Few-Shot and Zero-Shot Learning for Historical Text Normalization

Bollmann, Marcel; Korchagina, Natalia; Søgaard, Anders

Published in:
Proceedings of the 2nd Workshop on Deep Learning Approaches for Low-Resource NLP (DeepLo 2019)

DOI:
10.18653/v1/D19-6112

Publication date:
2019

Document version
Publisher's PDF, also known as Version of record

Document license:
CC BY

Citation for published version (APA):
Few-Shot and Zero-Shot Learning for Historical Text Normalization

Marcel Bollmann♦ and Natalia Korchagina♢ and Anders Søgaard♦

♦Department of Computer Science, University of Copenhagen
♢Institute of Computational Linguistics, University of Zurich

marcel@di.ku.dk, korchagina@ifi.uzh.ch, soegaard@di.ku.dk

Abstract

Historical text normalization often relies on small training datasets. Recent work has shown that multi-task learning can lead to significant improvements by exploiting synergies with related datasets, but there has been no systematic study of different multi-task learning architectures. This paper evaluates 63 multi-task learning configurations for sequence-to-sequence-based historical text normalization across ten datasets from eight languages, using autoencoding, grapheme-to-phoneme mapping, and lemmatization as auxiliary tasks. We observe consistent, significant improvements across languages when training data for the target task is limited, but minimal or no improvements when training data is abundant. We also show that zero-shot learning outperforms the simple, but relatively strong, identity baseline.

1 Introduction

Historical text normalization is the task of mapping variant spellings in historical documents—e.g., digitized medieval manuscripts—to a common form, typically their modern equivalent. The aim is to make these documents amenable to search by today’s scholars, processable by NLP tools, and accessible to lay people. Many historical documents were written in the absence of standard spelling conventions, and annotated datasets are rare and small, making automatic normalization a challenging task (cf. Piotrowski, 2012; Bollmann, 2018).

In this paper, we experiment with datasets in eight different languages: English, German, Hungarian, Icelandic, Portuguese, Slovene, Spanish, and Swedish. We use a standard neural sequence-to-sequence model, which has been shown to be competitive for this task (e.g., Korchagina, 2017; Bollmann, 2018; Tang et al., 2018). Our main focus is on analyzing the usefulness of multi-task learning strategies (a) to leverage whatever supervision is available for the language in question (few-shot learning), or (b) to do away with the need for supervision in the target language altogether (zero-shot learning).

Bollmann et al. (2017) previously showed that multi-task learning with grapheme-to-phoneme conversion as an auxiliary task improves a sequence-to-sequence model for historical text normalization of German texts; Bollmann et al. (2018) showed that multi-task learning is particularly helpful in low-resource scenarios. We consider three auxiliary tasks in our experiments—grapheme-to-phoneme mapping, autoencoding, and lemmatization—and focus on extremely low-resource settings.

Our paper makes several contributions:

(a) We evaluate 63 multi-task learning configurations across ten datasets in eight languages, and with three different auxiliary tasks.

(b) We show that in few-shot learning scenarios (ca. 1,000 tokens), multi-task learning leads to robust, significant gains over a state-of-the-art, single-task baseline.

(c) We are, to the best of our knowledge, the first to consider zero-shot historical text normalization, and we show significant improvements over the simple, but relatively strong, identity baseline.

While our focus is on the specific task of historical text normalization, we believe that our results can be of interest to anyone looking to apply multi-task learning in low-resource scenarios.

We note that 1,000 tokens is more instances than is typically considered in few-shot learning; e.g., Kimura et al. (2018) use up to 200 instances. We argue that for structured prediction it is reasonable to assume more data, yet we also consider scenarios down to as little as 100 instances.
Table 1 gives an overview of the languages and the size of the development set, which we use for evaluation.

