A Generative Model for Punctuation in Dependency Trees

Xiang Lisa Li* and **Dingquan Wang*** and **Jason Eisner**
Department of Computer Science, Johns Hopkins University
xli150@jhu.edu, {wdd,jason}@cs.jhu.edu

Abstract

Treebanks traditionally treat punctuation marks as ordinary words, but linguists have suggested that a tree's "true" punctuation marks are not observed (Nunberg, 1990). These latent "underlying" marks serve to delimit or separate constituents in the syntax tree. When the tree's yield is rendered as a written sentence, a string rewriting mechanism transduces the underlying marks into "surface" marks, which are part of the observed (surface) string but should not be regarded as part of the tree. We formalize this idea in a generative model of punctuation that admits efficient dynamic programming. We train it without observing the underlying marks, by locally maximizing the incomplete data likelihood (similarly to the EM algorithm). When we use the trained model to reconstruct the tree's underlying punctuation, the results appear plausible across 5 languages, and in particular are consistent with Nunberg's analysis of English. We show that our generative model can be used to beat baselines on punctuation restoration. Also, our reconstruction of a sentence's underlying punctuation lets us appropriately render the surface punctuation (via our trained underlying-to-surface mechanism) when we syntactically transform the sentence.

1 Introduction

Punctuation enriches the expressiveness of written language. When converting from spoken to written language, punctuation indicates pauses or pitches; expresses propositional attitude; and is conventionally associated with certain syntactic constructions such as apposition, parenthesis, quotation, and conjunction.

In this paper, we present a latent-variable model of punctuation usage, inspired by the rule-based approach to English punctuation of Nunberg (1990). Training our model on English data learns rules that are consistent with Nunberg's hand-crafted rules. Our system is automatic, so we use it to obtain rules for Arabic, Chinese, Spanish, and Hindi as well.

Moreover, our rules are stochastic, which allows us to reason probabilistically about ambiguous or missing punctuation. Across the 5 languages, our model predicts surface punctuation better than baselines, as measured both by perplexity (§4) and by accuracy on a punctuation restoration task (§6.1). We also use our model to correct the punctuation of non-native writers of English (§6.2), and to maintain natural punctuation style when syntactically transforming English sentences (§6.3). In principle, our model could also be used within a generative parser, allowing the parser to evaluate whether a candidate tree truly explains the punctuation observed in the input sentence (§8).

Punctuation is interesting In *The Linguistics of Punctuation*, Nunberg (1990) argues that punctuation (in English) is more than a visual counterpart of spoken-language prosody, but forms a linguistic system that involves "interactions of point indicators (i.e. commas, semicolons, colons, periods and dashes)." He proposes that much as in phonology (Chomsky and Halle, 1968), a grammar generates **underlying** punctuation which then transforms into the observed **surface** punctuation.

Consider generating a sentence from a syntactic grammar as follows:

```
Hail the king [, Arthur Pendragon ,]
    [, who wields [ " Excalibur " ] ,] .
```

Although the full tree is not depicted here, some of the constituents are indicated with brackets. In this underlying generated tree, each appositive NP is surrounded by commas. On the surface, however, the two adjacent commas after `Pendragon` will now be collapsed into one, and the final comma will be absorbed into the adjacent period. Furthermore, in American English, the typographic

*Equal contribution.

convention is to move the final punctuation inside the quotation marks. Thus a reader sees only this modified surface form of the sentence:

```
Hail the king, Arthur Pendragon,
    who wields "Excalibur."
```

Note that these modifications are *string* transformations that do not see or change the tree. The resulting surface punctuation marks may be clues to the parse tree, but (contrary to NLP convention) they should not be included as nodes in the parse tree. Only the underlying marks play that role.

Punctuation is meaningful Pang et al. (2002) use question and exclamation marks as clues to sentiment. Similarly, quotation marks may be used to mark titles, quotations, reported speech, or dubious terminology (University of Chicago, 2010). Because of examples like this, methods for determining the similarity or meaning of syntax trees, such as a tree kernel (Agarwal et al., 2011) or a recursive neural network (Tai et al., 2015), should ideally be able to consider where the underlying punctuation marks attach.

