Latent Structure Models for NLP

André Martins — Instituto de Telecomunicações & IST & Unbabel
Tsvetomila Mihaylova — Instituto de Telecomunicações
Nikita Nangia — NYU
Vlad Niculae — Instituto de Telecomunicações

deep-spin.github.io/tutorial

I. Introduction

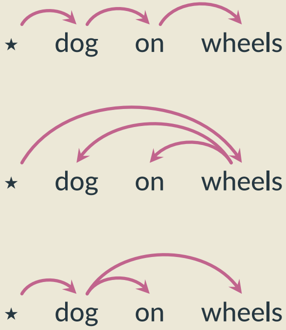
Structured prediction and NLP

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

Examples of structure in NLP

Dependency parsing

...



...

Examples of structure in NLP

Dependency parsing

…

⋆ dog on wheels

Exponentially many parse trees!

Cannot enumerate.

⋆ dog on wheels

…

Examples of structure in NLP

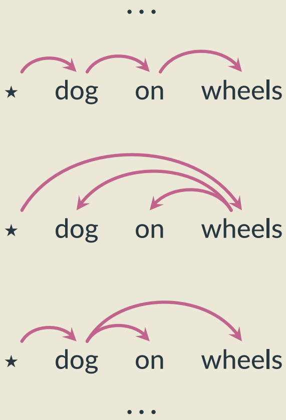
POS tagging

VERB PREP NOUN
dog on wheels

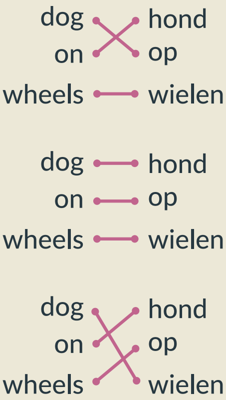
NOUN PREP NOUN
dog on wheels

NOUN DET NOUN
dog on wheels

Dependency parsing



Word alignments



NLP 5 years ago:

Structured prediction and pipelines

NLP 5 years ago:

Structured prediction and pipelines

- Big pipeline systems, connecting different structured predictors, trained separately
- **Advantages:** fast and simple to train, can rearrange pieces 😊

NLP 5 years ago:

Structured prediction and pipelines

- Big pipeline systems, connecting different structured predictors, trained separately
- **Advantages:** fast and simple to train, can rearrange pieces 😊
- **Disadvantage:** linguistic annotations required for each component 😰

NLP 5 years ago:

Structured prediction and pipelines

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

NLP today:

End-to-end training

NLP today:

End-to-end training

- Forget pipelines—train everything from scratch!
- No more error propagation or linguistic annotations! 🎉

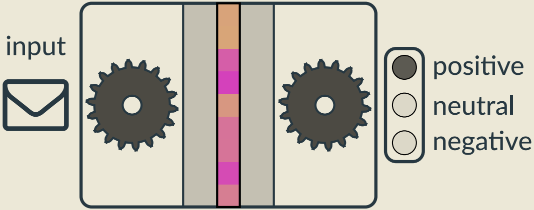
NLP today:

End-to-end training

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

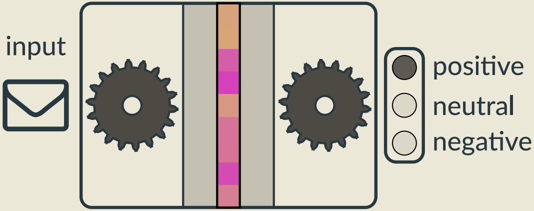
Representation learning

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



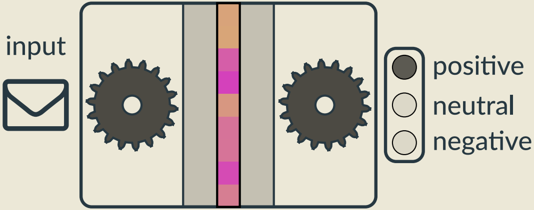
Representation learning

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



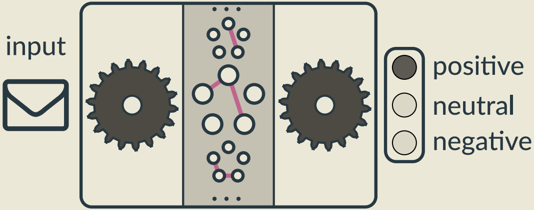
Latent structure models

- Seek *structured* hidden representations instead!



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:

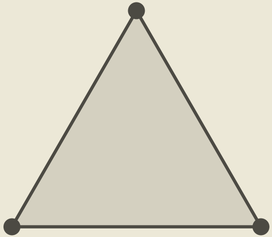
- It's not about continuous latent variables
- It's not about deep generative learning
- We won't cover GANs, VAEs, etc.
- There are (very good) recent tutorials on deep variational models for NLP:
  - "Variational Inference and Deep Generative Models" (Schulz and Aziz, ACL 2018)
  - "Deep Latent-Variable Models for Natural Language" (Kim, Wiseman, Rush, EMNLP 2018)

Background

Unstructured vs structured

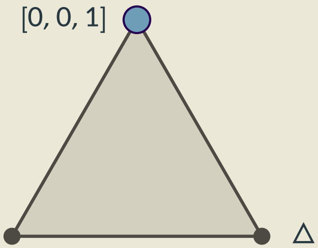
- To better explain the math, we'll often backtrack to *unstructured* models (where the latent variable is a categorical) before jumping to the *structured* ones

The unstructured case: Probability simplex



$\triangle$

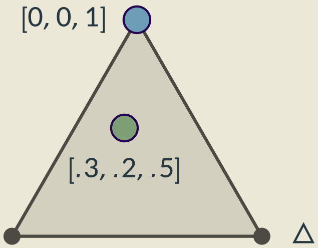
The unstructured case: Probability simplex



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

$$\boldsymbol{z}_c = [0, \ldots, 0, \underbrace{1}_{c^{\text{th}} \text{ position}}, 0, \ldots, 0].$$

The unstructured case: Probability simplex



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

$$\boldsymbol{z}_c = [0, \ldots, 0, \underbrace{1}_{c^{\text{th}} \text{ position}}, 0, \ldots, 0].$$

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

$$\boldsymbol{p} \geq \mathbf{0}, \ \sum_c p_c = 1.$$

What's the analogous of $\triangle$ for a structure?

- A structured object $\boldsymbol{z}$ can be represented as a *bit vector*.

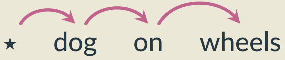
What's the analogous of $\triangle$ for a structure?

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

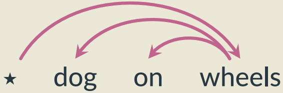
What's the analogous of △ for a structure?

- A structured object $\boldsymbol{z}$ 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!

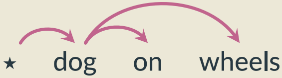
$\boldsymbol{z}_1 = [1, 0, 0, 0, 1, 0, 0, 0, 1]$



$\boldsymbol{z}_2 = [0, 0, 1, 0, 0, 1, 1, 0, 0]$

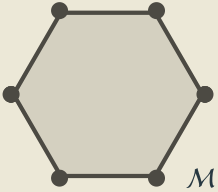


$\boldsymbol{z}_3 = [1, 0, 0, 0, 1, 0, 0, 1, 0]$



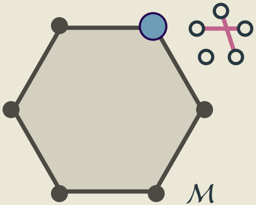
[Wainwright and Jordan, 2008]

The structured case: Marginal polytope



The structured case: Marginal polytope

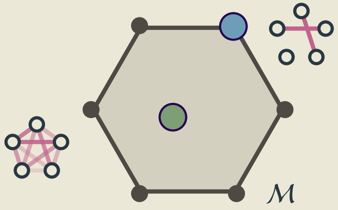
- Each vertex corresponds to one such *bit vector* $\boldsymbol{z}$



[Wainwright and Jordan, 2008]

The structured case: Marginal polytope

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

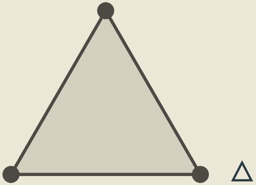


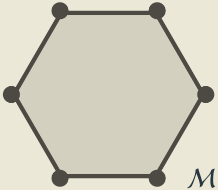
$$\boldsymbol{\mu} = \underbrace{p_1\boldsymbol{z}_1 + \ldots + p_N\boldsymbol{z}_N}_{\text{exponentially many terms}}, \quad \boldsymbol{p} \in \Delta.$$

$$\begin{aligned} p_1 &= 0.2, & \boldsymbol{z}_1 &= [1,0,0,0,1,0,0,0,1] \\ p_2 &= 0.7, & \boldsymbol{z}_2 &= [0,0,1,0,0,1,1,0,0] \\ p_3 &= 0.1, & \boldsymbol{z}_3 &= [1,0,0,0,1,0,0,1,0] \end{aligned} \quad \Rightarrow \quad \boldsymbol{\mu} = [.3,0,.7,0,.3,.7,.7,.1,.2].$$

Unstructured vs Structured

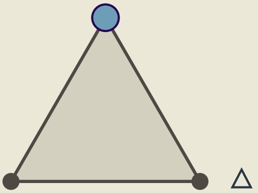
- Unstructured case: simplex Δ
- Structured case: marginal polytope $\mathcal{M}$



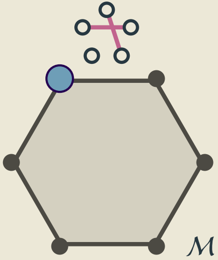


Unstructured vs Structured

- Unstructured case: simplex Δ

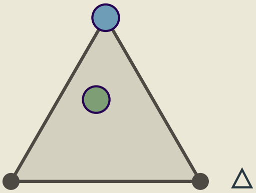


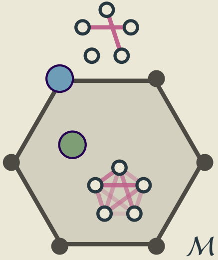
- Structured case: marginal polytope $\mathcal{M}$



Unstructured vs Structured

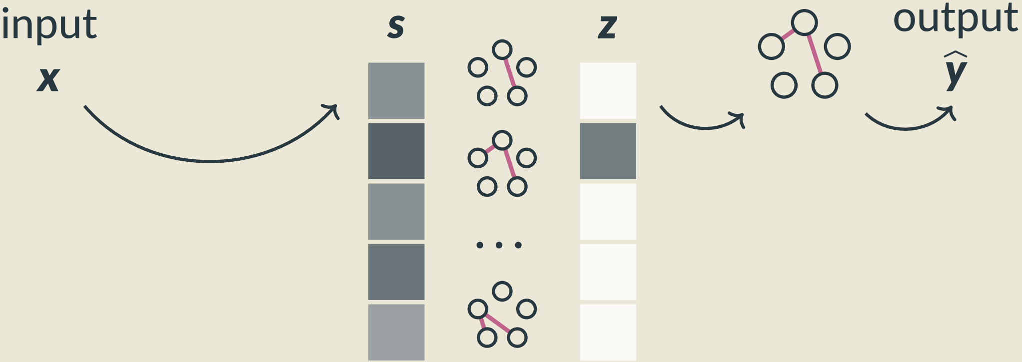
- Unstructured case: simplex $\triangle$
- Structured case: marginal polytope $\mathcal{M}$





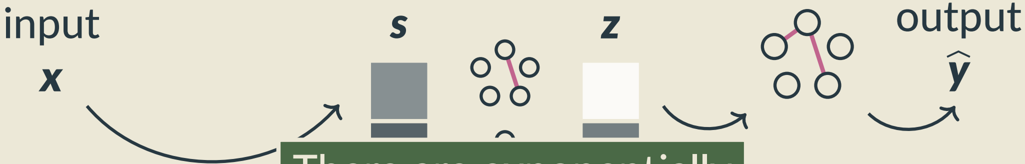
Computing the most likely structure

is a very high-dimensional argmax



Computing the most likely structure

is a very high-dimensional argmax

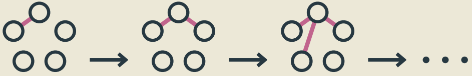


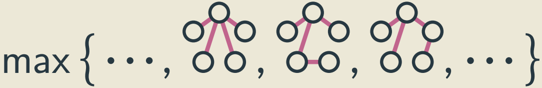
There are exponentially many structures

(*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.

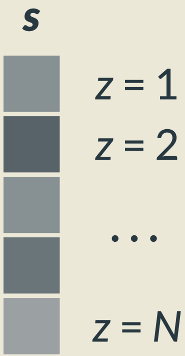
$$z = 1$$

$$z = 2$$

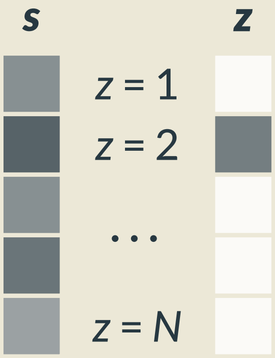
$$\dots$$

$$z = N$$

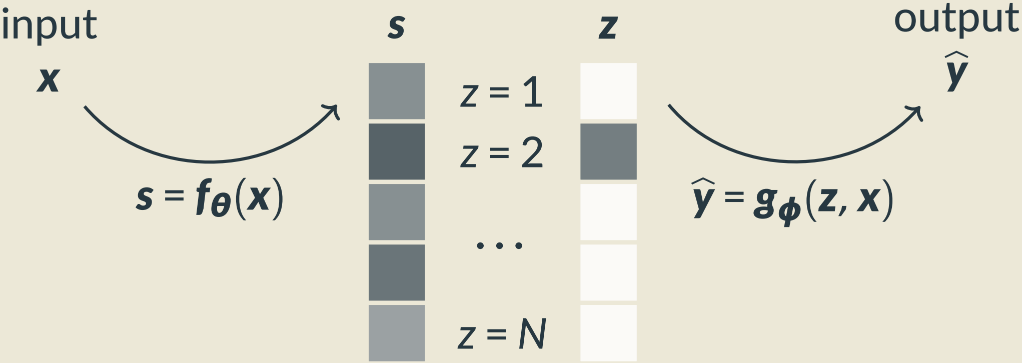
The challenge of discrete choices.



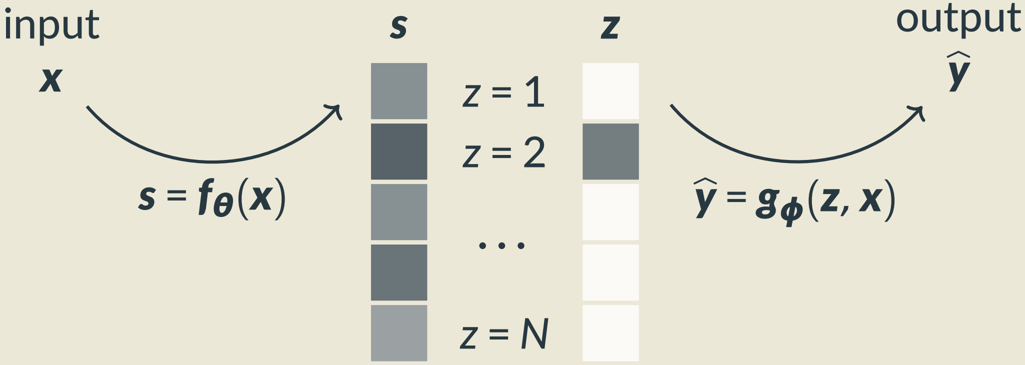
The challenge of discrete choices.



The challenge of discrete choices.

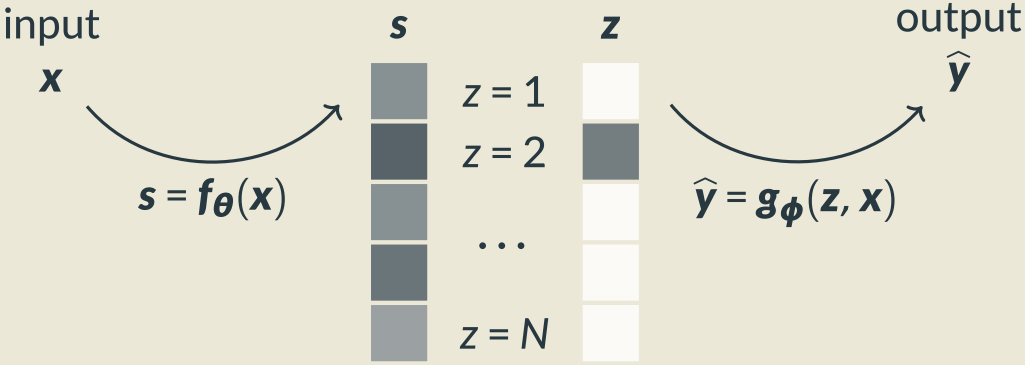


The challenge of discrete choices.



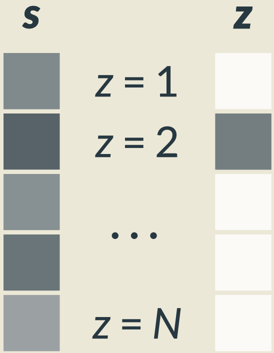
$$\frac{\partial L(\widehat{\boldsymbol{y}}, \boldsymbol{y})}{\partial \boldsymbol{w}} = ?$$

The challenge of discrete choices.



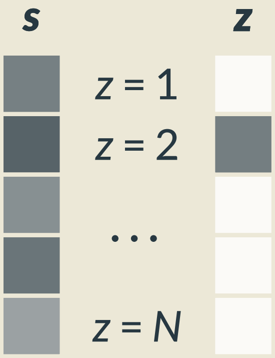
$\frac{\partial L(\widehat{\boldsymbol{y}}, \boldsymbol{y})}{\partial \boldsymbol{w}} = ?$ or, essentially, $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{s}} = ?$

Discrete mappings are “flat”



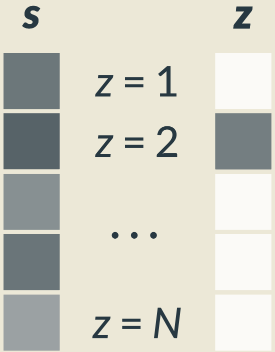
$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{s}} = ?$$

Discrete mappings are “flat”



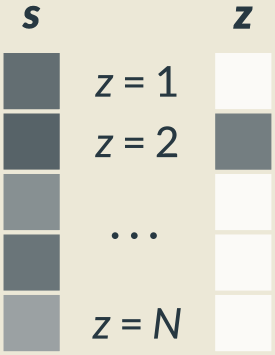
$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{s}} = ?$$

Discrete mappings are “flat”



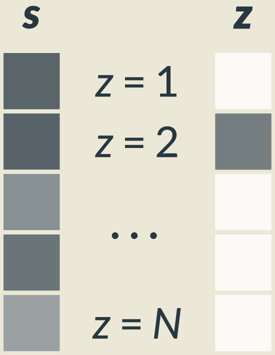
$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{s}} = ?$$

Discrete mappings are "flat"



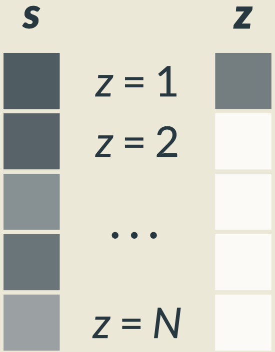
$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{s}} = ?$$

Discrete mappings are “flat”



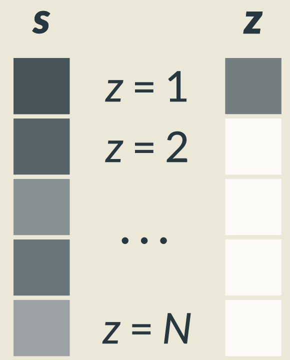
$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{s}} = ?$$

Discrete mappings are “flat”



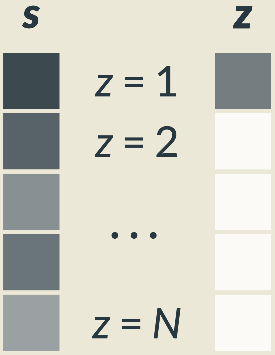
$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{s}} = ?$$

Discrete mappings are "flat"



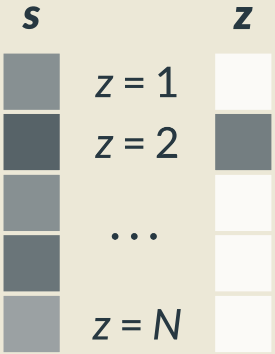
$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{s}} = ?$$

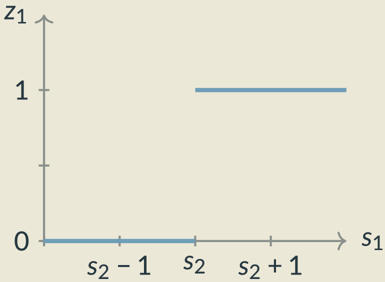
Discrete mappings are “flat”



$$\frac{\partial \mathbf{z}}{\partial \mathbf{s}} = ?$$

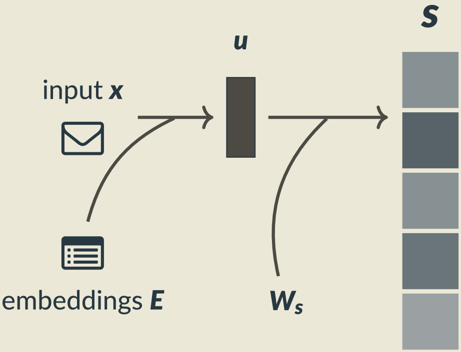
Argmax





$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{s}} = \mathbf{0}$$

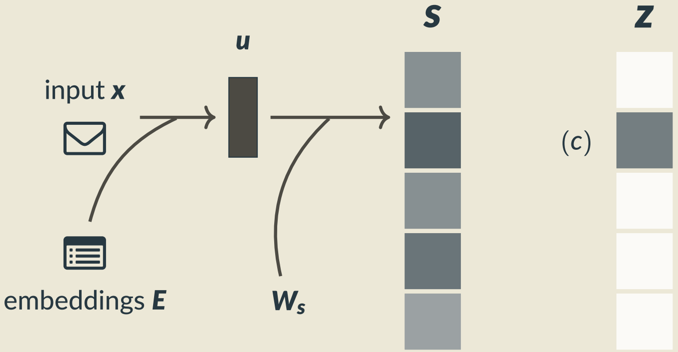
Example: Regression with latent categorization