2 Model architecture

We use a standard attentional encoder–decoder architecture (Bahdanau et al., 2014) with words as input sequences and characters as input symbols. Following the majority of previous work on this topic (cf. Sec. 5), we limit ourselves to word-by-word normalization, ignoring problems of contextual ambiguity. Our model consists of the following parts (which we will also refer to using the bolded letters):

- **Source embedding layer**: transforms input characters into dense vectors.
- **Encoder**: a single bidirectional LSTM that encodes the embedded input sequence.
- **Attention layer**: calculates attention from the encoded inputs and the current decoder state using a multi-layer perceptron (as in Bahdanau et al., 2014).
- **Target embedding layer**: transforms output characters into dense vectors.
- **Decoder**: a single LSTM that decodes the encoded sequence one character at a time, using the attention vector and the embedded previous output characters as input.
- **Prediction layer**: a final feed-forward layer that linearly transforms the decoder output and performs a softmax to predict a distribution over all possible output characters.

3 Hyperparameters

We tuned our hyperparameters on the English development section. We use randomly initialized embeddings of dimensionality 60, hidden layers of dimensionality 300, a dropout of 0.2 and a batch size of 30. We train the model for an unspecified number of epochs, instead relying on early stopping on a held-out validation set. Since we experiment with varying amounts of training data, we choose to derive this held-out data from the given training set, using only 90% of the tokens as actual training data and the remaining 10% to determine early stopping.

3 Multi-task learning

Multi-task learning (MTL) is a technique to improve generalization by training a model jointly on a set of related tasks. We follow the common approach of hard parameter sharing suggested by Caruana (1993), in which certain parts of a model architecture are shared across all tasks, while others are kept distinct for each one. Such approaches have been applied successfully to a variety of problems, e.g., machine translation (Dong et al., 2015), sequence labelling (Yang et al., 2016; Peng and Dredze, 2017), or discourse parsing (Braud et al., 2016).

Auxiliary tasks

We experiment with the following auxiliary tasks:

- **Autoencoding**. We use data extracted from Wikipedia and train our model to recreate the input words. In the normalization task, large parts of the input words often stay the same, so autoencoding might help to reinforce this behavior in the model.
- **Grapheme-to-phoneme mapping (g2p)**. This task uses the data by Deri and Knight.

4 Whenever possible, we used the dumps provided by the Polyglot project: https://sites.google.com/site/rmyeid/projects/polyglot. Since an Icelandic text dump was not available from Polyglot, we generated one ourselves using the Cirrus Extractor: https://github.com/attardi/wikiextractor. All dumps were cleaned from punctuation marks.
(2016) to map words (i.e., sequences of graphemes) to sequences of phonemes. Bollmann et al. (2017) previously showed that this task can improve historical normalization, possibly because changes in spelling are often motivated by phonological processes, an assumption also made by other normalization systems (Porta et al., 2013; Etxeberria et al., 2016).

- **Lemmatization.** We use the UniMorph dataset (Kirov et al., 2018) to learn mappings from inflected word forms to their lemmas. This task is similar to normalization in that it maps a set of different word forms to a single target form, which typically bears a high resemblance to the input words.

Since we train separate models for each historical dataset, we always use auxiliary data from the same language as the dataset.

**Training details** When training an MTL model, we make sure that each training update is based on a balanced combination of main and auxiliary task inputs; i.e., for each batch of 30 tokens of the historical normalization task, the model will see 10 tokens from each auxiliary task. Epochs are still counted based on the normalization task only. This way, we try to make up for the imbalanced quantity of different auxiliary datasets.

### 3.1 Experiment 1: What to share?

In previous work on multi-task learning, there is no clear consensus on which parts of a model to share and which to keep separate. Bollmann et al. (2017) share all parts of the model except for the final prediction layer, while other multi-task sequence-to-sequence models keep task-specific encoders and decoders (cf. also Sec. 5). In principle, though, the decision to share parameters between tasks can be made for each of the encoder–decoder components individually, allowing for many more possible MTL configurations.

**Setup** We explore the effect of different sharing configurations. The architecture described in Sec. 2 leaves us with $2^6 = 64$ possible model configurations. When all parameters are shared, this is identical to training a single model to perform all tasks at once; when none are shared, this is identical to a single-task model trained on historical normalization only. We identify an MTL configuration using letters (cf. the bold letters from Sec. 2) to indicate which parts of the model are shared; e.g., an “SE” model would share the source embeddings and the encoder, an “SEATD” model would share everything except the final prediction layer, and so on.

