Latent Structure Models for NLP

Tsvetomila Mihaylova  Instituto de Telecomunicações
Vlad Niculae  Instituto de Telecomunicações
work with:
André Martins  Instituto de Telecomunicações & IST & Unbabel
Nikita Nangia  NYU

.deep-spin.github.io/tutorial
I. Introduction
Structured prediction and NLP

- **Structured prediction**: a machine learning framework for predicting structured, constrained, and interdependent outputs
- **NLP** deals with *structured* and *ambiguous* textual data:
  - machine translation
  - speech recognition
  - syntactic parsing
  - semantic parsing
  - information extraction
  - ...
Examples of structure in NLP

Dependency parsing

...
Examples of structure in NLP

**POS tagging**
- **VERB** dog
  - PREP on
  - **NOUN** wheels
- **NOUN** dog
  - PREP on
  - **NOUN** wheels
- **NOUN** dog
  - **DET** on
  - **NOUN** wheels

**Dependency parsing**
- ... 
  - ⋆ dog  on  wheels
  - ⋆ dog  on  wheels
  - ⋆ dog  on  wheels

**Word alignments**
- dog  ↠ hond
  - on  ↠ op
  - wheels  ↢ wielen
- dog  ↢ hond
  - on  ↢ op
  - wheels  ↢ wielen
- dog  ↢ hond
  - on  ↢ op
  - wheels  ↢ wielen

Exponentially many structures!
Examples of structure in NLP

Dependency parsing

... 

⋆ dog on wheels

Exponentially many structures!

⋆ dog on wheels

...
NLP 5 years ago:
Structured prediction and pipelines
NLP 5 years ago:
Structured prediction and pipelines

- Big pipeline systems, connecting different structured predictors, trained separately
- **Advantages:** fast and simple to train, can rearrange pieces 😊
NLP 5 years ago:
Structured prediction and pipelines

• Big pipeline systems, connecting different structured predictors, trained separately
• **Advantages**: fast and simple to train, can rearrange pieces 😊
• **Disadvantage**: linguistic annotations required for each component 😞
NLP 5 years ago:
Structured prediction and pipelines

- Big pipeline systems, connecting different structured predictors, trained separately
- **Advantages:** fast and simple to train, can rearrange pieces 😊
- **Disadvantage:** linguistic annotations required for each component 😞
- **Bigger disadvantage:** error propagates through the pipeline 💩
NLP today:
End-to-end training
NLP today: End-to-end training

- Forget pipelines—train everything from scratch!
- No more error propagation or linguistic annotations! 🎉
NLP today:
End-to-end training

- Forget pipelines—train everything from scratch!
- No more error propagation or linguistic annotations!
- Treat everything as latent! 🙌
Representation learning

• Uncover hidden representations useful for the *downstream task*.
• Neural networks are well-suited for this: *deep computation graphs*. 

input

positive
neutral
negative

depth-spin.github.io/tutorial
Representation learning

- Uncover hidden representations useful for the *downstream task*.
- Neural networks are well-suited for this: *deep computation graphs*.
- Neural representations are unstructured, inscrutable. Language data has underlying structure!
Latent structure models

- Seek *structured* hidden representations instead!

Input:
- positive
- neutral
- negative
Latent structure models

- Seek *structured* hidden representations instead!

They can bring us:
- More interpretability;
- Be less inducive bias;
- Hopefully: smaller models.

input

positive
neutral
negative

deep-spin.github.io/tutorial
Latent structure models

• Seek \textit{structured} hidden representations instead!
• They can bring us:
  • More interpretability;
Latent structure models

- Seek *structured* hidden representations instead!
- They can bring us:
  - More interpretability;
  - Better inductive bias;
Latent structure models

- Seek *structured* hidden representations instead!
- They can bring us:
  - More interpretability;
  - Better inductive bias;
  - Hopefully: smaller models.
Latent structure models aren’t so new!

They have a very long history in NLP:

- IBM Models for SMT (latent word alignments) [Brown et al., 1993]
- HMMs [Rabiner, 1989]
- CRFs with hidden variables [Quattoni et al., 2007]
- Latent PCFGs [Petrov and Klein, 2008, Cohen et al., 2012]

Trained with EM, spectral learning, method of moments, ...

Often, very strict assumptions (e.g. strong factorizations)

Today, neural networks opened up some new possibilities!
What this tutorial is about:

- Discrete, combinatorial latent structures
- Often the structure is inspired by some linguistic intuition
- We’ll cover both:
  - RL methods (structure built incrementally, reward coming from downstream task)
  - ... vs end-to-end differentiable approaches (global optimization, marginalization)
  - stochastic computation graphs
  - ... vs deterministic graphs.
- All plugged in *discriminative* neural models.
This tutorial is not about:

- It’s not about continuous latent variables
- It’s not about deep generative learning
- We won’t cover GANs, VAEs, etc.
- There are (very good) recent tutorials on deep variational models for NLP:
  - “Variational Inference and Deep Generative Models” (Schulz and Aziz, ACL 2018)
Unstructured vs structured

- Simplest example of structure: Just a discrete choice among $N$ categories.
- We call this *unstructured*.
- It will provide an important starting point.
The challenge of discrete choices

\[ z = 1 \]
\[ z = 2 \]
\[ \ldots \]
\[ z = N \]
The challenge of discrete choices

\[ s \]

\[ z = 1 \]

\[ z = 2 \]

\[ \ldots \]

\[ z = N \]
The challenge of discrete choices

\begin{align*}
    s & \quad z \\
    z = 1 & \quad z = 2 \\
    z = 2 & \quad \ldots \\
    \vdots & \quad \vdots \\
    z = N & \quad \vdots \\
\end{align*}

or, essentially,

\[ \frac{\partial L(b, y)}{\partial w} = \text{?} \]
The challenge of discrete choices

input $x$

$s = f_\theta(x)$

output $\hat{y}$

$s$

$z = 1$

$z = 2$

$\ldots$

$z = N$

$\hat{y} = g_\phi(z, x)$
The challenge of discrete choices

\[ z = 1 \]
\[ z = 2 \]
\[ \ldots \]
\[ z = N \]

\[ s = f_\theta(x) \]

\[ \hat{y} = g_\phi(z, x) \]

\[ \frac{\partial L(\hat{y}, y)}{\partial w} = ? \]
The challenge of discrete choices

\[ z = 1 \]
\[ z = 2 \]
\[ \ldots \]
\[ z = N \]

\[ s = f_\theta(x) \]

\[ \hat{y} = g_\phi(z, x) \]

\[ \frac{\partial L(\hat{y}, y)}{\partial w} = ? \]

or, essentially,

\[ \frac{\partial z}{\partial s} = ? \]
Discrete mappings are “flat”

\[
\begin{align*}
  &s &\quad &z \\
  & &z = 1 &\quad &\text{gray}
  & &z = 2 &\quad &\text{gray}
  & &\ldots &\quad &\text{white}
  & &z = N &\quad &\text{white}
\end{align*}
\]

\[
\frac{\partial z}{\partial s} = ?
\]
Discrete mappings are “flat”

\[
\begin{align*}
\frac{\partial z}{\partial s} &= ? \\
\end{align*}
\]

\[
\begin{array}{c|c}
\hline
s & z \\
\hline
 & 1 \\
 & 2 \\
 & \ldots \\
 & N \\
\hline
\end{array}
\]
Discrete mappings are “flat”

\[
\begin{align*}
  z = 1 & \quad \text{when } s = 0 \\
  z = 2 & \quad \text{when } s = 1 \\
  \ldots & \\
  z = N & \quad \text{when } s = N-1 \\
  \frac{\partial z}{\partial s} & = ?
\end{align*}
\]
Discrete mappings are “flat”

\[
\begin{array}{c|c}
 s & z \\
 \hline
 1 & \quad \vdots \quad \\
 2 & \\
 N & \\
 \end{array}
\]

\[
\frac{\partial z}{\partial s} = ?
\]
Discrete mappings are “flat”

\[
\begin{array}{c}
\text{s} \\
\text{\begin{tabular}{c}
\text{z = 1} \\
\text{z = 2} \\
\text{\ldots} \\
\text{z = N}
\end{tabular}}
\end{array}
\begin{array}{c}
\text{z} \\
\text{\begin{tabular}{c}
\text{\frac{\partial z}{\partial s} = ?}
\end{tabular}}
\end{array}
\]
Discrete mappings are “flat”

\[
\begin{align*}
\partial z & = ? \\
\frac{\partial z}{\partial s} & = ?
\end{align*}
\]
Discrete mappings are “flat”

\[ s \]

\[
\begin{array}{c}
\text{z = 1} \\
\text{z = 2} \\
\vdots \\
\text{z = N}
\end{array}
\]

\[
\frac{\partial z}{\partial s} = ?
\]
Discrete mappings are “flat”

\[
\begin{align*}
  s & \quad z \\
  \text{z = 1} & \quad \text{z = 2} \\
  \text{...} & \\
  \text{z = N} & \\
  \frac{\partial z}{\partial s} &= ?
\end{align*}
\]
Argmax

\[ \frac{\partial z}{\partial s} = 0 \]
Computing the most likely structure is a very high-dimensional argmax
Computing the most likely structure is a very high-dimensional argmax.

There are exponentially many structures

(\(s\) cannot fit in memory; we cannot “loop” over \(s\) nor \(z\)
Dealing with the combinatorial explosion

1. Incremental structures
   - Build structure **greedily**, as sequence of discrete choices (e.g., shift-reduce).
   - Scores (partial structure, action) tuples.
   - **Advantages:** flexible, rich histories.
   - **Disadvantages:** greedy, local decisions are suboptimal, error propagation.

2. Factorization into parts
   - Optimizes **globally** (e.g. Viterbi, Chu-Liu-Edmonds, Kuhn-Munkres).
   - Scores smaller parts.
   - **Advantages:** optimal, elegant, can handle hard & global constraints.
   - **Disadvantages:** strong assumptions.
The unstructured case: Probability simplex

\[ \Delta \]

Each vertex is an indicator vector, representing one class: 

\[ z_c = [0, \ldots, 0, 1, \ldots, 0] \text{th position}, \ldots, 0 \]

Points inside are probability vectors, a convex combination of classes: 

\[ p = \sum_c p_c = 1 \]
The unstructured case: Probability simplex

- Each vertex is an *indicator vector*, representing one class:

\[ z_c = [0, \ldots, 0, 1_\text{c^{th} position}}, 0, \ldots, 0]. \]
The unstructured case: Probability simplex

- Each vertex is an indicator vector, representing one class:
  \[ z_c = [0, \ldots, 0, 1, 0, \ldots, 0] \text{ at the } c^{th} \text{ position} \]

- Points inside are probability vectors, a convex combination of classes:
  \[ p \geq 0, \quad \sum_c p_c = 1. \]
What’s the analogous of \( \triangle \) for a structure?

- A structured object \( z \) can be represented as a *bit vector*.
What’s the analogous of $\triangle$ for a structure?

- A structured object $z$ can be represented as a *bit vector*.
- Example:
  - a dependency tree can be represented a $O(L^2)$ vector indexed by arcs
  - each entry is 1 iff the arc belongs to the tree
  - **structural constraints**: not all bit vectors represent valid trees!
What’s the analogous of $\triangle$ for a structure?

- A structured object $z$ can be represented as a *bit vector*.
- Example:
  - a dependency tree can be represented a $O(L^2)$ vector indexed by arcs
  - each entry is 1 iff the arc belongs to the tree
  - **structural constraints**: not all bit vectors represent valid trees!

\[
\begin{align*}
z_1 &= [1, 0, 0, 0, 1, 0, 0, 0, 1] \\
z_2 &= [0, 0, 1, 0, 0, 1, 1, 0, 0] \\
z_3 &= [1, 0, 0, 0, 1, 0, 0, 1, 0]
\end{align*}
\]

* dog on wheels

* dog on wheels

* dog on wheels
The structured case: Marginal polytope

[Wainwright and Jordan, 2008]
The structured case: Marginal polytope

- Each vertex corresponds to one such *bit vector* $z$
The structured case: Marginal polytope

- Each vertex corresponds to one such bit vector $z$
- Points inside correspond to marginal distributions: convex combinations of structured objects

$$
\mu = p_1 z_1 + \ldots + p_N z_N , \ p \in \Delta.
$$

Exponentially many terms

$p_1 = 0.2, \ z_1 = [1, 0, 0, 0, 1, 0, 0, 0, 1]$
$p_2 = 0.7, \ z_2 = [0, 0, 1, 0, 0, 1, 1, 0, 0] \ \Rightarrow \ \mu = [0.3, 0.7, 0.3, 0.7, 0.7, 0.1, 0.2]$
$p_3 = 0.1, \ z_3 = [1, 0, 0, 0, 1, 0, 0, 1, 0]$
Unstructured vs Structured

- Unstructured case: simplex $\Delta$
- Structured case: marginal polytope $\mathcal{M}$
Unstructured vs Structured

- Unstructured case: simplex $\Delta$
- Structured case: marginal polytope $\mathcal{M}$
Unstructured vs Structured

- Unstructured case: simplex $\Delta$
- Structured case: marginal polytope $\mathcal{M}$
Example: Regression with latent categorization

\[ u = \frac{1}{|x|} \sum_j E_{x_j} \]

\[ s = W_s u \]
Example: Regression with latent categorization

$$u = \frac{1}{|x|} \sum_{j} E_{x_j}$$

$$s = W_s u$$

predict topic $c$ ($z = e_c$)

Workarounds: circumventing the issue, bypassing discrete variables
Example: Regression with latent categorization

\[
\begin{align*}
    u &= \frac{1}{|x|} \sum_j E_{x_j} \\
    s &= W_s u \\
    v &= \text{tanh}(W_v[u, z]) \\
    \hat{y} &= W_y v \\
    L &= (\hat{y} - y)^2
\end{align*}
\]

predict topic \(c\) (\(z = e_c\))
Example: Regression with latent categorization

\[ u = \frac{1}{|x|} \sum_j E_{x_j} \]

\[ s = W_s u \]

\[ v = \tanh (W_v[u, z]) \]

\[ \hat{y} = W_y v \]

\[ L = (\hat{y} - y)^2 \]
Example: Regression with latent categorization

\[ u = \frac{1}{|x|} \sum_j E_{x_j} \]
\[ s = W_s u \]
\[ v = \tanh (W_v [u, z]) \]
\[ \hat{y} = W_y v \]
\[ L = (\hat{y} - y)^2 \]

\[ \frac{\partial L}{\partial W_s} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial v} \frac{\partial v}{\partial z} \frac{\partial z}{\partial s} \frac{\partial s}{\partial W_s} \equiv 0 \]
Example: Regression with latent categorization

\[ u = \frac{1}{|x|} \sum_j E_j \]

\[ \hat{y} = W_v \tanh (W_v [u, z]) \]

\[ L = (\hat{y} - y)^2 \]

Workarounds: circumventing the issue, bypassing discrete variables
Example: Regression with latent categorization

\[
\begin{align*}
\text{input } x & \rightarrow \text{embeddings } E \\
& \quad \vdots \\
\text{embeddings } E & \rightarrow u \\
\rightarrow & \quad W_s \\
& \quad \vdots \\
& \quad S \\
& \quad \vdots \\
& \quad Z \\
& \quad \vdots \\
& \quad v \\
\rightarrow & \quad W_v \\
& \quad \vdots \\
& \quad W_y \\
\rightarrow & \quad \text{output } \hat{y} \\
\end{align*}
\]

\[
u = \frac{1}{|x|} \sum_j E_{x_j}
\]

\[
s = W_s u
\]

\[
v = \tanh (W_v [u, z])
\]

\[
\hat{y} = W_y v \\
L = (\hat{y} - y)^2
\]

Option 1. Pretrain latent classifier \( W_s \)
Example: Regression with latent categorization

\[ u = \frac{1}{|x|} \sum_j E_{x_j} \]
\[ s = W_s u \]
\[ v = \tanh (W_v[u, z]) \]
\[ \hat{y} = W_y v \]
\[ L = (\hat{y} - y)^2 \]

Option 2. Multi-task learning
Example: Regression with latent categorization

\[ u = \frac{1}{|x|} \sum_j E_{x_j} \]

Tackling discreteness end-to-end

\[ L = (\hat{y} - y)^2 \]
Example: Regression with latent categorization

\[ u = \frac{1}{|x|} \sum_j E_{x_j} \]
\[ s = W_s u \]
\[ v = \text{tanh} \left( W_v [u, z] \right) \]
\[ \hat{y} = W_y v \]
\[ L = \mathbb{E}_z (\hat{y} - y)^2 \]