$\boldsymbol{u} = \frac{1}{|\boldsymbol{x}|} \sum_j \boldsymbol{E}_{x_j}$

$\boldsymbol{s} = \boldsymbol{W_s u}$

Example: Regression with latent categorization

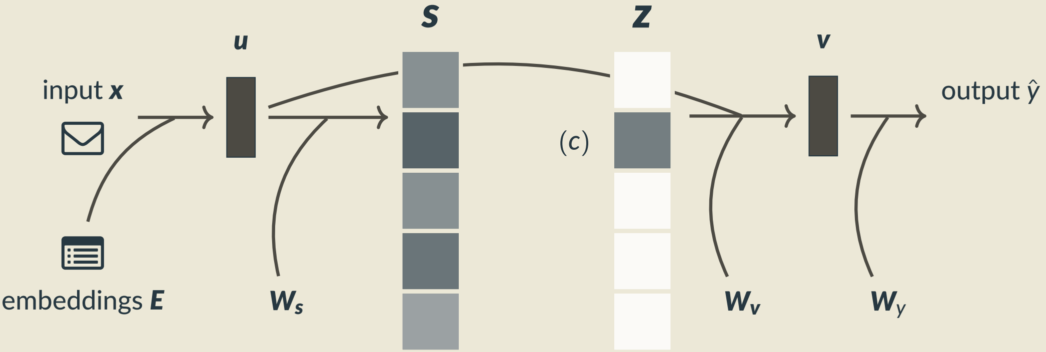


$\boldsymbol{u} = \frac{1}{|\boldsymbol{x}|} \sum_j \boldsymbol{E}_{x_j}$

$\boldsymbol{s} = \boldsymbol{W_s u}$

predict topic c ($\boldsymbol{z} = \boldsymbol{e}_c$)

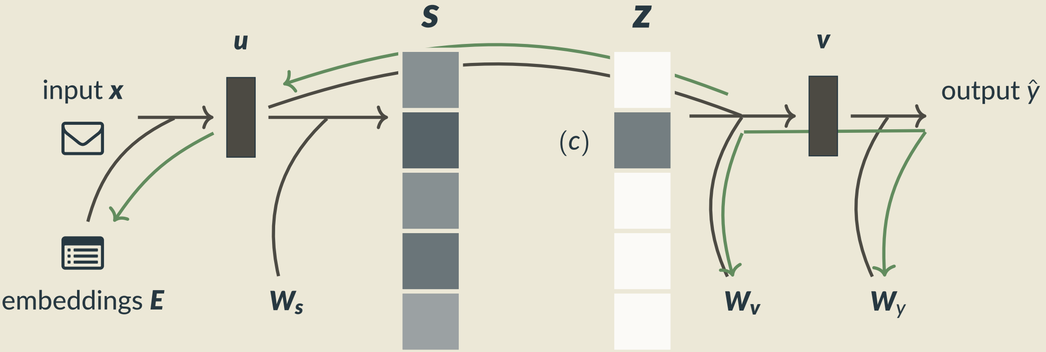
Example: Regression with latent categorization



$\boldsymbol{u} = \frac{1}{|\boldsymbol{x}|}\sum_j \boldsymbol{E}_{x_j}$ $\quad$ $\boldsymbol{s} = \boldsymbol{W_s}\boldsymbol{u}$ $\quad$ $\boldsymbol{v} = \tanh\left(\boldsymbol{W_v}[\boldsymbol{u}, \boldsymbol{z}]\right)$ $\quad$ $\hat{y} = \boldsymbol{W}_y\boldsymbol{v}$ $\quad$ $L = (\hat{y} - y)^2$

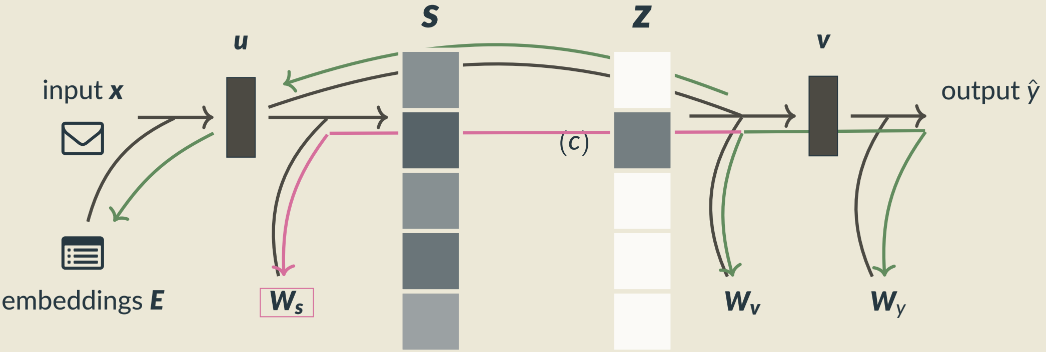
predict topic c ($\boldsymbol{z} = \boldsymbol{e}_c$)

Example: Regression with latent categorization



$\boldsymbol{u} = \frac{1}{|\boldsymbol{x}|} \sum_j \boldsymbol{E}_{x_j}$ $\quad$ $\boldsymbol{s} = \boldsymbol{W}_s \boldsymbol{u}$ $\quad$ $\boldsymbol{v} = \tanh\left(\boldsymbol{W}_v[\boldsymbol{u}, \boldsymbol{z}]\right)$ $\quad$ $\hat{y} = \boldsymbol{W}_y \boldsymbol{v}$ $\quad$ $L = (\hat{y} - y)^2$

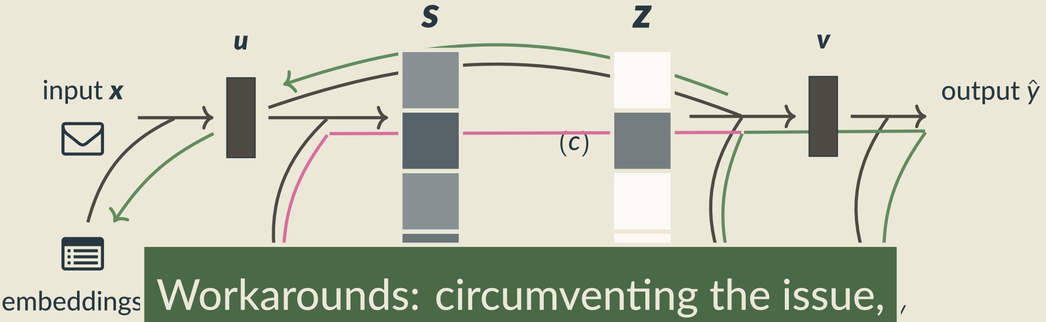
Example: Regression with latent categorization



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

$$\frac{\partial L}{\partial \boldsymbol{W_s}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \boldsymbol{v}} \frac{\partial \boldsymbol{v}}{\partial \boldsymbol{z}} \overbrace{\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{s}}}^{\equiv 0} \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{W_s}}$$

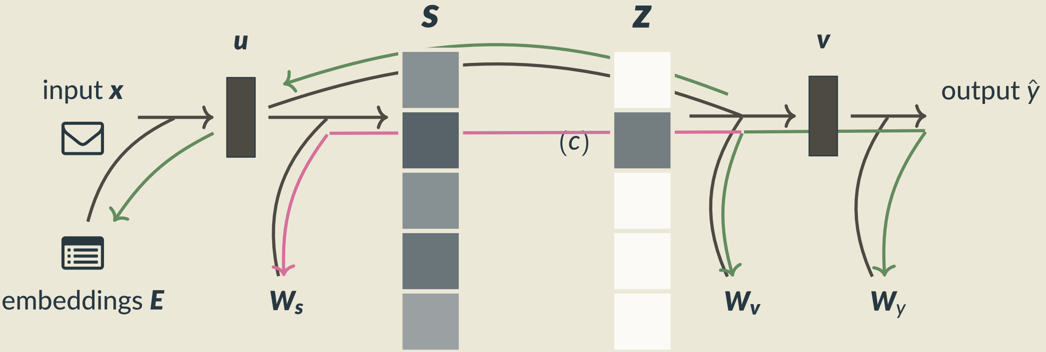
Example: Regression with latent categorization



$u = \frac{1}{|x|} \sum_j E_{x_j}$ … $-y)^2$

Workarounds: circumventing the issue, bypassing discrete variables

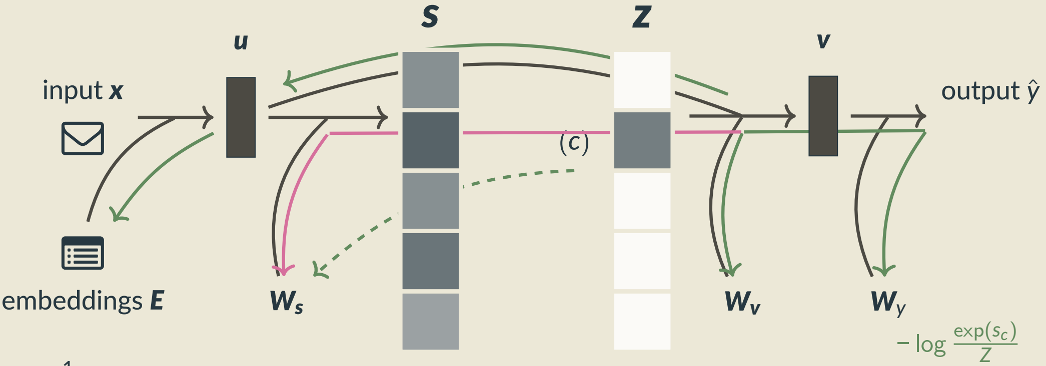
Example: Regression with latent categorization



$\boldsymbol{u} = \frac{1}{|\boldsymbol{x}|}\sum_j \boldsymbol{E}_{x_j}$ $\boldsymbol{s} = \boldsymbol{W_s u}$ $\boldsymbol{v} = \tanh\left(\boldsymbol{W_v}[\boldsymbol{u}, \boldsymbol{z}]\right)$ $\hat{y} = \boldsymbol{W}_y \boldsymbol{v}$ $L = (\hat{y} - y)^2$

Option 1. Pretrain latent classifier $\boldsymbol{W_s}$

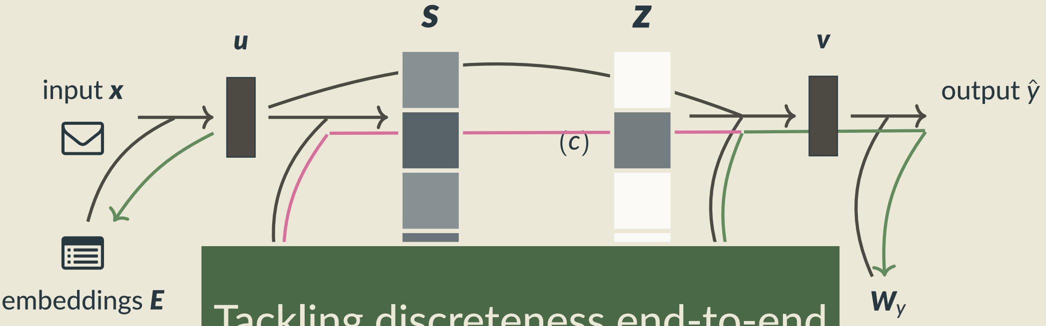
Example: Regression with latent categorization



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

Option 2. Multi-task learning

Example: Regression with latent categorization

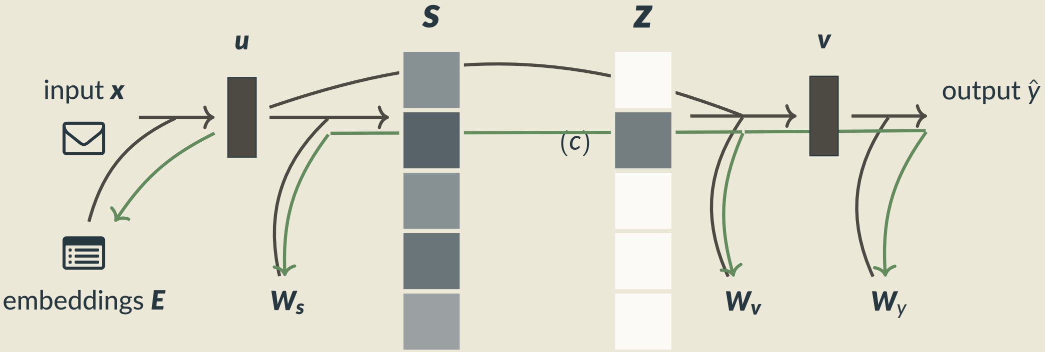


Tackling discreteness end-to-end

$\boldsymbol{u} = \frac{1}{|\boldsymbol{x}|}\sum_j \boldsymbol{E}_{x_j}$

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

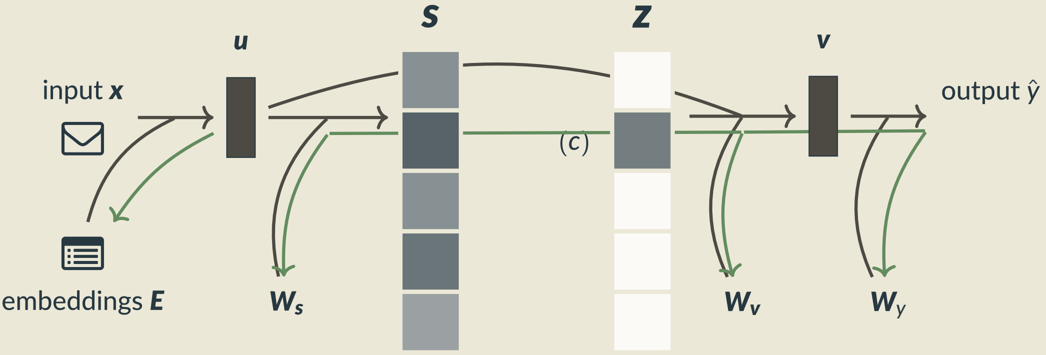
Example: Regression with latent categorization



$\boldsymbol{u} = \frac{1}{|\boldsymbol{x}|}\sum_j \boldsymbol{E}_{x_j}$ $\boldsymbol{s} = \boldsymbol{W_s u}$ $\boldsymbol{v} = \tanh\left(\boldsymbol{W_v}[\boldsymbol{u}, \boldsymbol{z}]\right)$ $\hat{y} = \boldsymbol{W}_y \boldsymbol{v}$ $L = \mathbb{E}_{\boldsymbol{z}}(\hat{y} - y)^2$

Option 3. Stochasticity! $\frac{\partial \mathbb{E}_{\boldsymbol{z}}(\hat{y}(\boldsymbol{z}) - y)^2}{\partial \boldsymbol{W}_s} \neq \boldsymbol{0}$

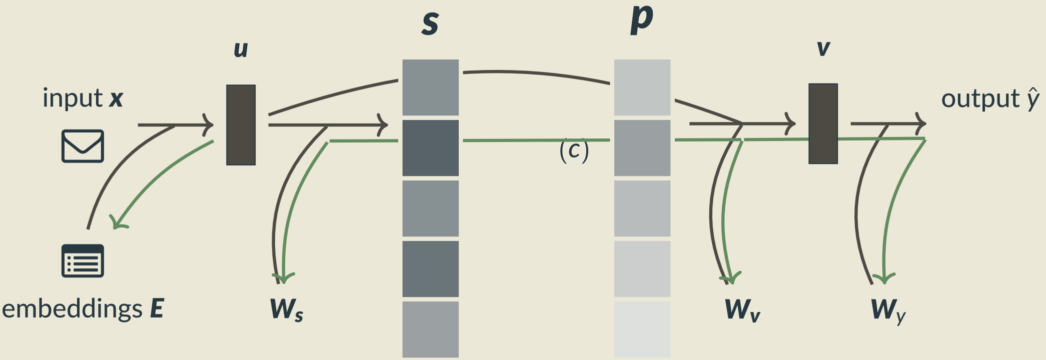
Example: Regression with latent categorization



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

Option 4. Gradient surrogates (*e.g.* straight-through, $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{s}} \leftarrow \boldsymbol{I}$)

Example: Regression with latent categorization



$\boldsymbol{u} = \frac{1}{|\boldsymbol{x}|} \sum_j \boldsymbol{E}_{x_j}$ $\quad$ $\boldsymbol{s} = \boldsymbol{W_s u}$ $\quad$ $\boldsymbol{v} = \tanh\left(\boldsymbol{W_v}[\boldsymbol{u}, \boldsymbol{p}]\right)$ $\quad$ $\hat{y} = \boldsymbol{W}_y \boldsymbol{v}$ $\quad$ $L = (\hat{y} - y)^2$

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

Dealing with discrete latent variables

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

Dealing with discrete latent variables

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

Roadmap of the tutorial

- Part 1: Introduction ✓
- Part 2: Reinforcement learning
- 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 $\mathbf{z}$,

Latent structure via marginalization

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

Latent structure via marginalization

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

Latent structure via marginalization

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

$$L(\hat{y}(\boldsymbol{z}; x), y) \text{ or simply } L(\boldsymbol{z})$$

Latent structure via marginalization

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

$$L(\hat{y}(\boldsymbol{z}; x), y) \text{ or simply } L(\boldsymbol{z})$$

Latent structure via marginalization

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

- But we don't know $\boldsymbol{z}$!

Latent structure via marginalization

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

$$L(\hat{y}(\boldsymbol{z}; x), y) \text{ or simply } L(\boldsymbol{z})$$

- But we don't know $\boldsymbol{z}$!
- In this section:
  we jointly learn a structured prediction model $\pi_{\boldsymbol{\theta}}(\boldsymbol{z} \mid x)$

Latent structure via marginalization

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

$$L(\hat{y}(\boldsymbol{z}; x), y) \text{ or simply } L(\boldsymbol{z})$$

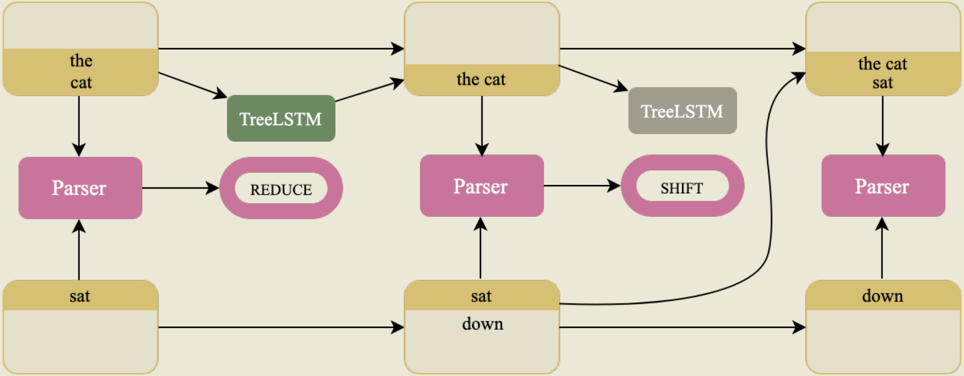
- But we don't know $\boldsymbol{z}$!
- In this section:
  we jointly learn a structured prediction model $\pi_{\boldsymbol{\theta}}(\boldsymbol{z} \mid x)$
  by optimizing the **expected loss**,

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

But first, supervised SPINN

[Bowman et al., 2016]

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

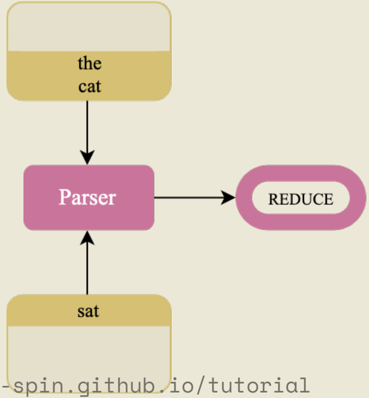
- Joint learning: Combines a constituency parser and a sentence representation model.
- The parser, $f_{\boldsymbol{\theta}}(x)$ is a transition-based **shift-reduce** parser. It looks at top two elements of stack and top element of the buffer.

[Bowman et al., 2016]

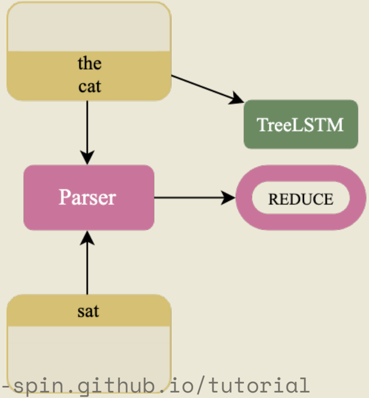
Stack-augmented Parser-Interpreter Neural-Network

- Joint learning: Combines a constituency parser and a sentence representation model.
- The parser, $f_{\boldsymbol{\theta}}(x)$ 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.

