Neural String Edit Distance

Jindřich Libovický[1] and **Alexander Fraser**[2]
[1]Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic
[2]Center for Information and Language Processing, LMU Munich, Germany
libovicky@ufal.mff.cuni.cz fraser@cis.lmu.de

Abstract

We propose the *neural string edit distance* model for string-pair matching and string transduction based on learnable string edit distance. We modify the original expectation-maximization learned edit distance algorithm into a differentiable loss function, allowing us to integrate it into a neural network providing a contextual representation of the input. We evaluate on cognate detection, transliteration, and grapheme-to-phoneme conversion, and show that we can trade off between performance and interpretability in a single framework. Using contextual representations, which are difficult to interpret, we match the performance of state-of-the-art string-pair matching models. Using static embeddings and a slightly different loss function, we force interpretability, at the expense of an accuracy drop.

1 Introduction

State-of-the-art models for string-pair classification and string transduction employ powerful neural architectures that lack interpretability. For example, BERT (Devlin et al., 2019) compares all input symbols with each other via 96 attention heads, whose functions are difficult to interpret. Moreover, attention itself can be hard to interpret (Jain and Wallace, 2019; Wiegreffe and Pinter, 2019).

In many tasks, such as in transliteration, a relation between two strings can be interpreted more simply as edit operations (Levenshtein, 1966). The edit operations define the alignment between the strings and provide an interpretation of how one string is transcribed into another. Learnable edit distance (Ristad and Yianilos, 1998) allows learning the weights of edit operations from data using the expectation-maximization (EM) algorithm. Unlike post-hoc analysis of black-box models, which depends on human qualitative judgment (Adadi and Berrada, 2018; Hoover et al., 2020; Lipton, 2018), the restricted set of edit operations allows direct interpretation. Unlike hard attention (Mnih et al., 2014; Indurthi et al., 2019) which also provides a discrete alignment between input and output, edit distance explicitly says how the input symbols are processed. Also, unlike models like Levenshtein Transformer (Gu et al., 2019), which does not explicitly align source and target uses edit operations to model intermediate generation steps only within the target string, learnable edit distance considers both source and target symbols to be a subject of the edit operations.

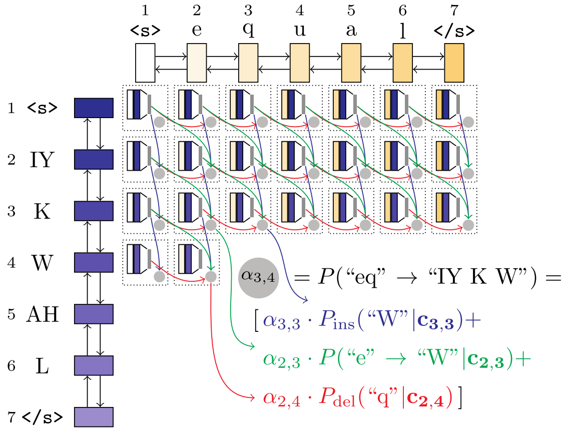
1
<s>
2
e
3
q
4
u
5
a
6
l
7
</s>
1 <s>
2 IY
3 K
4 W
5 AH
6 L
7 </s>
$\alpha_{3,4} = P(\text{“eq”} \rightarrow \text{“IY K W”}) =$
$[\alpha_{3,3} \cdot P_{\text{ins}}(\text{“W”}|\mathbf{c_{3,3}})+$
$\alpha_{2,3} \cdot P(\text{“e”} \rightarrow \text{“W”}|\mathbf{c_{2,3}})+$
$\alpha_{2,4} \cdot P_{\text{del}}(\text{“q”}|\mathbf{c_{2,4}})]$

Figure 1: An example of applying the dynamic programming algorithm used to compute the edit probability score. It gradually fills the table of probabilities that prefixes of the word “equal” transcribe into prefixes of phoneme sequence “IY K W AH L”. The probability (gray circles) depends on the probabilities of the prefixes and probabilities of plausible edit operations: insert (blue arrows), substitute (green arrows) and delete (red arrows).

We reformulate the EM training used to train learnable edit distance as a differentiable loss function that can be used in a neural network. We propose two variants of models based on *neural string edit distance*: a bidirectional model for string-pair matching and a conditional model for string transduction. We evaluate on cognate detection, transliteration, and grapheme-to-phoneme (G2P) conver-

Proceedings of the Sixth Workshop on Structured Prediction for NLP, pages 52 - 66
May 27, 2022 ©2022 Association for Computational Linguistics

sion. The model jointly learns to perform the task and to generate a latent sequence of edit operations explaining the output. Our approach can flexibly trade off performance and intepretability by using input representations with various degrees of contextualization and outperforms methods that offer a similar degree of interpretability (Tam et al., 2019).

2 Learnable Edit Distance

Edit distance (Levenshtein, 1966) formalizes transcription of a string $\mathbf{s} = (s_1, \ldots, s_n)$ of n symbols from alphabet $\mathcal{S}$ into a string $\mathbf{t} = (t_1, \ldots, t_m)$ of m symbols from alphabet $\mathcal{T}$ as a sequence of operations: delete, insert and substitute, which have different costs.

Ristad and Yianilos (1998) reformulated operations as random events drawn from a distribution of all possible operations: deleting any $s \in \mathcal{S}$, inserting any $t \in \mathcal{T}$, and substituting any pair of symbols from $\mathcal{S} \times \mathcal{T}$. The probability $\mathrm{P}(\mathbf{s}, \mathbf{t}) = \alpha_{n,m}$ of $\mathbf{t}$ being edited from $\mathbf{s}$ can be expressed recursively:

$$\begin{aligned}\alpha_{n,m} \quad = \quad & \alpha_{n,m-1} \cdot \mathrm{P}_{\text{ins}}(t_m) + \\ & \alpha_{n-1,m} \cdot \mathrm{P}_{\text{del}}(s_n) + \\ & \alpha_{n-1,m-1} \cdot \mathrm{P}_{\text{subs}}(s_n, t_m)\end{aligned} \tag{1}$$

This can be computed using the dynamic programming algorithm of Wagner and Fischer (1974), which also computes values of $\alpha_{i,j}$ for all prefixes $\mathbf{s}_{:i}$ and $\mathbf{t}_{:j}$. The operation probabilities only depend on the individual pairs of symbols at positions i, j, so the same dynamic programming algorithm is used for computing the *suffix-pair* transcription probabilities $\beta_{i,j}$ (the backward probabilities).

With a training corpus of pairs of matching strings, the operation probabilities can be estimated using the EM algorithm. In the expectation step, expected counts of all edit operations are estimated for the current parameters using the training data. Each pair of symbols s_i and t_j contribute to the expected counts of the operations:

$$E_{\text{subs}}(s_i, t_j) \mathrel{+}= \alpha_{i-1,j-1} \mathrm{P}_{\text{subs}}(s_i, t_j) \beta_{i,j} / \alpha_{n,m} \tag{2}$$

and analogically for the delete and insert operations. In the maximization step, operation probabilities are estimated by normalizing the expected counts. See Algorithms 1–5 in Ristad and Yianilos (1998) for more details.

3 Neural String Edit Distance Model

In our model, we replace the discrete table of operation probabilities with a probability estimation based on a continuous representation of the input, which brings in the challenge of changing the EM training into a differentiable loss function that can be back-propagated into the representation.

Computation of the transcription probability is shown in Figure 1. We use the same dynamic programming algorithm (Equation 1 and Algorithm 2 in Appendix A) that gradually fills a table of probabilities row by row. The input symbols are represented by learned, possibly contextual embeddings (yellow and blue boxes in Figure 1) which are used to compute a representation of symbol pairs with a small feed-forward network. The symbol pair representation is used to estimate the probabilities of insert, delete and substitute operations (blue, red and green arrows in Figure 1).

