Imputer:
Sequence Modeling via Imputation and Dynamic Programming

William Chan, Chitwan Saharia, Geoffrey Hinton, Mohammad Norouzi, Navdeep Jaitly
Google Brain

Presenter: Desh Raj
Overview

• Preliminary: end-to-end ASR model types
• The Imputer model
  • Training scheme
  • Decoding policies
• Experimental results
Preliminary
3 kinds of popular end-to-end ASR models:

1. CTC-based
2. RNN-Transducer
3. Encoder-decoder (with attention)

\[ P(y|x) = \sum_{\hat{y} \in \mathcal{B}(y,x)} \prod_{t=1}^{T} P(\hat{y}_t|x) \]

CTC

Conditional independence assumption
So all tokens can be generated in parallel

\[
P(y|x) = \sum_{\hat{y} \in \mathcal{B}(y,x)} \prod_{t=1}^{T} P(\hat{y}_t | x)
\]
RNN-Transducer

![Diagram of RNN-Transducer model]

\[
P(y|t, u) = \text{Softmax}(z_{t,u})
\]

\[
h_{t,u}^{\text{joint}} = \tanh(Ah_{t}^{\text{enc}} + Bp_{u} + b)
\]

\[
z_{t,u} = Dh_{t,u}^{\text{joint}} + d
\]

RNN-Transducer

Generated token depends on previous output
So have to generate sequentially

\[
\begin{align*}
\mathbf{h}_{t,u}^{\text{joint}} &= \tanh(A\mathbf{h}_{t}^{\text{enc}} + B\mathbf{p}_{u} + b) \\
\mathbf{z}_{t,u} &= D\mathbf{h}_{t,u}^{\text{joint}} + d
\end{align*}
\]

Encoder-decoder (with attention)

\[ P(y_u | y_{u-1}, \ldots, y_0, x) \]

\[ \beta_{t,u} = \langle \phi(h_t^{enc}), \psi(h_{u-1}^{att}) \rangle \]

\[ \alpha_{t,u} = \frac{e^{\beta_{t,u}}}{\sum_{i=1}^{T} e^{\beta_{i,u}}} \]

\[ c_u = \sum_t \alpha_{t,u} h_t^{enc} \]
Encoder-decoder (with attention)

Generated token depends on previous output

So have to generate sequentially

\[ \beta_{t,u} = \langle \phi(h^\text{enc}_t), \psi(h^\text{att}_{u-1}) \rangle \]

\[ \alpha_{t,u} = \frac{e^{\beta_{t,u}}}{\sum_{i=1}^{T} e^{\beta_{i,u}}} \]

\[ c_u = \sum_t \alpha_{t,u} h^\text{enc}_t \]

Trade-off

• Speed vs. performance tradeoff

• For faster inference, sequence generation must be independent of previous tokens e.g. CTC

• But for better performance, conditionally dependent sequence generation is required e.g. RNN-T
Trade-off

• Speed vs. performance tradeoff

• For faster inference, sequence generation must be independent of previous tokens e.g. CTC

• But for better performance, conditionally dependent sequence generation is required e.g. RNN-T
Trade-off

- Speed vs. performance tradeoff

- For **faster inference**, sequence generation must be independent of previous tokens e.g. CTC

- But for **better performance**, conditionally dependent sequence generation is required e.g. RNN-T
Motivating Question

• Can we have something in between?

• A model which is not fully autoregressive (i.e. does not take $O(n)$ steps during inference)

• But also does not have conditional independence assumptions
“Yes, we can.”

– Authors of the Imputer paper
A bit more on CTC

\[ p(Y \mid X) = \sum_{A \in A_{X,Y}} \prod_{t=1}^{T} p_t(a_t \mid X) \]

The CTC conditional probability marginalizes over the set of valid alignments computing the probability for a single alignment step-by-step.

The Imputer Decoding Model

1. Initial alignment is filled with mask tokens $\varnothing$.

Output sequence length $= |x|$ is divided into blocks of size $B$

A block is generated independent of other blocks
The Imputer Decoding Model

1. Initial alignment is filled with mask tokens ∅.

Output sequence length = |x| is divided into blocks of size B

A block is generated independent of other blocks

B = 1 ?
The Imputer Decoding Model

1. Initial alignment is filled with mask tokens $\varnothing$.

Output sequence length $= |x|$ is divided into blocks of size $B$

A block is generated independent of other blocks

$$B = 1 \ ? \quad \text{CTC}$$

$$B = |x| \ ?$$
The Imputer Decoding Model

1. Initial alignment is filled with mask tokens $\emptyset$.

A block is generated independent of other blocks

Output sequence length $= |x|$ is divided into blocks of size $B$

$B = 1$ ? CTC

$B = |x|$ ? Fully autoregressive
The Imputer Decoding Model

1. Initial alignment is filled with mask tokens $\emptyset$.

```
    $\emptyset$ $\emptyset$ $\emptyset$ $\emptyset$ $\emptyset$ $\emptyset$ $\emptyset$ $\emptyset$ $\emptyset$ $\emptyset$ $\emptyset$ $\emptyset$ $\emptyset$
```

