# Stem2Morph: Low-Resource Morphological Inflection

Team 9


| Diwan Anuj Jitendra | Aditya Vavre | Yash Sharma |
|---|---|---|
| 170070005 | 170050089 | 17D070059 |

# Problem Statement

- **Morphological Inflection** is the task of generating a target (inflected form) word from a source word (base form), given a set of morphological attributes, e.g. number, tense, and person etc.
- **Input:**
  i. **A lemma** (morphological stem of a word) $X = x_1 \ldots x_N$ . Each $x_i$ is a character.
  ii. **Morphological tags** $T = t_1 \ldots t_M$. For eg., MASC (Masculine), N (Noun), GEN (Genitive), etc. Full list of tags is available in the Appendix of https://unimorph.github.io/doc/unimorph-schema.pdf.
- **Output:** The appropriate morphological inflected form $Y = y_1 \ldots y_K$ of the lemma. Each $y_i$ is a character.
- The goal is thus to model $P(Y|X, T)$.

# Problem Statement

- In this project, we tackle the problem of low-resource morphological inflection. Specifically, given data in a high resource language and extremely limited data in a low resource language, transfer learn morphological knowledge.
- This automatically learned morphology can then be used for other low-resource downstream NLP tasks, such as morphologically rich NMT.

# Reference Paper(s)

We primarily refer to the paper "Pushing the Limits of Low-Resource Morphological Inflection" - Antonios Anastasopoulos and Graham Neubig, 2019 (https://www.aclweb.org/anthology/D19-1091.pdf) for the architecture and training algorithm.

We refer to the paper "The SIGMORPHON 2019 Shared Task: Morphological Analysis in Context and Cross-Lingual Transfer for Inflection" - Arya D. McCarthy, Ekaterina Vylomova et. al. 2019 (https://www.aclweb.org/anthology/W19-4226v3.pdf) for the dataset description.

# Data - from SIGMORPHON 2019 dataset

- Each example is a (X, T, Y) - lemma characters, morphological tokens,and inflected characters, respectively.
- All data be found here - https://github.com/sigmorphon/2019/tree/master/task1
- All high resource languages have exactly 10,000 train examples. All low resource languages have 100 train and 100 test examples.
- There are 100 language pairs of (high, low) languages.

| | | |
|---|---|---|
| tighten | tightened | V;V.PTCP;PST |
| misbelieve | misbelieved | V;PST |
| potentiate | potentiates | V;3;SG;PRS |
| पाना | पा रहा था | V;2;SG;PROG;PST;MASC |
| मिलाना | मिलाते होंगे | V;3;PL;HAB;LGSPEC3;MASC |
| धड़कना | धड़किए | V;2;PL;IMP;FORM |
| क्षत्रिय | क्षत्रियाणि | ADJ;VOC;PL;NEUT |
| पूर | पूराभ्याम् | ADJ;INS;DU;MASC |
| स्वतन्त्र | स्वतन्त्राः | ADJ;NOM;PL;MASC |

# Technique Used

**Architecture[1] -** Attention based seq-to-seq model:

1. 32 dimensional character embeddings
2. 1 bi-lstm encoder (1 layer) for input characters
3. 1 self-attention encoder with unidirectional  for morphological tokens
4. Both followed by 2 attention layers to get context vectors in output space weights from decoder
5. Unique decoder model to produce probability distribution over output.

Let data be of the form (X, T, Y). A four stage training process is followed.
1. Pass (X, [NULL], X) and (Y, T, Y) from both languages to the model to get a good starting set of parameters. This is called COPY data.
2. Pass the previous data with (X, T, Y), from both languages, with less weightage to "COPY" data.
3. Pass only (Y, T, Y) and (X, T, Y) of the target language only
4. Pass (X, T, Y) of the target language only

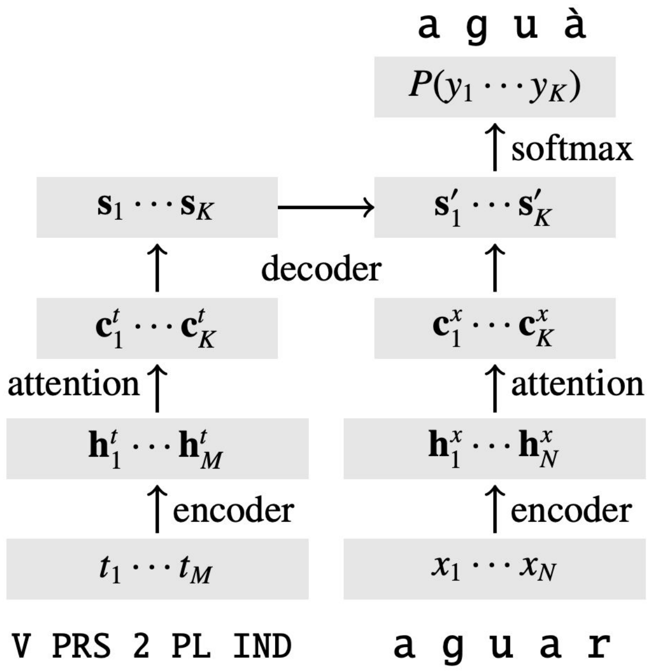


Figure 1: Visualization of our proposed two-step attention architecture. The decoder first attends over the tag sequence **T** and then uses the updated decoder state **s**′ to attend over the character sequence **X** in order to produce the inflected form **Y**. (Example from Asturian.)

