Over-Generative FST n-gram for OOV Word Recognition

Ronaldo Messina and Christopher Kermorvant

{rm,ck}@a2ia.com
Outline

1 Introduction
2 Hybrid grammars
3 Over-generative transducers
4 Experiments
5 Conclusion
Outline

1 Introduction
2 Hybrid grammars
3 Over-generative transducers
4 Experiments
5 Conclusion
Out of Vocabulary (OOV) words are a common issue in recognition:
Out of Vocabulary (OOV) words are a common issue in recognition:

- New words could appear, or just not be present in existing data
Out of Vocabulary (OOV) words are a common issue in recognition:
- New words could appear, or just not be present in existing data
- More data can be expensive to collect/transcribe
Out of Vocabulary (OOV) words are a common issue in recognition:

- New words could appear, or just not be present in existing data
- More data can be expensive to collect/transcribe
- No finite amount of data would be enough to reliably estimate parameters of statistical grammars (n-grams)
Out of Vocabulary (OOV) words are a common issue in recognition:
- New words could appear, or just not be present in existing data
- More data can be expensive to collect/transcribe
- No finite amount of data would be enough to reliably estimate parameters of statistical grammars (n-grams)

One possible solution: hybrid grammars [1]
Out of Vocabulary (OOV) words are a common issue in recognition:
- New words could appear, or just not be present in existing data
- More data can be expensive to collect/transcribe
- No finite amount of data would be enough to reliably estimate parameters of statistical grammars (n-grams)

One possible solution: hybrid grammars [1]
- Two-level grammar: word (top-level) and character-level
Out of Vocabulary (OOV) words are a common issue in recognition:
- New words could appear, or just not be present in existing data
- More data can be expensive to collect/transcribe
- No finite amount of data would be enough to reliably estimate parameters of statistical grammars (n-grams)

One possible solution: **hybrid grammars** [1]
- Two-level grammar: word (top-level) and character-level
- Word-level: a 3-gram with $K$ most-common words plus entry point for OOV words
Out of Vocabulary (OOV) words are a common issue in recognition:
- New words could appear, or just not be present in existing data
- More data can be expensive to collect/transcribe
- No finite amount of data would be enough to reliably estimate parameters of statistical grammars (n-grams)

One possible solution: **hybrid grammars** [1]
- Two-level grammar: word (top-level) and character-level
- Word-level: a 3-gram with $K$ most-common words plus entry point for OOV words
- Char-level: a longer n-gram (e.g. $n = 10$) for OOV words
Recognition system framework

Components of the system:
Introduction

Recognition system framework

Components of the system:
- Recurrent Neural Network (RNN) as Optical Model [2]
Recognition system framework

Components of the system:

- Recurrent Neural Network (RNN) as Optical Model [2]
Introduction

Recognition system framework

Components of the system:

- Recurrent Neural Network (RNN) as Optical Model [2]
- The search space where the decoder finds the best hypothesis is the composition of the RNN’s predictions and $H \circ C \circ L \circ G$
Recognition system framework

Components of the system:

- Recurrent Neural Network (RNN) as Optical Model [2]
- The search space where the decoder finds the best hypothesis is the composition of the RNN’s predictions and $H \circ C \circ L \circ G$
- $H \circ C \circ L \circ G$ is a static transducer in our setup
Practical issues

When using static hybrid word/OOV grammars:

- The static transducer is easily over 1TB in size
- Our proposal: combine paths in the hybrid fst ⇒ fewer instantiations of the OOV n-gram
- The result is an over-generative transducer: paths that did not exist in the original n-gram are created
Practical issues

When using static hybrid word/OOV grammars:

- The static transducer is easily over 1TB in size
  - \( \Rightarrow \) multiple instances of the character-level n-gram
Practical issues

When using static hybrid word/OOV grammars:
- The static transducer is easily over 1TB in size
  - $\Rightarrow$ multiple instances of the character-level n-gram
Practical issues

When using static hybrid word/OOV grammars:

- The static transducer is easily over 1TB in size
  - $\Rightarrow$ multiple instances of the character-level n-gram

