

Named Entity Recognition and Sequence Tagging: MEMM, CRF and RNN

Bonan Min

bonanmin@gmail.com

Some slides are based on class materials from Thien Huu Nguyen, Ralph Grishman, Yoav Goldberg

Named Entity Recognition (NER)

Identify names of entities (i.e., persons, organizations, locations, etc.) in text

Can be casted as a sequence labeling problem via the BIO (beginning-inside-other) tagging schema, thus can be solved by HMM

Alternative tagging scheme: BIOES (E=end; S=start of a single-token chunk)

Person				Organization	
Fred	Smith	works	for	Time	inc.
B_PER	I_PER	O	O	B_ORG	I_ORG

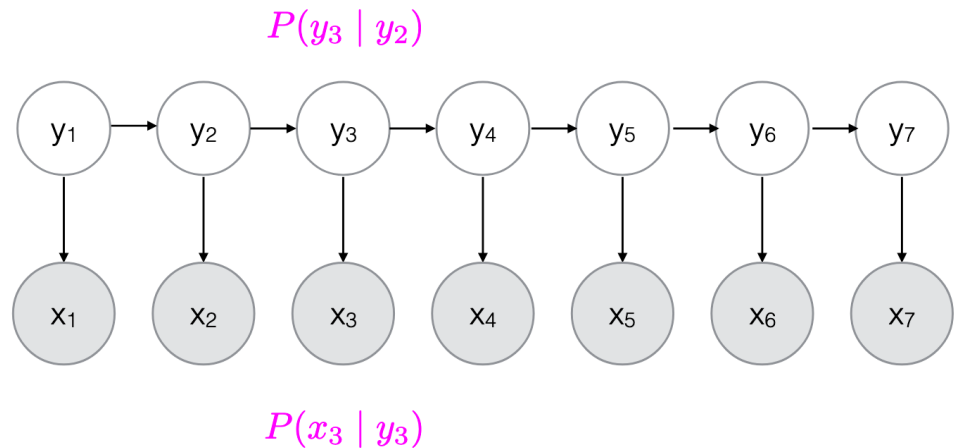
HMM for Sequence Labeling

Simple and fast to train and to use

Effective for POS tagging (one POS, one state)

can be made effective for name tagging (can capture context) by splitting states

but further splitting could lead to sparse data problems



We Want ...

We want to have a more flexible means of capturing our linguistic intuition that certain conditions lead to the increased likelihood of certain outcomes (i.e., feature engineering)

- that a name on a 'common first name' list increases the chance that this is the beginning of a person name
- that being in a sports story increases the chance of team (organization) names

Maximum entropy modeling (logistic regression) provides one mathematically well-founded method for combining such features in a probabilistic model

- But it's not for sequence tagging

Maximum Entropy Markov Model (MEMM)


Starting with the conditional probability distribution

$$P(y|x) = p(y_1, y_2, \dots, y_n|x) = \prod_{t=1}^n P(y_t|y_{<t}, \chi)$$

Using the first-order Markov assumption (the probability of the current state only depends on the previous state):

$$P(y_t|y_{<t}, \chi) \approx P(y_t|y_{t-1}, \chi)$$
$$P(y|x; \theta) = \prod_{t=1}^n P(y_t|y_{t-1}, \chi; \theta)$$

The probability for one step depends on the entire input sentence χ



Using logistic regression to model the probabilities $P(y_t|y_{t-1}, \chi; \theta)$, allowing flexible feature engineering

Maximum Entropy Markov Model (MEMM)

Using logistic regression to model the probabilities $P(y_t|y_{t-1}, x; \theta)$

Defining K binary features $f_i(y_{t-1}, x)$ over the the prior label y_{t-1} and the entire input sentence x . For examples:

$$f_i(y_{t-1}, x) = \begin{cases} 1 & \text{if } x_i = \textit{Smith} \text{ and } y_{t-1} = \textit{B_PER} \\ 0 & \text{otherwise} \end{cases}$$

$$f_i(y_{t-1}, x) = \begin{cases} 1 & \text{if } x_i \text{ is capitalized} \\ 0 & \text{otherwise} \end{cases}$$

$$f_i(y_{t-1}, x) = \begin{cases} 1 & \text{if } x_i \text{ is in the list of common names and } y_{t-1} = 0 \\ 0 & \text{otherwise} \end{cases}$$

Then:

$$P(y_t|y_{t-1}, x; \theta) = \frac{\exp\left(\sum_{i=1}^K w_i^{y_t} f_i(y_{t-1}, x)\right)}{\mathbf{z}(y_{t-1}, x)}$$

Where \mathbf{z} is the normalizing factor and $w^{y_t} = [w_1^{y_t}, w_2^{y_t}, \dots, w_K^{y_t}]$ is the model parameter specific to y_t

Maximum Entropy Markov Model (MEMM)

In order to train the MEMM model (i.e., finding the model parameters), we can also optimize the likelihood function over the training dataset:

$$L(\theta) = - \sum_{(x,y) \in D} \log P(y|x, \theta)$$

There is no closed-form solution for this optimization problem (as HMM); an iterative solver is required

The good thing is the function is convex, so easier to solve those with gradient descent

Feature Engineering

The main task when using a MaxEnt classifier (e.g., MEMM) is to select an appropriate set of features

- words in the immediate neighborhood are typical basic features: w_{i-1} , w_i , w_{i+1}
- patterns constructed for rule-based taggers are likely candidates: w_{i+1} is an initial
- membership on word lists: w_i is a common first name (from Census)

Greedy Decoding for MEMM

At $i = 0$, select:

$$y_1^* = \operatorname{argmax}_s P(y_1 = s | y_0 = \text{start}, x) = \operatorname{argmax}_s P(y_1 = s | x)$$

At $i > 0$, select:

$$y_i^* = \operatorname{argmax}_s P(y_i = s | y_{i-1} = y_{i-1}^*, x)$$

Note that we need to condition on the predicted label from the previous step y_{i-1}^* here as this is now known in the inference/test time.

Viterbi Decoding for MEMM

In HMM, we infer the best label sequence via the **joint probability** $\operatorname{argmax}_y P(x, y)$ using the recurrence:

$$v_t(s) = \max_{s' \in Y} [v_{t-1}(s') P(y_t = s | y_{t-1} = s') P(x_t | y_t = s)]$$

