AI can learn from data.
But can it learn to reason?

Guy Van den Broeck

Bosch Center for AI - Oct 4 2023
Outline

1. The paradox of learning to reason from data

2. Architectures for learning and reasoning

   - Constrained language generation
   - Constrained structured prediction
Outline

1. The paradox of learning to reason from data
   - deep learning

2. Architectures for learning and reasoning
   - logical reasoning + deep learning
     a. Constrained language generation
     b. Constrained structured prediction
Can Language Models Perform Logical Reasoning?

Language Models achieve high performance on various “reasoning” benchmarks in NLP.

It is unclear whether they solve the tasks following the rules of logical deduction.

**Language Models:**

\[
\text{input} \rightarrow ? \rightarrow \text{Carol is the grandmother of Justin.}
\]

**Logical Reasoning:**

\[
\text{input} \rightarrow \text{Justin in Kristin’s son; Carol is Kristin’s mother;} \rightarrow \text{Carol is Justin’s mother’s mother;} \text{ if } X \text{ is } Y\text{'s mother’s mother then } X \text{ is } Y\text{'s grandmother} \rightarrow \text{Carol is the grandmother of Justin.}
\]
Problem Setting: SimpleLogic

The easiest of reasoning problems:

1. **Propositional logic** fragment
   - bounded vocabulary & number of rules
   - bounded reasoning depth ($\leq 6$)
   - finite space ($\approx 10^{360}$)

2. **No language variance**: templated language

3. **Self-contained**
   No prior knowledge

4. **Purely symbolic** predicates
   No shortcuts from word meaning

5. **Tractable** logic (definite clauses)
   Can always be solved efficiently

Honghua Zhang, Liunian Harold Li, Tao Meng, Kai-Wei Chang and Guy Van den Broeck. *On the Paradox of Learning to Reason from Data*, 2022
Generate textual train and test examples of the form:

Rules: If witty, then diplomatic. If careless and condemned and attractive, then blushing. If dishonest and inquisitive and average, then shy. If average, then stormy. If popular, then blushing. If talented, then hurt. If popular and attractive, then thoughtless. If blushing and shy and stormy, then inquisitive. If adorable, then popular. If cooperative and wrong and stormy, then thoughtless. If popular, then sensible. If cooperative, then wrong. If shy and cooperative, then witty. If polite and shy and thoughtless, then talented. If polite, then condemned. If polite and wrong, then inquisitive. If dishonest and inquisitive, then talented. If blushing and dishonest, then careless. If inquisitive and dishonest, then troubled. If blushing and stormy, then shy. If diplomatic and talented, then careless. If wrong and beautiful, then popular. If ugly and shy and beautiful, then stormy. If shy and inquisitive and attractive, then diplomatic. If witty and beautiful and frightened, then adorable. If diplomatic and cooperative, then sensible. If thoughtless and inquisitive, then diplomatic. If careless and dishonest and troubled, then cooperative. If hurt and witty and troubled, then dishonest. If scared and diplomatic and troubled, then average. If ugly and wrong and careless, then average. If dishonest and scared, then polite. If talented, then dishonest. If condemned, then wrong. If wrong and troubled and blushing, then scared. If attractive and condemned, then frightened. If hurt and condemned and shy, then witty. If cooperative, then attractive. If careless, then polite. If adorable and wrong and careless, then diplomatic. Facts: Alice sensible Alice condemned Alice thoughtless Alice polite Alice scared Alice average
Query: Alice is shy?
Training a transformer on SimpleLogic

(1) Randomly sample facts & rules.
Facts: B, C
Rules: A, B → D, B → E, B, C → F.

(2) Compute the correct labels for all predicates given the facts and rules.

Test accuracy for different reasoning depths

<table>
<thead>
<tr>
<th>Test</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>RP</td>
<td>99.9</td>
<td>99.8</td>
<td>99.7</td>
<td>99.3</td>
<td>98.3</td>
<td>97.5</td>
<td>95.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>LP</td>
<td>100.0</td>
<td>100.0</td>
<td>99.9</td>
<td>99.9</td>
<td>99.7</td>
<td>99.7</td>
<td>99.0</td>
</tr>
</tbody>
</table>

(1) Randomly assign labels to predicates.
True: B, C, E, F.
False: A, D.

