SEQUENCE LABELING

Natural Language Processing

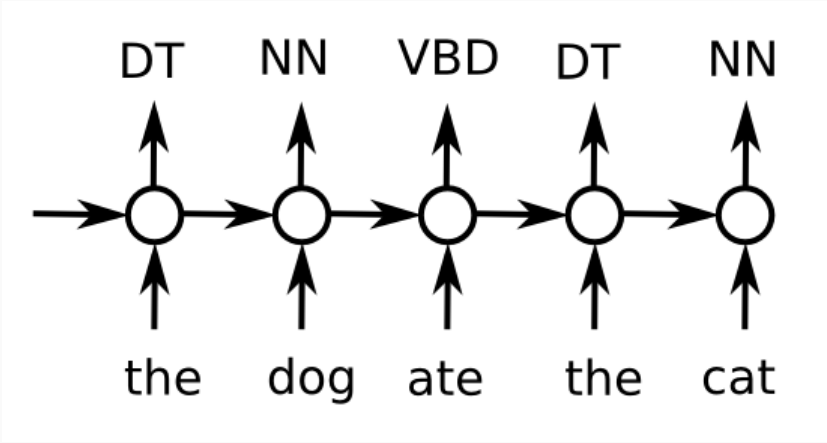
Sergey Nikolenko

Harbour Space University, Barcelona, Spain
January 18, 2018

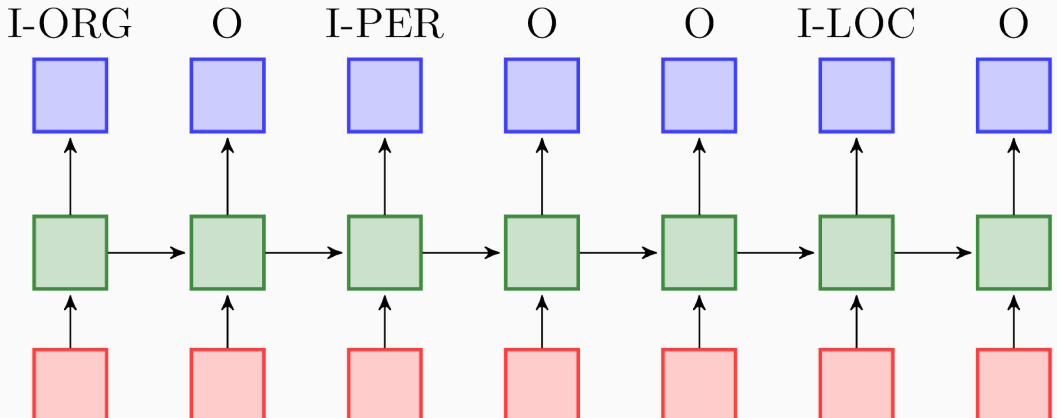
MOTIVATION

SEQUENCE LABELING

- Given a sequence of observations, find an appropriate label/state for each observation.
- The problem is to treat the sequence as a sequence, not just independent classification:
  - part-of-speech tagging



  - named entity recognition



SEQUENCE LABELING

- What kind of methods would you propose?
- Local classifiers: predict y from x:

$$y \approx \mathbf{w}^\top f(\mathbf{x}, i).$$

- We can have feature-rich classifiers with lots of different features, but predictions will be independent for each $\mathbf{x}_i$.
- Anything else?

HIDDEN MARKOV MODELS

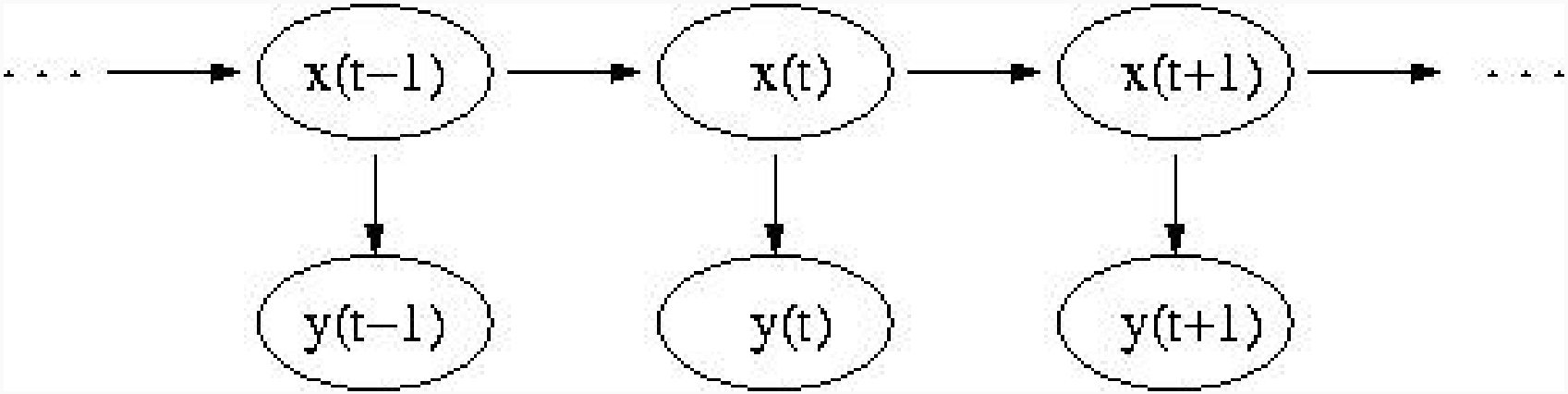
- A Markov chain is defined by initial probability distribution $p^0(x)$ and transition probabilities $T(x'; x)$.
- $T(x'; x)$ is the distribution of the next element in the chain depending on the previous one; distribution on step $(t + 1)$ is

