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

Dependency parsing

Exponentially many parse trees!
Cannot enumerate.
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**

- ... (diagram with arrows)
- dog on wheels

**Word alignments**

- dog on wheels
- hond on wielen
- dog on wheels
- hond on wielen
- dog on wheels
- hond on wielen
- dog on wheels
- hond on wielen

Exponentially many parse trees!

Cannot enumerate.

deep-spin.github.io/tutorial
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*! 🙌
• Uncover hidden representations useful for the *downstream task*.
• Neural networks are well-suited for this: *deep computation graphs*.
• Uncover hidden representations useful for the *downstream task*.  
• Neural networks are well-suited for this: *deep computation graphs*.  
• Neural representations are unstructured, inescrutable.  
  Language data has underlying structure!
Latent structure models

• Seek *structured* hidden representations instead!
Latent structure models

- Seek *structured* hidden representations instead!
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!
Why do we love latent structure models?

• The inferred latent variables can bring us some **interpretability**
• They offer a way of injecting prior knowledge as a **structured bias**
• Hopefully: Higher predictive power with fewer model parameters
Why do we love latent structure models?

- The inferred latent variables can bring us some **interpretability**
- They offer a way of injecting prior knowledge as a **structured bias**
- Hopefully: Higher predictive power with fewer model parameters
  - smaller carbon footprint!
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)
Background
To better explain the math, we'll often backtrack to *unstructured* models (where the latent variable is a categorical) before jumping to the *structured* ones.
The unstructured case: Probability simplex

Each vertex is an indicator vector, representing one class: $z_c = [0, \ldots, 0, 1 | \{z\}^c_{th\ position}, 0, \ldots, 0]$.

Points inside are probability vectors, a convex combination of classes: $p_0, \sum_c p_c = 1$. 

(deep-spin.github.io/tutorial)
The unstructured case: Probability simplex

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

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

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

- 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 as 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!

  \[z_1 = [1, 0, 0, 0, 1, 0, 0, 1]\]
  \[z_2 = [0, 0, 1, 0, 0, 1, 1, 0]\]
  \[z_3 = [1, 0, 0, 0, 1, 0, 0, 1, 0]\]
The structured case: Marginal polytope

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

\[ \mu = p_1 z_1 + \ldots + p_N z_N \mid \{z\} \text{exponentially many terms} \]

\[ p_1 = 0.2, z_1 = [1, 0, 0, 0, 1, 0, 0, 0, 1] \]

\[ p_2 = 0.7, z_2 = [0, 0, 1, 0, 0, 1, 1, 0, 0] \]

\[ p_3 = 0.1, z_3 = [1, 0, 0, 0, 1, 0, 0, 1, 0] \]

\[ \mu = [0.3, 0, 0.7, 0, 0.3, 0.7, 0.7, 0.1, 0.2] \]
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

$$\mu = p_1 z_1 + \ldots + p_N z_N \quad , \quad p \in \Delta.$$  
(exponentially many terms)

$p_1 = 0.2, \quad z_1 = [1, 0, 0, 0, 1, 0, 0, 0, 1]$

$p_2 = 0.7, \quad z_2 = [0, 0, 1, 0, 0, 1, 1, 0, 0] \quad \Rightarrow \quad \mu = [0.3, 0.7, 0.3, 0.7, 0.7, 0.1, 0.2]$

$p_3 = 0.1, \quad z_3 = [1, 0, 0, 0, 1, 0, 0, 1, 0]$
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 $\mathcal{M}$
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 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.

<table>
<thead>
<tr>
<th>$s$</th>
<th>$z$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$z = 1$</td>
</tr>
<tr>
<td></td>
<td>$z = 2$</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>$z = N$</td>
</tr>
</tbody>
</table>
The challenge of discrete choices.

input \( x \)

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

\( z \)

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

output \( \hat{y} \)

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

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

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

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

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

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

\[ \frac{\partial L(\hat{y}, y)}{\partial w} = ? \quad \text{or, essentially,} \quad \frac{\partial z}{\partial s} = ? \]
Discrete mappings are “flat”

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

\[ s \]

<table>
<thead>
<tr>
<th>( z )</th>
<th>( \frac{\partial z}{\partial s} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( z = 1 )</td>
<td>?</td>
</tr>
<tr>
<td>( z = 2 )</td>
<td>?</td>
</tr>
<tr>
<td>( z = N )</td>
<td>?</td>
</tr>
</tbody>
</table>
Discrete mappings are “flat”

\[
s \quad z \quad s \quad \frac{\partial z}{\partial s} = ?
\]

\[
\begin{align*}
\text{s} & \quad \text{z} \\
\text{z = 1} & \quad \text{z = 2} \\
\text{z = 2} & \\
\text{...} & \\
\text{z = N} &
\end{align*}
\]
Discrete mappings are “flat”

\[ s \]

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

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

\[
\begin{array}{c|c}
 s & z \\
 \hline
 z = 1 & \text{ } \\
 z = 2 & \text{ } \\
 \ldots & \text{ } \\
 z = N & \text{ } \\
\end{array}
\]

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

<table>
<thead>
<tr>
<th>$s$</th>
<th>$z$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$z = 1$</td>
</tr>
<tr>
<td></td>
<td>$z = 2$</td>
</tr>
<tr>
<td></td>
<td>$\ldots$</td>
</tr>
<tr>
<td></td>
<td>$z = N$</td>
</tr>
</tbody>
</table>

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

\[
s \quad z = 1 \\
\quad z = 2 \\
\quad \ldots \\
\quad z = N \\
\frac{\partial z}{\partial s} = ?
\]
Discrete mappings are “flat”

<table>
<thead>
<tr>
<th>$s$</th>
<th>$z$</th>
<th>$\frac{\partial z}{\partial s}$ = ?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$z = 1$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$z = 2$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\ldots$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$z = N$</td>
<td></td>
</tr>
</tbody>
</table>
Argmax

\[
\begin{align*}
  \frac{\partial z}{\partial s} &= 0 \\
  z &= 1 \\
  z &= 2 \\
  \ldots \\
  z &= N
\end{align*}
\]
Example: Regression with latent categorization

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

\[ s = W_s u \]