(2) Set B, C (randomly chosen among B, C, E, F) as facts and sample rules (randomly) consistent with the label assignments.
Has the transformer learned to reason from data?

1. Easiest of reasoning problems (no variance, self-contained, purely symbolic, tractable)
2. RP/LP data covers the whole problem space
3. The learned model has almost 100% test accuracy
4. There exist transformer parameters that compute the ground-truth reasoning function:

   **Theorem 1:** For a BERT model with \( n \) layers and 12 attention heads, by construction, there exists a set of parameters such that the model can correctly solve any reasoning problem in SimpleLogic that requires at most \( n - 2 \) steps of reasoning.

Surely, under these conditions, the transformer has learned the ground-truth reasoning function!
The Paradox of Learning to Reason from Data

<table>
<thead>
<tr>
<th></th>
<th>Train</th>
<th>Test</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>RP</td>
<td>RP</td>
<td>99.9</td>
<td>99.8</td>
<td>99.7</td>
<td>99.3</td>
<td>98.3</td>
<td>97.5</td>
<td>95.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LP</td>
<td>99.8</td>
<td>99.8</td>
<td>99.3</td>
<td>96.0</td>
<td>90.4</td>
<td>75.0</td>
<td>57.3</td>
<td></td>
</tr>
<tr>
<td>LP</td>
<td>RP</td>
<td>97.3</td>
<td>66.9</td>
<td>53.0</td>
<td>54.2</td>
<td>59.5</td>
<td>65.6</td>
<td>69.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LP</td>
<td>100.0</td>
<td>100.0</td>
<td>99.9</td>
<td>99.9</td>
<td>99.7</td>
<td>99.7</td>
<td>99.0</td>
<td></td>
</tr>
</tbody>
</table>

The BERT model trained on one distribution fails to generalize to the other distribution within the same problem space.

1. If the transformer **has learned** to reason, it should not exhibit such generalization failure.

2. If the transformer **has not learned** to reason, it is baffling how it achieves near-perfect in-distribution test accuracy.

Honghua Zhang, Liunian Harold Li, Tao Meng, Kai-Wei Chang and Guy Van den Broeck. *On the Paradox of Learning to Reason from Data*, 2022
Why? Statistical Features

Monotonicity of entailment:
*Any rules can be freely added to the axioms of any proven fact.*

The more rules given, the more likely a predicate will be proven.

\[ \text{Pr}(\text{label} = \text{True} \mid \text{Rule } \# = x) \text{ should increase (roughly) monotonically with } x \]

(a) Statistics for examples generated by Rule-Priority (RP).
(b) Statistics for examples generated by Label-Priority (LP).
(c) Statistics for examples generated by uniform sampling;
Model leverages statistical features to make predictions

RP_b downsamples from RP such that \( \Pr(\text{label} = \text{True} \mid \text{rule#} = x) = 0.5 \) for all \( x \)

<table>
<thead>
<tr>
<th></th>
<th>Train</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>RP</td>
<td>99.9</td>
<td>99.8</td>
</tr>
<tr>
<td>RP_b</td>
<td>99.0</td>
<td>99.3</td>
</tr>
</tbody>
</table>

1. Accuracy drop from RP to RP_b indicates that the model is using rule# as a statistical feature to make predictions.

2. Potentially countless statistical features

3. Such features are **inherent to the reasoning problem**, cannot make data “clean”
First Conclusion

Experiments unveil the fundamental difference between

1. learning to reason, and
2. learning to achieve high performance on benchmarks using statistical features.

Be careful deploying AI in applications where this difference matters.

FAQ: Do bigger transformers solve this problem? No, already 99% accurate…

FAQ: Will reasoning emerge? Perhaps on 99% of human behavior…
Outline

1. The paradox of learning to reason from data
   - deep learning

2. Architectures for learning and reasoning
   - logical reasoning + deep learning
     a. Constrained language generation
     b. Constrained structured prediction
Outline

1. The paradox of learning to reason from data
   deep learning

2. Architectures for learning and reasoning
   logical reasoning + deep learning
   
   a. Constrained language generation
   b. Constrained structured prediction
Generative models are still hard to control
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.
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 $p(\text{next-token|prefix}) = \begin{array}{c|c}
\text{cold} & 0.05 \\
\text{warm} & 0.10 \\
\end{array}$

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

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

Prefix: “The weather is”

Constraint α: 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 (just like auto-regressive LMs) and allow efficient computation of various probabilistic queries.

