Lecture 8

Dynamic Programming

Part 1: Directed Acyclic Graphs

Machine Learning for Structured Data
Vlad Niculae · LTL, UvA · https://vene.ro/mlsd

Computations For Structures

Recall: Structured outputs are:

- discrete objects
- made of smaller parts
- which interact with each other and/or constrain each other,

and we must know how to compute:

- $\text{score}(y)$
- for prediction: $\arg\max_{y \in \mathcal{Y}} \text{score}(y)$
- for learning: $\log \sum_{y \in \mathcal{Y}} \exp\left(\text{score}(y)\right)$

For large problems, we can't enumerate $\mathcal{Y}$ (could be exponentially large).

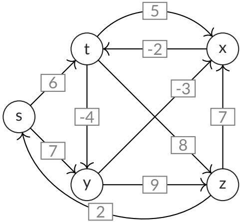
So, we must actually make use of its structure.

Recap: Graphs

Definition 1: Weighted directed graph

A weighted directed graph is $G = (V, E, w)$ where:

- V is the set of vertices (nodes) of G.
- $E \subset V \times V$ is the set of arcs of G:
  $uv \in E$ means there is an arc from node $u \in V$ to node $v \in V$ $(u \neq v)$.
  Arcs are ordered pairs, so $uv \neq vu$.
- $w : E \to \mathbb{R}$ is a weight function assigning a weight to each edge.

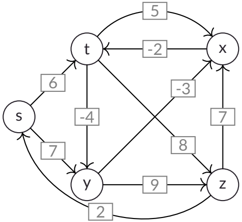


Recap: Graphs

Definition 1: Weighted directed graph

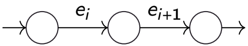
A weighted directed graph is $G = (V, E, w)$ where:

- V is the set of vertices (nodes) of G.
- $E \subset V \times V$ is the set of arcs of G:
  $uv \in E$ means there is an arc from node $u \in V$ to node $v \in V$ $(u \neq v)$.
  Arcs are ordered pairs, so $uv \neq vu$.
- $w : E \to \mathbb{R}$ is a weight function assigning a weight to each edge.



Definition 2: Paths

A path A in G is a sequence of edges: $A = e_1 e_2 \dots e_k$, with each $e_i \in E$, two-by-two "linked", i.e., if $e_i = u_i v_i$ and $e_{i+1} = u_{i+1} v_{i+1}$
then we must have $v_i = u_{i+1}$.

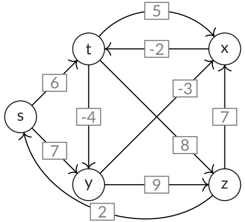


Recap: Graphs

Definition 1: Weighted directed graph

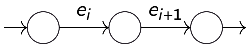
A weighted directed graph is $G = (V, E, w)$ where:

- V is the set of vertices (nodes) of G.
- $E \subset V \times V$ is the set of arcs of G:
  $uv \in E$ means there is an arc from node $u \in V$ to node $v \in V$ $(u \neq v)$.
  Arcs are ordered pairs, so $uv \neq vu$.
- $w : E \to \mathbb{R}$ is a weight function assigning a weight to each edge.



Definition 2: Paths

A path A in G is a sequence of edges: $A = e_1 e_2 \ldots e_k$, with each $e_i \in E$, two-by-two "linked", i.e., if $e_i = u_i v_i$ and $e_{i+1} = u_{i+1} v_{i+1}$
then we must have $v_i = u_{i+1}$.



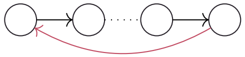
The weight of a path is the sum of arc weights: $w(A) = \sum_{e \in P} w(e)$.

We denote path concatenation by $A_1 \frown A_2$ (when legal).

Directed Acyclic Graphs

Definition 3: Cycle

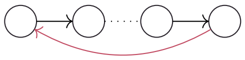
A cycle is a path $e_1 e_2 \dots e_k$ wherein the last edge e_k points to the node from which the first edge e_1 departs.



Directed Acyclic Graphs

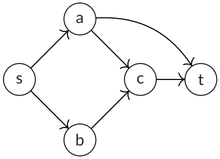
Definition 3: Cycle

A cycle is a path $e_1 e_2 \ldots e_k$ wherein the last edge e_k points to the node from which the first edge e_1 departs.



Definition 4. Directed acyclic graph (DAG)

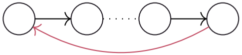
A DAG is a directed graph that contains no cycles.



Directed Acyclic Graphs

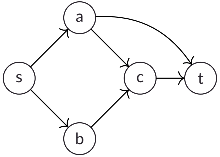
Definition 3: Cycle

A cycle is a path $e_1 e_2 \dots e_k$ wherein the last edge e_k points to the node from which the first edge e_1 departs.



Definition 4. Directed acyclic graph (DAG)

A DAG is a directed graph that contains no cycles.



Definition 4. Topological ordering

A topological ordering of a directed graph $G = (V, E)$ is an ordering of its nodes $v_1, v_2, \dots, v_n$ such that if $v_i v_j \in E$ then $i < j$.