Punctuation is helpful Surface punctuation remains correlated with syntactic phrase structure. NLP systems for generating or editing text must be able to deploy surface punctuation as human writers do. Parsers and grammar induction systems benefit from the presence of surface punctuation marks (Jones, 1994; Spitkovsky et al., 2011). It is plausible that they could do better with a linguistically informed model that explains exactly *why* the surface punctuation appears where it does. Patterns of punctuation usage can also help identify the writer's native language (Markov et al., 2018).

Punctuation is neglected Work on syntax and parsing tends to treat punctuation as an afterthought rather than a phenomenon governed by its own linguistic principles. Treebank annotation guidelines for punctuation tend to adopt simple heuristics like "attach to the highest possible node that preserves projectivity" (Bies et al., 1995; Nivre et al., 2018).[1] Many dependency parsing works exclude punctuation from evaluation (Nivre et al., 2007b; Koo and Collins, 2010; Chen and Manning, 2014; Lei et al., 2014; Kiperwasser and Goldberg, 2016), although some others retain punctuation (Nivre et al., 2007a; Goldberg and Elhadad, 2010; Dozat and Manning, 2017).

[1] http://universaldependencies.org/u/dep/punct.html

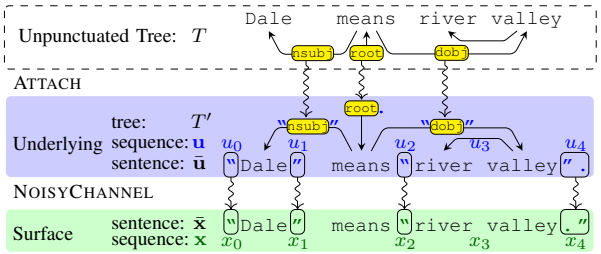


Figure 1: The generative story of a sentence. Given an unpunctuated tree T at top, at each node $w \in T$, the ATTACH process stochastically attaches a left puncteme l and a right puncteme r, which may be empty. The resulting tree T' has underlying punctuation $\mathbf{u}$. Each slot's punctuation $u_i \in \mathbf{u}$ is rewritten to $x_i \in \mathbf{x}$ by NOISYCHANNEL.

In tasks such as word embedding induction (Mikolov et al., 2013; Pennington et al., 2014) and machine translation (Zens et al., 2002), punctuation marks are usually either removed or treated as ordinary words (Řehůřek and Sojka, 2010).

Yet to us, building a parse tree on a *surface* sentence seems as inappropriate as morphologically segmenting a *surface* word. In both cases, one should instead analyze the latent *underlying* form, jointly with recovering that form. For example, the proper segmentation of English `hoping` is not `hop-ing` but `hope-ing` (with underlying `e`), and the proper segmentation of `stopping` is neither `stopp-ing` nor `stop-ping` but `stop-ing` (with only one underlying `p`). Cotterell et al. (2015, 2016) get this right for morphology. We attempt to do the same for punctuation.

2 Formal Model

We propose a probabilistic generative model of sentences (Figure 1):

$$p(\bar{\mathbf{x}}) = \sum_{T,T'} p_{\text{syn}}(T) \cdot p_{\boldsymbol{\theta}}(T' \mid T) \cdot p_{\boldsymbol{\phi}}(\bar{\mathbf{x}} \mid \bar{\mathbf{u}}(T')) \quad (1)$$

First, an *unpunctuated* dependency tree T is stochastically generated by some recursive process p_{syn} (e.g., Eisner, 1996, Model C).[2] Second, each constituent (i.e., dependency subtree) sprouts optional underlying punctuation at its left and right edges, according to a probability distribution $p_{\boldsymbol{\theta}}$ that depends on the constituent's syntactic role (e.g., `dobj` for "direct object"). This *punctuated* tree T' yields the underlying string $\bar{\mathbf{u}} = \bar{\mathbf{u}}(T')$, which is edited by a finite-state noisy channel $p_{\boldsymbol{\phi}}$ to arrive at the surface sentence $\bar{\mathbf{x}}$.

[2] Our model could be easily adapted to work on constituency trees instead.