$$p^{t+1}(x') = \int T(x'; x)p^t(x)dx.$$

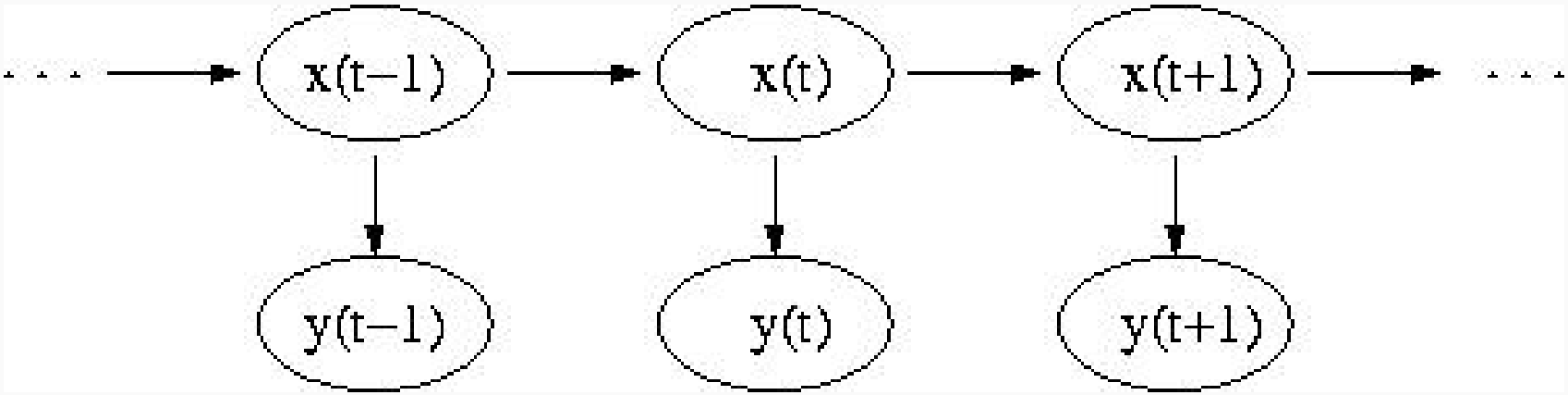
- In the discrete case, $T(x'; x)$ is a matrix of probabilities $p(x' = i|x = j)$.

DISCRETE MARKOV CHAINS

- We are in the discrete case.
- A Markov model is when we can observe certain functions of a Markov chain.



- Here $x(t)$ is the process (chain states) itself, and $y(t)$ are observables.
- The problem is to find hidden parameters of the process.



- Markov property: next state does not depend on the history, only on the previous state:

$$p(x(t) = x_j | x(t-1) = x_{j_{t-1}}, \dots, x(1) = x_{j_1}) = \\ = p(x(t) = x_j | x(t-1) = x_{j_{t-1}}).$$

- Moreover, these probabilities $a_{ij} = p(x(t) = x_j | x(t-1) = x_i)$ do not depend on t.
- These probabilities comprise the transition matrix $A = (a_{ij})$, with natural properties $a_{ij} \geq 0$, $\sum_j a_{ij} = 1$.

DIRECT PROBLEM

- Natural problem: what is the probability to get a certain sequence of events?
- I.e., for a sequence $Q = q_{i_1} \dots q_{i_k}$ find

$$p(Q|\text{model}) = p(q_{i_1})p(q_{i_2}|q_{i_1}) \dots p(q_{i_k}|q_{i_{k-1}}).$$

- Looks trivial. What's hard in the real world?

HIDDEN MARKOV MODELS

- In the real world we do not know the model.
- And, moreover, we do not observe $x(t)$, i.e., real model states, but rather $y(t)$, i.e., observe functions of them (data).
- Example: speech recognition.

PROBLEMS IN HIDDEN MARKOV MODELS

- First: find the probability of a sequence of observations in a given model.
- Second: find the “optimal” sequence of states in a given model and a given sequence of observations.
- Third: find the maximum likelihood model (model parameters).

STATES AND OBSERVABLES

- $X = \{x_1, \dots, x_n\}$ — set of states.
- $V = \{v_1, \dots, v_m\}$ — alphabet from which we choose observables y (set of values of y).
- q_t — state at time t, y_t — observable at time t.

DISTRIBUTIONS

- $a_{ij} = p(q_{t+1} = x_j | q_t = x_i)$ — transition probability from i to j.
- $b_j(k) = p(v_k | x_j)$ — probability to get data v_k in state j.
- Initial distribution $\pi = \{\pi_j\}$, $\pi_j = p(q_1 = x_j)$.
- We denote the data by $D = d_1 \ldots d_T$ (sequence of observables, d_i take values from V).

PROBLEM

- We can now formalize the problem setting.
- First problem: for a given model $\lambda = (A, B, \pi)$ and sequence D, find $p(D|\lambda)$. By itself it simply shows how well the model fits this data.
- Second problem: for a given model λ and sequence D find the "optimal" sequence of states $Q = q_1 \ldots q_T$. Two kinds of optimality: "bitwise" and general.
- Third problem: optimize model parameters $\lambda = (A, B, \pi)$ in order to maximize $p(D|\lambda)$ for a given D (find the maximum likelihood model). This is the main problem, training hidden Markov models.