TOs:
s, a, b, c, t
s, b, a, c, t

G is a DAG if and only if G admits a topological ordering.
Rough intuition: "backward" edges against the ordering $\Longleftrightarrow$ cycles.

Lecture 8

Dynamic Programming

Part 2: Optimal Paths: The Viterbi Algorithm

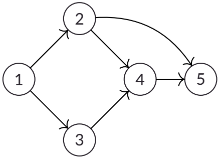
Machine Learning for Structured Data
Vlad Niculae · LTL, UvA · `https://vene.ro/mlsd`

Dynamic Programming

Paths In DAGs

Label nodes in topological order $V = \{1, \ldots, n\}$.

Let $\mathcal{Y}_i$ be the set of paths starting at 1 and ending at i.



Paths In DAGs

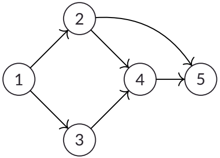
Label nodes in topological order $V = \{1, \ldots, n\}$.

Let $\mathcal{Y}_i$ be the set of paths starting at 1 and ending at i.

Let's assume our space of structures is $\mathcal{Y} = \mathcal{Y}_n$.

Important things to compute:

- $\text{score}(y) = w(y)$
- $\text{argmax}_{y \in \mathcal{Y}_n} w(y)$
- $\log \sum_{y \in \mathcal{Y}_n} \exp w(y)$

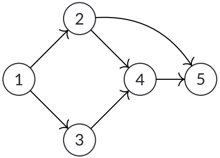


Paths In DAGs

Label nodes in topological order $V = \{1, \ldots, n\}$.

Let $\mathcal{Y}_i$ be the set of paths starting at 1 and ending at i.

Let's assume our space of structures is $\mathcal{Y} = \mathcal{Y}_n$.



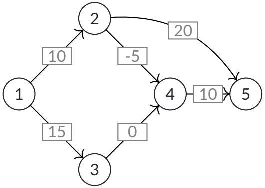
Important things to compute:

- $\text{score}(y) = w(y)$
- $\text{argmax}_{y \in \mathcal{Y}_n} w(y)$
- $\log \sum_{y \in \mathcal{Y}_n} \exp w(y)$

Later, I'll show you some structured problems that can be usefully reduced to paths in a DAG, and some that cannot.

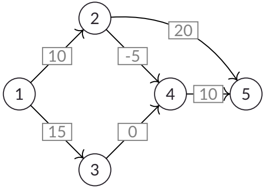
Max-Scoring Path

- The greedy path from 1 to 5 might not be best.
- From *Data Structures and Algorithms* you might recall Dijkstra's algorithm.
  - Requires no "negative cycles" — always true for DAGs.
  - Complexity: $\Theta(|V| \log |V| + |E|)$ with "Fibonacci heaps"; $\Theta(|V|^2)$ with a straightforward implementation. .

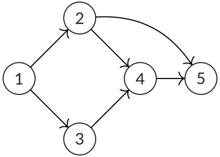


Max-Scoring Path

- The greedy path from 1 to 5 might not be best.
- From *Data Structures and Algorithms* you might recall Dijkstra's algorithm.
  - Requires no "negative cycles" — always true for DAGs.
  - Complexity: $\Theta(|V| \log |V| + |E|)$ with "Fibonacci heaps"; $\Theta(|V|^2)$ with a straightforward implementation. .
- In the case of DAGs, we can do better.



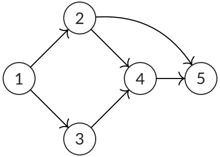
Dynamic Programming Recurrence



Goal: the max weight of a path from 1 to i:

$$m_i = \max_{y \in \mathcal{Y}_i} w(y).$$

Dynamic Programming Recurrence



Goal: the max weight of a path from 1 to i:

$$m_i = \max_{y \in \mathcal{Y}_i} w(y).$$

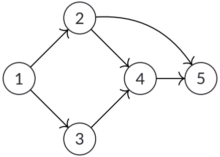
Define predecessors of i as $P_i := \{j \in V : ji \in E\}$.

Insight 1.

Any path from to i is an extension of some path to predecessor $j \in P_i$ by arc ji.

In other words: if $y \in \mathcal{Y}_i$ then $y = y' \frown ji$ for some $j \in P_i$ and some $y' \in \mathcal{Y}_j$.

Dynamic Programming Recurrence



Goal: the max weight of a path from 1 to i:

$$m_i = \max_{y \in \mathcal{Y}_i} w(y).$$

Define predecessors of i as $P_i := \{j \in V : ji \in E\}$.

Insight 1.

Any path from to i is an extension of some path to predecessor $j \in P_i$ by arc ji.

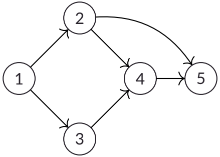
In other words: if $y \in \mathcal{Y}_i$ then $y = y' \frown ji$ for some $j \in P_i$ and some $y' \in \mathcal{Y}_j$.