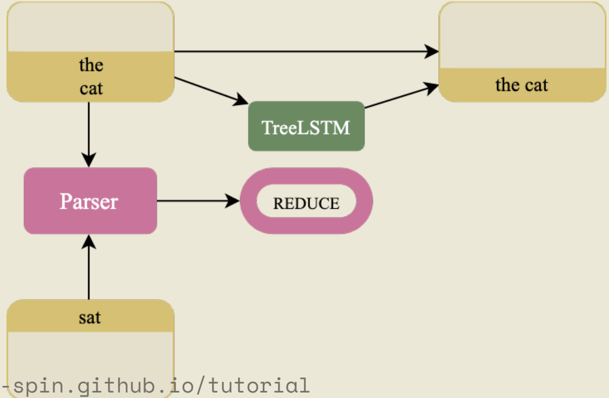
Stack-augmented Parser-Interpreter Neural-Network



Stack-augmented Parser-Interpreter Neural-Network

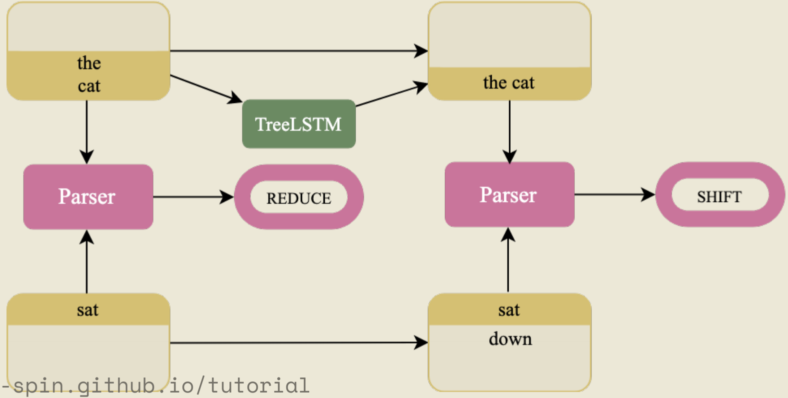


Stack-augmented Parser-Interpreter Neural-Network



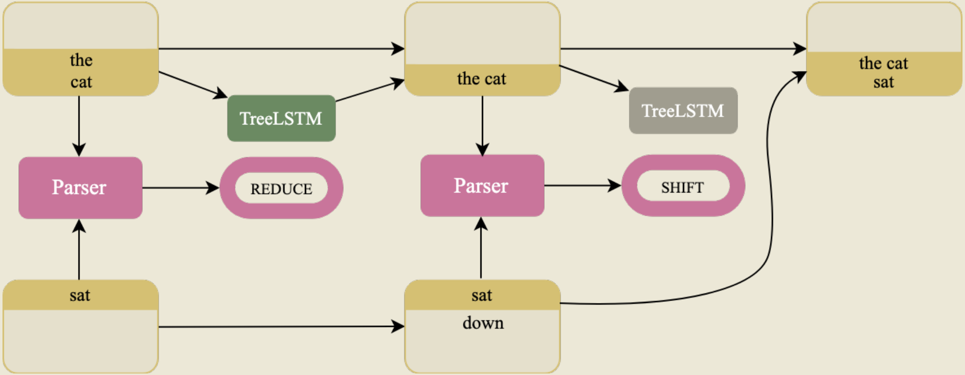
Stack-augmented Parser-Interpreter Neural-Network

[Bowman et al., 2016]

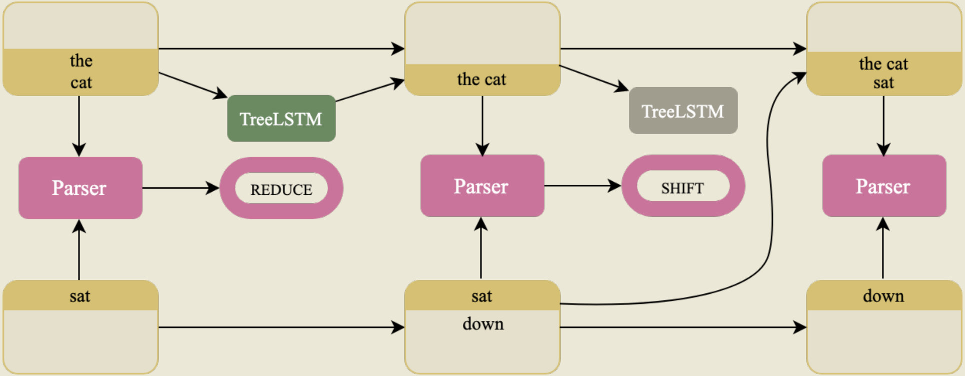


[Bowman et al., 2016]

Stack-augmented Parser-Interpreter Neural-Network



Stack-augmented Parser-Interpreter Neural-Network



Shift-Reduce parsing

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

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

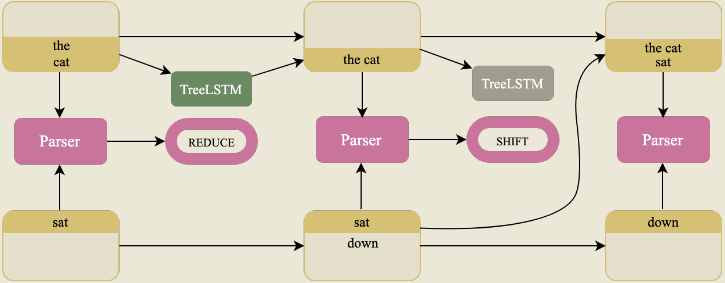
where, $z_j \in \{0, 1\} \ \forall j \in [1, 2L-1]$

Shift-Reduce parsing

A sequence of Bernoulli trials but with conditional dependence,

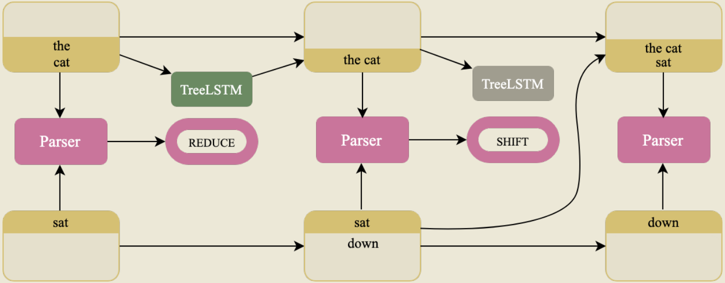
$$p(z_1, z_2, \ldots, z_{2L-1}) = \prod_{j=1}^{2L-1} p(z_j \mid z_{<j})$$

Latent structure learning with SPINN



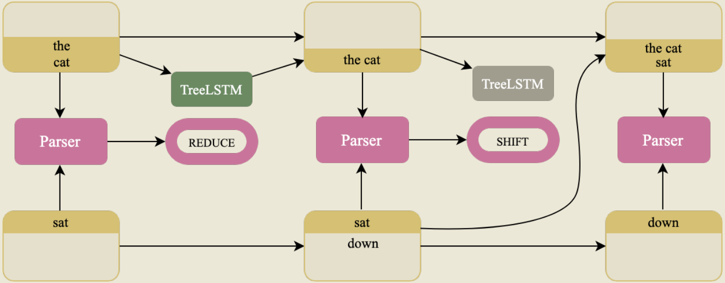
Latent structure learning with SPINN

- But now, remove syntactic supervision from SPINN.



Latent structure learning with SPINN

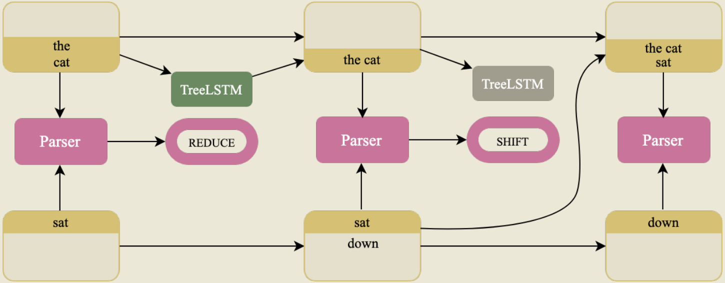
- But now, remove syntactic supervision from SPINN.



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

Latent structure learning with SPINN

- But now, remove syntactic supervision from SPINN.



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

SPINN with REINFORCE

Some basic terminology,

- The action space is $z_j \in \{\textsc{shift}, \textsc{reduce}\}$, and $\boldsymbol{z}$ is a sequence of actions.

[Williams, 1992]

SPINN with REINFORCE

Some basic terminology,

- The action space is $z_j \in \{\textsc{shift}, \textsc{reduce}\}$, and $\boldsymbol{z}$ is a sequence of actions.
- Training parser network parameters, $\boldsymbol{\theta}$ with REINFORCE

SPINN with REINFORCE

Some basic terminology,

- The action space is $z_j \in \{\textsc{shift}, \textsc{reduce}\}$, and $\boldsymbol{z}$ is a sequence of actions.
- Training parser network parameters, $\boldsymbol{\theta}$ with REINFORCE
- The state, $\boldsymbol{h}$, is the top two elements of the stack and the top element of the buffer.

[Williams, 1992]

SPINN with REINFORCE

Some basic terminology,

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

SPINN with REINFORCE

Some basic terminology,

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

[Williams, 1992]

SPINN with REINFORCE

Some basic terminology,

- The action space is $z_j \in \{\textsc{shift}, \textsc{reduce}\}$, and $\boldsymbol{z}$ is a sequence of actions.
- Training parser network parameters, $\boldsymbol{\theta}$ with REINFORCE
- The state, $\boldsymbol{h}$, is the top two elements of the stack and the top element of the buffe
- Lear
- Maxi ke sentence classification.

NOTE: Only a single reward at the end of parsing.

Through the looking glass of REINFORCE

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

Through the looking glass of REINFORCE

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

(By definition of expectation. How to evaluate?)

Through the looking glass of REINFORCE

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

(By definition of expectation. How to evaluate?)

$$= \sum_{\boldsymbol{z}} L(\boldsymbol{z}) \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\boldsymbol{z} \mid x)$$

Through the looking glass of REINFORCE

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

(By definition of expectation. How to evaluate?)

$$= \sum_{\boldsymbol{z}} L(\boldsymbol{z}) \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\boldsymbol{z} \mid x)$$

$$= \sum_{\boldsymbol{z}} L(\boldsymbol{z}) \pi_{\boldsymbol{\theta}}(\boldsymbol{z} \mid x) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{z} \mid x)$$

(By Leibniz integral rule for $\log$*)*

Through the looking glass of REINFORCE

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

(By definition of expectation. How to evaluate?)

$$= \sum_{\mathbf{z}} L(\mathbf{z}) \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\mathbf{z} \mid x)$$

$$= \sum_{\mathbf{z}} L(\mathbf{z}) \pi_{\boldsymbol{\theta}}(\mathbf{z} \mid x) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{z} \mid x)$$

(By Leibniz integral rule for $\log$ *)*

$$= \mathbb{E}_{\mathbf{z} \sim \pi_{\boldsymbol{\theta}}(\mathbf{z}|x)}[L(\mathbf{z}))\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{z} \mid x)]$$

SPINN with REINFORCE, aka RL-SPINN

Yogatama et al. [2017] uses REINFORCE to train SPINN!

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.

SPINN with REINFORCE, aka RL-SPINN

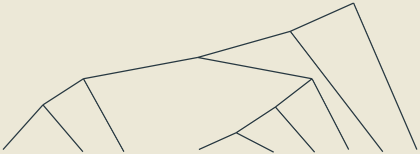
Yogatama et al. [2017] uses REINFORCE to train SPINN!

However, this vanilla implementation isn't very effective at learning syntax.

This model fails to solve a simple toy problem.

[Nangia and Bowman, 2018]

Toy problem: ListOps



[max 2 9 [min 4 7] 0]

Toy problem: ListOps

Model	Accuracy $\mu(\sigma)$	Accuracy max	Self F1
LSTM	71.5 (1.5)	**74.4**	-
RL-SPINN	60.7 (2.6)	64.8	30.8
Random Trees	-	-	30.1

Model	F1 wrt. LB	F1 wrt. RB	F1 wrt. GT	Avg. Depth
48D RL-SPINN	**64.5**	**16.0**	32.1	**14.6**
128D RL-SPINN	43.5	13.0	**71.1**	10.4
GT Trees	41.6	8.8	100.0	9.6
Random Trees	24.0	24.0	24.2	5.2

[Nangia and Bowman, 2018]

Toy problem: ListOps

Model	Accuracy $\mu(\sigma)$	Accuracy max	Self F1
LSTM	71.5 (1.5)	**74.4**	-
RL-SPINN	60.7 (2.6)	64.8	30.8
Random Trees	-	-	

But why?

	F1 wrt. LB	F1 wrt. RB	F1 wrt. GT	Avg. Depth
	4.5	**16.0**	32.1	**14.6**
128D RL-SPINN	43.5	13.0	**71.1**	10.4
GT Trees	41.6	8.8	100.0	9.6
Random Trees	24.0	24.0	24.2	5.2

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.

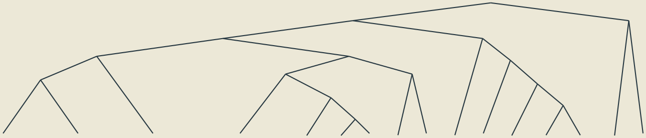
$$\begin{aligned} 3 \text{ tokens} &\Rightarrow 5 \text{ trees} \\ 5 \text{ tokens} &\Rightarrow 42 \text{ trees} \\ 10 \text{ tokens} &\Rightarrow 16796 \text{ trees} \end{aligned}$$

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”:



[sm [sm [sm [max 5 6] 2] 0] 5 0 8 6]

Figure: Truth: 7; Pred: 7

High variance

Sometimes it doesn't:

[max [med [med 1 [sm 3 1 3] 9] 6] 5]

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, $\mathcal{R}$, 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, $\mathcal{R}$, and the baseline prediction.

  So,

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

Control variates

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

  So,

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

  Which we can do because,

$$\sum_{z} b(\boldsymbol{x}) \nabla \pi(\boldsymbol{z}) = b(\boldsymbol{x}) \sum_{z} \nabla \pi(\boldsymbol{z}) = b(\boldsymbol{x}) \nabla 1 = 0$$

Issues with SPINN with REINFORCE

This system faces two big problems,

1. High variance of gradients
2. Coadaptation

Coadaptation problem

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

Coadaptation problem

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

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

Coadaptation problem

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

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

Difference in variance of two gradient estimates.

Coadaptation problem

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

Generally, $\boldsymbol{\phi}$ will be learned more quickly than $\boldsymbol{\theta}$,

ma

Dif

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 Optmization

Making REINFORCE+SPINN work

Havrylov et al. [2019] use,

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

They solve ListOps!

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**

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

So far:

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

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

- Optimized with the **REINFORCE** estimator.

So far:

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

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

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

So far:

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

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

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

In this section:

- Consider the **deterministic alternative**:

So far:

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

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

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

In this section:

- Consider the **deterministic alternative**:

  pick "best" structure $\hat{\boldsymbol{z}}(x) := \arg\max_{\boldsymbol{z}\in\mathcal{M}} \pi_{\boldsymbol{\theta}}(\boldsymbol{z} \mid x)$

So far:

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

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

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

In this section:

- Consider the **deterministic alternative**:

  pick “best” structure $\hat{\mathbf{z}}(x) := \arg\max_{\mathbf{z}\in\mathcal{M}} \pi_{\boldsymbol{\theta}}(\mathbf{z} \mid x)$

  incur loss $L\big(\hat{\mathbf{z}}(x)\big)$

So far:

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

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

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

In this section:

- Consider the **deterministic alternative**:
  pick "best" structure $\hat{\boldsymbol{z}}(x) := \arg\max_{\boldsymbol{z}\in\mathcal{M}} \pi_{\boldsymbol{\theta}}(\boldsymbol{z} \mid x)$
  incur loss $L\big(\hat{\boldsymbol{z}}(x)\big)$
- 3A: try to optimize the deterministic loss directly

So far:

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

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

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

In this section:

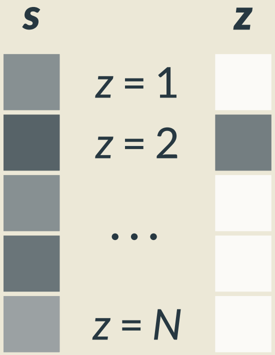
- Consider the **deterministic alternative**:

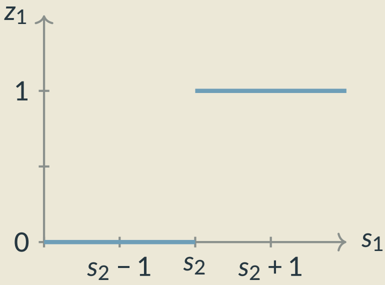
  pick “best” structure $\hat{\boldsymbol{z}}(x) := \arg\max_{\boldsymbol{z}\in\mathcal{M}} \pi_{\boldsymbol{\theta}}(\boldsymbol{z} \mid x)$

  incur loss $L\big(\hat{\boldsymbol{z}}(x)\big)$
- 3A: try to optimize the deterministic loss directly
- 3B: use this strategy to reduce variance in the stochastic model.

Recap: The argmax problem

$$\boldsymbol{z} = \arg\max(\boldsymbol{s})$$

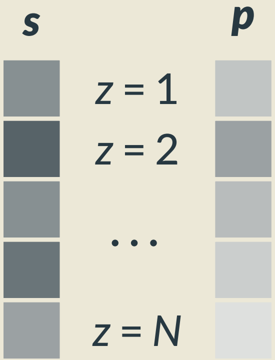


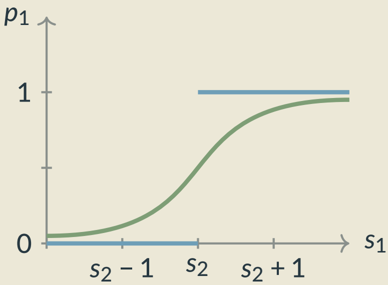


$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{s}} = \boldsymbol{0}$$

Softmax

$$p_j = \exp(s_j)/Z$$



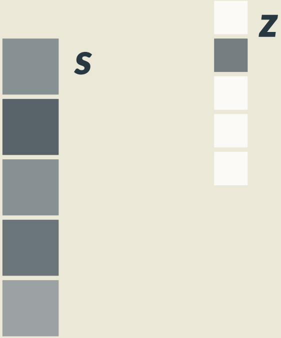


$$\frac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}} = \operatorname{diag}(\boldsymbol{p}) - \boldsymbol{p}\boldsymbol{p}^\top$$

Straight-Through Estimator

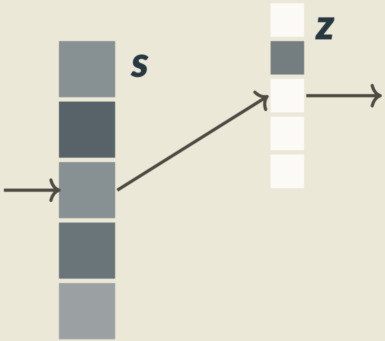
Straight-Through Estimator

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



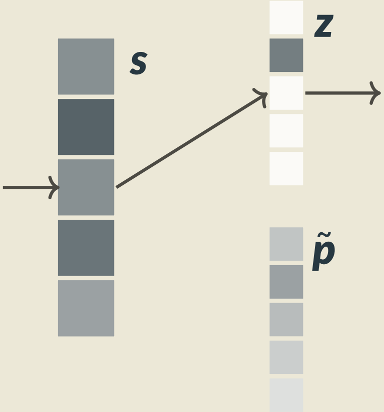
Straight-Through Estimator

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



Straight-Through Estimator

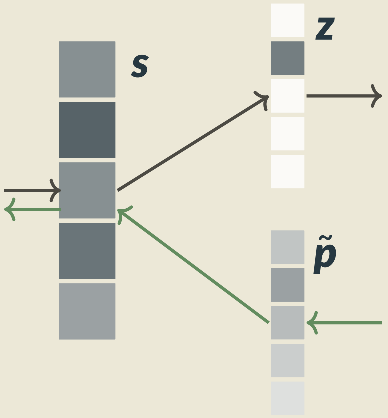
- Forward: $\boldsymbol{z} = \arg\max(\boldsymbol{s})$
- Backward: pretend $\boldsymbol{z}$ was some continuous $\tilde{\boldsymbol{p}}$; $\frac{\partial \tilde{\boldsymbol{p}}}{\partial \boldsymbol{s}} \neq \boldsymbol{0}$



[Hinton, 2012, Bengio et al., 2013]

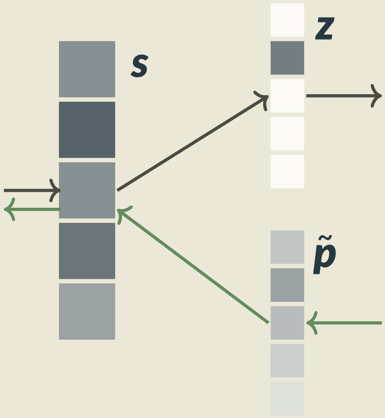
Straight-Through Estimator

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



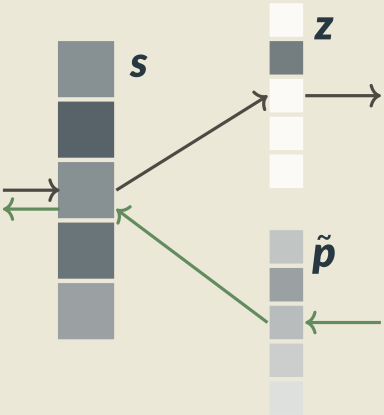
Straight-Through Estimator

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



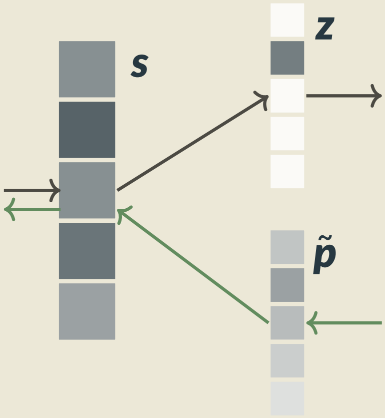
Straight-Through Estimator

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



Straight-Through Estimator

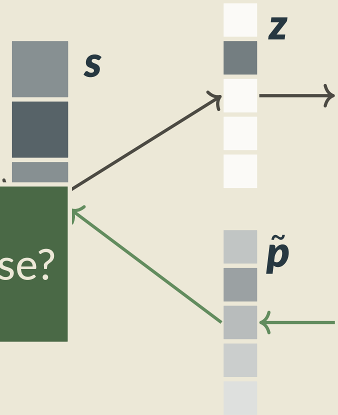
- Forward: $\boldsymbol{z} = \arg\max(\boldsymbol{s})$
- Backward: pretend $\boldsymbol{z}$ was some continuous $\tilde{\boldsymbol{p}}$; $\frac{\partial \tilde{\boldsymbol{p}}}{\partial \boldsymbol{s}} \neq \boldsymbol{0}$
  - simplest: identity, $\tilde{\boldsymbol{p}}(\boldsymbol{s}) = \boldsymbol{s}$,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



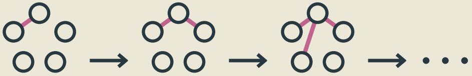
Straight-Through Estimator

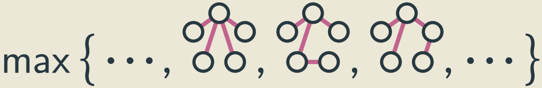
- Forward: $\boldsymbol{z} = \arg\max(\boldsymbol{s})$
- Backward: pretend $\boldsymbol{z}$ was some continuous $\tilde{\boldsymbol{p}}$; $\frac{\partial \tilde{\boldsymbol{p}}}{\partial \boldsymbol{s}} \neq \boldsymbol{0}$
  - simplest: identity, $\tilde{\boldsymbol{p}}(\boldsymbol{s}) = \boldsymbol{s}$,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

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

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

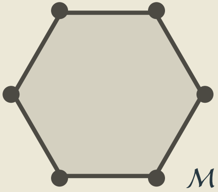
- Build a structure as a sequence of discrete choices (e.g., shift-reduce)
- Assigns a score to any (partial structure, action) tuple.
- In this case, we just apply the straight-through estimator for each step.
- Forward: the **highest scoring action** for each step
- Backward: pretend that we had used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

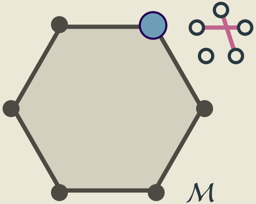
  Example: Latent Tree Learning with Differentiable Parsers: Shift-Reduce Parsing and Chart Parsing [Maillard and Clark, 2018] (STE through beam search).