This third step may alter the sequence of punctuation tokens at each **slot** between words—for example, in §1, collapsing the double comma , , between Pendragon and who. $\mathbf{u}$ and $\mathbf{x}$ denote just the punctuation at the slots of $\bar{\mathbf{u}}$ and $\bar{\mathbf{x}}$ respectively, with u_i and x_i denoting the punctuation token sequences at the i^{th} slot. Thus, the transformation at the i^{th} slot is $u_i \mapsto x_i$.

Since this model is generative, we could train it without any supervision to explain the observed surface string $\bar{\mathbf{x}}$: maximize the likelihood $p(\bar{\mathbf{x}})$ in (1), marginalizing out the possible T, T' values.

In the present paper, however, we exploit known T values (as observed in the "depunctuated" version of a treebank). Because T is observed, we can jointly train $\boldsymbol{\theta}, \boldsymbol{\phi}$ to maximize just

$$p(\mathbf{x} \mid T) = \sum_{T'} p_{\boldsymbol{\theta}}(T' \mid T) \cdot p_{\boldsymbol{\phi}}(\mathbf{x} \mid \mathbf{u}(T')) \quad (2)$$

That is, the p_{syn} model that generated T becomes irrelevant, but we still try to predict what surface punctuation will be added to T. We still marginalize over the underlying punctuation marks $\mathbf{u}$. These are *never observed*, but they must explain the surface punctuation marks $\mathbf{x}$ (§2.2), and they must be explained in turn by the syntax tree T (§2.1). The trained generative model then lets us restore or correct punctuation in new trees T (§6).

2.1 Generating Underlying Punctuation

The ATTACH model characterizes the probability of an underlying punctuated tree T' given its corresponding unpunctuated tree T, which is given by

$$p_{\boldsymbol{\theta}}(T' \mid T) = \prod_{w \in T} p_{\boldsymbol{\theta}}(l_w, r_w \mid w) \quad (3)$$

where $l_w, r_w \in \mathcal{V}$ are the left and right **punctemes** that T' attaches to the tree node w. Each puncteme (Krahn, 2014) in the finite set $\mathcal{V}$ is a string of 0 or more underlying punctuation **tokens**.[3] The probability $p_{\boldsymbol{\theta}}(l, r \mid w)$ is given by a log-linear model

$$p_{\boldsymbol{\theta}}(l, r|w) \propto \begin{cases} \exp \boldsymbol{\theta}^\top \mathbf{f}(l, r, w) & \text{if } (l, r) \in \mathcal{W}_{d(w)} \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

[3] Multi-token punctemes are occasionally useful. For example, the puncteme ... might consist of either 1 or 3 tokens, depending on how the tokenizer works; similarly, the puncteme ?! might consist of 1 or 2 tokens. Also, if a single constituent of T gets surrounded by both parentheses and quotation marks, this gives rise to punctemes ("(and)"). (A better treatment would add the parentheses as a separate puncteme pair at a unary node above the quotation marks, but that would have required T' to introduce this extra node.)

1. Point Absorption	3. Period Absorption
„ ↦ , ,. ↦ . -, ↦ -	.? ↦ ? .! ↦ !
-; ↦ ; ;. ↦ .	abbv. ↦ abbv
2. Quote Transposition	**4. Bracket Absorptions**
", ↦ ," ". ↦ ."	,) ↦) -) ↦) (, ↦ (
	," ↦ " ", ↦ "

Table 1: Some of Nunberg's punctuation interaction rules in English, in priority order. The absorption rules ensure that when there are two adjacent tokens, the "weaker" one is deleted (where the strength ordering is {?, !, (,), ", "} > . > {;, :} > - > ,), except that bracketing tokens such as () and "" do not absorb tokens *outside* the material they bracket.

where $\mathcal{V}$ is the finite set of possible punctemes and $\mathcal{W}_d \subseteq \mathcal{V}^2$ gives the possible puncteme pairs for a node w that has dependency relation $d = d(w)$ to its parent. $\mathcal{V}$ and $\mathcal{W}_d$ are estimated heuristically from the tokenized surface data (§4). $\mathbf{f}(l, r, w)$ is a sparse binary feature vector, and $\boldsymbol{\theta}$ is the corresponding parameter vector of feature weights. The feature templates in Appendix A[4] consider the symmetry between l and r, and their compatibility with (a) the POS tag of w's head word, (b) the dependency paths connecting w to its children and the root of T, (c) the POS tags of the words flanking the slots containing l and r, (d) surface punctuation already added to w's subconstituents.

