Lecture 13

Sampling and Intractable Models

Part 1: Sampling From Structured Probabilistic Models

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

Sampling and Intractable Models

How can we use a structured model?

We've been working with the probabilistic model over structures:

$$\Pr(\boldsymbol{y} \mid x) = \frac{\exp\left(\text{score}(\boldsymbol{y})\right)}{Z}.$$

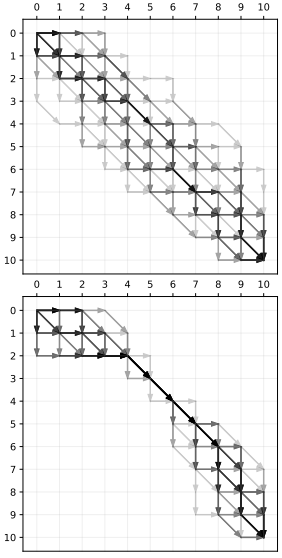
We've studied

- finding the best structure
- computing $\log Z$ (for learning and assessing probabilities)

Today's focus: sampling from $\Pr(\boldsymbol{y} \mid x)$.

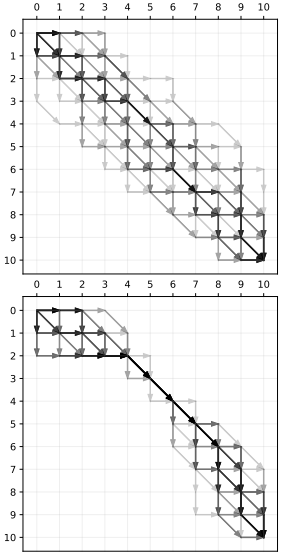
Why sample from a structured model?

- Explore ambiguity / uncertainty
  - is one configuration way better than the rest? or are there several good configurations that differ by a little?



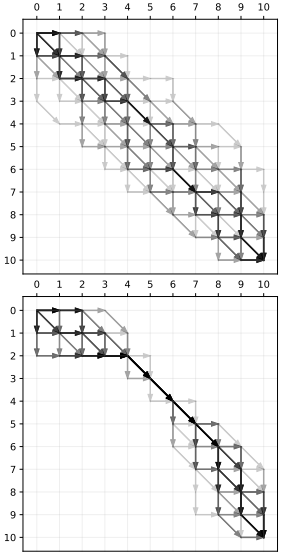
Why sample from a structured model?

- Explore ambiguity / uncertainty
  - is one configuration way better than the rest? or are there several good configurations that differ by a little?
- Increase output diversity, can be more robust to errors.



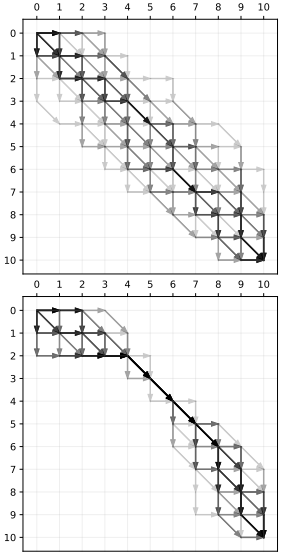
Why sample from a structured model?

- Explore ambiguity / uncertainty
  - is one configuration way better than the rest? or are there several good configurations that differ by a little?
- Increase output diversity, can be more robust to errors.
  - maybe our model's top-1 is wrong, but the right structure is among the top-3.



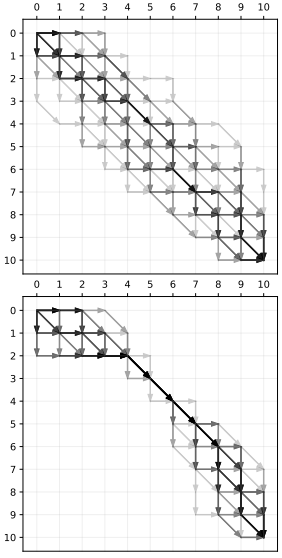
Why sample from a structured model?