Formally, we embed the source sequence $\mathbf{s}$ of length n into a matrix $\mathbf{h}^{\mathbf{s}} \in \mathbb{R}^{n \times d}$ and analogically $\mathbf{t}$ into $\mathbf{h}^{\mathbf{t}} \in \mathbb{R}^{m \times d}$ (yellow and blue boxes in Figure 1). We represent the symbol-pair contexts as a function of the respective symbol representations (small gray rectangles in Figure 1) as a function of repspective symbol representation $\mathbf{c}_{i,j} = f(\mathbf{h}_i^s, \mathbf{h}_j^t)$ depending on the task.

The logits (i.e., the probability scores before normalization) for the edit operations are obtained by concatenation of the following vectors (corresponds to red, green and blue arrows in Figure 1):

- $\mathbf{z}_{\text{del}}^{i,j} = \mathrm{Linear}(\mathbf{c}_{i-1,j}) \in \mathbb{R}^{d_{\text{del}}}$,
- $\mathbf{z}_{\text{ins}}^{i,j} = \mathrm{Linear}(\mathbf{c}_{i,j-1}) \in \mathbb{R}^{d_{\text{ins}}}$,
- $\mathbf{z}_{\text{subs}}^{i,j} = \mathrm{Linear}(\mathbf{c}_{i-1,j-1}) \in \mathbb{R}^{d_{\text{subs}}}$,

where $\mathrm{Linear}(\mathbf{x}) = \mathbf{Wx} + \mathbf{b}$ where $\mathbf{W}$ and $\mathbf{b}$ are trainable parameters of a linear projection and d_{del}, d_{ins} and d_{subs} are the numbers of possible delete, insert and substitute operations given the vocabularies. The distribution $\mathrm{P}_{i,j} \in \mathbb{R}^{d_{\text{del}}+d_{\text{ins}}+d_{\text{subs}}}$ over operations that lead to prefix pair $\mathbf{s}_{:i}$ and $\mathbf{t}_{:j}$ in a single derivation step is

$$\mathrm{P}_{i,j} = \mathrm{softmax}(\mathbf{z}_{\text{del}}^{i,j} \oplus \mathbf{z}_{\text{ins}}^{i,j} \oplus \mathbf{z}_{\text{subs}}^{i,j}).i, j \tag{3}$$

The probabilities $\mathrm{P}_{\text{del}}^{i,j}$, $\mathrm{P}_{\text{ins}}^{i,j}$ and $\mathrm{P}_{\text{subs}}^{i,j}$ are obtained by taking the respective values from the distribution corresponding to the logits.[1] Note that $\mathrm{P}_{i,j}$ only depends on (possibly contextual) input embeddings $\mathbf{h}_i^s$, $\mathbf{h}_{i-1}^s$, $\mathbf{h}_j^t$, and $\mathbf{h}_{j-1}^t$, but not on the derivation of prefix $\mathbf{t}_{:j}$ from $\mathbf{s}_{:i}$.

[1] Using Python-like notation $\mathrm{P}_{\text{del}}^{i,j} = \mathrm{P}_{i,j}[\,:d_{\text{del}}]$, $\mathrm{P}_{\text{ins}}^{i,j} = \mathrm{P}_{i,j}[d_{\text{del}} : d_{\text{del}}{+}d_{\text{ins}}]$, $\mathrm{P}_{\text{subs}}^{i,j} = \mathrm{P}_{i,j}[d_{\text{del}}{+}d_{\text{ins}} :\,]$.

Algorithm 1 Expectation-Maximization Loss

```
1: $\mathcal{L}_{\text{EM}} \leftarrow 0$
2: for $i = 1 \dots n$ do
3:     for $j = 1 \dots m$ do
4:         plausible $\leftarrow \mathbf{0}$                          ▷ Indication vector
5:         ▷ I.e., operations that can be used given $s_i$ and $t_j$
6:         if $j > 1$ then                                     ▷ Insertion is plausible
7:             plausible $\mathrel{+}= \mathbb{1}$(insert $t_j$)
8:             $E^{\text{ins}}_{i,j} \leftarrow \alpha_{i,j-1} \cdot \mathrm{P}_{\text{ins}}(\bullet|\mathbf{c}_{i,j-1}) \cdot \beta_{i,j}$
9:         if $i > 1$ then                                     ▷ Deletion is plausible
10:            plausible $\mathrel{+}= \mathbb{1}$(delete $s_i$)
11:            $E^{\text{del}}_{i,j} \leftarrow \alpha_{i-1,j}\mathrm{P}_{\text{del}}(\bullet|\mathbf{c}_{i-1,j})\beta_{i,j}$
12:        if $i > 1$ and $j > 1$ then                         ▷ Subs. is plausible
13:            plausible $\mathrel{+}= \mathbb{1}$(substitute $s_i \rightarrow t_j$)
14:            $E^{\text{subs}}_{i,j} \leftarrow \alpha_{i-1,j-1} \cdot \mathrm{P}_{\text{subs}}(\bullet|\mathbf{c}_{i-1,j-1}) \cdot \beta_{i,j}$
15:        expected $\leftarrow$ normalize(plausible $\odot$
16:                              $\left[E^{\text{ins}}_{i,j} \oplus E^{\text{del}}_{i,j} \oplus E^{\text{subs}}_{i,j}\right]$)
17:        ▷ Expected distr. can only contain plausible ops.
18:        $\mathcal{L}_{\text{EM}} \mathrel{+}=$ KL($\mathrm{P}_{i,j}$|| expected)
19: return $\mathcal{L}_{\text{EM}}$
```

The transduction probability $\alpha_{i,j}$, i.e., a probability that $\mathbf{s}_{:i}$ transcribes to $\mathbf{t}_{:j}$ (gray circles in Figure 1) is computed in the same way as in Equation 1.

The same algorithm with the reversed order of iteration can be used to compute probabilities $\beta_{i,j}$, the probability that suffix $\mathbf{s}_{i:}$ transcribes to $\mathbf{t}_{j:}$. The complete transduction probability is the same, i.e., $\beta_{1,1} = \alpha_{n,m}$. Tables α and β are used to compute the EM training loss $\mathcal{L}_{\text{EM}}$ (Algorithm 1) which is then optimized using gradient-based optimization. Symbol $\bullet$ in the probability stands for all possible operations (the operations that the model can assign a probability score to), "normalize"' means scale the values such that they sum up to one.

Unlike the statistical model that uses a single discrete multinomial distribution and stores the probabilities in a table, in our neural model the operation probabilities are conditioned on continuous vectors. For each operation type, we compute the expected distribution given the α and β tables (line 6–14). From this distribution, we only select operations that are plausible given the context (line 15), i.e., we zero out the probability of all operations that do not involve symbols s_i and t_j. Finally (line 18), we measure the KL divergence of the predicted operation distribution $\mathrm{P}_{i,j}$ (Equation 3) from the expected distribution, which is the loss function $\mathcal{L}_{\text{EM}}$.

With a trained model, we can estimate the probability of $\mathbf{t}$ being a good transcription of $\mathbf{s}$. Also, by replacing the summation in Equation 1 by the $\max$ operation, we can obtain the most probable operation sequence of operation transcribing $\mathbf{s}$ to $\mathbf{t}$ using the Viterbi (1967) algorithm.

Note that the interpretability of our model depends on how contextualized the input representations $\mathbf{h}^s$ and $\mathbf{h}^t$ are. The degree of contextualization spans from static symbol embeddings with the same strong interpretability as statistical models, to Transformers with richly contextualized representations, which, however, makes our model more similar to standard black-box models.

3.1 String-Pair Matching

Here, our goal is to train a binary classifier deciding if strings $\mathbf{t}$ and $\mathbf{s}$ match. We consider strings matching if $\mathbf{t}$ can be obtained by editing $\mathbf{s}$, with the probability $\mathrm{P}(\mathbf{s}, \mathbf{t}) = \alpha_{n,m}$ higher than a threshold. The model needs to learn to assign a high probability to derivations of matching the source string to the target string and low probability to derivations matching different target strings.

