classify sentiment
 - How can you not see this movie?*
 - You should not see this movie.*
- Model: bag of words
- How well will the classifier work?
 - Similar unigrams and bigrams
- Generally: need to maintain a **state** to capture distant influences

Finite State Machines

- Simple, classical way of representing state
- Current state: saves necessary past information
- Example: email address parsing



Deterministic Finite State Machines

- S – states
- Σ – vocabulary
- $s_0 \in S$ – start state
- $R: S \times \Sigma \rightarrow S$ – transition function
- What does it do?
 - Maps input $w_1, \dots, w_n$ to states $s_1, \dots, s_n$
 - For all $i \in \{1, \dots, n\}$
$$s_i = R(s_{i-1}, w_i)$$
- Can we use it for POS tagging? Language modeling?

Types of State Machines

- Acceptor
 - Compute final state s_n and make a decision based on it: $y = O(s_n)$
- Transducers
 - Apply function $y_i = O(s_i)$ to produce output for each intermediate state
- Encoders
 - Compute final state s_n , and use it in another model

Recurrent Neural Networks

- Motivation:
 - Neural network model, but with state
 - How can we borrow ideas from FSMs?
- RNNs are FSMs ...
 - ... with a twist
 - No longer finite in the same sense

RNN

- $S = \mathbb{R}^{d_{hid}}$ - hidden state space
- $\Sigma = \mathbb{R}^{d_{in}}$ - input state space
- $\mathbf{s}_0 \in S$ - initial state vector
- $R : \mathbb{R}^{d_{in}} \times \mathbb{R}^{d_{hid}} \rightarrow \mathbb{R}^{d_{hid}}$ - transition function
- Simple definition of R :

$$R_{Elman}(\mathbf{s}, \mathbf{x}) = \tanh([\mathbf{x}, \mathbf{s}]\mathbf{W} + \mathbf{b})$$

RNN

- Map from dense sequence to dense representation

- $\mathbf{x}_1, \dots, \mathbf{x}_n \rightarrow \mathbf{s}_1, \dots, \mathbf{s}_n$

- For all $i \in \{1, \dots, n\}$

$$\mathbf{s}_i = R(\mathbf{s}_{i-1}, \mathbf{x})$$

- R is parameterized, and parameters are shared between all steps

- Example:

$$\mathbf{s}_4 = R(\mathbf{s}_3, \mathbf{x}_4) = \dots = R(R(R(R(\mathbf{s}_0, \mathbf{x}_1), \mathbf{x}_2), \mathbf{x}_3), \mathbf{x}_4)$$

RNNs

- Hidden states $\mathbf{s}_i$ can be used in different ways
- Similar to finite state machines
 - Acceptor
 - Transducer
 - Encoder
- Output function maps vectors to symbols:

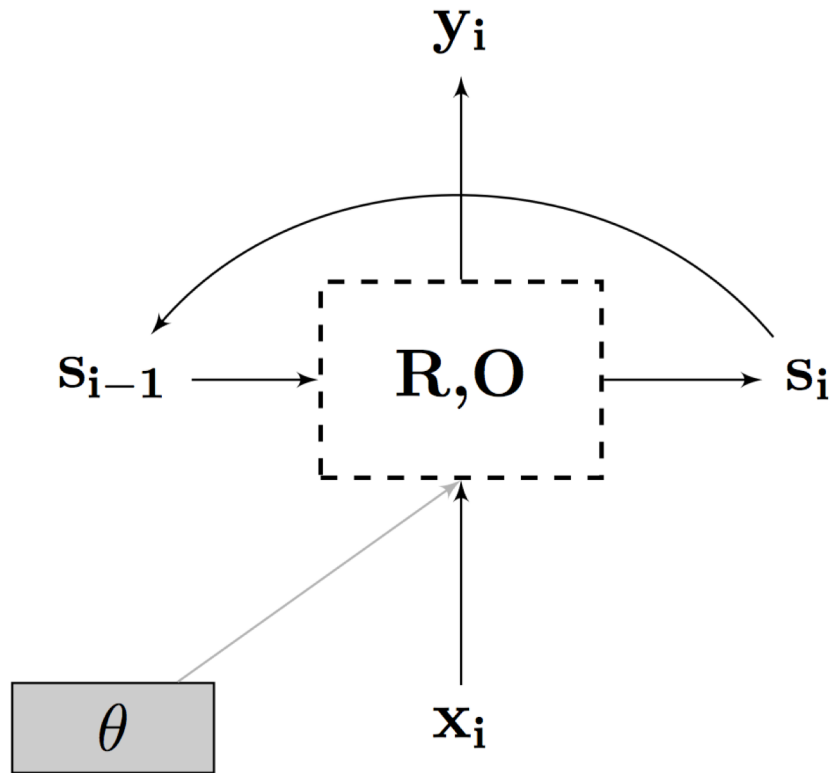
$$O: \mathbb{R}^{d_{hid}} \rightarrow \mathbb{R}^{d_{out}}$$

- For example: single layer + softmax

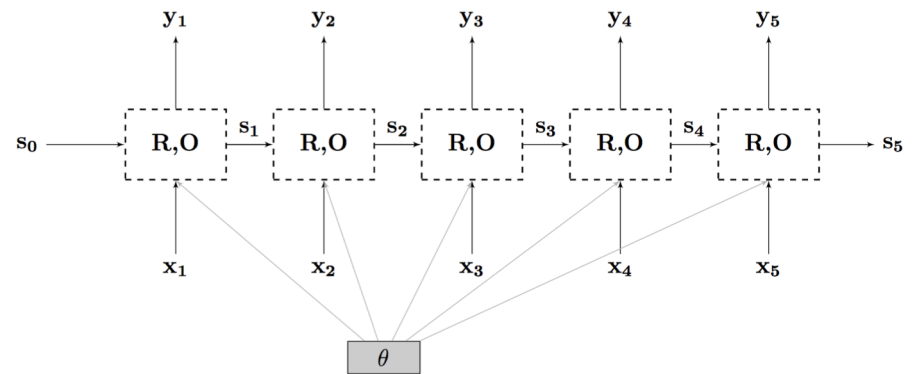
$$O(\mathbf{s}_i) = \text{softmax}(\mathbf{s}_i \mathbf{W} + \mathbf{b})$$