2. Imputer conditions on a previous alignment.

```
    A $\emptyset$ $\emptyset$ $-$ $\emptyset$ $\emptyset$ E $\emptyset$ $\emptyset$ $\emptyset$ F
```

New alignment for the block is generated depending on input $x$ and previous alignment.
The Imputer Decoding Model

1. Initial alignment is filled with mask tokens $\emptyset$.

\[
\begin{array}{cccccccccccc}
\emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\
\end{array}
\]

2. Imputer conditions on a previous alignment.

\[
\begin{array}{cccccccccccc}
A & \emptyset & \emptyset & \emptyset & \_ & \emptyset & \emptyset & E & \emptyset & \emptyset & \emptyset & F \\
\end{array}
\]

3. Imputer generates a new alignment. For each block, the token with the largest probability is selected and merged with the existing alignment.

\[
\begin{array}{cccccccccccc}
A & B & \emptyset & \emptyset & \_ & C & D & E & \emptyset & \emptyset & \_ & F \\
\end{array}
\]

Keep the token with largest probability and merge with existing alignment
The Imputer Decoding Model

1. Initial alignment is filled with mask tokens $\varnothing$.
   
   \[
   \begin{array}{cccccccccc}
   \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \\
   \end{array}
   \]

2. Imputer conditions on a previous alignment.
   
   \[
   \begin{array}{cccccccccc}
   A & \varnothing & \varnothing & \varnothing & - & \varnothing & \varnothing & E & \varnothing & \varnothing & \varnothing & F \\
   \end{array}
   \]

3. Imputer generates a new alignment. For each block, the token with the largest probability is selected and merged with the existing alignment.
   
   \[
   \begin{array}{cccccccccc}
   A & B & \varnothing & \varnothing & - & C & D & E & \varnothing & \varnothing & - & F \\
   \end{array}
   \]

1 token “committed” in each step
The Imputer Decoding Model

1. Initial alignment is filled with mask tokens \( \emptyset \).
2. Imputer conditions on a previous alignment.
3. Imputer generates a new alignment. For each block, the token with the largest probability is selected and merged with the existing alignment.

1 token “committed” in each step

So how many steps would we need in total to get the sequence?
The Imputer Decoding Model

1. Initial alignment is filled with mask tokens $\emptyset$.

```
\emptyset \emptyset \emptyset \emptyset \emptyset \emptyset \emptyset \emptyset \emptyset \emptyset \emptyset
```

2. Imputer conditions on a previous alignment.

```
A \emptyset \emptyset \emptyset \_ \emptyset \emptyset E \emptyset \emptyset \emptyset F
```

3. Imputer generates a new alignment. For each block, the token with the largest probability is selected and merged with the existing alignment.

```
A B \emptyset \emptyset \_ C D E \emptyset \emptyset \emptyset \_ F
```

Sequence is generated in B steps
The Imputer Decoding Model

1. Initial alignment is filled with mask tokens $\emptyset$.

2. Imputer conditions on a previous alignment.

3. Imputer generates a new alignment. For each block, the token with the largest probability is selected and merged with the existing alignment.

In a block, token generation is dependent on other tokens within the block.
The model

\[ p_\theta(a | \tilde{a}, x) = \prod_i p(a_i | \tilde{a}, x; \theta) \]
The model

By conditioning on previous alignment, the tokens become conditioned on each other

\[ p_\theta(a|\tilde{a}, x) = \prod_i p(a_i|\tilde{a}, x; \theta) \]
Training objective

\[ \log p_\theta(y|x) = \log \sum_{a \in \beta(y)} \sum_{\tilde{a} \in \gamma(a)} p_\theta(a|\tilde{a}, x) \]
Training objective

\[ \log p_\theta(y|x) = \log \sum_{a \in \beta(y)} \sum_{\tilde{a} \in \gamma(a)} p_\theta(a|\tilde{a}, x) \]
Training objective

\[ J(\theta) = \mathbb{E}_{a \sim q}[\mathbb{E}_{\tilde{a} \sim r}[\log p_{\theta}(a|\tilde{a}, x)]] \]
Training objective

\[ J(\theta) = \mathbb{E}_{a \sim q}[\mathbb{E}_{\tilde{a} \sim r}[\log p_\theta(a | \tilde{a}, x)]] \]

1. How to sample alignments from q?
2. How to sample masks from r?
Alignment policy

• Suppose we have an “expert” model pretrained e.g. CTC

• **Method 1**: Get all alignments, store them offline, and sample from this.

• **Method 2**: Get best alignment and add noise

\[
J(\theta) = \mathbb{E}_{a \sim q}[\mathbb{E}_{\tilde{a} \sim r}[^{\log} p_\theta(a|\tilde{a}, x)]]
\]
Masking policy

- **Method 1**: Uniform or Bernoulli distribution -> every block may have different number of masked tokens

- **Method 2**: Choose $b$ in $[0,B)$ and mask out $b$ tokens in each block randomly

\[ J(\theta) = \mathbb{E}_{a \sim q} \left[ \mathbb{E}_{\tilde{a} \sim r} \log p_\theta(a | \tilde{a}, x) \right] \]
How to train your Imputer?