FIRST PROBLEM

- Formally the first problem looks like

$$p(D|\lambda) = \sum_{Q} p(D|Q,\lambda)p(D|\lambda) =$$

$$= \sum_{q_1,\dots,q_T} b_{q_1}(d_1)\dots b_{q_T}(d_T)\pi_{q_1} a_{q_1 q_2}\dots a_{q_{T-1}q_T}.$$

FIRST PROBLEM

- This is a marginalization problem.
- We use the so-called *forward–backward procedure*, in essence dynamical programming on a lattice.
- We will sequentially compute intermediate values of the form

$$\alpha_t(i) = p(d_1 \dots d_t, q_t = x_i | \lambda),$$

  i.e., the required probabilities with account for current state.

SOLVING THE FIRST PROBLEM

- Initialize $\alpha_1(i) = \pi_i b_i(d_1)$.
- Induction step:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{n} \alpha_t(i) a_{ij}\right] b_j(d_{t+1}).$$

- After we get to step T, we can compute what we need:

$$p(D|\lambda) = \sum_{i=1}^{n} \alpha_T(i).$$

- This is simply the forward pass, we did not need a backward pass here.
- What would it compute?

BACKWARD PASS

- It would compute conditional probabilities $\beta_t(i) = p(d_{t+1} \dots d_T | q_t = x_i, \lambda)$.
- We can initialize $\beta_T(i) = 1$ and proceed by induction:

$$\beta_t(i) = \sum_{j=1}^{n} a_{ij} b_j(d_{t+1}) \beta_{t+1}(j).$$

- We'll need it later to solve the second and third problems.

TWO VERSIONS OF THE SECOND PROBLEM

- There are two versions for the second problem.
- First, solve it “bit by bit”: “what is the most probable state at time j?”
- Second, solve it “in general”: “what is the most probable sequence of states?”.

BITWISE SOLUTION

- Consider auxiliary variables

$$\gamma_t(i) = p(q_t = x_i | D, \lambda).$$

- The problem is to find

$$q_t = \arg\max_{1 \leq i \leq n} \gamma_t(i), \quad 1 \leq t \leq T.$$

- How can we do it?

BITWISE SOLUTION

- We express them via α and β:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{p(D|\lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{n} \alpha_t(i)\beta_t(i)}.$$

- The denominator does not matter since we need $\arg\max$.

SOLVING FOR THE SEQUENCE

- To find the most probable sequence, we will use the so-called *Viterbi algorithm* (that is, dynamic programming).
- Now auxiliary variables are

$$\delta_t(i) = \max_{q_1,\cdots,q_{t-1}} p\left(q_1 q_2 \ldots q_t = x_i, d_1 d_2 \ldots d_t | \lambda\right).$$

SOLVING FOR THE SEQUENCE

- That is, $\delta_t(i)$ is the maximal probability to reach state x_i on step t among all paths with given observables.
- By induction:

$$\delta_{t+1}(j) = \left[\max_i \delta_t(i) a_{ij}\right] b_j(d_{t+1}).$$

- Note that we also need to remember the arguments, not only values; $\psi_t(j)$ on the next slide.

SOLVING FOR THE SEQUENCE: ALGORITHM

- Initialize $\delta_1(i) = \pi_i b_i(d_1)$, $\psi_1(i) = []$.
- Induction:

$$\delta_t(j) = \max_{1 \le i \le n} \left[\delta_{t-1}(i) a_{ij}\right] b_j(d_t),$$

$$\psi_t(j) = \arg\max_{1 \le i \le n} \left[\delta_{t-1}(i) a_{ij}\right].$$

- When we reach step T, final step:

$$p^* = \max_{1 \le i \le n} \delta_T(i), \qquad q_T^* = \arg\max_{1 \le i \le n} \delta_T(i).$$

- And the sequence follows: $q_t^* = \psi_{t+1}(q_{t+1}^*)$.

THIRD PROBLEM

- We cannot find a global maximum of $p(D|\lambda)$ analytically.
- We will use local optimization.
- The *Baum–Welch algorithm*: a special case of EM.

AUXILIARY VARIABLES

- Now auxiliary variables are probabilities of the event that at time t we are in state x_i, and at time $t+1$ — in state x_j:

$$\xi_t(i,j) = p(q_t = x_i, q_{t+1} = x_j | D, \lambda).$$

- Rewriting via already familiar variables:

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(d_{t+1}) \beta_{t+1}(j)}{p(D|\lambda)} = \frac{\alpha_t(i) a_{ij} b_j(d_{t+1}) \beta_{t+1}(j)}{\sum_i \sum_j \alpha_t(i) a_{ij} b_j(d_{t+1}) \beta_{t+1}(j)}.$$

- Note also that $\gamma_t(i) = \sum_j \xi_t(i,j)$.

IDEA

- $\sum_t \gamma_t(i)$ is the expected number of transitions from states x_i; $\sum_t \xi_t(i,j)$, from x_i to x_j.
- On the M-step we will reestimate the probabilities:

$$\bar{\pi}_i = \text{expected frequency of } x_i \text{ on step } 1 = \gamma_1(i),$$

$$\bar{a}_{ij} = \frac{\text{no. of transitions from } x_i \text{ to } x_j}{\text{no. of transitions from } x_i} = \frac{\sum_t \xi_t(i,j)}{\sum_t \gamma_t(i)}.$$

$$\bar{b}_j(k) = \frac{\text{no. of times in } x_i \text{ observing } v_k}{\text{no. of times in } x_i} = \frac{\sum_{t:d_t=v_k} \gamma_t(i)}{\sum_t \gamma_t(i)}.$$

- EM-algorithm: start with $\lambda = (A, B, \pi)$, compute $\bar{\lambda} = (\bar{A}, \bar{B}, \bar{\pi})$, recompute the parameters again, and so on.