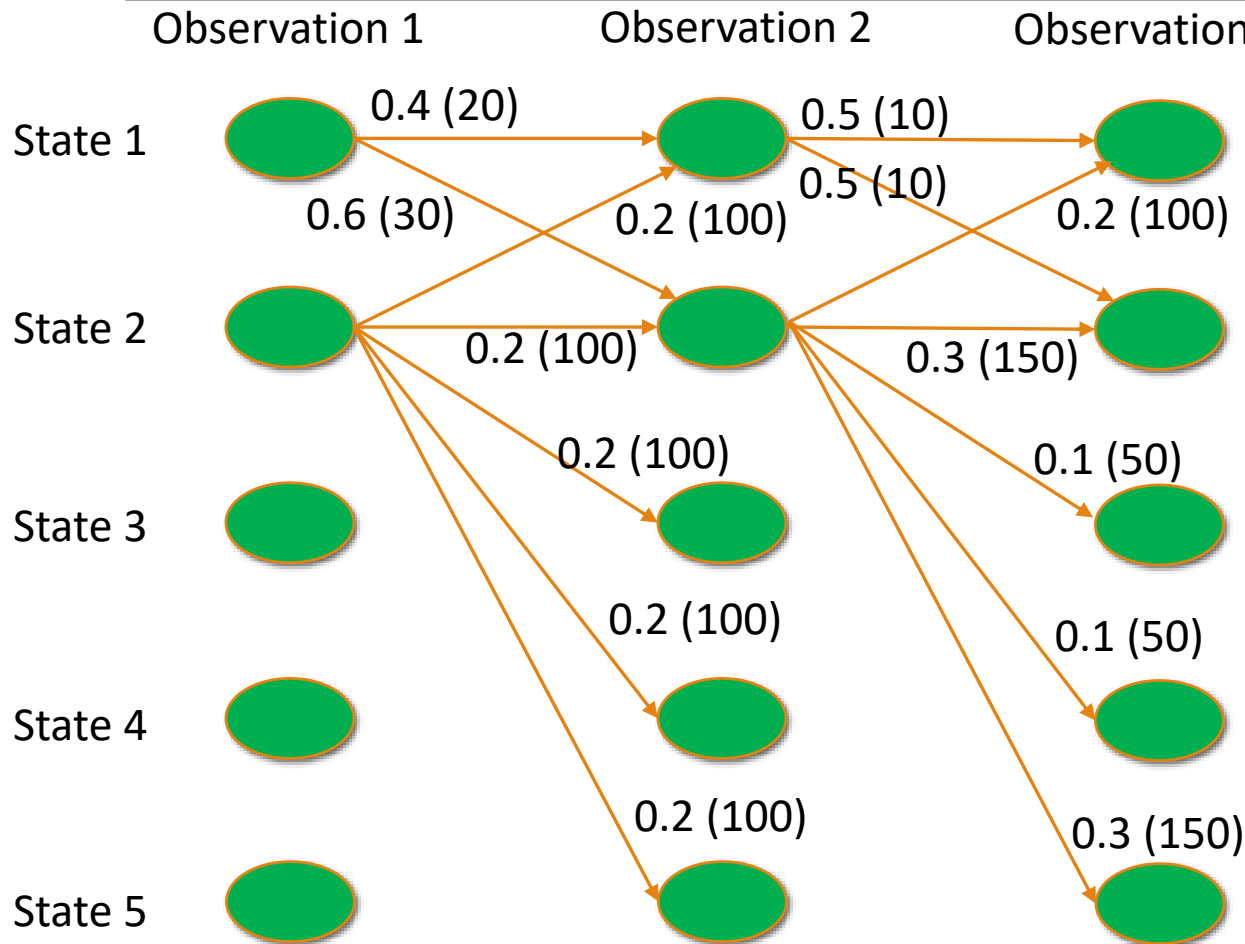
In MEMM, we infer the best label sequence via the **conditional probability** $\operatorname{argmax}_y P(y|x)$ using the recurrence:

$$v_t(s) = \max_{y_1, y_2, \dots, y_{t-1}} P(y_1, y_2, \dots, y_{t-1}, y_t = s | x)$$

$$v_t(s) = \max_{s' \in Y} (v_{t-1}(s') P(y_t = s | y_{t-1} = s', x))$$

$$p^* = \max_{S \in Y} v_n(S)$$

The Label Bias Problem in MEMM



The scores in the bracket represent the ability to go from one state to another state given the observation, i.e., $\exp(\sum_{i=1}^K w_i^{y_t} f_i(y_{t-1}, x))$

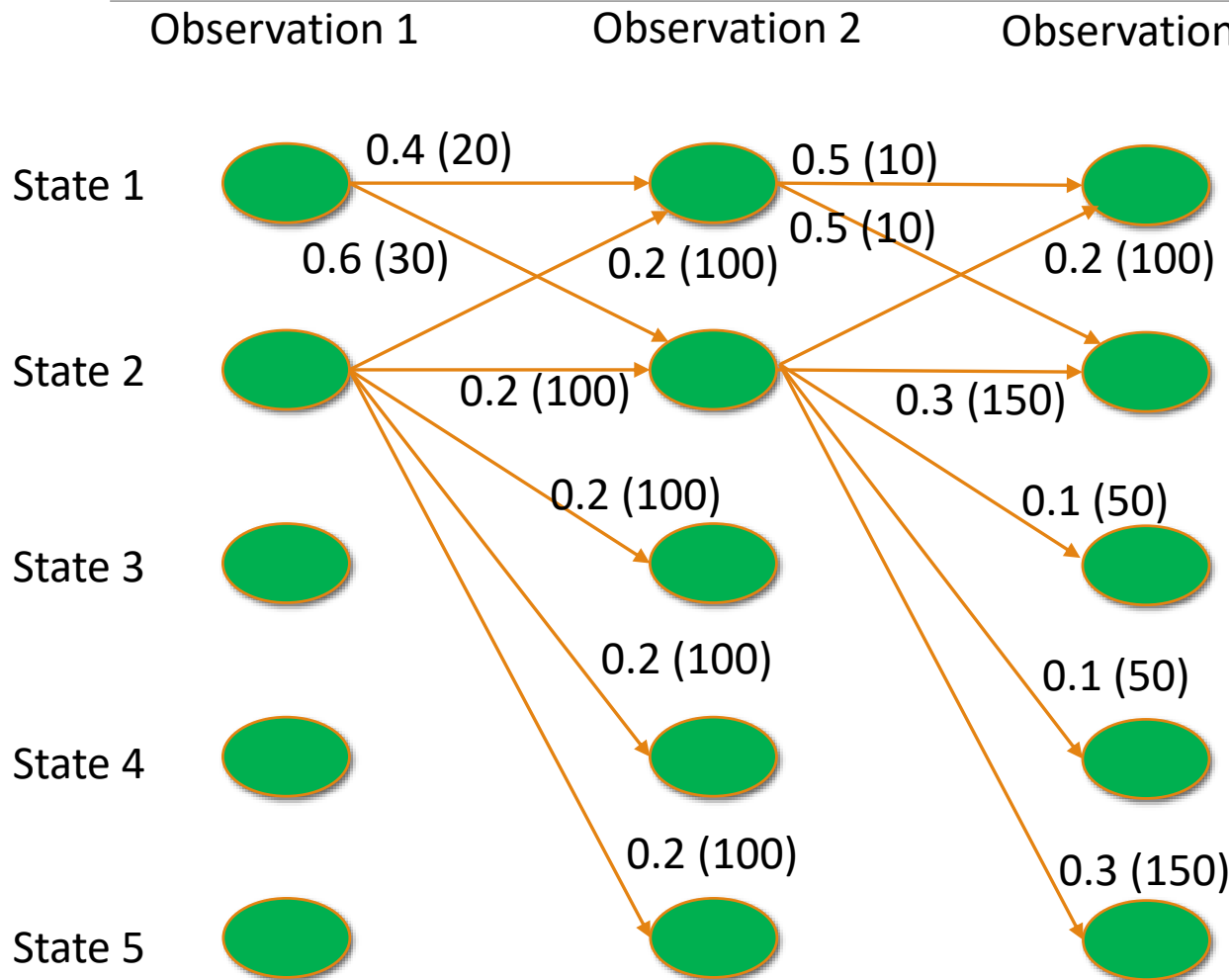
Based on these scores, the best paths should be: 2 -> 2 or 2 -> 2 -> 5

However, if we normalize at each state to obtain the probabilities, the best paths should be: 1 -> 1 -> 1 or 1 -> 1 -> 2

$$1 \rightarrow 1 \rightarrow 1, 1 \rightarrow 1 \rightarrow 2: 0.4 * 0.5 = 0.2$$

$$2 \rightarrow 2 \rightarrow 2, 2 \rightarrow 2 \rightarrow 5: 0.2 * 0.3 = 0.06$$

The Label Bias Problem in MEMM



This is because the prediction at each state/word is modeled by a probability, thus necessitating the normalization at each state (**local normalization**)

