Neuro-Symbolic AI with Tractable Deep Generative Models

Guy Van den Broeck

NeurIPS MATH-AI Workshop - Dec 15 2023
Outline

1. Language generation *with constraints*
2. Structured output learning *with constraints*
3. Autoregressive model learning *with constraints*
Outline

1. Language generation with constraints
2. Structured output learning with constraints
3. Autoregressive model learning with constraints
Generate a sentence using "frisbee", "caught" and "dog", following the given order.
Generate a sentence using "frisbee", "caught" and "dog", following the given order.

After a perfect throw, the frisbee glided through the air, and the dog, with incredible agility, caught it mid-flight.
Generate a sentence using "frisbee", "caught" and "dog", following the given order.

After a perfect throw, the frisbee glided through the air, and the dog, with incredible agility, caught it mid-flight.

That's not correct. Generate a sentence using "frisbee", "caught" and "dog". The keywords should appear in the order as specified.

Here's the correct sentence: The dog caught the frisbee in mid-air, showing off its amazing catching skills.
A frisbee is caught by a dog.
A pair of frisbee players are caught in a dog fight.
What do we have?

Prefix: “The weather is”

Constraint $\alpha$: text contains “winter”

Model only does $p(\text{next-token} | \text{prefix}) = \begin{array}{|c|c|} \hline \text{cold} & 0.05 \\ \text{warm} & 0.10 \\ \hline \end{array}$

Train some $q(. | \alpha)$ for a specific task distribution $\alpha \sim p_{\text{task}}$ (amortized inference, encoder, masked model, seq2seq, prompt tuning,...)

Train $q(\text{next-token} | \text{prefix}, \alpha)$
What do we need?

Prefix: “The weather is”

Constraint $\alpha$: text contains “winter”

Generate from $p(\text{next-token}|\text{prefix, } \alpha) = \propto \sum_{\text{text}} p(\text{next-token, text, prefix, } \alpha)$

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cold</td>
<td>0.50</td>
</tr>
<tr>
<td>warm</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Marginalization!
Tractable Probabilistic Models

Tractable Probabilistic Models (TPMs) model *joint probability distributions* and allow *efficient* probabilistic inference.

e.g., efficient marginalization:

\[ p_{\text{TPM}}(3\text{rd token} = \text{frisbee}, 5\text{th token} = \text{dog}) \]

Easily understood as *tractable probabilistic circuits*.

For now… keep it simple… just a Hidden Markov Model (HMM)

Step 1: Distill an HMM $p_{hmm}$ that approximates $p_{gpt}$

1. HMM with 4096 hidden states and 50k emission tokens
2. Data sampled from GPT2-large (domain-adapted), minimizing $\text{KL}(p_{gpt} \parallel p_{HMM})$
3. Leverages latent variable distillation for training at scale [ICLR 23]. (Cluster embeddings of examples to estimate latent $Z_i$)

CommonGen: a Challenging Benchmark

Given 3-5 keywords, generate a sentence using all keywords, in any order and any form of inflections. e.g.,

Input: snow drive car

Reference 1: A car drives down a snow covered road.

Reference 2: Two cars drove through the snow.

Constraint $\alpha$ in CNF: $(w_{1,1} \lor \ldots \lor w_{1,d_1}) \land \ldots \land (w_{m,1} \lor \ldots \lor w_{m,d_m})$

Each clause represents the inflections for one keyword.
Computing $p(\alpha \mid x_{1:t+1})$

For constraint $\alpha$ in CNF:

$$(w_{1,1} \lor \ldots \lor w_{1,d_1}) \land \ldots \land (w_{m,1} \lor \ldots \lor w_{m,d_m})$$

e.g., $\alpha = ("swims" \lor "like swimming") \land ("lake" \lor "pool")$

**Efficient algorithm:**
For $m$ clauses and sequence length $n$, time-complexity for HMM generation is $O(2^{|m|}n)$

**Trick:** dynamic programming with clever preprocessing and local belief updates

GeLaTo
Overview

Lexical Constraint $\alpha$: sentence contains keyword “winter”

