ECE 175B: Probabilistic Reasoning and Graphical Models
Lecture 15: Inference on Discrete Variable Serial Chains III

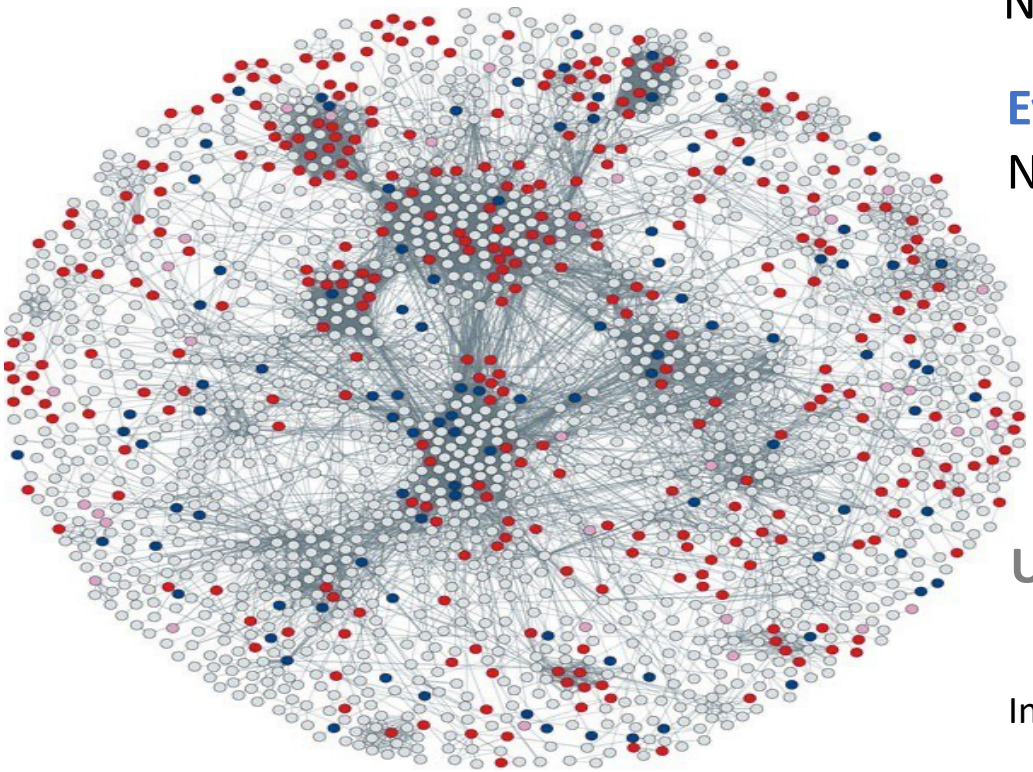
Florian Meyer & Ken Kreutz-Delgado

Electrical and Computer Engineering Department
University of California San Diego

UC San Diego
JACOBS SCHOOL OF ENGINEERING

Querying a Distribution Encoded in a Graph

$P_{\mathbf{X}}(x) = P_{\mathbf{W},\mathbf{E}}(w,e) = P_{\mathbf{V},\mathbf{Y}}(v,y) = P_{\mathbf{Y},\mathbf{E},\mathbf{Z}}(y,e,z)$



Query nodes = Y = "Outcome"

Non-Query nodes = **V** = **X** - **Y**

Evidence nodes = E

Non-Evidence nodes = **W** = **X** - **E**

$\mathbf{Y} \cap \mathbf{E} = \emptyset$

$\mathbf{Z} = \mathbf{X} - (\mathbf{Y} \cup \mathbf{E})$

Unobserved nodes = Z

In the standard manner we set $P_{\mathbf{X}}(x) = P(x)$, etc.

Two Basic Queries

- **Probability of Outcome $\mathbf{Y = y}$ Given Evidence $\mathbf{E = e}$**

  Compute the value of $\mathbf{P(y \mid e)}$

- **Most Probable Value of Outcome $\mathbf{Y = y}$ Given Evidence $\mathbf{E = e}$**

  Compute the most probable outcome

  $$\hat{\mathbf{y}}_{\text{map}} = \arg\max_{\mathbf{y}} \mathbf{P(y \mid e)}$$

How do we do this in an efficient, computationally tractable manner?