So, we want to avoid the normalization at each step and only normalize once over the entire input sequence to obtain the overall probability $P(y|x)$ (**global normalization**), leading to **Conditional Random Fields (CRF)**

An Example of Label Bias

Consider a simple MEMM for person and location names, in which states are (all names are two tokens):

- other
- B-person and e-person for person names
- B-locn and e-locn for location names

Corpus:

- Harvey Ford : person 9 times, location 1 time
- Harvey Park: location 9 times, person 1 time
- Myrtle Ford: person 9 times, location 1 time
- Myrtle Park: location 9 times, person 1 time

Second token a good indicator of person vs. location

An Example of Label Bias

Conditional probabilities:

- $p(\text{b-person} \mid \text{other}, w = \text{Harvey}) = 0.5$
- $p(\text{b-locn} \mid \text{other}, w = \text{Harvey}) = 0.5$
- $p(\text{b-person} \mid \text{other}, w = \text{Myrtle}) = 0.5$
- $p(\text{b-locn} \mid \text{other}, w = \text{Myrtle}) = 0.5$
- $p(\text{e-person} \mid \text{b-person}, w = \text{Ford}) = 1$
- $p(\text{e-person} \mid \text{b-person}, w = \text{Park}) = 1$
- $p(\text{e-locn} \mid \text{b-locn}, w = \text{Ford}) = 1$
- $p(\text{e-locn} \mid \text{b-locn}, w = \text{Park}) = 1$

Corpus:

Harvey Ford : person 9 times, location 1 time
Harvey Park: location 9 times, person 1 time
Myrtle Ford: person 9 times, location 1 time
Myrtle Park: location 9 times, person 1 time

Role of second token in distinguishing person vs. location completely lost

Problem: probabilities of outgoing arcs normalized separately for each state: the “label bias” problem

Conditional Random Fields (CRF)

Conditional Random Fields (CRFs) address this problem:

- MEMMs use a per-state exponential model
- CRFs have a single exponential model for the joint probability of the entire label sequence

Graphical comparison among HMMs, MEMMs and CRFs

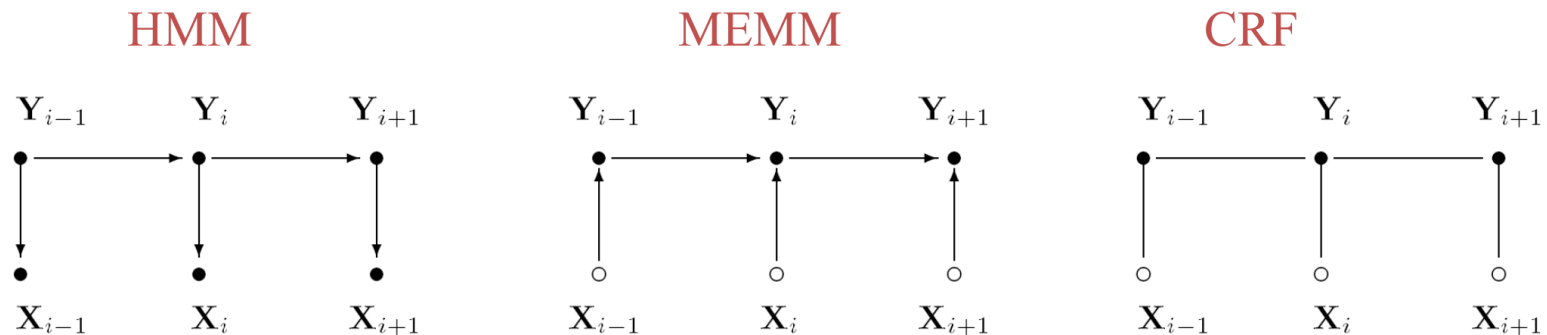


Figure 2. Graphical structures of simple HMMs (left), MEMMs (center), and the chain-structured case of CRFs (right) for sequences. An open circle indicates that the variable is not generated by the model.

From Lafferty et al.

Conditional Random Fields (CRF)

Both MEMM and CRF directly model $P(y|x)$

MEMM:

$$P(y|x; \theta) = \prod_{t=1}^n P(y_t | y_{t-1}, x; \theta)$$

CRF:

$$P(y|x; \theta) = \frac{\exp(\Phi(x, y)^T \theta)}{\sum_{y' \in Y} \exp(\Phi(x, y')^T \theta)}$$

Conditional Random Fields (CRF)

$$P(y|x; \theta) = \frac{\exp(\Phi(x,y)^T \theta)}{\sum_{y' \in Y} \exp(\Phi(x,y')^T \theta)} = \frac{\exp(\Phi(x,y)^T \theta)}{Z(x)}$$

Where

$$\Phi(x, y) = [\Phi_1(x, y), \dots, \Phi_k(x, y), \dots, \Phi_K(x, y)]$$
$$\Phi_k(x, y) = \sum_{i=1, \dots, n} \phi_k(y_{i-1}, y_i, x, i)$$

with $\phi_k(y_{i-1}, y_i, x, i)$ is a function to capture some features of the input sentence x and the transition from state y_{i-1} to state y_i at step i

- i.e., only capturing features at the edge and node level and **similar to those we use for MEMM**

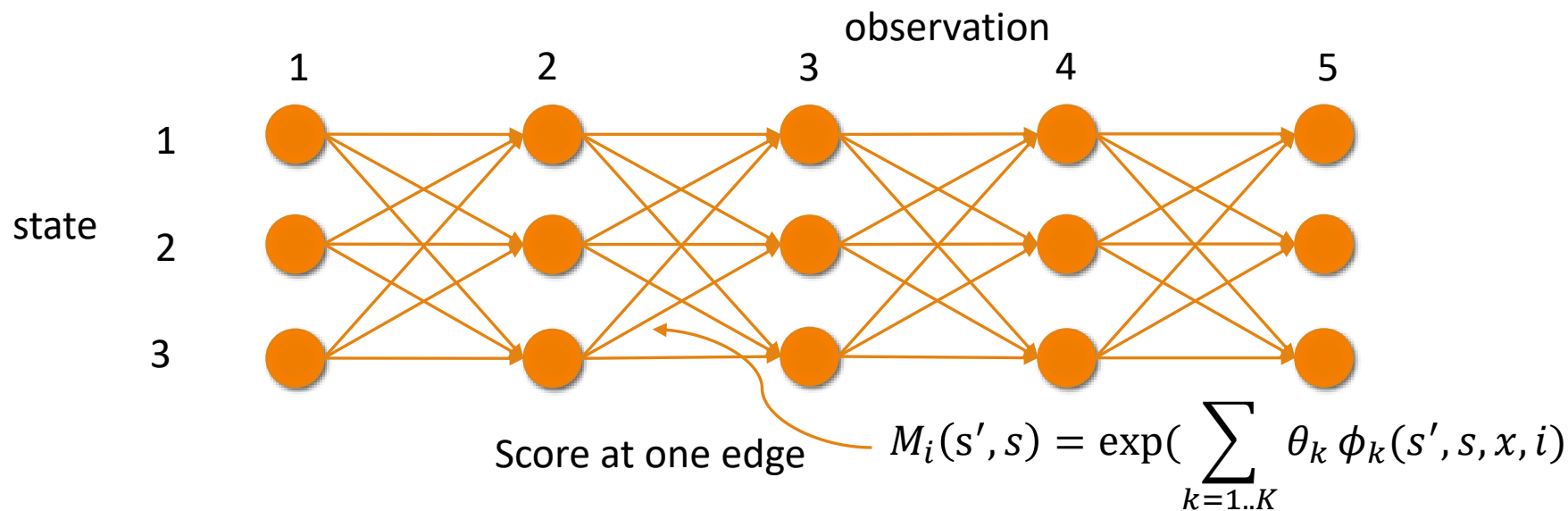
The element of θ corresponding to $\Phi_k(x, y)$ is θ_k

