Low Resource ASR: The surprising effectiveness of High Resource Transliteration

Shreya Khare†,1, Ashish Mittal†,1, Anuj Diwan†,2, Sunita Sarawagi2, Preethi Jyothi2, Samarth Bharadwaj1

1 IBM Research 2 IIT Bombay

† Equal contribution

Slides by Anuj Diwan
Introduction
Motivations

Many advances in speech and NLP are powered by availability of data.

Only high-resource languages consistently benefit!

Reference:
Motivations

A vast majority of the 7000 languages of the world, including most *Indian* languages, fall in the low-resource category.

Techniques for low-resource languages need to be less *data-intensive* and often require interesting, radically new approaches.

Reference:
Automatic Speech Recognition

Convert an input speech signal to its corresponding transcript.

Reference:
https://medium.com/@ageitgey/machine-learning-is-fun-part-6-how-to-do-speech-recognition-with-deep-learning-28293c162f7a
Low-resource Speech Recognition

Developing **Automatic Speech Recognition (ASR)** techniques for **low-resource** languages.

In this paper, we explore a **Transliteration-based Transfer** approach for low-resource multilingual ASR.
Transliteration-based Transfer
Low Resource ASR: The surprising effectiveness of High Resource Transliteration

Shreya Khare†,1, Ashish Mittal†,1, Anuj Diwan†,2, Sunita Sarawagi2, Preethi Jyothi2, Samarth Bharadwaj1

1 IBM Research, 2 IIT Bombay

Accepted at Interspeech 2021
Introduction: Transfer Learning

- Using knowledge gained while solving one problem to solve a **different but related** problem.
- Use larger quantities of data from high-resource languages and **transfer** this knowledge to the low-resource language task.
Existing Approach

**Pretrain** using unlabelled+labelled speech from one (or more) ‘source’ high-resource languages

*Learn a general ‘good’ representation of speech*

**Finetune** all/part of model on labelled speech from ‘target’ low-resource language

*Given the pretrained model, learn parameters for the specific target language*
Existing Approach

What if source and target languages have disjoint grapheme spaces?

English: A B C D E F G H I J K L M ...

Hindi: क ख ग घ ङ च छ ज झ ञ ...
Existing Approach

What if source and target languages have disjoint grapheme spaces?

- Pretrain only the encoder of the encoder-decoder ASR architecture.
- Pretrain both the encoder and decoder. Before finetuning, replace output softmax layer with target language output softmax layer.

*Sharing across languages is latent and not easily controllable!*
Our Approach

Encourage increased sharing across grapheme spaces.

1. **Transliterate** transcriptions in high-resource speech data.
   - *From* high-resource language
   - *To* low-resource language

2. **Pretrain** model on high-resource language using original audio and *transliterated* transcriptions.

3. **Finetune** model using limited data from low-resource language.

*We also call our approach Eng2Tgt.*
Our Approach

Extended Pretraining ASR
Using speech in **E** with text *transliterated* from **E → T**

Pretrained ASR
Using speech in **high** resource language (**E**) → **Fine-Tuning ASR**
Using speech in **low** resource language (**T**)
Our Approach: Transliteration

<table>
<thead>
<tr>
<th>Language</th>
<th>Transliteration</th>
<th>IPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>en</td>
<td>ground without overbrimming</td>
<td>/ɡɹaʊnd wɪð',aʊt,əvər'brɪmɪŋ/</td>
</tr>
<tr>
<td>hi</td>
<td>प्रांउंड विदांत ओवरब्रिंग</td>
<td>/ɡɾaːˈnuːnd wiːd,ɑːt,əʊvərmɪŋ/</td>
</tr>
<tr>
<td>gu</td>
<td>usraund viñānt oवरब्रिंग</td>
<td>/grəˈnuːnd wɪnənt,əʊvərbrɪmɪŋ/</td>
</tr>
<tr>
<td>bn</td>
<td>প্রাঙ্গন বিদাংত ওবারব্রিং</td>
<td>/ɡɾaʊnd wɪd,ɑːt,əʊvərbrɪmɪŋ/</td>
</tr>
<tr>
<td>te</td>
<td>ప్రాంగన విదంతో ఓబర్బ్రింంంంంం</td>
<td>/ɡɾuːnd vɨtaːtəʊərˌbrɪmɪŋ/</td>
</tr>
<tr>
<td>ko</td>
<td>그라운드 위드호우트 오버브리밍</td>
<td>/kɯɾaʊndw ɯɪð,ʰoʊət,əʊvərˈbrɪmɪŋ/</td>
</tr>
<tr>
<td>am</td>
<td>ጎరুণಡ ይõతõ ఓవర్బ్రింంంంంం</td>
<td>/ɡirounid wɪthˌəʊvərˈbrɪmɪmɪnɪɡ/</td>
</tr>
</tbody>
</table>
Experiments