Our proposal:
Practical issues

When using static hybrid word/OOV grammars:
- The static transducer is easily over 1TB in size
  - ⇒ multiple instances of the character-level n-gram

Our proposal:
- Combine paths in the hybrid fst ⇒ fewer instantiations of the OOV n-gram
When using static hybrid word/OOV grammars:

- The static transducer is easily over 1TB in size
  - ⇒ multiple instances of the character-level n-gram

Our proposal:

- Combine paths in the hybrid fst ⇒ fewer instantiations of the OOV n-gram
- The result is an over-generative transducer: paths that did not exist in the original n-gram are created
Outline

1 Introduction
2 Hybrid grammars
3 Over-generative transducers
4 Experiments
5 Conclusion
Hybrid grammars

Principle

- **Key idea**: [1] Two-level grammar: the first deals with the “usual” words and the second with the OOV words
  - The first is a 3-gram on words and the other a 10-gram on characters
**Key idea: [1]** Two-level grammar: the first deals with the “usual” words and the second with the OOV words
- The first is a 3-gram on words and the other a 10-gram on characters
- Limit the size of vocabulary to create a placeholder for OOV words in the word-level n-gram: \(<\text{unk}>\)
- The resulting OOV words are used to estimate a 10-gram on characters
- More data can be included to improve the representation of morphological aspects of the language
- Could deal with word tokens that are not frequent: hyphenated words, dates, codes, etc.
Outline

1. Introduction
2. Hybrid grammars
3. Over-generative transducers
4. Experiments
5. Conclusion
Size issues

- The number of $<\text{unk}>$ instances depends on vocabulary size (and n-gram pruning)
- Static fst size dominated by the number of $<\text{unk}>$
- Any practical vocabulary size / char-level n-gram results in a gargantuan fst
- Pruning is used to reduce the OOV n-gram
Dealing with transducer size

Our (first) solution is to factorize the LG transducer:

Create a $L \circ G_w$ for the word-level 3-gram, use a dummy symbol in $L$ for $\text{<unk}>$.

Merge all paths through $\text{<unk>}$ in $L \circ G_w$.

Create a $L \circ G_c$ for the char-level 10-gram.

Plug $L \circ G_c$ where the dummy symbol appears.

But merging all paths through $\text{<unk>}$ is too aggressive...
Dealing with transducer size

Our (first) solution is to factorize the LG transducer:

1. Create a $L \circ G_w$ for the word-level 3-gram, use a dummy symbol in $L$ for $<\text{unk}>$
Dealing with transducer size

Our (first) solution is to factorize the LG transducer:

1. Create a $L \circ G_w$ for the word-level 3-gram, use a dummy symbol in $L$ for $<\text{unk}>$
2. Merge all paths through $<\text{unk}>$ (more on that later) in $L \circ G_w$
Dealing with transducer size

Our (first) solution is to factorize the LG transducer:

1. Create a $L \circ G_w$ for the word-level 3-gram, use a dummy symbol in $L$ for $<\text{unk}>$
2. Merge all paths through $<\text{unk}>$ (more on that later) in $L \circ G_w$
3. Create a $L \circ G_c$ for the char-level 10-gram
Over-generative transducers

Dealing with transducer size

Our (first) solution is to factorize the LG transducer:

1. Create a $L \circ G_w$ for the word-level 3-gram, use a dummy symbol in $L$ for $\langle \text{unk} \rangle$
2. Merge all paths through $\langle \text{unk} \rangle$ (more on that later) in $L \circ G_w$
3. Create a $L \circ G_c$ for the char-level 10-gram
4. Plug $L \circ G_c$ where the dummy symbol appears
Dealing with transducer size

Our (first) solution is to factorize the LG transducer:

1. Create a $L \circ G_w$ for the word-level 3-gram, use a dummy symbol in $L$ for $<\text{unk}>$
2. Merge all paths through $<\text{unk}>$ (more on that later) in $L \circ G_w$
3. Create a $L \circ G_c$ for the char-level 10-gram
4. Plug $L \circ G_c$ where the dummy symbol appears

But merging all paths through $<\text{unk}>$ is too aggressive...
Path merging

$\beta$ 'B' 'C' <unk> $\epsilon$

Over-generative FST n-gram for OOV Word Recognition
Ronaldo Messina and Christopher Kermorvant
Path merging
One should not be too aggressive...