In Experiment 1, we only use the first 1,000 tokens of the English historical dataset for training. We combine this with all three auxiliary tasks (using their full datasets) and train one MTL model for each of the 64 different sharing configurations.

**Results** Figure 1 shows an excerpt of the results, evaluated on the dev set of the English dataset. The best MTL model achieves a normalization accuracy of 75.9%, while the worst model gets 58.6%. In total, 49 configurations outperform the single-task model, showing the general effectiveness of the MTL approach. Sharing more is generally better; nine out of the top ten configurations share at least four components. Figure 2 visualizes the accuracy distribution by the number of shared components in the MTL model, supporting this conclusion.

### 3.2 Experiment 2: Which auxiliary tasks?

In the previous experiment, we trained the models using all three auxiliary tasks at the same time. However, not all of these tasks might be equally helpful for learning the normalization task. While Bollmann et al. (2017) show the effectiveness of

---

5https://unimorph.github.io/
the grapheme-to-phoneme task, they only evaluate on German, and autoencoding and lemmatization have so far not been evaluated at all for improving historical text normalization.

Setup We want to investigate the improvements from each auxiliary task in isolation compared to (a) the single-task baseline and (b) the previous approach of training with all three auxiliary tasks simultaneously. For this, we select the best MTL configuration from Sec. 3.1, which is to share everything except the target embeddings (“SEADP”), and train one single-task model and four MTL models per dataset: one for each of the three auxiliary tasks, and one that uses all three tasks at the same time.

As before, we only use the first 1,000 tokens of each historical dataset. This also makes the results more comparable across datasets, as the size of the training set for the main task can affect the usefulness of multi-task learning.

Results Figure 3 shows the error reduction of the MTL models compared to the single-task setup. For most datasets, MTL improves the results; the main exception is Hungarian, where all three auxiliary tasks lead to a loss in accuracy. The results show that not all auxiliary tasks are equally beneficial. Autoencoding provides the largest error reduction in most cases, while lemmatization is often slightly worse, but provides the best result for German (Anselm) and Swedish. The grapheme-to-phoneme task, on the other hand, performs worst on average, yielding much less benefits on German (Ridges) and English, and even increases the error on Swedish.

Table 2a shows the accuracy scores for all datasets and models. The full MTL model—training jointly on all tasks—only achieves the best performance on four of the datasets. Since the dev sets used for this evaluation vary strongly in size, we also calculate the micro-average of the accuracy scores, i.e., the accuracy obtained over the concatenation of all datasets. Here, we can see that using only autoencoding as an auxiliary task actually produces the highest average accuracy.

3.3 Experiment 3: How much training data? All previous experiments have used 1,000 tokens from each historical dataset for training. Bollmann et al. (2018) show that the benefits of multi-task learning depend on training data quantity, so it is unclear whether the findings generalize to smaller or larger datasets.

Setup We analyze the benefit of MTL depending on the amount of training data that is used for the main task. We do this by training MTL models (using all three auxiliary tasks, as in Sec. 3.1) with varying amounts of historical training data, ranging from 100 tokens to 50,000 tokens. Different...
sharing configurations might conceivably give different benefits based on the training set size. We therefore evaluate each of the top three MTL configurations from Sec. 3.1, as well as the single-task model, across different data sizes.

**Results**  Figure 4 shows learning curves for all of our historical datasets. The quantity of improvements from MTL differs between datasets, but there is a clear tendency for MTL to become less beneficial as the size of the normalization training set increases. In some cases, using MTL with larger training set sizes even results in lower accuracy compared to training a single-task model to do normalization only. This suggests that multi-task learning—at least with the auxiliary tasks we have chosen here—is mostly useful when the training data for the main task is sparse.