Workarounds: circumventing the issue, bypassing discrete variables

Tackling discreteness end-to-end
Example: Regression with latent categorization

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

\[ s = W_s u \]

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

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

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

Workarounds: circumventing the issue, bypassing discrete variables, tackling discreteness end-to-end.
Example: Regression with latent categorization

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

\[ s = W_s u \]

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

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

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

\[ \frac{\partial L}{\partial W_s} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial v} \frac{\partial v}{\partial z} \frac{\partial z}{\partial s} \frac{\partial s}{\partial W_s} \]

---

**Workarounds:**
- Circumventing the issue
- Bypassing discrete variables
- Tackling discreteness end-to-end

---

deeppin.github.io/tutorial
Example: Regression with latent categorization

$$u = \frac{1}{|x|} \sum_j E_j$$

Workarounds: circumventing the issue, bypassing discrete variables

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

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

$s = W_s u$

$v = \text{tanh} (W_v [u, z])$

$\hat{y} = W_y v$

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

Option 1. Pretrain latent classifier $W_s$
Example: Regression with latent categorization

Option 2. Multi-task learning

Input $x$ 

Embeddings $E$

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

$E_s$ 

$$s = W_s u$$

$W_s$ 

$W_v$ 

$W_y$

Output $\hat{y}$

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

$$\hat{y} = W_y v$$

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

Workarounds: circumventing the issue, bypassing discrete variables

Tackling discreteness end-to-end
Example: Regression with latent categorization

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

Tackling discreteness end-to-end

Output \( \hat{y} \)

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

\[
\begin{align*}
\text{input } x & \quad u & \quad W_s & \quad S & \quad Z & \quad v & \quad \text{output } \hat{y} \\
\text{embeddings } E & \Rightarrow & W_s & \Rightarrow & (c) & \Rightarrow & W_v & \Rightarrow \text{output } \hat{y} \\
\end{align*}
\]

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

Option 3. Stochasticity! \[
\frac{\partial E_z (\hat{y}(z) - y)^2}{\partial W_s} \neq 0
\]
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 4. Gradient surrogates (e.g. straight-through, \( \frac{\partial z}{\partial s} \leftarrow I \))
Example: Regression with latent categorization

Option 5. Continuous relaxation (e.g. softmax)

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

\[
u = W_s u
\]

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

\[
\hat{y} = W_y v \quad L = (\hat{y} - y)^2
\]
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
- Part 3: Gradient surrogates

Coffee Break

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

- Given a sentence-label pair \((x, y)\) and its **known** parse tree \(z\),

\[ \text{we can make a prediction } ^\ast y(z; x), \]

\[ \text{and incur a loss, } L\left(^\ast y(z; x), y\right) \text{ or simply } L(z) \]

But we don't know \(z\)!

In this section: we jointly learn a structured prediction model \(\pi_{\theta}(z|x)\) by optimizing the expected loss, 

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

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

\[ L(\hat{y}(z; x), y) \] 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 \textbf{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 \(\pi_\theta(z | x)\) by optimizing the \textbf{expected loss},

\[ \mathbb{E}_{\pi_\theta(z|x)} [L(z)] \]
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 shift-reduce parser. It looks at top two elements of stack and top element of the buffer.
- TreeLSTM combines top two elements of the stack when the parser choses the 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]

deep-spin.github.io/tutorial
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, $z_j \in \{0, 1\} \ \forall j \in [1, 2L - 1]$
A sequence of Bernoulli trials but with conditional dependence,

\[ p(z_1, z_2, \ldots, z_{2L-1}) = \prod_{j=1}^{2L-1} p(z_j | z_{<j}) \]
Latent structure learning with SPINN

But now, removes syntactic supervision from SPINN. We model the parse, $z$, as a latent variable with our parser as the score function, $f_{\theta}(x)$.

With shift-reduce parsing, we're making discrete decisions. REINFORCE as a "natural" solution.
Latent structure learning with SPINN

• But now, remove syntactic supervision from SPINN.

[Diagram showing the process of parsing with TreeLSTM and REINFORCE for natural solutions]
Latent structure learning with SPINN

- But now, remove syntactic supervision from SPINN.

- We model the parse, $z$, as a latent variable with our parser as the score function estimator, $f_\theta(x)$. 
Latent structure learning with SPINN

- But now, remove syntactic supervision from SPINN.

- We model the parse, \( z \), as a latent variable with our parser as the score function estimator, \( f_\theta(x) \).

- With shift-reduce parsing, we’re making discrete decisions \( \Rightarrow \) 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.
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

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

(By definition of expectation. How to evaluate?)
Through the looking glass of REINFORCE

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

(By definition of expectation. How to evaluate?)

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

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

(By definition of expectation. How to evaluate?)

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

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

(By Leibniz integral rule for log)
Through the looking glass of REINFORCE

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

(By definition of expectation. How to evaluate?)

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

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

(By Leibniz integral rule for \log)

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

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

[Supply your notes]

### Table: Performance of Different Models

<table>
<thead>
<tr>
<th>Model</th>
<th>μ(σ) max</th>
<th>Self F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTM</td>
<td>71.5 (1.5)</td>
<td>74.4 -</td>
</tr>
<tr>
<td>RL-SPINN</td>
<td>60.7 (2.6)</td>
<td>64.8 30.8</td>
</tr>
<tr>
<td>Random Trees</td>
<td>-</td>
<td>- 30.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>F1 wrt. Avg. Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>48D RL-SPINN</td>
<td>64.5 16.0 32.1 14.6</td>
</tr>
<tr>
<td>128D RL-SPINN</td>
<td>43.5 13.0 71.1 10.4</td>
</tr>
<tr>
<td>GT Trees</td>
<td>41.6 8.8 100.0 9.6</td>
</tr>
<tr>
<td>Random Trees</td>
<td>24.0 24.0 24.2 5.2</td>
</tr>
</tbody>
</table>

Source: Nangia and Bowman, 2018

But why?