The structured case: Marginal polytope



The structured case: Marginal polytope

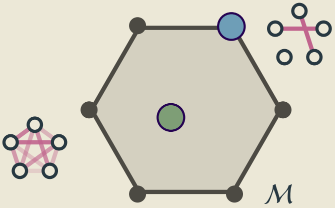
- Each vertex corresponds to one such *bit vector* $\boldsymbol{z}$



[Wainwright and Jordan, 2008]

The structured case: Marginal polytope

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

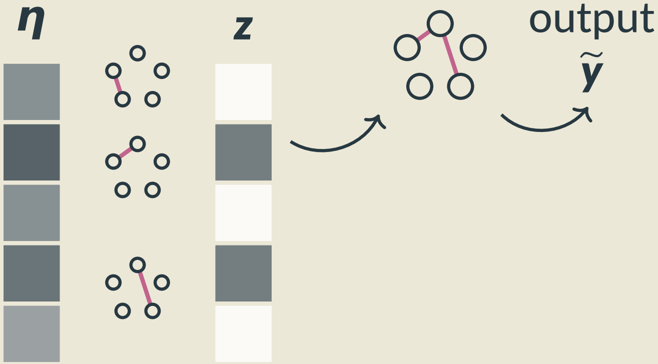


$$\boldsymbol{\mu} = \underbrace{p_1\boldsymbol{z}_1 + \ldots + p_N\boldsymbol{z}_N}_{\text{exponentially many terms}}, \ \boldsymbol{p} \in \Delta.$$

$$\begin{aligned} p_1 &= 0.2, & \boldsymbol{z}_1 &= [1, 0, 0, 0, 1, 0, 0, 0, 1] \\ p_2 &= 0.7, & \boldsymbol{z}_2 &= [0, 0, 1, 0, 0, 1, 1, 0, 0] \\ p_3 &= 0.1, & \boldsymbol{z}_3 &= [1, 0, 0, 0, 1, 0, 0, 1, 0] \end{aligned} \quad \Rightarrow \quad \boldsymbol{\mu} = [.3, 0, .7, 0, .3, .7, .7, .1, .2].$$

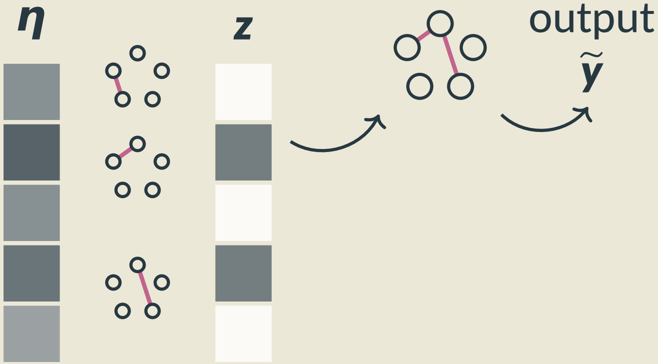
STE for factorization into parts

- $\eta(i \to j)$: score of arc $i \to j$
- $z(i \to j)$: is arc $i \to j$ selected?



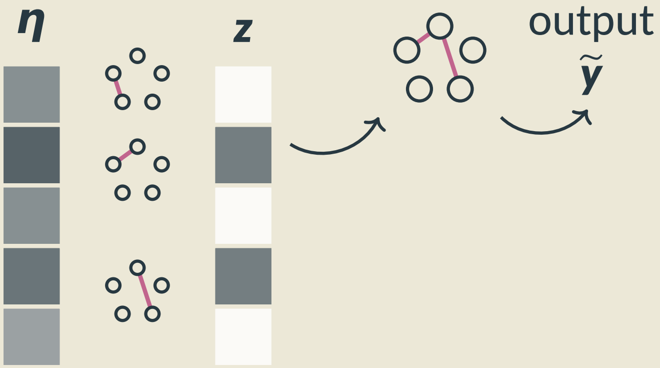
STE for factorization into parts

- $\eta(i \rightarrow j)$: score of arc $i \rightarrow j$
- $z(i \rightarrow j)$: is arc $i \rightarrow j$ selected?
- Forward: structured argmax (task-specific algorithm!)



STE for factorization into parts

- $\eta(i \to j)$: score of arc $i \to j$
- $z(i \to j)$: is arc $i \to j$ selected?
- Forward: structured argmax (task-specific algorithm!)
- Backward: identity $\frac{\partial \tilde{\boldsymbol{\mu}}}{\partial \boldsymbol{\eta}} = \boldsymbol{I}$



Algorithms for specific structures

	Best structure (MAP)
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]

Straight-Through Estimator

Revisited

Straight-Through Estimator

Revisited

- In the forward pass, $\boldsymbol{z} = \arg\max(\boldsymbol{s})$.

Straight-Through Estimator

Revisited

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

[Martins and Niculae]

Straight-Through Estimator

Revisited

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

Straight-Through Estimator

Revisited

- In the forward pass, $\boldsymbol{z} = \arg\max(\boldsymbol{s})$.
- if we had labels (multi-task learning), $L_{\text{MTL}} = L\big(\hat{y}(\boldsymbol{z}), y\big) + L_{\text{hid}}(\boldsymbol{s}, \boldsymbol{z}^{\text{true}})$
- One choice: perceptron loss $L_{\text{hid}}(\boldsymbol{s}, \boldsymbol{z}^{\text{true}}) = \boldsymbol{s}^\top \boldsymbol{z} - \boldsymbol{s}^\top \boldsymbol{z}^{\text{true}}$, $\quad \frac{\partial L_{\text{hid}}}{\partial \boldsymbol{s}} = \boldsymbol{z} - \boldsymbol{z}^{\text{true}}$.
- We don't have labels! Induce labels by "pulling back" the downstream target:
  the "best" (unconstrained) latent value would be: $\quad \arg\min_{\boldsymbol{z}} L\big(\hat{y}(\boldsymbol{z}), y\big)$

[Martins and Niculae]

Straight-Through Estimator

Revisited

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

Straight-Through Estimator

Revisited

- In the forward pass, $\boldsymbol{z} = \arg\max(\boldsymbol{s})$.
- if we had labels (multi-task learning), $L_{\text{MTL}} = L\big(\hat{y}(\boldsymbol{z}), y\big) + L_{\text{hid}}\big(\boldsymbol{s}, \boldsymbol{z}^{\text{true}}\big)$
- One choice: perceptron loss $L_{\text{hid}}\big(\boldsymbol{s}, \boldsymbol{z}^{\text{true}}\big) = \boldsymbol{s}^\top \boldsymbol{z} - \boldsymbol{s}^\top \boldsymbol{z}^{\text{true}}$; $\quad \frac{\partial L_{\text{hid}}}{\partial \boldsymbol{s}} = \boldsymbol{z} - \boldsymbol{z}^{\text{true}}$.
- We don't have labels! Induce labels by "pulling back" the downstream target:
  the "best" (unconstrained) latent value would be: $\arg\min_{\boldsymbol{z}} L\big(\hat{y}(\boldsymbol{z}), y\big)$
- One gradient descent step starting from $\boldsymbol{z}$: $\boldsymbol{z}^{\text{true}} \leftarrow \boldsymbol{z} - \frac{\partial L}{\partial \boldsymbol{z}}$

$$\frac{\partial L_{\text{MTL}}}{\partial \boldsymbol{s}} = \underbrace{\frac{\partial L}{\partial \boldsymbol{s}}}_{=\boldsymbol{0}} + \frac{\partial L_{\text{hid}}}{\partial \boldsymbol{s}}$$

Straight-Through Estimator

Revisited

- In the forward pass, $\boldsymbol{z} = \arg\max(\boldsymbol{s})$.
- if we had labels (multi-task learning), $L_{\text{MTL}} = L\big(\hat{y}(\boldsymbol{z}), y\big) + L_{\text{hid}}(\boldsymbol{s}, \boldsymbol{z}^{\text{true}})$
- One choice: perceptron loss $L_{\text{hid}}(\boldsymbol{s}, \boldsymbol{z}^{\text{true}}) = \boldsymbol{s}^\top \boldsymbol{z} - \boldsymbol{s}^\top \boldsymbol{z}^{\text{true}}$; $\quad \frac{\partial L_{\text{hid}}}{\partial \boldsymbol{s}} = \boldsymbol{z} - \boldsymbol{z}^{\text{true}}$.
- We don't have labels! Induce labels by "pulling back" the downstream target: the "best" (unconstrained) latent value would be: $\arg\min_{\boldsymbol{z}} L\big(\hat{y}(\boldsymbol{z}), y\big)$
- One gradient descent step starting from $\boldsymbol{z}$: $\boldsymbol{z}^{\text{true}} \leftarrow \boldsymbol{z} - \frac{\partial L}{\partial \boldsymbol{z}}$

$$\frac{\partial L_{\text{MTL}}}{\partial \boldsymbol{s}} = \underbrace{\frac{\partial L}{\partial \boldsymbol{s}}}_{=0} + \frac{\partial L_{\text{hid}}}{\partial \boldsymbol{s}} = \boldsymbol{z} - \left(\boldsymbol{z} - \frac{\partial L}{\partial \boldsymbol{z}}\right) = \frac{\partial L}{\partial \boldsymbol{z}}$$

[Peng et al., 2018, Martins and Niculae]

Straight Through in the structured case

- Structured STE: perceptron update with fake annotation

$$\arg\min_{\boldsymbol{\mu}\in\mathbb{R}^d} L(\hat{y}(\boldsymbol{\mu}), y) \qquad \approx \boldsymbol{z} - \nabla_{\boldsymbol{z}} L(\boldsymbol{z}) \rightarrow \boldsymbol{z}^{\text{true}}$$

(one step of gradient descent)

Straight Through in the structured case

- Structured STE: perceptron update with fake annotation

$$\arg\min_{\boldsymbol{\mu}\in\mathbb{R}^d} L(\hat{y}(\boldsymbol{\mu}), y) \qquad \approx \boldsymbol{z} - \nabla_{\boldsymbol{z}} L(\boldsymbol{z}) \longrightarrow \boldsymbol{z}^{\text{true}}$$

(one step of gradient descent)

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

$$\arg\min_{\boldsymbol{\mu}\in\mathcal{M}} L(\hat{y}(\boldsymbol{\mu}), y) \qquad \approx \operatorname{Proj}_{\mathcal{M}}\left(\boldsymbol{z} - \nabla_{\boldsymbol{z}} L(\boldsymbol{z})\right) \longrightarrow \boldsymbol{z}^{\text{true}}$$

(one step of *projected* gradient descent!)

Straight Through in the structured case

- Structured STE: perceptron update with fake annotation

$$\arg\min_{\boldsymbol{\mu}\in\mathbb{R}^d} L(\hat{y}(\boldsymbol{\mu}), y) \qquad \approx \boldsymbol{z} - \nabla_{\boldsymbol{z}} L(\boldsymbol{z}) \to \boldsymbol{z}^{\text{true}}$$

(one step of gradient descent)

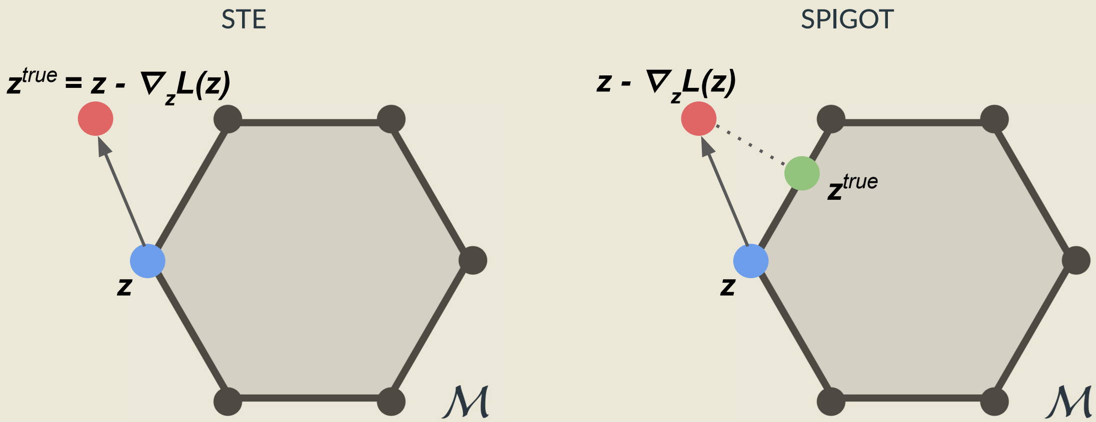
- SPIGOT takes into account the constraints; uses the fake annotation

$$\arg\min_{\boldsymbol{\mu}\in\mathcal{M}} L(\hat{y}(\boldsymbol{\mu}), y) \quad \approx \operatorname{Proj}_{\mathcal{M}}\left(\boldsymbol{z} - \nabla_{\boldsymbol{z}} L(\boldsymbol{z})\right) \to \boldsymbol{z}^{\text{true}}$$

(one step of *projected* gradient descent!)

- We discuss a generic way to compute the projection in part 3.

SPIGOT vs STE



Summary: Straight-Through Estimator

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.

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 $L(\hat{\boldsymbol{z}}(x))$

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 $$L\big(\hat{\boldsymbol{z}}(x)\big)$$

Now we will see how to apply STE for stochastic graphs, as an alternative approach of the score-function estimators.

Stochastic node in the computation graph

Stochastic node in the computation graph

Recall the stochastic objective:

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

Stochastic node in the computation graph

Recall the stochastic objective:

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

- REINFORCE (previous section).

Stochastic node in the computation graph

Recall the stochastic objective:

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

- REINFORCE (previous section). High variance. 😟

Stochastic node in the computation graph

Recall the stochastic objective:

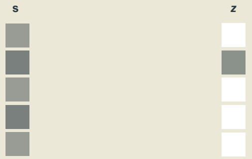
$$\mathbb{E}_{\pi_{\boldsymbol{\theta}}(\boldsymbol{z}|x)}\left[L(\boldsymbol{z})\right]$$

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

Categorical reparameterization

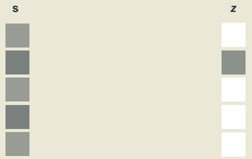
Categorical reparameterization

- Sampling from a categorical value in the middle of the computation graph.
  $\boldsymbol{z} \sim \pi_{\boldsymbol{\theta}}(\boldsymbol{z} \mid x) \propto \exp \boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{z} \mid x)$



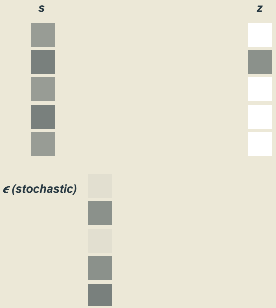
Categorical reparameterization

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



Categorical reparameterization

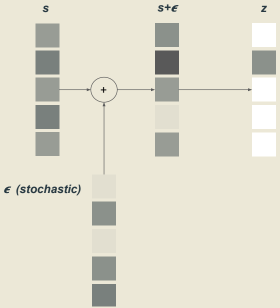
- Sampling from a categorical value in the middle of the computation graph.
  $\boldsymbol{z} \sim \pi_{\boldsymbol{\theta}}(\boldsymbol{z} \mid x) \propto \exp \boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{z} \mid x)$
- What is the gradient of a sample $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\theta}}$?!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



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

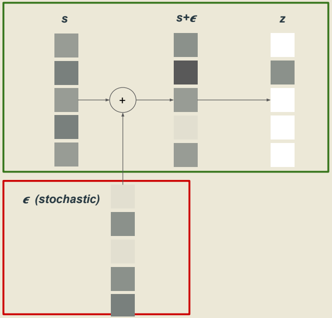
Categorical reparameterization

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



Categorical reparameterization

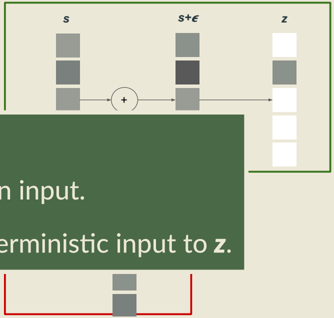
- Sampling from a categorical value in the middle of the computation graph. $\boldsymbol{z} \sim \pi_{\boldsymbol{\theta}}(\boldsymbol{z} \mid x) \propto \exp \boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{z} \mid x)$
- What is the gradient of a sample $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\theta}}$?!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



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

Categorical reparameterization

- Sampling from a categorical value in the middle of the computation graph. $\boldsymbol{z} \sim \pi_{\boldsymbol{\theta}}(\boldsymbol{z} \mid x) \propto \exp \boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{z} \mid x)$
- What is the g
- Reparameter stochasticity
- Makes $\boldsymbol{z}$ deterministic w.r.t. $\boldsymbol{s}$!



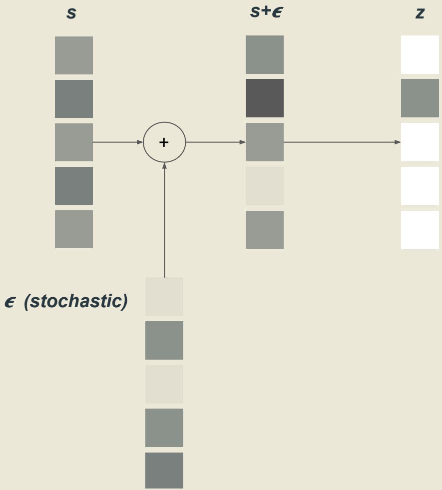
As a result:

Stochasticity is moved as an input.

We can backpropagate through the deterministic input to $\boldsymbol{z}$.

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

Categorical reparameterization



Sampling from a categorical variable

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

Sampling from a categorical variable

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

Sampling from a categorical variable

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

1. Inverse tranform sampling:

Sampling from a categorical variable

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

1. Inverse tranform sampling:

- $\boldsymbol{p} = \mathrm{softmax}(\boldsymbol{s})$

Sampling from a categorical variable

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

1. Inverse tranform sampling:

- $\boldsymbol{p} = \mathrm{softmax}(\boldsymbol{s})$
- $c_i = \sum_{j \leq i} p_i$

Sampling from a categorical variable

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

1. Inverse tranform sampling:

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

Sampling from a categorical variable

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

1. Inverse tranform sampling:

- $\boldsymbol{p} = \mathrm{softmax}(\boldsymbol{s})$
- $c_i = \sum_{j \leq i} p_i$
- $u \sim \mathrm{Uniform}(0, 1)$
- return $\boldsymbol{z} = \boldsymbol{e}_t$ s.t. $c_t \leq u < c_{t+1}$

Sampling from a categorical variable

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

1. Inverse tranform sampling:

- $\boldsymbol{p} = \text{softmax}(\boldsymbol{s})$
- $c_i = \sum_{j \leq i} p_i$
- $u \sim \text{Uniform}(0, 1)$
- return $\boldsymbol{z} = \boldsymbol{e}_t$ s.t. $c_t \leq u < c_{t+1}$

2. The Gumbel-max trick

Sampling from a categorical variable

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

1. Inverse tranform sampling:

- $\boldsymbol{p} = \text{softmax}(\boldsymbol{s})$
- $c_i = \sum_{j \leq i} p_i$
- $u \sim \text{Uniform}(0, 1)$
- return $\boldsymbol{z} = \boldsymbol{e}_t$ s.t. $c_t \leq u < c_{t+1}$

2. The Gumbel-max trick

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

Sampling from a categorical variable

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

1. Inverse tranform sampling:

- $\boldsymbol{p} = \text{softmax}(\boldsymbol{s})$
- $c_i = \sum_{j \leq i} p_i$
- $u \sim \text{Uniform}(0, 1)$
- return $\boldsymbol{z} = \boldsymbol{e}_t$ s.t. $c_t \leq u < c_{t+1}$

2. The Gumbel-max trick

- $u_i \sim \text{Uniform}(0, 1)$
- $\epsilon_i = -\log(-\log(u_i))$

Sampling from a categorical variable

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

1. Inverse tranform sampling:

- $\boldsymbol{p} = \mathrm{softmax}(\boldsymbol{s})$
- $c_i = \sum_{j \leq i} p_i$
- $u \sim \mathrm{Uniform}(0, 1)$
- return $\boldsymbol{z} = \boldsymbol{e}_t$ s.t. $c_t \leq u < c_{t+1}$

2. The Gumbel-max trick

- $u_i \sim \mathrm{Uniform}(0, 1)$
- $\epsilon_i = -\log(-\log(u_i))$
- $\boldsymbol{z} = \arg\max(\boldsymbol{s} + \boldsymbol{\epsilon})$

Sampling from a categorical variable

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

1. Inverse tranform sampling:

- $\boldsymbol{p} = \text{softmax}(\boldsymbol{s})$
- $c_i = \sum_{j \leq i} p_i$
- $u \sim \text{Uniform}(0, 1)$
- return $\boldsymbol{z} = \boldsymbol{e}_t$ s.t. $c_t \leq u < c_{t+1}$

2. The Gumbel-max trick

- $u_i \sim \text{Uniform}(0, 1)$
- $\epsilon_i = -\log(-\log(u_i))$
- $\boldsymbol{z} = \arg\max(\boldsymbol{s} + \boldsymbol{\epsilon})$

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 $\boldsymbol{s}$ (class i has a score s_i)

1. Inverse tranform sampling:

- $\boldsymbol{p} = \mathrm{softmax}(\boldsymbol{s})$
- $c_i = \sum_{j \le i} p_i$
- $u \sim \mathrm{Uniform}(0, 1)$
- return $\boldsymbol{z} = \boldsymbol{e}_t$ s.t. $c_t \le u < c_{t+1}$

2. The Gumbel-max trick

- $u_i \sim \mathrm{Uniform}(0, 1)$
- $\epsilon_i = -\log(-\log(u_i))$
- $\boldsymbol{z} = \arg\max(\boldsymbol{s} + \boldsymbol{\epsilon})$

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 $\boldsymbol{s}$ (class i has a score s_i)

1. Inverse tranform sampling:

- $\boldsymbol{p} = \text{softmax}(\boldsymbol{s})$
- $c_i = \sum_{j \leq i} p_i$
- $u \sim \text{Uniform}(0, 1)$
- return $\boldsymbol{z} = \boldsymbol{e}_t$ s.t. $c_t \leq u < c_{t+1}$

2. The Gumbel-max trick

- $u_i \sim \text{Uniform}(0, 1)$
- $\epsilon_i = -\log(-\log(u_i))$
- $\boldsymbol{z} = \arg\max(\boldsymbol{s} + \boldsymbol{\epsilon})$

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 $\boldsymbol{s}$ (class i has a score s_i)

1. Inverse tranform sampling:

- $\boldsymbol{p} = \text{softmax}(\boldsymbol{s})$
- $c_i = \sum_{j \leq i} p_i$
- $u \sim \text{Uniform}(0, 1)$
- return $\boldsymbol{z} = \boldsymbol{e}_t$ s.t. $c_t \leq u < c_{t+1}$

2. The Gumbel-max trick

- $u_i \sim \text{Uniform}(0, 1)$
- $\epsilon_i = -\log(-\log(u_i))$
- $\boldsymbol{z} = \arg\max(\boldsymbol{s} + \boldsymbol{\epsilon})$

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]

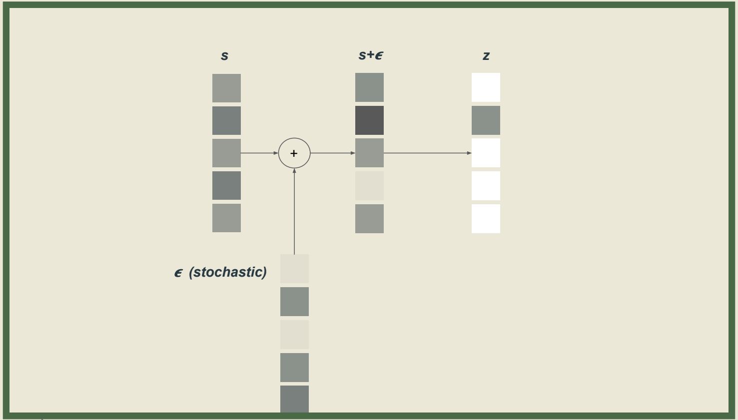
Sampling from a categorical variable

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

1. Inverse tranform sampling:

- $\boldsymbol{p} = \text{softmax}(\boldsymbol{s}$
- $c_i = \sum_{j \leq i} p_i$
- $u \sim \text{Uniform}($
- return $\boldsymbol{z} = \boldsymbol{e}_t$

2. The Gumbel-max trick



The t ... *now.)*

I ...).

