Neural Networks in NLP:
The Curse of Indifferentiability

Lili Mou
doublepower.mou@gmail.com
http://sei.pku.edu.cn/~moull12
Outline

• Preliminary

• Indifferentiability, solutions, and applications
  – The curse of indifferentiability
  – Solutions: Attention, reinforcement learning, etc.
  – Applications: Sequence-level objective, SeqGAN, etc.

• A case study in semantic parsing
The Curse of Indifferentiability

• Characters are discrete!
• Words are discrete!
• Phrases are discrete!
• Sentences are discrete!
• Paragraphs are discrete!
• All symbols are discrete!

• Word embeddings are continuous but are nothing!
Indifferentiability
Indifferentiability

Risk (e.g., BLEU)

Output (softmax)

Hidden layer(s) w/ LSTM units

Input (one-hot)

(a) Input sequence  (b) Output sequence
Indifferentiability

Risk (e.g., BLEU)

Output (softmax)

Hidden layer(s) w/ LSTM units

Input (one-hot)

(a) Input sequence (b) Output sequence
Indifferentiability

- Input: word embeddings 😊
- Output: argmax p(word) 😞
- Risk: a function of output 😞
Outline

• Preliminary

• **Indifferentiability, solutions, and applications**
  – The curse of indifferentiability
  – **Solutions**: Attention, reinforcement learning, etc.
  – Applications: Sequence-level objective, SeqGAN, etc.

• A case study in semantic parsing
Solution: Attempt #1

Classification of a particular word

=> Regression of word embeddings
Solution: Attempt #1

Classification of a particular word

=> Regression of word embeddings

• Total failure (but why?)
Solution: Attempt #2

- Attention (weighted sum)
Solution: Attempt #3

- Reinforcement learning (Trial-and-error)
  - Sample an action (sequence)
  - See what the reward is
**REINFORCE**

- Define an external cost function on a generated sequence
- Generate words by sampling
- Take the derivative of generated samples

\[
L_\theta = - \sum_{w_1^g, \ldots, w_T^g} p_\theta(w_1^g, \ldots, w_T^g) r(w_1^g, \ldots, w_T^g) = -\mathbb{E}_{[w_1^g, \ldots, w_T^g] \sim p_\theta} r(w_1^g, \ldots, w_T^g)
\]

- \( \partial J = \sum_{w} \left[ \partial p(\mathbf{w}) \right] r(\mathbf{w}) = \sum_{w} p(\mathbf{w}) [\partial \log p(\mathbf{w})] r(\mathbf{w}) \)

Caveats

- REINFORCE may be extremely difficult to train
  - Hard to get started
  - Poor local optima
  - Sensitive to hyperparameters

- Supervised pretraining
Solution: Attempt #4

- Gumble softmax
  - Sample from a class distribution
    \[ z = \text{one\_hot} \left( \arg\max_i [g_i + \log \pi_i] \right) \]
    where
    \[ g = - \log(-\log(u)) \]
  - Softmax approximation
    \[ y_i = \frac{\exp((\log(\pi_i) + g_i)/\tau)}{\sum_{j=1}^{k} \exp((\log(\pi_j) + g_j)/\tau)} \]

Solution: Attempt #4

- Interpolation between onehot and uniform (with class distribution information)
Outline

- Preliminary

- Indifferentiability, solutions, and applications
  - The curse of indifferentiability
  - Solutions: Attention, reinforcement learning, etc.
  - Applications: Sequence-level obj., SeqGAN, etc.

- A case study in semantic parsing
Application: Sequence-Level Objective

- REINFORCE towards BLEU
- Annealing
  - For 1..T words
  - Supervised training: 1..t
  - RL: t+1..T
## Results

<table>
<thead>
<tr>
<th>TASK</th>
<th>XENT</th>
<th>DAD</th>
<th>E2E</th>
<th>MIXER</th>
</tr>
</thead>
<tbody>
<tr>
<td>summarization</td>
<td>13.01</td>
<td>12.18</td>
<td>12.78</td>
<td>16.22</td>
</tr>
<tr>
<td>translation</td>
<td>17.74</td>
<td>20.12</td>
<td>17.77</td>
<td>20.73</td>
</tr>
<tr>
<td>image captioning</td>
<td>27.8</td>
<td>28.16</td>
<td>26.42</td>
<td>29.16</td>
</tr>
</tbody>
</table>

Application: SeqGAN

Generative Adversarial Network

• Two agents:
  – **Generative model**: Generate new samples that are as similar as the data
  – **Discriminative model**: Distinguish samples in disguise

• Each agent takes a step in turn
Objective of GAN

\[
\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]
\]