Proposition: DP recurrence for max

For any $i > 1$, the best path from 1 to i is the best among the extensions of the best path to the predecessors of i:

$$m_i = \max_{j \in P_i} \left(m_j + w(ji)\right)$$

Dynamic Programming Recurrence



Goal: the max weight of a path from 1 to i:

$$m_i = \max_{y \in \mathcal{Y}_i} w(y).$$

Define predecessors of i as $P_i := \{j \in V : ji \in E\}$.

Insight 1.

Any path from to i is an extension of some path to predecessor $j \in P_i$ by arc ji.

In other words: if $y \in \mathcal{Y}_i$ then $y = y' \frown ji$ for some $j \in P_i$ and some $y' \in \mathcal{Y}_j$.

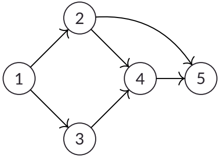
Proposition: DP recurrence for max

For any $i > 1$, the best path from 1 to i is the best among the extensions of the best path to the predecessors of i:

$$m_i = \max_{j \in P_i} \left(m_j + w(ji) \right)$$

Proof: $m_i := \max_{y \in \mathcal{Y}_i} w(y)$

Dynamic Programming Recurrence



Goal: the max weight of a path from 1 to i:

$$m_i = \max_{y \in \mathcal{Y}_i} w(y).$$

Define predecessors of i as $P_i := \{j \in V : ji \in E\}$.

Insight 1.

Any path from to i is an extension of some path to predecessor $j \in P_i$ by arc ji.

In other words: if $y \in \mathcal{Y}_i$ then $y = y' \frown ji$ for some $j \in P_i$ and some $y' \in \mathcal{Y}_j$.

Proposition: DP recurrence for max

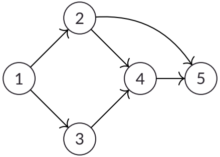
For any $i > 1$, the best path from 1 to i is the best among the extensions of the best path to the predecessors of i:

$$m_i = \max_{j \in P_i} \left(m_j + w(ji) \right)$$

Proof:

$$\begin{aligned} m_i &:= \max_{y \in \mathcal{Y}_i} w(y) \\ &= \max_{j \in P_i} \max_{y' \in \mathcal{Y}_j} \left(w(y') + w(ji) \right) \end{aligned}$$

Dynamic Programming Recurrence



Goal: the max weight of a path from 1 to i:

$$m_i = \max_{y \in \mathcal{Y}_i} w(y).$$

Define predecessors of i as $P_i := \{j \in V : ji \in E\}$.

Insight 1.

Any path from to i is an extension of some path to predecessor $j \in P_i$ by arc ji.

In other words: if $y \in \mathcal{Y}_i$ then $y = y' \frown ji$ for some $j \in P_i$ and some $y' \in \mathcal{Y}_j$.

Proposition: DP recurrence for max

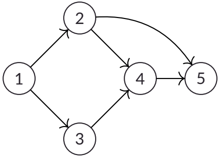
For any $i > 1$, the best path from 1 to i is the best among the extensions of the best path to the predecessors of i:

$$m_i = \max_{j \in P_i} \left(m_j + w(ji) \right)$$

Proof:

$$\begin{aligned} m_i &:= \max_{y \in \mathcal{Y}_i} w(y) \\ &= \max_{j \in P_i} \max_{y' \in \mathcal{Y}_j} \left(w(y') + w(ji) \right) \\ &= \max_{j \in P_i} \left(\max_{y' \in \mathcal{Y}_j} \left(w(y') \right) + w(ji) \right) \end{aligned}$$

Dynamic Programming Recurrence



Goal: the max weight of a path from 1 to i:

$$m_i = \max_{y \in \mathcal{Y}_i} w(y).$$

Define predecessors of i as $P_i := \{j \in V : ji \in E\}$.

Insight 1.

Any path from to i is an extension of some path to predecessor $j \in P_i$ by arc ji.

In other words: if $y \in \mathcal{Y}_i$ then $y = y' ^\frown ji$ for some $j \in P_i$ and some $y' \in \mathcal{Y}_j$.

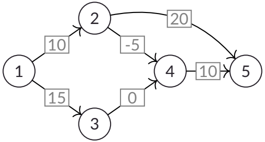
Proposition: DP recurrence for max

For any $i > 1$, the best path from 1 to i is the best among the extensions of the best path to the predecessors of i:

$$m_i = \max_{j \in P_i} \left(m_j + w(ji) \right)$$

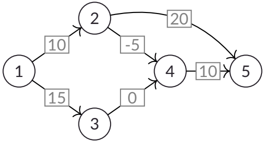
Proof:
$$\begin{aligned} m_i &:= \max_{y \in \mathcal{Y}_i} w(y) \\ &= \max_{j \in P_i} \max_{y' \in \mathcal{Y}_j} \left(w(y') + w(ji) \right) \\ &= \max_{j \in P_i} \left(\max_{y' \in \mathcal{Y}_j} (w(y')) + w(ji) \right) \\ &= \max_{j \in P_i} \left(m_j + w(ji) \right). \end{aligned}$$

The Viterbi Algorithm



$m_i = \max_{j \in P_i} \left(m_j + w(ji) \right)$ holds for any graph;
but we would chase our own tail forever.