Queries and Serial-Chain Factorization (SCF)

For distributions that have a **Serial-Chain Factorization** (SCF)

$$P(\mathbf{x}) = P(\mathbf{w}, \mathbf{e}) = \frac{1}{Z} \prod_{k=0}^{N-1} \psi_{k,k+1}(w_k, w_{k+1}) \iff P(\mathbf{w} \,|\, \mathbf{e}) = \frac{1}{Z(\mathbf{e})} \prod_{k=0}^{N-1} \psi_{k,k+1}(w_k, w_{k+1})$$

we can computationally efficiently compute $P(w_k|\mathbf{e})$ via the very similar Message Passing Algorithm (MPA) and Belief Propagation (BP) "forward–backward" algorithms.

The computation of $P(w_k|\mathbf{e})$ directly answers the query, "What is the probability that W_k takes the value w_k given the evidence?", which then enables the answer to the query, "What is the most probable value of W_k given the evidence?":

$$\hat{w}_k^{(\mathrm{map})} = \arg \max_{w_k} P(w_k|\mathbf{e}).$$

The MAP estimate $\hat{w}_k^{(\mathrm{map})}$ gives the Bayes optimal, *Minimum Probability of Error* (MPE), estimate of the state ("symbol") W_k at the *single* node ("time") k given *all* of the evidence $\mathbf{e} = \{e_1, \cdots, e_k, \cdots, e_N\}$. This is a very important result in digital cellular communication where it is known as the minimum symbol error rate solution.

An even more frequently used estimate of W_k in cellular communications, because it is less computationally expensive while given performance nearly as good as the MAP estimate, is provided by the **Viterbi Algorithm** (VA) described in a latter slide.

Variable Elimination – The Sum-Product Algorithm

When $P(\mathbf{w}|\mathbf{e})$ is a SCF distribution, the marginal distribution $P(w_k|\mathbf{e})$ has the **Sum-Product** form,

$$P(w_k|\mathbf{e}) \propto \sum_{\mathbf{w}\setminus w_k} \prod_{k=0}^{N-1} \psi(w_k, w_{k+1})$$

$$= \sum_{w_1} \cdots \sum_{w_{k-1}} \sum_{w_{k+1}} \cdots \sum_{w_N} \psi(w_0, w_1) \cdots \psi(w_{k-1}, w_k) \psi(w_k, w_{k+1}) \cdots \psi(w_{N-1}, w_N)$$

$$= \underbrace{\left(\sum_{w_1} \cdots \sum_{w_{k-1}} \psi(w_0, w_1) \cdots \psi(w_{k-1}, w_k) \right)}_{\triangleq m^-(w_k) = \text{message to } w_k \text{ from the past}} \underbrace{\left(\sum_{w_{k+1}} \cdots \sum_{w_N} \psi(w_k, w_{k+1}) \cdots \psi(w_{N-1}, w_N) \right)}_{\triangleq m^+(w_k) = \text{message to } w_k \text{ from the future}}$$

which results in the simple MPA "sum-product" recursions

$$m^-(w_k) = \sum_{w_{k-1}} \psi(w_{k-1}, w_k)\, m^-(w_{k-1}) \quad \text{and} \quad m^+(w_k) = \sum_{w_{k+1}} \psi(w_k, w_{k+1})\, m^+(w_{k+1}).$$

The variables in $\mathbf{w}\setminus w_k$ are one-by-one "eliminated" (efficiently dealt with) via a sum over a product.

There are other such **Variable Elimination structures**, e.g., **Max-Product**, **Min-Product**, **Max-Sum**, and **Min-Sum**, that each lead to corresponding efficient algorithms. As we shall see, they are closely related. The Max-Product algorithm is discussed in Barber Section 5.2.1 and the Max-Sum algorithm in Bishop Section 8.4.5. Variable Elimination is discussed in Koller Chapter 9.

The Max-Product Algorithm – I

Given $P(\mathbf{w}|\mathbf{e})$ we are often interested in efficiently computing

$$\check{\mathbf{w}}^{(\text{map})} = \arg\max_{\mathbf{w}} P(\mathbf{w}|\mathbf{e}) = \arg\max_{\mathbf{w}} P(\mathbf{w},\mathbf{e})$$

Note that $\check{\mathbf{w}}^{(\text{map})}$ is an estimate of the *entire sequence of states* ("symbols") viewed as a single entity $\mathbf{w} = \{w_1, \cdots, w_N\}$. I.e., we are asking for an *optimal sequence estimate*.