The symbol-pair context $\mathbf{c}_{i,j}$ is computed as

$$\mathrm{LN}\left(\mathrm{ReLU}\left(\mathrm{Linear}(\mathbf{h}^s_i \oplus \mathbf{h}^t_j)\right)\right) \in \mathbb{R}^d, \tag{4}$$

where LN stands for layer normalization and $\oplus$ means concatenation.

The statistical model assumes a single multinomial table over edit operations. A non-matching string pair gets little probability because all derivations (i.e., sequence of edit operations) of non-matching string pairs consist of low-probability operations and high probability is assigned to operations that are not plausible. In the neural model, the same information can be kept in model parameters and we can thus simplify the output space of the model (see Appendix B for thought experiments justifying the design choices).

We no longer need to explicitly model the probability of implausible operations and can only use *a single class* for each type of edit operation (insert, delete, substitute) and one additional *non-match* option that stands for the case when the inputs strings do not match and none of the plausible edit operations is probable (corresponding to the sum of probabilities of the implausible operations in the statistical model).

The value of $\mathrm{P}(\mathbf{s}, \mathbf{t}) = \alpha_{m,n}$ serves as a classification threshold for the binary classification. As additional training signal, we also explicitly optimize the probability using the binary cross-entropy as an auxiliary loss, pushing the value towards 1 for positive examples and towards 0 for negative examples. We set the classification threshold dynamically to maximize the validation F_1-score.

3.2 String Transduction

In the second use case, we use neural string edit distance as a string transduction model: given a source string, edit operations are applied to generate a target string. Unlike classification, we model the transcription process with vocabulary-specific-operations, but still use only a single class for deletion. For the insertion and substitution operation, we use $|\mathcal{T}|$ classes corresponding to the target string alphabet. Unlike classification, we do not add the non-match class. To better contextualize the generation, we add attention to the symbol-pair representation $\mathbf{c}_{i,j}$:

$$\mathrm{LN}\left(\mathrm{ReLU}\left(\mathrm{Linear}(\mathbf{h}_i^s \oplus \mathbf{h}_j^t)\right) \oplus \mathrm{Att}\left(\mathbf{h}_j^t, \mathbf{h}^s\right)\right) \tag{5}$$

of dimension $2d$, where $\mathrm{Att}(\mathbf{q}, \mathbf{v})$ is a multihead attention with queries $\mathbf{q}$ and keys and values $\mathbf{v}$.

While generating the string left-to-right, the only way a symbol can be generated is either by inserting it or by substituting a source symbol. Therefore, we estimate the probability of inserting symbol t_{j+1} given a target prefix $\mathbf{t}_{:j}$ from the probabilities of inserting a symbol after t_j or substituting any s_i by t_{j+1} (i.e., averaging over a row in Figure 1):

$$\begin{aligned} P(t_{j+1}|\hat{\mathbf{t}}_{:j}, \mathbf{s}) &= \sum_{j=1}^{|S|} \alpha_{i,j} \mathrm{P}_{\text{ins}}(t_{j+1}|\mathbf{c}_{i,j}) \\ &+ \sum_{j=2}^{|S|} \alpha_{i,j} \mathrm{P}_{\text{subs}}(s_i, t_{j+1}|\mathbf{c}_{i,j}). \end{aligned} \tag{6}$$

Probabilities P_{ins} and P_{subs} are respective parts of the distribution $\mathrm{P}_{i,j}$ (Equation 3). Probablity P_{del} is unkown at this point because computing it would be computed based on state $\mathbf{c}_{i,j+1}$ which is impossible without what the $(j+1)$-th target symbol is, where logits for P_{ins} and P_{subs} use $\mathbf{c}_{i,j}$ and $\mathbf{c}_{i-1,j}$. Therefore, we approximate Equation 3 as

$$\hat{\mathrm{P}}_{i,j} = \mathrm{softmax}\left(\mathbf{z}_{\text{ins}}^{i,j} \oplus \mathbf{z}_{\text{subs}}^{i,j}\right). \tag{7}$$

At inference time, we decide the next symbol $\hat{t}_j$ based on $\hat{P}_{i,j}$. Knowing the symbol allows computing the $\mathrm{P}_{i,j}$ distribution and values $\alpha_{\bullet,j}$ that are used in the next step of inference. The inference can be done using the beam search algorithm as is done with sequence-to-sequence (S2S) models.

We also use the probability distribution $\hat{P}$ to define an additional training objective which is the *negative log-likelihood* of the ground truth output with respect to this distribution, analogically to training S2S models,

$$\mathcal{L}_{\text{NLL}} = -\sum_{j=0}^{|\mathbf{t}|} \log \sum_{i=0}^{|\mathbf{s}|} \hat{\mathrm{P}}_{i,j}/|\mathbf{s}|. \tag{8}$$

3.3 Interpretability Loss

In our preliminary experiments with Viterbi decoding, we noticed that the model tends to avoid the substitute operation and chose an order of insert and delete operations that is not interpretable. To prevent this behavior, we introduce an additional regularization loss. To decrease the values of α that are further from the diagonal, we add the term $\sum_{i=1}^{n} \sum_{j=1}^{m} |i-j| \cdot \alpha_{i,j}$ to the loss function. Note that this formulation assumes that the source and target sequence have similar lengths. For tasks where the sequence lengths vary significantly, we would need to consider the sequence length in the loss function.

In the string transduction model, optimization of this term can lead to a degenerate solution by flattening all distributions and thus lowering all values in table α. We thus compensate for this loss by adding the $-\log \alpha_{n,m}$ term to the loss function which enforces increasing the α values.

4 Experiments

We evaluate the string-pair matching model on cognate detection, and the string transduction model on Arabic-to-English transliteration and English grapheme-to-phoneme conversion.

In all tasks, we study four ways of representing the input symbols with different degrees of contextualization. The interpretable context-free (unigram) encoder uses symbol embeddings summed with learned position embeddings. We use a 1-D convolutional neural network (CNN) for locally contexualized representation where hidden states correspond to consecutive input n-grams. We use bidirectional recurrent networks (RNNs) and Transformers (Vaswani et al., 2017) for fully contextualized input representations.

Architectural details and hyperparameters are listed in Appendix C. All hyperparameters are set manually based on preliminary experiments. Further hyperparameter tuning can likely lead to better accuracy of both baselines and our model.

	Method	# Param.	Indo-European			Austro-Asiatic		
			Plain	+ Int. loss	Time	Plain	+ Int. loss	Time
	Learnable edit distance	0.2M	32.8 ±1.8	—	0.4h	10.3 ±0.5	—	0.2h
	Transformer `[CLS]`	2.7M	93.5 ±2.1	—	0.7h	78.5 ±0.8	—	0.6h
STANCE	unigram	0.5M	46.2 ±4.9	—	0.2h	16.6 ±0.3	—	0.1h
	RNN	1.9M	80.6 ±1.2	—	0.3h	15.9 ±0.2	—	0.2h
	Transformer	2.7M	76.7 ±1.3	—	0.3h	16.7 ±0.3	—	0.2h
ours	unigram	0.5M	78.5 ±1.0	80.1 ±0.8	1.5h	47.8 ±0.7	48.4 ±0.6	0.7h
	CNN (3-gram)	0.7M	94.0 ±0.7	93.9 ±0.8	0.9h	77.9 ±1.5	76.2 ±1.9	0.5h
	RNN	1.9M	96.9 ±0.6	**97.1 ±0.6**	1.9h	**84.0 ±0.4**	83.7 ±0.5	1.2h
	Transformer	2.7M	87.2 ±1.6	87.3 ±1.8	1.6h	69.9 ±1.0	70.7 ±1.1	1.0h

Table 1: F_1 and training time for cognate detection. F_1 on validation is in Table 6 in the Appendix.