e.g., efficient marginalization:

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

in particular …

\[ \sum_{\text{sentence}} p_{TPM}(\text{sentence}, \text{next-token} = \"\text{warm}\", \text{prefix} = \"\text{The weather is}\", \alpha) \]

→ Efficient conditioning given lexical constraints:

\[ p_{TPM}(\text{next-token} \mid \text{prefix}, \alpha) \]

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

1. An HMM with 4096 hidden states and 50k emission tokens

2. Train the HMM on data sampled from GPT2-large (domain-adapted, either via prompting or fine-tuning), effectively minimizing $\text{KL}(p_{gpt} \parallel p_{HMM})$

3. Leverages the latent variable distillation technique for training probabilistic circuits at scale [ICLR 23]. (Cluster embeddings of examples to estimate latent $Z_i$)
Computing $p_{hmm}(\alpha \mid x_{1:t+1})$

For $\alpha$ in conjunctive normal form (CNF):

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

where each $w_{ij}$ is a keyword (i.e. a string of tokens), representing the constraint that $w_{ij}$ appears in the generated text.

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

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

**Trick:** dynamic programming with clever preprocessing and local belief updates
CommonGen: a Challenging Benchmark

Given 3-5 concepts (keywords), our goal is to generate a sentence using all keywords, which can appear 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.

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

Each clause represents the inflections for one keyword.
GeLaTo
Overview

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

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

---

![Diagram showing Pre-trained Language Model and Tractable Probabilistic Model](image)

- **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

---

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

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

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

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

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

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

<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 test</td>
<td>BLEU-4 CIDEr SPICE</td>
<td>Coverage Success Rate</td>
</tr>
<tr>
<td>Unsupervised</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InsNet (Lu et al., 2022a)</td>
<td>- -</td>
<td>18.7</td>
<td>100.0</td>
</tr>
<tr>
<td>NeuroLogic (Lu et al., 2021)</td>
<td>- 41.9</td>
<td>- 24.7</td>
<td>- 27.5</td>
</tr>
<tr>
<td>A*esque (Lu et al., 2022b)</td>
<td>- 44.3</td>
<td>- 28.6</td>
<td>- 29.6</td>
</tr>
<tr>
<td>NADÔ (Meng et al., 2022)</td>
<td>- -</td>
<td>26.2</td>
<td>96.1</td>
</tr>
<tr>
<td>GeLaTo</td>
<td>44.6 44.1</td>
<td>29.9 29.4</td>
<td>100.0 100.0</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</th>
<th>Generation Quality</th>
<th>Constraint Satisfaction</th>
<th>Constraint Satisfaction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>dev</td>
<td>test</td>
<td>dev</td>
<td>test</td>
</tr>
<tr>
<td><strong>Supervised</strong></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>14.7</td>
</tr>
<tr>
<td>A*esque (Lu et al., 2022b)</td>
<td>-</td>
<td>43.6</td>
<td>-</td>
<td>15.2</td>
</tr>
<tr>
<td>NADO (Meng et al., 2022)</td>
<td>44.4†</td>
<td>-</td>
<td>16.1†</td>
<td>32.0†</td>
</tr>
<tr>
<td>GeLaTo</td>
<td>46.0</td>
<td>45.6</td>
<td>34.1</td>
<td>32.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Advantages of our framework:

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

2. Training $p_{hmm}$ does not depend on $\alpha$, which is only imposed at inference (generation) time. Once $p_{hmm}$ is trained, we can impose whatever $\alpha$.

3. We can impose additional tractable constraints:
   - The keywords are generated following a particular order.
   - (Some) keywords must appear at a particular position.
   - (Some) keywords must not appear in the generated sentence.