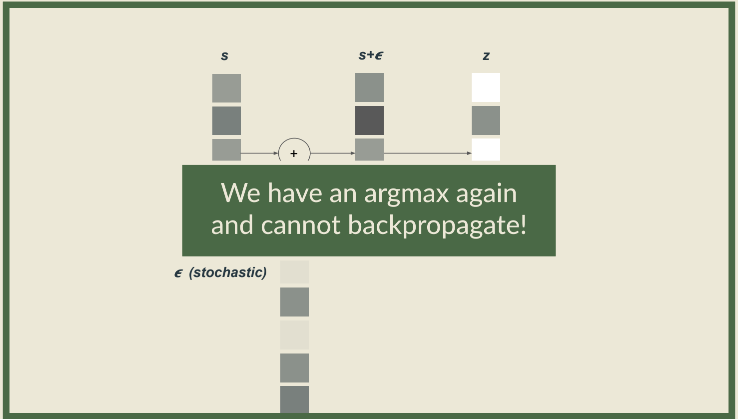
Sampling from a categorical variable

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

1. Inverse tranform sampling:

- $\boldsymbol{p} = \text{softmax}(\boldsymbol{s}$
- $c_i = \sum_{j \leq i} p_i$
- $u \sim \text{Uniform}($
- return $\boldsymbol{z} = \boldsymbol{e}_t$

2. The Gumbel-max trick



The t … now.)

…).

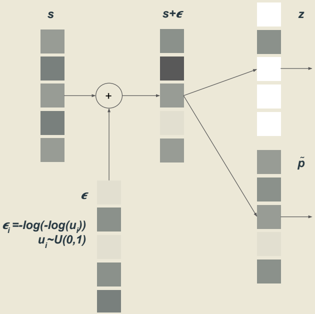
[Jang et al., 2017, Maddison et al., 2016]

Straight-Through Gumbel Estimator

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

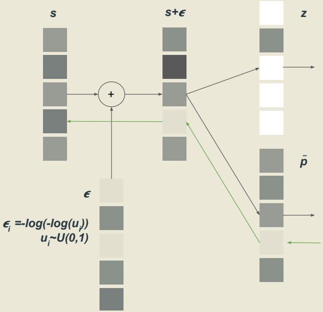
Straight-Through Gumbel Estimator

- Apply a variant of the Straight-Through Estimator to Gumbel-Max!
- Forward: $\boldsymbol{z} = \arg\max(\boldsymbol{s} + \boldsymbol{\epsilon})$



Straight-Through Gumbel Estimator

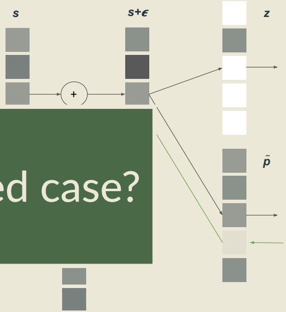
- Apply a variant of the Straight-Through Estimator to Gumbel-Max!
- Forward: $\boldsymbol{z} = \arg\max(\boldsymbol{s} + \boldsymbol{\epsilon})$
- Backward: pretend we had done $\tilde{\boldsymbol{p}} = \operatorname{softmax}(\boldsymbol{s} + \boldsymbol{\epsilon})$



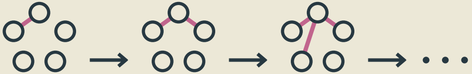
Straight-Through Gumbel Estimator

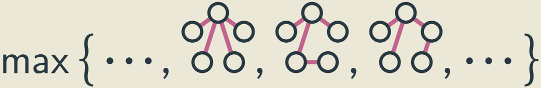
- Apply a variant of the Straight-Through Estimator to Gumbel-Max!
- Forward: $\boldsymbol{z}$ = arg
- Backward: prete
  $\tilde{\boldsymbol{p}} = \text{softmax}(\boldsymbol{s} + \boldsymbol{\epsilon}$

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.

Sampling from incremental structures

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 that we had used a **differentiable surrogate function**s

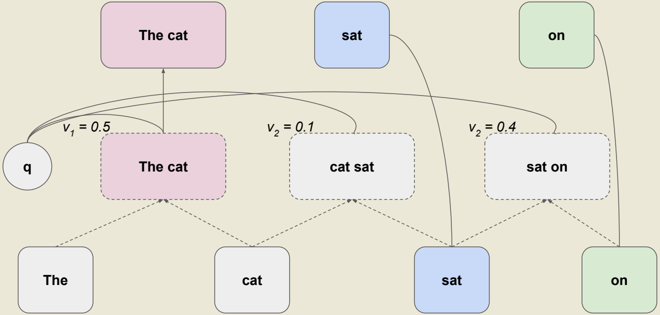
Sampling from incremental structures

- 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 that we had used a **differentiable surrogate functions**

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

[Papandreou and Yuille, 2011, Corro and Titov, 2019a,b]

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).

- $\boldsymbol{\epsilon} \sim G(0, 1)$
- $\tilde{\boldsymbol{\eta}} = \boldsymbol{\eta} + \boldsymbol{\epsilon}$
- $\arg\max_{\boldsymbol{z} \in \mathcal{Z}} \tilde{\boldsymbol{\eta}}^\top \boldsymbol{z}$

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

$$\mathbb{E}_{\pi_{\boldsymbol{\theta}}(\boldsymbol{z}|x)}\Big[L(\boldsymbol{z})\Big] \qquad L\Big(\arg\max_{\boldsymbol{z}} \pi_{\boldsymbol{\theta}}(\boldsymbol{z} \mid x)\Big)$$

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

- Straight Through
- SPIGOT

IV. End-to-end differentiable methods

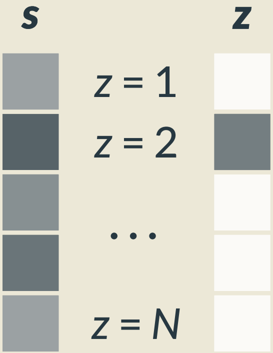
End-to-end differentiable methods

1. Digging into softmax
2. Alternatives to softmax
3. Generalizing to structured prediction
4. Stochasticity and global structures

Recall: Discrete choices & differentiability

input $\boldsymbol{x}$ → $\boldsymbol{s}$ → $\boldsymbol{z}$ → output $\widehat{\boldsymbol{y}}$

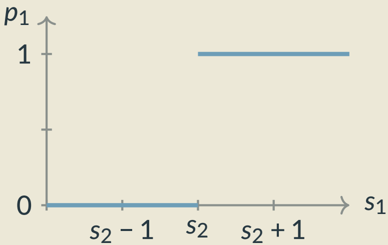
$\boldsymbol{s} = \boldsymbol{f_\theta}(\boldsymbol{x})$



$y = \boldsymbol{g_\phi}(\boldsymbol{z}, \boldsymbol{x})$

$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{s}} = \mathbf{0}$ or n/a

(argmax)



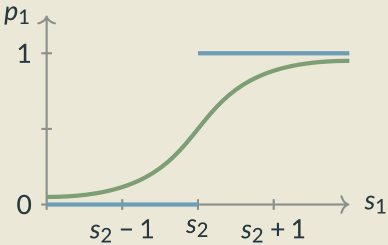
One solution: smooth relaxation

input $\boldsymbol{x}$ $\longrightarrow$ $\boldsymbol{s} = \boldsymbol{f_\theta}(\boldsymbol{x})$ $\longrightarrow$ $\boldsymbol{s}$ | $z = 1$, $z = 2$, $\dots$, $z = N$ | $\boldsymbol{p}$ $\longrightarrow$ $y = \boldsymbol{g_\phi}(\boldsymbol{z}, \boldsymbol{x})$ $\longrightarrow$ output $\widehat{\boldsymbol{y}}$

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

$\dfrac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}} =$ 😊

(softmax)



One solution: smooth relaxation

input $\boldsymbol{x}$

$\boldsymbol{s} = \boldsymbol{f_\theta}(\boldsymbol{x})$

$\boldsymbol{s}$ $\boldsymbol{p}$

$z = 1$

$z = 2$

$\dots$

$z = N$

😟

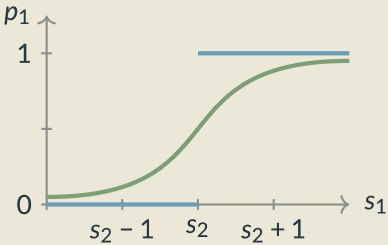
output $\widehat{\boldsymbol{y}}$

$y = \boldsymbol{g_\phi}(\boldsymbol{z}, \boldsymbol{x})$

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

$\frac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}} =$ 😊

(softmax)



Overview

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

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

$L\left(\arg\max_{z} \pi_{\boldsymbol{\theta}}(\mathbf{z} \mid x)\right)$

- Straight Through
- SPIGOT

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

- Structured Attn. Nets
- SparseMAP

Model restrictions:

- $\operatorname{dom} L$ may be only $\mathcal{Z}$,
- $\nabla_{\mathbf{z}} L$ need not exist!

- $L(\mathbf{z})$ with $\mathbf{z} \in \mathcal{Z}$ in forward
- needs (relaxed) $\nabla_{\mathbf{z}} L$ in backward.

- $L(\mathbf{z})$ must be relaxed and differentiable.
- (sparsity gets us closer to $\mathcal{Z}$).

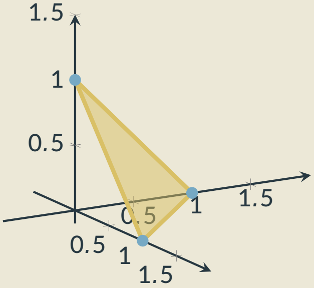
What is softmax?

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

What is softmax?

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

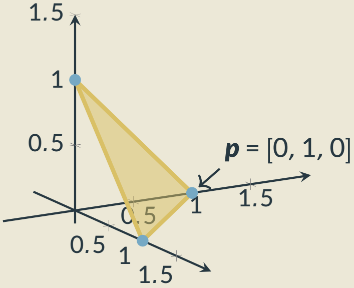
$\boldsymbol{p} \in \triangle$: probability distribution over choices



What is softmax?

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

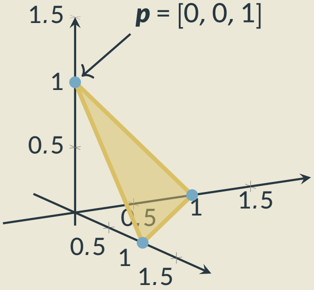
$\boldsymbol{p} \in \triangle$: probability distribution over choices



What is softmax?

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

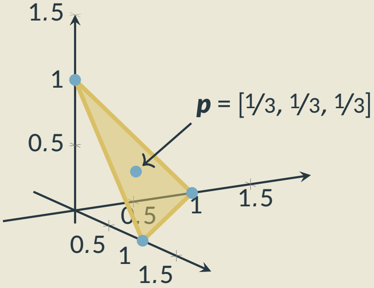
$\boldsymbol{p} \in \triangle$: probability distribution over choices



What is softmax?

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

$\boldsymbol{p} \in \triangle$: probability distribution over choices

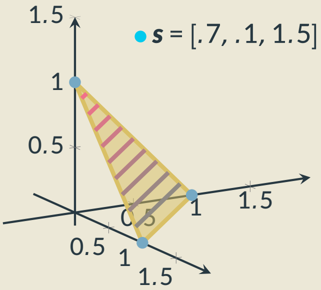


What is softmax?

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

$\boldsymbol{p} \in \triangle$: probability distribution over choices

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



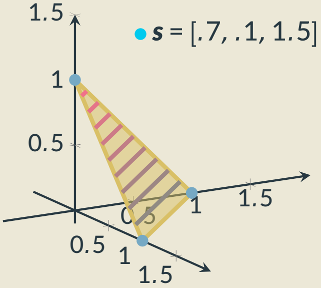
What is softmax?

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

$\boldsymbol{p} \in \triangle$: probability distribution over choices

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

argmax



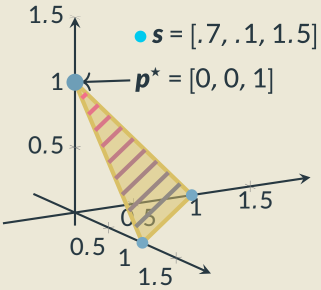
What is softmax?

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

$\boldsymbol{p} \in \triangle$: probability distribution over choices

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

argmax maximizes **expected score**



$$\arg\max_{\boldsymbol{p}\in\triangle} \boldsymbol{p}^\top \boldsymbol{s}$$

What is softmax?

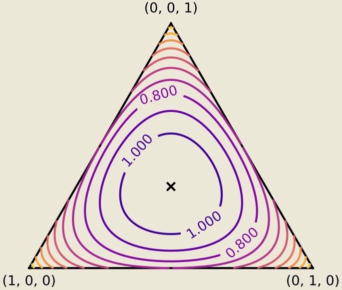
Often defined via $p_i = \dfrac{\exp s_i}{\sum_j \exp s_j}$, but where does it come from?

$\boldsymbol{p} \in \triangle$: probability distribution over choices

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

argmax maximizes **expected score**

Shannon entropy of $\boldsymbol{p}$: $\mathrm{H}(\boldsymbol{p}) = -\sum_i p_i \log p_i$



What is softmax?

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

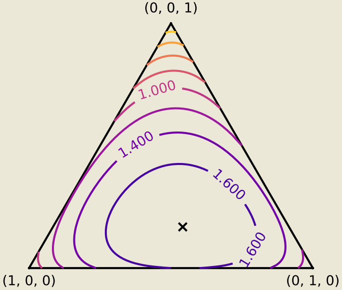
$\boldsymbol{p} \in \triangle$: probability distribution over choices

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

argmax maximizes **expected score**

Shannon entropy of $\boldsymbol{p}$: $\mathrm{H}(\boldsymbol{p}) = -\sum_i p_i \log p_i$

softmax maximizes **expected score** + **entropy**:



$$\underset{\boldsymbol{p} \in \triangle}{\arg\max}\, \boldsymbol{p}^\top \boldsymbol{s} + \mathrm{H}(\boldsymbol{p})$$

Variational form of softmax

Proposition. The unique solution to $\underset{\boldsymbol{p}\in\triangle}{\arg\max}\, \boldsymbol{p}^\top \boldsymbol{s} + \mathrm{H}(\boldsymbol{p})$ is given by $p_j = \frac{\exp s_j}{\sum_i \exp s_i}$.

Variational form of softmax

Proposition. The unique solution to $\underset{\boldsymbol{p}\in\triangle}{\arg\max}\, \boldsymbol{p}^\top \boldsymbol{s} + \mathrm{H}(\boldsymbol{p})$ is given by $p_j = \frac{\exp s_j}{\sum_i \exp s_i}$.

Explicit form of the optimization problem:

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

Variational form of softmax

Proposition. The unique solution to $\underset{\boldsymbol{p}\in\triangle}{\arg\max}\, \boldsymbol{p}^\top \boldsymbol{s} + \mathrm{H}(\boldsymbol{p})$ is given by $p_j = \frac{\exp s_j}{\sum_i \exp s_i}$.

Explicit form of the optimization problem:

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

Lagrangian:

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

Variational form of softmax

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

Explicit form of the optimization problem:

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

Lagrangian:

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

Optimality conditions (KKT):

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

Variational form of softmax

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

Explicit form of the optimization problem:

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

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

Lagrangian:

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

Optimality conditions (KKT):

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

$$\boldsymbol{p}^\top \boldsymbol{\nu} = 0$$

$$\boldsymbol{p} \in \triangle$$

$$\boldsymbol{\nu} \geq \mathbf{0}$$

Variational form of softmax

Proposition. The unique solution to $\underset{\boldsymbol{p} \in \triangle}{\arg\max}\, \boldsymbol{p}^\top \boldsymbol{s} + \mathrm{H}(\boldsymbol{p})$ is given by $p_j = \frac{\exp s_j}{\sum_i \exp s_i}$.

Explicit form of the optimization problem:

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

Lagrangian:

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

Optimality conditions (KKT):

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

$$\boldsymbol{p}^\top \boldsymbol{\nu} = 0$$

$$\boldsymbol{p} \in \triangle$$

$$\boldsymbol{\nu} \geq \mathbf{0}$$

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

if $p_i = 0$, r.h.s. must be $-\infty$,

thus $p_i > 0$, so $\nu_i = 0$.

Variational form of softmax

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

Explicit form of the optimization problem:

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

Lagrangian:

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

Optimality conditions (KKT):

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

$$\boldsymbol{p}^\top \boldsymbol{\nu} = 0$$

$$\boldsymbol{p} \in \triangle$$

$$\boldsymbol{\nu} \geq \mathbf{0}$$

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

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

$p_i = \exp(s_i)/\exp(\tau+1) = \exp(s_i)/Z$

Variational form of softmax

Proposition. The unique solution to $\underset{\boldsymbol{p}\in\triangle}{\arg\max}\, \boldsymbol{p}^\top \boldsymbol{s} + \mathrm{H}(\boldsymbol{p})$ is given by $p_j = \frac{\exp s_j}{\sum_i \exp s_i}$.

Explicit form of the optimization problem:

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

Lagrangian:

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

Optimality conditions (KKT):

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

$$\boldsymbol{p}^\top \boldsymbol{\nu} = 0$$

$$\boldsymbol{p} \in \triangle$$

$$\boldsymbol{\nu} \geq \mathbf{0}$$

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

if $p_i = 0$, r.h.s. must be $-\infty$,

thus $p_i > 0$, so $\nu_i = 0$.

$p_i = \exp(s_i)/\exp(\tau+1) = \exp(s_i)/Z$

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

Variational form of softmax

Proposition. The unique solution to $\displaystyle\arg\max_{\boldsymbol{p}\in\triangle} \boldsymbol{p}^\top \boldsymbol{s} + \mathrm{H}(\boldsymbol{p})$ is given by $p_j = \frac{\exp s_j}{\sum_i \exp s_i}$.

Explicit form of the optimization problem:

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

Lagrangian:

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

Optimality conditions (KKT):

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

$$\boldsymbol{p}^\top \boldsymbol{\nu} = 0$$

$$\boldsymbol{p} \in \triangle$$

$$\boldsymbol{\nu} \geq \mathbf{0}$$

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

if $p_i = 0$, r.h.s. must be $-\infty$,

thus $p_i > 0$, so $\nu_i = 0$.

$p_i = {}^{\exp(s_i)}/_{\exp(\tau+1)} = {}^{\exp(s_i)}/_{Z}$

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 $\underset{\boldsymbol{p}\in\triangle}{\arg\max}\, \boldsymbol{p}^\top \boldsymbol{s} + \mathrm{H}(\boldsymbol{p})$ is given by $p_j = \frac{\exp s_j}{\sum_i \exp s_i}$.

Explicit form of the optimization problem:

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

Lagrangian:

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

Optimality conditions (KKT):

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

$$\boldsymbol{p}^\top \boldsymbol{\nu} = 0$$

$$\boldsymbol{p} \in \triangle$$

$$\boldsymbol{\nu} \geq \mathbf{0}$$

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

$p_i = {}^{\exp(s_i)}/_{\exp(\tau+1)} = {}^{\exp(s_i)}/_{Z}$

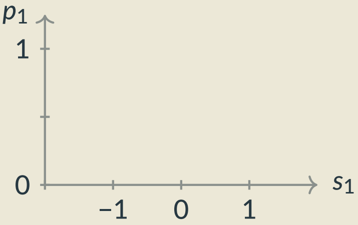
Must find Z such that $\sum_j p_j = 1$.
Answer: $Z = \sum_j \exp(s_j)$

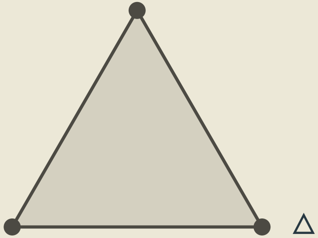
So, $p_i = \dfrac{\exp(s_i)}{\sum_j \exp(s_j)}$.