1. Imitation learning

   • Simply learn to copy the expert CTC model

   
   $J_{IM}(\theta) = \mathbb{E}_{a \sim q_\phi} \left[ \mathbb{E}_{\tilde{a} \sim q_\phi} \left[ \log p_\theta(a | \tilde{a}, x) \right] \right]$
How to train your Imputer?

2. Dynamic programming

- Marginalize over all possible compatible alignments

\[ J_{DP}(\theta) = \mathbb{E}_{a \sim q_{\phi'}} \left[ \mathbb{E}_{\tilde{a} \sim r} \left[ \log \sum_{a' \in \beta'(\tilde{a}, a)} p_{\theta}(a'|\tilde{a}, x) \right] \right] \]
Model architecture
New Alignment Posterior

\[ p_\theta(a|x, \tilde{a}) \]

Softmax

Self-Attention

Convolution

\( x: \) Audio Filter Banks

Embedding

\( \tilde{a}: \) Prior Alignment
(contains masked out tokens)
Experimental results
### WSJ - 82h

### LibriSpeech - 960h

**Table 1.** Wall Street Journal Character Error Rate (CER) and Word Error Rate (WER).

<table>
<thead>
<tr>
<th>Model</th>
<th>CER</th>
<th>WER</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq2seq</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bahdanau et al. (2016a)</td>
<td>6.4</td>
<td>18.6</td>
<td>n</td>
</tr>
<tr>
<td>Bahdanau et al. (2016b)</td>
<td>5.9</td>
<td>18.0</td>
<td>n</td>
</tr>
<tr>
<td>Chorowski &amp; Jaitly (2017)</td>
<td>-</td>
<td>10.6</td>
<td>n</td>
</tr>
<tr>
<td>Zhang et al. (2017)</td>
<td>-</td>
<td>10.5</td>
<td>n</td>
</tr>
<tr>
<td>Chan et al. (2017)</td>
<td>-</td>
<td>9.6</td>
<td>n</td>
</tr>
<tr>
<td>Kim et al. (2017)</td>
<td>7.4</td>
<td>-</td>
<td>n</td>
</tr>
<tr>
<td>Serdyuk et al. (2018)</td>
<td>6.2</td>
<td>-</td>
<td>n</td>
</tr>
<tr>
<td>Tjandra et al. (2018)</td>
<td>6.1</td>
<td>-</td>
<td>n</td>
</tr>
<tr>
<td>Sabour et al. (2019)</td>
<td>3.1</td>
<td>9.3</td>
<td>n</td>
</tr>
</tbody>
</table>

| CTC                          |     |     |            |
| Graves & Jaitly (2014)       | 8.4 | 27.3| 1          |
| Liu et al. (2017)            | -   | 16.7| 1          |
| CTC (Our Work)               | 5.6 | 15.2| 1          |

| Imputer (IM)                 | 6.2 | 16.5| 8          |
| Imputer (DP)                 | 4.9 | 12.7| 8          |

**Table 2.** LibriSpeech test-clean and test-other Word Error Rate (WER).

<table>
<thead>
<tr>
<th>Method</th>
<th>clean</th>
<th>other</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq2seq</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zeyer et al. (2018a)</td>
<td>4.9</td>
<td>15.4</td>
<td>n</td>
</tr>
<tr>
<td>Zeyer et al. (2018b)</td>
<td>4.7</td>
<td>15.2</td>
<td>n</td>
</tr>
<tr>
<td>Irie et al. (2019)</td>
<td>4.7</td>
<td>13.4</td>
<td>n</td>
</tr>
<tr>
<td>Sabour et al. (2019)</td>
<td>4.5</td>
<td>13.3</td>
<td>n</td>
</tr>
<tr>
<td>Luscher et al. (2019)</td>
<td>4.4</td>
<td>13.5</td>
<td>n</td>
</tr>
<tr>
<td>Park et al. (2019)</td>
<td>4.1</td>
<td>12.5</td>
<td>n</td>
</tr>
</tbody>
</table>

| ASG/CTC            |       |       |            |
| Collobert et al. (2016)| 7.2  | -     | 1          |
| Liptchinsky et al. (2017)| 6.7  | -     | 1          |
| CTC (Our Work)     | 4.6   | 13.0  | 1          |

| Imputer (IM)       | 5.5   | 14.6  | 8          |
| Imputer (DP)       | 4.0   | 11.1  | 8          |
Some other stuff

- Block size 8 found to be best for training and inference
- Too large blocks can cause training issues
- All masking policies (Bernoulli, Uniform, Block) perform similar
Stuff which did not work :) 

- Training with stale model samples instead of CTC expert
- Greedy decoding, simulated annealing decoding
Key takeaways

• Want something between CTC and fully autoregressive models

• Inference in constant time (B steps) + works better than both

• How to make things non-autoregressive? Use MASKING. (see BERT, Mask-Predict etc.)