Constrained Generation: $\Pr(x_{t+1} | \alpha, x_{1:t} = "the weather is")$

Pre-trained Language Model

Tractable Probabilistic Model

Minimize KL-divergence

| $x_{t+1}$ | $\Pr_{LM}(x_{t+1} | x_{1:t})$ | $\Pr_{TPM}(\alpha | x_{t+1}, x_{1:t})$ |
|-----------|-------------------------------|-----------------------------------|
| cold      | 0.05                          | cold 0.50                        |
| warm      | 0.10                          | warm 0.01                        |

GeLaTo
Overview

**Lexical Constraint** $\alpha$: sentence contains keyword “winter”

**Constrained Generation**: $\Pr(x_{t+1} | \alpha, x_{1:t} = "the weather is")$

---

*Pre-trained Language Model*

| $x_{t+1}$ | $\Pr_{LM}(x_{t+1} | x_{1:t})$ |
|-----------|-------------------------------|
| cold      | 0.05                          |
| warm      | 0.10                          |

*Tractable Probabilistic Model*

| $x_{t+1}$ | $\Pr_{TPM}(\alpha | x_{t+1}, x_{1:t})$ |
|-----------|---------------------------------|
| cold      | 0.50                            |
| warm      | 0.01                            |

Minimize KL-divergence

$$x_{t+1} \quad p(x_{t+1} | \alpha, x_{1:t})$$

| $x_{t+1}$ | $p(x_{t+1} | \alpha, x_{1:t})$ |
|-----------|-------------------------------|
| cold      | 0.025                          |
| warm      | 0.001                          |

Step 2: Control $p_{gpt}$ via $p_{hmm}$

**Unsupervised**

Language model is not fine-tuned/prompted to satisfy constraints

By Bayes rule:

$$p_{gpt}(x_{t+1} \mid x_{1:t}, \alpha) \propto p_{gpt}(\alpha \mid x_{1:t+1}) \cdot p_{gpt}(x_{t+1} \mid x_{1:t})$$

Assume $p_{hmm}(\alpha \mid x_{1:t+1}) \approx p_{gpt}(\alpha \mid x_{1:t+1})$, we generate from:

$$p(x_{t+1} \mid x_{1:t}, \alpha) \propto p_{hmm}(\alpha \mid x_{1:t+1}) \cdot p_{gpt}(x_{t+1} \mid x_{1:t})$$

<table>
<thead>
<tr>
<th>Method</th>
<th>ROUGE-L</th>
<th>Generation Quality</th>
<th>Constraint Satisfaction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>dev</td>
<td>test</td>
<td>Coverage</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dev</td>
<td>test</td>
</tr>
<tr>
<td>InsNet (Lu et al., 2022a)</td>
<td>-</td>
<td>-</td>
<td>18.7</td>
</tr>
<tr>
<td>NeuroLogic (Lu et al., 2021)</td>
<td>-</td>
<td>41.9</td>
<td>24.7</td>
</tr>
<tr>
<td>A*esque (Lu et al., 2022b)</td>
<td>-</td>
<td>44.3</td>
<td>28.6</td>
</tr>
<tr>
<td>NADÔ (Meng et al., 2022)</td>
<td>-</td>
<td>-</td>
<td>26.2</td>
</tr>
<tr>
<td>GeLaTo</td>
<td>44.6</td>
<td>44.1</td>
<td>29.9</td>
</tr>
</tbody>
</table>

Step 2: Control $p_{gpt}$ via $p_{hmm}$

**Supervised**

Language model is fine-tuned to perform constrained generation (e.g. seq2seq)

Empirically $p_{HMM}(\alpha | x_{1:t+1}) \approx p_{gpt}(\alpha | x_{1:t+1})$ does not hold well enough;

we view $p_{HMM}(x_{t+1} | x_{1:t}, \alpha)$ and $p_{gpt}(x_{t+1} | x_{1:t})$ as classifiers trained for the same task with different biases; thus we generate from their weighted geometric mean:

$$p(x_{t+1} | x_{1:t}, \alpha) \propto p_{hmm}(x_{t+1} | x_{1:t}, \alpha)^w \cdot p_{gpt}(x_{t+1} | x_{1:t})^{1-w}$$