The Viterbi Algorithm



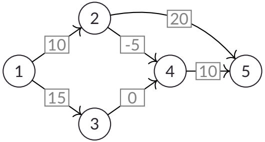
$m_i = \max_{j \in P_i} \left(m_j + w(ji)\right)$ holds for any graph;
but we would chase our own tail forever.

Insight 2.

In a topologically-ordered DAG, any path from 1 to i must only contain nodes $j < i$.

(So, we may compute $m_1, \ldots, m_n$ in order.)

The Viterbi Algorithm



$m_i = \max_{j \in P_i} \left(m_j + w(ji) \right)$ holds for any graph; but we would chase our own tail forever.

Insight 2.

In a topologically-ordered DAG, any path from 1 to i must only contain nodes $j < i$.

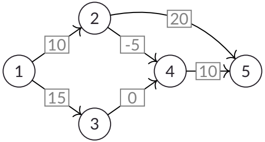
(So, we may compute $m_1, \ldots, m_n$ in order.)

General Viterbi algorithm for DAGs

input: Topologically-ordered DAG $G = (V, E, w)$, $V = \{1, \ldots, n\}$
output: maximum path weights $m_1, \ldots, m_n$.

initialize $m_1 \leftarrow 0$
for $i = 2, \ldots, n$ **do**
$\quad m_i \leftarrow \max_{j \in P_i} \left(m_j + w(ji) \right)$

The Viterbi Algorithm



$m_i = \max_{j \in P_i} \left(m_j + w(ji)\right)$ holds for any graph; but we would chase our own tail forever.

Insight 2.

In a topologically-ordered DAG, any path from 1 to i must only contain nodes $j < i$.

(So, we may compute $m_1, \ldots, m_n$ in order.)

Insight 3.

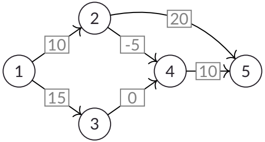
A path acheiving maximal weight is made up of the edges $j^\star i$, where $j^\star$ is the node selected by the max at each iteration.

General Viterbi algorithm for DAGs

input: Topologically-ordered DAG $G = (V, E, w)$, $V = \{1, \ldots, n\}$
output: maximum path weights $m_1, \ldots, m_n$.

initialize $m_1 \leftarrow 0$
for $i = 2, \ldots, n$ **do**
$m_i \leftarrow \max_{j \in P_i} \left(m_j + w(ji)\right)$

The Viterbi Algorithm



$m_i = \max_{j \in P_i} \left(m_j + w(ji) \right)$ holds for any graph; but we would chase our own tail forever.

Insight 2.

In a topologically-ordered DAG, any path from 1 to i must only contain nodes $j < i$.

(So, we may compute $m_1, \ldots, m_n$ in order.)

Insight 3.

A path acheiving maximal weight is made up of the edges $j^\star i$, where $j^\star$ is the node selected by the max at each iteration.

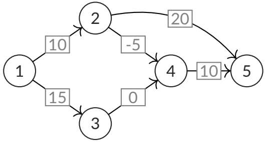
General Viterbi algorithm for DAGs

input: Topologically-ordered DAG
$G = (V, E, w)$, $V = \{1, \ldots, n\}$
output: maximum path weights $m_1, \ldots, m_n$.

initialize $m_1 \leftarrow 0$
for $i = 2, \ldots, n$ **do**
$m_i \leftarrow \max_{j \in P_i} \left(m_j + w(ji) \right)$
$\pi_i \leftarrow \arg\max_{j \in P_i} \left(m_j + w(ji) \right)$

Reconstruct path: follow backpointers
output: optimal path y from 1 to n (optional)
$y = []; i \leftarrow n$
while $i > 1$ **do**
$y \leftarrow \pi_i i ^\frown y$
$i \leftarrow \pi_i$

The Viterbi Algorithm



$m_i = \max_{j \in P_i} \left(m_j + w(ji)\right)$ holds for any graph; but we would chase our own tail forever.

Insight 2.

In a topologically-ordered DAG, any path from 1 to i must only contain nodes $j < i$.

(So, we may compute $m_1, \ldots, m_n$ in order.)

Insight 3.

A path acheiving maximal weight is made up of the edges $j^\star i$, where $j^\star$ is the node selected by the max at each iteration.

General Viterbi algorithm for DAGs

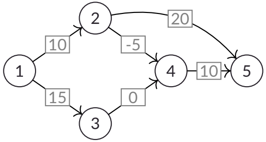
input: Topologically-ordered DAG
$G = (V, E, w)$, $V = \{1, \ldots, n\}$
output: maximum path weights $m_1, \ldots, m_n$.

initialize $m_1 \leftarrow 0$
for $i = 2, \ldots, n$ **do**
  $m_i \leftarrow \max_{j \in P_i} \left(m_j + w(ji)\right)$
  $\pi_i \leftarrow \arg\max_{j \in P_i} \left(m_j + w(ji)\right)$

Reconstruct path: follow backpointers
output: optimal path y from 1 to n (optional)
$y = []; i \leftarrow n$
while $i > 1$ **do**
  $y \leftarrow \pi_i i ^\frown y$
  $i \leftarrow \pi_i$

Complexity: $\Theta(|V| + |E|)$.