Conditional Random Fields (CRF)

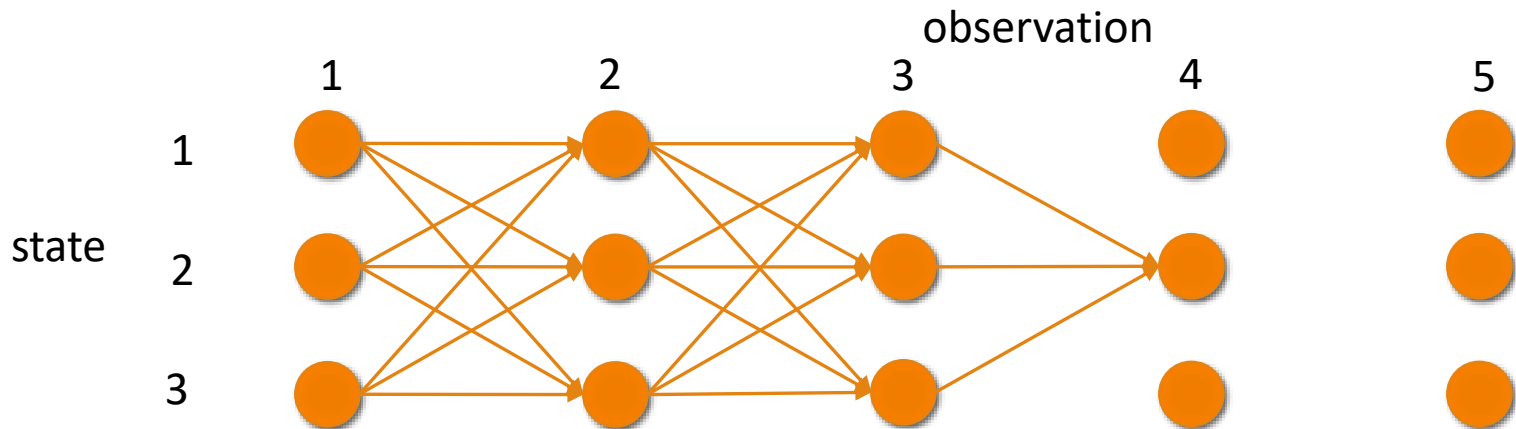
$$P(y|x; \theta) = \frac{\exp(\Phi(x,y)^T \theta)}{\sum_{y' \in Y} \exp(\Phi(x,y')^T \theta)} = \frac{\exp(\Phi(x,y)^T \theta)}{Z(x)}$$

The normalizing factor $Z(x)$ involve summing over an exponential number of terms (all the possible label sequence for the input sentence -- $|Y|^n$)

Using dynamic programming (i.e., the forward algorithm), we can the normalization in $O(n|Y|^2)$



Conditional Random Fields (CRF)



$\alpha_i(s)$: the total score for the length- i subpaths of the paths whose i -th state is s .

Initialization:

$$\alpha_1(s) = \exp(\sum_{k=1..K} \theta_k \phi_k(\text{start}, s, x, 1))$$

Recurrence:

$$\alpha_i(s) = \sum_{s' \in Y} \alpha_{i-1}(s') M_i(s', s)$$

Final normalization score:

$$Z(x) = \sum_{s \in Y} \alpha_n(s)$$

CRF Training

Loss function:

$$L(\theta) = -\log P(y|x; \theta) = -\log \frac{\exp(\Phi(x,y)^T \theta)}{\sum_{y' \in Y} \exp(\Phi(x,y')^T \theta)} = -\Phi(x,y)^T \theta + \log Z(x)$$

In most of the optimization technique for $L(\theta)$, we will need to compute its gradient:

$$\frac{\partial L(\theta)}{\partial \theta_k} = -\phi_k(x,y) + \sum_{y' \in Y} \frac{\exp(\Phi(x,y')^T \theta) \phi_k(x,y')}{Z(x)} = -\phi_k(x,y) + \boxed{\sum_{y' \in Y} P(y'|x) \phi_k(x,y')}$$

$$\sum_{y' \in Y} P(y'|x) \phi_k(x,y') = \sum_{i=1..n} \sum_{s' \in Y, s \in Y} \phi_k(s', s, x, i) \sum_{y' : y'_{i-1} = s', y'_i = s} P(y'|x)$$

Re-arrange variables for DP



Using this factorization, we can compute this quantity in $O(n|Y|^2)$ using the forward-backward algorithm

For details, see: Collins, "The Forward-Backward Algorithm"

Viterbi decoding for CRF

$$v_t(s) = \max_{y_1, y_2, \dots, y_{t-1}} P(y_1, y_2, \dots, y_{t-1}, y_t = s | x)$$

Initialization:

$$v_1(s) = \sum_{k=1..K} \exp(\theta_k \phi_k(\text{start}, s, x, 1))$$

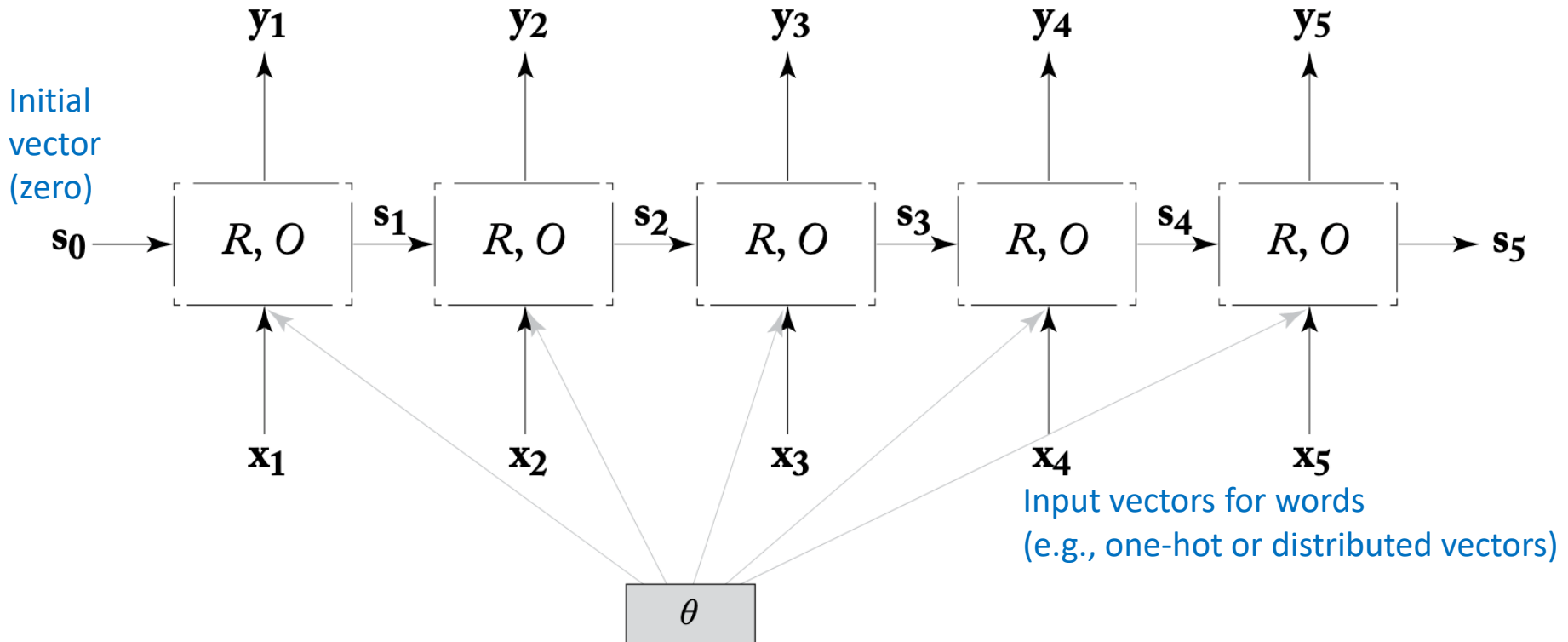
Recurrence:

$$v_i(s) = \max_{s' \in Y} [\alpha_{i-1}(s') M_i(s', s)]$$

Best score:

$$p^* = \max_{s \in Y} v_n(s)$$

Recurrent Neural Networks (RNN)



R : recurrence function

O : output function

s_i, y_i : hidden vector and output vector at step i .