Option 3. Stochasticity! \( \frac{\partial \mathbb{E}_z (\hat{y}(z) - y)^2}{\partial W_s} \neq 0 \)
Example: Regression with latent categorization

\[
\begin{align*}
    u &= \frac{1}{|x|} \sum_j E_{x_j} \\
    s &= W_s u \\
    v &= \tanh (W_v [u, z]) \\
    \hat{y} &= W_y v \\
    L &= (\hat{y} - y)^2
\end{align*}
\]

Option 4. Gradient surrogates (e.g. straight-through, \( \frac{\partial z}{\partial s} \leftarrow I \))
Example: Regression with latent categorization

\[ u = \frac{1}{|x|} \sum_j E_{x_j} \]

\[ s = W_s u \]

\[ v = \tanh (W_v [u, p]) \]

\[ \hat{y} = W_y v \]

\[ L = (\hat{y} - y)^2 \]

Option 5. Continuous relaxation (e.g. softmax)
Dealing with discrete latent variables

1. Pre-train external classifier
2. Multi-task learning
3. Stochastic latent variables
4. Gradient surrogates
5. Continuous relaxation
Dealing with discrete latent variables

1. Pre-train external classifier
2. Multi-task learning
3. Stochastic latent variables (Part 2)
4. Gradient surrogates (Part 3)
5. Continuous relaxation (Part 4)
Roadmap of the tutorial

- Part 1: Introduction ✓
- Part 2: Reinforcement learning

Coffee Break

- Part 3: Gradient surrogates
- Part 4: End-to-end differentiable models (1/2)

Coffee Break

- Part 4: End-to-end differentiable models (2/2)
- Part 5: Conclusions
II. Reinforcement Learning Methods
Latent structure via marginalization

- Given a sentence-label pair \((x, y)\) and its known parse tree \(z\),
Latent structure via marginalization

- Given a sentence-label pair \((x, y)\) and its **known** parse tree \(z\), we can make a prediction \(\hat{y}(z; x)\)
Latent structure via marginalization

- Given a sentence-label pair \((x, y)\) and its **known** parse tree \(z\), we can make a prediction \(\hat{y}(z; x)\) and incur a loss,

\[
L(\hat{y}(z; x), y)
\]
Latent structure via marginalization

- Given a sentence-label pair \((x, y)\) and its known parse tree \(z\),
  we can make a prediction \(\hat{y}(z; x)\)
  and incur a loss,

\[
L(\hat{y}(z; x), y) \text{ or simply } L(z)
\]
Latent structure via marginalization

- Given a sentence-label pair \((x, y)\) and its **known** parse tree \(z\), we can make a prediction \(\hat{y}(z; x)\) and incur a loss,
  \[
  L(\hat{y}(z; x), y) \text{ or simply } L(z)
  \]
- But we don’t know \(z\)!
Latent structure via marginalization

- Given a sentence-label pair \((x, y)\) and its **known** parse tree \(z\), we can make a prediction \(\hat{y}(z; x)\) and incur a loss,
  \[ L(\hat{y}(z; x), y) \text{ or simply } L(z) \]

- But we don’t know \(z\)!
- In this section:
  we jointly learn a structured prediction model \(\pi_\theta(z \mid x)\)
Latent structure via marginalization

- Given a sentence-label pair \((x, y)\) and its **known** parse tree \(z\), we can make a prediction \(\hat{y}(z; x)\) and incur a loss,

\[
L(\hat{y}(z; x), y) \text{ or simply } L(z)
\]

- But we don’t know \(z\)!
- In this section:
  - we jointly learn a structured prediction model \(\pi_\theta(z \mid x)\) by optimizing the **expected loss**,

\[
\mathbb{E}_{\pi_\theta(z \mid x)}[L(z)]
\]
But first, supervised SPINN
Stack-augmented Parser-Interpreter Neural Network

[ Bowman et al., 2016 ]
Stack-augmented Parser-Interpreter Neural-Network

- Joint learning: Combines a constituency parser and a sentence representation model.
Stack-augmented Parser-Interpreter Neural-Network

- Joint learning: Combines a constituency parser and a sentence representation model.
- The parser, $f_\theta(x)$ is a transition-based shift-reduce parser. It looks at top two elements of stack and top element of the buffer.
Stack-augmented Parser-Interpreter Neural-Network

- Joint learning: Combines a constituency parser and a sentence representation model.
- The parser, $f_\theta(x)$ is a transition-based \textbf{shift-reduce} parser. It looks at top two elements of stack and top element of the buffer.
- \textbf{TreeLSTM} combines top two elements of the stack when the parser chooses the \textbf{REDUCE} action.
Stack-augmented Parser-Interpreter Neural Network

[ Bowman et al., 2016 ]

Deep-Spin.github.io/tutorial
Stack-augmented Parser-Interpreter Neural Network

[ Bowman et al., 2016 ]
Stack-augmented Parser-Interpreter Neural Network

[Bowman et al., 2016]
Stack-augmented Parser-Interpreter Neural Network

[Bowman et al., 2016]
Stack-augmented Parser-Interpreter Neural Network

[Bowman et al., 2016]
Stack-augmented Parser-Interpreter Neural Network

[Bowman et al., 2016]
Shift-Reduce parsing

We can write a shift-reduce style parse as a sequence of Bernoulli random variables,

$$z = \{z_1, \ldots, z_{2L-1}\}$$

where, $$z_j \in \{0, 1\} \ \forall j \in [1, 2L - 1]$$
Shift-Reduce parsing

A sequence of Bernoulli trials but with conditional dependence,

\[ p(z_1, z_2, \ldots, z_{2L-1}) = \prod_{j=1}^{2L-1} p(z_j \mid z_{<j}) \]
But now, removes syntactic supervision from SPINN.

We model the parse, $z$, as a latent variable scored by $f_\theta(x)$. With shift-reduce parsing, we're making discreet decisions. REINFORCE as a "natural" solution.
Latent structure learning with SPINN

• But now, remove syntactic supervision from SPINN.
Latent structure learning with SPINN

- But now, remove syntactic supervision from SPINN.

- We model the parse, $z$, as a latent variable scored by $f_\theta(x)$. 
Latent structure learning with SPINN

- But now, remove syntactic supervision from SPINN.

- We model the parse, $z$, as a latent variable scored by $f_{\theta}(x)$.

- With shift-reduce parsing, we’re making discrete decisions ⇒ REINFORCE as a “natural” solution.
Unsupervised SPINN
Unsupervised SPINN

No syntactic supervision.
Only reward is from the downstream task.
We only get this reward after parsing the full sentence.
Some basic terminology,

- The action space is $z_j \in \{\text{SHIFT, REDUCE}\}$, and $z$ is a sequence of actions.
Some basic terminology,

- The action space is \( z_j \in \{\text{SHIFT, REDUCE}\} \), and \( z \) is a sequence of actions.
- Training parser network parameters, \( \theta \) with REINFORCE.
Some basic terminology,

- The action space is $z_j \in \{\text{SHIFT, REDUCE}\}$, and $z$ is a sequence of actions.
- Training parser network parameters, $\theta$ with REINFORCE
- The state, $h$, is the top two elements of the stack and the top element of the buffer.
Some basic terminology,

- The action space is $z_j \in \{\text{SHIFT, REDUCE}\}$, and $z$ is a sequence of actions.
- Training parser network parameters, $\theta$ with REINFORCE
- The state, $h$, is the top two elements of the stack and the top element of the buffer.
- Learning a policy network $\pi(z \mid h; \theta)$
Some basic terminology,

- The action space is \( z_j \in \{ \text{SHIFT, REDUCE} \} \), and \( z \) is a sequence of actions.
- Training parser network parameters, \( \theta \) with REINFORCE
- The state, \( h \), is the top two elements of the stack and the top element of the buffer.
- Learning a policy network \( \pi(z | h; \theta) \)
- Maximize the reward, where \( R \) is performance on the downstream task like sentence classification.

[Williams, 1992]
Some basic terminology,

- The action space is $z_j \in \{\text{SHIFT, REDUCE}\}$, and $z$ is a sequence of actions.
- Training parser network parameters, $\theta$ with REINFORCE
- The state, $h$, is the top two elements of the stack and the top element of the buffer.
- Learning a policy network $\pi(z_j|h; \theta)$.
- Maximizing the reward, where $R$ is performance on the downstream task like sentence classification.

NOTE: Only a single reward at the end of parsing.
Through the looking glass of REINFORCE

\[ \nabla_\theta \mathbb{E}_{z \sim \pi_\theta(z|x)} [L(z)] \]
Through the looking glass of REINFORCE

\[ \nabla_{\theta} E_{z \sim \pi_{\theta}(z|x)}[L(z)] = \nabla_{\theta} \left[ \sum_z L(z) \pi_{\theta}(z \mid x) \right] \]

(Need to turn it into \( E[\cdot] \) so we can MC-estimate)
Through the looking glass of REINFORCE

\[ \nabla_{\theta} \mathbb{E}_{z \sim \pi_{\theta}(z|x)}[L(z)] = \nabla_{\theta} \left[ \sum_{z} L(z) \pi_{\theta}(z | x) \right] \]

(Need to turn it into \( \mathbb{E}[\cdot] \) so we can MC-estimate)

\[ = \sum_{z} L(z) \nabla_{\theta} \pi_{\theta}(z | x) \]

\( \nabla \log f = \nabla f f \), so \( \nabla f = f \nabla \log f \).
Through the looking glass of REINFORCE

\[
\nabla_{\theta} \mathbb{E}_{z \sim \pi_{\theta}(z|x)} [L(z)] = \nabla_{\theta} \left[ \sum_z L(z) \pi_{\theta}(z \mid x) \right]
\]

(Need to turn it into \( \mathbb{E}[\cdot] \) so we can MC-estimate)

\[
= \sum_z L(z) \nabla_{\theta} \pi_{\theta}(z \mid x)
\]

\[
\nabla \log f = \frac{\nabla f}{f}, \text{ so } \nabla f = f \nabla \log f.
\]
Through the looking glass of REINFORCE

\[ \nabla_{\theta} \mathbb{E}_{z \sim \pi_{\theta}(z|x)}[L(z)] = \nabla_{\theta} \left[ \sum_{z} L(z) \pi_{\theta}(z \mid x) \right] \]

(Need to turn it into \( \mathbb{E}[\cdot] \) so we can MC-estimate)

\[ = \sum_{z} L(z) \nabla_{\theta} \pi_{\theta}(z \mid x) \]

\[ = \sum_{z} L(z) \pi_{\theta}(z \mid x) \nabla_{\theta} \log \pi_{\theta}(z \mid x) \]
Through the looking glass of REINFORCE

\[ \nabla_\theta \mathbb{E}_{z \sim \pi_\theta(z|x)} [L(z)] = \nabla_\theta \left[ \sum_z L(z) \pi_\theta(z \mid x) \right] \]

(Need to turn it into \( \mathbb{E}[\cdot] \) so we can MC-estimate)

\[ = \sum_z L(z) \nabla_\theta \pi_\theta(z \mid x) \]

\[ = \sum_z L(z) \pi_\theta(z \mid x) \nabla_\theta \log \pi_\theta(z \mid x) \]

\[ = \mathbb{E}_{z \sim \pi_\theta(z|x)} [L(z) \nabla_\theta \log \pi_\theta(z \mid x)] \]
Yogatama et al. [2017] uses REINFORCE to train SPINN!
Yogatama et al. [2017] uses REINFORCE to train SPINN!
However, this vanilla implementation isn’t very effective at learning syntax.
Yogatama et al. [2017] uses REINFORCE to train SPINN!

However, this vanilla implementation isn’t very effective at learning syntax. This model fails to solve a simple toy problem.
Toy problem: ListOps

\[[\text{max} \ 2 \ 9 \ \text{[min} \ 4 \ 7 \ ] \ 0 \ ]\]
**Toy problem: ListOps**

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTM</td>
<td>74.4</td>
</tr>
<tr>
<td>RL-SPINN</td>
<td>64.8</td>
</tr>
<tr>
<td>TreeLSTM with ground-truth trees</td>
<td>98.7</td>
</tr>
</tbody>
</table>

[Nangia and Bowman, 2018]
Toy problem: ListOps

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTM</td>
<td>74.4</td>
</tr>
<tr>
<td>RL-SPINN</td>
<td>4.8</td>
</tr>
<tr>
<td>TreeLSTM with ground-truth trees</td>
<td>8.7</td>
</tr>
</tbody>
</table>

But why?

[Nangia and Bowman, 2018]
RL-SPINN’s Troubles

This system faces at least two big problems,
This system faces at least two big problems,

1. High variance of gradients
2. Coadaptation
High variance

- We have a single reward at the end of parsing.
High variance

- We have a single reward at the end of parsing.
- We are sampling parses from very large search space! The Catalan number of binary trees.
High variance

- We have a single reward at the end of parsing.
- We are sampling parses from very large search space! **Catalan number** of binary trees.

\[
\begin{align*}
3 \text{ tokens} & \Rightarrow 5 \text{ trees} \\
5 \text{ tokens} & \Rightarrow 42 \text{ trees} \\
10 \text{ tokens} & \Rightarrow 16796 \text{ trees}
\end{align*}
\]
High variance

- We have a single reward at the end of parsing.
- We are sampling parses from very large search space! \textbf{Catalan number} of binary trees.
- And the policy is stochastic.
High variance

So, sometimes the policy lands in a “rewarding state”:

\[
[\text{sm} \ [\text{sm} \ [\text{sm} \ [\text{max} \ 5 \ 6] \ 2] \ 0] \ 5 \ 0 \ 8 \ 6]\]

Figure: Truth: 7; Pred: 7
High variance

Sometimes it doesn’t:

```
```

Figure: Truth: 6; Pred: 5
High variance

**Catalan number** of parses means we need many many samples to lower variance!
High variance

**Catalan number** of parses means we need many many samples to lower variance!

Possible solutions:

1. Gradient normalization
2. Control variates, aka baselines
Control variates

- A simple control variate: moving average of recent rewards
Control variates

- A simple control variate: moving average of recent rewards
- Parameters are updated using the advantage which is the difference between the reward, $R$, and the baseline prediction.
Control variates

- A simple control variate: moving average of recent rewards
- Parameters are updated using the advantage which is the difference between the reward, $R$, and the baseline prediction.

So,

$$
\nabla \mathbb{E}_{z \sim \pi(z)} = \mathbb{E}_{z \sim \pi(z)} [(L(z) - b(x)) \nabla \pi(z)]
$$
This system faces two big problems,

1. High variance of gradients
2. Coadaptation
Coadaptation problem

Learning composition function parameters $\phi$ with backpropagation, and parser parameters $\theta$ with REINFORCE.
Coadaptation problem

Learning composition function parameters $\phi$ with backpropagation, and parser parameters $\theta$ with REINFORCE.

Generally, $\phi$ will be learned more quickly than $\theta$, making it harder to explore the parsing search space and optimize for $\theta$. 

 Difference in variance of two gradient estimates. Possible solution: Proximal Policy Optimization [Schulman et al., 2017].
Coadaptation problem

Learning composition function parameters $\phi$ with backpropagation, and parser parameters $\theta$ with REINFORCE.

Generally, $\phi$ will be learned more quickly than $\theta$, making it harder to explore the parsing search space and optimize for $\theta$.

Difference in variance of two gradient estimates.
Coadaptation problem

Learning composition function parameters $\phi$ with backpropagation, and parser parameters $\theta$ with REINFORCE.

Generally, $\phi$ will be learned more quickly than $\theta$, making it harder to explore the parsing search space and optimize for $\theta$.

Difference in variance of two gradient estimates.

Possible solution: Proximal Policy Optimization [Schulman et al., 2017].
Making REINFORCE+SPINN work

Havrylov et al. [2019] use,

1. Input dependent control variate
2. Gradient normalization
3. Proximal Policy Optimization
Making REINFORCE+SPINN work

Havrylov et al. [2019] use,

1. Input dependent control variate
2. Gradient normalization
3. Proximal Policy Optimization

They solve ListOps!
Making REINFORCE+SPINN work

Havrylov et al. [2019] use,

1. Input dependent control variate
2. Gradient normalization
3. Proximal Policy Optimization

They solve ListOps!

However, does not learn English grammars.
Should I? Shouldn’t I?

• Unbiased!
Should I? Shouldn’t I?

- Unbiased!
- High variance 😞
Should I? Shouldn’t I?

- Unbiased!
- In a simple setting, with enough tricks, it can work! 😊

- High variance 😞
Should I? Shouldn’t I?