2.2 From Underlying to Surface

From the tree T', we can read off the sequence of underlying punctuation tokens u_i at each slot i between words. Namely, u_i concatenates the right punctemes of all constituents ending at i with the left punctemes of all constituents starting at i (as illustrated by the examples in §1 and Figure 1). The NOISYCHANNEL model then transduces u_i to a surface token sequence x_i, for each $i = 0, \ldots, n$ independently (where n is the sentence length).

Nunberg's formalism Much like Chomsky and Halle's (1968) phonological grammar of English, Nunberg's (1990) descriptive English punctuation grammar (Table 1) can be viewed computationally as a priority string rewriting system, or **Markov algorithm** (Markov, 1960; Caracciolo di Forino, 1968). The system begins with a token string u. At each step it selects the highest-priority local rewrite rule that can apply, and applies it as far left as possible. When no more rules can apply,

[4] The appendices are included only in this arXiv version, not in the TACL journal.

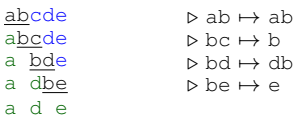


Figure 2: Editing abcde ↦ ade with a sliding window. (When an absorption rule maps 2 tokens to 1, our diagram leaves blank space that is not part of the output string.) At each step, the left-to-right process has already committed to the green tokens as output; has not yet looked at the blue input tokens; and is currently considering how to (further) rewrite the black tokens. The right column shows the chosen edit.

the final state of the string is returned as x.

Simplifying the formalism Markov algorithms are Turing complete. Fortunately, Johnson (1972) noted that in practice, phonological $u \mapsto x$ maps described in this formalism can usually be implemented with finite-state transducers (FSTs).

For computational simplicity, we will formulate our punctuation model as a probabilistic FST (PFST)—a locally normalized left-to-right rewrite model (Cotterell et al., 2014). The probabilities for each language must be learned, using gradient descent. Normally we expect most probabilities to be near 0 or 1, making the PFST nearly deterministic (i.e., close to a subsequential FST). However, permitting low-probability choices remains useful to account for typographical errors, dialectal differences, and free variation in the training corpus.

Our PFST *generates* a surface string, but the invertibility of FSTs will allow us to work backwards when *analyzing* a surface string (§3).

A sliding-window model Instead of having rule priorities, we apply Nunberg-style rules within a 2-token window that slides over u in a single left-to-right pass (Figure 2). Conditioned on the current window contents ab, a single edit is selected stochastically: either $ab \mapsto ab$ (no change), $ab \mapsto b$ (left absorption), $ab \mapsto a$ (right absorption), or $ab \mapsto ba$ (transposition). Then the window slides rightward to cover the next input token, together with the token that is (now) to its left. a and b are always real tokens, never boundary symbols. ϕ specifies the conditional edit probabilities.[5]

These specific edit rules (like Nunberg's) cannot insert new symbols, nor can they delete *all* of the underlying symbols. Thus, surface x_i is a good clue to u_i: all of its tokens must appear underlyingly, and if $x_i = \epsilon$ (the empty string) then $u_i = \epsilon$.

The model can be directly implemented as a PFST (Appendix D[4]) using Cotterell et al.'s (2014) more general PFST construction.

Our single-pass formalism is less expressive than Nunberg's. It greedily makes decisions based on at most one token of right context ("label bias"). It cannot rewrite ' " . ↦ . ' " or " , . ↦ . " because the . is encountered too late to percolate leftward; luckily, though, we can handle such English examples by sliding the window right-to-left instead of left-to-right. We treat the sliding direction as a language-specific parameter.[6]

2.3 Training Objective

Building on equation (2), we train $\boldsymbol{\theta}, \boldsymbol{\phi}$ to locally maximize the regularized conditional log-likelihood

$$\Big(\sum_{\mathbf{x},T} \log p(\mathbf{x} \mid T) - \xi \cdot \mathop{\mathbb{E}}_{T'}[c(T')]^2 \Big) - \varsigma \cdot ||\boldsymbol{\theta}||^2 \tag{5}$$

where the sum is over a training treebank.[7]