Kullback--Leibler divergence

- Kullback–Leibler divergence is an information theoretic measure of how different two distributions are:

$$D_{KL}(p_1, p_2) = \sum_x p_1(x) \log \frac{p_1(x)}{p_2(x)}.$$

- It is nonnegative and equal to zero only if $p_1 \equiv p_2$ (with probability 1).

- We define

$$p_1(Q) = \frac{p(Q, D|\lambda)}{p(D|\lambda)}, \quad p_2(Q) = \frac{p(Q, D|\lambda')}{p(D|\lambda')}.$$

- Then p_1 and p_2 are distributions, and the Kullback–Leibler divergence is

$$0 \leq D_{LK}(\lambda, \lambda') = \sum_Q \frac{p(Q, D|\lambda)}{p(D|\lambda)} \log \frac{p(Q, D|\lambda)p(D|\lambda')}{p(Q, D|\lambda')p(D|\lambda)} =$$

$$= \log \frac{p(D|\lambda')}{p(D|\lambda)} + \sum_Q \frac{p(Q, D|\lambda)}{p(D|\lambda)} \log \frac{p(Q, D|\lambda)}{p(Q, D|\lambda')}.$$

AUXILIARY FUNCTION

- We introduce the auxiliary function

$$Q(\lambda, \lambda') = \sum_Q p(Q|D, \lambda) \log p(Q|D, \lambda').$$

- Then the inequality implies that

$$\frac{Q(\lambda, \lambda') - Q(\lambda, \lambda)}{p(D|\lambda)} \leq \log \frac{p(D|\lambda')}{p(D|\lambda)}.$$

- That is, if $Q(\lambda, \lambda') > Q(\lambda, \lambda)$ then $p(D|\lambda') > p(D|\lambda)$.
- That is, if we maximize $Q(\lambda, \lambda')$ w.r.t. λ', we will be moving in the right direction.

FUNCTION q

- We need to maximize $Q(\lambda, \lambda')$. We rewrite

$$Q(\lambda, \lambda') = \sum_Q p(Q|D, \lambda) \log p(Q|D, \lambda') =$$

$$= \sum_Q p(Q|D, \lambda) \log \pi_{q_1} \prod_t a_{q_{t-1} q_t} b_{q_t}(d_t) =$$

$$= \sum_Q p(Q|D, \lambda) \log \pi_{q_1} + \sum_Q p(Q|D, \lambda) \sum_t \log a_{q_{t-1} q_t} b_{q_t}(d_t).$$

- The latter expression is easy to differentiate w.r.t. a_{ij}, $b_i(k)$, and π_i, add the corresponding Lagrange multipliers, and solve.
- We'll get exactly the Baum–Welch algorithm (check it!).

LINEAR FACTORIZED MODELS

LINEAR FACTORIZED MODELS

- Local classifiers: predict y from $\mathbf{x}$:

$$\hat{y} = \mathbf{w}^\top f(\mathbf{x}, i).$$

- We can have feature-rich classifiers with lots of different features, but predictions will be independent for each $\mathbf{x}_i$.
- HMM – maximize with Viterbi:

$$\pi_{y_1} b_{y_1}(\mathbf{x}_1) \prod_{i=1}^{n} a_{y_{i-1}, y_i} b_{y_i}(\mathbf{x}_i).$$

- E.g., in POS tagging we train POS→POS transition probabilities and word emission probabilities.
- But it's hard to add lots of features here.

LINEAR FACTORIZED MODELS

- Linear factorized models:

$$\hat{y} = \arg\max_{\mathbf{y}} \sum_{i=1}^{n} \mathbf{w}^\top f(\mathbf{x}, i, y_{i-1}, y_i).$$