The Viterbi Algorithm



General Viterbi algorithm for DAGs

input: Topologically-ordered DAG
$G = (V, E, w), V = \{1, \ldots, n\}$
output: maximum path weights $m_1, \ldots, m_n$.

initialize $m_1 \leftarrow 0$
for $i = 2, \ldots, n$ **do**
$\quad m_i \leftarrow \max_{j \in P_i} \left(m_j + w(ji)\right)$
$\quad \pi_i \leftarrow \arg\max_{j \in P_i} \left(m_j + w(ji)\right)$

Reconstruct path: follow backpointers
output: optimal path y from 1 to n (optional)
$y = []; i \leftarrow n$
while $i > 1$ **do**
$\quad y \leftarrow \pi_i i ^\frown y$
$\quad i \leftarrow \pi_i$

Complexity: $\Theta(|V| + |E|)$.

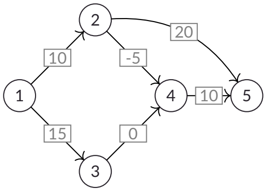
Lecture 8

Dynamic Programming

Part 3: Probabilities Over Paths: The Forward Algorithm

Machine Learning for Structured Data
Vlad Niculae · LTL, UvA · `https://vene.ro/mlsd`

Probability Distributions



A weighted DAG induces a probability distributions over all paths from 1 to n:

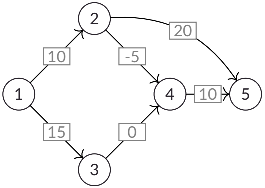
$$\Pr(y) = \frac{\exp(w(y))}{\sum_{y' \in \mathcal{Y}_n} \exp(w(y'))}$$

y	$w(y)$	$\exp(w(y))$	$\Pr(y)$
$1 \to 2 \to 5$			
$1 \to 2 \to 4 \to 5$			
$1 \to 3 \to 4 \to 5$			

To assess $\Pr(y)$ even for a single path, the denominator sums over all paths.

Next goal: calculate this denominator efficiently.

Probability Distributions



A weighted DAG induces a probability distributions over all paths from 1 to n:

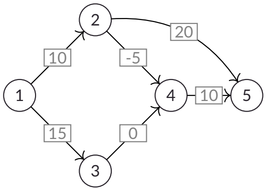
$$\Pr(y) = \frac{\exp(w(y))}{\sum_{y' \in \mathcal{Y}_n} \exp(w(y'))}$$

y	$w(y)$	$\exp(w(y))$	$\Pr(y)$
$1 \to 2 \to 5$	$10 + 20 = 30$		
$1 \to 2 \to 4 \to 5$	$10 - 5 + 10 = 15$		
$1 \to 3 \to 4 \to 5$	$15 + 0 + 10 = 25$		

To assess $\Pr(y)$ even for a single path, the denominator sums over all paths.

Next goal: calculate this denominator efficiently.

Probability Distributions



A weighted DAG induces a probability distributions over all paths from 1 to n:

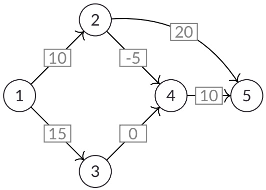
$$\Pr(y) = \frac{\exp(w(y))}{\sum_{y' \in \mathcal{Y}_n} \exp(w(y'))}$$

y	$w(y)$	$\exp(w(y))$	$\Pr(y)$
$1 \to 2 \to 5$	$10 + 20 = 30$	$1.1 \cdot 10^{13}$	
$1 \to 2 \to 4 \to 5$	$10 - 5 + 10 = 15$	$3.3 \cdot 10^{6}$	
$1 \to 3 \to 4 \to 5$	$15 + 0 + 10 = 25$	$7.2 \cdot 10^{10}$	

To assess $\Pr(y)$ even for a single path, the denominator sums over all paths.

Next goal: calculate this denominator efficiently.

Probability Distributions



A weighted DAG induces a probability distributions over all paths from 1 to n:

$$\Pr(y) = \frac{\exp(w(y))}{\sum_{y' \in \mathcal{Y}_n} \exp(w(y'))}$$

y	$w(y)$	$\exp(w(y))$	$\Pr(y)$
$1 \to 2 \to 5$	$10 + 20 = 30$	$1.1 \cdot 10^{13}$	.9930
$1 \to 2 \to 4 \to 5$	$10 - 5 + 10 = 15$	$3.3 \cdot 10^{6}$	.0001
$1 \to 3 \to 4 \to 5$	$15 + 0 + 10 = 25$	$7.2 \cdot 10^{10}$	.0069

To assess $\Pr(y)$ even for a single path, the denominator sums over all paths.