Graphical Representation

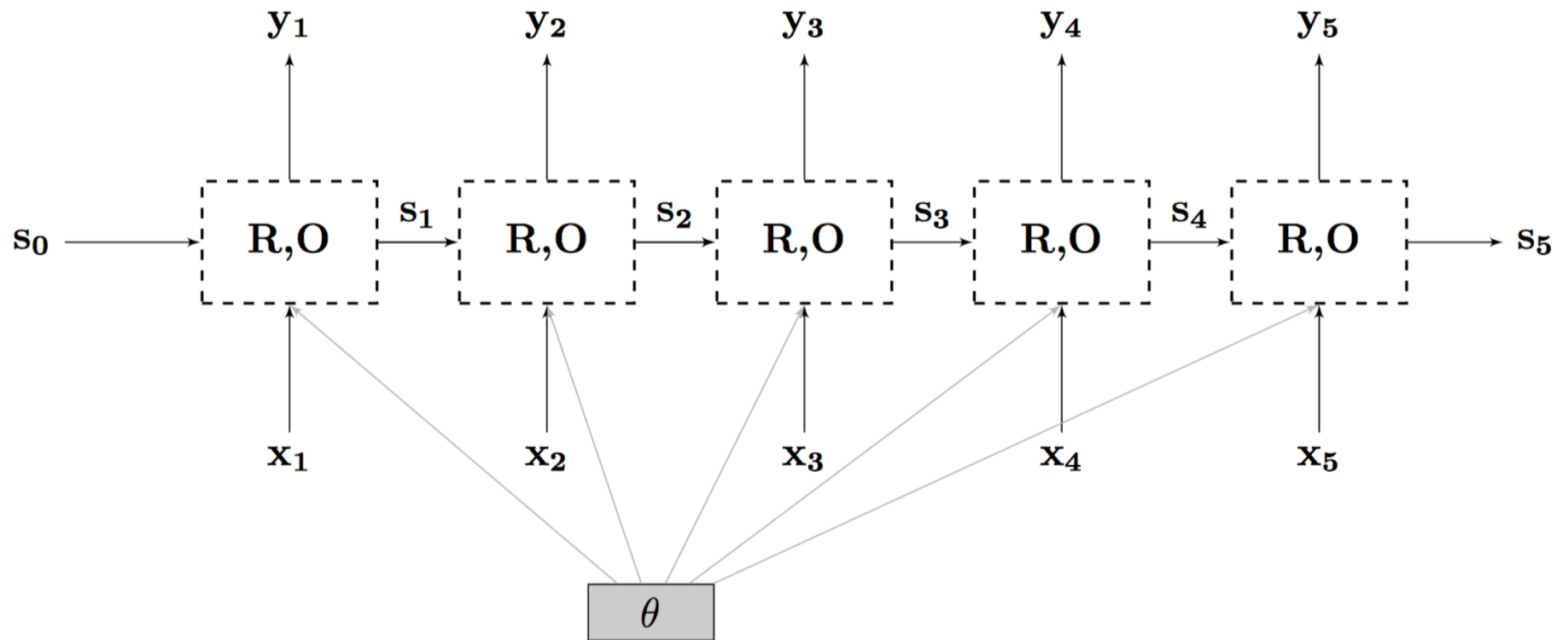
Recursive Representation



Unrolled Representation



Graphical Representation

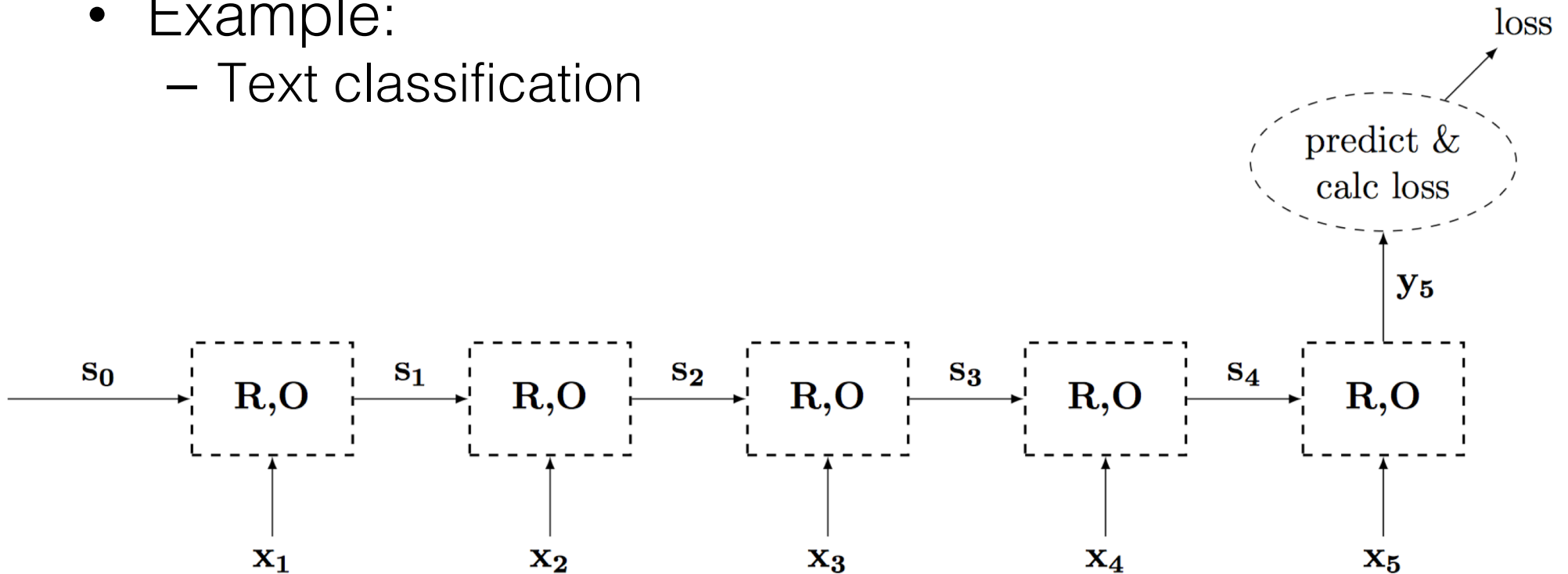


Training

- RNNs are trained with SGD and Backprop
- Define loss over outputs
 - Depends on supervision and task
- Backpropagation through time (BPTT)
 - Use unrolled representation
 - Run forward propagation
 - Run backward propagation
 - Update all weights
- Weights are shared between time steps
 - Sum the contributions of each time step to the gradient
- Inefficient
 - Batch helps, common but tricky to implement with variable-size models

RNN: Acceptor Architecture

- Only care about the output from the last hidden state
- Train: supervised, loss on prediction
- Example:
 - Text classification



Language Modeling

- Input: $X = x_1, \dots, x_n$
- Goal: compute $p(X)$
- Bi-gram decomposition:

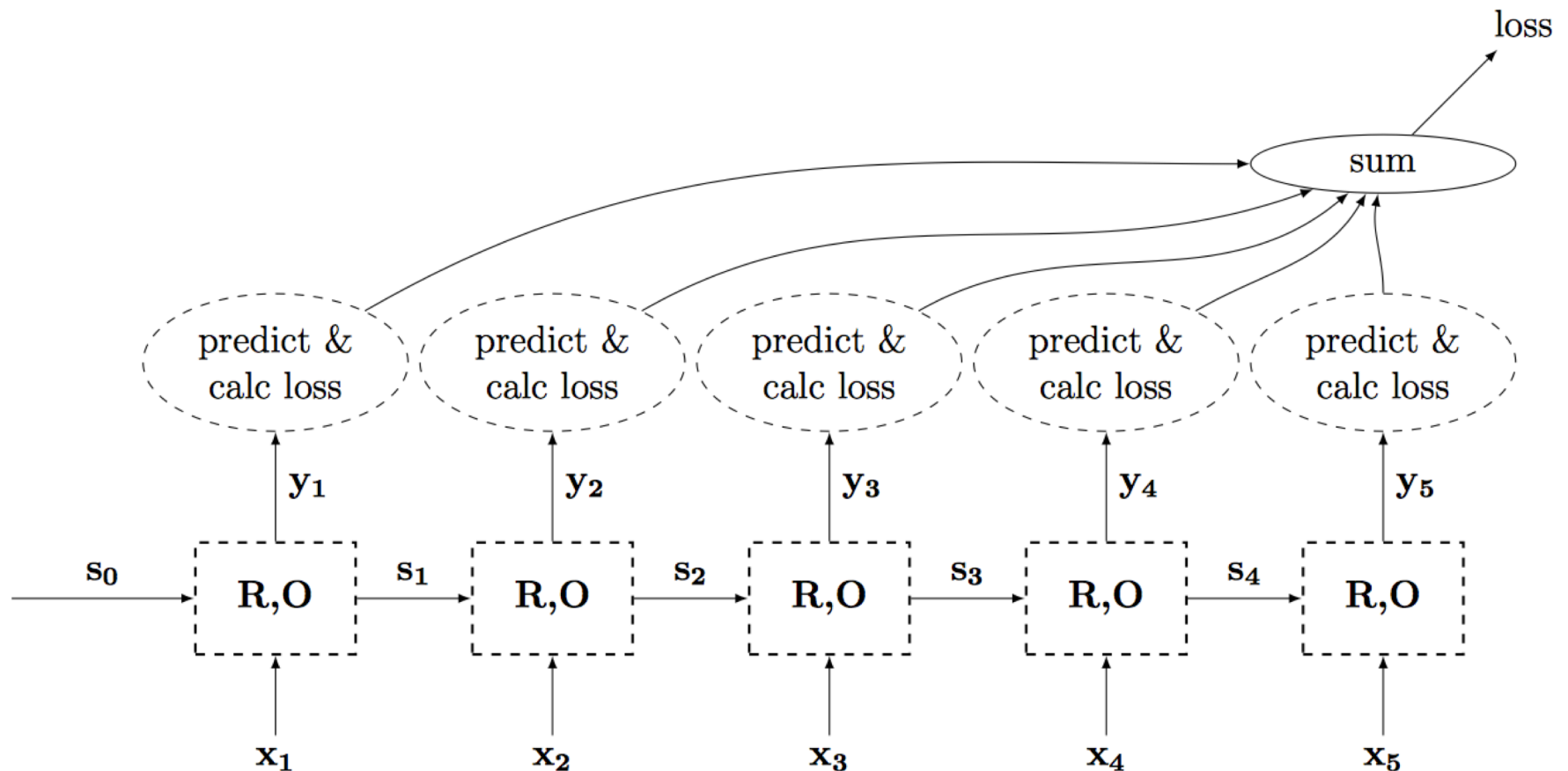
$$p(X) = \prod_{i=1}^n p(x_i \mid x_{i-1})$$

- With RNNs, can do non-Markovian models:

$$p(X) = \prod_{i=1}^n p(x_i \mid x_1, \dots, x_{i-1})$$

RNN: Transducer Architecture

- Predict output for every time step



Language Modeling

- Input: $X = x_1, \dots, x_n$
- Goal: compute $p(X)$
- Model:

$$p(X) = \prod_{i=1}^n p(x_i \mid x_1, \dots, x_{i-1})$$

$$p(x_i \mid x_1, \dots, x_{i-1}) = O(\mathbf{s}_i) = O(R(\mathbf{s}_{i-1}, \mathbf{x}_i))$$