However, preliminary experiments showed that increasing the model size only has a small effect on model accuracy. We run every experiment 5 times and report the mean performance and the standard deviation to control for training stability. The source code for the experiments is available at https://github.com/jlibovicky/neural-string-edit-distance.

Cognate Detection. Cognate detection is the task of detecting if words in different languages have the same origin. We experiment with Austro-Asiatic languages (Sidwell, 2015) and Indo-European languages (Dunn, 2012) normalized into the international phonetic alphabet as provided by Rama et al. (2018).[2]

For Indo-European languages, we have 9,855 words (after excluding singleton-class words) from 43 languages forming 2,158 cognate classes. For Austro-Asiatic languages, the dataset contains 11,828 words of 59 languages, forming only 98 cognate classes without singletons. We generate classification pairs from these datasets by randomly sampling 10 negative examples for each true cognate pair. We use 20k pairs for validation and testing, leaving 1.5M training examples for Indo-European and 80M for Austro-Asiatic languages.

Many cognate detection methods are unsupervised and are evaluated by comparison of a clustering from the method with true cognate classes. We train a supervised classifier, so we use F_1-score on our splits of the dataset.

Because the input and the output are from the same alphabet, we share the parameters of the encoders of the source and target sequences.

As a baseline we use the original statistical learnable edit distance (Ristad and Yianilos, 1998). The well-performing black-box model used as another baseline for comparison with our model is a Transformer processing a concatenation of the two input strings. Similar to BERT (Devlin et al., 2019), we use the representation of the first technical symbol as an input to a linear classifier. We also compare our results with the STANCE model (Tam et al., 2019), a neural model utilizing optimal-transport-based alignment over input text representation which makes similar claims about interpretability as we do. Similar to our model, we experiment with various degrees of representation contextualization.

Transliteration and G2P Conversion. For string transduction, we test our model on two tasks: Arabic-to-English transliteration (Rosca and Breuel, 2016)[3] and English G2P conversion using the CMUDict dataset (Weide, 2017)[4].

The Arabic-to-English transliteration dataset consists of 12,877 pairs for training, 1,431 for validation, and 1,590 for testing. The source-side alphabet uses 47 different symbols; the target side uses 39. The CMUDict dataset contains 108,952 training, 5,447 validation, and 12,855 test examples, 10,999 unique. The dataset uses 27 different graphemes and 39 phonemes.

We evaluate the output strings using Character Error Rate (CER): the standard edit distance between the generated hypotheses and the ground truth string divided by the ground-truth string length; and Word Error Rate (WER): the proportion of words that were transcribed incorrectly. The CMUDict dataset contains multiple transcriptions

[2] https://www.aclweb.org/anthology/attachments/N18-2063.Datasets.zip

[3] https://github.com/google/transliteration

[4] https://github.com/microsoft/CNTK/tree/master/Examples/SequenceToSequence/CMUDict/Data

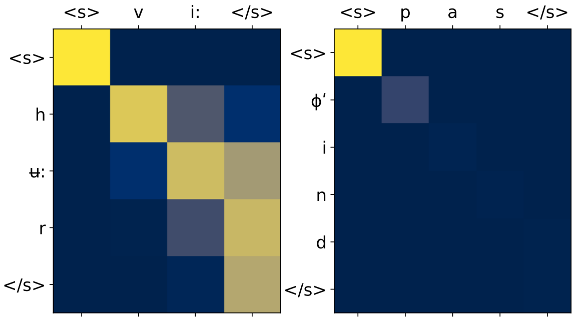

Figure 2: Visualization of the α table (0 is dark blue, 1 is yellow) for cognate detection using a unigram model. Left: A *cognate* pair, Right: a *non-cognate* pair

for some words, as is usually done we select the transcription with the lowest CER as a reference.

Unlike the string-matching task, the future target symbols are unknown. Therefore, when using the contextual representations, we encode the target string using a single-direction RNN and using a masked Transformer, respectively.

To evaluate our model under low-resource conditions, we conduct two sets of additional experiments with the transliteration of Arabic. We compare our unigram and RNN-based models with the RNN-based S2S model trained on smaller subsets of training data (6k, 3k, 1.5k, 750, 360, 180, and 60 training examples) and different embedding and hidden state size (8, 16, . . . , 512).

For the G2P task, where the source and target symbols can be approximately aligned, we further quantitatively assess the model's interpretability by measuring how well it captures alignment between the source and target string. We consider the substitutions in the Viterbi decoding to be aligned symbols. We compare this alignment with statistical word alignment and report the F_1 score. We obtain the source-target strings alignment using Efmaral (Östling and Tiedemann, 2016), a state-of-the-art word aligner, by running the aligner on the entire CMUDict dataset. We use grow-diagonal for alignment symmetrization.

The baseline models are RNN-based (Bahdanau et al., 2015) and Transformer-based (Vaswani et al., 2017) S2S models.

5 Results

Cognate Detection. The results of cognate detection are presented in Table 1 (learning curves are in Figure 5 in Appendix). In cognate detection, our model significantly outperforms both the statistical baseline and the STANCE model. The F_1-score achieved by the unigram model is worse than the Transformer classifier by a large margin. Local representation contextualization with CNN reaches similar performance as the black-box Transformer classifier while retaining a similar strong interpretability to the static embeddings. Models with RNN encoders outperform the baseline classifier, whereas the Transformer encoder yields slightly worse results. Detecting cognates seems to be more difficult in Austro-Asiatic languages than in Indo-European languages. The training usually converges before finishing a single epoch of the training data. An example of how the α captures the prefix-pair probabilities is shown in Figure 2. The interpretability loss only has a negligible (although mostly slightly negative) influence on the accuracy, within the variance of training runs. The ablation study on loss functions (Table 2) shows that the binary cross-entropy plays a more important role. The EM loss alone works remarkably well given that it was trained on positive examples only.

Loss functions	F_1
Complete loss	97.1 ±0.6
— binary XENT for $\alpha_{m,n}$	96.1 ±0.3
— expectation-maximization (Alg. 1)	96.3 ±0.7

Table 2: Ablation study for loss function on Cognate classification with a model with RNN contextualizer.

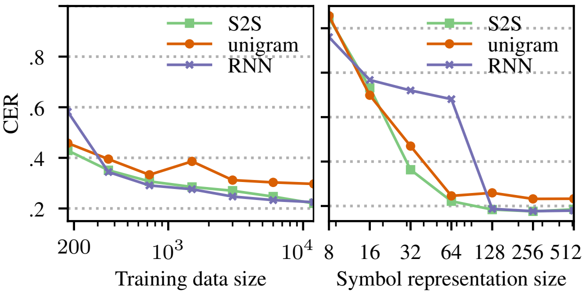

Figure 3: Character Error Rate for Arabic transliteration into English for *various training data sizes* (left) and *various representation sizes* (right).