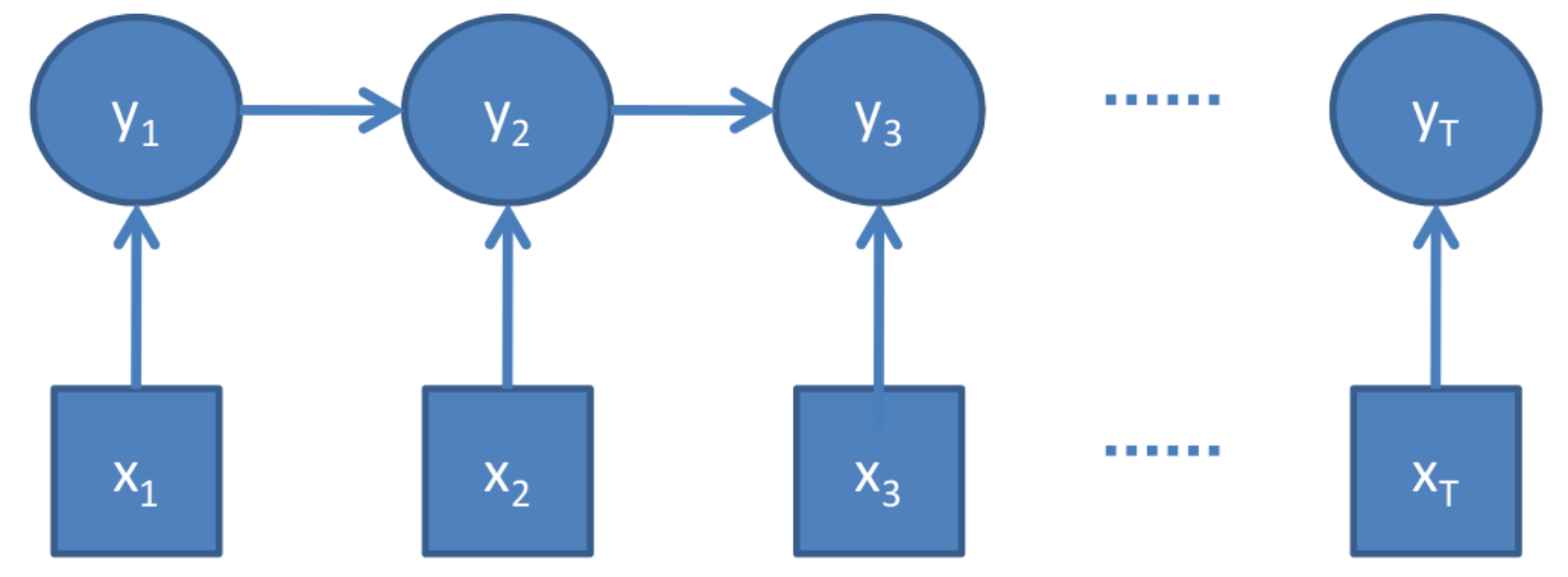
- We can find labels y_i with Viterbi algorithm such that the total sum is maximal.
- Training – structured perceptron: for several epochs,
  - for each training set sequence $(\mathbf{x}, \mathbf{y})$:
    - compute $\mathbf{z} = \arg\max_{\mathbf{z}} \mathbf{w}^\top f(\mathbf{x}, \mathbf{z})$ (with Viterbi)
    - if $\mathbf{z} \neq \mathbf{y}$ update

$$\mathbf{w} := \mathbf{w} + \eta \left(f(\mathbf{x}, \mathbf{y}) - f(\mathbf{x}, \mathbf{z}) \right).$$

- Usually averaged structured perceptron (return average weights over the training).

MAXIMAL ENTROPY MARKOV MODELS

- How can we add more features to HMMs?
- MEMM: invert the arrows in the HMM



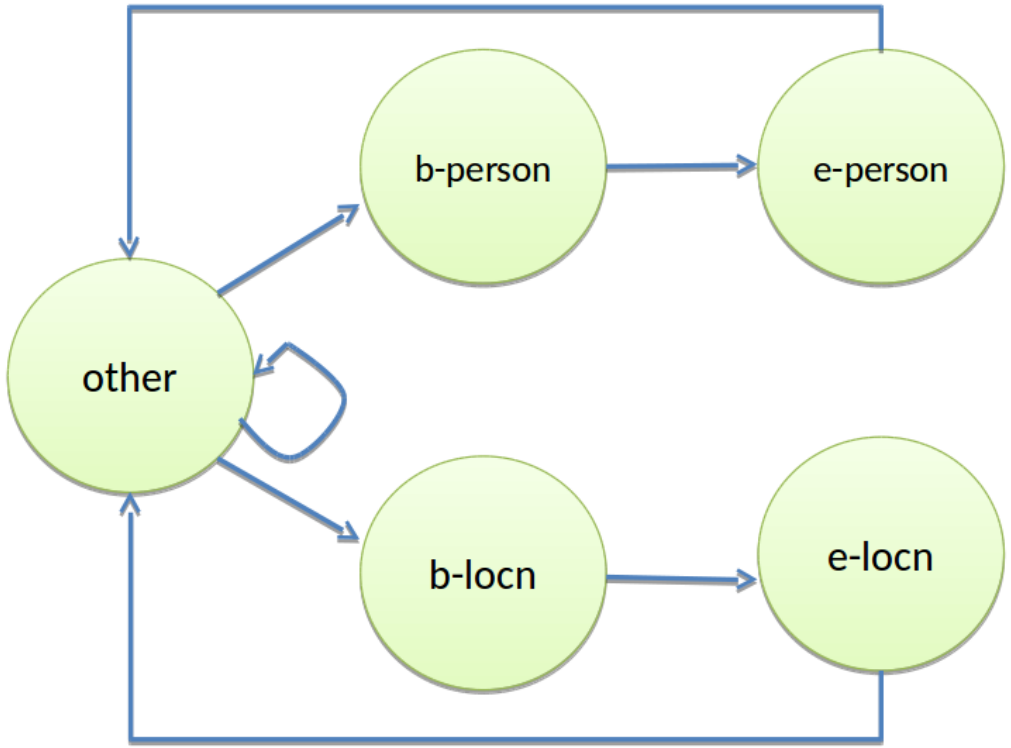
- The likelihood is $\prod_{t=1}^{T} p(y_t \mid y_{t-1}, \mathbf{x}_t)$.
- We can now use any kind of features for $\mathbf{x}_t$.