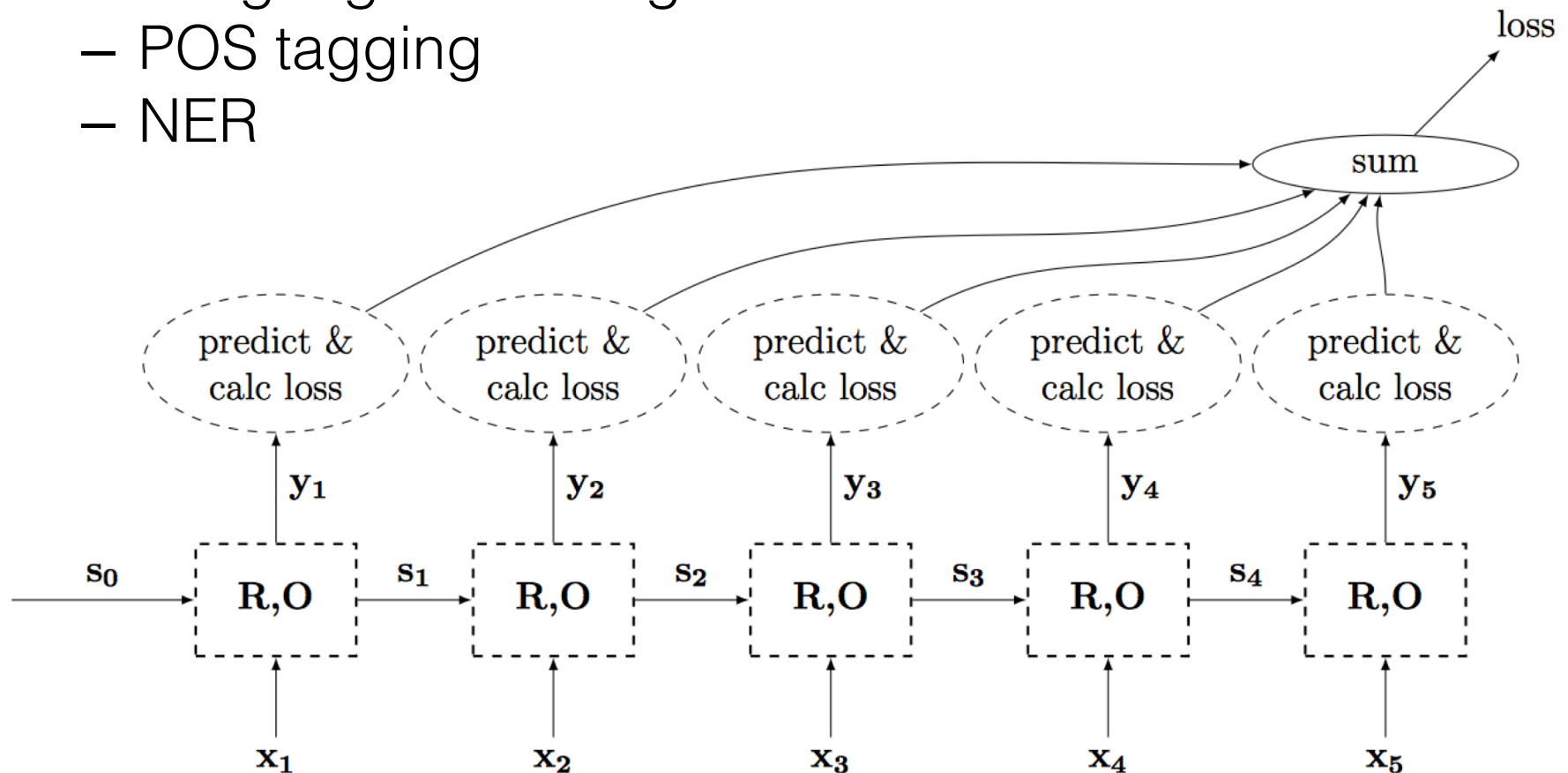
$$O(\mathbf{s}_i) = \text{softmax}(s_i \mathbf{W} + \mathbf{b})$$

- Predict next token $\hat{y}_i$ as we go:

$$\hat{y}_i = \operatorname{argmax} O(\mathbf{s}_i)$$

RNN: Transducer Architecture

- Predict output for every time step
- Examples:
 - Language modeling
 - POS tagging
 - NER



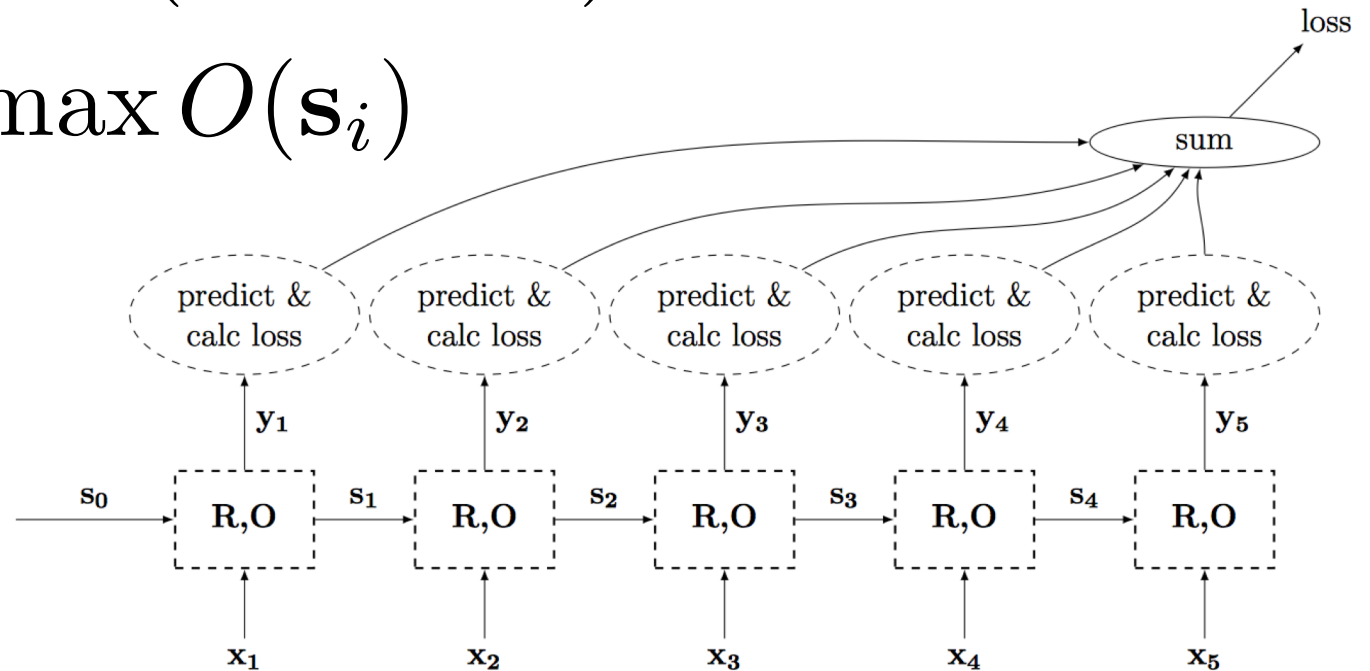
RNN: Transducer Architecture

$$X = \mathbf{x}_1, \dots, \mathbf{x}_n$$

$$\mathbf{s}_i = R(\mathbf{s}_{i-1}, \mathbf{x}_i), i = 1, \dots, n$$

$$O(\mathbf{s}_i) = \text{softmax}(\mathbf{s}_i \mathbf{W} + \mathbf{b})$$

$$\hat{y}_i = \arg \max O(\mathbf{s}_i)$$

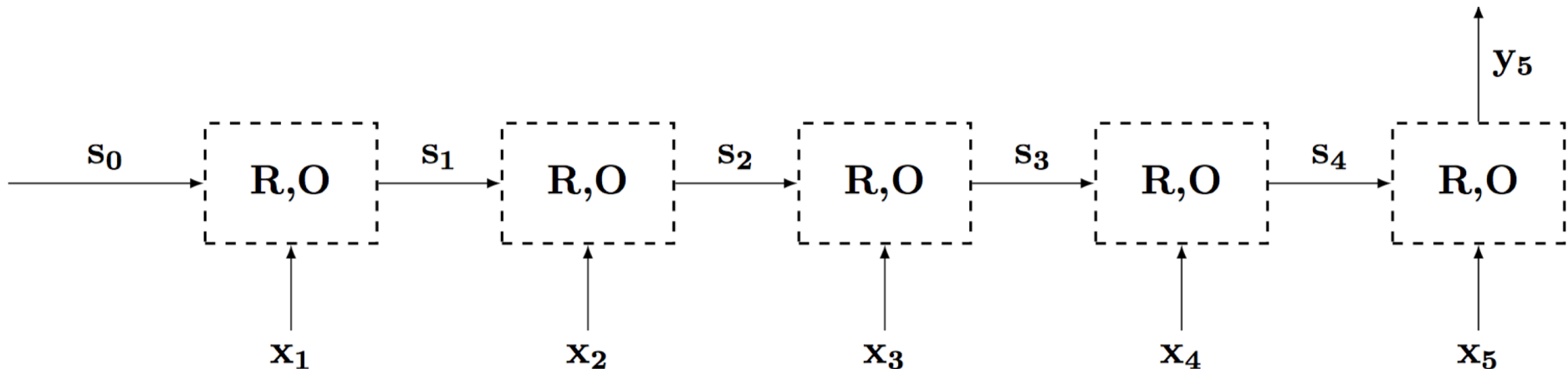


RNN: Encoder Architecture

- Similar to acceptor
- Difference: last state is used as input to another model and not for prediction