- Explore ambiguity / uncertainty
  - is one configuration way better than the rest? or are there several good configurations that differ by a little?
- Increase output diversity, can be more robust to errors.
  - maybe our model's top-1 is wrong, but the right structure is among the top-3.
- Do advanced statistical analysis on the model
  - e.g., estimate entropy $\mathbb{E}_Y[-\log \Pr(Y|x)]$



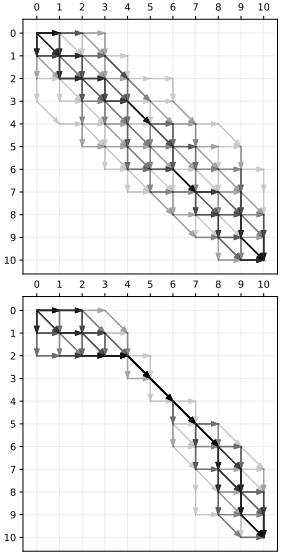
Why sample from a structured model?

- Explore ambiguity / uncertainty
  - is one configuration way better than the rest? or are there several good configurations that differ by a little?
- Increase output diversity, can be more robust to errors.
  - maybe our model's top-1 is wrong, but the right structure is among the top-3.
- Do advanced statistical analysis on the model
  - e.g., estimate entropy $\mathbb{E}_Y[-\log \Pr(Y|x)]$
- Enable learning in intractable cases (more today).



Why sample from a structured model?

- Explore ambiguity / uncertainty
  - is one configuration way better than the rest? or are there several good configurations that differ by a little?
- Increase output diversity, can be more robust to errors.
  - maybe our model's top-1 is wrong, but the right structure is among the top-3.
- Do advanced statistical analysis on the model
  - e.g., estimate entropy $\mathbb{E}_Y[-\log \Pr(Y|x)]$
- Enable learning in intractable cases (more today).
- Model structured latent variables and other advanced topics.



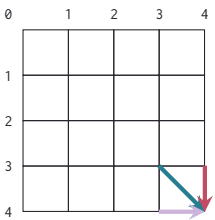
Dynamic Programming and Sampling

For structures that are tractable via DP, we also have an efficient sampling algorithm.

FFBS algorithm, discussed in the *Dynamic Programming* lecture.

For argmax, we backtrack through the table following the backpointer (the best among the possible arrows into that cell.)

To sample, follow random backpointers.



Lecture 13

Sampling and Intractable Models

Part 2: Learning Via Sampling

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

Sampling and Intractable Models

Learning Via Sampling

There are some useful models for which we cannot compute argmax or logsumexp, but we can approximately draw samples of $P(\boldsymbol{y}|\boldsymbol{x})$.

We can still (noisily) train such models using Monte Carlo gradient estimation.

Recall our probabilistic objective:

$$L_{\text{NLL}}(\boldsymbol{y}) = -\operatorname{score}(\boldsymbol{y}) + \log \sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp \operatorname{score}(\boldsymbol{y}').$$

Learning Via Sampling

There are some useful models for which we cannot compute argmax or logsumexp, but we can approximately draw samples of $P(\boldsymbol{y}|\boldsymbol{x})$.

We can still (noisily) train such models using Monte Carlo gradient estimation.

Recall our probabilistic objective:

$$L_{\text{NLL}}(\boldsymbol{y}) = -\operatorname{score}(\boldsymbol{y}) + \log \sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp \operatorname{score}(\boldsymbol{y}').$$

Its gradient:

$$\nabla_\theta L_{\text{NLL}}(\boldsymbol{y}) = -\nabla_\theta \operatorname{score}(\boldsymbol{y}) + \underbrace{\nabla_\theta \log \sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp \operatorname{score}(\boldsymbol{y}')}_{G_{\text{NLL}}}.$$

We normally leave it to pytorch. Let's look closer.