The expectation $\mathbb{E}[\cdots]$ is over $T' \sim p(\cdot \mid T, \mathbf{x})$. This *generalized expectation* term provides *posterior regularization* (Mann and McCallum, 2010; Ganchev et al., 2010), by encouraging parameters that reconstruct trees T' that use symmetric punctuation marks in a "typical" way. The function $c(T')$ counts the nodes in T' whose punctemes contain "unmatched" symmetric punctuation tokens: for example,) is "matched" only when it appears in a right puncteme with (at the comparable position in the same constituent's left puncteme. The precise definition is given in Appendix B.[4]

In our development experiments on English, the posterior regularization term was necessary to discover an aesthetically appealing theory of underlying punctuation. When we dropped this term

[5] Rather than learn a separate edit probability distribution for each bigram ab, one could share parameters across bigrams. For example, Table 1's caption says that "stronger" tokens tend to absorb "weaker" ones. A model that incorporated this insight would not have to learn $O(|\Sigma|^2)$ separate absorption probabilities (two per bigram ab), but only $O(|\Sigma|)$ strengths (one per unigram a, which may be regarded as a 1-dimensional embedding of the punctuation token a). We figured that the punctuation vocabulary Σ was small enough (Table 2) that we could manage without the additional complexity of embeddings or other featurization, although this does presumably hurt our generalization to rare bigrams.

[6] We could have handled all languages uniformly by making ≥ 2 passes of the sliding window (via a composition of ≥ 2 PFSTs), with at least one pass in each direction.

[7] In retrospect, there was no good reason to square the $\mathbb{E}_{T'}[c(T')]$ term. However, when we started redoing the experiments, we found the results essentially unchanged.

($\xi = 0$) and simply maximized the ordinary regularized likelihood, we found that the optimization problem was underconstrained: different training runs would arrive at different, rather arbitrary underlying punctemes. For example, one training run learned an ATTACH model that used underlying `"`. to terminate sentences, along with a NOISYCHANNEL model that absorbed the left quotation mark into the period. By encouraging the underlying punctuation to be symmetric, we broke the ties. We also tried making this a hard constraint ($\xi = \infty$), but then the model was unable to explain some of the training sentences at all, giving them probability of 0. For example, `I went to the " special place "` cannot be explained, because `special place` is not a constituent.[8]

3 Inference

In principle, working with the model (1) is straightforward, thanks to the closure properties of formal languages. Provided that p_{syn} can be encoded as a weighted CFG, it can be composed with the weighted tree transducer $p_{\boldsymbol{\theta}}$ and the weighted FST $p_{\boldsymbol{\phi}}$ to yield a new weighted CFG (similarly to Bar-Hillel et al., 1961; Nederhof and Satta, 2003). Under this new grammar, one can recover the optimal T, T' for $\bar{\mathbf{x}}$ by dynamic programming, or sum over T, T' by the inside algorithm to get the likelihood $p(\bar{\mathbf{x}})$. A similar approach was used by Levy (2008) with a different FST noisy channel.

In this paper we assume that T is observed, allowing us to work with equation (2). This cuts the computation time from $O(n^3)$ to $O(n)$.[9] Whereas the inside algorithm for (1) must consider $O(n^2)$ possible constituents of $\bar{\mathbf{x}}$ and $O(n)$ ways of building each, our algorithm for (2) only needs to iterate over the $O(n)$ true constituents of T and the 1 true way of building each. However, it must still consider the $|\mathcal{W}_d|$ puncteme pairs for each constituent.

3.1 Algorithms

Given an input sentence $\bar{\mathbf{x}}$ of length n, our job is to sum over possible trees T' that are consistent with T and $\bar{\mathbf{x}}$, or to find the best such T'. This is roughly a lattice parsing problem—made easier by knowing T. However, the possible $\bar{\mathbf{u}}$ values are characterized not by a lattice but by a *cyclic* WFSA (as $|u_i|$ is unbounded whenever $|x_i| > 0$).