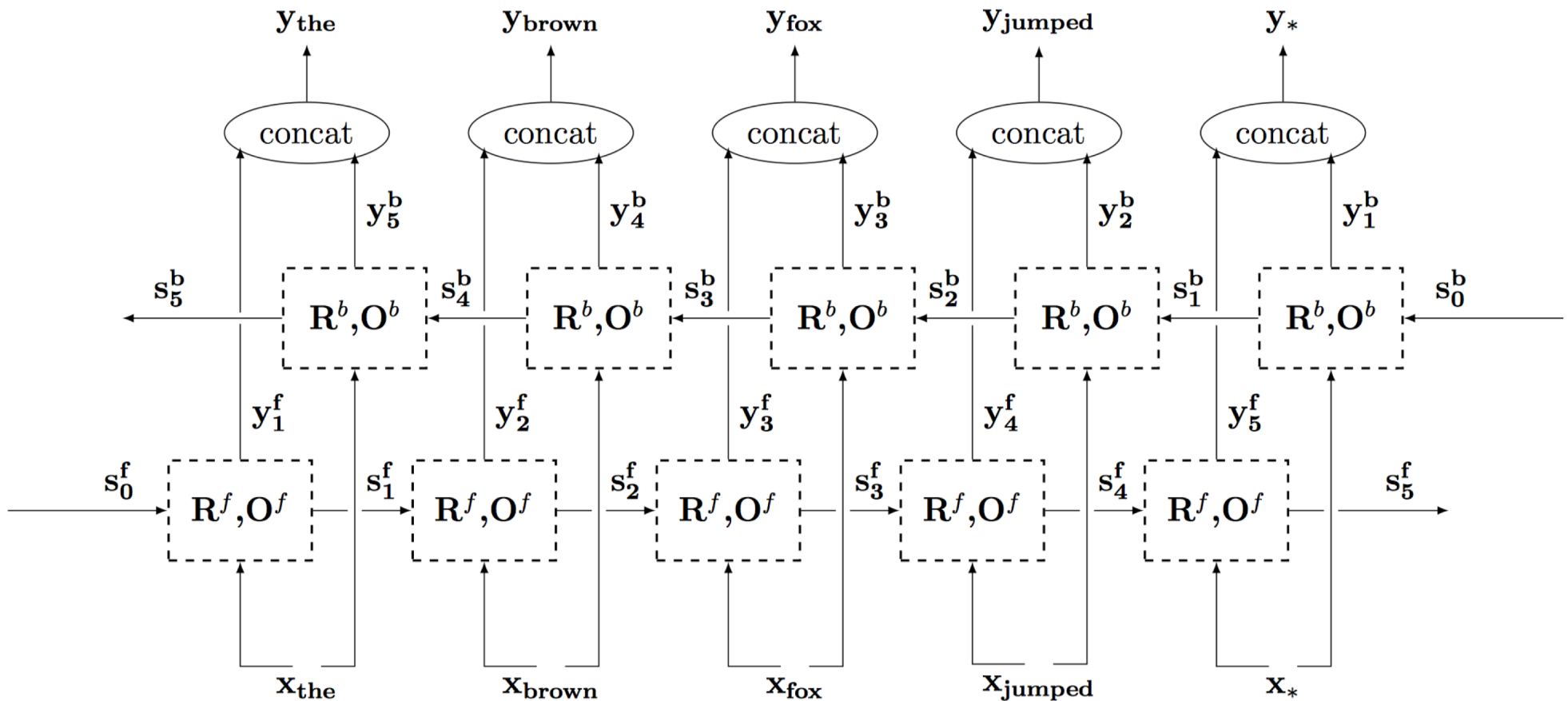
$$O(s_i) = s_i \rightarrow y_n = s_n$$

- Example:
 - Sentence embedding



Bidirectional RNNs

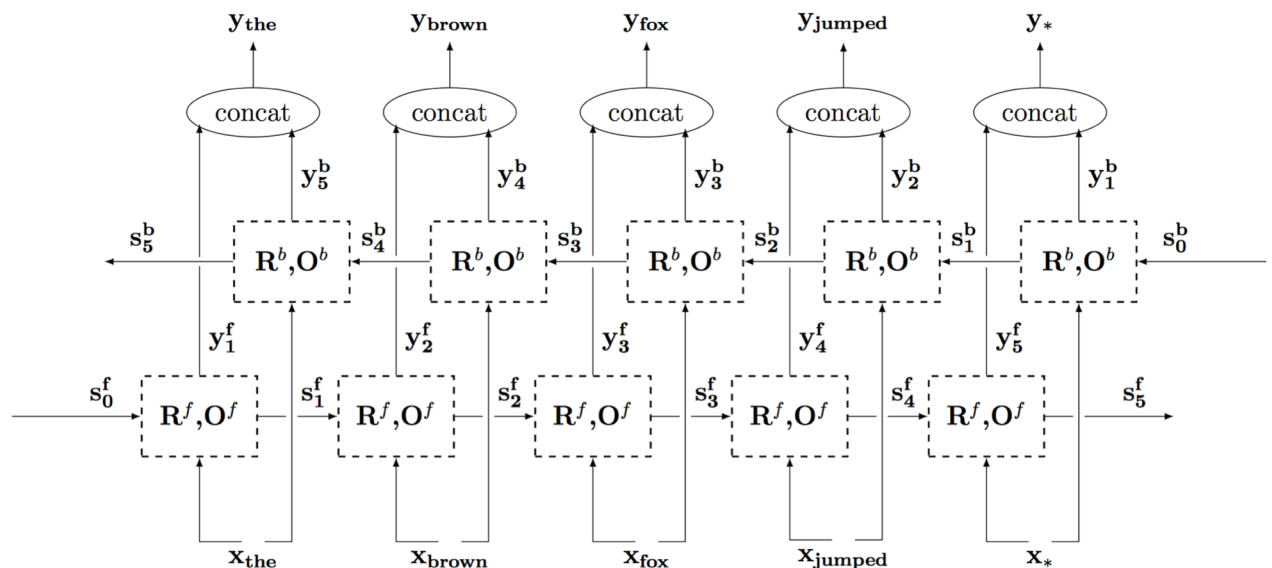
- RNN decisions are based on historical data only
 - How can we account for future input?
- When is it relevant? Feasible?



Bidirectional RNNs

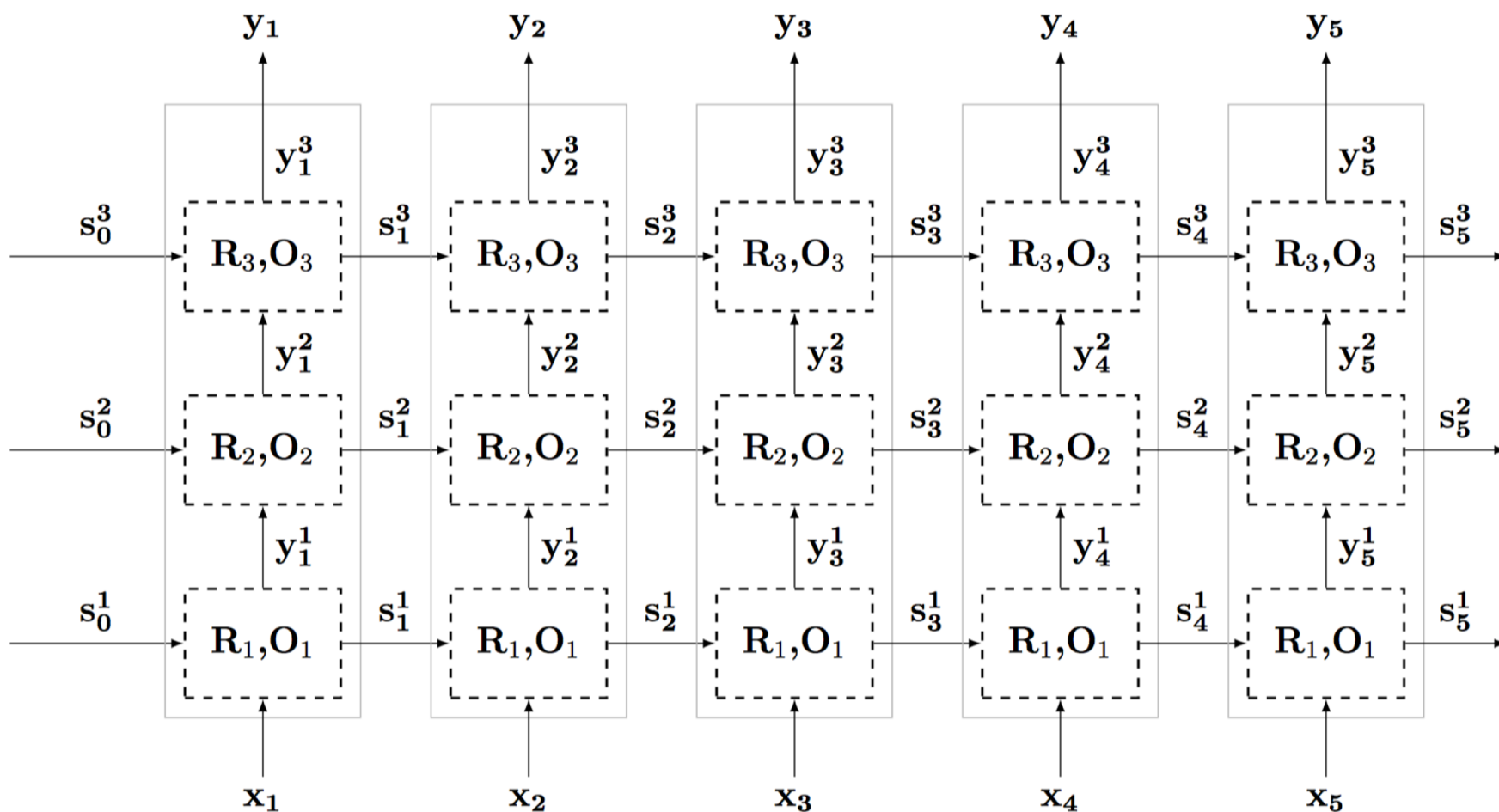
- RNN decisions are based on historical data only
 - How can we account for future input?
- When is it relevant? Feasible?
 - When all the input is available. Not for real-time input.
- Probabilistic model, for example for language modeling:

$$p(X) = \prod_{i=1}^n p(x_i \mid x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n)$$