- Unbiased!
- In a simple setting, with enough tricks, it can work! 😊
- High variance 😞
- Has not yet been very effective at learning English syntax.
Roadmap of the tutorial

- Part 1: Introduction ✓
- Part 2: Reinforcement learning ✓

Coffee Break

- Part 3: Gradient surrogates
- Part 4: End-to-end differentiable models (1/2)

Coffee Break

- Part 4: End-to-end differentiable models (2/2)
- Part 5: Conclusions
III. Gradient Surrogates
So far:

- Tackled **expected loss** in a **stochastic computation graph**

\[
\mathbb{E}_{\pi_\theta(z|x)}[L(z)]
\]

3A: try to optimize the **deterministic loss** directly
3B: use this strategy to reduce variance in the stochastic model.
So far:

- Tackled **expected loss** in a **stochastic computation graph**

\[ \mathbb{E}_{\pi_{\theta}(z|x)}[L(z)] \]

- Optimized with the **REINFORCE** estimator.

---

Additional discussion:

3A: Try to optimize the deterministic loss directly

3B: Use this strategy to reduce variance in the stochastic model.
So far:

- Tackled **expected loss** in a **stochastic computation graph**
  \[ \mathbb{E}_{\pi_{\theta}(z|x)}[L(z)] \]

- Optimized with the **REINFORCE** estimator.
- Struggled with variance & sampling.
So far:

- Tackled \textbf{expected loss} in a \textbf{stochastic computation graph}

\[ \mathbb{E}_{\pi_{\theta}(z|x)}[L(z)] \]

- Optimized with the \textbf{REINFORCE} estimator.
- Struggled with variance & sampling.

In this section:

- Consider the \textbf{deterministic alternative}:
So far:

- Tackled **expected loss** in a **stochastic computation graph**
  \[
  \mathbb{E}_{\pi_\theta(z|x)}[L(z)]
  \]
- Optimized with the **REINFORCE** estimator.
- Struggled with variance & sampling.

In this section:

- Consider the **deterministic alternative**:
  
  pick highest-score structure
  \[
  \hat{z}(x) := \arg \max_{z \in M} \pi_\theta(z \mid x)
  \]
So far:

- Tackled **expected loss** in a **stochastic computation graph**

\[ \mathbb{E}_{\pi_{\theta}(z|x)}[L(z)] \]

- Optimized with the **REINFORCE** estimator.
- Struggled with variance & sampling.

In this section:

- Consider the **deterministic alternative**: pick highest-score structure \( \hat{z}(x) := \arg \max_{z \in \mathcal{M}} \pi_{\theta}(z | x) \) incur loss \( L(\hat{z}(x)) \)
So far:

- Tackled **expected loss** in a **stochastic computation graph**
  \[ E_{\pi \theta(z|x)}[L(z)] \]
- Optimized with the **REINFORCE** estimator.
- Struggled with variance & sampling.

In this section:

- Consider the **deterministic alternative:**
  - pick highest-score structure \( \hat{z}(x) := \arg \max_{z \in M} \pi_\theta(z | x) \)
  - incur loss \( L(\hat{z}(x)) \)
- 3A: try to optimize the deterministic loss directly
So far:

- Tackled **expected loss** in a **stochastic computation graph**
  \[ \mathbb{E}_{\pi_{\theta}(z|x)}[L(z)] \]
- Optimized with the **REINFORCE** estimator.
- Struggled with variance & sampling.

In this section:

- Consider the **deterministic alternative**:
  \[ \hat{z}(x) := \arg\max_{z \in \mathcal{M}} \pi_{\theta}(z | x) \]
  incur loss \[ L(\hat{z}(x)) \]
- 3A: try to optimize the deterministic loss directly
- 3B: use this strategy to reduce variance in the stochastic model.
Recap: The argmax problem

\[ z = \text{arg max}(s) \]

\[ \frac{\partial z}{\partial s} = 0 \]
So \[ z = 1 \]

\[ z = 2 \]

\[ z = N \]

\[ p_j = \exp(s_j)/Z \]

\[ \frac{\partial p}{\partial s} = \text{diag}(p) - pp^\top \]
Straight-Through Estimator

[Forward]: $z = \arg \max(s)$

[Backward]: pretend $z$ was some continuous $\sim p$; 

\[
\frac{\partial \sim p}{\partial s} = 0
\]

Simplest identity, $\sim p(s) = s$, \[\frac{\partial \sim p}{\partial s} = I\]

Others, e.g. softmax $\sim p(s) = \text{softmax}(s)$, \[\frac{\partial \sim p}{\partial s} = \text{diag}(\sim p) - \sim p \sim p^\top\]

More explanation in a while

What about the structured case?
Straight-Through Estimator

- **Forward**: $z = \arg \max(s)$
**Straight-Through Estimator**

- **Forward:** $z = \text{arg max}(s)$

---

[Hinton, 2012, Bengio et al., 2013]
Straight-Through Estimator

- **Forward**: $z = \arg\max(s)$
- **Backward**: pretend $z$ was some continuous $\tilde{p}$; $\frac{\partial \tilde{p}}{\partial s} \neq 0$
Straight-Through Estimator

- **Forward**: $z = \text{arg max}(s)$
- **Backward**: pretend $z$ was some continuous $\tilde{p}$; $\frac{\partial \tilde{p}}{\partial s} \neq 0$
Straight-Through Estimator

- **Forward**: \( z = \arg\max(s) \)
- **Backward**: pretend \( z \) was some continuous \( \tilde{p} \); \( \frac{\partial \tilde{p}}{\partial s} \neq 0 \)
  - simplest: identity, \( \tilde{p}(s) = s, \frac{\partial \tilde{p}}{\partial s} = I \)
Straight-Through Estimator

- **Forward**: \( z = \arg\max(\mathbf{s}) \)
- **Backward**: pretend \( z \) was some continuous \( \tilde{\mathbf{p}} \); \( \frac{\partial \tilde{\mathbf{p}}}{\partial \mathbf{s}} \neq 0 \)
  - simplest: identity, \( \tilde{\mathbf{p}}(\mathbf{s}) = \mathbf{s}, \frac{\partial \tilde{\mathbf{p}}}{\partial \mathbf{s}} = \mathbf{I} \)
  - others, e.g. softmax \( \tilde{\mathbf{p}}(\mathbf{s}) = \text{softmax}(\mathbf{s}), \frac{\partial \tilde{\mathbf{p}}}{\partial \mathbf{s}} = \text{diag}(\tilde{\mathbf{p}}) - \tilde{\mathbf{p}}\tilde{\mathbf{p}}^\top \)

[Hinton, 2012, Bengio et al., 2013]
Straight-Through Estimator

- **Forward**: $z = \arg\max(s)$
- **Backward**: pretend $z$ was some continuous $\tilde{p}$; $\frac{\partial \tilde{p}}{\partial s} \neq 0$
  - simplest: identity, $\tilde{p}(s) = s$, $\frac{\partial \tilde{p}}{\partial s} = I$
  - others, e.g. softmax $\tilde{p}(s) = \text{softmax}(s)$, $\frac{\partial \tilde{p}}{\partial s} = \text{diag}(\tilde{p}) - \tilde{p} \tilde{p}^T$
- More explanation in a while
Straight-Through Estimator

- **Forward**: $z = \arg\max(s)$
- **Backward**: pretend $z$ was some continuous $\tilde{p}$; $\frac{\partial \tilde{p}}{\partial s} \neq 0$
  - simplest: identity, $\tilde{p}(s) = s$, $\frac{\partial \tilde{p}}{\partial s} = I$
  - others, e.g. softmax $\tilde{p}(s) = \text{softmax}(s)$, $\frac{\partial \tilde{p}}{\partial s} = \text{diag}(\tilde{p}) - \tilde{p}\tilde{p}^T$
- More explanation: What about the structured case?
Dealing with the combinatorial explosion

1. Incremental structures
   - Build structure **greedily**, as sequence of discrete choices (e.g., shift-reduce).
   - Scores (partial structure, action) tuples.
   - **Advantages**: flexible, rich histories.
   - **Disadvantages**: greedy, local decisions are suboptimal, error propagation.

2. Factorization into parts
   - Optimizes **globally** (e.g. Viterbi, Chu-Liu-Edmonds, Kuhn-Munkres).
   - Scores smaller parts.
   - **Advantages**: optimal, elegant, can handle hard & global constraints.
   - **Disadvantages**: strong assumptions.
STE for incremental structures

Build a structure as a sequence of discrete choices (e.g., shift-reduce).

Assign a score to any (parallel structure, action) tuple. In this case, we just apply the straight-through emission for each step.

Forward: the highest scoring action for each step.

Backward: pretend that we had used a discriminable surrogate function.

STE for incremental structures

- Build a structure as a sequence of discrete choices (e.g., shift-reduce)
STE for incremental structures

- Build a structure as a sequence of discrete choices (e.g., shift-reduce)
- Assigns a score to any (partial structure, action) tuple.

STE for incremental structures

• Build a structure as a sequence of discrete choices (e.g., shift-reduce)
• Assigns a score to any (partial structure, action) tuple.
• In this case, we just apply the straight-through estimator for each step.
STE for incremental structures

- Build a structure as a sequence of discrete choices (e.g., shift-reduce)
- Assigns a score to any (partial structure, action) tuple.
- In this case, we just apply the straight-through estimator for each step.
- **Forward:** the highest scoring action for each step
STE for incremental structures

- Build a structure as a sequence of discrete choices (e.g., shift-reduce)
- Assigns a score to any (partial structure, action) tuple.
- In this case, we just apply the straight-through estimator for each step.
- **Forward**: the **highest scoring action** for each step
- **Backward**: pretend that we had used a **differentiable surrogate function**
STE for incremental structures

- Build a structure as a sequence of discrete choices (e.g., shift-reduce)
- Assigns a score to any (partial structure, action) tuple.
- In this case, we just apply the straight-through estimator for each step.
- **Forward**: the **highest scoring action** for each step
- **Backward**: pretend that we had used a **differentiable surrogate function**

STE for the factorized approach

Requires a bit more work:

- Recap: marginal polytope
- Predicting structures globally: Maximum A Posteriori (MAP)
- Deriving Straight-Through and SPIGOT
The structured case: Marginal polytope

[Wainwright and Jordan, 2008]
The structured case: Marginal polytope

- Each vertex corresponds to one such bit vector $z$
The structured case: Marginal polytope

- Each vertex corresponds to one such *bit vector* $\mathbf{z}$
- Points inside correspond to *marginal distributions*: convex combinations of structured objects

$$\mu = p_1 \mathbf{z}_1 + \ldots + p_N \mathbf{z}_N \ , \ p \in \Delta.$$  

exponentially many terms

$$p_1 = 0.2, \quad \mathbf{z}_1 = [1, 0, 0, 0, 1, 0, 0, 0, 1]$$  
$$p_2 = 0.7, \quad \mathbf{z}_2 = [0, 0, 1, 0, 0, 1, 1, 0, 0] \quad \Rightarrow \quad \mu = [0.3, 0.7, 0.3, 0.7, 0.7, 0.1, 0.2].$$  
$$p_3 = 0.1, \quad \mathbf{z}_3 = [1, 0, 0, 0, 1, 0, 0, 1, 0]$$
Predicting structures from scores of parts

- $\eta(i \rightarrow j)$: score of arc $i \rightarrow j$
- $z(i \rightarrow j)$: is arc $i \rightarrow j$ selected?
Predicting structures from scores of parts

- $\eta(i \rightarrow j)$: score of arc $i \rightarrow j$
- $z(i \rightarrow j)$: is arc $i \rightarrow j$ selected?
- Task-specific algorithm for the highest-scoring structure.
### Algorithms for specific structures

<table>
<thead>
<tr>
<th>Structure Type</th>
<th>Algorithms</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Best structure (MAP)</strong></td>
<td>Viterbi</td>
<td>[Rabiner, 1989]</td>
</tr>
<tr>
<td></td>
<td>CKY</td>
<td></td>
</tr>
<tr>
<td><strong>Sequence tagging</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Rabiner, 1989]</td>
<td></td>
</tr>
<tr>
<td><strong>Constituent trees</strong></td>
<td>CKY</td>
<td>[Kasami, 1966], [Younger, 1967]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[Cocke and Schwartz, 1970]</td>
</tr>
<tr>
<td><strong>Temporal alignments</strong></td>
<td>DTW</td>
<td>[Sakoe and Chiba, 1978]</td>
</tr>
<tr>
<td><strong>Dependency trees</strong></td>
<td>Max. Spanning Arborescence</td>
<td>[Chu and Liu, 1965], [Edmonds, 1967]</td>
</tr>
<tr>
<td><strong>Assignments</strong></td>
<td>Kuhn-Munkres</td>
<td>[Kuhn, 1955], [Jonker and Volgenant, 1987]</td>
</tr>
</tbody>
</table>
Structured Straight-Through

- **Forward pass:**
  Find highest-scoring structure:
  \[ z = \arg \max_{z \in Z} \eta^T z \]
- **Backward pass:**
  pretend we used \( \tilde{\mu} = \eta \).
Straight-Through Estimator
Revisited

In the forward pass, $z = \arg\max(s)$. If we had labels (multi-task learning), $L_{MTL} = L_y(z) + L_{hid}(s, z_{true})$.

One choice: perceptron loss
$L_{hid}(s, z_{true}) = s^\top z - s^\top z_{true}$; $\frac{\partial L_{hid}}{\partial s} = z - z_{true}$.

We don't have labels! Induce labels by "pulling back" the downstream target: the "best" (unconstrained) latent value would be: $\arg\min_{\tilde{z}} \tilde{z}^2 R_D L^y(\tilde{z})$, $y$.

One gradient descent step starting from $z_{true}$:
$z_{true} - \frac{\partial L_{MTL}}{\partial z} \frac{\partial L_{MTL}}{\partial s} = 0 + \frac{\partial L_{hid}}{\partial s} = z - z_{true} - \frac{\partial L_{hid}}{\partial z}$.

[Martins and Niculae, 2019]
Straight-Through Estimator
Revisited

- In the forward pass, $z = \arg \max(s)$. 
Straight-Through Estimator
Revisited

• In the forward pass, \( z = \text{arg max}(s) \).
• if we had labels (multi-task learning), \( L_{\text{MTL}} = L(\hat{\text{y}}(z), y) + L_{\text{hid}}(s, z^{\text{true}}) \)
Straight-Through Estimator
Revisited

- In the forward pass, $z = \text{arg max}(s)$.
- if we had labels (multi-task learning), $L_{\text{MTL}} = L(\hat{y}(z), y) + L_{\text{hid}}(s, z^{\text{true}})$
- One choice: perceptron loss $L_{\text{hid}}(s, z^{\text{true}}) = s^T z - s^T z^{\text{true}}$. $\frac{\partial L_{\text{hid}}}{\partial s} = z - z^{\text{true}}$. 