θ : model parameters (to be learned during training)

Goldberg, 2017

Recurrent Neural Networks (RNN)

At each step, the R function takes two inputs (i.e., the hidden vector from the previous step s_{t-1} and the input vector from the current step x_t) to compute the hidden vector for the current step s_t :

$$s_t = R(s_{t-1}, x_t)$$

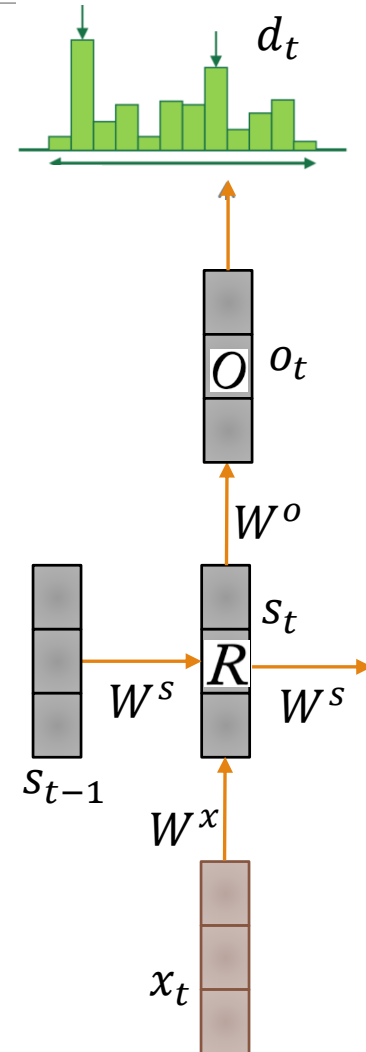
The hidden vector s_t can be used as the feature vector to make a prediction about the label for x_t (i.e., POS or NER). Essentially, we use the O function to transform s_t into a score vector o_t whose dimensions quantify the likelihood that x_t has the corresponding labels (i.e., $|o_t| = |Y|$):

$$o_t = O(s_t W^o + b^o)$$

o_t can be transformed into a probability distribution via the softmax function:
 $d_t = \text{softmax}(o_t)$

In the simplest version (i.e., vanilla RNN), O can be just the identity function (i.e., $O(x) = x$), while R can be a simple linear transformation followed by a non-linear function:

$$s_t = \sigma(s_{t-1} W^s + x_t W^x + b^s)$$



Recurrent Neural Networks (RNN)

The model parameters: $\theta = \{W^s, W^x, b^s, W^o, b^o\}$

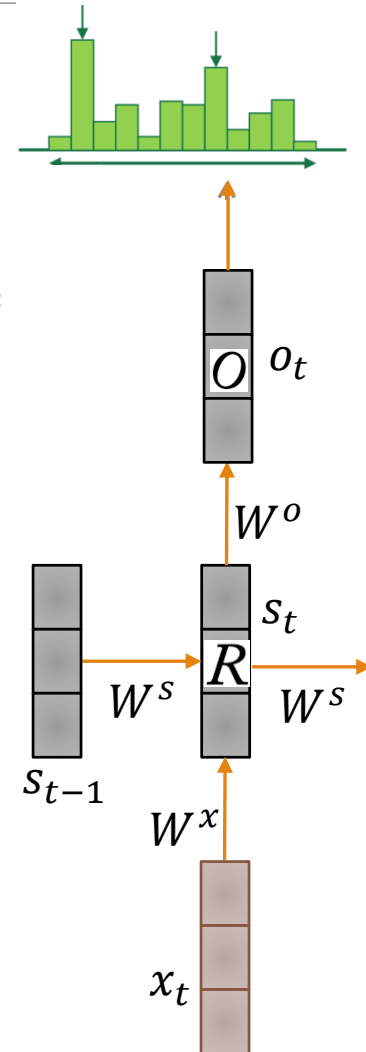
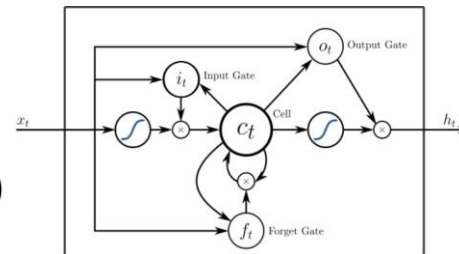
The recurrence nature (i.e., using the hidden vector from the previous step for the current computation) allows each hidden vector s_t to capture information about all the words before t : $s_t = f(s_0, s_1, \dots, s_{t-1})$

The use of the same parameters W^s, W^x, b^s in the recurrence function R causes the *gradient vanishing* problem (i.e., gradient becomes small in long sentences so the models cannot learn)

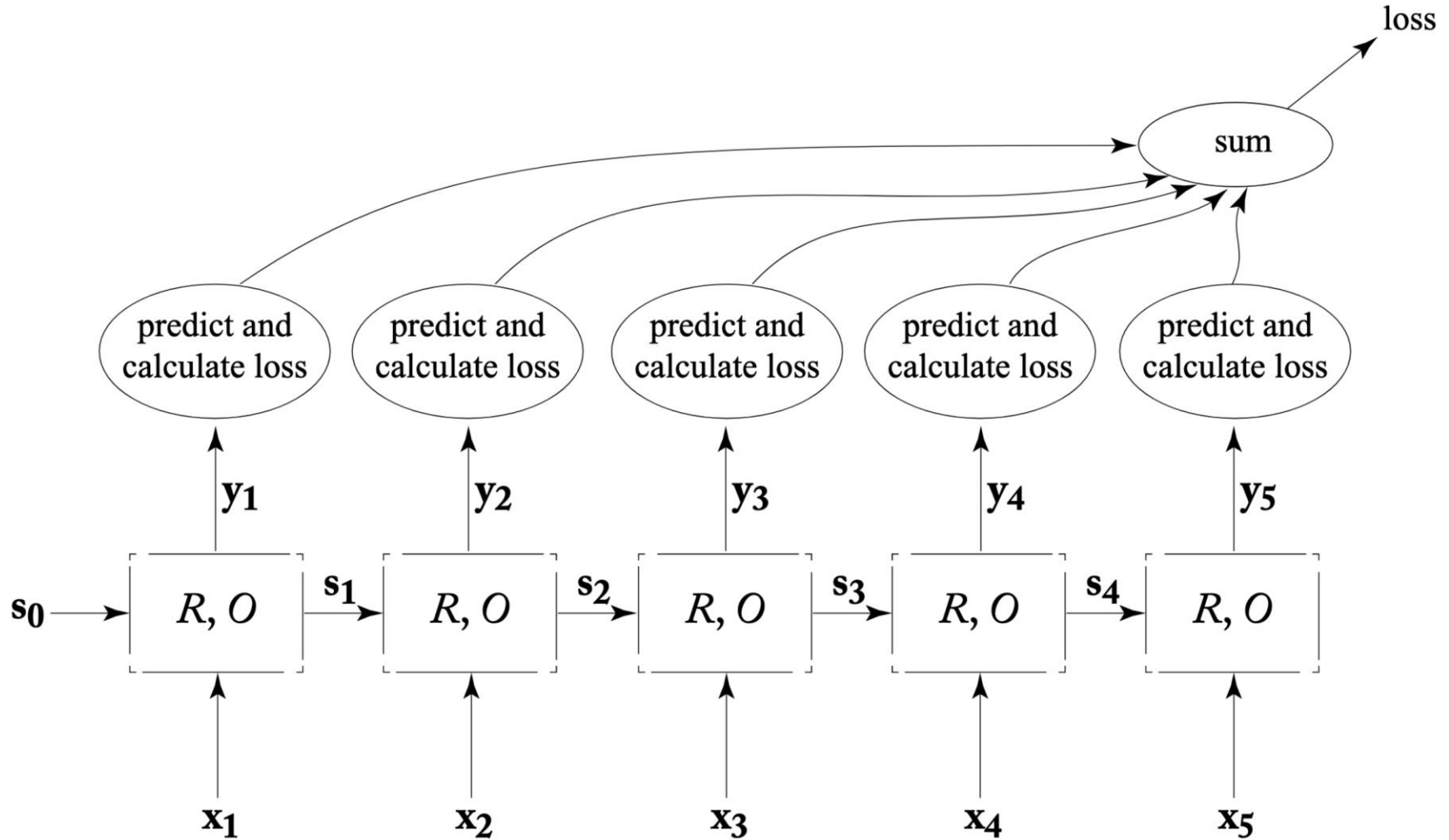
In practice, the LSTM cell is often used for R to mitigate this problem.