Learning Via Sampling

There are some useful models for which we cannot compute argmax or logsumexp, but we can approximately draw samples of $P(\boldsymbol{y}|\boldsymbol{x})$.

We can still (noisily) train such models using Monte Carlo gradient estimation.

Recall our probabilistic objective:

$$L_{\text{NLL}}(\boldsymbol{y}) = -\operatorname{score}(\boldsymbol{y}) + \log \sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp \operatorname{score}(\boldsymbol{y}').$$

Its gradient:

$$\nabla_\theta L_{\text{NLL}}(\boldsymbol{y}) = -\nabla_\theta \operatorname{score}(\boldsymbol{y}) + \underbrace{\nabla_\theta \log \sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp \operatorname{score}(\boldsymbol{y}')}_{G_{\text{NLL}}}.$$

$$G_{\text{NLL}} = \nabla_\theta \log \underbrace{\sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp\left(\operatorname{score}(\boldsymbol{y}')\right)}_{Z}$$

We normally leave it to pytorch. Let's look closer.

Learning Via Sampling

There are some useful models for which we cannot compute argmax or logsumexp, but we can approximately draw samples of $P(\boldsymbol{y}|\boldsymbol{x})$.

We can still (noisily) train such models using Monte Carlo gradient estimation.

Recall our probabilistic objective:

$$L_{\text{NLL}}(\boldsymbol{y}) = -\operatorname{score}(\boldsymbol{y}) + \log \sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp \operatorname{score}(\boldsymbol{y}').$$

Its gradient:

$$\nabla_\theta L_{\text{NLL}}(\boldsymbol{y}) = -\nabla_\theta \operatorname{score}(\boldsymbol{y}) + \underbrace{\nabla_\theta \log \sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp \operatorname{score}(\boldsymbol{y}')}_{G_{\text{NLL}}}.$$

$$\begin{aligned} G_{\text{NLL}} &= \nabla_\theta \log \underbrace{\sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp\left(\operatorname{score}(\boldsymbol{y}')\right)}_{Z} \\ &= \frac{1}{Z} \sum_{\boldsymbol{y}' \in \mathcal{Y}} \nabla_\theta \exp\left(\operatorname{score}(\boldsymbol{y}')\right) \end{aligned}$$

We normally leave it to pytorch. Let's look closer.

Learning Via Sampling

There are some useful models for which we cannot compute argmax or logsumexp, but we can approximately draw samples of $P(\boldsymbol{y}|\boldsymbol{x})$.

We can still (noisily) train such models using Monte Carlo gradient estimation.

Recall our probabilistic objective:

$$L_{\text{NLL}}(\boldsymbol{y}) = -\operatorname{score}(\boldsymbol{y}) + \log \sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp \operatorname{score}(\boldsymbol{y}').$$

Its gradient:

$$\nabla_\theta L_{\text{NLL}}(\boldsymbol{y}) = -\nabla_\theta \operatorname{score}(\boldsymbol{y}) + \underbrace{\nabla_\theta \log \sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp \operatorname{score}(\boldsymbol{y}')}_{G_{\text{NLL}}}.$$

$$\begin{aligned} G_{\text{NLL}} &= \nabla_\theta \log \underbrace{\sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp\left(\operatorname{score}(\boldsymbol{y}')\right)}_{Z} \\ &= \frac{1}{Z} \sum_{\boldsymbol{y}' \in \mathcal{Y}} \nabla_\theta \exp\left(\operatorname{score}(\boldsymbol{y}')\right) \\ &= \frac{1}{Z} \sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp\left(\operatorname{score}(\boldsymbol{y}')\right) \nabla_\theta \operatorname{score}(\boldsymbol{y}') \end{aligned}$$

We normally leave it to pytorch. Let's look closer.

Learning Via Sampling

There are some useful models for which we cannot compute argmax or logsumexp, but we can approximately draw samples of $P(\boldsymbol{y}|\boldsymbol{x})$.