Classic result, e.g., [Boyd and Vandenberghe, 2004, Wainwright and Jordan, 2008]

Generalizing softmax: Smoothed argmaxes

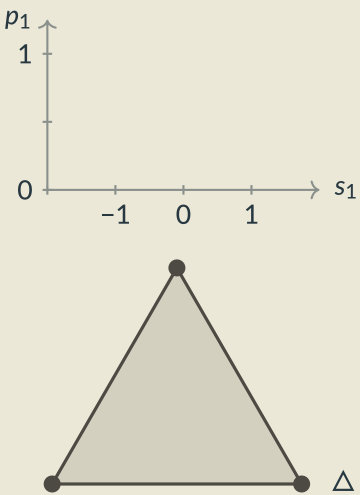
$$\hat{\boldsymbol{p}}_\Omega(\boldsymbol{s}) = \arg\max_{\boldsymbol{p}\in\triangle} \boldsymbol{p}^\top \boldsymbol{s} - \Omega(\boldsymbol{p})$$





Generalizing softmax: Smoothed argmaxes

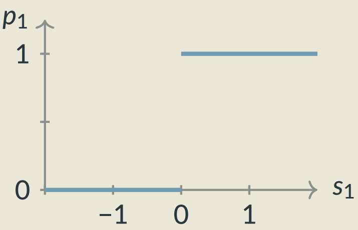
$$\hat{\boldsymbol{p}}_\Omega(\boldsymbol{s}) = \underset{\boldsymbol{p}\in\triangle}{\arg\max}\, \boldsymbol{p}^\top \boldsymbol{s} - \Omega(\boldsymbol{p})$$

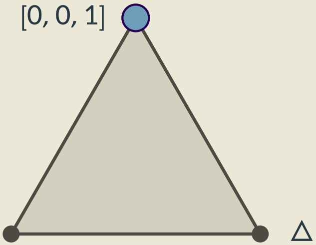


Generalizing softmax: Smoothed argmaxes

$$\hat{\boldsymbol{p}}_\Omega(\boldsymbol{s}) = \arg\max_{\boldsymbol{p}\in\triangle} \boldsymbol{p}^\top \boldsymbol{s} - \Omega(\boldsymbol{p})$$

- argmax: $\Omega(\boldsymbol{p}) = 0$

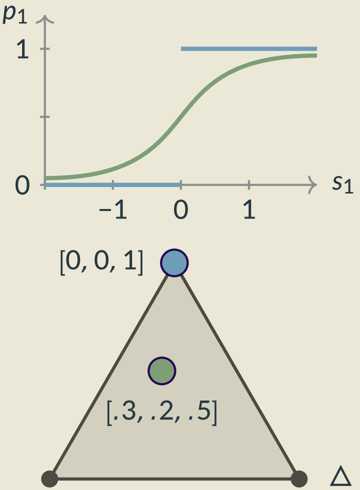




Generalizing softmax: Smoothed argmaxes

$$\hat{\boldsymbol{p}}_\Omega(\boldsymbol{s}) = \arg\max_{\boldsymbol{p}\in\triangle} \boldsymbol{p}^\top \boldsymbol{s} - \Omega(\boldsymbol{p})$$

- argmax: $\Omega(\boldsymbol{p}) = 0$
- softmax: $\Omega(\boldsymbol{p}) = \sum_j p_j \log p_j$

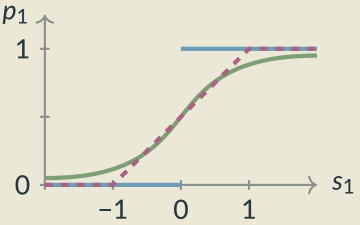


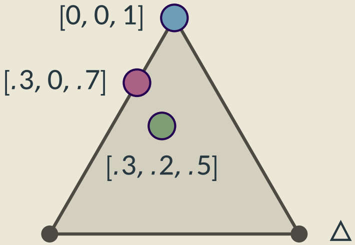
[Niculae and Blondel, 2017, Martins and Astudillo, 2016]

Generalizing softmax: Smoothed argmaxes

$$\hat{\boldsymbol{p}}_\Omega(\boldsymbol{s}) = \arg\max_{\boldsymbol{p}\in\triangle} \boldsymbol{p}^\top \boldsymbol{s} - \Omega(\boldsymbol{p})$$

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

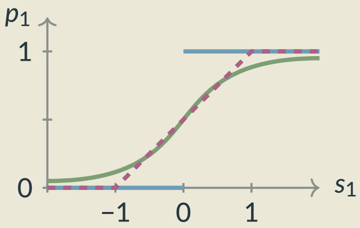


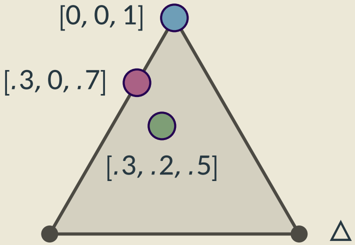


Generalizing softmax: Smoothed argmaxes

$$\hat{\boldsymbol{p}}_\Omega(\boldsymbol{s}) = \arg\max_{\boldsymbol{p}\in\triangle} \boldsymbol{p}^\top \boldsymbol{s} - \Omega(\boldsymbol{p})$$

- argmax: $\Omega(\boldsymbol{p}) = 0$
- softmax: $\Omega(\boldsymbol{p}) = \sum_j p_j \log p_j$
- sparsemax: $\Omega(\boldsymbol{p}) = 1/2 \|\boldsymbol{p}\|_2^2$
- α-entmax: $\Omega(\boldsymbol{p}) = 1/\alpha(\alpha-1) \sum_j p_i^\alpha$





Generalized entropy interpolates in between [Tsallis, 1988]
Used in Sparse Seq2Seq: [Peters et al., 2019]
(Mon 13:50, poster session 2D)

[Niculae and Blondel, 2017, Malaviya et al., 2018]

Generalizing softmax: Smoothed argmaxes

$$\hat{\boldsymbol{p}}_\Omega(\boldsymbol{s}) = \underset{\boldsymbol{p}\in\triangle}{\arg\max}\, \boldsymbol{p}^\top \boldsymbol{s} - \Omega(\boldsymbol{p})$$

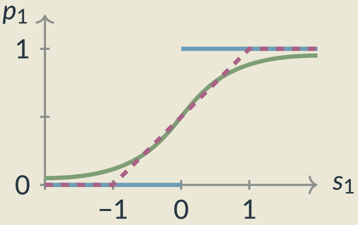
- argmax: $\Omega(\boldsymbol{p}) = 0$
- softmax: $\Omega(\boldsymbol{p}) = \sum_j p_j \log p_j$
- sparsemax: $\Omega(\boldsymbol{p}) = 1/2 \|\boldsymbol{p}\|_2^2$

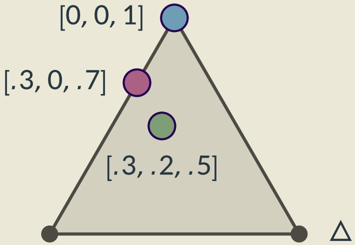
α-entmax: $\Omega(\boldsymbol{p}) = 1/\alpha(\alpha-1) \sum_j p_i^\alpha$

fusedmax: $\Omega(\boldsymbol{p}) = 1/2 \|\boldsymbol{p}\|_2^2 + \sum_j |p_j - p_{j-1}|$

csparsemax: $\Omega(\boldsymbol{p}) = 1/2 \|\boldsymbol{p}\|_2^2 + \iota(\boldsymbol{a} \le \boldsymbol{p} \le \boldsymbol{b})$

csoftmax: $\Omega(\boldsymbol{p}) = \sum_j p_j \log p_j + \iota(\boldsymbol{a} \le \boldsymbol{p} \le \boldsymbol{b})$

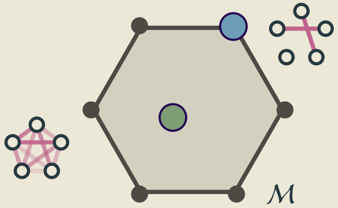




[Wainwright and Jordan, 2008]

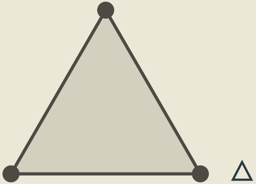
The structured case: Marginal polytope

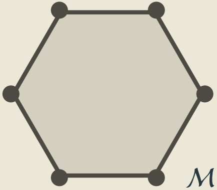
- Each vertex corresponds to one such *bit vector* $\boldsymbol{z}$
- Points inside correspond to *marginal distributions*: convex combinations of structured objects



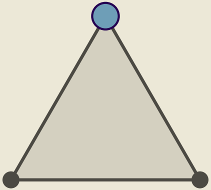
$$\boldsymbol{\mu} = \underbrace{p_1\boldsymbol{z}_1 + \ldots + p_N\boldsymbol{z}_N}_{\text{exponentially many terms}}, \quad \boldsymbol{p} \in \Delta.$$

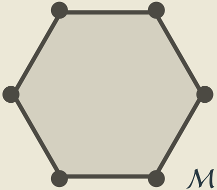
$$\begin{array}{ll} p_1 = 0.2, & \boldsymbol{z}_1 = [1, 0, 0, 0, 1, 0, 0, 0, 1] \\ p_2 = 0.7, & \boldsymbol{z}_2 = [0, 0, 1, 0, 0, 1, 1, 0, 0] \\ p_3 = 0.1, & \boldsymbol{z}_3 = [1, 0, 0, 0, 1, 0, 0, 1, 0] \end{array} \Rightarrow \quad \boldsymbol{\mu} = [.3, 0, .7, 0, .3, .7, .7, .1, .2].$$


$\triangle$


$\mathcal{M}$

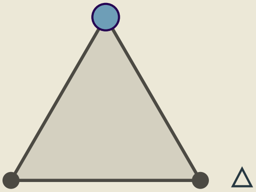
- **argmax** $\arg\max_{\boldsymbol{p} \in \triangle} \boldsymbol{p}^\top \boldsymbol{s}$

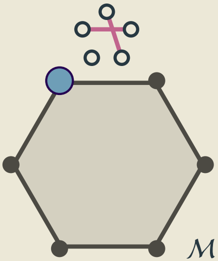




argmax $\arg\max_{\boldsymbol{p} \in \triangle} \boldsymbol{p}^\top \boldsymbol{s}$

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

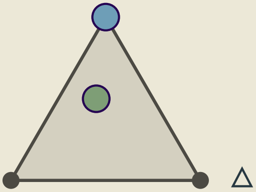


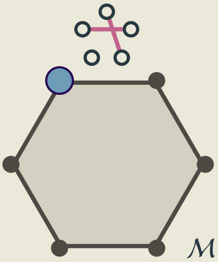


argmax $\arg\max_{\boldsymbol{p}\in\triangle} \boldsymbol{p}^\top \boldsymbol{s}$

softmax $\arg\max_{\boldsymbol{p}\in\triangle} \boldsymbol{p}^\top \boldsymbol{s} + \mathrm{H}(\boldsymbol{p})$

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



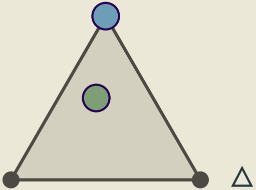


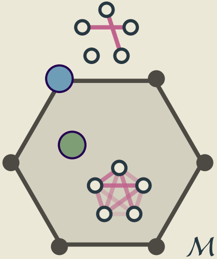
argmax $\arg\max_{\boldsymbol{p}\in\triangle} \boldsymbol{p}^\top \boldsymbol{s}$

softmax $\arg\max_{\boldsymbol{p}\in\triangle} \boldsymbol{p}^\top \boldsymbol{s} + \mathrm{H}(\boldsymbol{p})$

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

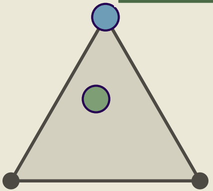
marginals $\arg\max_{\boldsymbol{\mu}\in\mathcal{M}} \boldsymbol{\mu}^\top \boldsymbol{\eta} + \widetilde{\mathrm{H}}(\boldsymbol{\mu})$



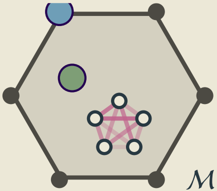


- **argmax** $\arg\max_{\boldsymbol{p}\in\triangle} \boldsymbol{p}^\top \boldsymbol{s}$
- **softmax** $\arg\max_{\boldsymbol{p}\in\triangle} \boldsymbol{p}^\top \boldsymbol{s} + \mathrm{H}(\boldsymbol{p})$
- **MAP** $\arg\max_{\boldsymbol{\mu}\in\mathcal{M}} \boldsymbol{\mu}^\top \boldsymbol{\eta}$
- **marginals** $\arg\max_{\boldsymbol{\mu}\in\mathcal{M}} \boldsymbol{\mu}^\top \boldsymbol{\eta} + \widetilde{\mathrm{H}}(\boldsymbol{\mu})$

Just like softmax relaxes argmax,
marginals relax MAP **differentiably**!



$\triangle$



$\mathcal{M}$

argmax $\arg\max_{\boldsymbol{p}\in\triangle} \boldsymbol{p}^\top \boldsymbol{s}$

softmax $\arg\max_{\boldsymbol{p}\in\triangle} \boldsymbol{p}^\top \boldsymbol{s} + \mathrm{H}(\boldsymbol{p})$

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

marginals $\arg\max_{\boldsymbol{\mu}\in\mathcal{M}} \boldsymbol{\mu}^\top \boldsymbol{\eta} + \widetilde{\mathrm{H}}(\boldsymbol{\mu})$

Just like softmax relaxes argmax, marginals relax MAP **differentiably**!

Unlike argmax/softmax, computation is not obvious!

$\triangle$

$\mathcal{M}$

Algorithms for specific structures

	Best structure (MAP)	Marginals
Sequence tagging	Viterbi [Rabiner, 1989]	Forward-Backward [Rabiner, 1989]
Constituent trees	CKY [Kasami, 1966, Younger, 1967] [Cocke and Schwartz, 1970]	Inside-Outside [Baker, 1979]
Temporal alignments	DTW [Sakoe and Chiba, 1978]	Soft-DTW [Cuturi and Blondel, 2017]
Dependency trees	Max. Spanning Arborescence [Chu and Liu, 1965, Edmonds, 1967]	Matrix-Tree [Kirchhoff, 1847]
Assignments	Kuhn-Munkres [Kuhn, 1955, Jonker and Volgenant, 1987]	#P-complete [Valiant, 1979, Taskar, 2004]

Algorithms for specific structures

		Best structure (MAP)	Marginals
	Sequence tagging	Viterbi [Rabiner, 1989]	Forward-Backward [Rabiner, 1989]
dyn. prog.	**Constituent trees**	CKY [Kasami, 1966, Younger, 1967] [Cocke and Schwartz, 1970]	Inside-Outside [Baker, 1979]
	Temporal alignments	DTW [Sakoe and Chiba, 1978]	Soft-DTW [Cuturi and Blondel, 2017]
	Dependency trees	Max. Spanning Arborescence [Chu and Liu, 1965, Edmonds, 1967]	Matrix-Tree [Kirchhoff, 1847]
	Assignments	Kuhn-Munkres [Kuhn, 1955, Jonker and Volgenant, 1987]	#P-complete [Valiant, 1979, Taskar, 2004]

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.

---

input: d tags, n tokens, $\boldsymbol{\eta}_U \in \mathbb{R}^{n\times d}$, $\boldsymbol{\eta}_V \in \mathbb{R}^{d\times d}$

initialize $\boldsymbol{\alpha}_1 = \mathbf{0}$, $\boldsymbol{\beta}_n = \mathbf{0}$

for $i \in 2, \ldots, n$ **do** *# forward log-probabilities*

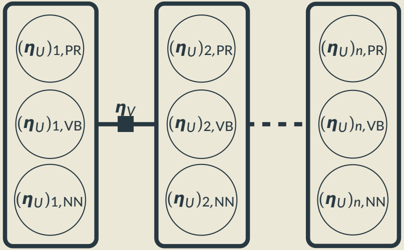
$\quad \alpha_{i,k} = \log \sum_{k'} \exp\left(\alpha_{i-1,k'} + (\boldsymbol{\eta}_U)_{i,k} + (\boldsymbol{\eta}_V)_{k',k}\right)$ for all k

for $i \in n-1, \ldots, 1$ **do** *# backward log-probabilities*

$\quad \beta_{i,k} = \log \sum_{k'} \exp\left(\beta_{i+1,k'} + (\boldsymbol{\eta}_U)_{i+1,k'} + (\boldsymbol{\eta}_V)_{k,k'}\right)$ for all k

$Z = \sum_k \exp \alpha_{n,k}$ *# partition function*

return $\boldsymbol{\mu} = \exp\left(\boldsymbol{\alpha} + \boldsymbol{\beta} - \log Z\right)$ *# marginals*



Derivatives of marginals 1: DP

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

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

Marginals in a sequence tagging model.

input: d tags, n tokens, $\boldsymbol{\eta}_U \in \mathbb{R}^{n \times d}$, $\boldsymbol{\eta}_V \in \mathbb{R}^{d \times d}$

initialize $\boldsymbol{\alpha}_1 = \mathbf{0}$, $\boldsymbol{\beta}_n = \mathbf{0}$

for $i \in 2, \ldots, n$ **do** *# forward log-probabilities*

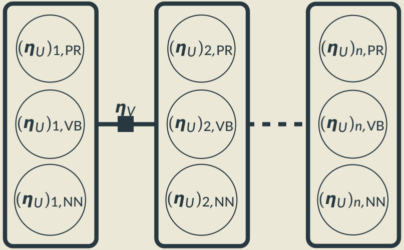
  $\alpha_{i,k} = \log \sum_{k'} \exp\left(\alpha_{i-1,k'} + (\boldsymbol{\eta}_U)_{i,k} + (\boldsymbol{\eta}_V)_{k',k}\right)$ for all k

for $i \in n-1, \ldots, 1$ **do** *# backward log-probabilities*

  $\beta_{i,k} = \log \sum_{k'} \exp\left(\beta_{i+1,k'} + (\boldsymbol{\eta}_U)_{i+1,k'} + (\boldsymbol{\eta}_V)_{k,k'}\right)$ for all k

$Z = \sum_k \exp \alpha_{n,k}$ *# partition function*

return $\boldsymbol{\mu} = \exp\left(\boldsymbol{\alpha} + \boldsymbol{\beta} - \log Z\right)$ *# marginals*



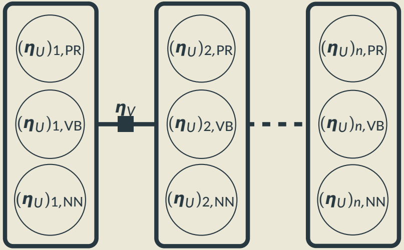
Derivatives of marginals 1: DP

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

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

Marginals in a sequence tagging model.

input: d tags, n tokens, $\boldsymbol{\eta}_U \in \mathbb{R}^{n\times d}$, $\boldsymbol{\eta}_V \in \mathbb{R}^{d\times d}$
initialize $\boldsymbol{\alpha}_1 = \mathbf{0}$, $\boldsymbol{\beta}_n = \mathbf{0}$
for $i \in 2, \ldots, n$ **do** *# forward log-probabilities*
    $\alpha_{i,k} = \log \sum_{k'} \exp\left(\alpha_{i-1,k'} + (\boldsymbol{\eta}_U)_{i,k} + (\boldsymbol{\eta}_V)_{k',k}\right)$ for all k
for $i \in n-1, \ldots, 1$ **do** *# backward log-probabilities*
    $\beta_{i,k} = \log \sum_{k'} \exp\left(\beta_{i+1,k'} + (\boldsymbol{\eta}_U)_{i+1,k'} + (\boldsymbol{\eta}_V)_{k,k'}\right)$ for all k
$Z = \sum_k \exp \alpha_{n,k}$ *# partition function*
return $\boldsymbol{\mu} = \exp\left(\boldsymbol{\alpha} + \boldsymbol{\beta} - \log Z\right)$ *# marginals*



Derivatives of marginals 1: DP

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

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

Marginals in a sequence tagging model.

---

input: d tags, n tokens, $\boldsymbol{\eta}_U \in \mathbb{R}^{n \times d}$, $\boldsymbol{\eta}_V \in \mathbb{R}^{d \times d}$

initialize $\boldsymbol{\alpha}_1 = \mathbf{0}$, $\boldsymbol{\beta}_n = \mathbf{0}$

for $i \in 2, \ldots, n$ **do** *# forward log-probabilities*

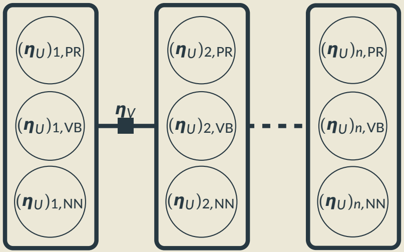
  $\alpha_{i,k} = \log \sum_{k'} \exp\left(\alpha_{i-1,k'} + (\boldsymbol{\eta}_U)_{i,k} + (\boldsymbol{\eta}_V)_{k',k}\right)$ for all k

for $i \in n-1, \ldots, 1$ **do** *# backward log-probabilities*

  $\beta_{i,k} = \log \sum_{k'} \exp\left(\beta_{i+1,k'} + (\boldsymbol{\eta}_U)_{i+1,k'} + (\boldsymbol{\eta}_V)_{k,k'}\right)$ for all k

$Z = \sum_k \exp \alpha_{n,k}$ *# partition function*

return $\boldsymbol{\mu} = \exp\left(\boldsymbol{\alpha} + \boldsymbol{\beta} - \log Z\right)$ *# marginals*



Derivatives of marginals 2: Matrix-Tree

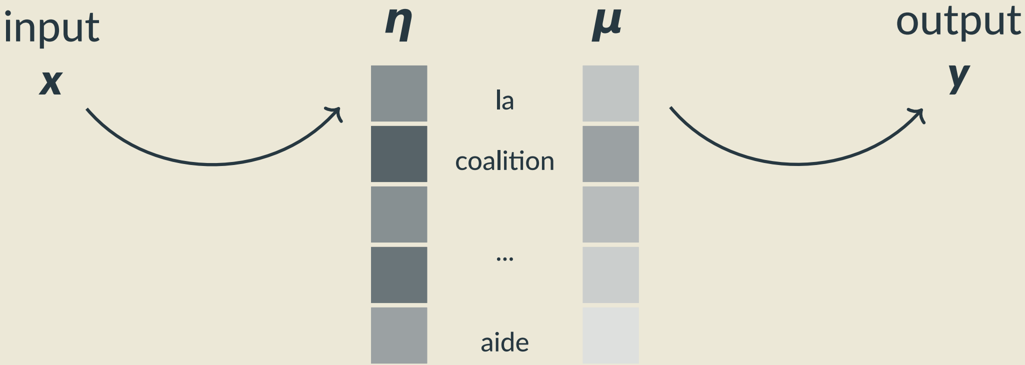
$\boldsymbol{L}(\boldsymbol{s})$: Laplacian of the edge score graph

$$Z = \det \boldsymbol{L}(\boldsymbol{s})$$

$$\boldsymbol{\mu} = \boldsymbol{L}(\boldsymbol{s})^{-1}$$

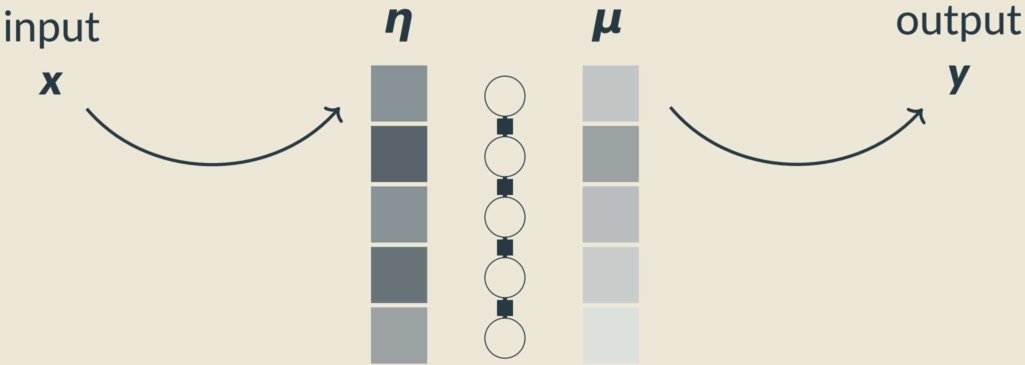
$$\nabla\boldsymbol{\mu} = \nabla\boldsymbol{L}^{-1} = \boldsymbol{L}^{-1}\left(\frac{\partial \boldsymbol{L}}{\partial \boldsymbol{\eta}}\right)\boldsymbol{L}^{-1}$$

