Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

FAKULTÄT
FÜR MATHEMATIK, INFORMATIK
UND NATURWISSENSCHAFTEN

FINDING THE WAY FROM Ä TO A: SUB-CHARACTER MORPHOLOGICAL INFLECTION FOR THE SIGMORPHON 2018 SHARED TASK

FYNN SCHRÖDER, MARCEL KAMLOT, GREGOR BILLING, ARNE KÖHN

MOTIVATION

Morphological inflection can easily be realised as string transduction on the character level. The lemma is modified to some extent, while the word root is retained.

There are several languages that **slightly modify the root**, for a variety of linguistic reasons (Kendris 2001; Wiese 2009):

Lemma	Inflection
Baumhaus	Baumh**ä**user
Kanarienvogel	Kanarienv**ö**geln
Milchkuh	Milchk**ü**hen

The example of a diacritical modification using the German *Umlaut* always keeps the graphical appearance of the underlying letter.

→ **The procedure of modifications can be generalised!**

PATCHES

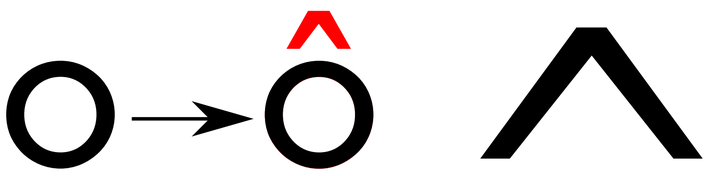
A *patch* denotes a shortcut operation between two graphically similar characters, for example `a` to `á`.

Formally, it acts as a **partial, symmetric function** $p(x)$ so that p can also be applied to other letters `e` to yield $p(\texttt{e}) = \texttt{é}$ — however in other cases $p(\texttt{b})$ is not defined for example.
Patching a patched letter $p(p(\texttt{o}))$ results in the original letter `o`.

â ê ô — applicable

ƀ ŵ — not applicable

To calculate meaningful patches, we **render the symbol pixels** into monospace fonts and compare them with element-wise `XOR` operations.
An implementation using **Unicode NFD decomposition** is also available.



STRING TRANSDUCER

The core transducer consists of edit actions drawing inspiration from Makarov, Ruzsics, and Clematide (2017). During traversal we keep an index pointer on the character currently being regarded.

- `EMIT` s: Append s to the output
- `COPY`: Append the pointer symbol to the output
- `PATCH` x: Apply the graphical patch matrix x to the pointer symbol, append the result to the output
- `MOVE`: Increment the pointer
- `EOW`: Stop traversing the string, return final result

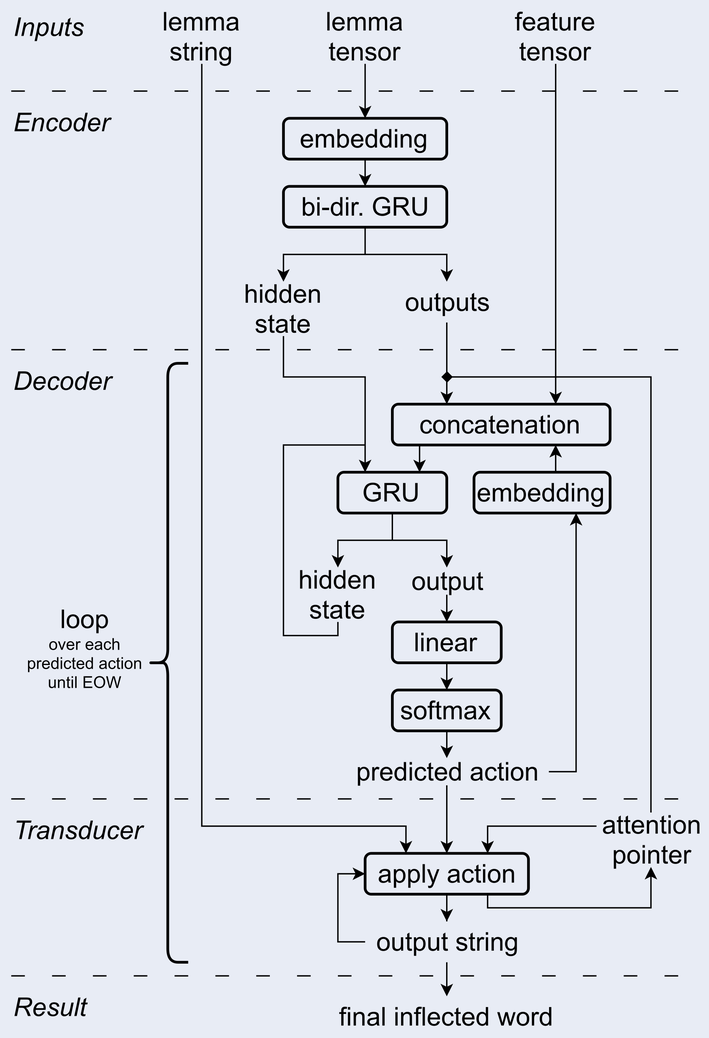
Necessary transducer actions are derived through a **static oracle** based on aligned input.

