Learning Better Transliterations

Jeff Pasternack
University of Illinois at Urbana-Champaign
Department of Computer Science
201 North Goodwin Avenue
Urbana, Illinois 61801-2302
jpaster2@uiuc.edu

Dan Roth
University of Illinois at Urbana-Champaign
Department of Computer Science
201 North Goodwin Avenue
Urbana, Illinois 61801-2302
danr@uiuc.edu

ABSTRACT

We introduce a new probabilistic model for transliteration that performs significantly better than previous approaches, is language-agnostic, requiring no knowledge of the source or target languages, and is capable of both generation (creating the most likely transliteration of a source word) and discovery (selecting the most likely transliteration from a list of candidate words). Our experimental results demonstrate improved accuracy over the existing state-of-the-art by more than 10% in Chinese, Hebrew and Russian. While past work has commonly made use of fixed-size n-gram features along with more traditional models such as HMM or Perceptron, we utilize an intuitive notion of "productions", where each source word can be segmented into a series of contiguous, non-overlapping substrings of any size, each of which independently transliterates to a substring in the target language with a given probability (e.g. $P(\text{wash}\Rightarrow\text{ваш}) = 0.95$). To learn these parameters, we employ Expectation-Maximization (EM), with the alignment between substrings in the source and target word training pairs as our latent data. Despite the size of the parameter space and the $2^{|w|-1}$ possible segmentations to consider for each word, by using dynamic programming each iteration of EM takes $O(m^6n)$ time, where m is the length of the longest word in the data and n is the number of word pairs, and is very fast in practice. Furthermore, discovering transliterations takes only $O(m^4w)$ time, where w is the number of candidate words to choose from, and generating a transliteration takes $O(m^2k^2)$ time, where k is a pruning constant (we used a value of 100). Additionally, we are able to obtain training examples in an unsupervised fashion from Wikipedia by using a relatively simple algorithm to filter potential word pairs.

Categories and Subject Descriptors

I.2.7 [**Computing Methodologies**]: Artificial Intelligence—*Natural Language Processing*; H.3.m [**Information Systems**]: Information Storage and Retrieval—*Miscellaneous*

General Terms

Algorithms, Experimentation, Languages

Keywords

transliteration, translation, probabilistic models, multi-lingual information retrieval

1. INTRODUCTION

Transliteration, where a word is transformed into another language with a pronunciation as close as possible to the original, is integral to both translation and multi-lingual information retrieval. Perhaps the most common method is the simplest: transliteration tables of deterministic mappings of short character sequences (e.g. shch⇒щ). Although ubiquitous and computationally trivial, accuracy is quite low; an ambiguous letter such as the 'a' in "Jan" and "Jane" will readily defy such an approach. Many grapheme models have attempted to overcome this by learning more comprehensive, weighted mappings based on unigram, bigram or trigram features, but it is easy to find problem cases here, too (e.g. "ough" in thought and though). Alternatively, one can employ phonetics, to predict first the sound of the source word and then the transcription of that sound into the target language; the phonetic model embodies a vast store of prior knowledge about the source language (or must itself be learned), but even assuming this exists, performance is limited since transliterations are influenced by both the original sound and the original spelling. Many pairs of words (Terra and Tera) are homophones but have differently-spelled transliterations (Russian: Терра and Тера).

We instead introduce a new grapheme model of transliteration that requires no prior knowledge of either the source or target language. Instead of learning mappings of fixed-size n-grams of the source language to fixed-size n-grams of the target language, we permit the mapped substrings to be of any length, allowing us to learn that walk⇒уок, even when wal⇒{уо, уол, вал, ...} is quite ambiguous. In general, longer source substrings have more certain mappings, and it is this phenomenon that helps us avoid deleterious overfitting even on small training sets: a longer character substring such as "doaldo" may only be seen once during training, but if we see the same substring again when predicting a transliteration, it's very likely that the same transliteration (доалдо) applies. Of course, because our mappings (which we will also refer to as *productions*) are variable length, the number of ways of segmenting a source word of length |S| is exponential, $2^{|S|-1}$; for example, "John" may be segmented as John (one segment), J-ohn, Jo-hn, Joh-n, J-o-hn, Jo-h-n, J-oh-n, and J-o-h-n, and the segmentation used determines which productions apply (do we transliterate the substrings "J" and "ohn"? "Jo" and "hn"?) Certainly, if we attempted to use a naive algorithm, the problem would be intractable but, as we shall see later when we present our training and prediction algorithms, we can overcome this with dynamic programming.

Our key contribution is a high performance, language-agnostic, supervised discriminative model for transliteration with relatively low time complexity that is capable of both transliteration generation and discovery. We use both the training set from [4] and data automatically extracted from Wikipedia with a simple filtering algorithm, comparing against both the supervised algorithm of [4] as well as the weakly supervised approaches of [2, 7] and show substantial improvement over past results in both cases, increasing accuracies by more than 10% for Russian, Hebrew and Chinese.

We next examine and compare to previous work before formally introducing our algorithms and EM derivation; we then present our evaluation methodology and results before finally concluding.

2. PREVIOUS WORK

Existing transliteration methods can be examined in four basic dimensions: whether they are probabilistic or non-probabilistic, whether they can generate transliterations directly or only discover them from lists of candidates, whether they employ supervised or unsupervised learning, and their features.