<table>
<thead>
<tr>
<th>Method</th>
<th>ROUGE-L dev</th>
<th>ROUGE-L test</th>
<th>BLEU-4 dev</th>
<th>BLEU-4 test</th>
<th>CIDEr dev</th>
<th>CIDEr test</th>
<th>SPICE dev</th>
<th>SPICE test</th>
<th>Coverage dev</th>
<th>Coverage test</th>
<th>Success Rate dev</th>
<th>Success Rate test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervised</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NeuroLogic (Lu et al., 2021)</td>
<td>-</td>
<td>42.8</td>
<td>-</td>
<td>26.7</td>
<td>-</td>
<td>14.7</td>
<td>-</td>
<td>30.5</td>
<td>-</td>
<td>97.7</td>
<td>-</td>
<td>93.9†</td>
</tr>
<tr>
<td>A*esque (Lu et al., 2022b)</td>
<td>-</td>
<td>43.6</td>
<td>-</td>
<td>28.2</td>
<td>-</td>
<td>15.2</td>
<td>-</td>
<td>30.8</td>
<td>-</td>
<td>97.8</td>
<td>-</td>
<td>97.9†</td>
</tr>
<tr>
<td>NADO (Meng et al., 2022)</td>
<td>44.4†</td>
<td>-</td>
<td>30.8</td>
<td>-</td>
<td>16.1†</td>
<td>-</td>
<td>32.0†</td>
<td>-</td>
<td>97.1</td>
<td>-</td>
<td>88.8†</td>
<td></td>
</tr>
<tr>
<td>GeLaTo</td>
<td>46.0</td>
<td>45.6</td>
<td>34.1</td>
<td>32.9</td>
<td>16.7</td>
<td>16.8</td>
<td>31.3</td>
<td>31.9</td>
<td>100.0</td>
<td>100.0</td>
<td>100.0</td>
<td>100.0</td>
</tr>
</tbody>
</table>

Advantages of GeLaTo:

1. Constraint $\alpha$ is guaranteed to be satisfied: for any next-token $x_{t+1}$ that would make $\alpha$ unsatisfiable, $p(x_{t+1} \mid x_{1:t}, \alpha) = 0$.

2. Training $p_{\text{hmm}}$ does not depend on $\alpha$, which is only imposed at inference (generation) time.

3. Can impose additional tractable constraints:
   - keywords follow a particular order
   - keywords appear at a particular position
   - keywords must not appear

Conclusion: you can control an intractable generative model using a tractable probabilistic circuit.
Outline

1. Language generation \textit{with constraints}

2. Structured output learning \textit{with constraints}

3. Autoregressive model learning \textit{with constraints}
Declarative Knowledge of the Output

How is the output structured? Are all possible outputs valid?

How are the outputs related to each other?

Learning this from data is inefficient. Much easier to express this declaratively.
Kareem Ahmed, Tao Li, Thy Ton, Quan Guo, Kai-Wei Chang, Parisa Kordjamshidi, Vivek Srikumar, Guy Van den Broeck and Sameer Singh. PYLON: A PyTorch Framework for Learning with Constraints

**PyTorch Code**

```python
for i in range(train_iters):
    ...
    py = model(x)
    ...
    loss = CrossEntropy(py,...)
```

1. Specify knowledge as a predicate

```python
def check(y):
    ...
    return isValid
```
PyTorch Code

```python
for i in range(train_iters):
    ...
    py = model(x)
    ...
    loss = CrossEntropy(py,...)
    loss += constraint_loss(check)(py)
```

1. Specify knowledge as a predicate
   ```python
def check(y):
    ...
    return isValid
   ```

2. Add as loss to training
   ```python
   loss += constraint_loss(check)
   ```
pyTorch Code

```python
for i in range(train_iters):
    ...
    py = model(x)
    ...
    loss = CrossEntropy(py,...)
    loss += constraint_loss(check)(py)
```

1. Specify knowledge as a predicate
   ```python
def check(y):
    ...
    return isValid
```

2. Add as loss to training
   ```python
   loss += constraint_loss(check)
   ```

3. `pylon` derives the gradients (solves a combinatorial problem)