Alignment is determined through a customised *Levenshtein* metric that additionally assumes cost 0 if there is a patch that transforms one symbol into the other.

NEURAL NETWORK

Our network is based on a string transducer and employs an **encoder-decoder architecture** with monotonic hard attention (Aharoni and Goldberg 2016) that has been found beneficial for morphological inflection (Aharoni, Goldberg, and Belinkov 2016).

By using this focused form of attention, our system is capable of meaningfully performing `COPY` and `PATCH` actions.



TRAINING

Training updates are performed via backpropagation with the **Adam** optimizer (Kingma and Ba 2014).

The **network target is a sequence of actions** to transform the lemma into a corresponding target form. To improve predictions, we employ **beam search** decoding over a custom loss function based on Andor et al. (2016)

$$L = -\frac{\sum_i^s l_i}{\ln(1+s)}$$

Training updates are performed once the correct path falls out of the beam. Unfortunately, upon trying to convert the training to use global normalization (Andor et al. 2016) our model failed to reliably converge.

NETWORK HICCUPS

Due to the static targets, our network sometimes **doesn't learn to emit** `EOW` tokens. We counteract this scenario by ignoring repeat `EMIT` actions when the lemma is fully read.

The monotonic attention prevents our network to focus on previously consumed parts of the word, so split inflections cause some trouble.

DATA ENHANCER

Some tracks feature as little as 61 training samples, so we considered additional ways of creating input data **without external resources**.

Training data is grouped into sets of common features and aligned. Only matching characters among the alignments are kept, and the gaps are filled by **language models** for different lengths of **character n-grams**.

These n-grams in turn have **gaps** to avoid training data repetition. They are filled by random letters drawn from an actual training set distribution.

1. align	`skapad` `#fixad`	`skappade` `##fixade`
2. retain matching characters	`____ad` `#___ad`	`_____ade` `##___ade`
3. fill n-grams	`iommad` `#he?ad`	`iomm?ade` `##he?ade`
4. fill gaps	`iommad` `#herad`	`iommpade` `##herade`

n-gram	letter	probability
?ad	r	0.44
	p	0.18
	t	0.10
?ade	r	0.53
	p	0.18
	n	0.09

EVALUATION AND INSIGHTS

Our results are in **solid mid-range** compared to other Shared Task submissions for 2018.

Some languages excel, some underperform miserably. In either case, **beam search yields an improvement**.

The "cut-off" errors can be corrected via the use of a **dynamic oracle** that allows for different target sequences.

N-Gram data enhancement has a **paradoxical drawback**:
Low volume = high demand but little data to learn from
High volume = no demand but plenty of data

Patches **can help** but are certainly **not the silver bullet**. Our highest improvement is 7 percent points, but it can noticeably deteriorate results in other cases.

→ Rely on **linguistic intuition** and **context**!

Shared Tasks are awesome, go contribute to one!

DEPARTMENT OF INFORMATICS
NATURAL LANGUAGE SYSTEMS GROUP

source code available at
https://gitlab.com/nats/sigmorphon18