We experimented with the following merging options:
One should not be too aggressive...

We experimented with the following merging options:

- Merge all paths through \(<\text{unk}\)>
  - There is a single instance of the char-level n-gram
One should not be too aggressive...

We experimented with the following merging options:

- **Merge all** paths through `<unk>`
  - There is a single instance of the char-level n-gram
- **Merge paths in separate instances of `<unk>`**, one for each unigram, digram, and trigram contexts
  - Results in three instances: `<unk-uni>`, `<unk-di>`, and `<unk-tri>`
One should not be too aggressive...

We experimented with the following merging options:

- Merge **all** paths through `<unk>`
  - There is a single instance of the char-level n-gram
- Merge paths in separate instances of `<unk>`, one for each unigram, digram, and trigram contexts
  - Results in three instances: `<unk-uni>`, `<unk-di>`, and `<unk-tri>`
- Like above, but the instances are now dependent on the position: there are two for digrams and three for trigrams
  - Six instances are present, e.g. `<unk-uni>`, `<unk-di-left>`, `<unk-tri-middle>`, ...
We experimented with the following merging options:

- Merge **all** paths through $\langle \text{unk} \rangle$
  - There is a single instance of the char-level n-gram
- Merge paths in separate instances of $\langle \text{unk} \rangle$, one for each unigram, digram, and trigram contexts
  - Results in three instances: $\langle \text{unk-uni} \rangle$, $\langle \text{unk-di} \rangle$, and $\langle \text{unk-tri} \rangle$
- Like above, but the instances are now dependent on the position: there are two for digrams and three for trigrams
  - Six instances are present, e.g. $\langle \text{unk-uni} \rangle$, $\langle \text{unk-di-left} \rangle$, $\langle \text{unk-tri-middle} \rangle$, ...

Simple re-labeling of $\langle \text{unk} \rangle$ in the ARPA file and merging of corresponding paths in $L \circ G_w$
Path merging
Path merging

Over-generative transducers

Other in-paths through <unk-3-r>

Other out-paths through <unk-3-r>

Other in-paths through <unk-2-r>

Other out-paths through <unk-2-r>
Outline

1 Introduction

2 Hybrid grammars

3 Over-generative transducers

4 Experiments

5 Conclusion
Datasets

Two sets were used: IAM (English) [5] and Rimes (French) [6]. IAM consists in handwritten phrases from the LOB corpus [7]. Rimes is a database with handwritten letters.

Training data consist of:

<table>
<thead>
<tr>
<th></th>
<th>IAM</th>
<th>Rimes</th>
</tr>
</thead>
<tbody>
<tr>
<td>OM</td>
<td>81.6k</td>
<td>3.27M</td>
</tr>
<tr>
<td>LM</td>
<td>10.3k</td>
<td>106k</td>
</tr>
<tr>
<td>OM,LM</td>
<td>90.6k</td>
<td>6.7k</td>
</tr>
</tbody>
</table>

Test datasets comprise:

<table>
<thead>
<tr>
<th></th>
<th>IAM</th>
<th>Rimes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pages</td>
<td>336</td>
<td>72</td>
</tr>
<tr>
<td>Words</td>
<td>25753</td>
<td>4920</td>
</tr>
<tr>
<td>Vocab</td>
<td>5373</td>
<td>2121</td>
</tr>
</tbody>
</table>
Experiments

Effect of the number of \textless\text{unk}\textgreater\ instances

IAM, 20k words in the hybrid LM
$L \circ G$ is the size of the hybrid grammar composed with the lexicon
Baseline: 22.2\% without OOV and 17.3\% with the hybrid model [1]

<table>
<thead>
<tr>
<th># \textless\text{unk}\textgreater</th>
<th>%WER</th>
<th>%CER</th>
<th>$L \circ G$ (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>24.5</td>
<td>10.3</td>
<td>96</td>
</tr>
<tr>
<td>1</td>
<td>23.5</td>
<td>9.4</td>
<td>120</td>
</tr>
<tr>
<td>3</td>
<td>22.4</td>
<td>9.1</td>
<td>158</td>
</tr>
<tr>
<td>6</td>
<td>19.6</td>
<td>8.5</td>
<td>212</td>
</tr>
</tbody>
</table>