[deep-spin.github.io/tutorial: Supply your notes]
## Toy problem: ListOps

### Accuracy Self Model $\mu(\sigma)$ $\mu(\sigma)$ $\mu(\sigma)$ max F1

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
<th>Self F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTM</td>
<td>71.5 (1.5)</td>
<td>74.4</td>
</tr>
<tr>
<td>RL-SPINN</td>
<td>60.7 (2.6)</td>
<td>64.8</td>
</tr>
<tr>
<td>Random Trees</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

### F1 wrt. Avg. Model LB RB GT Depth

<table>
<thead>
<tr>
<th>Model</th>
<th>F1 wrt. Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTM</td>
<td>48.0</td>
</tr>
<tr>
<td>RL-SPINN</td>
<td>46.5</td>
</tr>
<tr>
<td>GT Trees</td>
<td>41.6</td>
</tr>
<tr>
<td>Random Trees</td>
<td>24.0</td>
</tr>
</tbody>
</table>

**But why?**

- 128D RL-SPINN
  - F1: 43.5
  - RB: 13.0
  - GT: 71.1
  - Avg. Depth: 10.4

- GT Trees
  - F1: 41.6
  - RB: 8.8
  - GT: 100.0
  - Avg. Depth: 9.6

- Random Trees
  - F1: 24.0
  - RB: 24.0
  - GT: 24.2
  - Avg. Depth: 5.2

[39]

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

This system faces at least two big problems,
RL-SPINN’s Troubles

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! **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.

- 3 tokens ⇒ 5 trees
- 5 tokens ⇒ 42 trees
- 10 tokens ⇒ 16796 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.
- 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,

$$\nabla \mathbb{E}_{z \sim \pi(z)} = \mathbb{E}_{z \sim \pi(z)} [(L(z) - b(x)) \nabla \pi(z)]$$
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,

$$\nabla \mathbb{E}_{z \sim \pi(z)} = \mathbb{E}_{z \sim \pi(z)}[(L(z) - b(x)) \nabla \pi(z)]$$

Which we can do because,

$$\sum_z b(x) \nabla \pi(z) = b(x) \sum_z \nabla \pi(z) = b(x) \nabla 1 = 0$$
Issues with SPINN with REINFORCE

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 $\theta$. 
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 hard to explore the parsing search space and optimize for $\theta$.

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.
III. Gradient Surrogates
So far:

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

\[ E_{πθ(z|x)}[L(z)] \]

In this section:

Consider the deterministic alternative:

pick "best" structure \( z^* \text{\( x \) = arg max \( z \in \mathcal{M} \) \( \pi_θ(z | x) \) } \)

\[ L(z^*) \]

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.

---

In this section:
- Consider the deterministic alternative:
  - Pick "best" structure \( z(x) = \arg \max \pi_{\theta}(z|x) \)
  - Incur loss \( L(z) \)

**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**

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

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

  - try to optimize the deterministic loss directly
  - 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.

In this section:

- Consider the **deterministic alternative**:
  pick “best” structure
  \[
  \hat{z}(x) := \arg \max_{z \in \mathcal{M}} \pi_{\theta}(z | x)
  \]
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**:
  \[ \hat{z}(x) := \arg\max_{z \in M} \pi_\theta(z \mid x) \]
  \[ L(\hat{z}(x)) \]
  - pick "best" structure
  - incur loss

[deep-spin.github.io/tutorial]
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 “best” structure \( \hat{z}(x) := \text{arg max}_{z \in M} \pi_{\theta}(z \mid x) \)
  - 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**: pick "best" structure

\[ \hat{z}(x) := \arg \max_{z \in \mathcal{M}} \pi_\theta(z \mid x) \]

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

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

\[ \frac{\partial z}{\partial s} = 0 \]
$p_j = \exp(s_j)/Z$

$$\begin{align*}
s &= [z = 1, z = 2, \ldots, z = N] \\
p &= [\frac{\partial p}{\partial s} = \text{diag}(p) - pp^\top]
\end{align*}$$
Straight-Through Estimator

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

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

More explanation in a while

[Hinton, 2012, Bengio et al., 2013]
• **Forward:** $z = \text{arg max}(s)$

[Note on diagram: $s$ and $z$]

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

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

[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 \( \tilde{p}(s) = \text{softmax}(s), \frac{\partial \tilde{p}}{\partial s} = \text{diag}(\tilde{p}) - \tilde{p}\tilde{p}^T \)
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:** \( z = \text{arg max}(s) \)
- **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 in or forward 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)
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

Each vertex corresponds to one such bit vector $z$. Points inside correspond to marginal distributions:

$$\mu = p_1 z_1 + \ldots + p_N z_N \quad |\{z\}$$

Exponentially many terms.

- $p_1 = 0.2$, $z_1 = [1, 0, 0, 0, 1, 0, 0, 0, 1]$.
- $p_2 = 0.7$, $z_2 = [0, 0, 1, 0, 0, 1, 1, 0, 0]$.
- $p_3 = 0.1$, $z_3 = [1, 0, 0, 0, 1, 0, 0, 0, 1]$. 

$\mu = [0.3, 0, 0.7, 0, 0.3, 0.7, 0.7, 0.1, 0.2]$. 

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

- Each vertex corresponds to one such *bit vector* $z$

[Wainwright and Jordan, 2008]
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

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

exponentially many terms

$p_1 = 0.2, \quad \mathbf{z}_1 = [1, 0, 0, 0, 1, 0, 0, 0, 1]$

$p_2 = 0.7, \quad \mathbf{z}_2 = [0, 0, 1, 0, 0, 1, 1, 0, 0]$  \Rightarrow  \mu = [0.3, 0.7, 0.3, 0.7, 0.7, 1, 0.2]$
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

Best structure (MAP)

Sequence tagging
- Viterbi
  [Rabiner, 1989]
- CKY
  [Kasami, 1966, Younger, 1967]
  [Cocke and Schwartz, 1970]

Constituent trees
- DTW
  [Sakoe and Chiba, 1978]

Temporal alignments
- Max. Spanning Arborescence
  [Chu and Liu, 1965, Edmonds, 1967]

Dependency trees
- Kuhn-Munkres
  [Kuhn, 1955, Jonker and Volgenant, 1987]

Assignments
Structured Straight-Through

- **Forward pass:**
  Find highest-scoring structure:
  \[ z = \arg \max_{z \in \mathcal{Z}} \eta^T z \]

- **Backward pass:**
  pretend we used \( \tilde{\mu} = \eta \).
Straight-Through Estimator
Revisited

[105x225]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
$L_{hid}(s, z_{true}) = s^T z - s^T z_{true}$; $\frac{\partial L_{hid}}{\partial s} = z - z_{true}$.