Transliteration and G2P Conversion. The results for the two transduction tasks are presented in Table 3 (learning curves are in Figure 5 in Appendix). Our transliteration baseline slightly outperforms the baseline presented with the dataset (Rosca and Breuel, 2016, 22.4% CER, 77.1% WER). Our baselines for the G2P conversion perform slightly worse than the best models by

Method		# Param.	Arabic → English					CMUDict						
			Plain		+ Interpret. loss		Time	Plain			+ Interpret. loss			Time
			CER	WER	CER	WER		CER	WER	Align.	CER	WER	Align.	
RNN Seq2seq		3.3M	22.0 ±0.2	75.8 ±0.6	—	—	12m	5.8 ±0.1	23.6 ±0.9	24.5	—	—	—	1.8h
Transformer		3.1M	22.9 ±0.2	78.5 ±0.4	—	—	11m	6.5 ±0.1	26.6 ±0.3	33.2	—	—	—	1.1h
ours	unigram	0.7M	31.7 ±1.8	85.2 ±0.9	31.2 ±1.4	85.0 ±0.5	36m	20.9 ±0.3	67.5 ±1.0	55.7	20.6 ±0.3	66.3 ±0.2	59.5	2.4h
	CNN (3-gram)	1.1M	24.6 ±0.6	80.5 ±0.3	24.5 ±0.9	80.1 ±0.9	41m	12.8 ±1.0	48.4 ±3.1	35.4	12.8 ±0.2	48.4 ±0.6	38.1	2.5h
	Deep CNN	3.0M	24.4 ±0.5	80.0 ±0.7	23.8 ±0.3	79.3 ±0.1	52m	10.8 ±0.5	41.4 ±1.9	23.3	10.8 ±0.5	42.1 ±1.6	28.8	2.5h
	RNN	2.9M	24.1 ±0.2	77.0 ±2.0	22.0 ±0.3	77.4 ±0.8	60m	7.8 ±0.3	31.9 ±1.3	44.7	7.3 ±0.4	33.3 ±1.5	48.9	2.3h
	Transformer	3.2M	24.3 ±0.9	79.0 ±0.7	23.9 ±1.6	78.6 ±1.3	1.2h	10.7 ±1.0	41.8 ±3.1	33.3	10.2 ±1.1	43.6 ±3.2	37.9	2.3h

Table 3: Model error rates for Arabic-to-English transliteration and English G2P generation and respective training times. For the second data set, we also report the alignment F_1 scores (Align.). Our best models are in bold. The error rates on the validation data are in Table 7 in the Appendix.

Loss functions	CER	WER
Complete loss	22.5 ±0.3	77.4 ±0.8
— expectation maximization	68.2 ±7.4	93.5 ±1.0
— next symbol NLL	27.2 ±1.4	81.1 ±2.2
— $\alpha_{m,n}$ maximization	23.5 ±1.3	79.2 ±2.5

Table 4: Ablation study for loss function on Arabic-to-English transliteration using RNN and the underlying representation.

Yolchuyeva et al. (2019), which had 5.4% CER and 22.1% WER with a twice as large model, and 6.5% CER and 23.9% WER with a similarly sized one.

The transliteration of Arabic appears to be a simpler problem than G2P conversion. The performance matches S2S, has fast training times, and there is a smaller gap between the error rates of the context-free and contextualized models.

The training time of our transduction models is 2–3× higher than with the baseline S2S models because the baseline models use builtin PyTorch functions, whereas our model is implemented using loops using TorchScript[5] (15% faster than plain Python). The performance under low data conditions and with small model capacity is in Figure 3.

Models that use static symbol embeddings as the input perform worse than the black-box S2S models in both tasks. Local contextualization with CNN improves the performance over static symbol embeddings. Using the fully contextualized input representation narrows the performance gap between S2S models and neural string edit distance models at the expense of decreased interpretability because all input states can, in theory, contain information about the entire input sequence. The ability to preserve source-target alignment is highest when the input is represented by embeddings only. RNN models not only have the best accuracy, but also capture quite well the source-target alignment. We hypothesize that RNNs work well because of their inductive bias towards sequence processing, which might be hard to learn from position embeddings given the task dataset sizes.

Including the interpretability loss usually slightly improves the accuracy and improves the alignment between the source and target strings. It manifests both qualitatively (Table 5) and quantitatively in the increased alignment accuracy.

Compared to S2S models, beam search decoding leads to much higher accuracy gains, with beam search 5 reaching around 2× error reduction compared to greedy decoding. For all input representations except the static embeddings, length normalization does not improve decoding. Unlike machine translation models, accuracy doesn't degrade with increasing beam size. See Figure 4 in Appendix.

The ablation study on loss functions (Table 4) shows that all loss functions contribute to the final accuracy. The EM loss is most important, direct optimization of the likelihood is second.

6 Related Work

Weighted finite-state transducers. Rastogi et al. (2016) use a weighted-finite state transducer (WFST) with neural scoring function to model sequence transduction. As in our model, they back-propagate the error via a dynamic program. Our model is stronger because, in the WFST, the output symbol generation only depends on the contextualized source symbol embedding, disregarding the string generated so far.

Lin et al. (2019) extend the model by including contextualized target string representation and edit operation history. This makes their model more powerful than ours, but the loss function cannot be exactly computed by dynamic programming and

[5] https://pytorch.org/docs/stable/jit.html

graphemes	phonemes	edit operations
GOELLER	G OW L ER	G→G -O -E -L L→OW +L -E R→ER G→G O→OW -E -L L→L -E R→ER
VOGAN	V OW G AH N	V→V -O G→OW +G +AH -A N→N V→V +OW -O G→G -A N→N
FLAGSHIPS	F L AE G SH IH P S	F→F L→L -A -G S→AE +G -H +SH -I P→IH +P +S F→F L→L +AE -A G→G -S H→SH +IH -I P→P S→S
ENDLER	EH N D L ER	+EH -E N→N D→D L→L -E R→ER E→EH N→N D→D L→L -E R→ER
SWOOPED	S W UW P T	S→S W→W +UW -O -O P→P -E D→T S→S W→W -O O→UW P→P -E D→T

Table 5: Edit operations predicted by RNN-based model for grapheme (blue) to phoneme (green) conversion with and without the interpretability loss (when provided ground-truth target). Green boxes are insertions, blue boxes deletions, yellow boxes substitutions.

requires sampling possible operation sequences.

Segment to Segment Neural Transduction. Yu et al. (2016) use two operation algorithm (shift and emit) for string transduction. Unlike our model directly, it models independently the operation type and target symbols and lacks the concept of symbol substitution.

Neural sequence matching. Several neural sequence-matching methods utilize a scoring function similar to symbol-pair representation. Cuturi and Blondel (2017) propose integrating alignment between two sequences into a loss function that eventually leads to finding alignment between the sequences. The STANCE model (Tam et al., 2019), which we compare results with, first computes the alignment as an optimal transfer problem between the source and target representation. In the second step, they assign a score using a convolutional neural network applied to a soft-alignment matrix. We showed that our model reaches better accuracy with the same input representation. Similar to our model, these approaches provide interpretability via alignment. They allow many-to-many alignments, but cannot enforce a monotonic sequence of operations unlike WFSTs and our model.

Learnable edit distance. McCallum et al. (2005) used trainable edit distance in combination with CRFs for string matching. Recently, Riley and Gildea (2020) integrated the statistical learnable edit distance within a pipeline for unsupervised bilingual lexicon induction. As far as we know, our work is the first using neural networks directly in dynamic programming for edit distance.

Edit distance in deep learning. LaserTagger (Malmi et al., 2019) and EditNTS (Dong et al., 2019) formulate sequence generation as tagging of the source text with edit operations. They use standard edit distance to pre-process the data (so, unlike our model cannot work with different alphabets) and then learn to predict the edit operations. Levenshtein Transformer (Gu et al., 2019) is a partially non-autoregressive S2S model generating the output iteratively via insert and delete operations. It delivers a good trade-off of decoding speed and translation quality, but is not interpretable.

Dynamic programming in deep learning. Combining dynamic programming and neural-network-based estimators is a common technique, especially in sequence modeling. Connectionist Temporal Classification (CTC; Graves et al., 2006) uses the forward-backward algorithm to estimate the loss of assigning labels to a sequence with implicit alignment. The loss function of a linear-chain conditional random field propagated into a neural network (Do and Artieres, 2010) became the state-of-the-art for tasks like named entity recognition (Lample et al., 2016). Loss functions based on dynamic programming are also used in non-autoregressive neural machine translation (Libovický and Helcl, 2018; Saharia et al., 2020).