In general it can happen that the optimal sequence estimate does not produce the optimal single-node k ("time k") estimate:

$$\check{w}_k^{(\text{map})} \neq \hat{w}_k^{(\text{map})}.$$

Consistent with cellular comm we can call the optimal **sequence estimate** $\check{\mathbf{w}}^{(\text{map})}$ the *Viterbi solution*.

When $P(\mathbf{w},\mathbf{e})$ is a SCF distribution, the optimization $\max_{\mathbf{w}} P(\mathbf{w},\mathbf{e})$ has the **Max-Product** form,

$$\begin{aligned}
\max_{\mathbf{w}} P(\mathbf{w},\mathbf{e}) &= \max_{w_1,\cdots,w_N} \prod_{k=0}^{N-1} \psi(w_k, w_{k+1}) \\
&= \max_{w_1} \cdots \max_{w_{k-1}} \max_{w_{k+1}} \cdots \max_{w_N} \psi(w_0,w_1)\cdots\psi(w_{k-1},w_k)\psi(w_k,w_{k+1})\cdots\psi(w_{N-1},w_N) \\
&= \max_{w_1} \underbrace{\psi(w_0,w_1)}_{\psi(w_1)} \Bigg[\cdots \underbrace{\Bigg[\max_{w_k} \psi(w_{k-1},w_k) \underbrace{\Bigg[\max_{w_{k+1}} \psi(w_k,w_{k+1}) \Bigg[\cdots \Bigg[\max_{w_N} \psi(w_{N-1},w_N) \Big[1\Big] \Bigg] \cdots \Bigg]}_{c_k(w_k)} \Bigg]}_{c_{k-1}(w_{k-1})} \cdots \Bigg] \Bigg]
\end{aligned}$$

The Max-Product Algorithm – II

The function $c(w_k)$ is the (multiplicative) optimal "cost-to-go" and is computed "backwards" recursively via

$$c(w_{k-1}) = \max_{w_k} \psi_{k-1,k}(w_{k-1}, w_k; \mathbf{e})\, c(w_k)$$

for $k = N-1, \cdots, 1$ with boundary condition $c(w_N) = 1$. This shows that a sequence of Max-Product operations "eliminates" w_k from consideration in a step-by-step manner. Note that under our assumption that w_k is categorical, **at each step** k *we can (and do) readily compute the function* $c_k(w_k)$ as a look up table of values for each value that w_k can take. Thus at the end of the recursion we will have computed a sequence of cost-to-go functions $c_1(w_1), \cdots c_k(w_K), \cdots, c_{N-1}(w_{N-1})$.

Having determined the optimal cost-to-go functions $c_k(w_k)$ we use them to sequentially produce $\check{w}_k = \check{w}_k^{(\mathrm{map})}$ via a process called *backtracking* or *backtracing*:

$$\text{Viterbi solution } \check{\mathbf{w}} \text{ has components} = \begin{cases} \check{w}_1 = \arg\max\limits_{w_1} \psi_{0,1}(w_0, w_1)\, c_1(w_1) \\ \check{w}_2 = \arg\max\limits_{w_2} \psi(\check{w}_1, w_2)\, c_2(w_2) \\ \quad \vdots \\ \check{w}_k = \arg\max\limits_{w_k} \psi_{k-1,k}(\check{w}_{k-1}, w_k)\, c_k(w_k) \\ \quad \vdots \\ \check{w}_N = \arg\max\limits_{w_N} \psi_{N-1,N}(\check{w}_{N-1}, w_N)\, c_N(w_N) \end{cases}$$

with $\psi_{0,1}(w_0, w_1) = \psi_1(w_1)$ and $c_N(w_N) = 1$.