We don't have labels! Induce labels by "pulling back" the downstream target: the "best" (unconstrained) latent value would be: $\arg \min_{\sim z} \mathbb{E}_{D} L^y(\sim z)$.

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

[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), $L_{\text{MTL}} = L(\hat{y}(z), y) + L_{\text{hid}}(s, z^{\text{true}})$
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 \( L_{hid}(s, z^{true}) = s^Tz - s^Tz^{true} \), \( \frac{\partial L_{hid}}{\partial s} = z - z^{true} \).
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: $\arg\min_{\tilde{z} \in \mathbb{R}^D} L(\hat{y}(\tilde{z}), y)$
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: $\arg \min_{\tilde{z} \in \mathbb{R}^D} L(\hat{y}(\tilde{z}), y)$
- One gradient descent step starting from $z$: $z^{true} \leftarrow z - \frac{\partial L}{\partial z}$
Straight-Through Estimator
Revisited

- In the forward pass, $z = \arg \max(s)$.
- If we had labels (multi-task learning), $L_{MTL} = L(\hat{y}(z), y) + L_{hid}(s, z^{true})$
- 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: $\arg\min_{\tilde{z} \in \mathbb{R}^D} L(\hat{y}(\tilde{z}), y)$
- One gradient descent step starting from $z$: $z^{true} \leftarrow z - \frac{\partial L}{\partial z}$

\[
\frac{\partial L_{MTL}}{\partial s} = \underbrace{\frac{\partial L}{\partial s}}_{=0} + \frac{\partial L_{hid}}{\partial s}
\]
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: $\arg \min_{\tilde{z} \in \mathbb{R}^D} L(\hat{y}(\tilde{z}), y)$
- One gradient descent step starting from $z$: $z^{true} \leftarrow z - \frac{\partial L}{\partial z}$

$$\frac{\partial L_{MTL}}{\partial s} = \frac{\partial L}{\partial s} + \underbrace{\frac{\partial L_{hid}}{\partial s}}_{=0} = z - \left(z - \frac{\partial L}{\partial z}\right) = \frac{\partial L}{\partial z}$$
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)
Structured STE: perceptron update with induced annotation

\[
\begin{align*}
\arg \min_{\mu \in \mathbb{R}^D} L(\hat{y}(\mu), y) & \approx z - \nabla_z L(z) \rightarrow z^{\text{true}} \\
\text{(one step of gradient descent)}
\end{align*}
\]

SPIGOT takes into account the constraints; uses the induced annotation

\[
\begin{align*}
\arg \min_{\mu \in \mathcal{M}} L(\hat{y}(\mu), y) & \approx \text{Proj}_\mathcal{M}(z - \nabla_z L(z)) \rightarrow z^{\text{true}} \\
\text{(one step of projected gradient descent!)}
\end{align*}
\]
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)

- 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.
Summary: Straight-Through Estimator

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 $L^z(x)$. Now we will see how to apply STE for stochastic graphs, as an alternative approach of 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.
Summary: Straight-Through Estimator

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 $L(\hat{z}(x))$
Summary: Straight-Through Estimator

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:

$$E_{\pi}^{\theta}(z)$$

$$L(z)$$

REINFORCE (previous section).

High variance.

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

$$\mathbb{E}_{\pi_\theta(z|x)}[L(z)]$$
Stochastic node in the computation graph

Recall the stochastic objective:

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

- REINFORCE (previous section).
Recall the stochastic objective:

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

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

Recall the stochastic objective:

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

- 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 computation.

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

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.

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

What is the gradient of a sample \( \frac{\partial z}{\partial \theta} \)?!
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 \( \frac{\partial z}{\partial \theta} \) ?!
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. \( s \)!
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 \( z \) with respect to \( \theta \)?
- Reparameterization: Moving stochasticity out of the gradient path.
  Makes \( z \) deterministic with respect to \( s \).
  We can backpropagate through the deterministic input to \( z \).

As a result:
Stochasticity is moved as an input.
How do we sample from a categorical variable? [Jang et al., 2017, Maddison et al., 2016]

Categorical reparameterization

\[ s + \epsilon + z \]

\( \epsilon \) (stochastic)
Categorical reparameterization

How do we sample from a categorical variable?

\[ \mathcal{N}(s + \epsilon, \sigma), \mathcal{N}(\epsilon) \]

[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:
   $$p = \text{softmax}(s)$$
   $$c_i = \sum_j p_j$$
   Uniform($0, 1$)
   return $z$ 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.) 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:

$$p_i = \text{softmax}(s_i)$$

$$c_i = \sum_j s_j p_j$$

$$u \sim \text{Uniform}(0, 1)$$

$$z = \text{arg max} (s_i + \varepsilon_i)$$

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

References & more info:
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
   - $\epsilon_i = -\log(-\log(u_i))$
   - $z = \arg\max(s + \epsilon)$

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]

We have an argmax again and cannot backpropagate!
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:
   - $p = \text{softmax}(s)$
   - $c_i = \sum_{j\leq i} p_j$

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]

We have an argmax again and cannot backpropagate!
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:**
   - $p = \text{softmax}(s)$
   - $c_i = \sum_{j \leq i} p_j$
   - $u \sim \text{Uniform}(0, 1)$

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. $c_t \leq u < c_{t+1}$
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
   - $\epsilon_i = -\log(-\log(u_i))$
   - $z = \arg\max(s + \epsilon)$

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

RequiressamplingfromtheStandardGumbelDistribution $\mathcal{G}(0,1)$.

Derivation & more info:
- [Adams, 2013]
- [Vieira, 2014]

We have an argmax again and cannot backpropagate!
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 $G(0, 1)$.

Derivation & more info: [Adams, 2013], [Vieira, 2014]

We have an argmax again and cannot backpropagate!
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
   - $u_i \sim \text{Uniform}(0, 1)$
   - $\epsilon_i = -\log(-\log(u_i))$

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.)

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

Derivation & more info:
- [Adams, 2013]
- [Vieira, 2014]

We have an arg max again and cannot backpropagate!
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 $z = e_t \text{ s.t. } c_t \leq u < c_{t+1}$

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 $G(0,1)$. 