[Martins and Niculae, 2019]
Straight-Through Estimator
Revisited

- In the forward pass, $z = \arg \max(s)$.
- If we had labels (multi-task learning), $L_{MTL} = L(\hat{y}(z), y) + L_{hid}(s, z^{true})$
- One choice: perceptron loss $L_{hid}(s, z^{true}) = s^T z - s^T z^{true}$, $\frac{\partial L_{hid}}{\partial s} = z - z^{true}$.
- We don’t have labels! Induce labels by “pulling back” the downstream target: the “best” (unconstrained) latent value would be: $\arg \min_{\tilde{z} \in \mathbb{R}^d} L(\hat{y}(\tilde{z}), y)$

[Martins and Niculae, 2019]
depth-spin.github.io/tutorial
Straight-Through Estimator
Revisited

- In the forward pass, \( z = \arg \max(s) \).
- If we had labels (multi-task learning), \( L_{\text{MTL}} = L(\hat{y}(z), y) + L_{\text{hid}}(s, z^{\text{true}}) \).
- One choice: perceptron loss \( L_{\text{hid}}(s, z^{\text{true}}) = s^T z - s^T z^{\text{true}} \), \( \frac{\partial L_{\text{hid}}}{\partial s} = z - z^{\text{true}} \).
- We don’t have labels! Induce labels by “pulling back” the downstream target: the “best” (unconstrained) latent value would be: \( \arg\min_{\tilde{z} \in \mathbb{R}^D} L(\hat{y}(\tilde{z}), y) \).
- One gradient descent step starting from \( z \): \( z^{\text{true}} \leftarrow z - \frac{\partial L}{\partial z} \).
Straight-Through Estimator
Revisited

- In the forward pass, \( z = \arg \max(s) \).
- if we had labels (multi-task learning), \( L_{MTL} = L(\hat{y}(z), y) + L_{hid}(s, z^{\text{true}}) \)
- One choice: perceptron loss \( L_{hid}(s, z^{\text{true}}) = s^T z - s^T z^{\text{true}} \), \( \frac{\partial L_{hid}}{\partial s} = z - z^{\text{true}} \).
- We don’t have labels! Induce labels by “pulling back” the downstream target: the “best” (unconstrained) latent value would be: \( \arg \min_{\tilde{z} \in \mathbb{R}^D} L(\hat{y}(\tilde{z}), y) \)
- One gradient descent step starting from \( z \): \( z^{\text{true}} \leftarrow z - \frac{\partial L}{\partial z} \)

\[
\frac{\partial L_{MTL}}{\partial s} = \frac{\partial L}{\partial s} + \underbrace{\frac{\partial L_{hid}}{\partial s}}_{=0}
\]
Straight-Through Estimator
Revisited

- In the forward pass, $z = \arg \max(s)$.
- If we had labels (multi-task learning), $L_{MTL} = L(\hat{y}(z), y) + L_{hid}(s, z^{true})$
- One choice: perceptron loss $L_{hid}(s, z^{true}) = s^T z - s^T z^{true}$, $\frac{\partial L_{hid}}{\partial s} = z - z^{true}$.
- We don’t have labels! Induce labels by “pulling back” the downstream target: the “best” (unconstrained) latent value would be: $\arg \min_{z \in \mathcal{D}} L(\hat{y}(z), y)$
- One gradient descent step starting from $z$: $z^{true} \leftarrow z - \frac{\partial L}{\partial z}$

$$\frac{\partial L_{MTL}}{\partial s} = \frac{\partial L}{\partial s} + \frac{\partial L_{hid}}{\partial s} = z - \left( z - \frac{\partial L}{\partial z} \right) = \frac{\partial L}{\partial z}$$
Straight-Through in the structured case

- Structured STE: perceptron update with induced annotation

\[ \arg \min_{\mu \in \mathbb{R}^D} L(\hat{y}(\mu), y) \approx z - \nabla_z L(z) \rightarrow z^{\text{true}} \]

(one step of gradient descent)

\[ z^{\text{true}} = z - \nabla_z L(z) \]
Straight-Through in the structured case

- Structured STE: perceptron update with induced annotation

\[
\arg \min_{\mu \in \mathbb{R}^D} L(\hat{y}(\mu), y) \approx z - \nabla_z L(z) \to z^{\text{true}}
\]

(one step of gradient descent)

- SPIGOT takes into account the constraints; uses the induced annotation

\[
\arg \min_{\mu \in \mathcal{M}} L(\hat{y}(\mu), y) \approx \text{Proj}_\mathcal{M}(z - \nabla_z L(z)) \to z^{\text{true}}
\]

(one step of projected gradient descent!)
Structured STE: perceptron update with induced annotation

\[
\arg \min_{\mu \in \mathbb{R}^D} L(\hat{y}(\mu), y) \approx z - \nabla_z L(z) \rightarrow z^{\text{true}}
\]

(one step of gradient descent)

SPIGOT takes into account the constraints; uses the induced annotation

\[
\arg \min_{\mu \in \mathcal{M}} L(\hat{y}(\mu), y) \approx \text{Proj}_{\mathcal{M}} (z - \nabla_z L(z)) \rightarrow z^{\text{true}}
\]

(one step of projected gradient descent!)

We discuss a generic way to compute the projection in part 4.

[Peng et al., 2018, Martins and Niculae, 2019]
Summary: Straight-Through Estimator

We saw how to use the Straight-Through Estimator (STE) to allow learning models with argmax in the middle of the computation graph. We were optimizing $L(z(x))$.

Now we will see how to apply STE for stochastic graphs, as an alternative approach to REINFORCE.
Summary: Straight-Through Estimator

We saw how to use the *Straight-Through Estimator* to allow learning models with \textit{argmax} in the middle of the computation graph.
We saw how to use the *Straight-Through Estimator* to allow learning models with \( \text{argmax} \) in the middle of the computation graph.

We were optimizing \( L(\hat{z}(x)) \).
We saw how to use the *Straight-Through Estimator* to allow learning models with argmax in the middle of the computation graph.

We were optimizing $L(\hat{z}(x))$

Now we will see how to apply STE for stochastic graphs, as an alternative approach of REINFORCE.
Stochastic node in the computation graph

Recall the stochastic objective:

\[ E_{\pi_{\theta}} (z_j | x) \]

\[ L(z) \]

REINFORCE (previous section).

High variance.

An alternative using the reparameterization trick [Kingma and Welling, 2014].
Recall the stochastic objective:

\[ E_{\pi_\theta(z|x)}[L(z)] \]
Stochastic node in the computation graph

Recall the stochastic objective:

$$E_{\pi_\theta(z|x)}[L(z)]$$

- REINFORCE (previous section).
Stochastic node in the computation graph

Recall the stochastic objective:

$$\mathbb{E}_{\pi_\theta(z|x)}[L(z)]$$

- REINFORCE (previous section). High variance. 😞
Stochastic node in the computation graph

Recall the stochastic objective:

\[ \mathbb{E}_{\pi_{\theta}(z|x)}[L(z)] \]

- REINFORCE (previous section). High variance. 😞
- An alternative is using the reparameterization trick [Kingma and Welling, 2014].
Categorical reparameterization

Sampling from a categorical value in the middle of the computing graph.

\[ \pi(\theta(z|x)) \propto \exp(\theta(z|x)) \]

What is the gradient of a sample \( \frac{\partial z}{\partial \theta} \)?

Reparameterization: Move the stochasticity out of the gradient path. Makes \( z \) deterministic w.r.t. \( s \).
Categorical reparameterization

- Sampling from a categorical value in the middle of the computation graph.

\[ z \sim \pi_{\theta}(z \mid x) \propto \exp s_{\theta}(z \mid x) \]
Categorical reparameterization

- Sampling from a categorical value in the middle of the computation graph.
  \[ z \sim \pi_{\theta}(z \mid x) \propto \exp s_{\theta}(z \mid x) \]
- What is the gradient of a sample \( \frac{\partial z}{\partial \theta} \)?!

[Jang et al., 2017, Maddison et al., 2016]
Categorical reparameterization

- Sampling from a categorical value in the middle of the computation graph.
  \[ z \sim \pi_\theta(z \mid x) \propto \exp s_\theta(z \mid x) \]
- What is the gradient of a sample \( \frac{\partial z}{\partial \theta} \)?!
- Reparameterization: Move the stochasticity out of the gradient path.

[Jang et al., 2017, Maddison et al., 2016]
Categorical reparameterization

- Sampling from a categorical value in the middle of the computation graph.
  \[ z \sim \pi_{\theta}(z \mid x) \propto \exp s_{\theta}(z \mid x) \]
- What is the gradient of a sample \( \frac{\partial z}{\partial \theta} \)?!
- Reparameterization: Move the stochasticity out of the gradient path.
- Makes \( z \) deterministic w.r.t. \( s \)!

[Jang et al., 2017, Maddison et al., 2016]
Categorical reparameterization

- Sampling from a categorical value in the middle of the computation graph.
  \[ z \sim \pi_{\theta}(z \mid x) \propto \exp s_{\theta}(z \mid x) \]
- What is the gradient of a sample \( \frac{\partial z}{\partial \theta} \)?!
- Reparameterization: Move the stochasticity out of the gradient path.
- Makes \( z \) deterministic w.r.t. \( s \)!

\[ s \quad s + \epsilon \quad z \]

\( \epsilon \) (stochastic)
Categorical reparameterization

- Sampling from a categorical value in the middle of the computation graph.
  \[ z \sim \pi_\theta(z \mid x) \propto \exp s_\theta(z \mid x) \]
- What is the gradient of \( z \) with respect to \( \theta \)?
- Reparameterization: Move the stochasticity out of the gradient path.
  Makes \( z \) deterministic.
  As a result:
  Stochasticity is moved as an input.
  We can backpropagate through the deterministic input to \( z \).
Categorical reparameterization

\[ s \rightarrow s + \epsilon \rightarrow z \]

\[ \epsilon \text{ (stochastic)} \]

[Jang et al., 2017, Maddison et al., 2016]
Categorical reparameterization

How do we sample from a categorical variable?

[Jang et al., 2017, Maddison et al., 2016]
Sampling from a categorical variable

We want to sample from a categorical variable with scores $s$ (class $i$ has a score $s_i$)

1. Inverse transform sampling:
$$p = \text{softmax}(s)$$
$$c_i = \sum_j p_j$$
$$u \sim \text{Uniform}(0, 1)$$
$$z = \text{et. s.t.} \ c_t u < c_t + 1$$

2. The Gumbel-Max trick:
$$u_i \sim \text{Uniform}(0, 1)$$
$$\varepsilon_i = -\log(-\log(u_i))$$
$$z = \arg \max(s + \varepsilon)$$

The two methods are equivalent. (Not obvious, but we will not prove it now.)

Requires sampling from the Standard Gumbel Distribution $G(0, 1)$.

Derivation & more info: 
- Adams, 2013
- Vieira, 2014
Sampling from a categorical variable

We want to sample from a categorical variable with scores \( s \) (class \( i \) has a score \( s_i \))

1. Inverse transform sampling:

\[
\begin{align*}
p_i &= \text{softmax}(s) \\
c_i &= \sum_j p_j \\
u &\sim \text{Uniform}(0, 1) \\
return z &= \text{sample such that } c_t u < c_t + 1
\end{align*}
\]

2. The Gumbel-Max trick:

\[
\begin{align*}
u_i &\sim \text{Uniform}(0, 1) \\
\epsilon_i &= -\log(-\log(u_i)) \\
z &= \arg \max(s + \epsilon)
\end{align*}
\]

The two methods are equivalent.

(Not obvious, but we will not prove it now.)

Requires sampling from the Standard Gumbel Distribution \( G(0, 1) \).

References:

Sampling from a categorical variable

We want to sample from a categorical variable with scores \( s \) (class \( i \) has a score \( s_i \))

1. Inverse transform sampling:
   - \( p = \text{softmax}(s) \)

2. The Gumbel-Max trick:
   - \( u_i \sim \text{Uniform}(0, 1) \)
   - \( \epsilon_i = -\log(-\log(u_i)) \)
   - \( z = \arg \max(s + \epsilon) \)

The two methods are equivalent. (Not obvious, but we will not prove it now.) Requires sampling from the Standard Gumbel Distribution \( \mathcal{G}(0, 1) \). Derivation & more info:

- Adams, 2013
- Vieira, 2014
Sampling from a categorical variable

We want to sample from a categorical variable with scores $s$ (class $i$ has a score $s_i$)

1. Inverse transform sampling:
   - $p = \text{softmax}(s)$
   - $c_i = \sum_{j \leq i} p_j$

The two methods are equivalent. (Not obvious, but we will not prove it now.) Requires sampling from the Standard Gumbel Distribution $G(0,1)$.

Derivation & more info:
- [Adams, 2013]
- [Vieira, 2014]
We want to sample from a categorical variable with scores $s$ (class $i$ has a score $s_i$)

**1. Inverse transform sampling:**

- $p = \text{softmax}(s)$
- $c_i = \sum_{j \leq i} p_j$
- $u \sim \text{Uniform}(0, 1)$

The two methods are equivalent. (Not obvious, but we will not prove it now.) Requires sampling from the Standard Gumbel Distribution $G(0, 1)$. Derivation & more info: [Adams, 2013], [Vieira, 2014]
Sampling from a categorical variable

We want to sample from a categorical variable with scores $s$ (class $i$ has a score $s_i$)

1. Inverse transform sampling:
   - $p = \text{softmax}(s)$
   - $c_i = \sum_{j \leq i} p_j$
   - $u \sim \text{Uniform}(0, 1)$
   - return $z = e_t$ s.t. $c_t \leq u < c_{t+1}$

The two methods are equivalent.
(Not obvious, but we will not prove it now.)

Requiere sampling from the Standard Gumbel Distribution $G(0, 1)$.

Derivation & more info: [Adams, 2013, Vieira, 2014]
Sampling from a categorical variable

We want to sample from a categorical variable with scores $s$ (class $i$ has a score $s_i$)

1. Inverse transform sampling:
   \[
   p = \text{softmax}(s)
   \]
   \[
   c_i = \sum_{j \leq i} p_j
   \]
   \[
   u \sim \text{Uniform}(0, 1)
   \]
   \[
   \text{return } z = e_t \text{ s.t. } c_t \leq u < c_{t+1}
   \]

2. The Gumbel-Max trick:
   \[
   \epsilon_i = -\log(-\log(u_i))
   \]
   \[
   z = \arg\max (s + \epsilon)
   \]
   The two methods are equivalent. (Not obvious, but we will not prove it now.)

   Requires sampling from the Standard Gumbel Distribution $G(0, 1)$.

Reference & more info:
- Adams, 2013
- Vieira, 2014
Sampling from a categorical variable

We want to sample from a categorical variable with scores $s$ (class $i$ has a score $s_i$)

1. Inverse transform sampling:
   - $p = \text{softmax}(s)$
   - $c_i = \sum_{j \leq i} p_j$
   - $u \sim \text{Uniform}(0, 1)$
   - return $z = e_t$ s.t. $c_t \leq u < c_{t+1}$

2. The Gumbel-Max trick:
   - $u_i \sim \text{Uniform}(0, 1)$

The two methods are equivalent. (Not obvious, but we will not prove it now.)

Requires sampling from the Standard Gumbel Distribution $G(0, 1)$.
Sampling from a categorical variable

We want to sample from a categorical variable with scores $s$ (class $i$ has a score $s_i$)

1. Inverse transform sampling:
   - $p = \text{softmax}(s)$
   - $c_i = \sum_{j \leq i} p_j$
   - $u \sim \text{Uniform}(0, 1)$
   - return $z = e_t$ s.t. $c_t \leq u < c_{t+1}$

2. The Gumbel-Max trick:
   - $u_i \sim \text{Uniform}(0, 1)$
   - $\epsilon_i = -\log(-\log(u_i))$

The two methods are equivalent. (Not obvious, but we will not prove it now.) Requires sampling from the Standard Gumbel Distribution $G(0, 1)$.

Derivation & more info: [Adams, 2013, Vieira, 2014]
Sampling from a categorical variable

We want to sample from a categorical variable with scores $s$ (class $i$ has a score $s_i$)

1. **Inverse transform sampling:**
   - $p = \text{softmax}(s)$
   - $c_i = \sum_{j \leq i} p_j$
   - $u \sim \text{Uniform}(0, 1)$
   - return $z = e_t$ s.t. $c_t \leq u < c_{t+1}$

2. **The Gumbel-Max trick:**
   - $u_i \sim \text{Uniform}(0, 1)$
   - $\epsilon_i = -\log(-\log(u_i))$
   - $z = \arg\max(s + \epsilon)$

The two methods are equivalent. (Not obvious, but we will not prove it now.)

Requires sampling from the Standard Gumbel Distribution $G(0, 1)$.

Derivation & more info: [Adams, 2013], [Vieira, 2014]
Sampling from a categorical variable

We want to sample from a categorical variable with scores $s$ (class $i$ has a score $s_i$)

1. Inverse transform sampling:
   - $p = \text{softmax}(s)$
   - $c_i = \sum_{j \leq i} p_j$
   - $u \sim \text{Uniform}(0, 1)$
   - return $z = e_t$ s.t. $c_t \leq u < c_{t+1}$

2. The Gumbel-Max trick:
   - $u_i \sim \text{Uniform}(0, 1)$
   - $\epsilon_i = -\log(-\log(u_i))$
   - $z = \text{arg max}(s + \epsilon)$

The two methods are equivalent. (Not obvious, but we will not prove it now.)
Sampling from a categorical variable

We want to sample from a categorical variable with scores $s$ (class $i$ has a score $s_i$)

1. Inverse transform sampling:
   - $p = \text{softmax}(s)$
   - $c_i = \sum_{j \leq i} p_j$
   - $u \sim \text{Uniform}(0, 1)$
   - return $z = e_t$ s.t. $c_t \leq u < c_{t+1}$

2. The Gumbel-Max trick:
   - $u_i \sim \text{Uniform}(0, 1)$
   - $\epsilon_i = -\log(-\log(u_i))$
   - $z = \arg \max(s + \epsilon)$

The two methods are equivalent. (Not obvious, but we will not prove it now.)
Requires sampling from the Standard Gumbel Distribution $G(0,1)$. 

Sampling from a categorical variable

We want to sample from a categorical variable with scores \( s \) (class \( i \) has a score \( s_i \))

1. **Inverse transform sampling:**
   - \( p = \text{softmax}(s) \)
   - \( c_i = \sum_{j \leq i} p_j \)
   - \( u \sim \text{Uniform}(0, 1) \)
   - return \( z = e_t \text{ s.t. } c_t \leq u < c_{t+1} \)

2. **The Gumbel-Max trick:**
   - \( u_i \sim \text{Uniform}(0, 1) \)
   - \( \epsilon_i = -\log(-\log(u_i)) \)
   - \( z = \text{arg max}(s + \epsilon) \)

The two methods are equivalent. *(Not obvious, but we will not prove it now.)*

Requires sampling from the Standard Gumbel Distribution \( G(0, 1) \).