Conclusion: you can control an intractable generative model using a tractable generative model for (symbolic) reasoning.
Controlling the denoiser with a probabilistic circuit
## High-resolution image benchmarks

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Task</th>
<th>Tiramisu (ours)</th>
<th>CoPaint</th>
<th>RePaint</th>
<th>DDNM</th>
<th>DDRM</th>
<th>DPS</th>
<th>Resampling</th>
</tr>
</thead>
<tbody>
<tr>
<td>CelebA-HQ</td>
<td>Left</td>
<td>0.189</td>
<td>0.185</td>
<td>0.195</td>
<td>0.254</td>
<td>0.275</td>
<td>0.201</td>
<td>0.257</td>
</tr>
<tr>
<td></td>
<td>Top</td>
<td>0.187</td>
<td>0.182</td>
<td>0.187</td>
<td>0.248</td>
<td>0.267</td>
<td>0.187</td>
<td>0.251</td>
</tr>
<tr>
<td></td>
<td>Expand1</td>
<td><strong>0.454</strong></td>
<td>0.468</td>
<td>0.504</td>
<td>0.597</td>
<td>0.682</td>
<td>0.466</td>
<td>0.613</td>
</tr>
<tr>
<td></td>
<td>Expand2</td>
<td>0.442</td>
<td>0.455</td>
<td>0.480</td>
<td>0.585</td>
<td>0.686</td>
<td><strong>0.434</strong></td>
<td>0.601</td>
</tr>
<tr>
<td></td>
<td>V-strip</td>
<td>0.487</td>
<td>0.502</td>
<td>0.517</td>
<td>0.625</td>
<td>0.724</td>
<td>0.535</td>
<td>0.647</td>
</tr>
<tr>
<td></td>
<td>H-strip</td>
<td><strong>0.484</strong></td>
<td>0.488</td>
<td>0.517</td>
<td>0.626</td>
<td>0.731</td>
<td>0.492</td>
<td>0.639</td>
</tr>
<tr>
<td>ImageNet</td>
<td>Left</td>
<td><strong>0.286</strong></td>
<td>0.289</td>
<td>0.296</td>
<td>0.410</td>
<td>0.369</td>
<td>0.327</td>
<td>0.369</td>
</tr>
<tr>
<td></td>
<td>Top</td>
<td><strong>0.308</strong></td>
<td>0.312</td>
<td>0.336</td>
<td>0.427</td>
<td>0.373</td>
<td>0.343</td>
<td>0.368</td>
</tr>
<tr>
<td></td>
<td>Expand1</td>
<td><strong>0.616</strong></td>
<td>0.623</td>
<td>0.691</td>
<td>0.786</td>
<td>0.726</td>
<td>0.621</td>
<td>0.711</td>
</tr>
<tr>
<td></td>
<td>Expand2</td>
<td><strong>0.597</strong></td>
<td>0.607</td>
<td>0.692</td>
<td>0.799</td>
<td>0.724</td>
<td>0.618</td>
<td>0.721</td>
</tr>
<tr>
<td></td>
<td>V-strip</td>
<td>0.646</td>
<td>0.654</td>
<td>0.741</td>
<td>0.851</td>
<td>0.761</td>
<td><strong>0.637</strong></td>
<td>0.759</td>
</tr>
<tr>
<td></td>
<td>H-strip</td>
<td>0.657</td>
<td>0.660</td>
<td>0.744</td>
<td>0.851</td>
<td>0.753</td>
<td><strong>0.647</strong></td>
<td>0.774</td>
</tr>
<tr>
<td>LSUN-Bedroom</td>
<td>Left</td>
<td><strong>0.285</strong></td>
<td>0.287</td>
<td>0.314</td>
<td>0.345</td>
<td>0.366</td>
<td>0.314</td>
<td>0.367</td>
</tr>
<tr>
<td></td>
<td>Top</td>
<td><strong>0.310</strong></td>
<td>0.323</td>
<td>0.347</td>
<td>0.376</td>
<td>0.368</td>
<td>0.355</td>
<td>0.372</td>
</tr>
<tr>
<td></td>
<td>Expand1</td>
<td><strong>0.615</strong></td>
<td>0.637</td>
<td>0.676</td>
<td>0.716</td>
<td>0.695</td>
<td>0.641</td>
<td>0.699</td>
</tr>
<tr>
<td></td>
<td>Expand2</td>
<td><strong>0.635</strong></td>
<td>0.641</td>
<td>0.666</td>
<td>0.720</td>
<td>0.691</td>
<td>0.638</td>
<td>0.690</td>
</tr>
<tr>
<td></td>
<td>V-strip</td>
<td><strong>0.672</strong></td>
<td>0.676</td>
<td>0.711</td>
<td>0.760</td>
<td>0.721</td>
<td>0.674</td>
<td>0.725</td>
</tr>
<tr>
<td></td>
<td>H-strip</td>
<td>0.679</td>
<td>0.686</td>
<td>0.722</td>
<td>0.766</td>
<td>0.726</td>
<td><strong>0.674</strong></td>
<td>0.724</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td><strong>0.474</strong></td>
<td>0.481</td>
<td>0.518</td>
<td>0.596</td>
<td>0.591</td>
<td>0.489</td>
<td>0.571</td>
</tr>
</tbody>
</table>
Outline