Structured Attention Networks



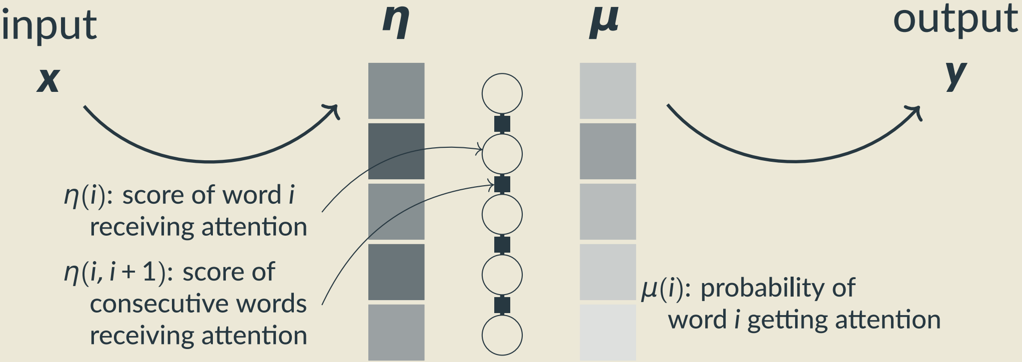
Structured Attention Networks

[Kim et al., 2017]



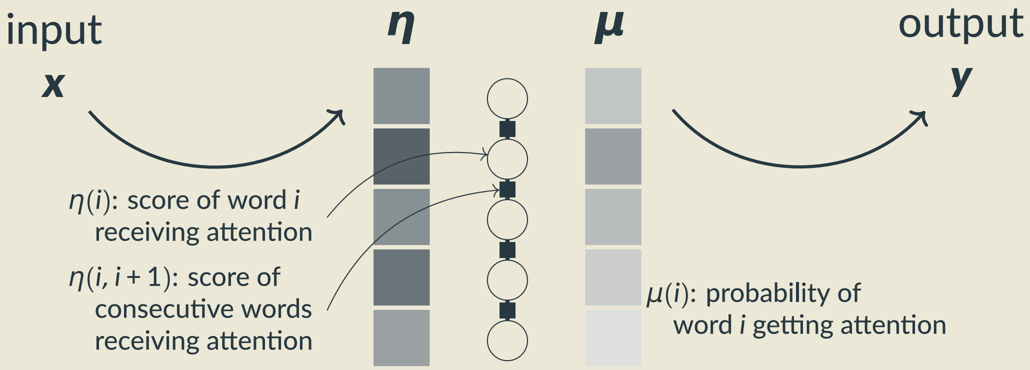
[Kim et al., 2017]

Structured Attention Networks



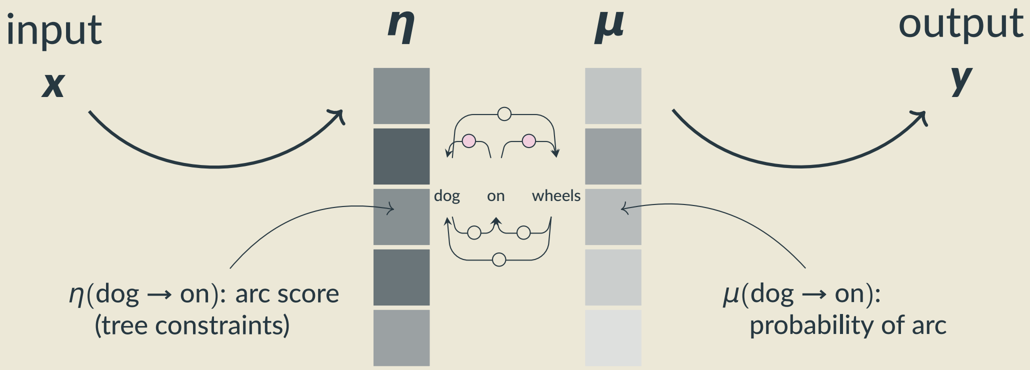
[Kim et al., 2017]

Structured Attention Networks



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

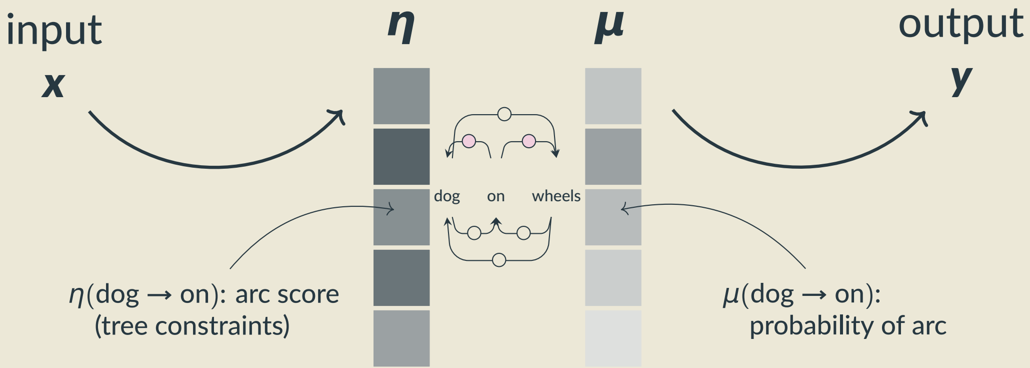
Structured Attention Networks



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

Similar idea for projective dependency trees with *inside–outside*

Structured Attention Networks



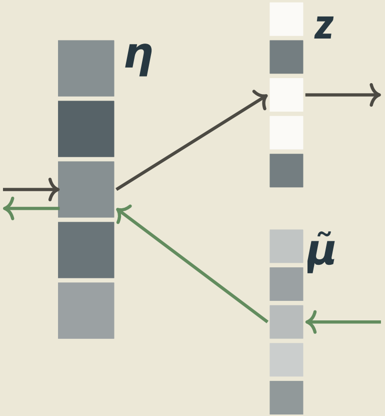
CRF marginals (from *forward–backward*) give attention weights $\in (0, 1)$

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 $\boldsymbol{z}$ (approximately)
  $$\boldsymbol{z} = \arg\max_{\boldsymbol{z} \in \mathcal{Z}} (\boldsymbol{\eta} + \boldsymbol{\epsilon})^\top \boldsymbol{z}$$
- Backward pass:
  pretend we did marginal inference
  $$\tilde{\boldsymbol{\mu}} = \arg\max_{\boldsymbol{\mu} \in \mathcal{M}} (\boldsymbol{\eta} + \boldsymbol{\epsilon})^\top \boldsymbol{z} + \tilde{\mathrm{H}}(\boldsymbol{\mu})$$
  (or some similar relaxation)



Back-propagating through marginals

Pros:

Back-propagating through marginals

Pros:

- Familiar algorithms for NLPers,

Back-propagating through marginals

Pros:

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

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.

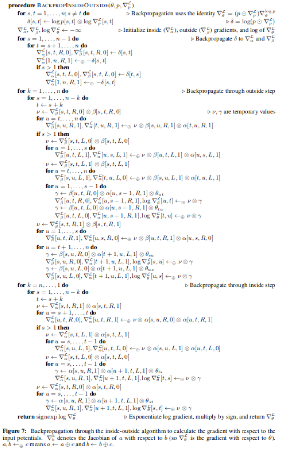
Back-propagating through marginals

Pros:

- Familiar algorithms for NLPers,
- (Structured Attention Net … xact.

Cons:

- (Structured Attention Net … inals are dense;
  (fixed by Perturb & MA … nation)
- Efficient & numerically st … ugh DPs is tricky;
  (somewhat alleviated b
- Not applicable when mar
- Case-by-case algorithms



```
procedure BackpropInsideOutside(θ, p, ∇^L_p)
  for s, t = 1, ..., n; s ≠ t do        ▷ Backpropagation uses the identity ∇^L_θ = (p ⊙ ∇^L_p)∇^{log p}_θ
    δ[s, t] ← log p[s, t] ⊗ log ∇^L_p[s, t]        ▷ δ = log(p ⊙ ∇^L_p)
  ∇^L_α, ∇^L_β, log ∇^L_θ ← −∞        ▷ Initialize inside (∇^L_α), outside (∇^L_β) gradients, and log of ∇^L_θ
  for s = 1, ..., n − 1 do        ▷ Backpropagate δ to ∇^L_α and ∇^L_β
    for t = s + 1, ..., n do
      ∇^L_α[s, t, R, 0], ∇^L_β[s, t, R, 0] ← δ[s, t]
      ∇^L_α[1, n, R, 1] ←_⊕ −δ[s, t]
      if s > 1 then
        ∇^L_α[s, t, L, 0], ∇^L_β[s, t, L, 0] ← δ[t, s]
        ∇^L_α[1, n, R, 1] ←_⊕ −δ[s, t]
  for k = 1, ..., n do        ▷ Backpropagate through outside step
    for s = 1, ..., n − k do
      t ← s + k
      ν ← ∇^L_β[s, t, R, 0] ⊗ β[s, t, R, 0]        ▷ ν, γ are temporary values
      for u = t, ..., n do
        ∇^L_β[s, u, R, 1], ∇^L_α[t, u, R, 1] ←_⊕ ν ⊗ β[s, u, R, 1] ⊗ α[t, u, R, 1]
      if s > 1 then
        ν ← ∇^L_β[s, t, L, 0] ⊗ β[s, t, L, 0]
        for u = 1, ..., s do
          ∇^L_β[u, t, L, 1], ∇^L_α[u, s, L, 1] ←_⊕ ν ⊗ β[u, t, L, 1] ⊗ α[u, s, L, 1]
        ν ← ∇^L_β[s, t, L, 1] ⊗ β[s, t, L, 1]
        for u = t, ..., n do
          ∇^L_β[s, u, L, 1], ∇^L_α[t, u, L, 0] ←_⊕ ν ⊗ β[s, u, L, 1] ⊗ α[t, u, L, 1]
        for u = 1, ..., s − 1 do
          γ ← β[u, t, R, 0] ⊗ α[u, s − 1, R, 1] ⊗ θ_ut
          ∇^L_β[u, t, R, 0], ∇^L_α[u, s − 1, R, 1], log ∇^L_θ[u, t] ←_⊕ ν ⊗ γ
          γ ← β[u, t, L, 0] ⊗ α[u, s − 1, R, 1] ⊗ θ_tu
          ∇^L_β[u, t, L, 0], ∇^L_α[u, s − 1, R, 1], log ∇^L_θ[t, u] ←_⊕ ν ⊗ γ
      ν ← ∇^L_β[s, t, R, 1] ⊗ β[s, t, R, 1]
      for u = 1, ..., s do
        ∇^L_β[u, t, R, 1], ∇^L_α[u, s, R, 0] ←_⊕ ν ⊗ β[u, t, R, 1] ⊗ α[u, s, R, 0]
      for u = t + 1, ..., n do
        γ ← β[s, u, R, 0] ⊗ α[t + 1, u, L, 1] ⊗ θ_su
        ∇^L_β[s, u, R, 0], ∇^L_α[t + 1, u, L, 1], log ∇^L_θ[s, u] ←_⊕ ν ⊗ γ
        γ ← β[s, u, L, 0] ⊗ α[t + 1, u, L, 1] ⊗ θ_us
        ∇^L_β[s, u, L, 0], ∇^L_α[t + 1, u, L, 1], log ∇^L_θ[u, s] ←_⊕ ν ⊗ γ
  for k = n, ..., 1 do        ▷ Backpropagate through inside step
    for s = 1, ..., n − k do
      t ← s + k
      ν ← ∇^L_α[s, t, R, 1] ⊗ α[s, t, R, 1]
      for u = s + 1, ..., t do
        ∇^L_α[u, t, R, 0], ∇^L_α[u, t, R, 1] ←_⊕ ν ⊗ α[s, u, R, 0] ⊗ α[u, t, R, 1]
      if s > 1 then
        ν ← ∇^L_α[s, t, L, 1] ⊗ α[s, t, L, 1]
        for u = s, ..., t − 1 do
          ∇^L_α[s, u, L, 1], ∇^L_α[u, t, L, 0] ←_⊕ ν ⊗ α[s, u, L, 1] ⊗ α[u, t, L, 0]
        ν ← ∇^L_α[s, t, L, 0] ⊗ α[s, t, L, 0]
        for u = s, ..., t − 1 do
          γ ← α[s, u, R, 1] ⊗ α[u + 1, t, L, 1] ⊗ θ_ts
          ∇^L_α[s, u, R, 1], ∇^L_α[u + 1, t, L, 1], log ∇^L_θ[t, s] ←_⊕ ν ⊗ γ
      ν ← ∇^L_α[s, t, R, 0] ⊗ α[s, t, R, 0]
      for u = s, ..., t − 1 do
        γ ← α[s, u, R, 1] ⊗ α[u + 1, t, L, 1] ⊗ θ_st
        ∇^L_α[s, u, R, 1], ∇^L_α[u + 1, t, L, 1], log ∇^L_θ[s, t] ←_⊕ ν ⊗ γ
  return signexp log ∇^L_θ        ▷ Exponentiate log gradient, multiply by sign, and return ∇^L_θ
```

Figure 7: Backpropagation through the inside-outside algorithm to calculate the gradient with respect to the input potentials. ∇^a_b denotes the Jacobian of a with respect to b (so $\nabla^{\mathcal{L}}_\theta$ is the gradient with respect to θ). $a, b \leftarrow_\oplus c$ means $a \leftarrow a \oplus c$ and $b \leftarrow b \oplus c$.

Back-propagating through marginals

Pros:

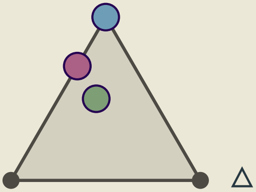
- Familiar algorithms for NLPers,
- (Structured Attention Networks:) All computations exact.

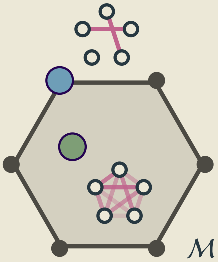
Cons:

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

- **argmax** $\arg\max_{\boldsymbol{p}\in\triangle} \boldsymbol{p}^\top \boldsymbol{s}$
- **softmax** $\arg\max_{\boldsymbol{p}\in\triangle} \boldsymbol{p}^\top \boldsymbol{s} + \mathrm{H}(\boldsymbol{p})$
- **sparsemax** $\arg\max_{\boldsymbol{p}\in\triangle} \boldsymbol{p}^\top \boldsymbol{s} - 1/2\|\boldsymbol{p}\|^2$

- **MAP** $\arg\max_{\boldsymbol{\mu}\in\mathcal{M}} \boldsymbol{\mu}^\top \boldsymbol{\eta}$
- **marginals** $\arg\max_{\boldsymbol{\mu}\in\mathcal{M}} \boldsymbol{\mu}^\top \boldsymbol{\eta} + \widetilde{\mathrm{H}}(\boldsymbol{\mu})$

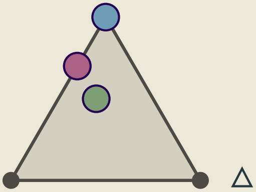


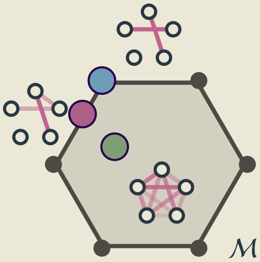


[Niculae et al., 2018a]

- **argmax** $\arg\max_{\boldsymbol{p}\in\triangle} \boldsymbol{p}^\top \boldsymbol{s}$
- **softmax** $\arg\max_{\boldsymbol{p}\in\triangle} \boldsymbol{p}^\top \boldsymbol{s} + \mathrm{H}(\boldsymbol{p})$
- **sparsemax** $\arg\max_{\boldsymbol{p}\in\triangle} \boldsymbol{p}^\top \boldsymbol{s} - 1/2\|\boldsymbol{p}\|^2$

- **MAP** $\arg\max_{\boldsymbol{\mu}\in\mathcal{M}} \boldsymbol{\mu}^\top \boldsymbol{\eta}$
- **marginals** $\arg\max_{\boldsymbol{\mu}\in\mathcal{M}} \boldsymbol{\mu}^\top \boldsymbol{\eta} + \widetilde{\mathrm{H}}(\boldsymbol{\mu})$
- **SparseMAP** $\arg\max_{\boldsymbol{\mu}\in\mathcal{M}} \boldsymbol{\mu}^\top \boldsymbol{\eta} - 1/2\|\boldsymbol{\mu}\|^2$





SparseMAP solution

$$\boldsymbol{\mu}^\star = \operatorname*{arg\,max}_{\boldsymbol{\mu}\in\mathcal{M}} \boldsymbol{\mu}^\top \boldsymbol{\eta} - 1/2\|\boldsymbol{\mu}\|^2$$

=  = .6  + .4 

($\boldsymbol{\mu}^\star$ is unique, but may have multiple decompositions $\boldsymbol{p}$. Active Set recovers a sparse one.)

Algorithms for SparseMAP

$$\boldsymbol{\mu}^\star = \arg\max_{\boldsymbol{\mu}\in\mathcal{M}} \boldsymbol{\mu}^\top\boldsymbol{\eta} - 1/2\|\boldsymbol{\mu}\|^2$$

Algorithms for SparseMAP

$$\boldsymbol{\mu}^\star = \underset{\boldsymbol{\mu}\in\mathcal{M}}{\arg\max}\, \boldsymbol{\mu}^\top \boldsymbol{\eta} - 1/2\|\boldsymbol{\mu}\|^2$$

linear constraints
(alas, exponentially many!)

quadratic objective

Algorithms for SparseMAP

$$\boldsymbol{\mu}^\star = \underset{\boldsymbol{\mu}\in\mathcal{M}}{\arg\max}\, \boldsymbol{\mu}^\top \boldsymbol{\eta} - 1/2\|\boldsymbol{\mu}\|^2$$

linear constraints
(alas, exponentially many!)

quadratic objective

Conditional Gradient

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

Algorithms for SparseMAP

$$\boldsymbol{\mu}^\star = \arg\max_{\boldsymbol{\mu} \in \mathcal{M}} \boldsymbol{\mu}^\top \boldsymbol{\eta} - 1/2\|\boldsymbol{\mu}\|^2$$

linear constraints
(alas, exponentially many!)

quadratic objective

Conditional Gradient

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

- select a new corner of $\mathcal{M}$

Algorithms for SparseMAP

$$\boldsymbol{\mu}^\star = \arg\max_{\boldsymbol{\mu}\in\mathcal{M}} \boldsymbol{\mu}^\top \boldsymbol{\eta} - 1/2\|\boldsymbol{\mu}\|^2$$

linear constraints
(alas, exponentially many!)

quadratic objective

Conditional Gradient

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

- select a new corner of $\mathcal{M}$

$$\arg\max_{\boldsymbol{\mu}\in\mathcal{M}} \boldsymbol{\mu}^\top \underbrace{\left(\boldsymbol{\eta} - \boldsymbol{\mu}^{(t-1)}\right)}_{\tilde{\boldsymbol{\eta}}}$$

Algorithms for SparseMAP

$$\boldsymbol{\mu}^\star = \operatorname*{arg\,max}_{\boldsymbol{\mu}\in\mathcal{M}} \boldsymbol{\mu}^\top \boldsymbol{\eta} - 1/2\|\boldsymbol{\mu}\|^2$$

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 $\boldsymbol{p}$
  - Update rules: vanilla, away-step, pairwise

Algorithms for SparseMAP

$$\boldsymbol{\mu}^\star = \arg\max_{\boldsymbol{\mu}\in\mathcal{M}} \boldsymbol{\mu}^\top \boldsymbol{\eta} - 1/2\|\boldsymbol{\mu}\|^2$$

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 $\boldsymbol{p}$
  - 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

$$\boldsymbol{\mu}^\star = \arg\max_{\boldsymbol{\mu}\in\mathcal{M}} \boldsymbol{\mu}^\top \boldsymbol{\eta} - 1/2\|\boldsymbol{\mu}\|^2$$

linear constraints
(alas, exponentially many!)

quadratic objective

Conditional Gradient

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

- select a new corne
- update the (sparse)
  - Update rules: vanilla
  - Quadratic objective:

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

  [Martins et al., 2015, Nocedal and Wright, 1999,

  Vinyes and Obozinski, 2017]

Active Set achieves
finite & **linear** convergence!

Algorithms for SparseMAP

$$\boldsymbol{\mu}^\star = \arg\max_{\boldsymbol{\mu}\in\mathcal{M}} \boldsymbol{\mu}^\top \boldsymbol{\eta} - 1/2\|\boldsymbol{\mu}\|^2$$

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 $\boldsymbol{p}$
  - 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

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

Algorithms for SparseMAP

$$\boldsymbol{\mu}^\star = \arg\max_{\boldsymbol{\mu} \in \mathcal{M}} \boldsymbol{\mu}^\top \boldsymbol{\eta} - 1/2 \|\boldsymbol{\mu}\|^2$$

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 $\boldsymbol{p}$
  - 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

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

computing $\left(\frac{\partial \boldsymbol{\mu}}{\partial \boldsymbol{\eta}}\right)^\top \boldsymbol{dy}$

takes $\mathcal{O}(\dim(\boldsymbol{\mu})\, \mathrm{nnz}(\boldsymbol{p}^\star))$

Algorithms for SparseMAP

$$\boldsymbol{\mu}^\star = \arg\max_{\boldsymbol{\mu}\in\mathcal{M}} \boldsymbol{\mu}^\top \boldsymbol{\eta} - 1/2\|\boldsymbol{\mu}\|^2$$

linear constraints
(alas, exponentially many!)

quadratic objective

Conditi

[Frank and Wolfe, 1956

- select a new c
- update the (sparse) coefficients of $\boldsymbol{p}$
  - 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]

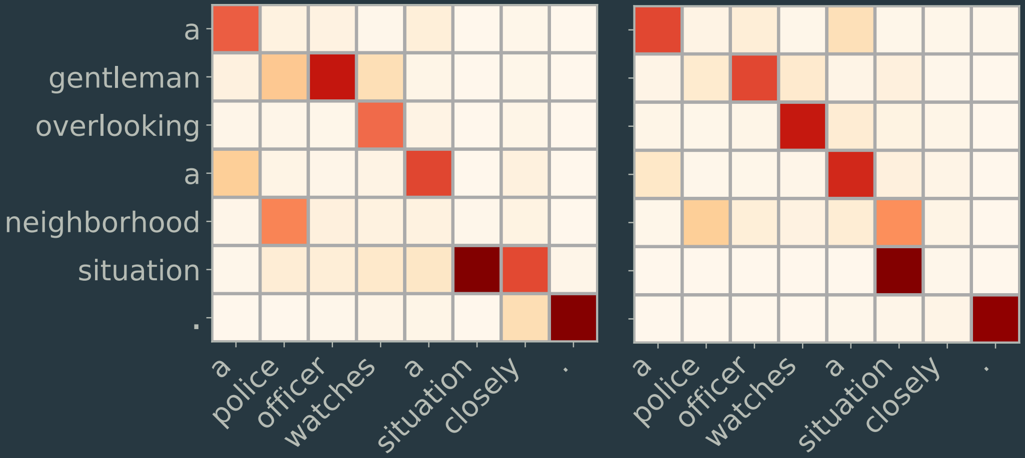
Completely modular: just add MAP

pass

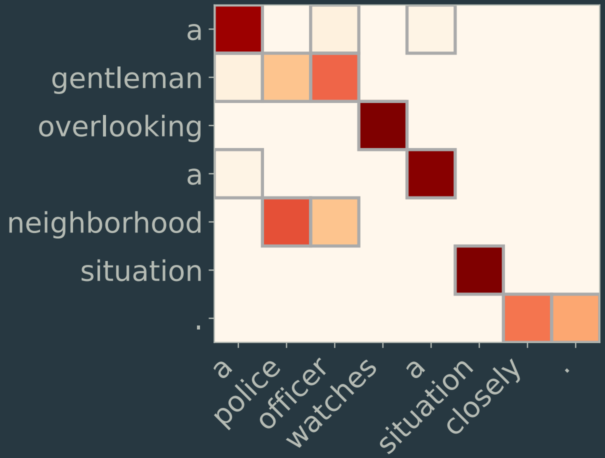
rse

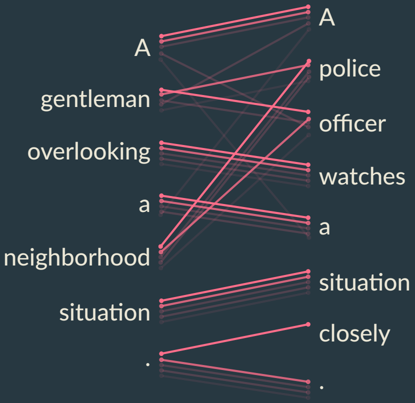
computing $\left(\frac{\partial\boldsymbol{\mu}}{\partial\boldsymbol{\eta}}\right)^\top \boldsymbol{dy}$
takes $\mathcal{O}(\dim(\boldsymbol{\mu})\,\mathrm{nnz}(\boldsymbol{p}^\star))$

[Chen et al., 2017]



[Niculae et al., 2018a]





Overview

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

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

$$L\Big(\arg\max_{\boldsymbol{z}} \pi_{\boldsymbol{\theta}}(\boldsymbol{z} \mid x)\Big)$$

- Straight Through
- SPIGOT

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

- Structured Attn. Nets
- SparseMAP

Overview

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

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

$$L\Big(\arg\max_{z} \pi_{\boldsymbol{\theta}}(\boldsymbol{z} \mid x)\Big)$$

- Straight Through
- SPIGOT

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

- Structured Attn. Nets
- SparseMAP

Model restrictions:

- $\operatorname{dom} L$ may be only $\mathcal{Z}$,
- $\nabla_{\boldsymbol{z}} L$ need not exist!