Cognate detection. Due to the limited amount of annotated data, cognate detection is usually approached using unsupervised methods. Strings are compared using measures such as pointwise mutual information (Jäger, 2014) or LexStat similarity (List, 2012), which are used as an input to a distance-based clustering algorithm (List et al., 2016). Jäger et al. (2017) used a supervised SVM classifier trained on one language family using features that were previously used for clustering and applied the classifier to other language families.

Transliteration. Standard S2S models (Bahdanau et al., 2015; Gehring et al., 2017; Vaswani et al., 2017) or CTC-based sequence-labeling (Graves et al., 2006) are the state of the art for both transliteration (Rosca and Breuel, 2016; Kundu et al., 2018) and G2P conversion (Yao and Zweig, 2015; Peters et al., 2017; Yolchuyeva et al., 2019).

7 Conclusions

We introduced neural string edit distance, a neural model of string transduction based on string edit distance. Our novel formulation of neural string edit distance critically depends on a differentiable loss. When used with context-free representations, it offers a direct interpretability via insert, delete and substitute operations, unlike widely used S2S models. Using input representations with differing amounts of contextualization, we can trade off interpretability for better performance. Our experimental results on cognate detection, Arabic-to-English transliteration and grapheme-to-phoneme conversion show that with contextualized input representations, the proposed model is able to match the performance of standard black-box models. We hope that our approach will help motivate more work on this type of interpretable model and that our framework will be useful in such future work.

Acknowledgments

The work at LMU Munich was supported by was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (No. 640550) and by the German Research Foundation (DFG; grant FR 2829/4-1). The work at CUNI was supported by the European Commission via its Horizon 2020 research and innovation programme (No. 870930).

References

Amina Adadi and Mohammed Berrada. 2018. Peeking inside the black-box: A survey on explainable artificial intelligence (xai). *IEEE Access*, 6:52138–52160.

Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *CoRR*, abs/1607.06450.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Mia Xu Chen, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George Foster, Llion Jones, Mike Schuster, Noam Shazeer, Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Zhifeng Chen, Yonghui Wu, and Macduff Hughes. 2018. The best of both worlds: Combining recent advances in neural machine translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 76–86, Melbourne, Australia. Association for Computational Linguistics.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Marco Cuturi and Mathieu Blondel. 2017. Soft-DTW: a differentiable loss function for time-series. volume 70 of *Proceedings of Machine Learning Research*, pages 894–903, International Convention Centre, Sydney, Australia. PMLR.

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. 2017. Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 933–941. PMLR.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Trinh–Minh–Tri Do and Thierry Artieres. 2010. Neural conditional random fields. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 177–184, Chia Laguna Resort, Sardinia, Italy. PMLR.

Yue Dong, Zichao Li, Mehdi Rezagholizadeh, and Jackie Chi Kit Cheung. 2019. EditNTS: An neural programmer-interpreter model for sentence simplification through explicit editing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3393–3402, Florence, Italy. Association for Computational Linguistics.

Michael Dunn. 2012. Indo-European lexical cognacy database (IELex). Nijmegen, The Netherlands. Max Planck Institute for Psycholinguistics.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1243–1252. PMLR.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. 2006. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376. ACM.

Jiatao Gu, Changhan Wang, and Junbo Zhao. 2019. Levenshtein transformer. In *Advances in Neural Information Processing Systems*, pages 11179–11189.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society.

Benjamin Hoover, Hendrik Strobelt, and Sebastian Gehrmann. 2020. exBERT: A Visual Analysis Tool to Explore Learned Representations in Transformer Models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 187–196, Online. Association for Computational Linguistics.

Sathish Reddy Indurthi, Insoo Chung, and Sangha Kim. 2019. Look harder: A neural machine translation model with hard attention. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3037–3043, Florence, Italy. Association for Computational Linguistics.

Gerhard Jäger. 2014. Phylogenetic inference from word lists using weighted alignment with empirically determined weights. In *Quantifying Language Dynamics*, pages 155–204. Brill.

Gerhard Jäger, Johann-Mattis List, and Pavel Sofroniev. 2017. Using support vector machines and state-of-the-art algorithms for phonetic alignment to identify cognates in multi-lingual wordlists. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1205–1216, Valencia, Spain. Association for Computational Linguistics.

Sarthak Jain and Byron C. Wallace. 2019. Attention is not Explanation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3543–3556, Minneapolis, Minnesota. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Soumyadeep Kundu, Sayantan Paul, and Santanu Pal. 2018. A deep learning based approach to transliteration. In *Proceedings of the Seventh Named Entities Workshop*, pages 79–83, Melbourne, Australia. Association for Computational Linguistics.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 260–270, San Diego, California. Association for Computational Linguistics.

Vladimir I Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710.

Jindřich Libovický and Jindřich Helcl. 2018. End-to-end non-autoregressive neural machine translation with connectionist temporal classification. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3016–3021, Brussels, Belgium. Association for Computational Linguistics.

Chu-Cheng Lin, Hao Zhu, Matthew R. Gormley, and Jason Eisner. 2019. Neural finite-state transducers: Beyond rational relations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 272–283, Minneapolis, Minnesota. Association for Computational Linguistics.

Zachary C. Lipton. 2018. The mythos of model interpretability. *Queue*, 16(3):31–57.

Johann-Mattis List. 2012. LexStat: Automatic detection of cognates in multilingual wordlists. In *Proceedings of the EACL 2012 Joint Workshop of LINGVIS & UNCLH*, pages 117–125, Avignon, France. Association for Computational Linguistics.

Johann-Mattis List, Philippe Lopez, and Eric Bapteste. 2016. Using sequence similarity networks to identify partial cognates in multilingual wordlists. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 599–605, Berlin, Germany. Association for Computational Linguistics.

Eric Malmi, Sebastian Krause, Sascha Rothe, Daniil Mirylenka, and Aliaksei Severyn. 2019. Encode, tag, realize: High-precision text editing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5054–5065, Hong Kong, China. Association for Computational Linguistics.

Andrew McCallum, Kedar Bellare, and Fernando C. N. Pereira. 2005. A conditional random field for discriminatively-trained finite-state string edit distance. In *UAI '05, Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence, Edinburgh, Scotland, July 26-29, 2005*, pages 388–395. AUAI Press.

Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. 2014. Recurrent models of visual attention. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2204–2212.

Robert Östling and Jörg Tiedemann. 2016. Efficient word alignment with Markov Chain Monte Carlo. *Prague Bulletin of Mathematical Linguistics*, 106:125–146.

Ben Peters, Jon Dehdari, and Josef van Genabith. 2017. Massively multilingual neural grapheme-to-phoneme conversion. In *Proceedings of the First Workshop on Building Linguistically Generalizable NLP Systems*, pages 19–26, Copenhagen, Denmark. Association for Computational Linguistics.

Taraka Rama, Johann-Mattis List, Johannes Wahle, and Gerhard Jäger. 2018. Are automatic methods for cognate detection good enough for phylogenetic reconstruction in historical linguistics? In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 393–400, New Orleans, Louisiana. Association for Computational Linguistics.

Pushpendre Rastogi, Ryan Cotterell, and Jason Eisner. 2016. Weighting finite-state transductions with neural context. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 623–633, San Diego, California. Association for Computational Linguistics.

Parker Riley and Daniel Gildea. 2020. Unsupervised bilingual lexicon induction across writing systems. *CoRR*, abs/2002.00037.

Eric Sven Ristad and Peter N. Yianilos. 1998. Learning string-edit distance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(5):522–532.

Mihaela Rosca and Thomas Breuel. 2016. Sequence-to-sequence neural network models for transliteration. *CoRR*, abs/1610.09565.

Chitwan Saharia, William Chan, Saurabh Saxena, and Mohammad Norouzi. 2020. Non-autoregressive machine translation with latent alignments. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1098–1108, Online. Association for Computational Linguistics.