Deep RNNs

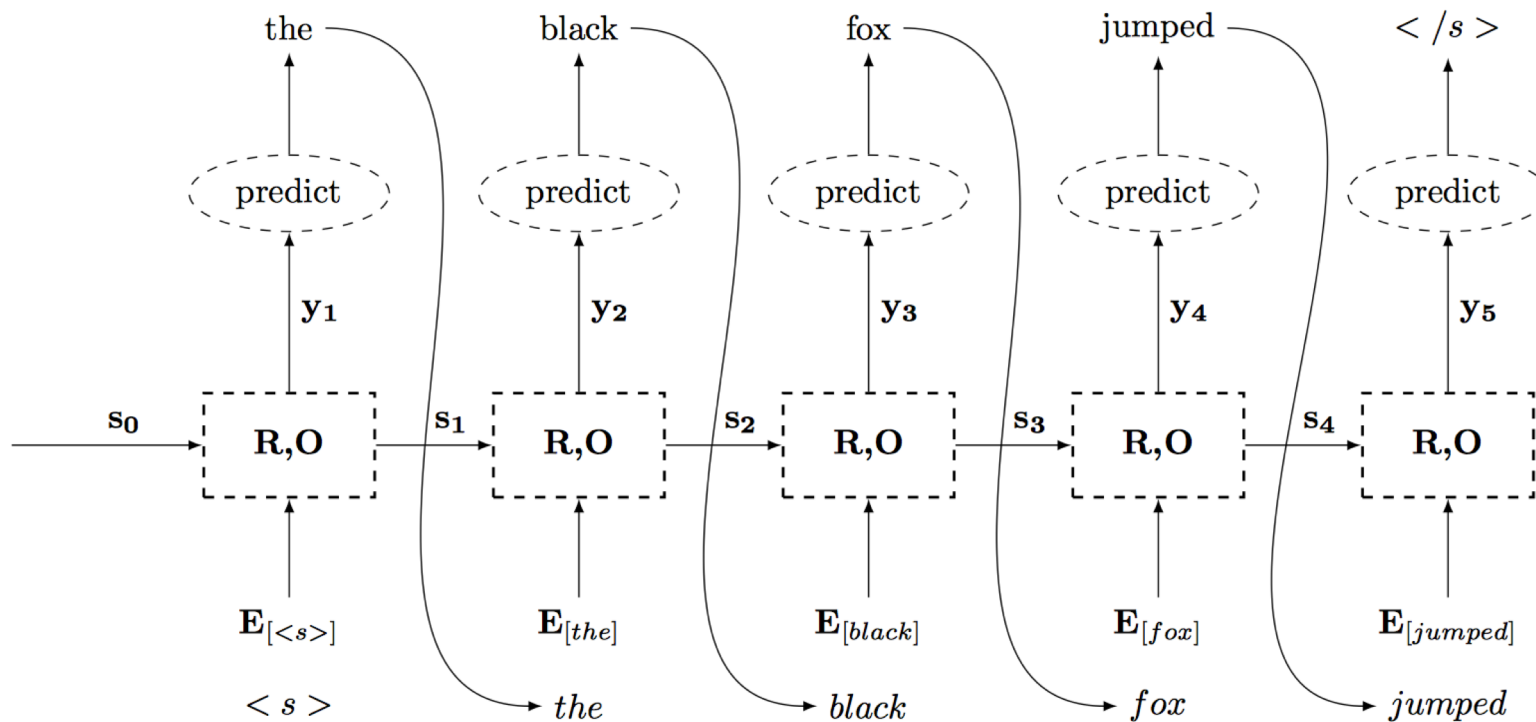
- Can also make RNNs deeper (vertically) to increase the model capacity



RNN: Generator

- Special case of the transducer architecture
- Generation conditioned on s_0
- Probabilistic model:

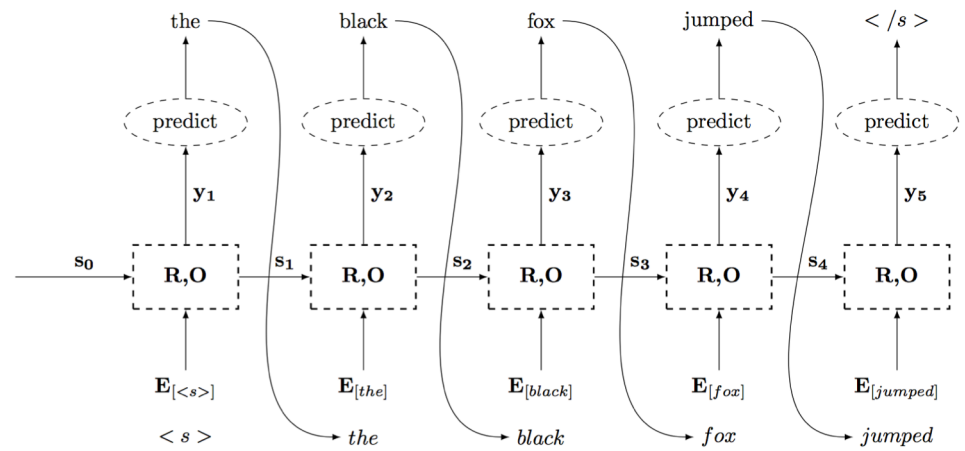
$$p(X | s_0) = \prod_{i=1}^n p(x_i | x_1, \dots, x_{i-1}, s_0)$$



RNN: Generator

- Stop when generating the STOP token
- During learning (usually): force predicting the annotated token and compute loss

$$\mathbf{s}_j = R(\mathbf{s}_{j-1}, \hat{\mathbf{t}}_{j-1})$$
$$O(\mathbf{s}_j) = \text{softmax}(\mathbf{s}_j \mathbf{W} + \mathbf{b})$$
$$\hat{\mathbf{t}}_j = \arg \max O(\mathbf{s}_j)$$



Example: Caption Generation

- Given: image I
- Goal: generate caption
- Set $\mathbf{s}_0 = \text{CNN}(I)$
- Model:

$$p(X | I) = \prod_{i=1}^n p(x_i | x_1, \dots, x_{i-1}, I)$$



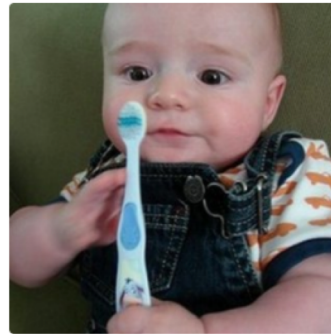
"little girl is eating piece of cake."



"baseball player is throwing ball in game."



"woman is holding bunch of bananas."



"a young boy is holding a baseball bat."



"a cat is sitting on a couch with a remote control."

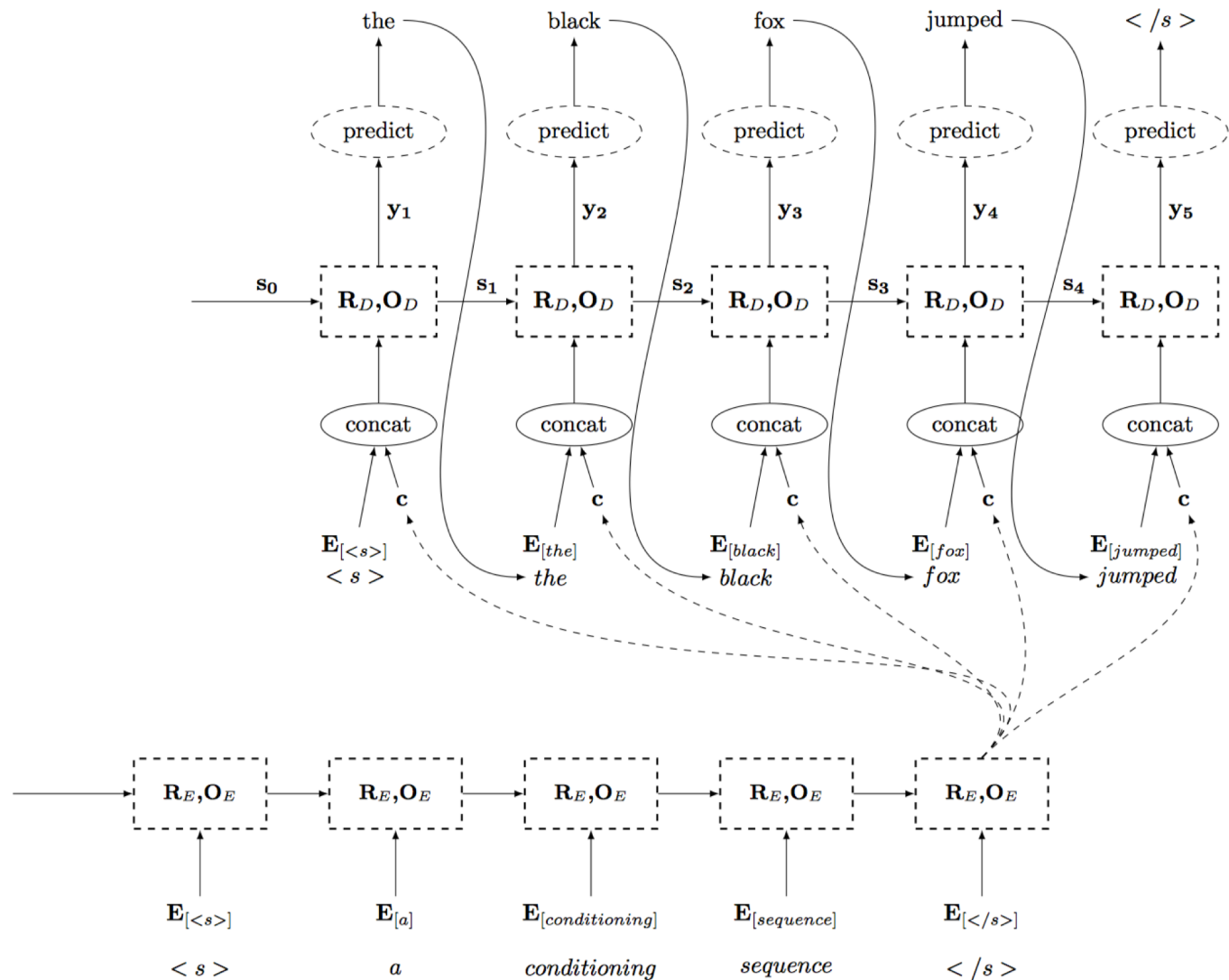


"a woman holding a teddy bear in front of a mirror."