- **Source language**: English
- **6 languages**: Hindi, Telugu, Gujarati, Bengali, Korean, Amharic (more info: BTP Report)
- **2 ASR architectures**: Transformer [1] and wav2vec2.0 [2]
- **2 Training Durations**:
  - Full and 10-hr for Transformer expts
  - 10-hr and 1-hr for wav2vec2.0 expts

*Note: For Amharic and Korean, we only report wav2vec2.0 WERs; the WERs from the Transformer model were unstable, possibly due to poor seeds and require further investigation.*


Experiments: Baselines

1. **NoPre**: Train from scratch on low-resource data *without* pretraining.

2. **EngPre**: Pretrain using *untransliterated* text from English data, followed by finetuning on low-resource data.

3. **Tgt2Eng**: Based on [3].
   a. Pretrain using untransliterated text from English data
   b. Transliterate low-resource data transcriptions to English (Latin script) and finetune on this data.
   c. This model produces Latin script transcriptions. Thus, finally, transliterate back to low-resource language script.

Experimental Setup: Transformer

**Transformer Architecture for Speech Recognition**

We use the ESPNet toolkit to train hybrid CTC-attention Transformers

Major hyperparameters:
- 12 encoder layers with 2048 units
- 6 decoder layers with 2048 units
- 0.3 CTC, 0.7 Attention

More info in the paper

Reference:
# Results: Transformer

<table>
<thead>
<tr>
<th>Duration</th>
<th>Method</th>
<th>Hin</th>
<th>Tel</th>
<th>Guj</th>
<th>Ben</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NoPre</td>
<td>16.3</td>
<td>29.5</td>
<td>19.2</td>
<td>36.2</td>
</tr>
<tr>
<td></td>
<td>EngPre</td>
<td>15.6</td>
<td>26.3</td>
<td>17.6</td>
<td>27.2</td>
</tr>
<tr>
<td></td>
<td>Tgt2Eng</td>
<td>25.2</td>
<td>86.4</td>
<td>44.2</td>
<td>75.5</td>
</tr>
<tr>
<td></td>
<td>Eng2Tgt</td>
<td>15.6</td>
<td>25.9</td>
<td>17</td>
<td>26.2</td>
</tr>
<tr>
<td></td>
<td>(Ours)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full</td>
<td>NoPre</td>
<td>65.5</td>
<td>87.1</td>
<td>55.2</td>
<td>93.4</td>
</tr>
<tr>
<td></td>
<td>EngPre</td>
<td>29.4</td>
<td>51.9</td>
<td>33.4</td>
<td>57.1</td>
</tr>
<tr>
<td></td>
<td>Tgt2Eng</td>
<td>40.1</td>
<td>91.3</td>
<td>55.8</td>
<td>85.6</td>
</tr>
<tr>
<td></td>
<td>Eng2Tgt</td>
<td>28</td>
<td>48.5</td>
<td>34.4</td>
<td>56.4</td>
</tr>
<tr>
<td></td>
<td>(Ours)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Word Error Rate (WER)** for different transliteration schemes for the **Transformer** architecture
Results: Transformer