2.1 Probabilistic vs. Non-Probabilistic

Probabilistic approaches include HMM [6], weighted finite state transducers [8, 11, 10], and joint source-channel models [5], while the non-probabilistic include Perceptron [7] and constrained optimization [4]. Probabilistic models incorporate probability distributions as parameters, and may be generative (modeling the joint distribution of both the words to be transliterated and the transliterations) or discriminative (modeling transliteration alone and taking the source words as given). Note that whether a probabilistic model is generative is orthogonal to whether the model is capable of generating transliterations (as discussed next), despite the overlap in terminology. Non-probabilistic models depend on other parameters typically referred to as weights, and can be seen as analogous to discriminative probabilistic models in that they find transliterations for a given word (maximizing "score" rather than probability) but cannot "generate" source words or word pairs.

2.2 Generation vs. Discovery

Discovery, selecting the correct transliteration from a relatively small list of candidates, is a much easier task than generation, where the transliteration is created "from scratch". While some probabilistic methods (including ours, [6] and [1]) are capable of both, the remainder–and all non-probabilistic models–are limited to discovery alone. Generation is important in two cases: when a candidate list does not contain an esoteric or novel transliteration, or when the candidate list grows very long. When we perform discovery for Russian with the full, 47332 word candidate list from [7], for instance, each iteration of EM with 2778 training pairs takes about ten seconds, but finding predictions for the 727 evaluation words takes roughly seven *hours*, a problem also observed by [2]. In such cases we can avoid testing each candidate word by instead generating a list of transliterations and selecting the highest-probability possibility also appearing in the candidate list (checked with a constant-time lookup on a hashtable).

2.3 Supervised vs. Unsupervised

Most methods are fully-supervised with a minimum of several thousand labeled examples. Some models (e.g. [12]) may alternatively require prior knowledge of the source and target language phonetics. However, [7] uses an "almost unsupervised" setting with twenty labeled examples and takes advantage of the temporal alignment within a multilingual corpora to learn its parameters. [2] forgoes any word pairs and instead uses Romanization tables and expert knowledge in the form of constraints. By contrast, we use labeled examples, but extract them automatically from Wikipedia. Given the size and coverage of Wikipedia (there are currently 27 languages with more than 100,000 pages each, and 89 with at least 10,000), this means such training pairs can be readily obtained for most languages of interest.

2.4 Features

The most common features used in both the probabilistic and non-probabilistic grapheme models are unigrams, bigrams, and trigrams, and possibly unigrams alone as in [2]. [1] and [6] use GIZA++ [9] to align source and target words, which then allows them to map a sequence of English characters to a single Arabic letters by considering the sequence to be a "composite character", though only the most frequently occurring such "composites" are kept (the top 50 and top 100, respectively). [10]'s features are perhaps the most relevant to our work, as they also use variable-length substring to substring mappings to construct a transducer, although the length is still limited at 4 for best performance.

3. OUR MODEL

Let S be the original word in the source language and T be the transliteration in the target language, with a joint probability of $P(S, T) = P(T|S)P(S)$. When we predict a transliteration, the source word is fixed, so $P(S)$ is constant and we instead seek to find a transliteration T for S such that $P(T|S)$ is maximized.

We conceptualize the transliterator, given S, as first selecting a segmentation for S according to a distribution over all possible segmentations, and then transliterating each segment independently according to a probabilistic mapping, which then become the segments of the transliteration T. For example, given the word "perelman" we can segment it as per-el-man, perel-man, per-elman, etc. If we choose per-el-man, we transliterate "per", "el", and "man"; after training our model for Hebrew, we have $P_{prod}(\text{פר}|per) = 0.9993$, $P_{prod}(\text{ל}|el) = 0.9145$, and $P_{prod}(\text{מן}|man) = 0.9857$, so the probability of generating the correct transliteration (פרלמן) given this segmentation is proportional to the product of these probabilities.

Let seg_W be a particular segmentation of a word W into $|seg_W|$ segments, where the i^{th} segment of W is denoted as seg^i_W, and let all_W be the set of all $2^{|W|-1}$ possible segmentations. Then $all_{W,U} = \{(seg_W, seg_U) \in all_W \times all_U : |seg_W| = |seg_U|\}$ is the set of all pairs of segmentations of

W and U where the number of segments in both is equal. Then we can calculate $P(T|S)$ as:

$$P(T|S) = \frac{1}{Z} \sum_{\substack{(seg_S, seg_T) \in \\ all_{S,T}}} P(seg_S) \prod_{i=1}^{|seg_S|} P_{prod}(seg_T^i | seg_S^i)$$

where Z is the normalization constant, and $P_{prod}(t|s)$ is the probability of transliterating substring s as t.

Notice that for any particular segmentation of S, there are many possible segmentations of T of the same length, so the segments of seg_S may transliterate to T in multiple ways; e.g. in whe-eler (וילר), “whe” may transliterate to either ו or וי with varying probability, and “eler” may transliterate to ילר or לר. Of course, different segmentations of S (e.g. whe-eler and wh-eeler) can also ultimately produce the same transliteration T. Because of this, we marginalize over all possible segmentations of S and T (where the number of segments is equal), effectively weighting each possibility by $P(seg_S)$. When a seg_S^i character sequence is unseen during training, we may either assume that $P_{prod}(seg_T^i|seg_S^i) = 0$ or employ a smoothing mechanism as we did in our experiments, discussed later.