LSTM units allow gradients to also flow *unchanged*

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



Training RNN



Bidirectional RNN

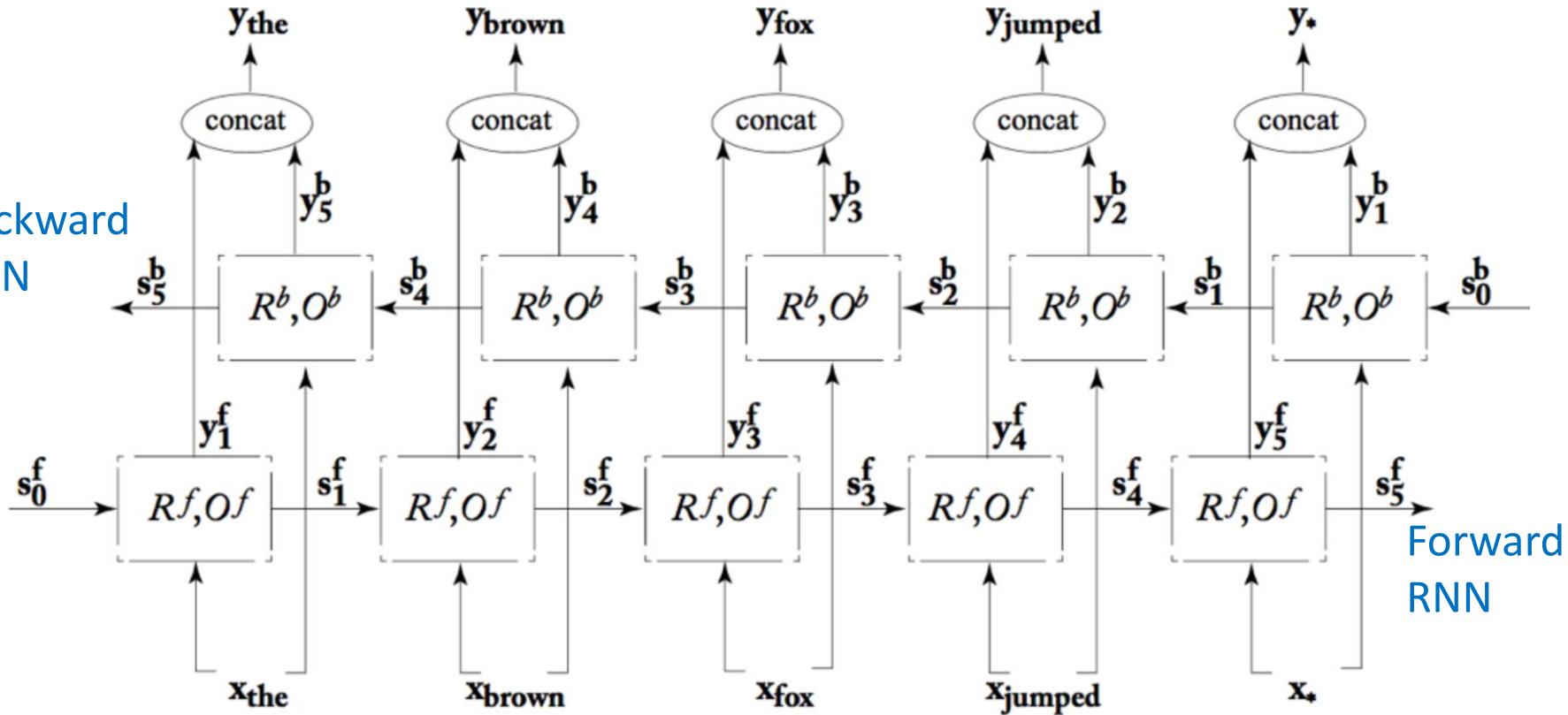
A city or a football team?

Liverpool suffered an upset first home league defeat of the season, beaten 1-0 by a Guy Whittingham goal for Sheffield Wednesday.

- The information on the left is not enough to predict the label for the current word.