<table>
<thead>
<tr>
<th>Duration</th>
<th>Method</th>
<th>Hin</th>
<th>Tel</th>
<th>Guj</th>
<th>Ben</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NoPre</td>
<td>16.3</td>
<td>29.5</td>
<td>19.2</td>
<td>36.2</td>
</tr>
<tr>
<td></td>
<td>EngPre</td>
<td>15.6</td>
<td>26.3</td>
<td>17.6</td>
<td>27.2</td>
</tr>
<tr>
<td></td>
<td>Tgt2Eng</td>
<td>25.2</td>
<td>86.4</td>
<td>44.2</td>
<td>75.5</td>
</tr>
<tr>
<td></td>
<td>Eng2Tgt (Ours)</td>
<td>15.6</td>
<td>25.9</td>
<td>17</td>
<td>26.2</td>
</tr>
<tr>
<td>Full</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>NoPre</td>
<td>65.5</td>
<td>87.1</td>
<td>55.2</td>
<td>93.4</td>
</tr>
<tr>
<td></td>
<td>EngPre</td>
<td>29.4</td>
<td>51.9</td>
<td>33.4</td>
<td>57.1</td>
</tr>
<tr>
<td></td>
<td>Tgt2Eng</td>
<td>40.1</td>
<td>91.3</td>
<td>55.8</td>
<td>85.6</td>
</tr>
<tr>
<td></td>
<td>Eng2Tgt (Ours)</td>
<td>28</td>
<td>48.5</td>
<td>34.4</td>
<td>56.4</td>
</tr>
<tr>
<td>10 hour</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **NoPre** is worse than all three methods that use the English corpus.
- Our approach is better than baselines in most cases.
- Larger gains in low-resource 10-hour setting.
- Tgt2Eng is worse than all other methods. Likely due to lossy transliterations:

  चीफ -> chif -> चिफ
  निर्णयों -> nirnyon -> निर्णयोन
  आर्थिक -> aarthik -> आर्थिक
  पीपोदर -> pepodar -> पेरोदर
Experimental Setup: wav2vec2.0

**wav2vec2** Architecture for Speech Recognition

We use the [fairseq](https://github.com/pytorch/fairseq) toolkit

Model architecture and training schedules are according to the wav2vec2.0 paper

Before pretraining, all methods are **initialized** using the wav2vec2.0 model estimated using **unsupervised pretraining** on the complete Librispeech dataset

Thus, the **NoPre** baseline is replaced with the **SelfSup** baseline.

*Reference:

More info in the paper
Results: wav2vec2.0

<table>
<thead>
<tr>
<th>Method</th>
<th>Hin</th>
<th>Tel</th>
<th>Guj</th>
<th>Ben</th>
<th>Kor</th>
<th>Amh</th>
</tr>
</thead>
<tbody>
<tr>
<td>SelfSup</td>
<td>23.8</td>
<td>35.7</td>
<td>25.2</td>
<td>29.4</td>
<td>21.79(14.3)</td>
<td>26.54</td>
</tr>
<tr>
<td>EngPre</td>
<td>24.0</td>
<td>37.6</td>
<td>25.0</td>
<td>32.3</td>
<td>13.16(9.4)</td>
<td>26.78</td>
</tr>
<tr>
<td>Ours</td>
<td>23.6</td>
<td>34.5</td>
<td>23.2</td>
<td>28.2</td>
<td>13.16(9.6)</td>
<td>27.32</td>
</tr>
</tbody>
</table>

Word Error Rate (WER) for different transliteration schemes for the wav2vec2.0 architecture. For Korean, Character Error Rate (CER) also reported in parentheses.