Kareem Ahmed, Tao Li, Thy Ton, Quan Guo, Kai-Wei Chang, Parisa Kordjamshidi, Vivek Srikumar, Guy Van den Broeck and Sameer Singh. **PYLON: A PyTorch Framework for Learning with Constraints**
a) A network uncertain over both valid & invalid predictions

\[ p(y|x) \]

\[ m(\alpha) \]

b) A network uncertain over both valid & invalid predictions

\[ p(y|x) \]

\[ m(\alpha) \]

c) A network allocating most of its mass to models of constraint

\[ L^s(\alpha, p) \propto -\log \sum_{x=x} \prod_{i:x=X_i} p_i \prod_{i:x=-X_i} (1 - p_i) \]

Semantic Loss

Probability of satisfying constraint \( \alpha \) after sampling from neural net output layer \( p \)

In general: \#P-hard 😞

Do this probabilistic-logical reasoning during learning in a computation graph
\[ \alpha: \quad A \land B \Rightarrow C \]

\[ -\log(p) \quad \text{Semantic Loss} \]

\[ p \quad \text{Probability} \]
### Architecture Evaluation

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Exact Match</th>
<th>Hamming Score</th>
<th>Consistency</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18+FIL</td>
<td>55.0</td>
<td>97.7</td>
<td>56.9</td>
</tr>
<tr>
<td>ResNet-18+(L_{SL})</td>
<td>59.4</td>
<td>97.7</td>
<td>61.2</td>
</tr>
</tbody>
</table>
Semantic Probabilistic Layers

- How to give a 100% guarantee that Boolean constraints will be satisfied?
- Bake the constraint into the neural network as a special layer

- Secret sauce is again tractable circuits – computation graphs for reasoning

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Exact Match</th>
<th>Hamming Score</th>
<th>Consistency</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18+FIL</td>
<td>55.0</td>
<td>97.7</td>
<td>56.9</td>
</tr>
<tr>
<td>ResNet-18+(L_{SL})</td>
<td>59.4</td>
<td>97.7</td>
<td>61.2</td>
</tr>
<tr>
<td>ResNet-18+SPL</td>
<td>75.1</td>
<td>97.6</td>
<td>100.0</td>
</tr>
<tr>
<td>Overparam. SPL</td>
<td>78.2</td>
<td>96.3</td>
<td>100.0</td>
</tr>
</tbody>
</table>
Outline

1. Language generation *with constraints*
2. Structured output learning *with constraints*
3. Autoregressive model learning *with constraints*
Autoregressive distributions are hard…

Pr(α) is **computationally hard**, even when α is trivial:

*What is prob. that LLM ends the sentence with “NeurIPS”?*
Autoregressive distributions are hard...

Pr(α) is **computationally hard**, even when α is trivial:

*What is prob. that LLM ends the sentence with “NeurIPS”?*

Why did it work before?

$$L^s(\alpha, p) \propto -\log \sum_{x\models \alpha} \prod_{i: x\models X_i} p_i \prod_{i: x\models \neg X_i} (1 - p_i)$$

Probability of satisfying constraint α after sampling from neural net output layer p

**ASSUMING INDEPENDENT BERNOULLI’S**
Basic Idea:
Use how likely a constraint is to be satisfied around a model sample \((x)\) as a proxy for how likely it is to be satisfied under the entire distribution. Average over many such samples.
Formally, minimize the pseudo-semantic loss

$$\mathcal{L}_{\text{pseudo}}^{SL} := - \log \mathbb{E}_{\tilde{y} \sim p} \sum_{y = \alpha}^{n} \prod_{i=1}^{\alpha} p(y_i \mid \tilde{y}_{-i})$$
Formally, minimize the *pseudo-semantic loss*\n
\[ \mathcal{L}_{\text{pseudo}}^{\text{SL}} := -\log \mathbb{E}_{\tilde{y} \sim p} \sum_{y \models \alpha} \prod_{i=1}^{n} p(y_i | \tilde{y}_{-i}) \]
Formally, minimize the \textit{pseudo-semantic loss}

$$\mathcal{L}_{\text{pseudo}}^{SL} := - \log \mathbb{E}_{\tilde{y} \sim p} \sum_{x} \prod_{y \models \alpha} p(y_i | \tilde{y}_{-i})$$

How good is this approximation?