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. $c_t < z < c_{t+1}$

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.)

Requirers 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 $z = e_t$ such that $c_t < u < c_{t+1}$

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.)

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

Derivation & more info:
- [Adams, 2013]
- [Vieira, 2014]

We have an argmax again and cannot backpropagate!
Straight-Through Gumbel Estimator

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

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

\[ \text{Forward: pretend we had} \quad p = \text{softmax}(s + \epsilon) \]

What about the structured case?
Straight-Through Gumbel Estimator

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

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

"Forward: pretend we had" \( p = \text{softmax}(s + \epsilon) \)

\( \epsilon = -\log(-\log(u_i)) \)

\( u_i \sim U(0,1) \)

[Jang et al., 2017, Maddison et al., 2016]
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
  \( \tilde{p} = \text{softmax}(s + \epsilon) \)

\[
\epsilon = -\log(-\log(u_i))
\]

\[
u_i \sim \text{U}(0,1)
\]
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?

$\epsilon = -\log(-\log(u))$
$u \sim U(0,1)$

[Hinton et al., 2015, Maddison et al., 2016]
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].

depth-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
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 normal Gumbel distribution.
- $\epsilon \sim G(0, 1)$
Sampling from factorized models

Perturb-and-MAP

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

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

\[ \epsilon \sim G(0, 1) \]
\[ \tilde{\eta} = \eta + \epsilon \]
\[ \arg \max_{z \in Z} \tilde{\eta}^T z \]
Sampling from factorized models
Perturb-and-MAP

Reparameterize by perturbing the arc scores. (inexact!)

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

\[ \epsilon \sim G(0, 1) \]
\[ \tilde{\eta} = \eta + \epsilon \]
\[ \arg \max_{z \in \mathcal{Z}} \tilde{\eta}^T z \]
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...
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 Gumbel (Perturb & MAP)
- Straight-Through
- SPIGOT
- Structured Attn. Nets
- SparseMAP

And more, after the break!
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

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

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

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

\[ (\text{argmax}) \]

\[ y = g_\phi(z, x) \]
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 \( f(\mathbb{E}[z]) \)

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

(softmax)

\[ y = g_\phi(z, x) \]
One solution: smooth relaxation

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

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

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

(softmax)

\[ y = g_\phi(z, x) \]
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
What is softmax?

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

\[ p_{\top} \text{ argmax } \] maximizes expected score

Shannon entropy of \( p \):

\[ H(p) = -\sum_i p_i \log p_i \]

so \( \text{maximizes} \) expected score + entropy:

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

1.5

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

Maximizes expected score

Shannon entropy of $p$:

$H(p) = -\sum_i p_i \log p_i$
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

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

$\exp s_i$
What is softmax?

Often defined via

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

but where does it come from?

\(p \in \Delta: \text{probability distribution over choices}\)

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

argmax
What is softmax?

Often defined via 

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

but where does it come from?

\[ p \in \Delta: \text{probability distribution over choices} \]

Expected score under \( p \): 

\[ \mathbb{E}_{i \sim p} s_i = p^\top s \]

\textbf{argmax} maximizes \textit{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 \( p \): \( \mathbb{E}_{i \sim p} s_i = p^T s \)

\textbf{argmax} maximizes expected score

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

\textbf{softmax} maximizes \textbf{expected score} + \textbf{entropy}:

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

**Proposition.** The unique solution to \( \operatorname{arg\,max}_{p \in \Delta} p^T s + H(p) \) 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:

- 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:

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

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

\[
p^T \nu = 0
\]

\[
p \in \Delta
\]

\[
\nu \geq 0
\]

[Boyd and Vandenberghe, 2004; Wainwright and Jordan, 2008]
**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:

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

Lagrangian:

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

Optimality conditions (KKT):

\[
\begin{aligned}
0 &= \nabla_{p_i} \mathcal{L}(\mathbf{p}, \mathbf{\nu}, \tau) = -s_i + \log p_i + 1 - \nu_i + \tau \\
\mathbf{p}^\top \mathbf{\nu} &= 0 \\
\mathbf{p} &\in \Delta \\
\mathbf{\nu} &\geq 0
\end{aligned}
\]
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):

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

$p^T \nu = 0$

$p \in \Delta$

$\nu \geq 0$

log $p_i = s_i + \nu_i - (\tau + 1)$

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

$\log p_i = s_i + \nu_i - (\tau + 1)$

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

$p_i = \frac{\exp(s_i)}{Z}$

Must find $Z$ such that $\sum_j p_j = 1$. 

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

Classic result, e.g., [Boyd and Vandenberghe, 2004, Wainwright and Jordan, 2008].
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 Cnd \( Z \) 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]
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 \( p \geq 0, \ p^T \mathbf{1} = 1 \)

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 \( \sum_j p_j = 1 \).
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):

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

\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:

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

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):

\[
\begin{align*}
0 &= -s_i + \log p_i + 1 - \nu_i + \tau \\
p^T \nu &= 0 \\
p &\in \Delta \\
\nu &\geq 0
\end{align*}
\]

\[
\log p_i = s_i + \nu_i - (\tau + 1)
\]

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

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

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

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

[Niculae and Blondel, 2017]
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

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

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

---

[Niculae and Blondel, 2017]
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, Martins 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]
(Mon 13:50, poster session 2D)
Generalizing softmax: Smoothed argmaxes

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

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

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

\( \mu = [0.3, 0.7, 0.3, 0.7, 0.1, 0.2] \)

\( p_1 = 0.2, \quad z_1 = [1, 0, 0, 0, 1, 0, 0, 0, 1] \)
\( p_2 = 0.7, \quad z_2 = [0, 0, 1, 0, 0, 1, 1, 0, 0] \)
\( p_3 = 0.1, \quad z_3 = [1, 0, 0, 0, 1, 0, 0, 1, 0] \)
Niculae et al., 2018

$$\text{argmax}$$

$$\arg\max_p 2\Delta p^\top s + H(p)$$

$$\text{sparsemax}$$

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

MAP

$$\arg\max_\mu 2\Delta \mu^\top \eta + e H(\mu)$$

$$\text{SparseMAP}$$

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

Just like so[max relaxes arg max, marginals relax MAP differently.