Bidirectional RNN

Backward
RNN



Forward
RNN

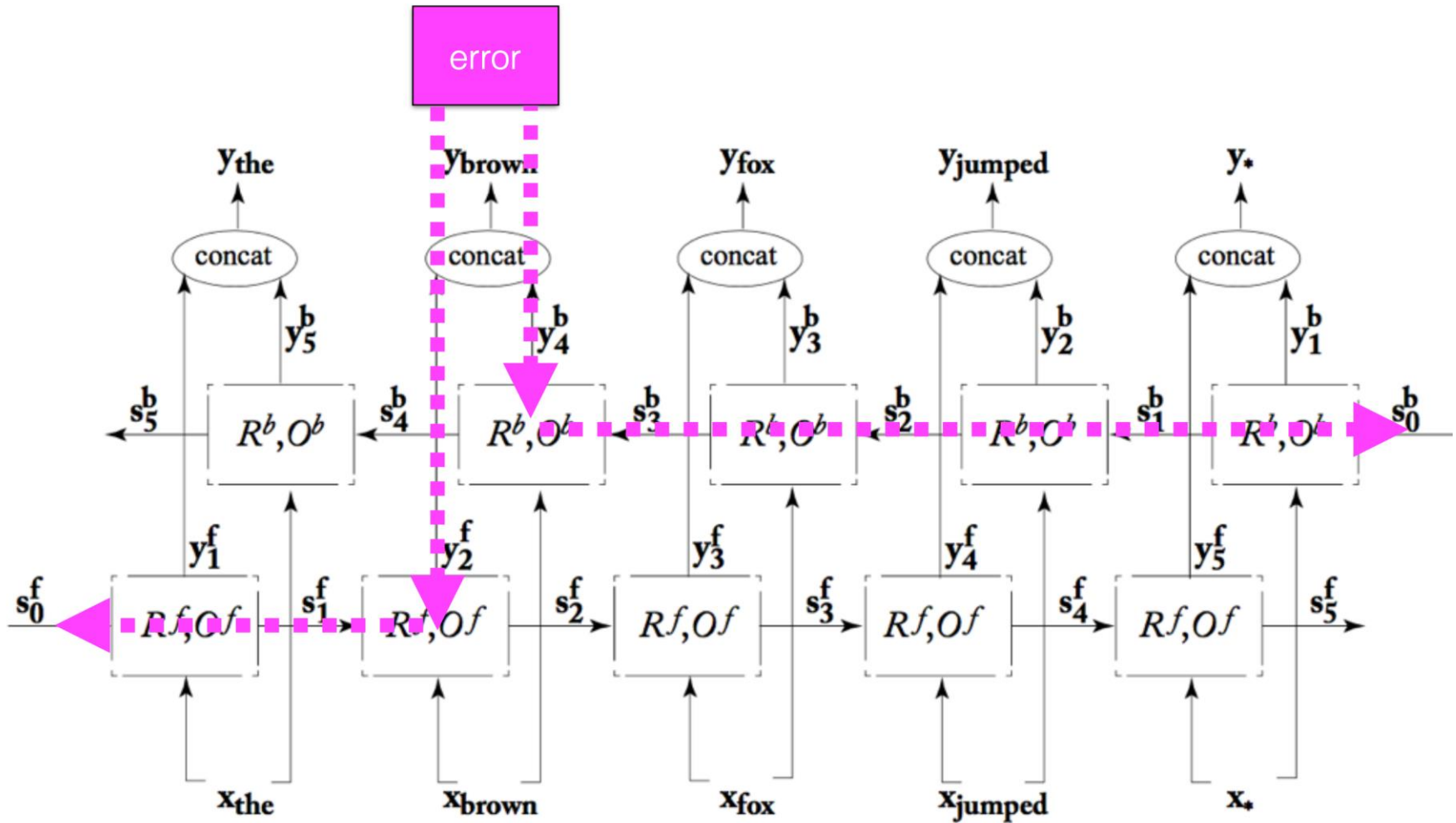
$$s_i^f = \sigma(s_{i-1}^f W_s^f + x_i W_x^f + b^f)$$

$$s_i^b = \sigma(s_{i-1}^b W_s^b + x_i W_x^b + b^b)$$

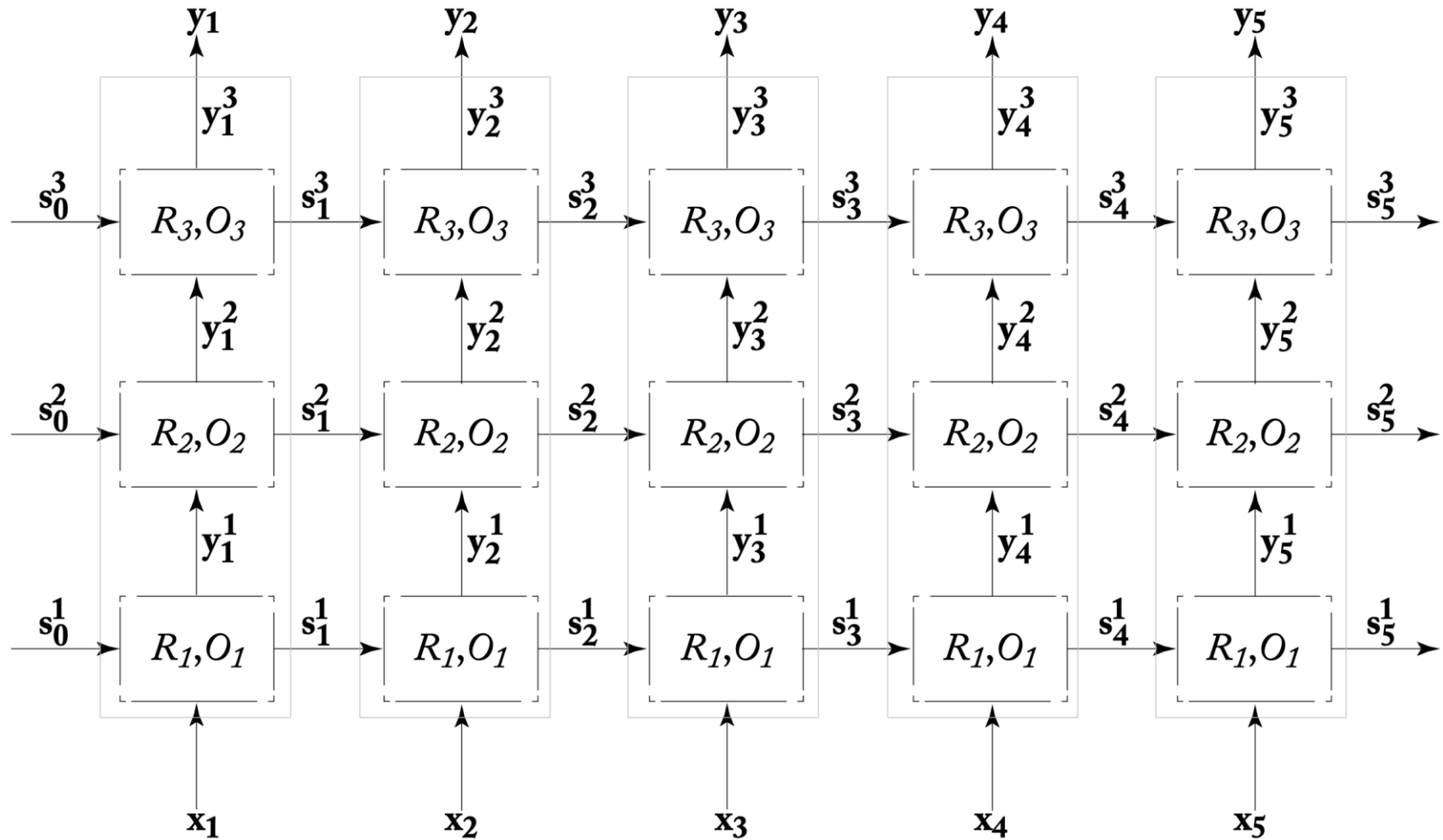
$$y_i = \text{softmax}([s_i^f, s_i^b] W^o + b^o), \theta = [W_s^f, W_x^f, b^f, W_s^b, W_x^b, b^b, W^o, b^o]$$

So, one hidden vector has access to the context information across the whole sentence

Bidirectional RNN



Go Deeper (Stacked RNN)



Incorporating CRF

RNN makes prediction for words independently

- The features/representations share the parameters
- But the output predictions are independent

We want to capture the dependencies between the output labels, i.e., I_PER can only be preceded by B_PER

- The later predictions can influence the prior predictions (e.g., fixing prior's error)

CRF can achieve this via the global normalization of the label sequence probabilities