- Most of the gain comes when there is no “crossing” of n-gram boundaries (six instances)
- Similar gain (5\% absolute) as in the original article [1]
### Results: IAM test set

<table>
<thead>
<tr>
<th>Vocabulary</th>
<th>%OOV</th>
<th>%WER (base)</th>
<th>%WER (hybrid)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5k</td>
<td>14.7</td>
<td>36.6</td>
<td>24.6</td>
</tr>
<tr>
<td>10k</td>
<td>10.2</td>
<td>29.9</td>
<td>21.0</td>
</tr>
<tr>
<td>15k</td>
<td>8.0</td>
<td>26.7</td>
<td>20.0</td>
</tr>
<tr>
<td>20k</td>
<td>6.7</td>
<td>24.5</td>
<td>19.6</td>
</tr>
<tr>
<td>25k</td>
<td>5.7</td>
<td>23.4</td>
<td>19.4</td>
</tr>
<tr>
<td>full (106k)</td>
<td>2.1</td>
<td>19.1</td>
<td>n/a</td>
</tr>
</tbody>
</table>

- Marginal gain going from 15k to 25k words in the vocabulary
- Hybrid modeling is effective in coping with the OOV words
- The full vocabulary LM makes fewer errors in the in-vocabulary part, which is larger than the OOV part
### Results: Rimes test set

<table>
<thead>
<tr>
<th>Vocabulary</th>
<th>%OOV</th>
<th>%WER (base)</th>
<th>%WER (hybrid)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1k</td>
<td>12.8</td>
<td>24.5</td>
<td>14.4</td>
</tr>
<tr>
<td>1.5k</td>
<td>10.0</td>
<td>20.9</td>
<td>13.8</td>
</tr>
<tr>
<td>2k</td>
<td>8.3</td>
<td>18.6</td>
<td>13.7</td>
</tr>
<tr>
<td>2.5k</td>
<td>7.4</td>
<td>17.4</td>
<td>13.5</td>
</tr>
<tr>
<td>3k</td>
<td>6.9</td>
<td>16.8</td>
<td>13.3</td>
</tr>
<tr>
<td>full (6.7k)</td>
<td>5.1</td>
<td>14.6</td>
<td>n/a</td>
</tr>
</tbody>
</table>

- Performance stabilizes about 1.5k words in the vocabulary
- Again, the gap between no OOV modeling and hybrid grammar reduces with increase of vocabulary size
Outline

1 Introduction

2 Hybrid grammars

3 Over-generative transducers

4 Experiments

5 Conclusion
Conclusions and future work

Making the grammar over-generative does not hurt performance, as shown by the results. It is a practical way to create static transducers for recognition. Punctuation symbols could be added in the char-level n-gram. We are currently experimenting adding them to the backoff state, with promising results. The method could be extended to "class-specific" tokens.
Conclusions and future work

- Making the grammar over-generative does not hurt performance, as shown by the results.
Conclusions and future work

- Making the grammar over-generative does not hurt performance, as shown by the results.
- It is a practical way to create static transducers for recognition.
Conclusions and future work

- Making the grammar over-generative does not hurt performance, as shown by the results.
- It is a practical way to create static transducers for recognition.
- Punctuation symbols could be added in the char-level n-gram.
Conclusions and future work

- Making the grammar over-generative does not hurt performance, as shown by the results
- It is a practical way to create static transducers for recognition

- Punctuation symbols could be added in the char-level n-gram
  - We are currently experimenting adding them to the backoff state, with promising results
Conclusions and future work

- Making the grammar over-generative does not hurt performance, as shown by the results.
- It is a practical way to create static transducers for recognition.

- Punctuation symbols could be added in the char-level n-gram:
  - We are currently experimenting adding them to the backoff state, with promising results.
- The method could be extended to “class-specific” tokens.
Thank you!