Unlike arg max so[max, computation is not obvious!
**argmax** \( \arg \max_{p \in \Delta} p^\top s \)

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

\[ \text{MAP} \ \arg\max_{\mu \in \mathcal{M}} \mu^T \eta \]

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

Just like \( \text{so}\max \) relaxes \( \text{argmax} \), marginals relax MAP differently.

Unlike \( \text{argmax} / \text{so}\max \), computation is not obvious!

[diagram of \( \Delta \) and \( \mathcal{M} \)]
- **argmax** $\underset{p \in \Delta}{\arg \max} p^T s$
- **softmax** $\underset{p \in \Delta}{\arg \max} p^T s + H(p)$
- **MAP** $\underset{\mu \in \mathcal{M}}{\arg \max} \mu^T \eta$

Unlike argmax/softmax, computing is not obvious!
argmax $\arg \max_{p \in \Delta} p^T s$

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

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

marginals $\arg \max_{\mu \in \mathcal{M}} \mu^T \eta + \tilde{H}(\mu)$
Just like softmax relaxes argmax, marginals relax MAP differentiably!
argmax \ arg \ max_{p \in \Delta} p^T s

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

softmax \ arg \ max_{p \in \Delta} p^T s + H(p)
marginals \ arg \ max_{\mu \in \mathcal{M}} \mu^T \eta + \hat{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><strong>Sequence tagging</strong></td>
<td>Viterbi</td>
<td>Forward-Backward</td>
</tr>
<tr>
<td></td>
<td>[Rabiner, 1989]</td>
<td>[Rabiner, 1989]</td>
</tr>
<tr>
<td><strong>Constituent trees</strong></td>
<td>CKY</td>
<td>Inside-Outside</td>
</tr>
<tr>
<td></td>
<td>[Kasami, 1966, Younger, 1967]</td>
<td>[Baker, 1979]</td>
</tr>
<tr>
<td></td>
<td>[Cocke and Schwartz, 1970]</td>
<td></td>
</tr>
<tr>
<td><strong>Temporal alignments</strong></td>
<td>DTW</td>
<td>Soft-DTW</td>
</tr>
<tr>
<td></td>
<td>[Sakoe and Chiba, 1978]</td>
<td>[Cuturi and Blondel, 2017]</td>
</tr>
<tr>
<td><strong>Dependency trees</strong></td>
<td>Max. Spanning Arborescence</td>
<td>Matrix-Tree</td>
</tr>
<tr>
<td><strong>Assignments</strong></td>
<td>Kuhn-Munkres</td>
<td>#P-complete</td>
</tr>
<tr>
<td></td>
<td>[Kuhn, 1955, Jonker and Volgenant, 1987]</td>
<td>[Valiant, 1979, Taskar, 2004]</td>
</tr>
<tr>
<td></td>
<td>[Valiant, 1979]</td>
<td></td>
</tr>
</tbody>
</table>

*References:*

- Rabiner, 1989
- Sakoe and Chiba, 1978
- Chu and Liu, 1965, Edmonds, 1967
- Kuhn, 1955, Jonker and Volgenant, 1987
- Valiant, 1979, Taskar, 2004
Algorithms for specific structures

**Best structure (MAP)**
- Viterbi
  - [Rabiner, 1989]
- CKY
  - [Kasami, 1966, Younger, 1967]
  - [Cocke and Schwartz, 1970]
- DTW
  - [Sakoe and Chiba, 1978]

**Margins**
- Forward-Backward
  - [Rabiner, 1989]
- Inside-Outside
  - [Baker, 1979]
- Soft-DTW
  - [Cuturi and Blondel, 2017]
- Matrix-Tree
  - [Kirchhoff, 1847]
- #P-complete
  - [Valiant, 1979, Taskar, 2004]

**Sequence tagging**
- Viterbi
  - [Rabiner, 1989]

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

**Temporal alignments**
- DTW
  - [Sakoe and Chiba, 1978]

**Dependency trees**
- Max. Spanning Arborescence
  - [Chu and Liu, 1965, Edmonds, 1967]

**Assignments**
- Kuhn-Munkres
  - [Kuhn, 1955, Jonker and Volgenant, 1987]

**Assignments**
- Kuhn-Munkres
  - [Kuhn, 1955, Jonker and Volgenant, 1987]
Derivatives of marginals 1: DP

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

deep-spin.github.io/tutorial
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
   # 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]

---

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. \[ \alpha_{i,k} = \log \sum_{k'} \exp \left( \alpha_{i-1,k'} + (\eta_U)_{i,k} + (\eta_V)_{k',k} \right) \text{ for all } k \]
5. for $i \in n-1, \ldots, 1$ do
   # backward log-probabilities
6. \[ \beta_{i,k} = \log \sum_{k'} \exp \left( \beta_{i+1,k'} + (\eta_U)_{i+1,k'} + (\eta_V)_{k,k'} \right) \text{ for all } k \]
7. $Z = \sum_k \exp \alpha_{n,k}$
   # partition function
8. **return** $\mu = \exp \left( \alpha + \beta - \log Z \right)$
   # marginals