This just leaves us with calculating $P(seg_S)$, the probability of a particular segmentation of S. Clearly, choosing a bad segmentation is detrimental: wh-e-e-ler can map the first e to י, but cannot produce an ל from the second. Rather than trying to learn a robust model for segmentation, however, we adopt a simple approach, taking $P(seg_S) \propto c^{|seg_S|}$, where $c > 0$ is a constant that determines the relative preference for feasible segmentations with fewer segments (but note that if $|seg_S| > |T|$, that segmentation is impossible since there can be no mapping from each segment to at least one character of T) . At $c = 0.5$, a segmentation containing one segment is twice as likely as a segmentation with two segments, which is itself twice as likely as a segmentation with three segments; at $c = 1$, all segmentations are equiprobable. Since the number of segmentations of length m in a word of length n is $\binom{n-1}{m-1}$, a value of $c = 0.5$ does not mean that the total probability of all segments of size k is twice as much as those of size k-1, however. With $c = 1$, for example, the total probability of segmentations of length $\frac{n-1}{2}$ is greatest (assuming $|T| \geq \frac{n-1}{2}$). Now we can explicitly write out $P(T|S)$ as:

$$P(T|S) = \frac{1}{Z} \sum_{\substack{(seg_S, seg_T) \in \\ all_{S,T}}} c^{|seg_S|} \prod_{i=1}^{|seg_S|} P_{prod}(seg_T^i | seg_S^i)$$

In our experiments, we tried both values of both 0.5 and 1 for c, with relatively little difference (no more than a few percent in accuracy). In general we found that a value of 0.5 yielded slightly better performance when generating transliterations by favoring fewer (and thus longer) segments that had a greater chance of producing the exact transliteration, but a value of 1 was slightly better in discovery because biasing towards longer segments also encouraged the selection of the wrong candidate when none of the candidates were exact transliterations (this was particularly true for the Russian evaluation data, where the nouns often had endings associated with their grammatical case that were independent of the source word and not present in our training examples). For clarity, all the results we report in this paper use $c = 1$.

Our algorithm for finding $P(T|S)$ using our model is given as Algorithm 1. We use the notation $U[a, b]$ to refer to a substring of string U starting from the character at index a and ending with the character at index b (the first character in a string being at index 1). The algorithm recurses to find $P(T[j+1, |T|] \mid S[j+1, |S|])$, and implements dynamic programming using a $MemoizationTable$ to store the results of these subproblems. As a result, the total number of recursions is $O(|S| \cdot |T|)$, and as the amount of work in each call is also $O(|S| \cdot |T|)$, the total time complexity is $O(|S|^2 \cdot |T|^2)$.

Additionally, we may also wish to *generate* transliterations from a source word S. Our algorithm for finding the k-highest probability transliterations is given by Algorithm 2, where k also serves as a pruning constant that limits the amount of work done. The algorithm recurses $O(|S|)$ times, and does $O(|S| \cdot k^2)$ work in each call, for a total time complexity of $O(|S|^2 \cdot k^2)$. In practice we found that values of k above 100 produced essentially identical results; consequently, in our generation experiment, $k = 100$.

Algorithm 1 Finding $P(T|S)$ with Dynamic Programming

```
Require: Production probabilities $P_{prod}(t|s)$
  if $(T, S) \in MemoizationTable$ then
    return $MemoizationTable(T, S)$
  else if $|S| = |T| = 0$ then
    return 1 {base case, $P(\text{null}|\text{null}) = 1$}
  else if $|S| = 0 \vee |T| = 0$ then
    return 0 {$P(\text{null}|\text{non-null}) = P(\text{null}|\text{non-null}) = 0$}
  else
    $R \Leftarrow 0$
    for $i = 1 \ldots |S| - 1$ do
      for $j = 1 \ldots |T| - 1$ do
        $O \Leftarrow c \cdot P_{prod}(T[1, j] \mid S[1, i])$
        $R \Leftarrow R + O \cdot P(T[j+1, |T|] \mid S[j+1, |S|])$ {recurse}
      end for
    end for
    $MemoizationTable(T, S) \Leftarrow R$
    return $R$
  end if
```

3.1 Training the Model

Our model is trained using Expectation-Maximization [3] to iteratively update our $P_{prod}(t|s)$ model parameters; we discuss the initial parameters used in our experiments later, in section 4.3. Before we present the formal definition and derivation of our EM parameter update rule in the next section, we first give a more intuitive description along with the dynamic programming algorithm used to efficiently implement these updates.

We learn from example pairs of words from the source and target languages, (S, T). If we knew the “valid” segmentations for each S and their corresponding segmentations for each T (i.e. the “alignments” between character sequences), finding the production probabilities for substrings in the target (t) and source (s) languages, $P_{prod}(t|s)$, would be a trivial matter of counting. Unfortunately, we do not have this information. While [1] and [6] use GIZA++ to find alignments, we will instead take them as our latent (hidden) parameters. In the expectation step, we use our current parameters (the production probabilities) to assign a probability to every possible alignment of S and T. Then, in the maximization step, we simply count the number of

Algorithm 2 Generating Top K Transliterations for S

```
Require: Production probabilities $P_{prod}(t|s)$
Require: Pruning constant $k$
  if $S \in MemoizationTable$ then
    return $MemoizationTable(S)$
  else if $|S| = 0$ then
    $R \Leftarrow EmptyTable$
    $R[\text{``''}] \Leftarrow 1$ {$P(null|null) = 1$}
    return $R$ {base case}
  else
    for $i = 1 \ldots |S| - 1$ do
      $R \Leftarrow EmptyTable$
      $Q \Leftarrow TopK(S[i+1, |S|])$ {recurse}
      for all $k$-highest $t \in P_{prod}(t \mid S[1, i])$ do
        for all $a \in Q$ do
          $R[t + a] \Leftarrow R[t + a] + Q[a] \cdot P_{prod}(t \mid S[1, i])$
        end for
      end for
    end for
    Remove all but the $k$-highest entries from $R$
    $MemoizationTable(S) \Leftarrow R$
    return $R$
  end if
```

each pair of aligned substrings, weighted by the probability of the alignments in which they appear. Then, we sum the weighted counts for each pair over all the training examples, and condition over the source substrings to get our new $P_{prod}(t|s)$ parameters.