Variable Elimination Variants of Max-Product Algorithm

There are different, but equivalent, ways to formulate the Viterbi solution optimization problem:

$$\check{\mathbf{w}}^{(\mathrm{map})} = \arg\max_{\mathbf{w}} P(\mathbf{w}, \mathbf{e}) = \arg\max_{\mathbf{w}} \ln P(\mathbf{w}, \mathbf{e}) = \arg\min_{\mathbf{w}} \Big(-\ln P(\mathbf{w}, \mathbf{e}) \Big)$$

The first variant leads to the **Max-Sum** formulation:

$$\max_{\mathbf{w}} P(\mathbf{w}, \mathbf{e}) = \max_{w_1, \cdots, w_N} \sum_{k=0}^{N-1} \ln \psi(w_k, w_{k+1})$$

The second variant leads to the **Min-Sum** formulation:

$$\max_{\mathbf{w}} P(\mathbf{w}, \mathbf{e}) = \min_{w_1, \cdots, w_N} \sum_{k=0}^{N-1} \Big(-\ln \psi(w_k, w_{k+1}) \Big)$$

These formulations are well-defined because of our running assumption that all distributions are positive which ensures that the (evidence-dependent)potentials $\psi(w_k, w_{k+1}) = \psi_{k,k+1}(w_k, w_{k+1}; \mathbf{e})$ are always strictly positive. They lead to efficient algorithms which all produce the Viterbi solutions but which can have different computational properties.

We now look in more detail at the Min-Sum formulation. The resulting *Min-Sum Algorithm* is famously known as the **Viterbi Algorithm** (VA), after its inventor Andrew Viterbi. Viterbi is a world-famous former UCSD professor who co-founded Qualcomm with another former UCSD professor Irwin Jacobs.

The Viterbi Algorithm - I

Define

$$\ell(w_k, w_{k+1}) \stackrel{\text{def}}{=} -\ln \psi(w_k, w_{k+1})$$

Then the Min-Sum formulation yields

$$\max_{\mathbf{w}} P(\mathbf{w}, \mathbf{e}) = \min_{w_1, \cdots, w_N} \sum_{k=0}^{N-1} \ell(w_k, w_{k+1})$$

$$= \min_{w_1} \cdots \min_{w_{k-1}} \min_{w_{k+1}} \cdots \min_{w_N} \Big[\ell(w_0, w_1) + \cdots + \ell(w_{k-1}, w_k) + \ell(w_k, w_{k+1}) + \cdots + \ell(w_{N-1}, w_N$$

$$= \min_{w_1} \underbrace{\ell(w_0, w_1)}_{\ell(w_1)} + \Big[\min_{w_2} \ell(w_1, w_2) + \cdots$$

$$+ \underbrace{\Big[\min_{w_k} \ell(w_{k-1}, w_k) + \underbrace{\Big[\min_{w_{k+1}} \ell(w_k, w_{k+1}) + \Big[\cdots \Big[\min_{w_N} \ell(w_{N-1}, w_N) + \Big[0 \Big] \Big] \cdots \Big] \Big]}_{O_k(w_k)} \cdots \Big]}_{O_{k-1}(w_{k-1})}$$

Here the function $O_k(w_k)$ is the (additive) optimal cost to go and is computed backwards recursively as

$$O_{k-1}(w_{k-1}) = \min_{w_k} \Big(\ell_{k-1,k}(w_{k-1}, w_k; \mathbf{e}) + O_k(w_k) \Big)$$

for $k = N-1, \cdots, 0$ with boundary condition $O(w_N) = 0$.

The Viterbi Algorithm - II

Note that the recursion for the additive cost to go $O_k(w)$, which applies in the negative-log domain, looks like the negative logarithm of the recursion for the multiplicative cost to go $c_k(w_k)$ which applies in the original domain.

As before, now that we've determined the optimal cost-to-go functions $O_k(w_k)$ we use backtracking to sequentially produce the Viterbi estimates $\check{w}_k = \check{w}_k^{(\mathrm{map})}$:

$$\text{Viterbi solution } \check{\mathbf{w}} \text{ has components} = \begin{cases} \check{w}_1 = \arg\min\limits_{w_1} \left(\ell_{0,1}(w_0, w_1) + O_1(w_1) \right) \\ \check{w}_2 = \arg\min\limits_{w_2} \left(\ell_{1,2}(\check{w}_1, w_2) + O_2(w_2) \right) \\ \quad \vdots \\ \check{w}_k = \arg\min\limits_{w_k} \left(\ell_{k-1,k}(\check{w}_{k-1}, w_k) + O_k(w_k) \right) \\ \quad \vdots \\ \check{w}_N = \arg\min\limits_{w_N} \left(\ell_{N-1,N}(\check{w}_{N-1}, w_N) + O_N(w_N) \right) \end{cases}$$

with $\ell_{0,1}(w_0, w_1) = \ell_1(w_1)$ and $O_N(w_N) = 0$.