Idea: Incorporate CRF as the final layer in the RNN models for sequence labeling

Incorporating CRF

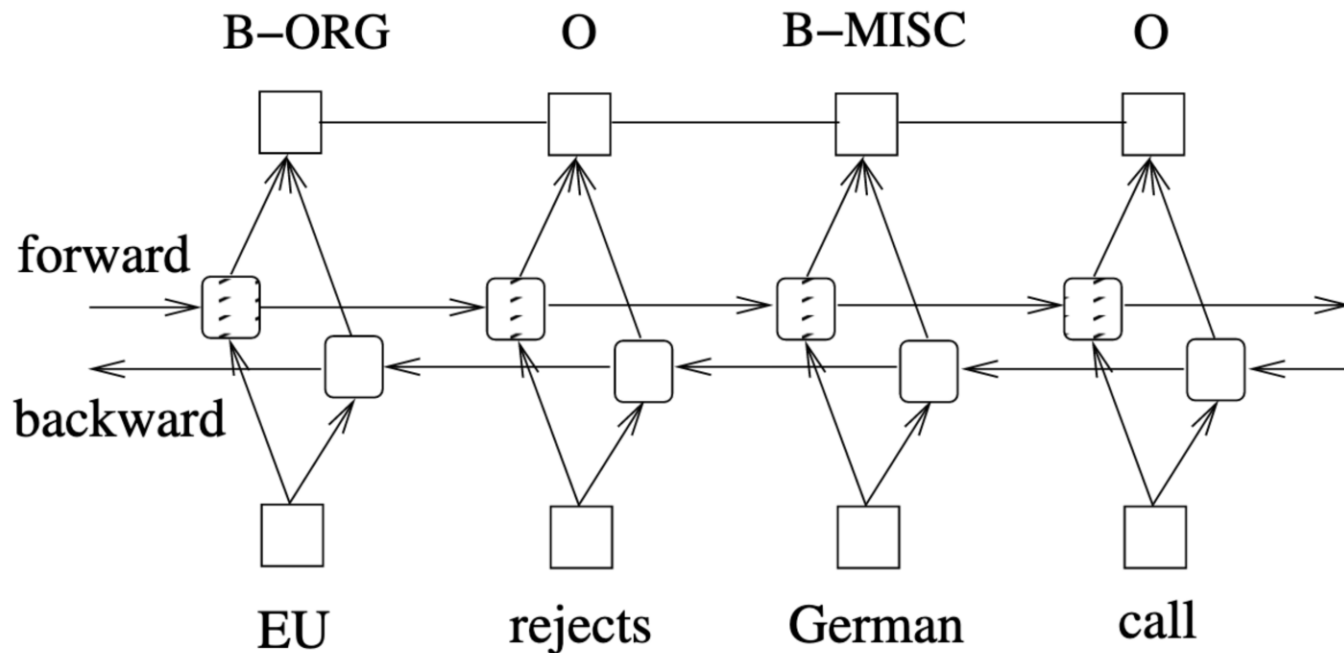
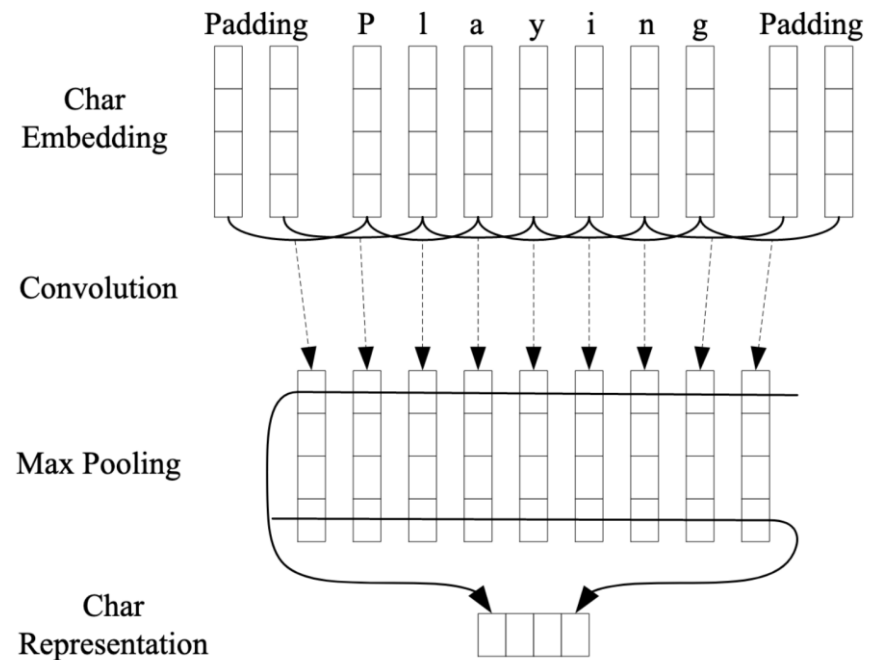
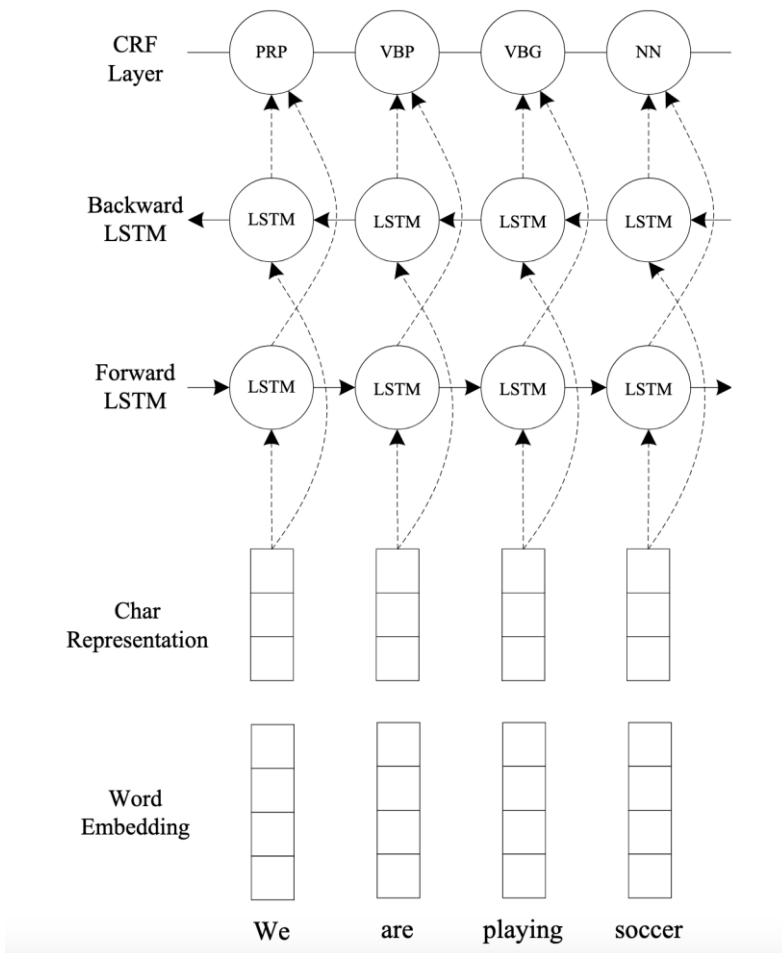


Figure 7: A BI-LSTM-CRF model.

Huang et al. 2015, "Bidirectional LSTM-CRF Models for Sequence Tagging"

Incorporating CRF



Ma and Hovy (2016), "End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF"

Incorporating CRF

Layer	Hyper-parameter	POS	NER
CNN	window size	3	3
	number of filters	30	30
LSTM	state size	200	200
	initial state	0.0	0.0
	peepholes	no	no
Dropout	dropout rate	0.5	0.5
	batch size	10	10
	initial learning rate	0.01	0.015
	decay rate	0.05	0.05
	gradient clipping	5.0	5.0

Model	POS		NER					
	Dev	Test	Dev			Test		
	Acc.	Acc.	Prec.	Recall	F1	Prec.	Recall	F1
BRNN	96.56	96.76	92.04	89.13	90.56	87.05	83.88	85.44
BLSTM	96.88	96.93	92.31	90.85	91.57	87.77	86.23	87.00
BLSTM-CNN	97.34	97.33	92.52	93.64	93.07	88.53	90.21	89.36
BRNN-CNN-CRF	97.46	97.55	94.85	94.63	94.74	91.35	91.06	91.21