Our training algorithm is presented as Algorithm 3. Each alignment (a pair of segmentations, seg_S and seg_T, of equal length) has a probability given by:

$$\frac{1}{y} \cdot c^{|seg_S|} \prod_{i=1}^{|seg_S|} P_{prod}(seg_T^i | seg_S^i)$$

Where y is a normalization constant. For each training example, we (in principle) find all such alignments and, for each pair of substrings (s, t), find the sum of the probabilities of all alignments that align s with t. Our algorithm accomplishes this tractably by, as with prediction, memoizing the subproblems. The returned value is a pair, Q, where $Q[1]$ is the normalization constant y and $Q[0]$ is a table of substring pairs (s, t) and their associated (unnormalized) probabilities. These probabilities are normalized, and then added to each substring pair's overall count (over all examples) is increased by that probability (so if, for an individual example, the sum of all the probabilities of alignments in which a substring pair (s, t) appears is 0.4, the total count for (s, t) is incremented by 0.4).

The algorithm recurses $O(|S| \cdot |T|)$ times, and does up to $O(|S|^2 \cdot |T|^2)$ work in each recursion, giving a time complexity of $O(|S|^3 \cdot |T|^3)$ for each training word pair. This may appear relatively expensive, but we found it to be very fast in practice, and training over several thousand pairs took at most a few dozen seconds for each EM iteration.

3.2 EM Update Rule & Derivation

Let $\theta = \{P_{prod}(t|s)\}$ be our model parameters (all conditional production probabilities), let our training source and target word pairs be $\mathbf{X} = (\mathbf{S}, \mathbf{T})$, and let $n = |\mathbf{X}|$ be the number of such pairs. Additionally, let $\mathbf{A} = \{a\}$

Algorithm 3 Finding $Counts(S, T)$ for Training Pair (S,T)

```
Require: Current production probabilities $P_{prod}(t|s)$
  if $(S, T) \in MemoizationTable$ then
    return $MemoizationTable(T, S)$
  else if $|S| = |T| = 0$ then
    return $(EmptyTable, 1)$ {base case, $P(\text{null}|\text{null}) = 1$}
  else if $|S| = 0 \vee |T| = 0$ then
    return $(EmptyTable, 0)$ {$P(\text{null}|\text{non-null}) = P(\text{null}|\text{non-null}) = 0$}
  else
    $R \Leftarrow 0$
    $C \Leftarrow EmptyTable$
    for $i = 1 \ldots |S| - 1$ do
      for $j = 1 \ldots |T| - 1$ do
        $O \Leftarrow c \cdot P_{prod}(T[1, j] \mid S[1, i])$
        $Q \Leftarrow Counts(S[i+1, |S|], T[j+1, |T|])$ {Recurse}
        $R \Leftarrow R + O \cdot Q[1]$
        $C[S[1, i], T[1, j]] \Leftarrow C[S[1, i], T[1, j]] + R$
        for all $(x, y) \in Q[0]$ do
          $C[x, y] \Leftarrow C[x, y] + Q[0][x, y] \cdot O$
        end for
      end for
    end for
    $MemoizationTable(S, T) \Leftarrow (C, R)$
    return $(C, R)$
  end if
```

be the set of all possible alignments between all word pairs, where $a = (a_1, a_2, ..., a_n)$ and each $a_i \in all_{\mathbf{S}_i, \mathbf{T}_i}$ is a sequence of $|a_i|$ pairs of aligned substrings (u, v) that comprise a valid alignment for $\mathbf{X}_i$. For example, given a word pair $\mathbf{X}_k$ = (edgar,**אדגר**), one possible value for a_k would be "ed-gar $\Rightarrow$ **גר-אד**".

Now, given a previous estimation of our parameters $\theta' = \{P'_{prod}(t|s)\}$, our Q function is:

$$Q(\theta|\theta') = \mathbb{E}_{\mathbf{A}|\mathbf{X},\theta'}[\log \mathcal{L}(\theta|\mathbf{X}, \mathbf{A})]$$

The parameter set θ that maximizes this expected log likelihood is then given by $argmax_\theta Q(\theta|\theta')$, where each production's probability $P_{prod}(t|s)$ is a normalized sum of all the probabilities of all the alignments, weighted by how many times the production occurs in each alignment:

$$\begin{aligned} P_{prod}(t|s) &= \frac{1}{\lambda_s} \sum_{i=1}^{n} \sum_{a_i} \#_{s,t}(a_i) P(a_i|\theta') \\ &= \frac{1}{\lambda_s} \sum_{i=1}^{n} \frac{1}{y'_i} \sum_{a_i} \#_{s,t}(a_i) \prod_{(v|u) \in a_i} c P'_{prod}(v|u) \end{aligned}$$

Where λ_s is a normalizing constant such that, for all substrings s, $\sum_t P(t|s) = 1$, each $y'_i = \sum_{a_i} \prod_{(u,v) \in a_i} c \cdot P'_{prod}(v|u)$ is a per-word pair normalization constant that ensures that $\sum_{a_i} P(a_i|\theta') = 1$, and $\#_{s,t}(a_i)$ is the number of times the production $s \Rightarrow t$ occurs in a particular alignment a_i.