Derivation & more info: [Adams, 2013, Vieira, 2014]
Sampling from a categorical variable

We want to sample from a categorical variable with scores $s$ (class $i$ has a score $s_i$)

1. Inverse transform sampling:
   - $p = \text{softmax}(s)$
   - $c_i = \sum_{j \leq i} p_j$
   - $u \sim \text{Uniform}(0, 1)$
   - return $z = e_t$ s.t. $c_t < c_t + 1$

2. The Gumbel-Max trick:
   - $\epsilon_i = -\log(-\log(u_i))$
   - $z = \text{arg max}(s + \epsilon)$

The two methods are equivalent. (Not obvious, but we will not prove it now.)

Requiere sampling from the Standard Gumbel Distribution $\mathcal{G}(0, 1)$.

References:
- Adams, 2013
- Vieira, 2014
Sampling from a categorical variable

We want to sample from a categorical variable with scores $s$ (class $i$ has a score $s_i$)

1. **Inverse transform sampling:**
   - $p = \text{softmax}(s)$
   - $c_i = \sum_{j \leq i} p_j$
   - $u \sim \text{Uniform}(0,1)$
   - return $z = e_t$ s.t. $c_t < c_{t+1}$

2. **The Gumbel-Max trick:**
   - $\epsilon_i = -\log(-\log(u_i))$
   - $z = \text{arg max}(s + \epsilon)$

The two methods are equivalent. (Not obvious, but we will not prove it now.)

Requiere sampling from the Standard Gumbel Distribution $G(0,1)$.

Derivation & more info: [Adams, 2013, Vieira, 2014]
Straight-Through Gumbel Estimator

Apply a variant of the Straight-Through Estimator to Gumbel-Max!

[Jang et al., 2017, Maddison et al., 2016]

Forward: $z = \arg \max (s + \epsilon)$

Backward: pretend we had $\tilde{p} = \text{softmax}(s + \epsilon)$

What about the structured case?
Straight-Through Gumbel Estimator

Apply a variant of the Straight-Through Estimator to Gumbel-Max!

- **Forward:** $z = \arg\max(s + \epsilon)$

```
\[
\begin{align*}
\epsilon &= -\log(-\log(u_i)) \\
u_i &\sim U(0,1)
\end{align*}
\]
```
Straight-Through Gumbel Estimator

Apply a variant of the Straight-Through Estimator to Gumbel-Max!

- Forward: $z = \arg \max (s + \epsilon)$
- Backward: pretend we had done $\tilde{p} = \text{softmax}(s + \epsilon)$

\[\epsilon_i = -\log(-\log(u_i)) \quad u_i \sim U(0,1)\]
Straight-Through Gumbel Estimator

Apply a variant of the Straight-Through Estimator to Gumbel-Max!

- **Forward**: $z = \text{arg max}(s + \epsilon)$
- **Backward**: pretend we had done
  
  $\tilde{p} = \text{softmax}(s + \epsilon)$

What about the structured case?

$\epsilon_i = \log(-\log(u_i))$

$u_i \sim \text{U}(0, 1)$
Dealing with the combinatorial explosion

1. Incremental structures
   - Build structure greedily, as sequence of discrete choices (e.g., shift-reduce).
   - Scores (partial structure, action) tuples.
   - **Advantages:** flexible, rich histories.
   - **Disadvantages:** greedy, local decisions are suboptimal, error propagation.

2. Factorization into parts
   - Optimizes globally (e.g., Viterbi, Chu-Liu-Edmonds, Kuhn-Munkres).
   - Scores smaller parts.
   - **Advantages:** optimal, elegant, can handle hard & global constraints.
   - **Disadvantages:** strong assumptions.
Sampling from incremental structures

Build a structure as a sequence of discrete choices (e.g., shift-reduce).
Assign a score to any (parallel structure, action) tuple.
Reparameterize the scores with Gumbel-Max; now we have a deterministic node.

Forward: the argmax from the reparameterized scores for each step.

Backward: pretend we had used a differentiable surrogate function.
Example: GumbelTree-LSTM [Cho et al., 2018].

deep-spin.github.io/tutorial
Sampling from incremental structures

- Build a structure as a sequence of discrete choices (e.g., shift-reduce)
Sampling from incremental structures

- Build a structure as a sequence of discrete choices (e.g., shift-reduce)
- Assigns a score to any (partial structure, action) tuple.
Sampling from incremental structures

- Build a structure as a sequence of discrete choices (e.g., shift-reduce)
- Assigns a score to any (partial structure, action) tuple.
- Reparameterize the scores with Gumbel-Max - now we have a deterministic node.
Sampling from incremental structures

- Build a structure as a sequence of discrete choices (e.g., shift-reduce)
- Assigns a score to any (partial structure, action) tuple.
- Reparameterize the scores with Gumbel-Max - now we have a deterministic node.
- **Forward**: the argmax from the reparameterized scores for each step

Example: GumbelTree-LSTM [Cho et al., 2018]
Sampling from incremental structures

• Build a structure as a sequence of discrete choices (e.g., shift-reduce)
• Assigns a score to any (partial structure, action) tuple.
• Reparameterize the scores with Gumbel-Max - now we have a deterministic node.
• Forward: the **argmax** from the reparameterized scores for each step
• Backward: pretend we had used a **differentiable surrogate function**
Sampling from incremental structures

- Build a structure as a sequence of discrete choices (e.g., shift-reduce)
- Assigns a score to any (partial structure, action) tuple.
- Reparameterize the scores with Gumbel-Max - now we have a deterministic node.
- Forward: the **argmax** from the reparameterized scores for each step
- Backward: pretend we had used a **differentiable surrogate function**
  
  Example: Gumbel Tree-LSTM [Choi et al., 2018].
Example: Gumbel Tree-LSTM

- Building task-specific tree structures.
- Straight-Through Gumbel-Softmax at each step to select one arc.
Sampling from factorized models
Perturb-and-MAP

Reparameterize by **perturbing the arc scores**. (inexact!)
Sampling from factorized models
Perturb-and-MAP

Reparameterize by *perturbing the arc scores*. (inexact!)

- Sample from the standard Gumbel distribution.
- $\varepsilon \sim G(0, 1)$
Sampling from factorized models
Perturb-and-MAP

Reparameterize by **perturbing the arc scores.** (inexact!)

- Sample from the standard Gumbel distribution.
- Perturb the arc scores with the Gumbel noise.
- \( \epsilon \sim G(0, 1) \)
- \( \tilde{\eta} = \eta + \epsilon \)
Sampling from factorized models
Perturb-and-MAP

Reparameterize by perturbing the arc scores. (inexact!)

- Sample from the standard Gumbel distribution.
- Perturb the arc scores with the Gumbel noise.
- Compute MAP (task-specific algorithm).

- $\epsilon \sim G(0, 1)$
- $\tilde{\eta} = \eta + \epsilon$
- $\arg \max_{z \in \mathcal{Z}} \tilde{\eta}^T z$
Sampling from factorized models

Perturb-and-MAP

Reparameterize by **perturbing the arc scores.** (inexact!)

- Sample from the standard Gumbel distribution.
- Perturb the arc scores with the Gumbel noise.
- Compute MAP (task-specific algorithm).
- Backward: we could use Straight-Through with Identity.

\[ \varepsilon \sim G(0, 1) \]

\[ \tilde{\eta} = \eta + \varepsilon \]

\[ \arg \max_{\mathbf{z}\in\mathcal{Z}} \tilde{\eta}^\top \mathbf{z} \]
Summary: Gradient surrogates

• Based on the **Straight-Through Estimator**.
• Can be used for stochastic or deterministic computation graphs.
• **Forward pass**: Get an argmax (might be structured).
• **Backpropagation**: use a function, which we hope is close to argmax.
• Examples:
  • Argmax for iterative structures and factorization into parts
  • Sampling from iterative structures and factorization into parts
Gradient surrogates: Pros and cons

Pros

• Do not suffer from the high variance problem of REINFORCE.
• Allow for flexibility to select or sample a latent structured in the middle of the computation graph.
• Efficient computation.

Cons

• The Gumbel sampling with Perturb-and-MAP is an approximation.
• Bias, due to function mismatch on the backpropagation (next section will address this problem.)
Overview

\[ \mathbb{E}_{\pi_\theta(z|x)}[L(z)] \quad L(\arg \max_z \pi_\theta(z \mid x)) \]

- REINFORCE
- Straight-Through Gumbel (Perturb & MAP)
- Straight-Through
- SPIGOT

And more, in the next section!

Thank you!
Overview

\[ \mathbb{E}_{\pi_{\theta}(z|x)}[L(z)] \quad L(\text{arg max}_z \pi_{\theta}(z \mid x)) \quad L(\mathbb{E}_{\pi_{\theta}(z|x)}[z]) \]

- REINFORCE
- Straight-Through Gumbel (Perturb & MAP)

- Straight-Through
- SPIGOT

- Structured Attn. Nets
- SparseMAP

And more, in the next section!
IV. End-to-end Differentiable Relaxations
End-to-end differentiable relaxations

1. Digging into softmax
2. Alternatives to softmax
3. Generalizing to structured prediction
4. Stochasticity and global structures
Recall: Discrete choices & differentiability

\[ s = f_\theta(x) \]

\[ s = f_\theta(x) \]

\[ z = 1 \]
\[ z = 2 \]
\[ \ldots \]
\[ z = N \]

\[ \frac{\partial z}{\partial s} = 0 \text{ or } n/a \]

(\text{argmax})
One solution: smooth relaxation

\[ s = f_\theta(x) \]

\[ z = 1 \]
\[ z = 2 \]
\[ \ldots \]
\[ z = N \]

\[ p = \text{softmax}(s) = \mathbb{E}[z], \text{ i.e.} \]
replace \( \mathbb{E}[f(z)] \) with \( f(\mathbb{E}[z]) \)

\[ \frac{\partial p}{\partial s} = \smiley \]

(softmax)

\[ y = g_\phi(z, x) \]

\[ p_1 \rightarrow \hat{y} \]

\[ s_2 - 1 \quad s_2 \quad s_2 + 1 \]

\[ 0 \quad 1 \]

input

\[ x \]

output

\[ \hat{y} \]
One solution: smooth relaxation

\[ s = f_\theta(x) \]

\[ p = \text{softmax}(s) = \mathbb{E}[z], \text{ i.e. replace } \mathbb{E}[f(z)] \text{ with } f(\mathbb{E}[z]) \]

\[ \frac{\partial p}{\partial s} = \begin{cases} 1 \text{ (softmax)} & \text{if } s = 1 \\ 0 & \text{otherwise} \end{cases} \]
Overview

\begin{align*}
E_{\pi_\theta(z|x)}[L(z)] & \quad L(\arg \max_z \pi_\theta(z \mid x)) \quad L(E_{\pi_\theta(z|x)}[z])
\end{align*}

- REINFORCE
- Straight-Through Gumbel (Perturb & MAP)
- Straight-Through
- SPIGOT

...and more, in the next section!
What is softmax?

Often defined via

\[
p_i = \frac{\exp s_i}{\sum_j \exp s_j},
\]

but where does it come from?
What is softmax?

Often defined via $p_i = \frac{\exp s_i}{\sum_j \exp s_j}$, but where does it come from?

$p \in \Delta$: probability distribution over choices
What is softmax?

Often defined via $p_i = \frac{\exp s_i}{\sum_j \exp s_j}$, but where does it come from?

$p \in \Delta$: probability distribution over choices
What is softmax?

Often defined via \( p_i = \frac{\exp s_i}{\sum_j \exp s_j} \), but where does it come from?

\( p \in \Delta \): probability distribution over choices.
What is softmax?

Often defined via $p_i = \frac{\exp s_i}{\sum_j \exp s_j}$, but where does it come from?

$p \in \Delta$: probability distribution over choices

$p = [1/3, 1/3, 1/3]$
What is softmax?

Often defined via \( p_i = \frac{\exp s_i}{\sum_j \exp s_j} \), but where does it come from?

\( p \in \Delta \): probability distribution over choices

Expected score under \( p \): \( \mathbb{E}_{i \sim p} s_i = p^\top s \)

\( s = [.7, .1, 1.5] \)
What is softmax?

Often defined via $p_i = \frac{\exp s_i}{\sum_j \exp s_j}$, but where does it come from?

$p \in \Delta$: probability distribution over choices

Expected score under $p$: $\mathbb{E}_{i \sim p} s_i = p^\top s$

argmax

$s = [.7, .1, 1.5]$
What is softmax?

Often defined via $p_i = \frac{\exp s_i}{\sum_j \exp s_j}$, but where does it come from?

$p \in \Delta$: probability distribution over choices

Expected score under $p$: $\mathbb{E}_{i \sim p} s_i = p^\top s$

$\text{argmax}$ maximizes expected score
What is softmax?

Often defined via \( p_i = \frac{\exp s_i}{\sum_j \exp s_j} \), but where does it come from?

\( p \in \Delta \): probability distribution over choices

Expected score under \( p \): \( \mathbb{E}_{i \sim p} s_i = p^\top s \)

\texttt{argmax} maximizes \textbf{expected score}

Shannon entropy of \( p \): \( H(p) = - \sum_i p_i \log p_i \)

![Diagram showing the softmax function](https://deep-spin.github.io/tutorial)
What is softmax?

Often defined via \( p_i = \frac{\exp s_i}{\sum_j \exp s_j} \), but where does it come from?

\( p \in \Delta \): probability distribution over choices

Expected score under \( p \): \( \mathbb{E}_{i \sim p} s_i = p^\top s \)

argmax maximizes expected score

Shannon entropy of \( p \): \( H(p) = -\sum_i p_i \log p_i \)

softmax maximizes expected score + entropy:

\[
\arg \max_{p \in \Delta} p^\top s + H(p)
\]
Variational form of softmax

**Proposition.** The unique solution to \( \arg \max_{p \in \Delta} p^T s + H(p) \) is given by \( p_j = \frac{\exp s_j}{\sum_i \exp s_i} \).
Variational form of softmax

**Proposition.** The unique solution to \( \arg \max_{p \in \Delta} p^T s + H(p) \) is given by \( p_j = \frac{\exp s_j}{\sum_i \exp s_i} \).

Explicit form of the optimization problem:

maximize \( \sum_j p_j s_j - p_j \log p_j \)

subject to \( p \geq 0, \ p^T 1 = 1 \)
Variational form of softmax

**Proposition.** The unique solution to \( \arg \max_{\mathbf{p} \in \Delta} \mathbf{p}^\top \mathbf{s} + H(\mathbf{p}) \) is given by \( p_j = \frac{\exp s_j}{\sum_i \exp s_i} \).

Explicit form of the optimization problem:

\[
\begin{align*}
\text{maximize} & \quad \sum_j p_j s_j - p_j \log p_j \\
\text{subject to} & \quad \mathbf{p} \geq 0, \quad \mathbf{p}^\top \mathbf{1} = 1
\end{align*}
\]

Lagrangian:

\[
\mathcal{L}(\mathbf{p}, \nu, \tau) = -\sum_j p_j s_j - p_j \log p_j - \mathbf{p}^\top \nu + \tau(\mathbf{p}^\top \mathbf{1} - 1)
\]
Variational form of softmax

**Proposition.** The unique solution to \( \arg \max_{p \in \Delta} p^T s + H(p) \) is given by \( p_j = \frac{\exp s_j}{\sum_i \exp s_i} \).

Explicit form of the optimization problem:

- maximize \( \sum_j p_j s_j - p_j \log p_j \)
- subject to \( p \geq 0, \ p^T 1 = 1 \)

Lagrangian:

\[
\mathcal{L}(p, \nu, \tau) = -\sum_j p_j s_j - p_j \log p_j - p^T \nu + \tau (p^T 1 - 1)
\]

Optimality conditions (KKT):

\[
0 = \nabla_{p_i} \mathcal{L}(p, \nu, \tau) = -s_i + \log p_i + 1 - \nu_i + \tau \\
p^T \nu = 0 \\
p \in \Delta \\
\nu \geq 0
\]
Variational form of softmax

**Proposition.** The unique solution to \( \arg \max_{p \in \Delta} p^T s + H(p) \) is given by \( p_j = \frac{\exp s_j}{\sum_i \exp s_i} \).