---

`deep-spin.github.io/tutorial`
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.

```plaintext
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
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
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 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

input $x$ \rightarrow \eta \rightarrow \mu \rightarrow output $y$

CRF marginals (from forward-backward) give attention weights $\eta(0, 1)$.

Similar idea for projected dependency trees with inside-outside and non-projected with the Matrix-Treetheorem [Liu and Lapata, 2018].

[Kim et al., 2017]

[deep-spin.github.io/tutorial]
Structured Attention Networks

Input $x$ is transformed through layers $\eta$ and $\mu$ to produce output $y$.

CRF marginals (from forward–backward)
give attention weights $(0, 1)$.

Similar idea for projected dependency trees with inside–outside and non-projective with the Matrix-Tree Theorem [Liu and Lapata, 2018].

[Kim et al., 2017]
Structured Attention Networks

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

$\eta(i)$: score of word $i$ receiving attention

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

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

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

\[ \eta(\text{dog} \rightarrow \text{on}): \text{arc score (tree constraints)} \]

\[ \mu(\text{dog} \rightarrow \text{on}): \text{probability of arc} \]

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

Similar idea for projective dependency trees with inside–outside

[Kim et al., 2017]
Structured Attention Networks

CRF marginals (from forward–backward) give attention weights $\in (0, 1)$
Similar idea for projective dependency trees with 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)
  $z = \arg \max_{z \in \mathcal{Z}} (\eta + \epsilon)^T z$

• Backward pass:
  pretend we did marginal inference
  $\tilde{\mu} = \arg \max_{\mu \in \mathcal{M}} (\eta + \epsilon)^T z + \tilde{H}(\mu)$
  (or some similar relaxation)
Back-propagating through marginals

Pros:

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

Cons:

(Structured Action 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.

Deep-Spin GitHub.io Tutorial
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;
- ELcient & numerically stable back-propagation through DPs is tricky;
- 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

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.
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 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, 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.
\[ \text{argmax} \arg \max_{p \in \Delta} p^\top s \]
\[ \text{softmax} \arg \max_{p \in \Delta} p^\top s + H(p) \]
\[ \text{sparsemax} \arg \max_{p \in \Delta} p^\top s - \frac{1}{2} \|p\|^2 \]
\[ \text{MAP} \arg \max_{\mu \in \mathcal{M}} \mu^\top \eta \]
\[ \text{marginals} \arg \max_{\mu \in \mathcal{M}} \mu^\top \eta + \tilde{H}(\mu) \]

[Niculae et al., 2018a]
- \textbf{argmax} \arginp \in \Delta \max p^\top s \\

- \textbf{softmax} \arginp \in \Delta \max p^\top s + H(p) \\

- \textbf{sparsemax} \arginp \in \Delta \max p^\top s - \frac{1}{2} \|p\|^2 \\

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

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

\[
= 0.6 \cdot 0 + 0.4 \cdot 0 = 0.6
\]

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

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

Completely modular: just add MAP
Algorithms for SparseMAP

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

linear constraints
(alas, exponentially many!)

quadratic objective

Conditional Gradient

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

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

linear constraints (\textit{alas, exponentially many!})

quadratic objective

Conditional Gradient

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

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

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

linear constraints
(alas, exponentially many!)

quadratic objective

Conditional Gradient

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

- select a new corner of \( \mathcal{M} \)

\[ \text{arg max} \quad \mu^T (\tilde{\eta} - \mu^{(t-1)}) \]

\[ \mu \in \mathcal{M} \]

\[ \tilde{\eta} \]

Completely modular: just add MAP
Algorithms for SparseMAP

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

linear constraints
(alas, exponentially many!)

quadratic objective

Conditional Gradient

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

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

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

linear constraints
(alas, exponentially many!)

quadratic objective

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

• select a new corner of \( \mathcal{M} \)
• update the (sparse) coefficients of \( \mathbf{p} \)
  • 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,
    Vinyes and Obozinski, 2017]
Algorithms for SparseMAP

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

linear constraints (alas, exponentially many!)

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

Conditional Gradient

- select a new corner
- update the (sparse)

Active Set achieves finite & linear convergence!

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

[Martins et al., 2015, Nocedal and Wright, 1999, Vinyes and Obozinski, 2017]
Algorithms for SparseMAP

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

linear constraints (alas, exponentially many!)

Conditional Gradient

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

- select a new corner of $M$
- 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,
    Vinyes and Obozinski, 2017]

Backward pass

$$\frac{\partial \mu}{\partial \eta}$$ is sparse

[deep-spin.github.io/tutorial 91]
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, Vinyes and Obozinski, 2017]

**Backward pass**

\[ \frac{\partial \mu}{\partial \eta} \text{ is sparse} \]

computing \( (\frac{\partial \mu}{\partial \eta})^T dy \)
takes \( O(\text{dim}(\mu) \text{nnz}(p^*)) \)
Algorithms for SparseMAP

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

linear constraints (alas, exponentially many!)

**Completely modular: just add MAP**

Frank and Wolfe, 1956

- select a new corner of \( \mathcal{M} \)
- update the (sparse) coefficients of \( \mathbf{p} \)
  - 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, Vinyes and Obozinski, 2017]

Completely modular: just add MAP

[deep-spin.github.io/tutorial]
<table>
<thead>
<tr>
<th>Gentleman</th>
<th>Overlooking</th>
<th>Neighborhood</th>
<th>Situation</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
</tbody>
</table>

A police officer watches a situation closely.
Niculae et al., 2018a

A police officer watches a situation closely.

A gentleman overlooking a neighborhood situation.
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
Structured latent variables without sampling

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

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

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 $\pi_\theta(z)$?
  $$
  \sum_h h^2 H \frac{\partial E}{\partial \theta} \propto \exp f_\theta(z)
  $$
  
  so $\arg\max$ $\pi_\theta(z)$ = 1 if $z = \text{MAP}(f_\theta(z))$ else 0

- $\arg\max$ Sparse MAP

- e.g., a TreeLSTM defined by $z$

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