For each slot $0 \le i \le n$, transduce the surface punctuation string x_i by the *inverted* PFST for $p_{\boldsymbol{\phi}}$ to obtain a weighted finite-state automaton (WFSA) that describes *all possible* underlying strings u_i.[10] This WFSA accepts each possible u_i with weight $p_{\boldsymbol{\phi}}(x_i \mid u_i)$. If it has N_i states, we can represent it (Berstel and Reutenauer, 1988) with a family of sparse weight matrices $\mathbf{M}_i(v) \in \mathbb{R}^{N_i \times N_i}$, whose element at row s and column t is the weight of the $s \to t$ arc labeled with v, or 0 if there is no such arc. Additional vectors $\boldsymbol{\lambda}_i, \boldsymbol{\rho}_i \in \mathbb{R}^{N_i}$ specify the initial and final weights. ($\boldsymbol{\lambda}_i$ is one-hot if the PFST has a single initial state, of weight 1.)

For any puncteme l (or r) in $\mathcal{V}$, we define $\mathbf{M}_i(l) = \mathbf{M}_i(l_1)\mathbf{M}_i(l_2)\cdots\mathbf{M}_i(l_{|l|})$, a product over the 0 or more tokens in l. This gives the total weight of all $s \to^* t$ WFSA paths labeled with l.

Algorithm 1 The algorithm for scoring a given $(T, \mathbf{x})$ pair. The code in blue is used during training to get the posterior regularization term in (5).

Input: $T, \mathbf{x}$ ▷ *Training pair (omits T', $\mathbf{u}$)*
Output: $p(\mathbf{x} \mid T)$, $\mathbb{E}[c(T')]$

procedure TOTALSCORE($T, \mathbf{x}$)
  for $i = 1$ **to** n **do**
    compute WFSA $(\mathbf{M}_i, \boldsymbol{\lambda}_i, \boldsymbol{\rho}_i)$
  $E \leftarrow 0$ ▷ *exp. count of unmatched punctemes*
  procedure IN(w) ▷ $w \in T$
    $i, k \leftarrow$ slots at left, right of w constit
    $j \leftarrow$ slot at right of w headword
    $\mathbf{M}_{\text{left}} \leftarrow (\prod_{w' \in \text{leftkids}(w)} \text{IN}(w'))\boldsymbol{\rho}_{j-1}$
    $\mathbf{M}_{\text{right}} \leftarrow \boldsymbol{\lambda}_j^\top (\prod_{w' \in \text{rightkids}(w)} \text{IN}(w'))$
    $\mathbf{M}' \leftarrow \mathbf{M}_{\text{left}} \cdot 1 \cdot \mathbf{M}_{\text{right}}$ ▷ $\mathbb{R}^{N_j \times 1}, \mathbb{R}^{1 \times N_j}$
    $\mathbf{M} \leftarrow \mathbf{0}$ ▷ $\mathbb{R}^{N_i \times N_k}$
    for $(l, r) \in \mathcal{W}_{d(w)}$ **do**
      $p \leftarrow p_{\boldsymbol{\theta}}(l, r \mid w)$
      $\mathbf{M} \leftarrow \mathbf{M} + p \cdot \mathbf{M}_i(l)\mathbf{M}'\mathbf{M}_k(r)$
      $E \leftarrow E + p \cdot \mathbb{1}_{l,r}$ have unmatched punc
    return $\mathbf{M}$ ▷ $\mathbb{R}^{N_i \times N_k}$
  $M_{\text{root}} \leftarrow \text{IN}(\text{root}(T))$
  return $\boldsymbol{\lambda}_0^\top M_{\text{root}} \boldsymbol{\rho}_n$, E ▷ $\mathbb{R}, \mathbb{R}$

[8] Recall that the NOISYCHANNEL model family (§2.2) requires the surface `"` before `special` to appear underlyingly, and also requires the surface ϵ after `special` to be empty underlyingly. These hard constraints clash with the $\xi = \infty$ hard constraint that the punctuation around `special` must be balanced. The surface `"` after `place` causes a similar problem: no edge can generate the matching underlying `"`.

[9] We do $O(n)$ multiplications of $N \times N$ matrices where $N = O(\#\text{ of punc types} \cdot \max \#\text{ of punc tokens per slot})$.