Since the accuracy scores of the different models are often within close range of each other, Figure 5 visualizes the three MTL configurations in terms of error reduction compared to the single-task model, averaged over all ten datasets. This again highlights the decreasing gains from MTL with increasing amounts of training data.

### 3.4 Comparison to previous work

**Bollmann (2019)** compares normalization models when trained with different amounts of data, including a setting with 1,000 tokens for training, allowing us to directly compare our results with those reported there. They show that multi-task learning can yield significant improvements compared to single-task models, especially when the amount of training data is limited.

Since the accuracy scores of the different models are often within close range of each other, Figure 5 visualizes the three MTL configurations in terms of error reduction compared to the single-task model, averaged over all ten datasets. This again highlights the decreasing gains from MTL with increasing amounts of training data.

### 3.5 Experiment 4: Zero-shot learning

Most previous work on historical text normalization has focused on a supervised scenario where some labeled data is available for the target domain, i.e., the particular historical language you are interested in. Since spelling variation is highly idiosyncratic in the absence of normative spelling guidelines, models are not expected to generalize beyond specific language stages, or sometimes even manuscript collections. This means that many historical text normalization projects require resources to annotate new data. This paper is the first to experiment with a zero-shot learning scenario that leverages existing data from other languages, but assumes no labeled data for the target language.

**Setup**  For the zero-shot experiments, we use the same model as for the single-task baseline; in other words, all layers are shared between all tasks and languages. Instead, to allow the model to discern between languages and tasks, we prepend two extra symbols to all model inputs: a language identifier and a task identifier.

### Table 2: Normalization accuracy on dev sets after training on 1,000 tokens. Best results highlighted in bold.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Single</th>
<th>Multi-task</th>
<th>Dataset</th>
<th>Best in (a) from Bollmann (2019)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Autoenc</td>
<td>Lemma</td>
<td>g2p</td>
<td>ALL 3</td>
</tr>
<tr>
<td>DE A</td>
<td>54.84</td>
<td>56.55</td>
<td>55.99</td>
<td>56.52</td>
</tr>
<tr>
<td>DE R</td>
<td>56.72</td>
<td>60.55</td>
<td>60.25</td>
<td>64.49</td>
</tr>
<tr>
<td>EN</td>
<td>66.95</td>
<td>76.94</td>
<td>68.72</td>
<td>72.01</td>
</tr>
<tr>
<td>ES</td>
<td>74.68</td>
<td>77.87</td>
<td>76.97</td>
<td>78.45</td>
</tr>
<tr>
<td>HU</td>
<td>42.44</td>
<td>61.27</td>
<td>58.60</td>
<td>52.74</td>
</tr>
<tr>
<td>IS</td>
<td>63.40</td>
<td>67.31</td>
<td>67.02</td>
<td>66.31</td>
</tr>
<tr>
<td>PT</td>
<td>72.23</td>
<td>76.28</td>
<td>73.89</td>
<td>74.27</td>
</tr>
<tr>
<td>SL B</td>
<td>74.06</td>
<td>75.59</td>
<td>78.97</td>
<td>72.30</td>
</tr>
<tr>
<td>SL G</td>
<td>86.34</td>
<td>87.86</td>
<td>86.15</td>
<td>87.40</td>
</tr>
<tr>
<td>SV</td>
<td>69.98</td>
<td>70.29</td>
<td>72.34</td>
<td>65.97</td>
</tr>
<tr>
<td>Micro-Avg</td>
<td>64.46</td>
<td>67.46</td>
<td>66.47</td>
<td>65.79</td>
</tr>
</tbody>
</table>

(a) Single-task vs. multi-task models

(b) Comparison to previous work

---

7Bollmann (2019) only shows graphical plots for these results, but the exact figures were released at: [https://github.com/coastalcph/histnorm/blob/master/appendix_tab6.pdf](https://github.com/coastalcph/histnorm/blob/master/appendix_tab6.pdf)
Figure 4: Learning curves for all datasets, showing the normalization accuracy of a single-task and three multi-task learning models in relation to the training set size; note that the x-axis is log-scaled.
language identifier and a task identifier. For each language, we then train a single model on all tasks—normalization, lemmatization, autoencoding, and grapheme-to-phoneme transduction—and all languages, except for the normalization task of the target language. This way, the model can observe data from the normalization task (albeit in other languages) and from the target language (albeit from auxiliary tasks only), but does not see any normalization data from the target language. In those cases where there are two datasets from the same language, we leave out both of them from the training step. The model is similar to previous work on zero-shot neural machine translation (Johnson et al., 2016).