We can still (noisily) train such models using Monte Carlo gradient estimation.

Recall our probabilistic objective:

$$L_{\text{NLL}}(\boldsymbol{y}) = -\operatorname{score}(\boldsymbol{y}) + \log \sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp \operatorname{score}(\boldsymbol{y}').$$

Its gradient:

$$\nabla_\theta L_{\text{NLL}}(\boldsymbol{y}) = -\nabla_\theta \operatorname{score}(\boldsymbol{y}) + \underbrace{\nabla_\theta \log \sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp \operatorname{score}(\boldsymbol{y}')}_{G_{\text{NLL}}}.$$

$$\begin{aligned} G_{\text{NLL}} &= \nabla_\theta \log \underbrace{\sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp\left(\operatorname{score}(\boldsymbol{y}')\right)}_{Z} \\ &= \frac{1}{Z} \sum_{\boldsymbol{y}' \in \mathcal{Y}} \nabla_\theta \exp\left(\operatorname{score}(\boldsymbol{y}')\right) \\ &= \frac{1}{Z} \sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp\left(\operatorname{score}(\boldsymbol{y}')\right) \nabla_\theta \operatorname{score}(\boldsymbol{y}') \\ &= \mathbb{E}_{\Pr(Y|\boldsymbol{x})}[\nabla_\theta \operatorname{score}(Y)] \end{aligned}$$

We normally leave it to pytorch. Let's look closer.

Learning Via Sampling

There are some useful models for which we cannot compute argmax or logsumexp, but we can approximately draw samples of $P(\boldsymbol{y}|\boldsymbol{x})$.

We can still (noisily) train such models using Monte Carlo gradient estimation.

Recall our probabilistic objective:

$$L_{\text{NLL}}(\boldsymbol{y}) = -\operatorname{score}(\boldsymbol{y}) + \log \sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp \operatorname{score}(\boldsymbol{y}').$$

Its gradient:

$$\nabla_\theta L_{\text{NLL}}(\boldsymbol{y}) = -\nabla_\theta \operatorname{score}(\boldsymbol{y}) + \underbrace{\nabla_\theta \log \sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp \operatorname{score}(\boldsymbol{y}')}_{G_{\text{NLL}}}.$$

We normally leave it to pytorch. Let's look closer.

$$\begin{aligned}
G_{\text{NLL}} &= \nabla_\theta \log \underbrace{\sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp\left(\operatorname{score}(\boldsymbol{y}')\right)}_{Z} \\
&= \frac{1}{Z} \sum_{\boldsymbol{y}' \in \mathcal{Y}} \nabla_\theta \exp\left(\operatorname{score}(\boldsymbol{y}')\right) \\
&= \frac{1}{Z} \sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp\left(\operatorname{score}(\boldsymbol{y}')\right) \nabla_\theta \operatorname{score}(\boldsymbol{y}') \\
&= \mathbb{E}_{\Pr(Y|\boldsymbol{x})}[\nabla_\theta \operatorname{score}(Y)] \\
&\approx \nabla_\theta \frac{1}{S} \sum_{i=1}^{S} \operatorname{score}(\tilde{\boldsymbol{y}}^{(i)}) \quad \text{for samples } \tilde{\boldsymbol{y}}^{(1)}, \ldots \tilde{\boldsymbol{y}}^{(S)}.
\end{aligned}$$

Learning Via Sampling: Surrogate Loss

True probabilistic objective:

$$L_{\text{NLL}}(\boldsymbol{y}) = -\operatorname{score}(\boldsymbol{y}) + \log \sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp \operatorname{score}(\boldsymbol{y}').$$

Surrogate objective: sample $\tilde{\boldsymbol{y}}^{(1)}, \ldots, \tilde{\boldsymbol{y}}^{(S)}$ from $\Pr(Y|\boldsymbol{x})$, and compute

$$\tilde{L}_{\text{NLL}}(\boldsymbol{y}) = -\operatorname{score}(\boldsymbol{y}) + \frac{1}{S} \sum_{i=1}^{S} \operatorname{score}(\tilde{\boldsymbol{y}}^{(i)}).$$

In the limit,

$$\lim_{S \to \infty} \nabla_\theta \tilde{L}_{\text{NLL}}(\boldsymbol{y}) = \nabla_\theta L_{\text{NLL}}(\boldsymbol{y}).$$

Give pytorch $\tilde{L}_{\text{NLL}}$ and it will compute a consistent estimate of the gradient.