Next goal: calculate this denominator efficiently.

Log-Probability DP Recurrence

Since $\exp w(y)$ can be huge, it's better to work with log-probabilities:

$$\log \Pr(y) = w(y) - \log \sum_{y' \in \mathcal{Y}_n} \exp w(y')$$

so we aim to compute this log-sum-exp directly.

Log-Probability DP Recurrence

Since $\exp w(y)$ can be huge, it's better to work with log-probabilities:

$$\log \Pr(y) = w(y) - \log \sum_{y' \in \mathcal{Y}_n} \exp w(y')$$

so we aim to compute this log-sum-exp directly.

Insight 1 (from before).

If $y \in \mathcal{Y}_i$ then $y = y' \frown ji$ for some $j \in P_i$ and some $y' \in \mathcal{Y}_j$.

Log-Probability DP Recurrence

Since $\exp w(y)$ can be huge, it's better to work with log-probabilities:

$$\log \Pr(y) = w(y) - \log \sum_{y' \in \mathcal{Y}_n} \exp w(y')$$

so we aim to compute this log-sum-exp directly.

Insight 1 (from before).

If $y \in \mathcal{Y}_i$ then $y = y' \frown ji$ for some $j \in P_i$ and some $y' \in \mathcal{Y}_j$.

Insight 4: addition distributes over log-sum-exp.

$$c + \log \sum_i \exp(z_i) = \log \sum_i \exp(c + z_i)$$

Log-Probability DP Recurrence

Since $\exp w(y)$ can be huge, it's better to work with log-probabilities:

$$\log \Pr(y) = w(y) - \log \sum_{y' \in \mathcal{Y}_n} \exp w(y')$$

so we aim to compute this log-sum-exp directly.

Insight 1 (from before).

If $y \in \mathcal{Y}_i$ then $y = y' \frown ji$ for some $j \in P_i$ and some $y' \in \mathcal{Y}_j$.

Insight 4: addition distributes over log-sum-exp.

$$c + \log \sum_i \exp(z_i) = \log \sum_i \exp(c + z_i)$$

Denote $q_i := \log \sum_{y \in \mathcal{Y}_i} \exp(w(y))$.

Proposition: DP recurrence for log-sum-exp.

$$q_i = \log \sum_{j \in P_i} \exp\left(q_j + w(ji)\right)$$

Compare with the DP recurrence for max:

$$m_i = \max_{j \in P_i}(m_j + w(ji)).$$

Log-Probability DP Recurrence

Since $\exp w(y)$ can be huge, it's better to work with log-probabilities:

$$\log \Pr(y) = w(y) - \log \sum_{y' \in \mathcal{Y}_n} \exp w(y')$$

so we aim to compute this log-sum-exp directly.

Insight 1 (from before).

If $y \in \mathcal{Y}_i$ then $y = y' \frown ji$ for some $j \in P_i$ and some $y' \in \mathcal{Y}_j$.

Insight 4: addition distributes over log-sum-exp.

$$c + \log \sum_i \exp(z_i) = \log \sum_i \exp(c + z_i)$$

Denote $q_i := \log \sum_{y \in \mathcal{Y}_i} \exp(w(y))$.

Proposition: DP recurrence for log-sum-exp.

$$q_i = \log \sum_{j \in P_i} \exp\left(q_j + w(ji)\right)$$

Compare with the DP recurrence for max:

$$m_i = \max_{j \in P_i} (m_j + w(ji)).$$

Proof: $q_i = \log \sum_{j \in P_i} \sum_{y' \in \mathcal{Y}_j} \exp\left(w(y') + w(ji)\right)$

Log-Probability DP Recurrence

Since $\exp w(y)$ can be huge, it's better to work with log-probabilities:

$$\log \Pr(y) = w(y) - \log \sum_{y' \in \mathcal{Y}_n} \exp w(y')$$

so we aim to compute this log-sum-exp directly.

Insight 1 (from before).

If $y \in \mathcal{Y}_i$ then $y = y' \frown ji$ for some $j \in P_i$ and some $y' \in \mathcal{Y}_j$.

Insight 4: addition distributes over log-sum-exp.

$$c + \log \sum_i \exp(z_i) = \log \sum_i \exp(c + z_i)$$

Denote $q_i := \log \sum_{y \in \mathcal{Y}_i} \exp(w(y))$.

Proposition: DP recurrence for log-sum-exp.

$$q_i = \log \sum_{j \in P_i} \exp\left(q_j + w(ji)\right)$$

Compare with the DP recurrence for max:

$$m_i = \max_{j \in P_i}(m_j + w(ji)).$$

Proof: $$q_i = \log \sum_{j \in P_i} \sum_{y' \in \mathcal{Y}_j} \exp\left(w(y') + w(ji)\right)$$

$$= \log \sum_{j \in P_i} \exp\left(\log \sum_{y' \in \mathcal{Y}_j} \exp(w(y')) + w(ji)\right)$$

Log-Probability DP Recurrence

Since $\exp w(y)$ can be huge, it's better to work with log-probabilities:

$$\log \Pr(y) = w(y) - \log \sum_{y' \in \mathcal{Y}_n} \exp w(y')$$

so we aim to compute this log-sum-exp directly.