As before, we include only 1,000 tokens from each historical dataset for training. In each training update, we use an equal number of samples from each dataset/task combination, and define an epoch to consist of 1,000 samples from each of these combinations. Since we do not want to feed the model any normalization data from the target language during training, we cannot use early stopping, but instead train for a fixed number of 10 epochs.

Results Table 3 shows the accuracy of zero-shot normalization compared to the naive identity baseline, i.e., the accuracy obtained by simply leaving the input word forms unchanged. The zero-shot approach improves over this baseline for half of the datasets, sometimes by up to 12 percentage points (DE_R). Micro-averaging the results shows an overall advantage for zero-shot learning.

```
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Identity</th>
<th>Zero-shot</th>
</tr>
</thead>
<tbody>
<tr>
<td>DE_A</td>
<td>30.16</td>
<td>40.94</td>
</tr>
<tr>
<td>DE_R</td>
<td>43.57</td>
<td>55.92</td>
</tr>
<tr>
<td>EN</td>
<td>75.47</td>
<td>56.31</td>
</tr>
<tr>
<td>ES</td>
<td>72.29</td>
<td>42.95</td>
</tr>
<tr>
<td>HU</td>
<td>17.81</td>
<td>20.58</td>
</tr>
<tr>
<td>IS</td>
<td>47.77</td>
<td>42.95</td>
</tr>
<tr>
<td>PT</td>
<td>65.18</td>
<td>67.64</td>
</tr>
<tr>
<td>SL_B</td>
<td>39.84</td>
<td>42.95</td>
</tr>
<tr>
<td>SL_G</td>
<td>85.88</td>
<td>84.99</td>
</tr>
<tr>
<td>SV</td>
<td>59.24</td>
<td>50.65</td>
</tr>
<tr>
<td>Micro-Avg</td>
<td>50.17</td>
<td>52.96</td>
</tr>
</tbody>
</table>
```

Table 3: Normalization accuracy on dev sets for zero-shot experiments. Best results highlighted in bold.

4 Analysis

The experiment in Sec. 3.2 has shown that not all auxiliary tasks are equally useful; furthermore, autoencoding is, on average, the most useful auxiliary task of the three, closely followed by lemmatization. This gives rise to the hypothesis that MTL mostly helps the model learn the identity mappings between characters.

To analyze this, we feed the historical data into the auxiliary models; i.e., we treat them as if they were a historical text normalization model. We then correlate their normalization accuracy with the error reduction over the baseline of the MTL model using this auxiliary task. Figure 6a shows a strong correlation for the autoencoding task, suggesting that the synergy between autoencoding and historical text normalization is higher when the two tasks are very related. Figure 6b shows the same correlation for lemmatization.

We can also compare the error reduction from MTL to the identity baseline (cf. Tab. 3). Figure 7 shows the correlation of these scores for the full MTL model trained with all three auxiliary tasks. The strong correlation suggests that the regularization effect introduced by MTL is particularly helpful with tasks where there is a strong similarity between input and output; or, in other words, that multi-task learning prevents the model from overgeneralizing based on the training data.

The previous correlation scores only consider the performance of models trained on 1,000 tokens of historical data. Sec. 3.3 showed that the benefit of MTL diminishes when the size of the historical training sets gets larger. Figure 8 presents learning

---

8The correlation is similar when using longest common subsequence or Levenshtein distance instead of accuracy.
Figure 6: Correlations (with 95% confidence intervals) between the performance of an auxiliary task model applied to normalization data and the error reduction when using this task in a multi-task learning setup.