1. The paradox of learning to reason from data
   
2. Architectures for learning and reasoning
   
   a. Constrained language generation
   b. Constrained structured prediction
Warcraft Shortest Path

// for a 12 × 12 grid, $2^{144}$ states but only $10^{10}$ valid ones!
<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>
</tbody>
</table>

- Is prediction the shortest path? This is the real task!
- Are individual edge predictions correct?
- Is output a path?

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.
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)
   ```

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,...)
    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

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

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

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

In general: \( \#P \)-hard 😞

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

- \( \log(\ ) \)

Semantic Loss
Probability
<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+$\mathcal{L}_{SL}$</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. SDD</td>
<td>78.2</td>
<td>96.3</td>
<td>100.0</td>
</tr>
</tbody>
</table>
Hierarchical Multi-Label Classification

“if the image is classified as a dog, it must also be classified as an animal”

“if the image is classified as an animal, it must be classified as either cat or dog”

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Exact Match HMCNN</th>
<th>Exact Match MLP+SPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CELL_CYCLE</td>
<td>3.05 ± 0.11</td>
<td>3.79 ± 0.18</td>
</tr>
<tr>
<td>DERISI</td>
<td>1.39 ± 0.47</td>
<td>2.28 ± 0.23</td>
</tr>
<tr>
<td>EISEN</td>
<td>5.40 ± 0.15</td>
<td>6.18 ± 0.33</td>
</tr>
<tr>
<td>EXPR</td>
<td>4.20 ± 0.21</td>
<td>5.54 ± 0.36</td>
</tr>
<tr>
<td>GASCH1</td>
<td>3.48 ± 0.96</td>
<td>4.65 ± 0.30</td>
</tr>
<tr>
<td>GASCH2</td>
<td>3.11 ± 0.08</td>
<td>3.95 ± 0.28</td>
</tr>
<tr>
<td>SEQ</td>
<td>5.24 ± 0.27</td>
<td>7.98 ± 0.28</td>
</tr>
<tr>
<td>SPO</td>
<td><strong>1.97 ± 0.06</strong></td>
<td><strong>1.92 ± 0.11</strong></td>
</tr>
<tr>
<td>DIATOMS</td>
<td>48.21 ± 0.57</td>
<td><strong>58.71 ± 0.68</strong></td>
</tr>
<tr>
<td>ENRON</td>
<td>5.97 ± 0.56</td>
<td><strong>8.18 ± 0.68</strong></td>
</tr>
<tr>
<td>IMCLEF07A</td>
<td>79.75 ± 0.38</td>
<td><strong>86.08 ± 0.45</strong></td>
</tr>
<tr>
<td>IMCLEF07D</td>
<td>76.47 ± 0.35</td>
<td><strong>81.06 ± 0.68</strong></td>
</tr>
</tbody>
</table>
SIMPLE: Gradient Estimator for $k$-Subset Sampling

We achieve lower bias and variance by exact, discrete samples and exact derivative of conditional marginals.

and SotA Learning to Explain (L2X) and sparse discrete VAE results.
Secret Sauce: Probabilistic Circuits

Tutorial (3h)

Overview Paper (80p)

Probabilistic Circuits:
A Unifying Framework for Tractable Probabilistic Models

YooJung Choi
Antonio Vergari
Guy Van den Broeck
Computer Science Department
University of California
Los Angeles, CA, USA

Contents
1 Introduction 3
2 Probabilistic Inference: Models, Queries, and Tractability 4
  2.1 Probabilistic Models 5
  2.2 Probabilistic Queries 6
  2.3 Tractable Probabilistic Inference 8
  2.4 Properties of Tractable Probabilistic Models 9

https://youtu.be/2RAG5-L9R70

Outline

1. The paradox of learning to reason from data

2. Architectures for learning and reasoning

   logical (and probabilistic) reasoning + deep learning

   a. Constrained language generation
   b. Constrained structured prediction
Thanks

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

Honghua                  Kareem                    Zhe                    Meihua                    Anji