- **G(z):** A generated sample from distribution z
- **D(x) = Estimated (by D) prob. that x is a real data sample**
  - \(D(x)=1: D\) regards x as a training sample w.p.1
  - \(D(x)=0: D\) regards x as a generative sample w.p.1
Objective of GAN

\[
\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]
\]

Algorithm

\begin{align*}
\text{for number of training iterations do} & \\
\text{for } k \text{ steps do} & \\
& \bullet \text{Sample minibatch of } m \text{ noise samples } \{z^{(1)}, \ldots, z^{(m)}\} \text{ from noise prior } p_g(z). \\
& \bullet \text{Sample minibatch of } m \text{ examples } \{x^{(1)}, \ldots, x^{(m)}\} \text{ from data generating distribution } p_{\text{data}}(x). \\
& \bullet \text{Update the discriminator by ascending its stochastic gradient:} \\
& \quad \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \right]. \\
\text{end for} & \\
& \bullet \text{Sample minibatch of } m \text{ noise samples } \{z^{(1)}, \ldots, z^{(m)}\} \text{ from noise prior } p_g(z). \\
& \bullet \text{Update the generator by descending its stochastic gradient:} \\
& \quad \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right). \\
\text{end for}
\end{align*}
Curse of Indifferentiability

\[
\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]
\]

Algorithm

\[
\begin{align*}
\text{for number of training iterations do} & \\
\quad \text{for } k \text{ steps do} & \\
\quad \quad \text{• Sample minibatch of } m \text{ noise samples } \{z^{(1)}, \ldots, z^{(m)}\} \text{ from noise prior } p_g(z). & \\
\quad \quad \text{• Sample minibatch of } m \text{ examples } \{x^{(1)}, \ldots, x^{(m)}\} \text{ from data generating distribution } p_{\text{data}}(x). & \\
\quad \quad \text{• Update the discriminator by ascending its stochastic gradient:} & \\
\quad \quad \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D(x^{(i)}) + \log \left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \right]. & \\
\quad \text{end for} & \\
\quad \text{• Sample minibatch of } m \text{ noise samples } \{z^{(1)}, \ldots, z^{(m)}\} \text{ from noise prior } p_g(z). & \\
\quad \text{• Update the generator by descending its stochastic gradient:} & \\
\quad \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log \left(1 - D\left(G\left(z^{(i)}\right)\right)\right). & \\
\end{align*}
\]
Solution

- REINFORCE!

Table 2: Chinese poem generation performance comparison.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Human score</th>
<th>$p$-value</th>
<th>BLEU-2</th>
<th>$p$-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLE</td>
<td>0.4165</td>
<td></td>
<td>0.6670</td>
<td></td>
</tr>
<tr>
<td>SeqGAN</td>
<td>0.5356</td>
<td>0.0034</td>
<td>0.7389</td>
<td>$&lt; 10^{-6}$</td>
</tr>
<tr>
<td>Real data</td>
<td>0.6011</td>
<td></td>
<td>0.746</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Obama political speech generation performance.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>BLEU-3</th>
<th>$p$-value</th>
<th>BLEU-4</th>
<th>$p$-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLE</td>
<td>0.519</td>
<td>$&lt; 10^{-6}$</td>
<td>0.416</td>
<td>0.00014</td>
</tr>
<tr>
<td>SeqGAN</td>
<td>0.556</td>
<td></td>
<td>0.427</td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Music generation performance comparison.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>BLEU-4</th>
<th>$p$-value</th>
<th>MSE</th>
<th>$p$-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLE</td>
<td>0.9210</td>
<td>$&lt; 10^{-6}$</td>
<td>22.38</td>
<td>0.00034</td>
</tr>
<tr>
<td>SeqGAN</td>
<td>0.9406</td>
<td></td>
<td>20.62</td>
<td></td>
</tr>
</tbody>
</table>

- Does SeqGAN provide a more powerful density estimator?
Application: Rationale neural predictions

Lei, Tao, Regina Barzilay, and Tommi Jaakkola.
Objective

\[ \mathcal{L}(z, x, y) = \| \text{enc}(z, x) - y \|^2_2 \]

\[ \Omega(z) = \lambda_1 \| z \| + \lambda_2 \sum_{t} |z_t - z_{t-1}| \]

\[ \text{cost}(z, x, y) = \mathcal{L}(z, x, y) + \Omega(z) \]
Training

- REINFORCE!
Red: appearance
Blue: Smell
Green: Palate

very dark beer. pours a nice finger and a half of creamy foam and stays throughout the beer. smells of coffee and roasted malt. has a major coffee-like taste with hints of chocolate. if you like black coffee, you will love this porter. creamy smooth mouthfeel and definitely gets smoother on the palate once it warms. it's an ok porter but i feel there are much better one's out there.