Learning Via Sampling: Surrogate Loss

True probabilistic objective:

$$L_{\text{NLL}}(\boldsymbol{y}) = -\operatorname{score}(\boldsymbol{y}) + \log \sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp \operatorname{score}(\boldsymbol{y}').$$

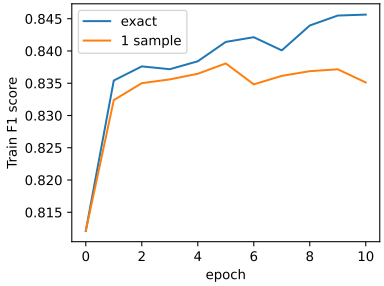
Surrogate objective: sample $\tilde{\boldsymbol{y}}^{(1)}, \ldots, \tilde{\boldsymbol{y}}^{(S)}$ from $\Pr(Y|\boldsymbol{x})$, and compute

$$\tilde{L}_{\text{NLL}}(\boldsymbol{y}) = -\operatorname{score}(\boldsymbol{y}) + \frac{1}{S} \sum_{i=1}^{S} \operatorname{score}(\tilde{\boldsymbol{y}}^{(i)}).$$

In the limit,

$$\lim_{S \to \infty} \nabla_\theta \tilde{L}_{\text{NLL}}(\boldsymbol{y}) = \nabla_\theta L_{\text{NLL}}(\boldsymbol{y}).$$

Give pytorch $\tilde{L}_{\text{NLL}}$ and it will compute a consistent estimate of the gradient.



This is an approximation: when available, exact logsumexp should be better.

Using multiple samples helps, but also increases cost.

Lecture 13

Sampling and Intractable Models

Part 3: Intractable Structured Models

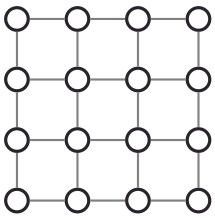
Machine Learning for Structured Data
Vlad Niculae · LTL, UvA · https://vene.ro/mlsd

Sampling and Intractable Models

1. Sampling From Structured Probabilistic Models
2. Learning Via Sampling
3. **Intractable Structured Models**

The Potts Model for Grid Tagging

The Potts model is the grid analogy of our sequence tagging model.



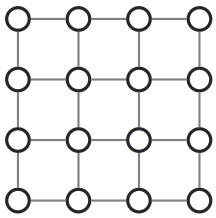
We've seen it before for semantic image segmentation.

Originally from physics (neighboring particles influence each other).

Configurations $\boldsymbol{y}$ are matrices with $y_{ij} \in \{1, \ldots, K\}$.

The Potts Model for Grid Tagging

The Potts model is the grid analogy of our sequence tagging model.



We've seen it before for semantic image segmentation.

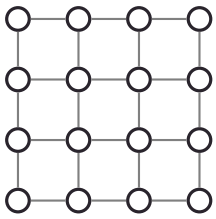
Originally from physics (neighboring particles influence each other).

Configurations $\boldsymbol{y}$ are matrices with $y_{ij} \in \{1, \ldots, K\}$.

How many possible labelings?

The Potts Model for Grid Tagging

The Potts model is the grid analogy of our sequence tagging model.



We've seen it before for semantic image segmentation.

Originally from physics (neighboring particles influence each other).

Configurations $\boldsymbol{y}$ are matrices with $y_{ij} \in \{1, \ldots, K\}$.