# Our Work

1. The existing implementation from the paper is in Dynet. We implemented the paper entirely in **Pytorch**. This reimplementation can a) support GPU training and many more architectures supported by Pytorch b) reproduce and confirm the paper's results on a new library.
2. We implemented a demo for the models we trained on both code bases (Dynet and Pytorch) so the inflected words can be obtained in a more user-friendly way.
3. We discuss some new experiments highlighting the importance of transferring from high resource AND related languages.
4. We qualitatively analyze the errors made by the system for one of the languages.

# Metrics for evaluation

The performance of inflection systems is typically evaluated with:

- **Exact-match token-level accuracy**
- **Average character-level Levenshtein distance** between the predictions and their corresponding true forms.

> **What is Levenshtein distance?**
> The Levenshtein distance between two words is the minimum number of single-character edits (i.e. insertions, deletions or substitutions) required to change one word into the other.

We want a model with high exact-match accuracy and low Levenshtein distance. The two are usually inversely correlated.

# Description of Experiments

1. Reproducing results of the Dynet implementation using our Pytorch implementation. We do this for one language pair, Adyghe--Kabardian.
2. Results of the model on pairs of languages used for the demo.
3. Pretraining on 3 different languages with different genetic distance from Kashubian to evaluate correlation b/w accuracy and genetic distance
4. Exploring effect of using language A to transfer learn for language B vs just using language B. We run this in two settings:
    a. A = High resource Hindi, B = Low resource Bengali
    b. A = B = Low resource Bengali
5. Qualitative analysis of errors made by the Bengali and Telugu systems

# Results

| Language Pair | Metric | Dynet - Dev Set | Dynet - Test Set | PyTorch - Dev Set | PyTorch - Test Set |
|---|---|---|---|---|---|
| Adyghe--Kabardian | Accuracy | 0.94 | 0.94 | 0.88 | 0.95 |
| | Levenshtein | 0.06 | 0.06 | 0.12 | 0.05 |

| Language Pairs | Metric | Dev | Test |
|---|---|---|---|
| Hindi - Bengali | Acc | 0.41 | 0.42 |
| | Lev | 1.35 | 1.29 |
| Kannada - Telugu | Acc | 0.84 | 0.74 |
| | Lev | 0.26 | 0.76 |
| Urdu - Old-English | Acc | 0.174 | 0.165 |
| | Lev | 1.951 | 2.045 |

# Results

| Language Pairs | Metric | Dev | Test |
|---|---|---|---|
| Polish - Kashubian | Acc | 0.72 | **0.68** |
| | Lev | 0.34 | **0.42** |
| Slovac/Czech - Kashubian | Acc | 0.5 | 0.48 |
| | Lev | 0.72 | 0.86 |
| Basque - Kashubian | Acc | 0.46 | 0.34 |
| | Lev | 1.06 | 1.34 |

| Language Pairs | Metric | Test |
|---|---|---|
| Hindi - Bengali | Acc | **0.42** |
| | Lev | **1.29** |
| Bengali - Bengali | Acc | 0.23 |
| | Lev | 2.76 |

# Demo and Case Study

- We'll show a short demo of the model trained on Hindi and Bengali
- Discussion of the experiments:
  a. Expt 1: We can see that our Pytorch model was able to successfully reproduce the test set results of the Dynet model on Adyghe--Kabardian.
  b. Expt 2: We obtain reasonable results for Bengali and Telugu. Urdu - Old English has poor performance since the languages are pretty dissimilar.
  c. Expt 3: We see that closer the genetic distance of the high resource language, better the transfer learning performance, as expected.
  d. Expt 4: Not using a high-resource language to transfer learn worsens performance.
  e. Expt 5: ref: ভালবাসা bhālabāsā : love -- ভালবাসছিলি bhālabāsachili : loved it (V;2;PST;PROG;LGSPEC1)
     pred: ভালবাসা bhālabāsā : love -- ভালছাস bhālachāsa - no coherent meaning, frequent mistakes on this word
     ref: জাগানো jāgānō:wake up -- জাগিয়েছিলি jāgiẏēchili: woke up (V;2;PST;PRF;LGSPEC1)
     pred: জাগানো jāgānō: wake up -- জেগায়িলি jēgāẏili - no coherent meaning
     Using a language model over the output may improve the model. This can be explored in future work.

# Conclusion and Future Work

- We explore the task of low-resource morphological inflection. We implement the model in Pytorch, develop a demo, and run a few interesting experiments to explore the behaviour of the model.
- In future work, we would like to:
  - Apply an LM over the output to correct many of the errors we observed
  - Try other NN architectures, linguistically-inspired NN architectures
  - Apply the trained model for improving downstream tasks, such as NMT for morphologically rich languages