Explicit form of the optimization problem:

maximize \( \sum_j p_j s_j - p_j \log p_j \)

subject to \( p \geq 0, \ p^T 1 = 1 \)

Lagrangian:

\[
L(p, \nu, \tau) = -\sum_j p_j s_j - p_j \log p_j - p^T \nu + \tau (p^T 1 - 1)
\]

Optimality conditions (KKT):

\[
0 = \nabla_{p_i} L(p, \nu, \tau) = -s_i + \log p_i + 1 - \nu_i + \tau
\]

\[
p^T \nu = 0
\]

\[
p \in \Delta
\]

\[
\nu \geq 0
\]
Variational form of softmax

**Proposition.** The unique solution to \( \arg\max_{\mathbf{p} \in \Delta} \mathbf{p}^\top \mathbf{s} + H(\mathbf{p}) \) is given by \( p_j = \frac{\exp s_j}{\sum_i \exp s_i} \).

Explicit form of the optimization problem:

maximize \( \sum_j p_j s_j - p_j \log p_j \)

subject to \( \mathbf{p} \geq 0, \mathbf{p}^\top \mathbf{1} = 1 \)

Lagrangian:

\( \mathcal{L}(\mathbf{p}, \boldsymbol{\nu}, \tau) = -\sum_j p_j s_j - p_j \log p_j - \mathbf{p}^\top \boldsymbol{\nu} + \tau (\mathbf{p}^\top \mathbf{1} - 1) \)

Optimality conditions (KKT):

\[ 0 = \nabla_{p_i} \mathcal{L}(\mathbf{p}, \boldsymbol{\nu}, \tau) = -s_i + \log p_i + 1 - \nu_i + \tau \]

\[ \mathbf{p}^\top \boldsymbol{\nu} = 0 \]

\[ \mathbf{p} \in \Delta \]

\[ \boldsymbol{\nu} \geq 0 \]
**Variational form of softmax**

**Proposition.** The unique solution to \( \arg \max_{\mathbf{p} \in \Delta} \mathbf{p}^\top \mathbf{s} + H(\mathbf{p}) \) is given by 

\[
p_j = \frac{\exp s_j}{\sum_i \exp s_i}.
\]

Explicit form of the optimization problem:

maximize \( \sum_j p_j s_j - p_j \log p_j \)

subject to \( p \geq 0, \mathbf{p}^\top \mathbf{1} = 1 \)

Lagrangian:

\[
\mathcal{L}(\mathbf{p}, \mathbf{\nu}, \tau) = -\sum_i p_i s_i - p_j \log p_j - \mathbf{p}^\top \mathbf{\nu} + \tau (\mathbf{p}^\top \mathbf{1} - 1)
\]

Optimality conditions (KKT):

\[
0 = \nabla_{p_i} \mathcal{L}(\mathbf{p}, \mathbf{\nu}, \tau) = -s_i + \log p_i + 1 - \nu_i + \tau
\]

\[
\mathbf{p}^\top \mathbf{\nu} = 0
\]

\[
p \in \Delta
\]

\[
\nu \geq 0
\]
**Proposition.** The unique solution to \( \arg \max_{p \in \Delta} p^T s + H(p) \) is given by 
\[
p_j = \frac{\exp(s_j)}{\sum_i \exp(s_i)}.
\]

Explicit form of the optimization problem:

maximize \( \sum_j p_j s_j - p_j \log p_j \)

subject to \( p \geq 0, \ p^T 1 = 1 \)

Lagrangian:
\[
\mathcal{L}(p, \nu, \tau) = -\sum_j p_j s_j - p_j \log p_j - p^T \nu + \tau(p^T 1 - 1)
\]

Optimality conditions (KKT):

\[
0 = \nabla_{p_i} \mathcal{L}(p, \nu, \tau) = -s_i + \log p_i + 1 - \nu_i + \tau
\]

\[
p^T \nu = 0
\]

\[
p \in \Delta
\]

\[
\nu \geq 0
\]
**Variational form of softmax**

**Proposition.** The unique solution to \( \arg \max_{\mathbf{p} \in \Delta} \mathbf{p}^T \mathbf{s} + H(\mathbf{p}) \) is given by \( p_j = \frac{\exp s_j}{\sum_i \exp s_i} \).

Explicit form of the optimization problem:

maximize \( \sum_j p_j s_j - p_j \log p_j \)

subject to \( \mathbf{p} \geq 0, \mathbf{p}^T \mathbf{1} = 1 \)

Lagrangian:

\[
\mathcal{L}(\mathbf{p}, \mathbf{\nu}, \tau) = -\sum_j p_j s_j - p_j \log p_j - \mathbf{p}^T \mathbf{\nu} + \tau (\mathbf{p}^T \mathbf{1} - 1)
\]

Optimality conditions (KKT):

\[
0 = \nabla_{p_i} \mathcal{L}(\mathbf{p}, \mathbf{\nu}, \tau) = -s_i + \log p_i + 1 - \nu_i + \tau
\]

\[\mathbf{p}^T \mathbf{\nu} = 0\]

\( \mathbf{p} \in \Delta \)

\( \mathbf{\nu} \geq 0 \)

Log \( p_i = s_i + \nu_i - (\tau + 1) \)

if \( p_i = 0 \), r.h.s. must be \(-\infty\),
thus \( p_i > 0 \), so \( \nu_i = 0 \).

\( p_i = \frac{\exp(s_i)}{\exp(\tau+1)} = \frac{\exp(s_i)}{Z} \)

Must find \( Z \) such that \( \sum_j p_j = 1 \).
Answer: \( Z = \sum_j \exp(s_j) \)
### Variational form of softmax

**Proposition.** The unique solution to \( \arg \max_{p \in \Delta} p^T s + H(p) \) is given by \( p_j = \frac{\exp s_j}{\sum_i \exp s_i} \).

Explicit form of the optimization problem:
- maximize \( \sum_j p_j s_j - p_j \log p_j \)
- subject to \( p \geq 0, \ p^T 1 = 1 \)

Lagrangian:
\[
L(p, \nu, \tau) = -\sum_j p_j s_j - p_j \log p_j - p^T \nu + \tau(p^T 1 - 1)
\]

Optimality conditions (KKT):
- \( 0 = \nabla_{p_j} L(p, \nu, \tau) = -s_i + \log p_i + 1 - \nu_i + \tau \)
- \( p^T \nu = 0 \)
- \( p \in \Delta \)
- \( \nu \geq 0 \)

Log:
- \( \log p_i = s_i + \nu_i - (\tau + 1) \)
  - if \( p_i = 0 \), r.h.s. must be \(-\infty\),
  - thus \( p_i > 0 \), so \( \nu_i = 0 \).

For \( p_i = \frac{\exp(s_i)}{\exp(\tau+1)} = \frac{\exp(s_i)}{Z} \)

Must find \( Z \) such that \( \sum_j p_j = 1 \).

Answer: \( Z = \sum_j \exp(s_j) \)

So, \( p_i = \frac{\exp(s_i)}{\sum_j \exp(s_j)} \).

Classic result, e.g., [Boyd and Vandenberghe, 2004, Wainwright and Jordan, 2008]
Generalizing softmax: Smoothed argmaxes

\[ \hat{p}_\Omega(s) = \arg \max_{p \in \Delta} p^\top s - \Omega(p) \]
Generalizing softmax: Smoothed argmaxes

\[ \hat{p}_\Omega(s) = \arg \max_{p \in \Delta} p^\top s - \Omega(p) \]
Generalizing softmax: Smoothed argmaxes

\[ \hat{p}_\Omega(s) = \arg \max_{p \in \Delta} p^\top s - \Omega(p) \]

- argmax: \( \Omega(p) = 0 \)
Generalizing softmax: Smoothed argmaxes

\[ \hat{p}_\Omega(s) = \arg\max_{p \in \Delta} p^\top s - \Omega(p) \]

- argmax: \( \Omega(p) = 0 \)
- softmax: \( \Omega(p) = \sum_j p_j \log p_j \)
Generalizing softmax: Smoothed argmaxes

\[ \hat{p}_\Omega(s) = \arg \max_{p \in \Delta} p^\top s - \Omega(p) \]

- argmax: \( \Omega(p) = 0 \)
- softmax: \( \Omega(p) = \sum_j p_j \log p_j \)
- sparsemax: \( \Omega(p) = \frac{1}{2} \| p \|_2^2 \)

Niculae and Blondel, 2017, Martens and Astudillo, 2016
Generalizing softmax: Smoothed argmaxes

\[ \hat{p}_\Omega(s) = \arg \max_{p \in \Delta} p^\top s - \Omega(p) \]

- **argmax:** \( \Omega(p) = 0 \)
- **softmax:** \( \Omega(p) = \sum_j p_j \log p_j \)
- **sparsemax:** \( \Omega(p) = \frac{1}{2} \|p\|_2^2 \)
- **\( \alpha \)-entmax:** \( \Omega(p) = \frac{1}{\alpha(\alpha - 1)} \sum_j p_j^\alpha \)

Generalized entropy interpolates in between [Tsallis, 1988]

Used in Sparse Seq2Seq: [Peters et al., 2019] and Adaptively Sparse Transformers [Correia et al., 2019]
Generalizing softmax: Smoothed argmaxes

\[ \hat{p}_\Omega(s) = \arg \max_{p \in \Delta} p^\top s - \Omega(p) \]

- argmax: \( \Omega(p) = 0 \)
- softmax: \( \Omega(p) = \sum_j p_j \log p_j \)
- sparsemax: \( \Omega(p) = \frac{1}{2}\|p\|_2^2 \)
- \( \alpha \)-entmax: \( \Omega(p) = \frac{1}{\alpha(\alpha-1)} \sum_j p_j^\alpha \)
- fusedmax: \( \Omega(p) = \frac{1}{2}\|p\|_2^2 + \sum_j |p_j - p_{j-1}| \)
- csparsemax: \( \Omega(p) = \frac{1}{2}\|p\|_2^2 + \lambda(a \leq p \leq b) \)
- csoftmax: \( \Omega(p) = \sum_j p_j \log p_j + \lambda(a \leq p \leq b) \)
The structured case: Marginal polytope

- Each vertex corresponds to one such bit vector \( z \)
- Points inside correspond to marginal distributions: convex combinations of structured objects

\[
\mu = p_1 z_1 + \ldots + p_N z_N , \; p \in \Delta.
\]

\begin{align*}
p_1 &= 0.2, \quad z_1 = [1, 0, 0, 0, 1, 0, 0, 0, 1] \\
p_2 &= 0.7, \quad z_2 = [0, 0, 1, 0, 0, 1, 1, 0, 0] \\
p_3 &= 0.1, \quad z_3 = [1, 0, 0, 0, 1, 0, 0, 1, 0]
\end{align*}

\[\Rightarrow \mu = [0.3, 0.7, 0.3, 0.7, 0.7, 1, 0.2].\]
Niculae et al., 2018a

argmax

$\arg\max_p \Delta_p \top s$ so$[\max$

$\arg\max_p \Delta_p \top s + H(p)$

sparsemax

$\arg\max_p \Delta_p \top s - 1/2 \|p\|_2$

MAP

$\arg\max_\mu \Delta_\mu \top \eta$

marginals

$\arg\max_\mu \Delta_\mu \top \eta + e H(\mu)$

SparseMAP

$\arg\max_\mu \Delta_\mu \top \eta - 1/2 \|\mu\|_2$

Just like so$[\max$ relaxes argmax, marginals relax MAP differently!

Unlike argmax/sparsemax, computation is not obvious!
\[ \text{argmax} \quad \arg \max_{p \in \Delta} p^\top s \]

Unlike \text{argmax}/\text{so}[\max, \text{marginals relax MAP} \text{ diagramably}!\]

\[ \text{SparseMAP} \quad \arg \max_{\mu \in \mathcal{M}} \mu^\top \eta - \frac{1}{2} \|\mu\|_2^2 \]

\[ \text{MAP} \quad \arg \max_{\mu \in \mathcal{M}} \mu^\top \eta + e H(\mu) \]

\[ \Delta \]

\[ \mathcal{M} \]
\[
\text{argmax } \arg \max_{p \in \Delta} p^\top s
\]

\[
\text{MAP } \arg \max_{\mu \in \mathcal{M}} \mu^\top \eta
\]
- **argmax** \( \arg \max_{p \in \Delta} p^\top s \)
- **softmax** \( \arg \max_{p \in \Delta} p^\top s + H(p) \)

- **MAP** \( \arg \max_{\mu \in \mathcal{M}} \mu^\top \eta \)

Just like **softmax** relaxes **argmax**, marginals relax **MAP**! Unlike **argmax**, computation is not obvious!
- \text{argmax} \; \arg \max_{\mathbf{p} \in \Delta} \mathbf{p}^\top \mathbf{s} \\
- \text{softmax} \; \arg \max_{\mathbf{p} \in \Delta} \mathbf{p}^\top \mathbf{s} + H(\mathbf{p}) \\
- \text{MAP} \; \arg \max_{\mu \in \mathcal{M}} \mu^\top \eta \\
- \text{marginals} \; \arg \max_{\mu \in \mathcal{M}} \mu^\top \eta + \tilde{H}(\mu)
argmax $\arg \max_{\mathbf{p} \in \Delta} \mathbf{p}^\top \mathbf{s}$

softmax $\arg \max_{\mathbf{p} \in \Delta} \mathbf{p}^\top \mathbf{s} + H(\mathbf{p})$

MAP $\arg \max_{\mathbf{\mu} \in \mathcal{M}} \mathbf{\mu}^\top \mathbf{\eta}$

marginals $\arg \max_{\mathbf{\mu} \in \mathcal{M}} \mathbf{\mu}^\top \mathbf{\eta} + \tilde{H}(\mathbf{\mu})$

Just like softmax relaxes argmax, marginals relax MAP differentiably!
argmax $\arg\max_{p \in \Delta} p^T s$

MAP $\arg\max_{\mu \in M} \mu^T \eta$

softmax $\arg\max_{p \in \Delta} p^T s + H(p)$

marginals $\arg\max_{\mu \in M} \mu^T \eta + \tilde{H}(\mu)$

Just like softmax relaxes argmax, marginals relax MAP differentiably!

Unlike argmax/softmax, computation is not obvious!
## Algorithms for specific structures

<table>
<thead>
<tr>
<th>Structure Type</th>
<th>Best structure (MAP)</th>
<th>Marginals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assignments</td>
<td>Kuhn-Munkres [Kuhn, 1955, Jonker and Volgenant, 1987]</td>
<td></td>
</tr>
</tbody>
</table>
Algorithms for specific structures

Sequence tagging
- Best structure (MAP)
  - Viterbi [Rabiner, 1989]

Constituent trees
- CKY [Kasami, 1966, Younger, 1967]
- [Cocke and Schwartz, 1970]

Temporal alignments
- DTW [Sakoe and Chiba, 1978]

Dependency trees

Assignments
- Kuhn-Munkres [Kuhn, 1955, Jonker and Volgenant, 1987]
- #P-complete [Valiant, 1979, Taskar, 2004]

Marginals
- Forward-Backward [Rabiner, 1989]
- Inside-Outside [Baker, 1979]
- Soft-DTW [Cuturi and Blondel, 2017]
- Matrix-Tree [Kirchhoff, 1847]
Derivatives of marginals 1: DP

Dynamic programming: marginals by Forward-Backward, Inside-Outside, etc.

\[ \text{input: } d \text{ tags, } n \text{ tokens, } \eta \]

\[
\alpha_1 = 0, \quad \beta_n = 0
\]

\[
\text{for } i = 2, \ldots, n \text{ do}
\]

\[
\alpha_i, k = \log \sum_{k'} \exp \left( \alpha_{i-1} - k' + \eta_U^i - k + \eta_V^k \right)
\]

\[
\beta_i, k = \log \sum_{k'} \exp \left( \beta_{i+1} + k' + \eta_U^{i+1} + k' + \eta_V^k \right)
\]

\[
Z = \sum_k \exp \alpha_n, k
\]

\[ \text{return } \mu = \exp (\alpha + \beta - \log Z) \]

Marginals in a sequence tagging model.
Derivatives of marginals 1: DP

Dynamic programming: marginals by Forward-Backward, Inside-Outside, etc.

Marginals in a sequence tagging model.

1. input: $d$ tags, $n$ tokens, $\eta_U \in \mathbb{R}^{n \times d}$, $\eta_V \in \mathbb{R}^{d \times d}$
2. initialize $\alpha_1 = 0, \beta_n = 0$
3. for $i \in 2, \ldots, n$ do  
   # forward log-probabilities
   \[ \alpha_{i,k} = \log \sum_{k'} \exp(\alpha_{i-1,k'} + (\eta_U)_{i,k} + (\eta_V)_{k',k}) \]  
   for all $k$
4. for $i \in n-1, \ldots, 1$ do  
   # backward log-probabilities
   \[ \beta_{i,k} = \log \sum_{k'} \exp(\beta_{i+1,k'} + (\eta_U)_{i+1,k'} + (\eta_V)_{k,k'}) \]  
   for all $k$
5. $Z = \sum_k \exp \alpha_{n,k}$  
   # partition function
6. return $\mu = \exp(\alpha + \beta - \log Z)$  
   # marginals
Derivatives of marginals 1: DP

Dynamic programming: marginals by **Forward-Backward**, **Inside-Outside**, etc.