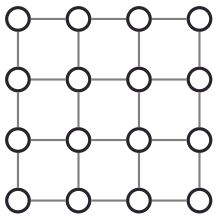
How many possible labelings?

Denoting $V = \{(0,0), (0,1), \ldots, (n,m)\}$ and E the set of edges,

$$\text{score}(\boldsymbol{y}) = \underbrace{\sum_{(i,j) \in V} a_{ij y_{ij}}}_{\text{unary}} + \underbrace{\sum_{((i,j),(i',j')) \in E} t_{y_{ij}, y_{i'j'}}}_{\text{pairwise}}$$

The Potts Model for Grid Tagging

The Potts model is the grid analogy of our sequence tagging model.



We've seen it before for semantic image segmentation.

Originally from physics (neighboring particles influence each other).

Configurations $\boldsymbol{y}$ are matrices with $y_{ij} \in \{1, \ldots, K\}$.

How many possible labelings?

Denoting $V = \{(0,0), (0,1), \ldots, (n,m)\}$ and E the set of edges,

$$\text{score}(\boldsymbol{y}) = \underbrace{\sum_{(i,j) \in V} a_{ij y_{ij}}}_{\text{unary}} + \underbrace{\sum_{((i,j),(i',j')) \in E} t_{y_{ij}, y_{i'j'}}}_{\text{pairwise}}$$

An algorithm is available only for $K = 2$ (a.k.a., Ising model) and only when the pairwise score $t \geq 0$ ("ferromagnetic").

What to do in general?

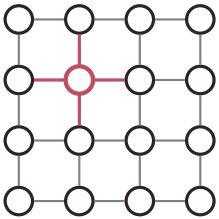
Gibbs sampling

Initialize $\boldsymbol{y}$.

Repeat:

pick a variable $y_v : v \in V$
sample y_v given $\boldsymbol{y}_{\neg v}$ (all other variables)

After enough repetitions, converges to a sample from $P(\boldsymbol{y})$. (i.e., configurations with high score are more likely than ones with low score).



In general:

$$\Pr(y_v = c \mid \boldsymbol{y}_{\neg v}) = \frac{\exp \text{score}(y_1, \ldots, y_{v-1}, c, y_{v+1}, \ldots)}{\sum_{c' \in [K]} \exp \text{score}(y_1, \ldots, y_{v-1}, c', y_{v+1}, \ldots)}$$

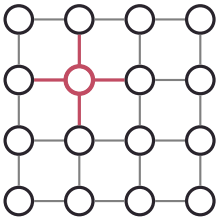
Gibbs sampling

Initialize $\boldsymbol{y}$.

Repeat:

pick a variable $y_v : v \in V$
sample y_v given $\boldsymbol{y}_{\neg v}$ (all other variables)

After enough repetitions, converges to a sample from $P(\boldsymbol{y})$. (i.e., configurations with high score are more likely than ones with low score).



In general:

$$\Pr(y_v = c \mid \boldsymbol{y}_{\neg v}) = \frac{\exp \text{score}(y_1, \ldots, y_{v-1}, c, y_{v+1}, \ldots)}{\sum_{c' \in [K]} \exp \text{score}(y_1, \ldots, y_{v-1}, c', y_{v+1}, \ldots)}$$

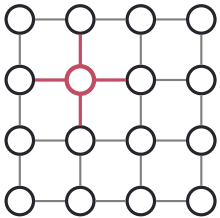
Gibbs sampling

Initialize $\boldsymbol{y}$.

Repeat:

pick a variable $y_v : v \in V$
sample y_v given $\boldsymbol{y}_{\neg v}$ (all other variables)

After enough repetitions, converges to a sample from $P(\boldsymbol{y})$. (i.e., configurations with high score are more likely than ones with low score).