PROOF. To begin, we rewrite the Q function as a sum over all possible assignments to the latent parameters a, weighted by the probability of a and $\mathbf{X}$ given θ'.

$$\begin{aligned} Q(\theta|\theta') &= \mathbb{E}_{\mathbf{A}|\mathbf{X},\theta'}[\log \mathcal{L}(\theta|\mathbf{X}, \mathbf{A})] \\ &= \sum_{a \in \mathbf{A}} \log(P(\mathbf{X}, a|\theta)) \cdot P(\mathbf{X}, a|\theta') \end{aligned}$$

Consider now that any valid alignment $a_i \in a$ implies $\mathbf{X}_i$; for example, the alignment sa-rah $\Rightarrow$ ש-רה implies the word pair (sarah, שרה). Consequently, we have $P(\mathbf{X}, a|\theta) = P(\mathbf{X}|a, \theta) \cdot P(a|\theta) = 1 \cdot P(a|\theta)$, giving us:

$$\begin{aligned}
Q(\theta|\theta') &= \sum_{a \in \mathbf{A}} \log(P(a|\theta))P(a|\theta') \\
&= \sum_{a \in \mathbf{A}} \log(\prod_{i=1}^{n} \frac{1}{y_i} \prod_{(u,v)\in a_i} cP_{prod}(v|u))P(a|\theta') \\
&= \sum_{a \in \mathbf{A}} \log(\prod_{i=1}^{n} \prod_{(u,v)\in a_i} \frac{c}{y_i} P_{prod}(v|u))P(a|\theta') \\
&= \sum_{a \in \mathbf{A}} (\sum_{i=1}^{n} \sum_{(u,v)\in a_i} (\log \frac{c}{y_i} + \log P_{prod}(v|u)))P(a|\theta') \\
&= (\sum_{a \in \mathbf{A}} (\sum_{i=1}^{n} \sum_{(u,v)\in a_i} \log \frac{c}{y_i})P(a|\theta')) \\
&\quad + (\sum_{a \in \mathbf{A}} (\sum_{i=1}^{n} \sum_{(u,v)\in a_i} \log P_{prod}(v|u))P(a|\theta'))
\end{aligned}$$

Where y_i is a normalizing constant ensuring $\sum_{a_i} P(a_i|\theta) = 1$. Notice, however, that now the first term of the Q function contains none of the θ parameters to be maximized, and therefore we can drop this term to obtain a simpler Q' such that $argmax_\theta Q' = argmax_\theta Q$:

$$Q'(\theta|\theta') = \sum_{a \in \mathbf{A}} (\sum_{i=1}^{n} \sum_{(u,v)\in a_i} \log P_{prod}(v|u))P(a|\theta')$$

Observing $P(a|\theta') = \prod_{j=1}^{n} P(a_j|\theta')$ and rewriting the summation over values of the alignment vector a as summations over its components $a_1 \ldots a_n$, we have:

$$\begin{aligned}
&= \sum_{a_1} \sum_{a_2} \cdots \sum_{a_n} (\sum_{i=1}^{n} \sum_{(u,v)\in a_i} \log P_{prod}(v|u)) \cdot \prod_{j=1}^{n} P(a_j|\theta') \\
&= \sum_{i=1}^{n} \sum_{a_1} \sum_{a_2} \cdots \sum_{a_n} \sum_{(u,v)\in a_i} (\log P_{prod}(v|u)) \cdot \prod_{j=1}^{n} P(a_j|\theta')
\end{aligned}$$

Now we can "pull apart" the product $\prod_{j=1}^{n} P(a_j|\theta')$ and reorder our summations as follows:

$$\begin{aligned}
= &\sum_{i=1}^{n} \sum_{a_i} P(a_i|\theta') \sum_{(u,v)\in a_i} (\log P_{prod}(v|u)) \\
&\cdot \sum_{a_1} P(a_1|\theta') \sum_{a_2} P(a_2|\theta') \cdots \sum_{a_{i-1}} P(a_{i-1}|\theta') \\
&\cdot \sum_{a_{i+1}} P(a_{i+1}|\theta') \cdots \sum_{a_n} P(a_n|\theta')
\end{aligned}$$

We simplify this by noting that, for any k, $\sum_{a_k} P(a_k|\theta') = 1$. So we have $\sum_{a_n} P(a_n|\theta') = 1$, and then $\sum_{a_{n-1}} P(a_{n-1}|\theta') \cdot 1 = 1$, and so on, repeatedly "collapsing" these summations until we are left with just:

$$\begin{aligned}
&= \sum_{i=1}^{n} \sum_{a_i} P(a_i|\theta') \sum_{(u,v)\in a_i} \log P_{prod}(v|u) \\
&= \sum_{i=1}^{n} \sum_{a_i} \sum_{(u,v)\in a_i} \log(P_{prod}(v|u))P(a_i|\theta')
\end{aligned}$$

To enforce the constraints that $\sum_v P_{prod}(v|u) = 1$ for all u, we rewrite our equation as the Lagrange function $\mathcal{F}$, with λ_u as our Lagrange multipliers:

$$\begin{aligned}
\mathcal{F} &= \sum_{i=1}^{n} \sum_{a_i} \sum_{(u,v)\in a_i} \log(P_{prod}(v|u))P(a_i|\theta') \\
&\quad - \sum_{u} \lambda_u (\sum_{v} P_{prod}(v|u) - 1)
\end{aligned}$$