Examples from Karpathy
and Fei-Fei 2015

Sequence-to-Sequence

- Connect encoder and generator
- Many alternatives:
 - Set generator $\mathbf{s}_0^d$ to encoder output $\mathbf{s}_n^e$
 - Concatenate generator $\mathbf{s}_0^d$ with each step input during generation
- Examples:
 - Machine translation
 - Chatbots
 - Dialog systems
- Can also generate other sequences – not only natural language!



Sequence-to-Sequence

$$X = \mathbf{x}_1, \dots, \mathbf{x}_n$$

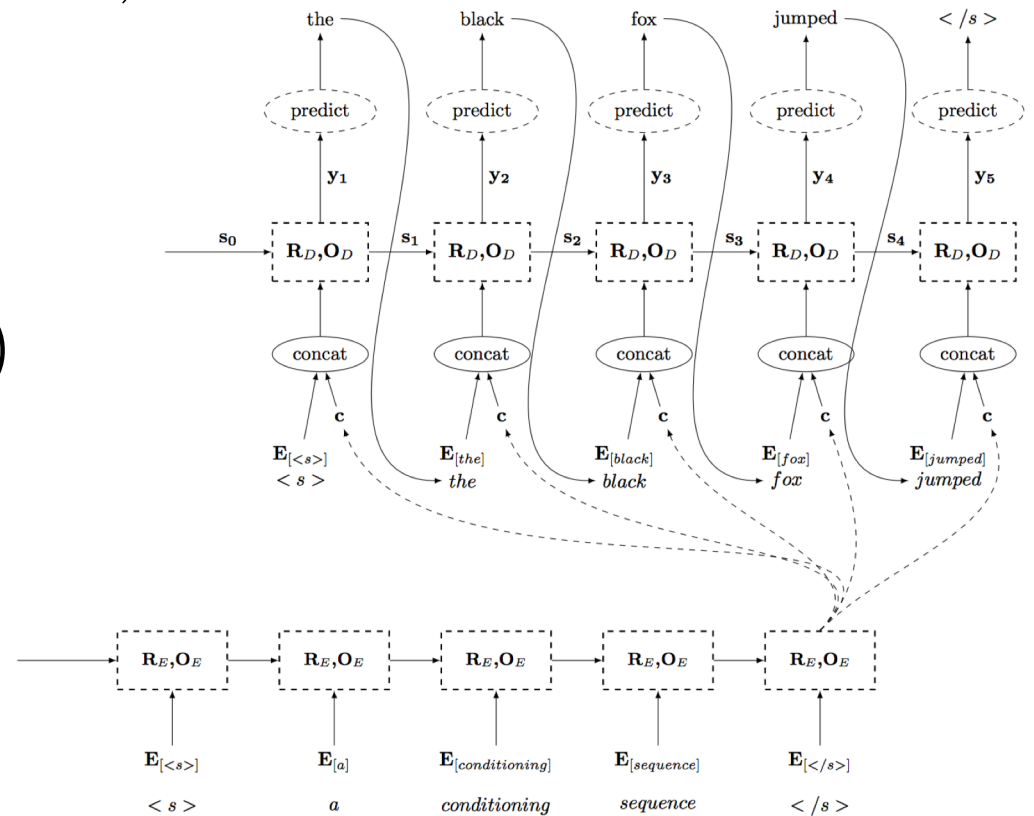
$$\mathbf{s}_i^E = R_E(\mathbf{s}_{i-1}^E, \mathbf{x}_i), i = 1, \dots, n$$

$$\mathbf{c} = O_E(\mathbf{s}_n^E)$$

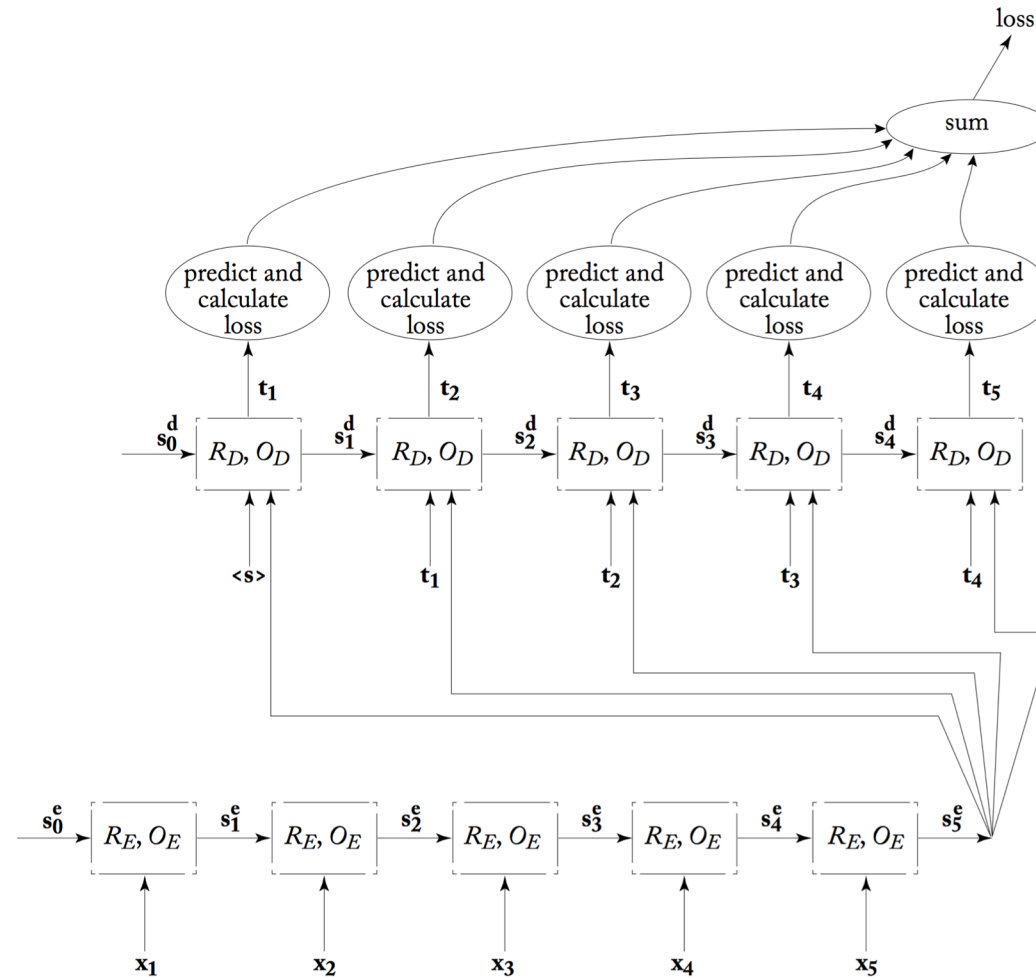
$$\mathbf{s}_j^D = R(\mathbf{s}_{j-1}^D, [\hat{\mathbf{t}}_{j-1}; \mathbf{c}])$$

$$O(\mathbf{s}_j^D) = \text{softmax}(\mathbf{s}_j^D \mathbf{W} + \mathbf{b})$$

$$\hat{\mathbf{t}}_j = \arg \max O(\mathbf{s}_j^D)$$



Sequence-to-Sequence Training Graph



Long-range Interactions

- Promise: Learn long-range interactions of language from data
- Example:

How can you not see this movie?

You should not see this movie.
- Sometimes: requires "remembering" early state
 - Key signal here is at s_1 , but gradient is at s_n

Long-term Gradients

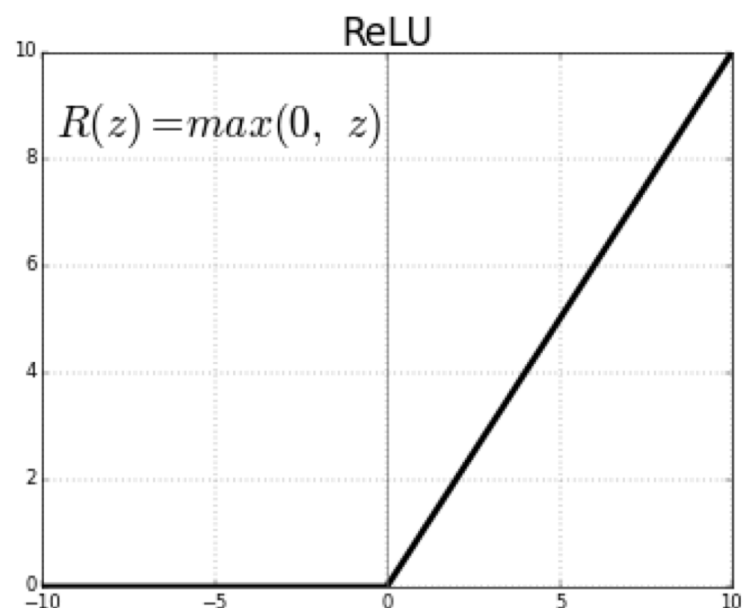
- Gradient go through (many) multiplications
- OK at end layers → close to the loss
- But: issue with early layers
- For example, derivative of tanh

$$\frac{d}{dx} \tanh x = 1 - \tanh^2 x$$

- Large activation → gradient disappears
- In other activation functions, values can become larger and larger