In general:

$$\Pr(y_v = c \mid \boldsymbol{y}_{\neg v}) = \frac{\exp \text{score}(y_1, \ldots, y_{v-1}, c, y_{v+1}, \ldots)}{\sum_{c' \in [K]} \exp \text{score}(y_1, \ldots, y_{v-1}, c', y_{v+1}, \ldots)}$$

For Potts model, only the 4 neighbors matter: very fast.

$$P(y_{ij} = c \mid \boldsymbol{y}_{\neg ij}) \propto \exp\left(a_{ijc} + t_{y_{i-\mathbf{1},j},c} + t_{y_{i,j-\mathbf{1}},c} + t_{y_{i+\mathbf{1},j},c} + t_{y_{i,j+\mathbf{1}},c}\right)$$

Sequence Tagging with Global Scores

In sequence tagging, the efficient DP algorithm was possible thanks to the “chain” structure of the score

$$\text{score}(\boldsymbol{y}) = \sum_{j=1}^{n} a_{j,y_j} + \sum_{j=2}^{n} t_{y_{j-1},y_j}.$$

In some cases you might want score terms that don’t decompose so nicely:

- a reward score if all tags are equal
- a reward score for the number of times each tag was used

Sequence Tagging with Global Scores

In sequence tagging, the efficient DP algorithm was possible thanks to the "chain" structure of the score

$$\text{score}(\boldsymbol{y}) = \sum_{j=1}^{n} a_{j,y_j} + \sum_{j=2}^{n} t_{y_{j-1},y_j}.$$

In some cases you might want score terms that don't decompose so nicely:

- a reward score if all tags are equal
- a reward score for the number of times each tag was used

In general, DP might no longer be applicable. But Gibbs still works:

Sequence Tagging with Global Scores

In sequence tagging, the efficient DP algorithm was possible thanks to the “chain” structure of the score

$$\text{score}(\boldsymbol{y}) = \sum_{j=1}^{n} a_{j,y_j} + \sum_{j=2}^{n} t_{y_{j-1},y_j}.$$

In some cases you might want score terms that don’t decompose so nicely:

- a reward score if all tags are equal
- a reward score for the number of times each tag was used

In general, DP might no longer be applicable. But Gibbs still works:

ACAAGTCT

Sequence Tagging with Global Scores

In sequence tagging, the efficient DP algorithm was possible thanks to the “chain” structure of the score

$$\text{score}(\boldsymbol{y}) = \sum_{j=1}^{n} a_{j,y_j} + \sum_{j=2}^{n} t_{y_{j-1},y_j}.$$

In some cases you might want score terms that don’t decompose so nicely:

- a reward score if all tags are equal
- a reward score for the number of times each tag was used

In general, DP might no longer be applicable. But Gibbs still works:

```
ACAAG-CT
```

Sequence Tagging with Global Scores

In sequence tagging, the efficient DP algorithm was possible thanks to the "chain" structure of the score

$$\text{score}(\boldsymbol{y}) = \sum_{j=1}^{n} a_{j,y_j} + \sum_{j=2}^{n} t_{y_{j-1},y_j}.$$

In some cases you might want score terms that don't decompose so nicely:

- a reward score if all tags are equal
- a reward score for the number of times each tag was used

In general, DP might no longer be applicable. But Gibbs still works:

```
ACAAG-CT   ACAAGACT
           ACAAGCCT
           ACAAGGCT
           ACAAGTCT
```

Sequence Tagging with Global Scores

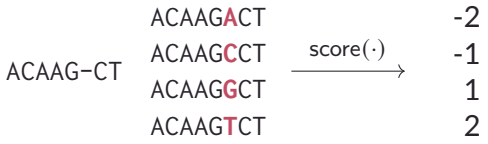
In sequence tagging, the efficient DP algorithm was possible thanks to the "chain" structure of the score

$$\text{score}(\boldsymbol{y}) = \sum_{j=1}^{n} a_{j,y_j} + \sum_{j=2}^{n} t_{y_{j-1},y_j}.$$

In some cases you might want score terms that don't decompose so nicely:

- a reward score if all tags are equal
- a reward score for the number of times each tag was used

In general, DP might no longer be applicable. But Gibbs still works:



Sequence Tagging with Global Scores

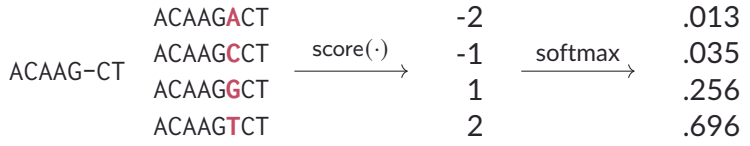
In sequence tagging, the efficient DP algorithm was possible thanks to the "chain" structure of the score

$$\text{score}(\boldsymbol{y}) = \sum_{j=1}^{n} a_{j,y_j} + \sum_{j=2}^{n} t_{y_{j-1},y_j}.$$

In some cases you might want score terms that don't decompose so nicely:

- a reward score if all tags are equal
- a reward score for the number of times each tag was used

In general, DP might no longer be applicable. But Gibbs still works:



Sequence Tagging with Global Scores

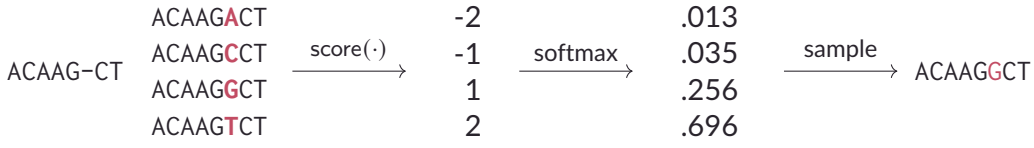
In sequence tagging, the efficient DP algorithm was possible thanks to the "chain" structure of the score

$$\text{score}(\boldsymbol{y}) = \sum_{j=1}^{n} a_{j,y_j} + \sum_{j=2}^{n} t_{y_{j-1},y_j}.$$

In some cases you might want score terms that don't decompose so nicely:

- a reward score if all tags are equal
- a reward score for the number of times each tag was used

In general, DP might no longer be applicable. But Gibbs still works:



Markov Chain Monte Carlo (MCMC)

Gibbs sampling is an instance of a powerful algorithm family known as MCMC.

These algorithms do not directly generate a single sample, but rather describe a process (chain) that slowly converges to the desired distribution.

Consecutive steps in the chain are nearly identical: they would not be good samples. You must wait enough steps in between. Finicky in general.

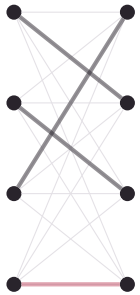
Gibbs Works Often But Not Always.

For highly constrained problems, Gibbs sampling might not work.

For instance, for the assignment problem, if $n = m$:

Given all other assignments but one, there is only one possible choice to sample from, so Gibbs is instantly stuck.

(There are some possible variations that might work here, hope is not lost, but it is definitely not a straightforward application.)



Other Ways To Sample

Sampling from a (discrete) distribution in general is a problem that is still widely researched.

This is especially the case if the distribution does not have a computable normalizing constant

e.g., $\Pr(y|x) = \exp \text{score}(y)/Z$ when we can compute $\text{score}(y)$ but not Z.

Some useful, very general methods, rely on sampling from an easier *proposal* distribution: accept-reject sampling, importance sampling. The performance depends heavily on how different the proposal is from the desired distribution.

Refer to texts about Monte Carlo methods to learn more.

Summary

Sampling from structured models

- another fundamental computation alongside argmax and logsumexp
- if the structure is solvable by DP, sampling is efficient: like following backpointers, but random, using quantities computed by the DP.
- if we can sample, we can learn probabilistic models, even without logsumexp.

Gibbs sampling

- sample structures by updating variables one at a time
- works for many (but not all) structures, including Potts models for grids.
- simple, but not great; hard to say how many updates needed.