- **Local:**
  \~30 bits entropy vs \~80 for GPT-2.
- **Fidelity:**
  4 bits KL-divergence from GPT-2.
How to compute pseudo-semantic loss?

\[ p_\theta \sim abc \]
\[ \rightarrow \{ \begin{array}{ccc}
abc & abc & abc \\
\bar{abc} & \bar{abc} & \bar{abc}
\end{array} \]
\[ \rightarrow \{ \begin{array}{ccc}
p(abc) = 0.13 & p(abc) = 0.13 & p(abc) = 0.13 \\
p(\bar{abc}) = 0.15 & p(\bar{abc}) = 0.21 & p(ab\bar{c}) = 0.16
\end{array} \]
\[ \rightarrow \{ \begin{array}{ccc}
p(a|bc) = 0.46 & p(b|ac) = 0.38 & p(c|ab) = 0.45 \\
p(\bar{a}|bc) = 0.54 & p(\bar{b}|ac) = 0.62 & p(\bar{c}|ab) = 0.55
\end{array} \]
## Sudoku

<table>
<thead>
<tr>
<th>Test accuracy %</th>
<th>Exact</th>
<th>Consistent</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConvNet</td>
<td>16.80</td>
<td>16.80</td>
</tr>
<tr>
<td>ConvNet + SL</td>
<td>22.10</td>
<td>22.10</td>
</tr>
<tr>
<td>RNN</td>
<td>22.40</td>
<td>22.40</td>
</tr>
<tr>
<td>RNN + PseudoSL</td>
<td><strong>28.20</strong></td>
<td><strong>28.20</strong></td>
</tr>
</tbody>
</table>
Detoxify LLMs by disallowing bad words

Constraint $\alpha$ is a list of 403 toxic words
Evaluation is a toxicity classifier

<table>
<thead>
<tr>
<th>Models</th>
<th>Full</th>
<th>Avg. Toxicity (↓)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Toxic</td>
</tr>
<tr>
<td>GPT-2</td>
<td>0.11 ± 0.15</td>
<td>0.69 ± 0.13</td>
</tr>
<tr>
<td>GPT-2 + NeuroLogic [25]</td>
<td>0.08 ± 0.14</td>
<td>0.66 ± 0.13</td>
</tr>
<tr>
<td>GPT-2 + Word Banning</td>
<td>0.12 ± 0.16</td>
<td>0.69 ± 0.13</td>
</tr>
<tr>
<td>PseudoSL</td>
<td>0.06 ± 0.09</td>
<td>0.59 ± 0.04</td>
</tr>
<tr>
<td>PseudoSL + NeuroLogic [25]</td>
<td>0.05 ± 0.10</td>
<td>0.68 ± 0.15</td>
</tr>
<tr>
<td>PseudoSL + Word Banning</td>
<td>0.06 ± 0.09</td>
<td>0.58 ± 0.01</td>
</tr>
</tbody>
</table>
Detoxify LLMs by disallowing bad words

Constraint $\alpha$ is a list of 403 toxic words
Evaluation is a toxicity classifier

<table>
<thead>
<tr>
<th>Models</th>
<th>GPT-2</th>
<th>SGEAT</th>
<th>PseudoSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain-</td>
<td>0.12 ± 0.15</td>
<td>0.07 ± 0.09</td>
<td>0.07 ± 0.09</td>
</tr>
<tr>
<td>Adaptive</td>
<td>0.67 ± 0.12</td>
<td>0.64 ± 0.11</td>
<td>0.61 ± 0.09</td>
</tr>
<tr>
<td>Training</td>
<td>0.10 ± 0.11</td>
<td>0.06 ± 0.08</td>
<td>0.07 ± 0.09</td>
</tr>
<tr>
<td>Valid. PPL</td>
<td>24.52</td>
<td>25.93</td>
<td>26.60</td>
</tr>
</tbody>
</table>
Outline

1. Language generation with constraints
2. Structured output learning with constraints
3. Autoregressive model learning with constraints
Thanks

This was the work of many wonderful students/postdocs/collaborators!

References: http://starai.cs.ucla.edu/publications/