Paul Sidwell. 2015. Austroasiatic dataset for phylogenetic analysis: 2015 version. *Mon-Khmer Studies (Notes, Reviews, Data-Papers)*, 44:lxviii–ccclvii.

Derek Tam, Nicholas Monath, Ari Kobren, Aaron Traylor, Rajarshi Das, and Andrew McCallum. 2019. Optimal transport-based alignment of learned character representations for string similarity. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5907–5917, Florence, Italy. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 5998–6008.

Andrew J. Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans. Inf. Theory*, 13(2):260–269.

Robert A. Wagner and Michael J. Fischer. 1974. The string-to-string correction problem. *J. ACM*, 21(1):168–173.

Robert Weide. 2017. The Carnegie-Mellon pronouncing dictionary [cmudict. 0.7]. Pittsburgh, PA, USA. Carnegie Mellon University.

Sarah Wiegreffe and Yuval Pinter. 2019. Attention is not not explanation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 11–20, Hong Kong, China. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Kaisheng Yao and Geoffrey Zweig. 2015. Sequence-to-sequence neural net models for grapheme-to-phoneme conversion. In *INTERSPEECH 2015, 16th Annual Conference of the International Speech Communication Association, Dresden, Germany, September 6-10, 2015*, pages 3330–3334. ISCA.

Sevinj Yolchuyeva, Géza Németh, and Bálint Gyires-Tóth. 2019. Transformer based grapheme-to-phoneme conversion. In *Interspeech 2019, 20th Annual Conference of the International Speech Communication Association, Graz, Austria, 15-19 September 2019*, pages 2095–2099. ISCA.

Lei Yu, Jan Buys, and Phil Blunsom. 2016. Online segment to segment neural transduction. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1307–1316, Austin, Texas. Association for Computational Linguistics.

A Inference algorithm

Algorithm 2 is a procedural implementation of Equation 1. In the Viterbi decoding used for obtaining the alignment, the summation on line 6, 8 and 10 is replaced by taking the maximum.

Algorithm 2 Forward evaluation

```
1:  $\alpha \in \mathbb{R}^{n \times m} \leftarrow \mathbf{0}$
2:  $\alpha_{0,0} \leftarrow 1$
3:  for $i = 1 \dots n$ do
4:      for $j = 1 \dots m$ do
5:          if $j > 0$ then
6:              $\alpha_{i,j} \mathrel{+}= \mathrm{P}_{\text{ins}}(t_j|\mathbf{c}_{i,j-1}) \cdot \alpha_{i,j-1}$
7:          if $i > 0$ then
8:              $\alpha_{i,j} \mathrel{+}= \mathrm{P}_{\text{del}}(s_i|\mathbf{c}_{i-1,j}) \cdot \alpha_{i-1,j}$
9:          if $i > 0$ and $j > 0$ then
10:             $\alpha_{i,j} \mathrel{+}= \mathrm{P}_{\text{subs}}(s_i \rightarrow t_j|\mathbf{c}_{i-1,j-1}) \cdot \alpha_{i-1,j-1}$
```

B Motivation for design choices in the string-matching model

Let us assume a toy example transliteration. The source alphabet is $\{\mathtt{A}, \mathtt{B}, \mathtt{C}\}$, the target alphabet is $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, the transcription rules are:

1. If B is at the beginning of the string, delete it.
2. Multiple As rewrite to a single a.
3. Rewrite B to b and C to c.

The statistical learnable edit distance would not be capable of properly learning rules 1 and 2 because it would not know that B was at the beginning of the string and if an occurrence of A is the first A. This problem gets resolved by introducing a contextualized representation of the input.

The original statistical EM algorithm only needs positive examples to learn the operation distribution. For instance, rewriting B to c will end up as improbable due to the inherent limitation of a single sharing static probability table. Using a single table regardless of the context means that if some operations become more probable, the others must become less probable. A neural network does not have such limitations. A neural model can in theory find solutions that maximize the probability of the training data, however, do not correspond to the original set of rules by finding a highly probable sequence of operations for any string pair. For instance, it can learn to count the positions in the string:

1′. Whatever symbols at the same position i (s_i and t_i) are, substitute s_i with t_j with the probability of 1.

2′. If $i < j$, assign probability of 1 to deleting s_i.

3′. If $i > j$, assign probability of 1 to inserting t_j.

For this reason, we introduce the binary cross-entropy as an additional loss function. This should steer the model away from degenerate solutions assigning a high probability score to any input string pair.

But our ablation study in Table 2 showed that even without the binary cross-entropy loss, the model converges to a good non-degenerate solution.

This thought experiment shows keeping the full table of possible model outcomes is no longer crucial for the modeling strength. Let us assume that the output distribution of the neural model contains all possible edit operations as they are in the static probability tables of the statistical model. The model can learn to rely on the position information only and select the correct symbols in the output probability distribution ignoring the actual content of the symbols, using their embeddings as a key to identify the correct item from the output distribution. The model can thus learn to ignore the function the full probability table had in the statistical model. Also, given the inputs, it is always clear what the plausible operations are, it is easy for the model not to assign any probability to the implausible operations (unlike the statistical model).

These thoughts lead us to the conclusion that there is no need to keep the full output distribution and we only can use four target classes: one for insertion, one for deletion, one for substitution, and one special class that would get the part of probability mass that would be assigned to implausible operations in the statistical model. We call the last one the *non-match* option.

C Model Hyperparameters

Following Gehring et al. (2017), the CNN uses gated linear units as non-linearity (Dauphin et al., 2017), layer normalization (Ba et al., 2016) and residual connections (He et al., 2016). The symbol embeddings are summed with learnable position embeddings before the convolution.

The RNN uses gated recurrent units (Cho et al., 2014) and follows the scheme of Chen et al. (2018), which includes residual connections (He et al., 2016), layer normalization (Ba et al., 2016), and multi-headed scaled dot-product attention (Vaswani et al., 2017).

The Transformers follow the architecture decisions of BERT (Devlin et al., 2019) as implemented in the Transformers library (Wolf et al., 2020).

All hyperparameters are set manually based on preliminary experiments. For all experiments, we use embedding size of 256. The CNN encoder uses a single layer with kernel size 3 and ReLU non-linearity. For both the RNN and Transformer models, we use 2 layers with 256 hidden units. The Transformer uses 4 attention heads of dimension 64 in the self-attention. The same configuration is used for the encoder-decoder attention for both RNN and Transformer. We use the same hyperparameters also for the baselines.

We include all main loss functions with weight 1.0, i.e., for string-pair matching: the EM loss, non-matching negative log-likelihood and binary cross-entropy; for string transduction: the EM loss and next symbol negative log-likelihood. We test each model with and without the interpretability loss, which is included with weight 0.1.

We optimize the models using the Adam optimizer (Kingma and Ba, 2015) with an initial learning rate of 10^{-4}, and batch size of 512. We validate the models every 50 training steps. We decrease the learning rate by a factor of 0.7 if the validation performance does not increase in two consecutive validations. We stop the training after the learning rate decreases 10 times.

D Notes on Reproducibility

The training times were measured on machines with GeForce GTX 1080 Ti GPUs and with Intel Xeon E5–2630v4 CPUs (2.20GHz). We report average wall time of training including data preprocessing, validation and testing. The measured time might be influenced by other processes running on the machines.

Validation scores are provided in Tables 6 and 7.