Exploding Gradients

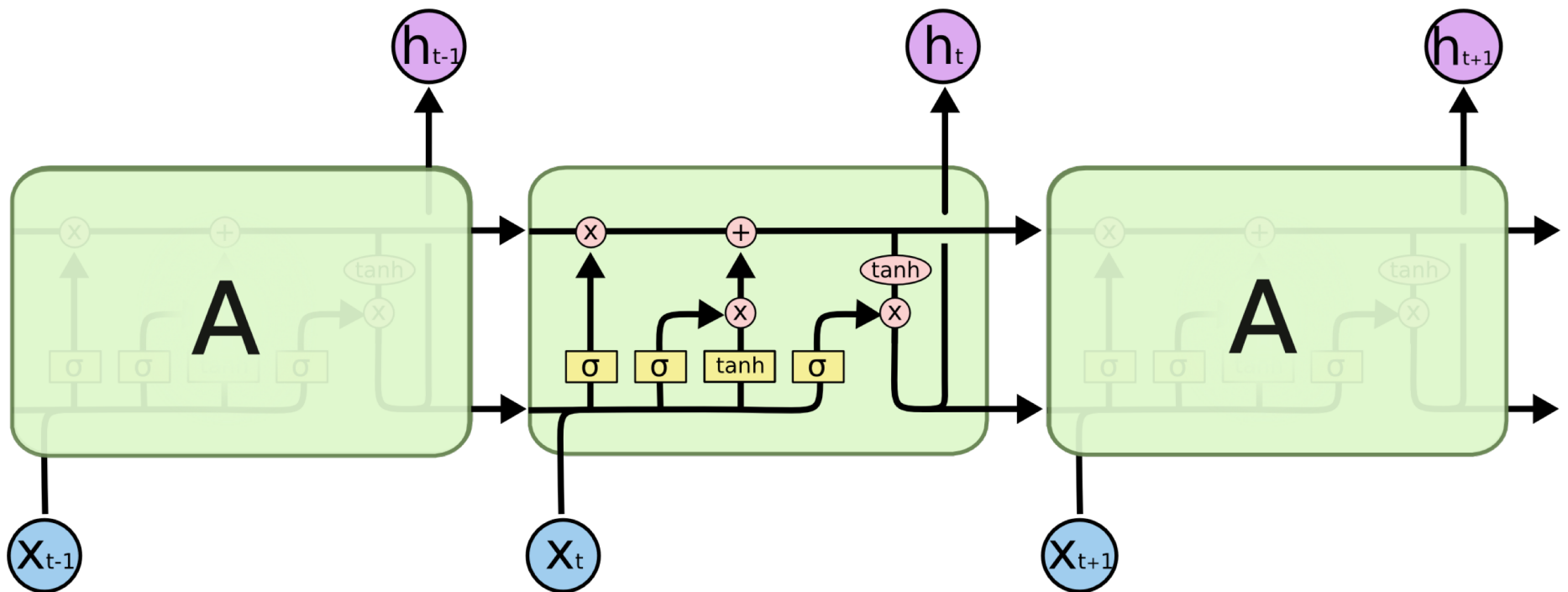
- Common when there is not saturation in activation (e.g., ReLU) and we get exponential blowup
- Result: reasonable short-term gradient, but bad long-term ones
- Common heuristic:
 - Gradient clipping: bounding all gradients by maximum value



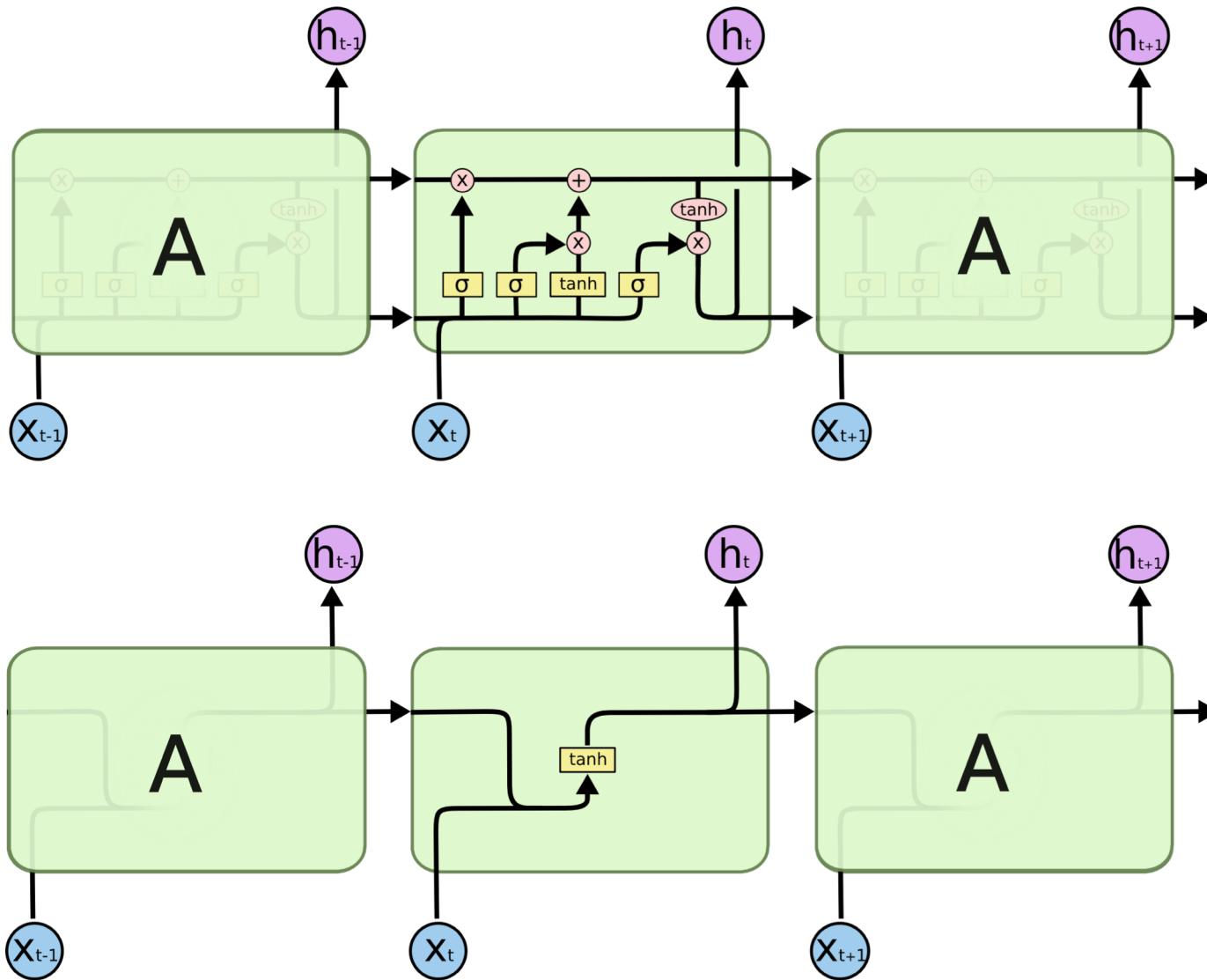
Vanishing Gradients

- Occurs when multiplying small values
 - For example: when $\tanh$ saturates
- Mainly affects long-term gradients
- Solving this is more complex

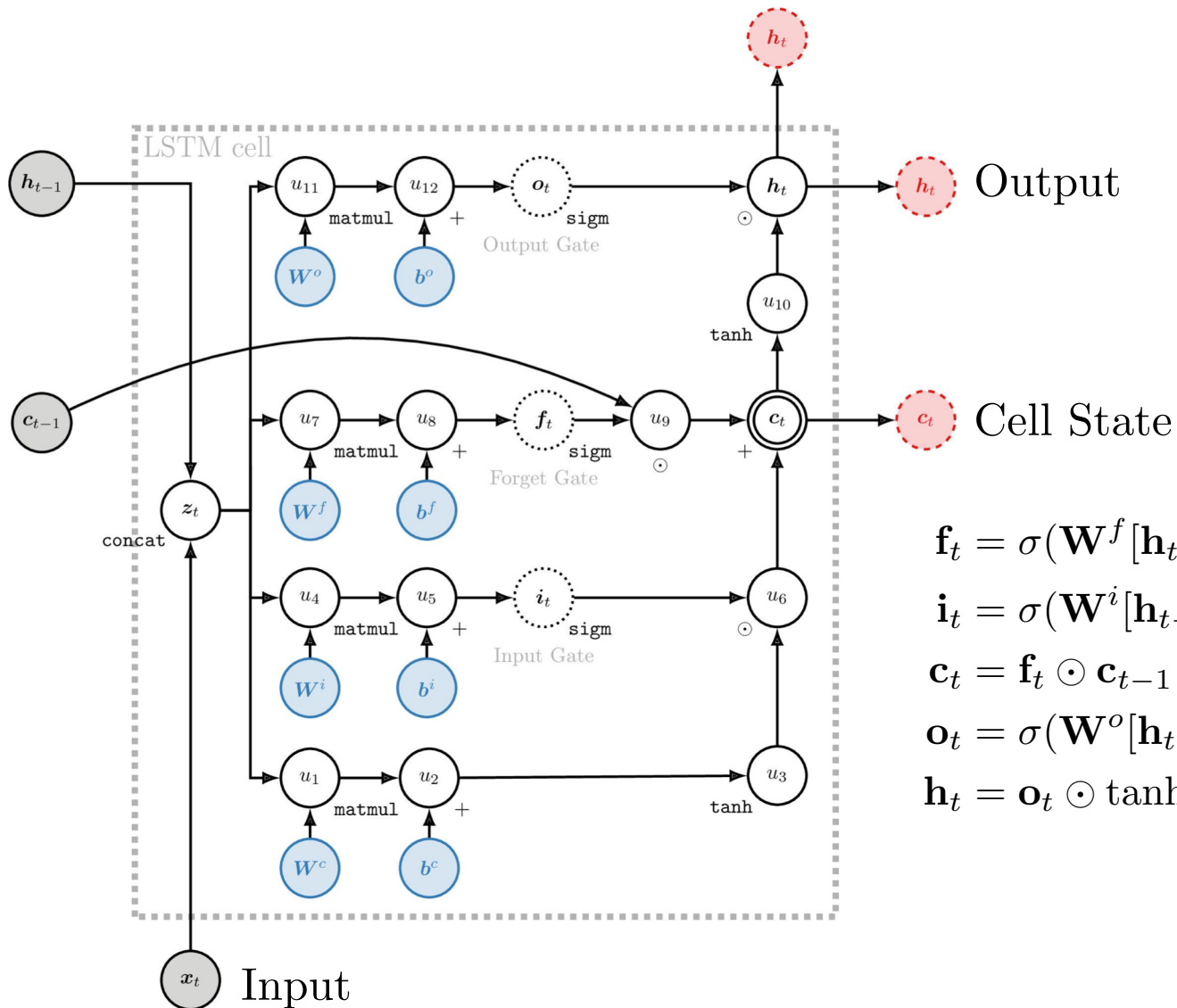
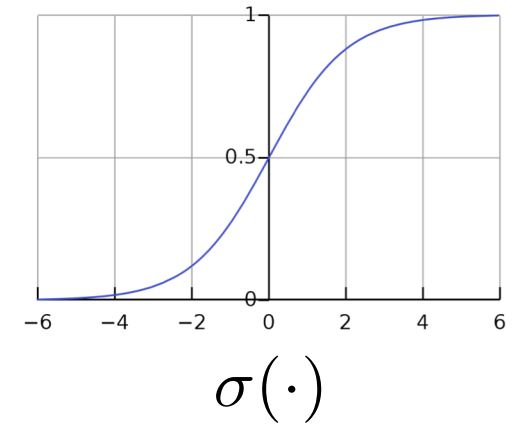
Long Short-term Memory (LSTM)