Note: We dropped Tgt2Eng since it fared badly in the Transformer expts.
Results: wav2vec2.0

<table>
<thead>
<tr>
<th>Method</th>
<th>Hin</th>
<th>Tel</th>
<th>Guj</th>
<th>Ben</th>
<th>Kor</th>
<th>Amh</th>
</tr>
</thead>
<tbody>
<tr>
<td>SelfSup</td>
<td>23.8</td>
<td>35.7</td>
<td>25.2</td>
<td>29.4</td>
<td>21.79 (14.3)</td>
<td>26.54</td>
</tr>
<tr>
<td>EngPre</td>
<td>24.0</td>
<td>37.6</td>
<td>25.0</td>
<td>32.3</td>
<td>13.16 (9.4)</td>
<td>26.78</td>
</tr>
<tr>
<td>Ours</td>
<td>23.6</td>
<td>34.5</td>
<td>23.2</td>
<td>28.2</td>
<td>13.16 (9.6)</td>
<td>27.32</td>
</tr>
<tr>
<td>SelfSup</td>
<td>28.9</td>
<td>42.1</td>
<td>57.1</td>
<td>83.1</td>
<td>99.87 (83.3)</td>
<td>52.30</td>
</tr>
<tr>
<td>EngPre</td>
<td>29.9</td>
<td>48.1</td>
<td>62.1</td>
<td>92.3</td>
<td>66.36 (40.8)</td>
<td>53.75</td>
</tr>
<tr>
<td>Ours</td>
<td>28.5</td>
<td>41.5</td>
<td>55.2</td>
<td>88.9</td>
<td>62.08 (37.2)</td>
<td>53.29</td>
</tr>
</tbody>
</table>

- wav2vec SelfSup much better than Transformer NoPre
- Our method clearly outperforms EngPre in most settings on all languages
- Major exception is Amharic; we investigate this further

Our approach works even on a SOTA system like wav2vec that leverages powerful pretrained models!
Analysis and Discussions

*Under what conditions is our approach most effective?*

We propose that **two** properties should simultaneously hold:

- **High acoustic consistency** of the transliteration library
- **High phonological overlap** between the two languages
Analysis: Methodology

1. **Acoustic Consistency of Transliterations:**
   - Convert original English text to IPA (phones) using a g2p tool *(epitran)*
   - Convert transliterated text to IPA using native-language g2p tools
   - Compute PER between the two IPA sequences

2. **Phonological Similarity between Languages:**
   - Compute unigram distribution of phones in English and in low-resource language
   - Compute KL divergence between the two distributions
### Analysis: Results

<table>
<thead>
<tr>
<th>Language</th>
<th>am</th>
<th>bn</th>
<th>hi</th>
<th>te</th>
<th>gu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transliteration PER</td>
<td>89</td>
<td>90</td>
<td>76</td>
<td>82</td>
<td>72</td>
</tr>
<tr>
<td>KL dist phones</td>
<td>8.2</td>
<td>13.6</td>
<td>10.2</td>
<td>11.4</td>
<td>15.6</td>
</tr>
</tbody>
</table>

- For Hindi and Telugu, where KL dist is low and PER is low, we get consistent improvements in results.
- Amharic has a large PER, which may explain its poor performance. However, more investigation is needed, since its KL dist is very low.
Analysis: Effect of Related Languages

<table>
<thead>
<tr>
<th></th>
<th>Full</th>
<th>10 Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hin2Tgt</td>
<td>18.3</td>
<td>35.4</td>
</tr>
<tr>
<td>Eng2Tgt40</td>
<td>21.3</td>
<td>38.1</td>
</tr>
</tbody>
</table>

WERs for Gujarati when pretrained using two approaches:

**Hin2Tgt**: Pretrain on 40 hrs of Hindi transliterated to Gujarati

**Eng2Tgt40**: Pretrain on 40 hrs of English transliterated to Gujarati

*Pretraining on a related language helps!*
Analysis: EngPre vs Eng2Tgt

Our analysis indicates that:

In **EngPre**, pretraining lets the model learn sound clusters, and then the fine-tuning phase is used to learn character labels for each such sound, in addition to learning new sounds which are missing in the English speech data.

In **Eng2Tgt**, the fine-tuning phase focuses more on the second aspect (learning new sounds) as the pretraining phase already attaches character labels to the sound clusters.
Future Work

- Extending this approach to multilingual ASR.
- Extending this approach to languages with no transliteration systems.
Thank you!