- Alg. consists of differentiable ops: PyTorch autograd can handle it! (v. bad idea)

---

Marginals in a sequence tagging model.

1. **Input**: $d$ tags, $n$ tokens, $\eta_U \in \mathbb{R}^{n \times d}$, $\eta_V \in \mathbb{R}^{d \times d}$
2. **Initialize** $\alpha_1 = 0, \beta_n = 0$
3. **For** $i \in 2, \ldots, n$ **do**
   - $\alpha_{i,k} = \log \sum_{k'} \exp \left( \alpha_{i-1,k'} + (\eta_U)_{i,k} + (\eta_V)_{k',k} \right)$  
     for all $k$
4. **For** $i \in n - 1, \ldots, 1$ **do**
   - $\beta_{i,k} = \log \sum_{k'} \exp \left( \beta_{i+1,k'} + (\eta_U)_{i+1,k'} + (\eta_V)_{k,k'} \right)$  
     for all $k$
5. $\mathbf{Z} = \sum_k \exp \alpha_{n,k}$  
   - # partition function
6. **Return** $\mu = \exp \left( \alpha + \beta - \log \mathbf{Z} \right)$  
   - # marginals
Derivatives of marginals 1: DP

**Dynamic programming:** marginals by **Forward-Backward**, **Inside-Outside**, etc.

- Alg. consists of differentiable ops: PyTorch autograd can handle it! (v. bad idea)
- Better book-keeping: Li and Eisner [2009], Mensch and Blondel [2018]

---

**Marginals in a sequence tagging model.**

1. input: \(d\) tags, \(n\) tokens, \(\eta_U \in \mathbb{R}^{n \times d}\), \(\eta_V \in \mathbb{R}^{d \times d}\)
2. initialize \(\alpha_1 = 0, \beta_n = 0\)
3. for \(i \in 2, \ldots, n\) do
   # forward log-probabilities
4.   \(\alpha_{i,k} = \log \sum_{k'} \exp (\alpha_{i-1,k'} + (\eta_U)_{i,k} + (\eta_V)_{k',k})\) for all \(k\)
5. for \(i \in n - 1, \ldots, 1\) do
   # backward log-probabilities
6.   \(\beta_{i,k} = \log \sum_{k'} \exp (\beta_{i+1,k'} + (\eta_U)_{i+1,k'} + (\eta_V)_{k,k'})\) for all \(k\)
7. \(Z = \sum_k \exp \alpha_{n,k}\)
   # partition function
8. return \(\mu = \exp (\alpha + \beta - \log Z)\)
   # marginals
Derivatives of marginals 1: DP

**Dynamic programming:** marginals by **Forward-Backward**, **Inside-Outside**, etc.

- Alg. consists of differentiable ops: PyTorch autograd can handle it! (v. bad idea)
- Better book-keeping: Li and Eisner [2009], Mensch and Blondel [2018]
- With circular dependencies, this breaks! Can get an approximation [Stoyanov et al., 2011]

**Marginals in a sequence tagging model.**

1. input: $d$ tags, $n$ tokens, $\eta_U \in \mathbb{R}^{n \times d}$, $\eta_V \in \mathbb{R}^{d \times d}$
2. initialize $\alpha_1 = 0$, $\beta_n = 0$
3. for $i \in 2, \ldots, n$ do
6. $\beta_i,k = \log \sum_{k'} \exp \left( \beta_{i+1,k'} + (\eta_U)_{i+1,k'} + (\eta_V)_{k,k'} \right)$ for all $k$
7. $Z = \sum_k \exp \alpha_{n,k}$
8. return $\mu = \exp (\alpha + \beta - \log Z)$
Derivatives of marginals 2: Matrix-Tree

$L(s)$: Laplacian of the edge score graph

\[ Z = \det L(s) \]
\[ \mu = L(s)^{-1} \]
\[ \nabla \mu = \nabla L^{-1} = L^{-1} \left( \frac{\partial L}{\partial \eta} \right) L^{-1} \]
Structured Attention Networks

\[ \eta, \mu \]

input \( x \) → \eta → \mu → output \( y \)

La coalition aide

Liu and Lapata, 2018

Kim et al., 2017
Structured Attention Networks

\[ \begin{align*}
\text{input} & \quad \eta & \quad \mu & \quad \text{output} \\
 x & & & y
\end{align*} \]

CRF marginals (from forward–backward) give a network weights \((0, 1)\).

Similar idea for projected dependency trees with inside–outside and non-projected with the Matrix-Tree theorem \cite{LiuLapata2018}.

\cite{Kimetal2017} deep-spin.github.io/tutorial
Structured Attention Networks

**Input** $x$

- $\eta(i)$: score of word $i$ receiving attention
- $\eta(i, i+1)$: score of consecutive words receiving attention

**Output** $y$

- $\mu(i)$: probability of word $i$ getting attention

$\eta$ and $\mu$ are functions that assign scores to words based on attention mechanisms. CRF marginals from forward-backward are used to compute attention weights.

Similar idea for projected dependency trees with inside-outside and non-projected with the Matrix-Tree Theorem ([Liu and Lapata, 2018](https://deep-spin.github.io/tutorial)).

[Kim et al., 2017]
Structured Attention Networks

\[ \eta(i): \text{score of word } i \text{ receiving attention} \]
\[ \eta(i, i+1): \text{score of consecutive words receiving attention} \]
\[ \mu(i): \text{probability of word } i \text{ getting attention} \]

CRF marginals (from forward-backward) give attention weights \(\in (0, 1)\)

[Kim et al., 2017]
Structured Attention Networks

$\eta$ (dog $\rightarrow$ on): arc score (tree constraints)

$\mu$ (dog $\rightarrow$ on): probability of arc

CRF marginals (from \textit{forward–backward}) give attention weights $\in (0, 1)$

Similar idea for projective dependency trees with \textit{inside–outside}

[Kim et al., 2017]
Structured Attention Networks

CRF marginals (from \textit{forward–backward}) give attention weights \( \in (0, 1) \)

Similar idea for projective dependency trees with \textit{inside–outside}

and non-projective with the Matrix-Tree theorem [Liu and Lapata, 2018].
Differentiable Perturb & Parse
Extending Gumbel-Softmax to structured stochastic models

- Forward pass:
  sample structure $z$ (approximately)
  
  $$z = \arg\max_{z \in Z} (\eta + \epsilon)^\top z$$

- Backward pass:
  pretend we did marginal inference
  
  $$\tilde{\mu} = \arg\max_{\mu \in \mathcal{M}} (\eta + \epsilon)^\top z + \tilde{H}(\mu)$$

(or some similar relaxation)
Back-propagating through marginals

Pros:

- Familiar algorithms for NLPers, (StructuredActionNetworks) all computations exact.
- Forward pass marginals are dense; (Fixed by Perturb & MAP, at cost of rough approximation)
- Efficient & numerically stable back-propagation through DPs is tricky; (somewhat alleviated by Mensch and Blondel [2018])
- Not applicable when marginals are unavailable.
- Case-by-case algorithms required, can get tedious.
Back-propagating through marginals

Pros:
- Familiar algorithms for NLPers,
Back-propagating through marginals

Pros:

- Familiar algorithms for NLPers,
- (Structured Attention Networks:) All computations exact.

Cons:

- Forward pass marginals are dense;
  - (Fixed by Perturb & MAP, at cost of rough approximation)
  - Efficient & numerically stable back-propagation through DPs is tricky;
    - (Somewhat alleviated by Menne and Blondel [2018])
- Not applicable when marginals are unavailable.
  - Case-by-case algorithms required, can get tedious.
Back-propagating through marginals

Pros:

- Familiar algorithms for NLPers,
- (Structured Attention Networks:) All computations exact.

Cons:

- (Structured Attention Networks:) forward pass marginals are dense; (fixed by Perturb & MAP, at cost of rough approximation)
Back-propagating through marginals

Pros:
- Familiar algorithms for NLPers,
- (Structured Attention Networks:) All computations exact.

Cons:
- (Structured Attention Networks:) forward pass marginals are dense;
  (fixed by Perturb & MAP, at cost of rough approximation)
- Efficient & numerically stable back-propagation through DPs is tricky;
  (somewhat alleviated by Mensch and Blondel [2018])
Back-propagating through marginals

Pros:
- Familiar algorithms for NLPers,
- (Structured Attention Networks:) All computations exact.

Cons:
- (Structured Attention Networks:) forward pass marginals are dense; (fixed by Perturb & MAP, at cost of rough approximation)
- Efficient & numerically stable back-propagation through DPs is tricky; (somewhat alleviated by Mensch and Blondel [2018])
- Not applicable when marginals are unavailable.
Back-propagating through marginals

Pro:\n- Familiar algorithms for NLPers,
- (Structured Attention Networks:) All computations exact.

Con:\n- (Structured Attention Networks:) forward pass marginals are dense; (fixed by Perturb & MAP, at cost of rough approximation)
- Efficient & numerically stable back-propagation through DPs is tricky; (somewhat alleviated by Mensch and Blondel [2018])
- Not applicable when marginals are unavailable.
- Case-by-case algorithms required, can get tedious.
Back-propagating through marginals

Pros:

- Familiar algorithms for NLPers,
- (Structured Attention Networks:) All computations exact.

Cons:

- (Structured Attention Networks:) forward pass marginals are dense; (fixed by Perturb & MAP approximation)
- Efficient & numerically stable back-propagation through DPs is tricky; (somewhat alleviated by 
- Not applicable when marginals are unavailable.
- Case-by-case algorithms required, can get tedious.
Back-propagating through marginals

Pros:
- Familiar algorithms for NLPers,
- (Structured Attention Networks:) All computations exact.

Cons:
- (Structured Attention Networks:) forward pass marginals are dense; (fixed by Perturb & MAP, at cost of rough approximation)
- Efficient & numerically stable back-propagation through DPs is tricky; (somewhat alleviated by Mensch and Blondel [2018])
- Not applicable when marginals are unavailable.
- Case-by-case algorithms required, can get tedious.
• argmax $\arg \max_{p \in \Delta} p^T s$

• softmax $\arg \max_{p \in \Delta} p^T s + H(p)$

• sparsemax $\arg \max_{p \in \Delta} p^T s - \frac{1}{2} \|p\|^2$

MAP $\arg \max_{\mu \in \mathcal{M}} \mu^T \eta$

marginals $\arg \max_{\mu \in \mathcal{M}} \mu^T \eta + \tilde{H}(\mu)$

[Niculae et al., 2018a]
- \textbf{argmax} \ \arg \max_{p \in \Delta} p^T s

- \textbf{softmax} \ \arg \max_{p \in \Delta} p^T s + H(p)

- \textbf{sparsemax} \ \arg \max_{p \in \Delta} p^T s - \frac{1}{2}\|p\|^2

\begin{align*}
\text{MAP} \ & \arg \max_{\mu \in \mathcal{M}} \mu^T \eta \\
\text{marginals} \ & \arg \max_{\mu \in \mathcal{M}} \mu^T \eta + \tilde{H}(\mu) \\
\text{SparseMAP} \ & \arg \max_{\mu \in \mathcal{M}} \mu^T \eta - \frac{1}{2}\|\mu\|^2
\end{align*}
SparseMAP solution

\[ \mu^* = \arg \max_{\mu \in \mathcal{M}} \mu^\top \eta - \frac{1}{2} \| \mu \|^2 \]

= \begin{array}{c}
\bullet \bullet \\
\bullet \\
\end{array} = 0.6 \begin{array}{c}
\bullet \bullet \\
\bullet \\
\end{array} + 0.4 \begin{array}{c}
\bullet \\
\end{array}

(\mu^* \text{ is unique, but may have multiple decompositions } p. \text{ Active Set recovers a sparse one.})
Algorithms for SparseMAP

$$\mu^* = \underset{\mu \in M}{\arg \max} \mu^T \eta - \frac{1}{2} \| \mu \|^2$$

This is also $\text{proj}_M$ required by SPIGOT!
Algorithms for SparseMAP

\[ \mu^* = \arg \max_{\mu \in \mathcal{M}} \mu^T \eta - \frac{1}{2} \| \mu \|^2 \]

Linear constraints (alias, exponentially many!)

Quadratic objective
Algorithms for SparseMAP

\[ \mu^* = \arg \max_{\mu \in \mathcal{M}} \mu^T \eta - \frac{1}{2} \| \mu \|^2 \]

linear constraints
(alas, exponentially many!)

quadratic objective

Conditional Gradient

[Frank and Wolfe, 1956, Lacoste-Julien and Jaggi, 2015]
Algorithms for SparseMAP

$$\mu^* = \arg \max \frac{1}{2} \| \mu \|^2$$

linear constraints
(alas, exponentially many!)

Conditioned Gradient

[Frank and Wolfe, 1956, Lacoste-Julien and Jaggi, 2015]

- select a new corner of $M$
Algorithms for SparseMAP

\[ \mu^* = \arg \max_{\mu \in \mathcal{M}} \mu^T \eta - \frac{1}{2} \|\mu\|^2 \]

linear constraints
(alas, exponentially many!)

Conditional Gradient

[Frank and Wolfe, 1956, Lacoste-Julien and Jaggi, 2015]

• select a new corner of \( \mathcal{M} \)
Algorithms for SparseMAP

\[ \mu^* = \arg \max_{\mu \in \mathcal{M}} \mu^\top \eta - \frac{1}{2} \| \mu \|^2 \]

linear constraints (alas, exponentially many!)

**Conditional Gradient**

[Frank and Wolfe, 1956, Lacoste-Julien and Jaggi, 2015]

- select a new corner of \( \mathcal{M} \)
- update the (sparse) coefficients of \( p \)
  - Update rules: vanilla, away-step, pairwise

[deep-spin.github.io/tutorial]
Algorithms for SparseMAP

$$\mu^* = \arg \max_{\mu \in \mathcal{M}} \mu^T \eta - \frac{1}{2} \|\mu\|^2$$

linear constraints (alas, exponentially many!)

Conditional Gradient

[Frank and Wolfe, 1956, Lacoste-Julien and Jaggi, 2015]

• select a new corner of $\mathcal{M}$
• update the (sparse) coefficients of $p$
  • Update rules: vanilla, away-step, pairwise
  • Quadratic objective: Active Set
    a.k.a. Min-Norm Point, [Wolfe, 1976]
    [Martins et al., 2015, Nocedal and Wright, 1999]
Algorithms for SparseMAP

\[ \mu^* = \arg \max_{\mu \in \mathcal{M}} \mu^T \eta - \frac{1}{2} \|\mu\|^2 \]

linear constraints (alas, exponentially many!)

quadratic objective

Conditional Gradient

[Frank and Wolfe, 1956, Lacoste-Julien and Jaggi, 2015]

- select a new corner
- update the (sparse)

Active Set achieves finite & linear convergence!

- Update rules: vanilla
- Quadratic objective:

\[ \text{Active Set achieves finite & linear convergence!} \]

a.k.a. Min-Norm Point, [Wolfe, 1976]

[ Martins et al., 2015, Nocedal and Wright, 1999]
Algorithms for SparseMAP

$$\mu^* = \arg \max_{\mu \in \mathcal{M}} \mu^T \eta - \frac{1}{2} \|\mu\|^2$$

Conditional Gradient

[Frank and Wolfe, 1956, Lacoste-Julien and Jaggi, 2015]

- select a new corner of \( \mathcal{M} \)
- update the (sparse) coefficients of \( p \)
  - Update rules: vanilla, away-step, pairwise
  - Quadratic objective: **Active Set**
    a.k.a. Min-Norm Point, [Wolfe, 1976]
    [Martins et al., 2015, Nocedal and Wright, 1999]

Backward pass

\( \frac{\partial \mu}{\partial \eta} \) is sparse
Algorithms for SparseMAP

\[ \mu^* = \arg \max_{\mu} \mu^T \eta - \frac{1}{2} \| \mu \|^2 \]

linear constraints
(alas, exponentially many!)

Conditional Gradient

[Frank and Wolfe, 1956, Lacoste-Julien and Jaggi, 2015]
- select a new corner of \( M \)
- update the (sparse) coefficients of \( p \)
  - Update rules: vanilla, away-step, pairwise
  - Quadratic objective: Active Set
    a.k.a. Min-Norm Point, [Wolfe, 1976]
    [Martins et al., 2015, Nocedal and Wright, 1999]

Backward pass

\[ \frac{\partial \mu}{\partial \eta} \text{ is sparse} \]
computing \( \left( \frac{\partial \mu}{\partial \eta} \right)^T dy \)
takes \( O(\dim(\mu) \cdot \text{nnz}(p^*)) \)
Algorithms for SparseMAP

\[ \mu^* = \arg \max_{\mu \in \mathcal{M}} \mu^T \eta - \frac{1}{2} \|\mu\|^2 \]

linear constraints
(alas, exponentially many!)