- $L(\boldsymbol{z})$ with $\boldsymbol{z} \in \mathcal{Z}$ in forward
- needs (relaxed) $\nabla_{\boldsymbol{z}} L$ in backward.

- $L(\boldsymbol{z})$ must be relaxed and differentiable.
- (sparsity gets us closer to $\mathcal{Z}$).

Structured latent variables without sampling

$$\mathbb{E}_{\boldsymbol{z}}\big[L(\boldsymbol{z})\big] = \sum_{\boldsymbol{z}\in\mathcal{Z}} L\big(\hat{y}\ \ (\boldsymbol{z})\big)\,\pi\ \ (\boldsymbol{z}\mid x)$$

Structured latent variables without sampling

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

Structured latent variables without sampling

e.g., a TreeLSTM defined by $\boldsymbol{z}$

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

Structured latent variables without sampling

e.g., a TreeLSTM defined by $\boldsymbol{z}$

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

parsing model,
using some scorer $f_{\boldsymbol{\theta}}(\boldsymbol{z}; x)$

Structured latent variables without sampling

sum over
all possible trees

e.g., a TreeLSTM defined by $\boldsymbol{z}$

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

parsing model,
using some scorer $f_{\boldsymbol{\theta}}(\boldsymbol{z}; x)$

Exponentially large sum!

Structured latent variables without sampling

sum over
all possible trees

e.g., a TreeLSTM defined by $\boldsymbol{z}$

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

parsing model,
using some scorer $f_{\boldsymbol{\theta}}(\boldsymbol{z}; x)$

How to define $\pi_{\boldsymbol{\theta}}$?

idea 1

idea 2

idea 3

Structured latent variables without sampling

sum over
all possible trees

e.g., a TreeLSTM defined by $\boldsymbol{z}$

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

parsing model,
using some scorer $f_{\boldsymbol{\theta}}(\boldsymbol{z}; x)$

How to define $\pi_{\boldsymbol{\theta}}$?

$$\sum_{h\in\mathcal{H}}$$

idea 1

idea 2

idea 3

Structured latent variables without sampling

sum over
all possible trees

e.g., a TreeLSTM defined by $\boldsymbol{z}$

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

parsing model,
using some scorer $f_{\boldsymbol{\theta}}(\boldsymbol{z}; x)$

How to define $\pi_{\boldsymbol{\theta}}$?

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

idea 1

idea 2

idea 3

Structured latent variables without sampling

sum over
all possible trees

e.g., a TreeLSTM defined by $\boldsymbol{z}$

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

parsing model,
using some scorer $f_{\boldsymbol{\theta}}(\boldsymbol{z}; x)$

How to define $\pi_{\boldsymbol{\theta}}$?

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

idea 1 $\pi_{\boldsymbol{\theta}}(\boldsymbol{z}) \propto \exp\left(f_{\boldsymbol{\theta}}(\boldsymbol{z})\right)$ softmax

idea 2

idea 3

Structured latent variables without sampling

sum over
all possible trees

e.g., a TreeLSTM defined by $\boldsymbol{z}$

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

parsing model,
using some scorer $f_{\boldsymbol{\theta}}(\boldsymbol{z}; x)$

How to define $\pi_{\boldsymbol{\theta}}$?

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

idea 1 $\pi_{\boldsymbol{\theta}}(\boldsymbol{z}) \propto \exp\left(f_{\boldsymbol{\theta}}(\boldsymbol{z})\right)$ softmax 😊

idea 2

idea 3

Structured latent variables without sampling

sum over
all possible trees

e.g., a TreeLSTM defined by $\boldsymbol{z}$

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

parsing model,
using some scorer $f_{\boldsymbol{\theta}}(\boldsymbol{z}; x)$

How to define $\pi_{\boldsymbol{\theta}}$?

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

idea 1 $\pi_{\boldsymbol{\theta}}(\boldsymbol{z}) \propto \exp\left(f_{\boldsymbol{\theta}}(\boldsymbol{z})\right)$ softmax 😱 😊

idea 2

idea 3

Structured latent variables without sampling

sum over
all possible trees

e.g., a TreeLSTM defined by $\mathbf{z}$

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

parsing model,
using some scorer $f_{\boldsymbol{\theta}}(\mathbf{z}; x)$

How to define $\pi_{\boldsymbol{\theta}}$?

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

idea 2

idea 3

Structured latent variables without sampling

sum over
all possible trees

e.g., a TreeLSTM defined by $\boldsymbol{z}$

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

parsing model,
using some scorer $f_{\boldsymbol{\theta}}(\boldsymbol{z}; x)$

How to define $\pi_{\boldsymbol{\theta}}$?

			$\sum_{h\in\mathcal{H}}$	$\frac{\partial \mathbb{E}[L(\boldsymbol{z})]}{\partial \boldsymbol{\theta}}$
idea 1	$\pi_{\boldsymbol{\theta}}(\boldsymbol{z}) \propto \exp\big(f_{\boldsymbol{\theta}}(\boldsymbol{z})\big)$	softmax	😱	😊
idea 2	$\pi_{\boldsymbol{\theta}}(\boldsymbol{z}) = 1$ if $\boldsymbol{z} = \mathrm{MAP}(f_{\boldsymbol{\theta}}(\cdot))$ else 0	argmax		
idea 3				

Structured latent variables without sampling

sum over
all possible trees

e.g., a TreeLSTM defined by $\boldsymbol{z}$

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

parsing model,
using some scorer $f_{\boldsymbol{\theta}}(\boldsymbol{z}; x)$

How to define $\pi_{\boldsymbol{\theta}}$?

			$\sum_{h\in\mathcal{H}}$	$\frac{\partial\mathbb{E}[L(\boldsymbol{z})]}{\partial\boldsymbol{\theta}}$
idea 1	$\pi_{\boldsymbol{\theta}}(\boldsymbol{z}) \propto \exp\left(f_{\boldsymbol{\theta}}(\boldsymbol{z})\right)$	softmax	😱	😊
idea 2	$\pi_{\boldsymbol{\theta}}(\boldsymbol{z}) = 1$ if $\boldsymbol{z} = \mathrm{MAP}(f_{\boldsymbol{\theta}}(\cdot))$ else 0	argmax	😊	
idea 3				

Structured latent variables without sampling

sum over
all possible trees

e.g., a TreeLSTM defined by $\boldsymbol{z}$

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

parsing model,
using some scorer $f_{\boldsymbol{\theta}}(\boldsymbol{z}; x)$

How to define $\pi_{\boldsymbol{\theta}}$?

			$\sum_{h\in\mathcal{H}}$	$\frac{\partial\mathbb{E}[L(\boldsymbol{z})]}{\partial\boldsymbol{\theta}}$
idea 1	$\pi_{\boldsymbol{\theta}}(\boldsymbol{z}) \propto \exp\big(f_{\boldsymbol{\theta}}(\boldsymbol{z})\big)$	softmax	😱	😊
idea 2	$\pi_{\boldsymbol{\theta}}(\boldsymbol{z}) = 1$ if $\boldsymbol{z} = \mathrm{MAP}(f_{\boldsymbol{\theta}}(\cdot))$ else 0	argmax	😊	😟
idea 3				

Structured latent variables without sampling

sum over
all possible trees

e.g., a TreeLSTM defined by $\boldsymbol{z}$

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

parsing model,
using some scorer $f_{\boldsymbol{\theta}}(\boldsymbol{z}; x)$

How to define $\pi_{\boldsymbol{\theta}}$?

STE / SPIGOT relax $\hat{y}$ in backward.

$$\frac{\partial \mathbb{E}\big[L(\boldsymbol{z})\big]}{\partial \boldsymbol{\theta}}$$

idea 1 $\pi_{\boldsymbol{\theta}}($ … 😊

idea 2 $\pi_{\boldsymbol{\theta}}(\boldsymbol{z}) = 1$ if $\boldsymbol{z} = \mathrm{MAP}(f_{\boldsymbol{\theta}}(\cdot))$ else 0 argmax 😊 😟

idea 3

Structured latent variables without sampling

sum over
all possible trees

e.g., a TreeLSTM defined by $\boldsymbol{z}$

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

parsing model,
using some scorer $f_{\boldsymbol{\theta}}(\boldsymbol{z}; x)$

How to define $\pi_{\boldsymbol{\theta}}$?

			$\sum_{h\in\mathcal{H}}$	$\frac{\partial \mathbb{E}[L(\boldsymbol{z})]}{\partial \boldsymbol{\theta}}$
idea 1	$\pi_{\boldsymbol{\theta}}(\boldsymbol{z}) \propto \exp\big(f_{\boldsymbol{\theta}}(\boldsymbol{z})\big)$	softmax	😱	😊
idea 2	$\pi_{\boldsymbol{\theta}}(\boldsymbol{z}) = 1$ if $\boldsymbol{z} = \mathrm{MAP}(f_{\boldsymbol{\theta}}(\cdot))$ else 0	argmax	😊	😟
idea 3		SparseMAP	😊	😊

Structured latent variables without sampling

 $= .7\times$  $+ .3\times$ 

recall our shorthand $L(\boldsymbol{z}) = L(\hat{y}_{\boldsymbol{\phi}}(\boldsymbol{z}), y)$

Structured latent variables without sampling

 $= .7\times$  $+ .3\times$  $+0\times$  $+ \ldots$

recall our shorthand $L(\boldsymbol{z}) = L(\hat{y}_{\boldsymbol{\phi}}(\boldsymbol{z}), y)$

Structured latent variables without sampling

 $= .7\times$  $+ .3\times$  $+0\times$  $+ \dots$

$\mathbb{E}[L(\boldsymbol{z})] = .7\times L($  $) + .3\times L($  $)$

recall our shorthand $L(\boldsymbol{z}) = L(\hat{y}_{\boldsymbol{\phi}}(\boldsymbol{z}), y)$

Stanford Sentiment (Accuracy)

Socher et al	
Bigram Naive Bayes	83.1
[Niculae et al., 2018b]	
TreeLSTM w/ CoreNLP	83.2
TreeLSTM w/ SparseMAP	84.7
[Corro and Titov, 2019b]	
GCN w/ CoreNLP	83.8
GCN w/ Perturb-and-MAP	84.6

Stanford Natural Language Inference (Accuracy)

[Kim et al., 2017]	
Simple Attention	86.2
Structured Attention	86.8
[Liu and Lapata, 2018]	
100D SAN -	86.8
Yogatama et al	
100D RL-SPINN	80.5
[Choi et al., 2018]	
100D ST Gumbel-Tree	82.6
300D -	85.6
600D -	86.0
[Corro and Titov, 2019b]	
Latent Tree + 1 GCN -	85.2
Latent Tree + 2 GCN -	86.2

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, $p = 21.4\%$

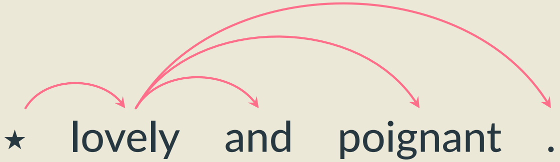
$\star$ lovely and poignant .

[Niculae et al., 2018b]

Syntax vs. Composition Order

$p = 22.6\%$

CoreNLP parse, $p = 21.4\%$



. . .

[Niculae et al., 2018b]

Syntax vs. Composition Order

$p = 22.6\%$

★ lovely and poignant .

CoreNLP parse, $p = 21.4\%$

★ lovely and poignant .

…

$p = 15.33\%$

★ a deep and meaningful film .

$p = 15.27\%$

★ a deep and meaningful film .

…

CoreNLP parse, $p = 0\%$

★ a deep and meaningful film .

Overview

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

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

$$L\Big(\arg\max_{z} \pi_{\boldsymbol{\theta}}(\mathbf{z} \mid x)\Big)$$

- Straight Through
- SPIGOT

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

- Structured Attn. Nets
- SparseMAP

Model restrictions:

- $\operatorname{dom} L$ may be only $\mathcal{Z}$,
- $\nabla_{\mathbf{z}} L$ need not exist!

- $L(\mathbf{z})$ with $\mathbf{z} \in \mathcal{Z}$ in forward
- needs (relaxed) $\nabla_{\mathbf{z}} L$ in backward.

- $L(\mathbf{z})$ must be relaxed and differentiable.
- (sparsity gets us closer to $\mathcal{Z}$).

Overview

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

- REINFORCE[SPL]
- Straight Through–Gumbel (Perturb & MAP)[SPL,MRG]
- SparseMAP[MAP+]

$$L\Big(\arg\max_{z} \pi_{\boldsymbol{\theta}}(\boldsymbol{z} \mid x)\Big)$$

- Straight Through[MAP,MRG]
- SPIGOT[MAP+]

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

- Structured Attn. Nets[MRG]
- SparseMAP[MAP+]

Computation:

SPL: Sampling. (Simple in incremental/unstructured, hard for most global structures.)

MAP: Finding the highest-scoring structure.

MRG: Marginal inference.

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).

References I

Ryan Adams. The gumbel-max trick for discrete distributions, 2013. URL `https://lips.cs.princeton.edu/the-gumbel-max-trick-for-discrete-distributions/`. Blog post.

James K Baker. Trainable grammars for speech recognition. *The Journal of the Acoustical Society of America*, 65(S1): S132–S132, 1979.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Mathieu Blondel, André FT Martins, and Vlad Niculae. Learning classifiers with Fenchel-Young losses: Generalized entropies, margins, and algorithms. In *Proc. of AISTATS*, 2019.

Samuel R. Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D. Manning, and Christopher Potts. A fast unified model for parsing and sentence understanding. In *Proc. of ACL*, 2016. doi: 10.18653/v1/P16-1139.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

Peter F Brown, Vincent J Della Pietra, Stephen A Della Pietra, and Robert L Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311, 1993.

References II

Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Si Wei, Hui Jiang, and Diana Inkpen. Enhanced LSTM for natural language inference. In *Proc. of ACL*, 2017.

Jihun Choi, Kang Min Yoo, and Sang-goo Lee. Learning to compose task-specific tree structures. In *Proc. of AAAI*, 2018.

Yoeng-Jin Chu and Tseng-Hong Liu. On the shortest arborescence of a directed graph. *Science Sinica*, 14: 1396–1400, 1965.

William John Cocke and Jacob T Schwartz. *Programming languages and their compilers*. Courant Institute of Mathematical Sciences., 1970.

Shay B Cohen, Karl Stratos, Michael Collins, Dean P Foster, and Lyle Ungar. Spectral learning of latent-variable PCFGs. In *Proc. of ACL*, 2012.

Caio Corro and Ivan Titov. Differentiable Perturb-and-Parse: Semi-Supervised Parsing with a Structured Variational Autoencoder. In *Proc. of ICLR*, 2019a.

Caio Corro and Ivan Titov. Learning latent trees with stochastic perturbations and differentiable dynamic programming. In *Proc. of ACL*, 2019b.

References III

Marco Cuturi and Mathieu Blondel. Soft-DTW: a differentiable loss function for time-series. In *Proc. of ICML*, 2017.

Jack Edmonds. Optimum branchings. *J. Res. Nat. Bur. Stand.*, 71B:233–240, 1967.

Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Nav. Res. Log.*, 3(1-2):95–110, 1956.

Serhii Havrylov, Germán Kruszewski, and Armand Joulin. Cooperative Learning of Disjoint Syntax and Semantics. In *Proc. NAACL-HLT*, 2019.

Geoffrey Hinton. Neural networks for machine learning. In *Coursera video lectures*, 2012.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-softmax. In *Proc. of ICLR*, 2017.

Roy Jonker and Anton Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38(4):325–340, 1987.

Tadao Kasami. An efficient recognition and syntax-analysis algorithm for context-free languages. *Coordinated Science Laboratory Report no. R-257*, 1966.

Yoon Kim, Carl Denton, Loung Hoang, and Alexander M Rush. Structured attention networks. In *Proc. of ICLR*, 2017.

Diederik P Kingma and Max Welling. Auto-encoding Variational Bayes. 2014.

References IV

Gustav Kirchhoff. Ueber die auflösung der gleichungen, auf welche man bei der untersuchung der linearen vertheilung galvanischer ströme geführt wird. *Annalen der Physik*, 148(12):497–508, 1847.

Harold W Kuhn. The Hungarian method for the assignment problem. *Nav. Res. Log.*, 2(1-2):83–97, 1955.

Simon Lacoste-Julien and Martin Jaggi. On the global linear convergence of Frank-Wolfe optimization variants. In *Proc. of NeurIPS*, 2015.

Zhifei Li and Jason Eisner. First-and second-order expectation semirings with applications to minimum-risk training on translation forests. In *Proc. of EMNLP*, 2009.

Yang Liu and Mirella Lapata. Learning structured text representations. *TACL*, 6:63–75, 2018.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *Proc. of ICLR*, 2016.

Jean Maillard and Stephen Clark. Latent tree learning with differentiable parsers: Shift-Reduce parsing and chart parsing. *arXiv preprint arXiv:1806.00840*, 2018.

Chaitanya Malaviya, Pedro Ferreira, and André FT Martins. Sparse and constrained attention for neural machine translation. In *Proc. of ACL*, 2018.

References V

André FT Martins and Ramón Fernandez Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *Proc. of ICML*, 2016.

André FT Martins and Vlad Niculae. Notes on latent structure models and SPIGOT. *preprint arXiv:1907.10348*.

André FT Martins, Mário AT Figueiredo, Pedro MQ Aguiar, Noah A Smith, and Eric P Xing. AD3: Alternating directions dual decomposition for MAP inference in graphical models. *JMLR*, 16(1):495–545, 2015.

Arthur Mensch and Mathieu Blondel. Differentiable dynamic programming for structured prediction and attention. In *Proc. of ICML*, 2018.

Nikita Nangia and Samuel Bowman. ListOps: A diagnostic dataset for latent tree learning. In *Proc. of NAACL SRW*, 2018.

Vlad Niculae and Mathieu Blondel. A regularized framework for sparse and structured neural attention. In *Proc. of NeurIPS*, 2017.

Vlad Niculae, André FT Martins, Mathieu Blondel, and Claire Cardie. SparseMAP: Differentiable sparse structured inference. In *Proc. of ICML*, 2018a.

References VI

Vlad Niculae, André FT Martins, and Claire Cardie. Towards dynamic computation graphs via sparse latent structure. In *Proc. of EMNLP*, 2018b.

Jorge Nocedal and Stephen Wright. *Numerical Optimization.* Springer New York, 1999.

George Papandreou and Alan L Yuille. Perturb-and-MAP random fields: Using discrete optimization to learn and sample from energy models. In *Proc. of ICCV*, 2011.

Hao Peng, Sam Thomson, and Noah A Smith. Backpropagating through structured argmax using a SPIGOT. In *Proc. of ACL*, 2018.

Ben Peters, Vlad Niculae, and André FT Martins. Sparse sequence-to-sequence models. In *Proc. of ACL*, 2019.

Slav Petrov and Dan Klein. Discriminative log-linear grammars with latent variables. In *Advances in neural information processing systems*, pages 1153–1160, 2008.

Ariadna Quattoni, Sybor Wang, Louis-Philippe Morency, Michael Collins, and Trevor Darrell. Hidden conditional random fields. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 29(10):1848–1852, 2007.

Lawrence R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. *P. IEEE*, 77(2):257–286, 1989.

References VII

Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Trans. on Acoustics, Speech, and Sig. Proc.*, 26:43–49, 1978.

Veselin Stoyanov, Alexander Ropson, and Jason Eisner. Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. In *Proc. of AISTATS*, 2011.

Ben Taskar. *Learning structured prediction models: A large margin approach*. PhD thesis, Stanford University, 2004.

Constantino Tsallis. Possible generalization of Boltzmann-Gibbs statistics. *Journal of Statistical Physics*, 52:479–487, 1988.

Leslie G Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8(2):189–201, 1979.

Tim Vieira. Gumbel-max trick, 2014. URL `https://timvieira.github.io/blog/post/2014/07/31/gumbel-max-trick/`. Blog post.

Marina Vinyes and Guillaume Obozinski. Fast column generation for atomic norm regularization. In *Proc. of AISTATS*, 2017.

Martin J Wainwright and Michael I Jordan. *Graphical models, exponential families, and variational inference.*, volume 1. Now Publishers, Inc., 2008.

References VIII

Adina Williams, Andrew Drozdov, and Samuel R Bowman. Do latent tree learning models identify meaningful structure in sentences? *TACL*, 2018.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8, 1992.

Philip Wolfe. Finding the nearest point in a polytope. *Mathematical Programming*, 11(1):128–149, 1976.

Dani Yogatama, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Wang Ling. Learning to compose words into sentences with reinforcement learning. In *Proc. of ICLR*, 2017.

Daniel H Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2): 189–208, 1967.