MAXIMAL ENTROPY MARKOV MODELS

- But there is a “label bias” problem:

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

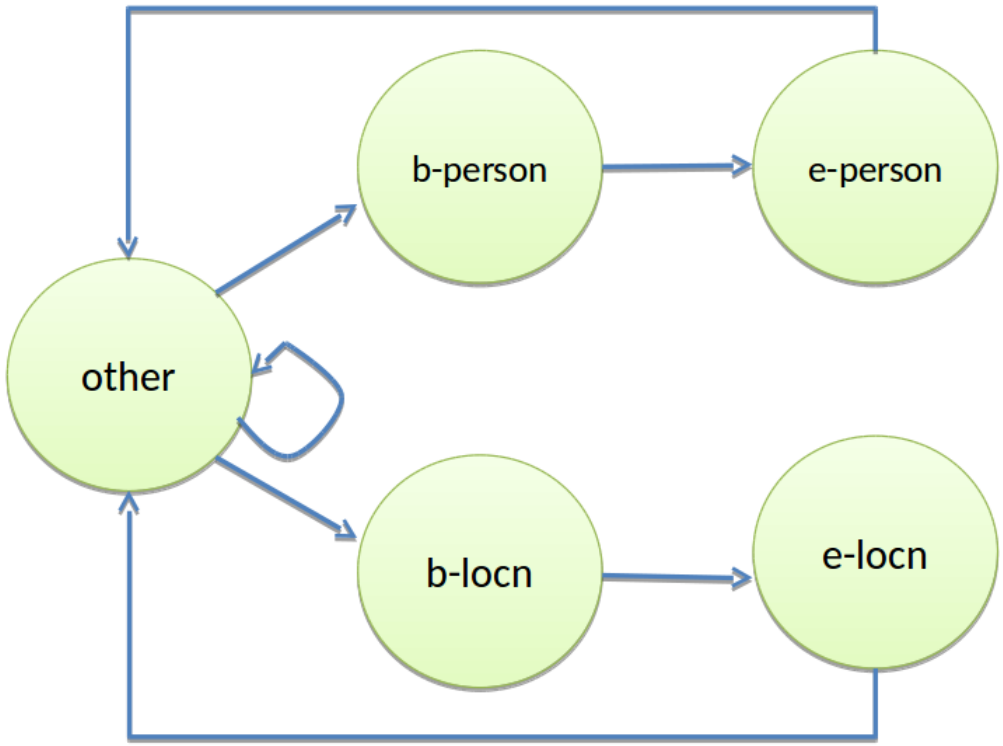


MAXIMAL ENTROPY MARKOV MODELS

- The second token has no choice now, even though it is important:

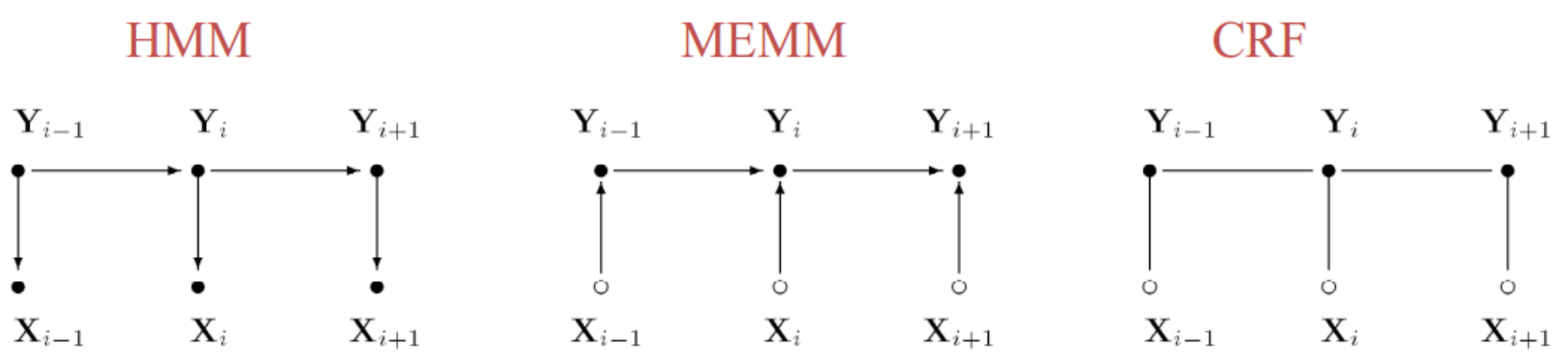
Conditional probabilities:

p(b-person | other, w = Harvey) = 0.5
p(b-locn | other, w = Harvey) = 0.5
p(b-person | other, w = Myrtle) = 0.5
p(b-locn | other, w = Myrtle) = 0.5
p(e-person | b-person, w = Ford) = 1
p(e-person | b-person, w = Park) = 1
p(e-locn | b-locn, w = Ford) = 1
p(e-locn | b-locn, w = Park) = 1



CONDITIONAL RANDOM FIELDS

- Hence, CRF:



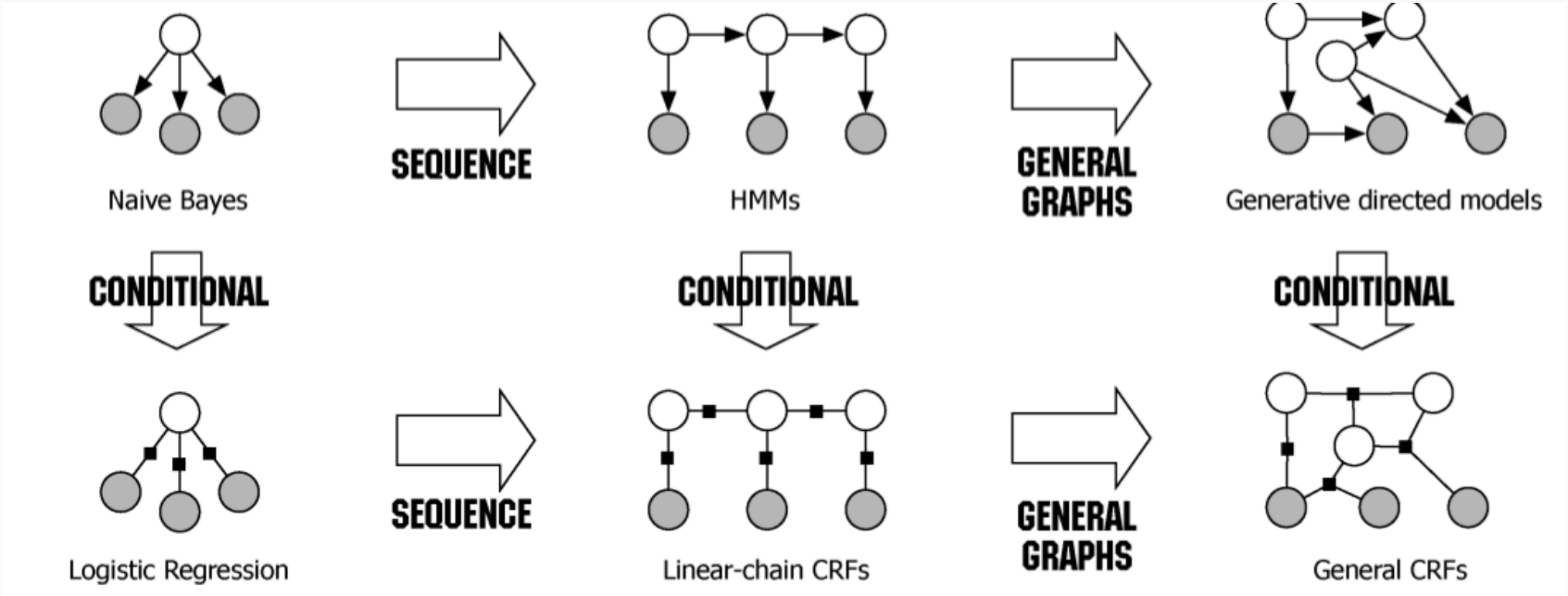
- Undirected graphical model, joint distribution

$$p(\mathbf{y}, \mathbf{x}) = \frac{1}{Z} \prod_{t=1}^{T} e^{\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}_t)}.$$

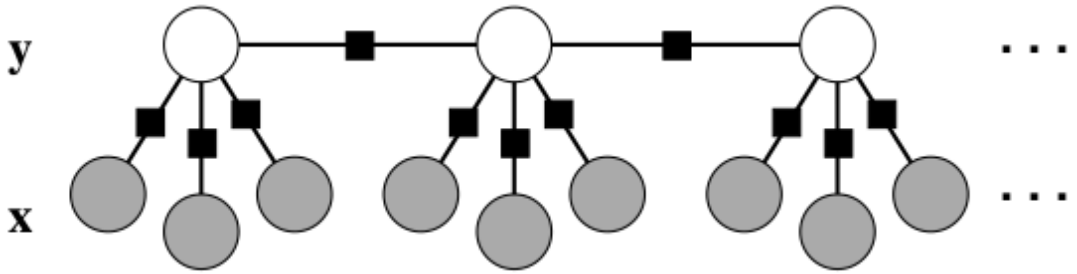
- And the inference problem is

$$p(\mathbf{y} \mid \mathbf{x}) = \frac{\prod_{t=1}^{T} e^{\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}_t)}}{\sum_{\mathbf{y}'} \prod_{t=1}^{T} e^{\sum_{k=1}^{K} \theta_k f_k(y'_t, y'_{t-1}, \mathbf{x}_t)}}.$$

- This is called linear-chain CRF. In this form:

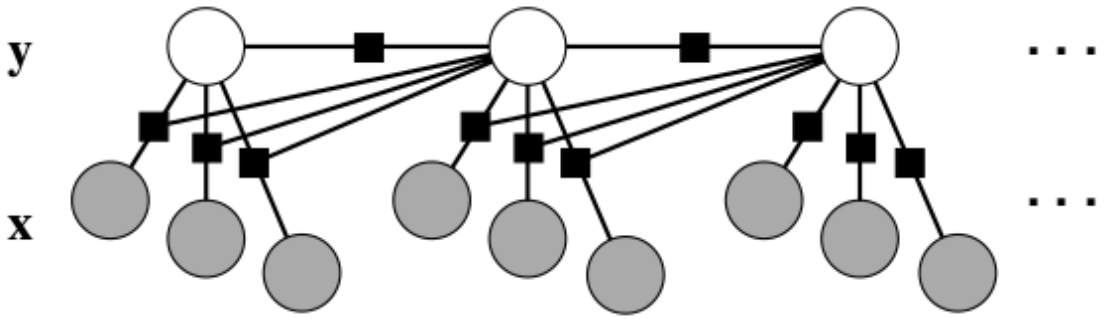


- But we can also let local features depend on more than that. In an HMM, the transition depends only on the hidden state:

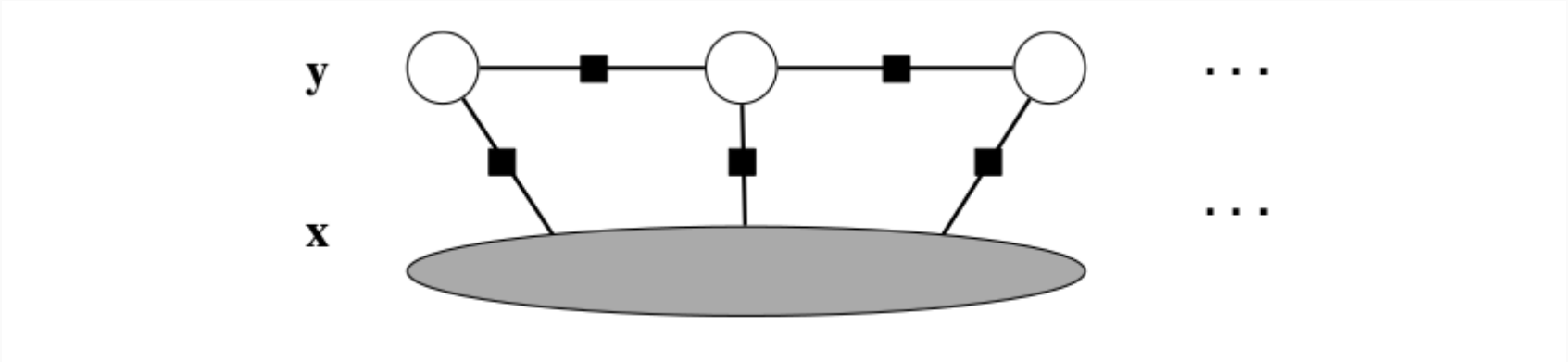


CONDITIONAL RANDOM FIELDS

- In a CRF, we can allow it to depend on the current observations:



- Or simply on all observations:

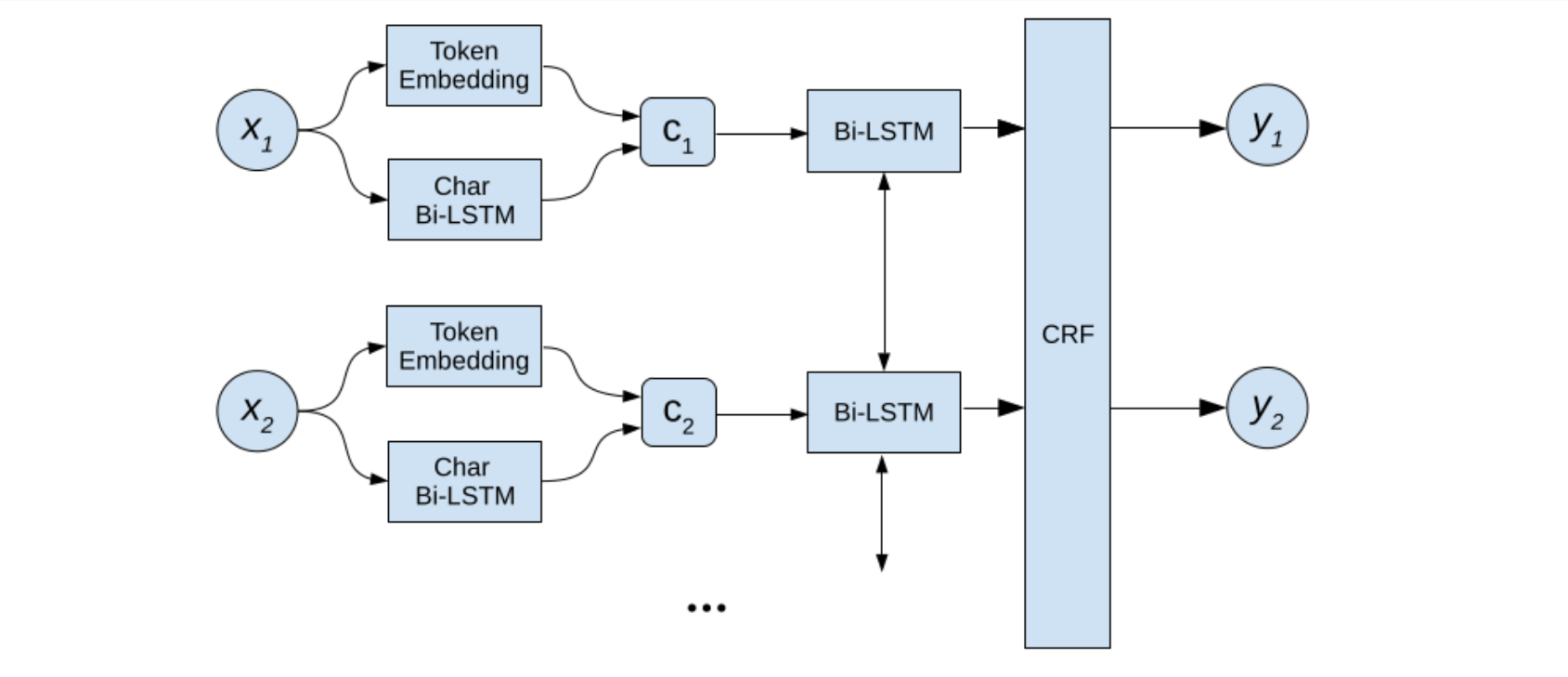


CONDITIONAL RANDOM FIELDS

- To train CRFs, we need to do approximate inference in undirected graphical models (the marginals $Z(\mathbf{x})$ are hard to compute).
- But wait, where is the deep learning?..
- You can do the same things with RNNs.
- But even better...

CONDITIONAL RANDOM FIELDS

- Put a CRF on top of features extracted by LSTMs:



- Current state of the art in NER, for example.

THANK YOU!

Thank you for your attention!