Insight 1 (from before).

If $y \in \mathcal{Y}_i$ then $y = y' \frown ji$ for some $j \in P_i$ and some $y' \in \mathcal{Y}_j$.

Insight 4: addition distributes over log-sum-exp.

$$c + \log \sum_i \exp(z_i) = \log \sum_i \exp(c + z_i)$$

Denote $q_i := \log \sum_{y \in \mathcal{Y}_i} \exp(w(y))$.

Proposition: DP recurrence for log-sum-exp.

$$q_i = \log \sum_{j \in P_i} \exp\left(q_j + w(ji)\right)$$

Compare with the DP recurrence for max:

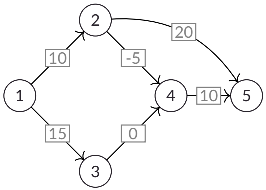
$$m_i = \max_{j \in P_i} (m_j + w(ji)).$$

Proof: $$q_i = \log \sum_{j \in P_i} \sum_{y' \in \mathcal{Y}_j} \exp\left(w(y') + w(ji)\right)$$

$$= \log \sum_{j \in P_i} \exp\left(\log \sum_{y' \in \mathcal{Y}_j} \exp(w(y')) + w(ji)\right)$$

$$= \log \sum_{j \in P_i} \exp\left(q_j + w(ji)\right).$$

The Forward Algorithm



General forward algorithm for DAGs

input: Topologically-ordered DAG $G = (V, E, w)$, $V = \{1, \dots, n\}$
output: $q_n := \log \sum_{y \in \mathcal{Y}_n} \exp w(y)$.

initialize $q_1 \leftarrow 0$
for $i = 2, \dots, n$ **do**
$$q_i \leftarrow \log \sum_{j \in P_i} \exp \left(q_j + w(ji)\right)$$

Complexity: $\Theta(|V| + |E|)$.

Lets us calculate the log-probability of any given sequence $\log \Pr(y)$.

Can use autodiff to get $\nabla_w \log \Pr(y)$.

Spot A Pattern?

Why are these two algorithms so similar?

Deriving the DP recurrences was almost identical.

Spot A Pattern?

Why are these two algorithms so similar?

Deriving the DP recurrences was almost identical.

The pattern:

- $x \oplus y = \max(x, y); \quad x \otimes y = x + y$ form a semiring over $\mathbb{R} \cup \{-\infty\}$.
- $x \oplus y = \log(e^x + e^y); x \otimes y = x + y$ form a semiring over $\mathbb{R} \cup \{-\infty\}$.

Spot A Pattern?

Why are these two algorithms so similar?

Deriving the DP recurrences was almost identical.

The pattern:

- $x \oplus y = \max(x, y); \quad x \otimes y = x + y$ form a semiring over $\mathbb{R} \cup \{-\infty\}$.
- $x \oplus y = \log(e^x + e^y); x \otimes y = x + y$ form a semiring over $\mathbb{R} \cup \{-\infty\}$.

This is a very productive generalization that leads to other algorithms too:

- the boolean semiring $x \oplus y = x \vee y, x \otimes y = x \wedge y$ over $\{0, 1\}$ yields an algorithm for path existence;
- there is a semiring that leads to top-k paths.

Lecture 8

Dynamic Programming

Part 4: Sampling Paths

Machine Learning for Structured Data
Vlad Niculae · LTL, UvA · `https://vene.ro/mlsd`

Dynamic Programming

Sampling Paths

Bonus goal: draw samples from the distribution over paths: $y_1, \ldots, y_k \sim \Pr(Y = y)$.

Motivation:

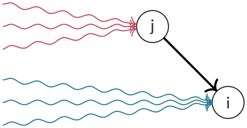
- analyze not just the most likely path, but a set of "typical" paths
- perform inferences

  $$\mathbb{E}_{\Pr(Y)}[F(Y)]$$

  for arbitrary functions F,
- train structured latent variable models

Sampling: One Arc At A Time

Probability that the last arc of a path ending in i is ji:

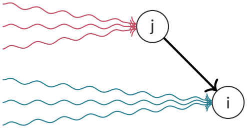


$\Pr(ji|y \text{ ends in } i) =$

Sampling: One Arc At A Time

Probability that the last arc of a path ending in i is ji:

$$\Pr(ji|y \text{ ends in } i) = \frac{\sum_{[y';ji]\in\mathcal{Y}_i} \exp(w(y') + w(ji))}{\sum_{y\in\mathcal{Y}_i} \exp(w(y))}$$

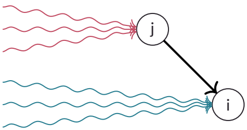


Sampling: One Arc At A Time

Probability that the last arc
of a path ending in i is ji:

$$\Pr(ji|y \text{ ends in } i) = \frac{\sum_{[y';ji]\in\mathcal{Y}_i} \exp(w(y') + w(ji))}{\sum_{y\in\mathcal{Y}_i} \exp(w(y))}$$