LSTM vs. Elman RNN



LSTM



$$\begin{aligned} f_t &= \sigma(\mathbf{W}^f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}^f) \\ i_t &= \sigma(\mathbf{W}^i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}^i) \\ c_t &= f_t \odot c_{t-1} + i_t \odot \tanh(\mathbf{W}^c[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}^c) \\ o_t &= \sigma(\mathbf{W}^o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}^o) \\ h_t &= o_t \odot \tanh(c_t) \end{aligned}$$

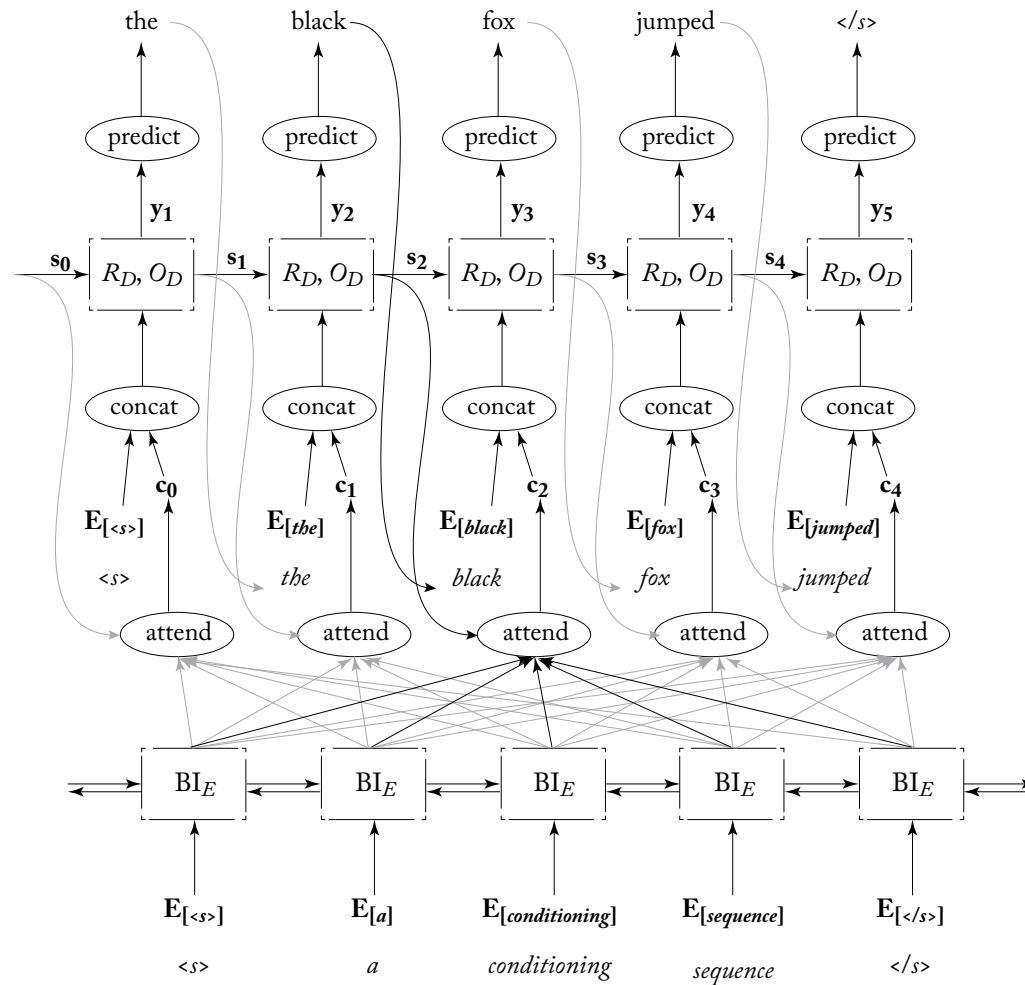
Attention

- In seq-to-seq models, a single vector is connects encoding and decoding
 - Worrying?
 - All the input string information must encoded into a fixed-length vector
 - The decoder must recover all this information from a fixed-length vector
- Attention relaxes the assumption that single vector must be used to encode the input sentence regardless of length

Attention

- Encode input sentence as a sequence of vectors
- At each step: pick what vector to use
- But: discrete choice is not differentiable
 - Make the choice soft

Attention



Attention

$$X = \mathbf{x}_1, \dots, \mathbf{x}_n$$

$$\mathbf{s}_i^E = R_E(\mathbf{s}_{i-1}^E, \mathbf{x}_i), i = 1, \dots, n$$

$$\mathbf{c}_i = O_E(\mathbf{s}_i^E)$$

$$\bar{\alpha}_i^j = \mathbf{s}_{j-1}^D \cdot \mathbf{c}_i$$

$$\alpha^j = \text{softmax}(\bar{\alpha}_1^j, \dots, \bar{\alpha}_n^j)$$

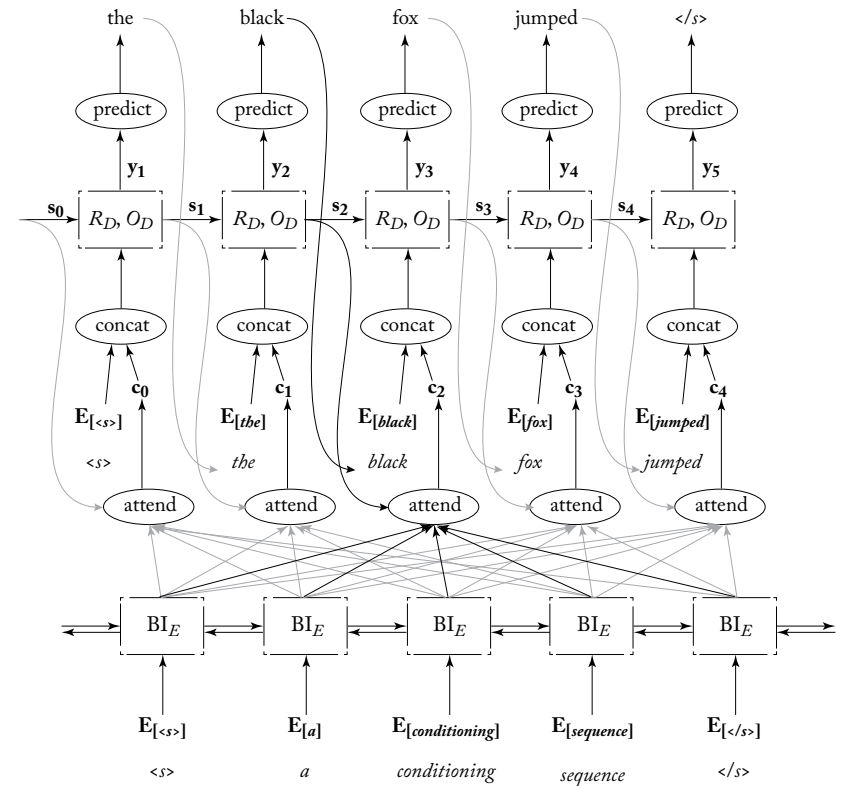
$$\mathbf{c}^j = \sum_{i=1}^n \alpha_i^j \mathbf{c}_i$$

$$\mathbf{s}_j^D = R(\mathbf{s}_{j-1}^D, [\hat{\mathbf{t}}_{j-1}; \mathbf{c}^j])$$

$$O(\mathbf{s}_j^D) = \text{softmax}(\mathbf{s}_j^D \mathbf{W} + \mathbf{b})$$

$$\hat{\mathbf{t}}_j = \arg \max O(\mathbf{s}_j^D)$$

Attend



Attention

- Many variants of attention function
 - Dot product (previous slide)
 - MLP
 - Bi-linear transformation
- Various ways to combine context vector into decoder computation
- See Luong et al. 2015