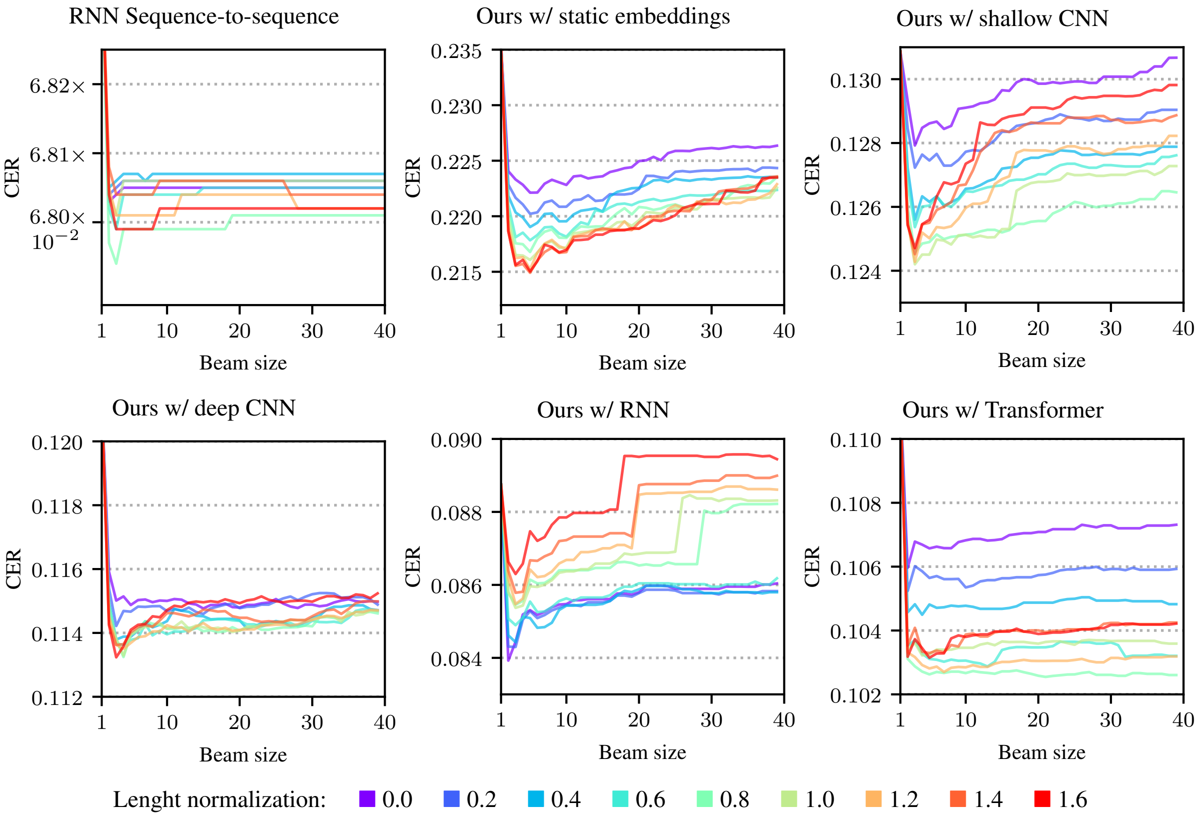

Figure 4: Effect of beam search on test data for grapheme-to-phoneme conversion.

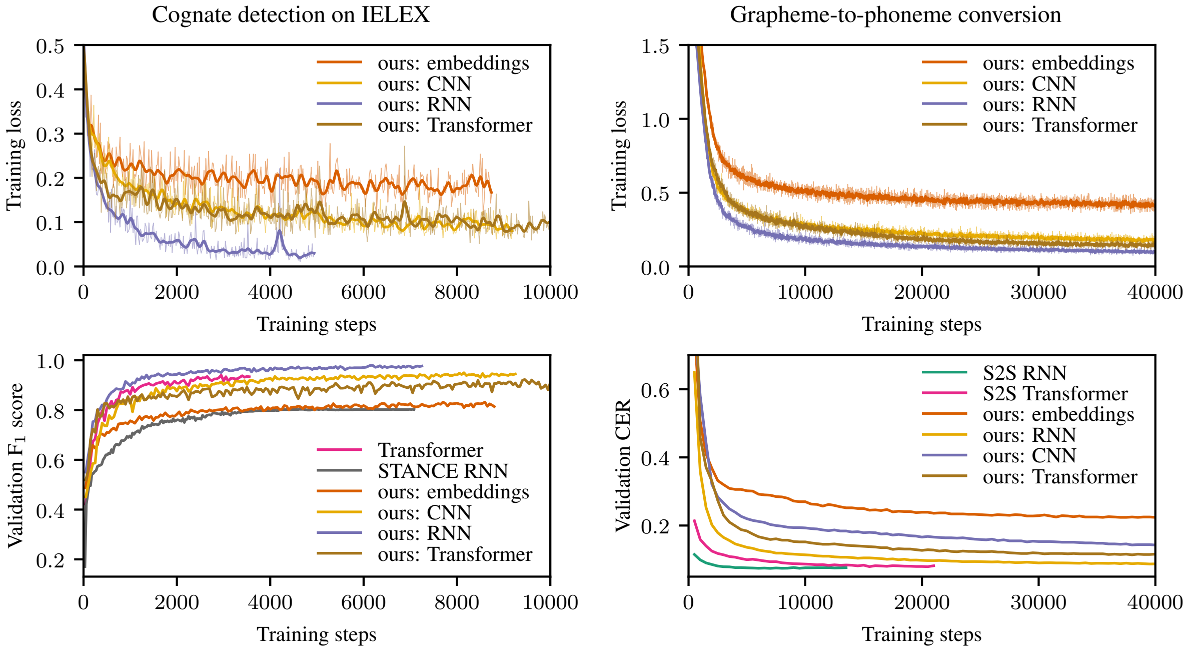

Figure 5: Learning curves for Cognate classification for Indo-European languages (left) and for grapheme-to-phoneme conversion (right).

Method		Indo-European		Austro-Asiatic	
		Base	+ Int. loss	Base	+ Int. loss
Transformer `[CLS]`		91.4 ±2.8	—	78.8 ±0.8	—
STANCE	unigram	46.5 ±4.7	—	16.5 ±0.4	—
	RNN	80.4 ±1.6	—	16.5 ±0.1	—
	Transformer	76.8 ±1.3	—	17.2 ±0.2	—
ours	unigram	81.2 ±1.0	82.0 ±0.5	52.6 ±0.8	53.9 ±0.6
	CNN (3-gram)	95.2 ±0.6	94.9 ±0.7	78.9 ±0.8	78.1 ±1.7
	RNN	97.2 ±0.2	88.8 ±1.1	82.8 ±0.6	83.1 ±0.7
	Transformer	88.8 ±1.6	88.7 ±1.1	71.5 ±1.1	71.5 ±1.1

Table 6: F_1-score for cognate detection on the *validation* data.

Method		Arabic → English				CMUDict			
		Base		+ Int. loss		Base		+ Int. loss	
		CER	WER	CER	WER	CER	WER	CER	WER
RNN Seq2seq		21.7 ±0.1	75.0 ±0.6	—	—	7.4 ±0.0	31.5 ±0.1	—	—
Transformer		22.8 ±0.2	77.7 ±0.6	—	—	7.8 ±0.1	32.7 ±0.3	—	—
ours	unigram	28.4 ±0.7	84.1 ±0.8	28.3 ±0.5	84.3 ±0.7	21.2 ±1.0	66.4 ±1.9	21.5 ±0.8	68.0 ±2.1
	CNN (3-gram)	34.4 ±1.1	86.5 ±0.8	32.2 ±1.1	86.5 ±0.8	36.0 ±5.7	80.9 ±3.2	33.8 ±3.5	79.0 ±2.8
	RNN	42.4 ±9.0	90.9 ±5.4	45.2 ±2.6	90.9 ±1.8	59.1 ±2.5	96.2 ±0.7	43.6 ±5.6	80.5 ±5.6
	Transformer	41.2 ±9.1	91.7 ±4.4	47.7 ±3.6	92.5 ±2.4	24.6 ±4.3	73.8 ±6.1	43.5 ±3.6	84.9 ±2.5

Table 7: Model error-rates for Arabic-to-English transliteration and English G2P generation on *validation* data.

.