Now we are ready to find the parameter set θ that maximizes $Q'(\theta|\theta')$. To do this, we first find the patrial derivative of our Lagrange function with respect to each particular parameter $P_{prod}(t|s) \in \theta$. All of the terms that do not include $P_{prod}(t|s)$ disappear, leaving us with:

$$\frac{\delta \mathcal{F}}{\delta P_{prod}(t|s)} = \sum_{i=1}^{n} \sum_{a_i} \#_{s,t}(a_i) \frac{P(a_i|\theta')}{P_{prod}(v|u)} - \lambda_s$$

Recall that $\#_{s,t}(a_i)$ is the number of times the production $s \Rightarrow t$ occurs in the alignment a_i. Next, we set the partial derivative to 0 to find the maximum:

$$\begin{aligned}
0 &= \sum_{i=1}^{n} \sum_{a_i} \#_{s,t}(a_i) \frac{P(a_i|\theta')}{P_{prod}(t|s)} - \lambda_s \\
\lambda_s &= \sum_{i=1}^{n} \sum_{a_i} \#_{s,t}(a_i) \frac{P(a_i|\theta')}{P_{prod}(t|s)} \\
P_{prod}(t|s) &= \sum_{i=1}^{n} \sum_{a_i} \#_{s,t}(a_i) \frac{P(a_i|\theta')}{\lambda_s} \\
P_{prod}(t|s) &= \frac{1}{\lambda_s} \sum_{i=1}^{n} \sum_{a_i} \#_{s,t}(a_i) P(a_i|\theta')
\end{aligned}$$

Finally, we expand $P(a_i|\theta') = \frac{1}{y'_i} \prod_{(v|u)\in a_i} cP'_{prod}(v|u)$:

$$P_{prod}(t|s) = \frac{1}{\lambda_s} \sum_{i=1}^{n} \frac{1}{y'_i} \sum_{a_i} \#_{s,t}(a_i) \prod_{(v|u)\in a_i} cP'_{prod}(v|u)$$

Which gives us our update rule. $\square$

4. EXPERIMENTAL SETUP AND DATA

We evaluated our model over three preexisting sets of data:

1. A set of 74,396 English-Chinese word pairs taken from a transliteration dictionary [2]

2. A set of English-Russian word pairs with 727 English words and multiple possible Russian transliterations for each, and a total of 50,648 candidate words [7]

3. 550 English-Hebrew word pairs split into a 250 word training set and 300 word test set [4]

4.1 Wikipedia Data

In the case of Russian and Hebrew, we also gathered additional training pairs from Wikipedia with a filtering algorithm. We exploit the fact that many articles have special links to the same article in other languages which can be identified by the language code prefix. For example, Bill Clinton's article contains a link to the Hebrew version of the article as [[he:ביל קלינטון]], easily identified by the "he:" prefix.

1. First, we obtained the English, Russian and Hebrew Wikipedias, available from download.wikipedia.org.

2. Then, in the English Wikipedia, we identified all the articles that were about people, which we took as those articles in the "[year] deaths" or "[year] births" categories and those that had {{persondata}} metadata in their wikitext. The goal in restricting ourselves to people is that person names are usually transliterated between languages, whereas other terms (such as countries, cities, and concepts) are frequently translated.

3. Next, we looked for articles in all Wikipedias that had alternate versions in our languages of interest. In the English Wikipedia, we looked for people articles with Hebrew and Russian versions, and in the Hebrew and Russian Wikipedias we looked for articles that had English versions that were about people. This gave us a list of English-Russian and English-Hebrew pairs of article titles.

4. The title pairs are not directly usable; the number of words in each title may be different, and the ordering may be different. Instead, for each two titles, we look at all the possible word pairs between them; for Bill Clinton, this would be (Bill, ביל), (Bill, קלינטון), (Clinton, ביל) and (Clinton, קלינטון), assigning to each pair 10 points if both titles have exactly one word, 5 points if both titles have the same number of words, and 1 point if the titles have different number of words.

5. We then sum the scores over all the title pairs. If a word pair (x, y) has a score of at least 15 and at least three times the score of any other pair containing either x or y, then we add it to our list of training examples.

Using this method, we were able to obtain 2862 English-Russian and 1166 English-Hebrew word pairs, although some noise did remain. Two common causes were different names for the same person (an English article might use Louis, while a Russian might use Ludwig) or different variants of the same name (Ovid in English versus Ovidius in Hebrew). While this may make this data unsuitable for reliable evaluation, it still performs well as a set of easy-to-obtain training examples.

It is worth observing, however, that this method would have a harder time on languages that traditionally lack spacing between words. We also tried gathering English-Chinese word pairs (which we did not use in our experiments) by extracting only those Chinese titles that had the separating dot • often found in foreign transliterated names, but only obtained 384 examples. This could be rectified with a Chinese word segmentation tool, but a more robust approach would be to first use a small number of examples extracted as above, then train the model, and then use the model to evaluate every possible pairing of the words from the English title and each possible segmentation of the Chinese string, keeping those pairs which are high-probability and effectively bootstrapping the model; we have not yet tried this, however, and leave it to future work.