- Exponentially large sum!

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

STE/SPIGOT relax $^y$ in backward.
Structured latent variables without sampling

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

sum over all possible trees
e.g., a TreeLSTM defined by $z$

Exponentially large sum!
Structured latent variables without sampling

How to define $\pi_\theta$?

- Idea 1: Sum over all possible trees.
- Idea 2: Exponentially large sum.
- Idea 3: Parsing model, using some scorer $f_\theta(z; x)$.

E.g., a TreeLSTM defined by $z$.
Structured latent variables without sampling

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

sum over all possible trees

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

to define \( \pi_\theta \)?

idea 1
idea 2
idea 3
Structured latent variables without sampling

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

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

1. Idea 1
2. Idea 2
3. Idea 3

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

sum over all possible trees

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

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

STE/SPIGOT relax \( y \) in backward.

Exponentially large sum!
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

\[ \sum_{h \in H} \frac{\partial E[L(z)]}{\partial \theta} \]
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**: Softmax
- **idea 3**: SparseMAP, e.g., a TreeLSTM defined by \( z \)

To find the optimal parameters \( \theta \), we can maximize

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

sum over all possible trees

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

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

Exponentially large sum!

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

STE/SPIGOT relax \( \phi \) in backward.
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: softmax
- Idea 3: e.g., a TreeLSTM defined by \(z\)

sum over all possible trees

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

STE/SPIGOT relax \(y\) in backward.
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 \( \pi_\theta \)?

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

- idea 2
- idea 3
Structured latent variables without sampling

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

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

1. **Idea 1**: \( \pi_\theta(z) \propto \exp(f_\theta(z)) \)
2. **Idea 2**: \( \pi_\theta(z) = 1 \) if \( z = \text{MAP}(f_\theta(\cdot)) \) else 0
3. **Idea 3**: Parsing model, using some scorer \( f_\theta(z; x) \)

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

Exponentially large sum!

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

STE/SPIGOT relax \( y \) in backward.
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 over all possible trees

E.g., a TreeLSTM defined by $z$

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

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

Exponentially large sum!
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** \( \pi_\theta(z) = 1 \) if \( z = \text{MAP}(f_\theta(\cdot)) \) else 0

**Idea 3**

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

sum over all possible trees

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

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

Exponentially large sum!

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

STE/SPIGOT relax \( y \) in backward.
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 \( \pi(\theta) \)?

**STE / SPIGOT relax \( \hat{y} \) in backward.**

**Idea 1**
- \( \pi(\theta) \)

**Idea 2**
- \( \pi(\theta) = 1 \) if \( z = \text{MAP}(f(\theta; \cdot)) \) else 0

**Idea 3**
- \( \arg\max \)

sum over all possible trees

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

depending on some parsing model, using some scorer \( f(\theta; z; x) \)

\( \partial \mathbb{E}[L(z)]/\partial \theta \)

Exponentially large sum!

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

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

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 over all possible trees
- e.g., a TreeLSTM defined by \( z \)
- parsing model, using some scorer \( f_\theta(z; x) \)

- softmax
- argmax
- SparseMAP

STE/SPIGOT relax $^y$ in backward.
Structured latent variables without sampling

\[ L(z) = 0.7 \times \hat{L}(z) + 0.3 \times \hat{L}(\phi(z), y) \]

recall our shorthand \( L(z) = L(\hat{y}_\phi(z), y) \)
Structured latent variables without sampling

\[
\mathcal{L}(\mathbf{z}) = 0.7 \times \mathcal{L}(\mathbf{\hat{\phi}}(\mathbf{z}), \mathbf{y}) + 0.3 \times \mathcal{L}(\mathbf{\hat{\phi}}(\mathbf{z}), \mathbf{y}) + 0 \times \mathcal{L}(\mathbf{\hat{\phi}}(\mathbf{z}), \mathbf{y}) + \ldots
\]

recall our shorthand \( \mathcal{L}(\mathbf{z}) = \mathcal{L}(\mathbf{\hat{\phi}}(\mathbf{z}), \mathbf{y}) \)
Structured latent variables without sampling

\[ L(z) = 0.7 \times L(z) + 0.3 \times L(z) + 0 \times L(z) + \ldots \]

\[ \mathbb{E}[L(z)] = 0.7 \times L(\hat{y}_\phi(z), y) + 0.3 \times L(\hat{y}_\phi(z), y) \]

recall our shorthand \( L(z) = L(\hat{y}_\phi(z), y) \)
### Stanford Sentiment (Accuracy)

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socher et al</td>
<td>83.1</td>
</tr>
<tr>
<td>Bigram Naive Bayes</td>
<td></td>
</tr>
<tr>
<td>TreeLSTM w/ CoreNLP</td>
<td>83.2</td>
</tr>
<tr>
<td>TreeLSTM w/ SparseMAP</td>
<td>84.7</td>
</tr>
<tr>
<td>[Corro and Titov, 2019b]</td>
<td></td>
</tr>
<tr>
<td>GCN w/ CoreNLP</td>
<td>83.8</td>
</tr>
<tr>
<td>GCN w/ Perturb-and-MAP</td>
<td>84.6</td>
</tr>
<tr>
<td>[Niculae et al., 2018b]</td>
<td></td>
</tr>
</tbody>
</table>

### Stanford Natural Language Inference (Accuracy)

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Kim et al., 2017]</td>
<td></td>
</tr>
<tr>
<td>Simple Attention</td>
<td>86.2</td>
</tr>
<tr>
<td>Structured Attention</td>
<td>86.8</td>
</tr>
<tr>
<td>[Liu and Lapata, 2018]</td>
<td></td>
</tr>
<tr>
<td>100D SAN</td>
<td>86.8</td>
</tr>
<tr>
<td>Yogatama et al</td>
<td></td>
</tr>
<tr>
<td>100D RL-SPINN</td>
<td>80.5</td>
</tr>
<tr>
<td>[Choi et al., 2018]</td>
<td></td>
</tr>
<tr>
<td>100D ST Gumbel-Tree</td>
<td>82.6</td>
</tr>
<tr>
<td>300D</td>
<td>85.6</td>
</tr>
<tr>
<td>600D</td>
<td>86.0</td>
</tr>
<tr>
<td>[Corro and Titov, 2019b]</td>
<td></td>
</tr>
<tr>
<td>Latent Tree + 1 GCN</td>
<td>85.2</td>
</tr>
<tr>
<td>Latent Tree + 2 GCN</td>
<td>86.2</td>
</tr>
</tbody>
</table>
V. Conclusions
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

\[ p = 22.6\% \]

CoreNLP parse, \( p = 21.4\% \)

\[ p = 15.33\% \]

\[ * \text{lovely and poignant} . \]

\[ * \text{a deep and meaningful film}. \]

\[ p = 15.27\% \]

CoreNLP parse, \( p = 0\% \)

\[ * \text{a deep and meaningful film}. \]

\[ * \text{a deep and meaningful film}. \]
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
- Straight-Through Gumbel (Perturb & MAP)
- SparseMAP
- Straight-Through
- SPIGOT
- Structured Attn. Nets
- SparseMAP
Overview

\[ E_{\pi_{\theta}(z|x)}[L(z)] \quad L\left( \arg \max_z \pi_{\theta}(z \mid x) \right) \quad L\left( E_{\pi_{\theta}(z|x)}[z] \right) \]

- 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:

- SPL: Sampling. (Simple in incremental/unstructured, hard for most global structures.)
- MAP: Finding the highest-scoring structure.
- MRG: Marginal inference.
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]
References I


References III


References VI