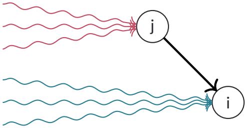
$$= \frac{\exp(w(ji))\sum_{y'\in\mathcal{Y}_j} \exp(w(y'))}{\sum_{y\in\mathcal{Y}_i} \exp(w(y))}$$



Sampling: One Arc At A Time

Probability that the last arc of a path ending in i is ji:

$$
\begin{aligned}
\Pr(ji \mid y \text{ ends in } i) &= \frac{\sum_{[y';ji] \in \mathcal{Y}_i} \exp(w(y') + w(ji))}{\sum_{y \in \mathcal{Y}_i} \exp(w(y))} \\
&= \frac{\exp(w(ji)) \sum_{y' \in \mathcal{Y}_j} \exp(w(y'))}{\sum_{y \in \mathcal{Y}_i} \exp(w(y))} \\
&= \exp(w(ji) + q_j - q_i)
\end{aligned}
$$



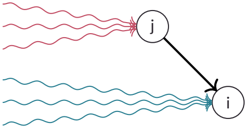
Sampling: One Arc At A Time

Probability that the last arc
of a path ending in i is ji:

$$\Pr(ji \mid y \text{ ends in } i) = \frac{\sum_{[y';ji] \in \mathcal{Y}_i} \exp(w(y') + w(ji))}{\sum_{y \in \mathcal{Y}_i} \exp(w(y))}$$

$$= \frac{\exp(w(ji)) \sum_{y' \in \mathcal{Y}_j} \exp(w(y'))}{\sum_{y \in \mathcal{Y}_i} \exp(w(y))}$$

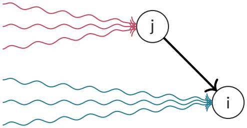
$$= \exp(w(ji) + q_j - q_i)$$



All paths end in n, so draw the final arc jn first.

Sampling: One Arc At A Time

Probability that the last arc of a path ending in i is ji:



$$
\begin{aligned}
\Pr(ji \mid y \text{ ends in } i) &= \frac{\sum_{[y';ji] \in \mathcal{Y}_i} \exp(w(y') + w(ji))}{\sum_{y \in \mathcal{Y}_i} \exp(w(y))} \\
&= \frac{\exp(w(ji)) \sum_{y' \in \mathcal{Y}_j} \exp(w(y'))}{\sum_{y \in \mathcal{Y}_i} \exp(w(y))} \\
&= \exp(w(ji) + q_j - q_i)
\end{aligned}
$$

All paths end in n, so draw the final arc jn first.

Repeat same reasoning on the subgraph with nodes $1, \ldots, j$, i.e., replace n with j and repeat until we hit 1.

Resembles the backpointers from Viterbi: think “stochastic backpointers”.

Sampling: One Arc At A Time

Probability that the last arc of a path ending in i is ji:

$$\Pr(ji | y \text{ ends in } i) = \frac{\sum_{[y';ji] \in \mathcal{Y}_i} \exp(w(y') + w(ji))}{\sum_{y \in \mathcal{Y}_i} \exp(w(y))}$$

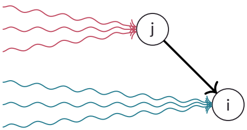
$$= \frac{\exp(w(ji)) \sum_{y' \in \mathcal{Y}_j} \exp(w(y'))}{\sum_{y \in \mathcal{Y}_i} \exp(w(y))}$$

$$= \exp(w(ji) + q_j - q_i)$$

All paths end in n, so draw the final arc jn first.

Repeat same reasoning on the subgraph with nodes $1, \ldots, j$, i.e., replace n with j and repeat until we hit 1.

Resembles the backpointers from Viterbi: think "stochastic backpointers".



Forward filtering, backward sampling for DAGs

input: Topologically-ordered DAG;
output: y: a sample from $\Pr(y)$.

initialize $q_1 \leftarrow 0$
for $i = 2, \ldots, n$ **do**
  $q_i \leftarrow \log \sum_{j \in P_i} \exp\left(q_j + w(ji)\right)$

$y = []; i \leftarrow n$
while $i > 1$ **do**
  sample $j \in P_i$ w.p. $p_j = \exp(w(ji) + q_j - q_i)$
  $y \leftarrow ji \frown y$
  $i \leftarrow j$

Conclusions

If we can cast our problem as finding paths in a DAG, then dynamic programming (DP) lets us calculate:

- $\text{argmax}_{y \in \mathcal{Y}} \text{score}(y)$
- $\log \sum_{y \in \mathcal{Y}} \exp \text{score}(y)$ and therefore probabilities
- samples from the distribution over structures

in linear time $\Theta(|V| + |E|)$.

Next we see a bunch of structures that fit this pattern, and some that do not.

Some structures solvable by DP cannot be represented via DAGs.