4.2 Performance Measures

We used two performance measures in our evaluation. The first is accuracy, which is simply the percentage of predicted transliterations that were exactly correct. The second is mean reciprocal rank (MRR). For each test example, we produce an ordered list of candidates (or generated words, in the case of generation) for the source word. The rank R for the example is the position in the list of the correct transliteration, ranging from 1 (the first word) to the number of candidates (the last word). Thus for n test examples we have:

$$MRR = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{R_i}$$

4.3 EM Initialization and Stopping Conditions

We initialized our $P_{prod}(t, s)$ parameters for EM by simply counting the number of training examples each substring pair could align and then normalizing. For instance, if ("rel", רל) could align in 50 training examples, then the count for the pair would be 50; if there were 100 possible alignments for "rel" in total, our initial $P_{prod}(\text{רל}|\text{rel}) = 50/100 = 0.5$.

The other question when running EM is when to stop. In each of the experiments we ran, we randomly selected a portion of the training data as a holdout set, trained the model on the remaining data, and then recorded the performance on the holdout set after each iteration of EM and found the iteration with the highest accuracy and, in the event of a tie, the highest mean reciprocal rank. Then, to evaluate, we trained on the entire training set (including the holdout set) and ran EM for the same number of iterations before performing predictions over the evaluation data.

4.4 Smoothing

For the Hebrew data, especially when we train on just the 250 training examples of [4], data sparsity makes smoothing necessary to ameliorate situations where the transliteration cannot be identified because the source word contains substrings that were never seen during training (for example, if we were to transliterate "Sarah" but had never seen "Sa" or "ar"). In the case of Russian, we have a slightly different problem, in that many of the "transliterations" for the English words in the test set have a noun case suffix attached (such that they are no longer exact transliterations of the English word), e.g. Edward is paired with эдварда. Since the Wikipedia-sourced training data included Russian words without such suffixes, without smoothing the model may not be able to align a final "d" with a "да" because such a mapping is never seen in training, or, if it is, it is (correctly) set to 0 probability after several iterations of EM.

We thus use smoothed conditional probabilities $P^s_{prod}(t|s)$ such that, for any t and s, $P^s_{prod}(t|s) = \max(P_{prod}(t|s), \gamma^{|s|})$, where $\gamma \ll \min P_{prod}(t|s)$ is an arbitrarily low positive constant, even if an $s \Rightarrow t$ production was never observed during training. Note that, since there are in principle an infinite number of possible t for any given s, this would make

Table 1: Chinese Results

Model	Accuracy	MRR
Ours	0.958	0.966
[2]	0.847	0.899

$\sum_t P_{prod}(t|s) = \infty$; we resolve this by observing that, for any finite set of training and test data, the number of possible productions is also finite, and so γ can be set sufficiently low to yield a distribution. In our discovery experiments, we use $\gamma = 10^{-10}$. Smoothing ensures that, where possible, the model will select those candidates that can be produced from the source word without using 0 probability "impossible" productions (since γ is arbitrarily low) but, when no such candidates exist, those that can be produced from the source word with the fewest characters involved in "impossible" productions will be preferred. For example, the model can now identify здварда as the correct transliteration for Edward because now $P^s_{prod}(\text{да}|d)$ is 10^{-10} rather than 0.

4.5 Bi-Directionality

When we evaluate a pair of words S and T, we may not know whether S generated T or whether T generated S. Many English names, for instance, are transliterated from Hebrew (e.g. Abraham, אברהם). Because our conditional model is directional ($P(S|T) \neq P(T|S)$), for our discovery experiments we learn both $P(S|T)$ and $P(T|S)$, and take the probability of T being a transliteration of S (or vice versa) as the geometric mean: $\sqrt{P(T|S)P(S|T)}$. Relative to relying on $P(T|S)$ alone, we found that this bi-directional approach improved accuracy several percent on the test data sets.

5. RESULTS

We compare against three different approaches: [7]'s weakly-supervised model (Russian), [4]'s supervised model (Hebrew) and [2]'s constraint-driven model using Romanization tables and expert knowledge (Hebrew, Russian and Chinese).

5.1 Chinese

As [2] did, we randomly selected 600 evaluation pairs and then added another 100 candidate words, for a total of 700 candidates. We then used the remaining 73,696 pairs to train. Our results, along with [2], are shown in table 1, showing an approximately 11% increase in accuracy and a 0.067 increase in MRR. It must be noted that [2] does not learn from the (vast) numbers of training examples available in a supervised fashion as we do, although it does use expert knowledge in the form of numerous constraints and a Romanization table.

5.2 Russian

For Russian, we used the evaluation data of [7], 727 English words and possible Russian transliterations for each, along with a vast candidate set of 50,648 words. We also considered two additional derived subsets: the subset of 300 words and corresponding 300 candidates also used by [2], and the same subset of 300 words with 1000 corresponding candidates (each English word having, on average, 3.3 Russian transliterations).

A major concern with the Russian data set is that the Russian transliteration lists for each English word are quite noisy; although they usually contain the correct transliteration, they often contain numerous other, incorrect Russian words, or the correct word but with a noun case suffix added (these suffixes add additional phonemes that are independent of both the Russian transliteration and the English source word). The reason for the smaller, 300 words with 300 candidates set used by [2] was primarily because testing with fifty thousand candidates is a large computational hurdle (it takes our model about seven hours), but it also provides an interesting look at how performance improves as the candidate set shrinks. The problem with this subset, though, is that each English word is paired arbitrarily with one of its possible transliterations from the noisy list of possibilities, resulting in many bad pairs. To correct this, we also tried adding in all the transliterations for each English word as candidates for 1000 candidate words total; as we suspected, although the ratio of candidates to "correct" transliterations remains the same (both being increased by a factor of 3.3), performance greatly improves. Our results, and those of the other methods, are show in table 2.