**Conditional Gradient**
[Frank and Wolfe, 1956, Lacoste-Julien and Jaggi, 2015]
- select a new corner of \( \mathcal{M} \)
- update the (sparse) coefficients of \( p \)
  - Update rules: vanilla, away-step, pairwise
  - Quadratic objective: **Active Set**
    a.k.a. Min-Norm Point, [Wolfe, 1976]
    [Martins et al., 2015, Nocedal and Wright, 1999]

Completely modular: just add MAP

Backward pass
\( \partial \mu / \partial \eta \) is sparse
computing \( \left( \frac{\partial \mu}{\partial \eta} \right)^T dy \)
takes \( O(\text{dim}(\mu) \cdot \text{nnz}(p^*)) \)

Deep-Spin.github.io/tutorial
a gentleman overlooking a neighborhood situation.

a police officer watches a situation closely.

a police officer watches a situation closely. a police officer watches a situation closely.
A police officer watches a situation closely.

A gentleman overlooking a neighborhood situation.
Overview

\[ \mathbb{E}_{\pi_{\theta}(z|x)}[L(z)] \quad \quad L(\arg\max_z \pi_{\theta}(z \mid x)) \quad \quad L(\mathbb{E}_{\pi_{\theta}(z|x)}[z]) \]

- REINFORCE
- Straight-Through Gumbel (Perturb & MAP)
- Straight-Through
- SPIGOT
- Structured Attn. Nets
- SparseMAP
Structured latent variables without sampling

\[ E_z[L(z)] = \sum_{z \in Z} L(\hat{y}(z)) \pi(z | x) \]
Structured latent variables without sampling

\[ \mathbb{E}_z[L(z)] = \sum_{z \in \mathcal{Z}} L(\hat{y}_\phi(z)) \pi(\theta(z | x)) \]
Structured latent variables without sampling

\[ \mathbb{E}_z[L(z)] = \sum_{z \in Z} L(\hat{y}_\phi(z)) \pi_\theta(z | x) \]

e.g., a TreeLSTM defined by \( z \)

How to define \( \pi_\theta(z) \)?

\[ \sum_h h^2 H \partial E_L(z) \partial \theta \]

\[ \pi_\theta(z) \propto \exp f_\theta(z) \]

\[ \text{argmax} \]

SparseMAP

\[ \text{e.g., a TreeLSTM defined by } z \]

\[ \text{Exponentially large sum!} \]

All methods we've seen require sampling; hard in general.
Structured latent variables without sampling

\[ \mathbb{E}_z[L(z)] = \sum_{z \in \mathcal{Z}} L(\hat{y}_\phi(z)) \pi_\theta(z | x) \]

- Example: a TreeLSTM defined by \( z \)
- Parsing model, using some scorer \( f_\theta(z; x) \)

All methods we've seen require sampling; hard in general.
Structured latent variables without sampling

\[ \mathbb{E}_z[L(z)] = \sum_{z \in \mathcal{Z}} L(\hat{y}_\phi(z)) \pi_\theta(z \mid x) \]

sum over all possible trees

e.g., a TreeLSTM defined by \( z \)

Exponentially large sum!

parsing model, using some scorer \( f_\theta(z; x) \)
Structured latent variables without sampling

\[ \mathbb{E}_z[L(z)] = \sum_{z \in Z} L(\hat{y}_\phi(z)) \pi_\theta(z \mid x) \]

How to define \( \pi_\theta \)?

- **Idea 1**: sum over all possible trees
- **Idea 2**: e.g., a TreeLSTM defined by \( z \)
- **Idea 3**: parsing model, using some scorer \( f_\theta(z; x) \)

sum over all possible trees

e.g., a TreeLSTM defined by \( z \)

parsing model, using some scorer \( f_\theta(z; x) \)
Structured latent variables without sampling

\[ \mathbb{E}_z[L(z)] = \sum_{z \in \mathcal{Z}} \left[ L(\hat{y}_\phi(z)) \right] \pi_\theta(z | x) \]

How to define \( \pi_\theta \)?

- **idea 1**
  - sum over all possible trees

- **idea 2**
  - e.g., a TreeLSTM defined by \( z \)

- **idea 3**
  - parsing model, using some scorer \( f_\theta(z; x) \)

\[ \sum_{h \in \mathcal{H}} \]

All methods we've seen require sampling, hard for general.
Structured latent variables without sampling

\[ \mathbb{E}_z[L(z)] = \sum_{z \in Z} L(\hat{y}_\phi(z)) \pi_\theta(z | x) \]

How to define \( \pi_\theta \)?

- Idea 1: sum over all possible trees
- Idea 2: parsing model, using some scorer \( f_\theta(z; x) \)
- Idea 3: exponentially large sum!

All methods we've seen require sampling; hard in general.

e.g., a TreeLSTM defined by \( z \)
Structured latent variables without sampling

\[ \mathbb{E}_z[L(z)] = \sum_{z \in Z} L(\hat{y}_\phi(z)) \pi_\theta(z \mid x) \]

How to define \( \pi_\theta \)?

- **idea 1**: \( \pi_\theta(z) \propto \exp(f_\theta(z)) \)
- **idea 2**: softmax
- **idea 3**

sum over all possible trees

\[ \sum_{h \in H} \frac{\partial \mathbb{E}[L(z)]}{\partial \theta} \]

e.g., a TreeLSTM defined by \( z \)

parsing model, using some scorer \( f_\theta(z; x) \)

Exponentially large sum!
Structured latent variables without sampling

\[ \mathbb{E}_z[L(z)] = \sum_{z \in \mathcal{Z}} L(\hat{y}_\phi(z)) \pi_\theta(z \mid x) \]

How to define \( \pi_\theta \)?

- idea 1: \( \pi_\theta(z) \propto \exp(f_\theta(z)) \)
- idea 2:
- idea 3: parsing model, using some scorer \( f_\theta(z; x) \)

sum over all possible trees

\[ \sum_{h \in \mathcal{H}} \frac{\partial \mathbb{E}[L(z)]}{\partial \theta} \]

e.g., a TreeLSTM defined by \( z \)

softmax

\text{Exponentially large sum! \text{All methods we've seen require sampling; hard in general.}}
Structured latent variables without sampling

\[ E_z[L(z)] = \sum_{z \in Z} L(\hat{y}_\phi(z)) \pi_\theta(z \mid x) \]

How to define \( \pi_\theta \)?

idea 1 \( \pi_\theta(z) \propto \exp(f_\theta(z)) \)

idea 2

idea 3

softmax

\[ \sum_{h \in H} \frac{\partial E[L(z)]}{\partial \theta} \]
Structured latent variables without sampling

\[
\mathbb{E}_z[L(z)] = \sum_{z \in Z} L(\hat{y}_\phi(z)) \pi_\theta(z \mid x)
\]

How to define \(\pi_\theta\)?

All methods we’ve seen require sampling; hard in general.

idea 2

idea 3
Structured latent variables without sampling

\[
\mathbb{E}_z[L(z)] = \sum_{z \in \mathcal{Z}} L(\hat{y}_\phi(z)) \pi_\theta(z | x)
\]

How to define \( \pi_\theta \)?

- Idea 1: \( \pi_\theta(z) \propto \exp(f_\theta(z)) \)
- Idea 2: \( \pi_\theta(z) = 1 \) if \( z = \text{MAP}(f_\theta(\cdot)) \) else 0
- Idea 3: 

\[
\sum_{h \in H} \frac{\partial \mathbb{E}[L(z)]}{\partial \theta}
\]

e.g., a TreeLSTM defined by \( z \)

SparseMAP, e.g., a TreeLSTM defined by \( z \)

sum over all possible trees

deep-spin.github.io/tutorial
Structured latent variables without sampling

\[ \mathbb{E}_z[L(z)] = \sum_{z \in \mathcal{Z}} L(\hat{y}_\phi(z)) \pi_\theta(z | x) \]

How to define \( \pi_\theta \)?

- **Idea 1**: \( \pi_\theta(z) \propto \exp(f_\theta(z)) \)
- **Idea 2**: \( \pi_\theta(z) = 1 \) if \( z = \text{MAP}(f_\theta(\cdot)) \) else 0
- **Idea 3**: \( \pi_\theta(z) = \text{argmax} \)

E.g., a TreeLSTM defined by \( z \)
Structured latent variables without sampling

\[ \mathbb{E}_z[L(z)] = \sum_{z \in \mathcal{Z}} L(\hat{y}_\phi(z)) \pi_\theta(z | x) \]

How to define \( \pi_\theta \)?

- **idea 1**: \( \pi_\theta(z) \propto \exp(f_\theta(z)) \)
- **idea 2**: \( \pi_\theta(z) = 1 \) if \( z = \text{MAP}(f_\theta(\cdot)) \) else 0
- **idea 3**: \( \pi_\theta(z) \)

\[ \sum_{h \in \mathcal{H}} \frac{\partial \mathbb{E}[L(z)]}{\partial \theta} \]

e.g., a TreeLSTM defined by \( z \)

sum over all possible trees

parsing model, using some scorer \( f_\theta(z; x) \)

Deep Spin's tutorial: deep-spin.github.io/tutorial
Structured latent variables without sampling

\( \mathbb{E}_z[L(z)] = \sum_{z \in \mathcal{Z}} L(\hat{y}_\phi(z)) \pi_\theta(z | x) \)

**How to define \( \pi_\theta \)?

- **idea 1** \( \pi_\theta(z) \propto \exp(f_\theta(z)) \)
- **idea 2** \( \pi_\theta(z) = 1 \) if \( z = \text{MAP}(f_\theta(\cdot)) \) else 0
- **idea 3** SparseMAP

\[ \sum_{h \in \mathcal{H}} \frac{\partial \mathbb{E}_z[L(z)]}{\partial \theta} \]

- **softmax** 😱 😊
- **argmax** 😊 😞 😊
- **SparseMAP** 😊 😊
Structured latent variables without sampling

\[
\begin{align*}
\begin{array}{c}
\end{align*}
\end{array}
\]
Structured latent variables without sampling

\[ E[L(z)] = .7 \times L(\cdot \cdot \cdot) + .3 \times L(\cdot \cdot \cdot) + 0 \times \cdot \cdot \cdot + \ldots \]
Structured latent variables without sampling

\[ \mathbb{E}[L(z)] = 0.7 \times L(\cdot) + 0.3 \times L(\cdot) + \ldots \]

recall our shorthand \( L(z) = L(\hat{\phi}(z), y) \)
V. Conclusions
<table>
<thead>
<tr>
<th>Stanford Sentiment (Accuracy)</th>
<th>Stanford Natural Language Inference (Accuracy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Socher et al., 2013]</td>
<td>[Kim et al., 2017]</td>
</tr>
<tr>
<td>Bigram Naive Bayes</td>
<td>Simple Attention 86.2</td>
</tr>
<tr>
<td>[Niculae et al., 2018b]</td>
<td>Structured Attention 86.8</td>
</tr>
<tr>
<td>DepTreeLSTM w/ CoreNLP</td>
<td>[Liu and Lapata, 2018]</td>
</tr>
<tr>
<td>83.2</td>
<td>100D Structured Attention 86.8</td>
</tr>
<tr>
<td>DepTreeLSTM w/ SparseMAP</td>
<td>[Yogatama et al., 2017]</td>
</tr>
<tr>
<td>84.7</td>
<td>100D RL-SPINN 80.5</td>
</tr>
<tr>
<td>[Corro and Titov, 2019b]</td>
<td>[Choi et al., 2018]</td>
</tr>
<tr>
<td>GCN w/ CoreNLP</td>
<td>100D ST Gumbel-Tree 82.6</td>
</tr>
<tr>
<td>83.8</td>
<td>300D - 85.6</td>
</tr>
<tr>
<td>GCN w/ Perturb-and-MAP</td>
<td>600D - 86.0</td>
</tr>
<tr>
<td>84.6</td>
<td></td>
</tr>
<tr>
<td>[Choi et al., 2018]</td>
<td>[Corro and Titov, 2019b]</td>
</tr>
<tr>
<td>ST Gumbel-Tree</td>
<td>Latent Tree + 1 GCN - 85.2</td>
</tr>
<tr>
<td>90.7</td>
<td>Latent Tree + 2 GCN - 86.2</td>
</tr>
<tr>
<td>[Havrylov et al., 2019]</td>
<td>[Havrylov et al., 2019]</td>
</tr>
<tr>
<td>TreeLSTM + tricks</td>
<td>100D TreeLSTM + tricks 84.3</td>
</tr>
<tr>
<td>90.2</td>
<td></td>
</tr>
</tbody>
</table>
Is it syntax?! 

- Unlike e.g. unsupervised parsing, the structures we learn are guided by a **downstream objective** (typically discriminative).
- They don’t typically resemble grammatical structure (yet) [Williams et al., 2018] (future work: more inductive biases and constraints?)
Is it syntax?! 

- Unlike e.g. unsupervised parsing, the structures we learn are guided by a **downstream objective** (typically discriminative).
- They don’t typically resemble grammatical structure (yet) [Williams et al., 2018] (future work: more inductive biases and constraints?)
- Common to compare latent structures with parser outputs. But is this always a meaningful comparison?
Syntax vs. Composition Order

CoreNLP parse, $p = 21.4\%$

★ lovely and poignant.
Syntax vs. Composition Order

$p = 22.6\%$

★ lovely and poignant .

CoreNLP parse,  $p = 21.4\%$

★ lovely and poignant .

⋯
Syntax vs. Composition Order

Niculae et al., 2018b

$p = 22.6\%$

$\star$ lovely and poignant.

CoreNLP parse, $p = 21.4\%$

$p = 15.33\%$

$\star$ a deep and meaningful film.

$p = 15.27\%$

$\star$ a deep and meaningful film.

CoreNLP parse, $p = 0\%$

$\star$ a deep and meaningful film.
Conclusions

- Latent structure models are desirable for interpretability, structural bias, and higher predictive power with fewer parameters.
- Stochastic latent variables can be dealt with RL or straight-through gradients.
- Deterministic argmax requires surrogate gradients (e.g. SPIGOT).
- Continuous relaxations of argmax include SANs and SparseMAP.
- Intuitively, some of these different methods are trying to do similar things or require the same building blocks (e.g. SPIGOT and SparseMAP).
- ... we didn't even get into deep generative models! These tools apply, but there are new challenges. [Corro and Titov, 2019a, Kim et al., 2019a,b, Kawakami et al., 2019]
Overview

\[
\mathbb{E}_{\pi_{\theta}(z|x)}[L(z)] \quad L(\arg \max_z \pi_{\theta}(z \mid x)) \quad L(\mathbb{E}_{\pi_{\theta}(z|x)}[z])
\]

- REINFORCE
- Straight-Through
- Straight-Through Gumbel (Perturb & MAP)
- SPIGOT
- Structured Attn. Nets
- SparseMAP
- SparseMAP

And more, in the next section! Thank you!
Overview

$\mathbb{E}_{\pi_\theta(z|x)}[L(z)]$  
$L(\arg\max_z \pi_\theta(z \mid x))$  
$L(\mathbb{E}_{\pi_\theta(z|x)}[z])$

- REINFORCE$^{SPL}$
- Straight-Through Gumbel (Perturb & MAP)$^{SPL,MRG}$
- SparseMAP$^{MAP+}$
- Straight-Through$^{MAP,MRG}$
- SPIGOT$^{MAP+}$
- Structured Attn. Nets$^{MRG}$
- SparseMAP$^{MAP+}$

Computation:

SPL: Sampling. (Simple in incremental/unstructured, hard for most global structures.)

MAP: Finding the highest-scoring structure.

MRG: Marginal inference.
Overview

\[ L\left( \arg \max_z \pi_\theta(z \mid x) \right) \]

\[ L\left( \mathbb{E}_{\pi_\theta(z \mid x)}[z] \right) \]

- REINFORCE\textsuperscript{SPL}
- Straight-Through Gumbel (Perturb & MAP)\textsuperscript{SPL,MRG}
- SparseMAP\textsuperscript{MAP+}
- Straight-Through\textsuperscript{MAP,MRG}
- SPIGOT\textsuperscript{MAP+}
- Structured Attn. Nets\textsuperscript{MRG}
- SparseMAP\textsuperscript{MAP+}

Computation:

\textit{SPL: Sampling. (Simple in incremental/unstructured, hard for most global structures.)}

\textit{MAP: Finding the highest-scoring structure.}

\textit{MRG: Marginal inference.}

Thank you!
References I


References II


References III


References VII