Figure 7: Correlation (with 95% confidence interval) between the identity baseline and the error reduction of the full MTL model with all three auxiliary tasks.

curves that have been micro-averaged over all ten datasets, but evaluated on different subsets of the data: (a) tokens that have been seen during training (“knowns”) or not (“unknowns”); and (b) tokens that stay identical in the reference normalization or not. On average, the performance of the MTL models is comparable to that of the single-task model for known tokens and non-identity normalizations. In other words, most of the gain from MTL comes from helping the model learn the identity mappings, which becomes less relevant the more historical training data is available.

5 Related work

On previous approaches to historical text normalization, Bollmann (2019, Sec. 2) gives an extensive overview. Common approaches include rule-based algorithms—with either manually crafted or automatically learned rules—or distance metrics to compare historical spellings to modern lexicon forms (Baron and Rayson, 2008; Bollmann, 2012; Pettersson et al., 2013a). Finite-state transducers are sometimes used to model this, but also to explicitly encode phonological transformations which often underlie the spelling variation (Porta et al., 2013; Etxeberria et al., 2016).

Character-based statistical machine translation (CSMT) has been successfully applied to normalization on many languages (Pettersson et al., 2013b; Scherrer and Erjavec, 2016; Domingo and Casacuberta, 2018); neural encoder–decoder models with character-level input can be seen as the neural equivalent to the statistical MT approach (Bollmann et al., 2017; Tang et al., 2018) and have been shown to be competitive with it (Robertson and Goldwater, 2018; Hämäläinen et al., 2018), although Bollmann (2019) suggests that they are still inferior to CSMT in low-resource scenarios.

All these methods rely on individual word forms as their input; there is almost no work on incorporating sentence-level context for this task (but cf. Jurish, 2010).

MTL architectures In Sec. 3.1, we explored what to share between tasks in our multi-task architecture. A common approach is to share only the first layers (e.g., Yang et al., 2016; Peng and Dredze, 2017). Multi-task encoder–decoder models will often keep the whole encoder and decoder task- or language-specific (Dong et al., 2015; Luong et al., 2015). Firat et al. (2016) explore the effect of sharing the attentional component across
all languages, while Anastasopoulos and Chiang (2018) compare both parallel and cascading model configurations.

A different MTL approach is to share all parts of a model, but prepend a task-specific symbol to the input string to enable it to learn task-specific features (cf. Sec. 3.5). Milde et al. (2017) use this approach for grapheme-to-phoneme conversion; Kann et al. (2017) apply it to morphological paradigm completion.

**Auxiliary tasks for MTL** For which auxiliary task(s) to use (Sec 3.2), few systematic studies exist. Most approaches use tasks that are deemed to be related to the main task—e.g., combining machine translation with syntactic parsing (Kiperwasser and Ballesteros, 2018)—and justify their choice by the effectiveness of the resulting model. Bingel and Søgaard (2017) analyze beneficial task relations for MTL in more detail, but only consider sequence labelling tasks. For zero-shot learning (Sec. 3.5), we use an architecture very similar to Johnson et al. (2016), also used for grapheme-to-phoneme mapping in Peters et al. (2017).

### 6 Conclusion

We performed an extensive evaluation of a neural encoder–decoder model on historical text normalization, using little or even no training data for the target language, and using multi-task learning (MTL) strategies to improve accuracy. We found that sharing more components between main and auxiliary tasks is usually better, and autoencoding generally provides the most benefit for our task. Analysis showed that this is mainly because MTL helps the model learn that most characters should stay the same, and that its beneficial effect vanishes as the size of the training set increases. While our models did not beat the non-neural models of Bollmann (2019), we believe our work still provides interesting insights into the impact of MTL for low-resource scenarios.

### Acknowledgments

We would like to thank the anonymous reviewers of this as well as previous iterations of this paper for several helpful comments.

This research has received funding from the European Research Council under the ERC Starting Grant LOWLANDS No. 313695. Marcel Bollmann was partly funded from the European Union’s Horizon 2020 research and innovation programme under the Marie Skodowska-Curie grant agreement No. 845995.
References