Table 2: Russian Results. Test Set is the number of English words and number of Russian candidates.

Model	Test Set	Accuracy	MRR
Ours	727/50,648	0.846	0.893
[2]	727/50,648	0.73	Not reported
[7]	727/50,648	0.63	Not reported
Ours	300/300	0.930	0.950
[2]	300/300	0.900	0.931
Ours	300/1000	0.980	0.984

Table 3: Hebrew Results. Training Set gives the number of training examples used, where applicable.

Model	Training Set	Accuracy	MRR
Ours (Discovery)	250	0.953	0.970
[2] (Discovery)	N/A	0.850	0.899
[4] (Discovery)	250	Not reported	0.894
Ours (Discovery)	1466	0.987	0.992
Ours (Generation)	1466	0.397	0.505

Over the entire evaluation set (with all 50,648 candidates) we have improved accuracy by 11.6% over [2], although we used 2862 training examples that were automatically collected from Wikipedia as described earlier. Our results on the smaller set of 300 evaluation words with 300 candidates are only 3% higher in accuracy, but this seems to be because the heavy noise in the data imposes a performance ceiling. When we increase the number of candidates to combat this, we gain 5% accuracy to reach 98%.

5.3 Hebrew

For Hebrew we trained on the 250 example training set of [4] as well as a combined set of those 250 examples and the 1166 examples automatically collected from Wikipedia, evaluating over a set of 300 pairs. Here we can directly compare to [4], a supervised method trained on the aforementioned 250 examples, although [2]'s result is in fact slightly better. Since we had a relatively noise-free test set, we also attempted the much harder task of generating the 300 transliterations rather than selecting them from the candidate list. Results are shown in table 3.

Training on the same data, we improve more than 0.075 in MRR compared to [4] and, compared to [2], boost accuracy by more than 10%, and incorporating data from Wikipedia adds another 3.4% to this figure. Our results for generation are much lower, but that is expected–generation requires producing the correct transliteration exactly, a feat that is difficult to achieve with high accuracy without a much larger training set.

6. CONCLUSION

It is clear that word-by-word transliteration has some inherent limitations; as [8] points out, for example, attempting to transliterate Catalina or Katarina from Katakana (which does not distinguish "r" and "l") is impossible without context. More generally, words in any language may have varying pronunciations or multiple valid transliterations into a given language, and many words we wish to "transliterate" fall somewhere between true transliteration and outright translation, such as the names of countries (England and אנגליה). We have nevertheless have demonstrated a model that has both high absolute performance and compares very favorably to several state-of-the-art systems, with accuracies more than 10% higher than those previously obtained, and shown how it can be trained with examples automatically collected from Wikipedia. Also important is that it is a truly language-agnostic "universal transliterator": no language-specific design elements are incorporated, and it is capable of handling quite diverse languages with good results, as our experiments with Chinese, Hebrew and Russian have collectively demonstrated.

Acknowledgments

The authors would like to thank Alex Klementiev, Dan Goldwasser, and Ming-Wei Chang for making their data available to us and providing numerous clarifications, and for Alex's assistance with the Russian language. This research is partly supported by MIAS, a DHS Center for Multimodal Information Access and Synthesis at UIUC.

7. REFERENCES

[1] N. Abduljaleel and L. Larkey. Statistical Transliteration for English-Arabic Cross Language Information Retrieval. In *CIKM*, pages 139–146, 2003.

[2] M. Chang, D. Goldwasser, D. Roth, and Y. Tu. Unsupervised Constraint Driven Learning For Transliteration Discovery. In *Proc. of the Annual Meeting of the North American Association of Computational Linguistics (NAACL)*, May 2009.

[3] A. Dempster, N. Laird, D. Rubin, et al. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[4] D. Goldwasser and D. Roth. Transliteration as Constrained Optimization. In *Proc. of the Conference on Empirical Methods for Natural Language Processing (EMNLP)*, pages 353–362, Oct 2008.

[5] L. Haizhou, Z. Min, and S. Jian. A Joint Source-Channel Model for Machine Transliteration. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics.* Association for Computational Linguistics Morristown, NJ, USA, 2004.

[6] M. Kashani, F. Popowich, and F. Sadat. Automatic Transliteration of Proper Nouns from Arabic to English. The Challenge of Arabic for NLP/MT. In *Proceddings of International conference at the British Computer Society*, 2006.

[7] A. Klementiev and D. Roth. Weakly Supervised Named Entity Transliteration and Discovery from Multilingual Comparable Corpora. In *Proc. of the Annual Meeting of the ACL*, pages USS, TL, ADAPT, July 2006.

[8] K. Knight and J. Graehl. Machine transliteration. *Computational Linguistics*, 24(4):599–612, 1998.

[9] F. J. Och and H. Ney. A Systematic Comparison of Various Statistical Alignment Models. *Computational Linguistics*, 29(1):19–51, 2003.

[10] T. Sherif and G. Kondrak. Bootstrapping a Stochastic Transducer for Arabic-English Transliteration Extraction. In *ACL*, volume 45, page 864, 2007.

[11] B. Stalls and K. Knight. Translating Names and Technical Terms in Arabic Text. In *Proceedings of the COLING/ACL Workshop on Computational Approaches to Semitic Languages*, pages 34–41, 1998.

[12] T. Tao, S. Yoon, A. Fister, R. Sproat, and C. Zhai. Unsupervised Named Entity Transliteration Using Temporal and Phonetic Correlation. *EMNLP*, pages 22–23, 2006.