[10] Constructively, compose the u-to-x PFST (from the end of §2.2) with a straight-line FSA accepting only x_i, and project the resulting WFST to its input tape (Pereira and Riley, 1996), as explained at the end of Appendix D.

The subprocedure in Algorithm 1 essentially extends this to obtain a new matrix $\textsc{In}(w) \in \mathbb{R}^{N_i \times N_k}$, where the subtree rooted at w stretches from slot i to slot k. Its element $\textsc{In}(w)_{st}$ gives the total weight of all **extended paths** in the $\bar{\mathbf{u}}$ WFSA from state s at slot i to state t at slot k. An extended path is defined by a choice of underlying punctemes at w and all its descendants. These punctemes determine an s-to-final path at i, then initial-to-final paths at $i+1$ through $k-1$, then an initial-to-t path at k. The weight of the extended path is the product of all the WFSA weights on these paths (which correspond to transition probabilities in $p_{\boldsymbol{\phi}}$ PFST) times the probability of the choice of punctemes (from $p_{\boldsymbol{\theta}}$).

This inside algorithm computes quantities needed for training (§2.3). Useful variants arise via well-known methods for weighted derivation forests (Berstel and Reutenauer, 1988; Goodman, 1999; Li and Eisner, 2009; Eisner, 2016).

Specifically, to modify Algorithm 1 to *maximize* over T' values (§§6.2–6.3) instead of summing over them, we switch to the **derivation semiring** (Goodman, 1999), as follows. Whereas $\textsc{In}(w)_{st}$ used to store the *total* weight of all extended paths from state s at slot i to state t at slot j, now it will store the weight of the *best* such extended path. It will also store that extended path's choice of underlying punctemes, in the form of a puncteme-annotated version of the subtree of T that is rooted at w. This is a potential subtree of T'.

Thus, each element of $\textsc{In}(w)$ has the form (r, D) where $r \in \mathbb{R}$ and D is a tree. We define addition and multiplication over such pairs:

$$(r, D) + (r', D') = \begin{cases} (r, D) & \text{if } r > r' \\ (r', D') & \text{otherwise} \end{cases} \quad (6)$$

$$(r, D) \cdot (r', D') = (rr', DD') \quad (7)$$

where DD' denotes an ordered combination of two trees. Matrix products $\mathbf{UV}$ and scalar-matrix products $p \cdot \mathbf{V}$ are defined in terms of element addition and multiplication as usual:

$$(\mathbf{UV})_{st} = \textstyle\sum_r \mathbf{U}_{sr} \cdot \mathbf{V}_{rt} \quad (8)$$

$$(p \cdot \mathbf{V})_{st} = p \cdot \mathbf{V}_{st} \quad (9)$$

What is DD'? For presentational purposes, it is convenient to represent a punctuated dependency tree as a bracketed string. For example, the underlying tree T' in Figure 1 would be `[ [" Dale "] means [" [ river ] valley "] ]` where the words correspond to nodes of T. In this case, we can represent every D as a *partial* bracketed string and define DD' by string concatenation. This presentation ensures that multiplication (7) is a complete and associative (though not commutative) operation, as in any semiring. As base cases, each real-valued element of $\mathbf{M}_i(l)$ or $\mathbf{M}_k(r)$ is now paired with the string $[\,l$ or $r\,]$ respectively,[11] and the real number 1 at line 10 is paired with the string w. The real-valued elements of the $\boldsymbol{\lambda}_i$ and $\boldsymbol{\rho}_i$ vectors and the $\mathbf{0}$ matrix at line 11 are paired with the empty string ϵ, as is the real number p at line 13.

In practice, the D strings that appear within the matrix $\mathbf{M}$ of Algorithm 1 will always represent complete punctuated trees. Thus, they can actually be represented in memory as such, and different trees may share subtrees for efficiency (using pointers). The product in line 10 constructs a matrix of trees with root w and differing sequences of left/right children, while the product in line 14 annotates those trees with punctemes l, r.

To sample a possible T' from the derivation forest in proportion to its probability (§6.1), we use the same algorithm but replace equation (6) with

$$(r, D) + (r', D') = \begin{cases} (r + r', D) & \text{if } u < \frac{r}{r+r'} \\ (r + r', D') & \text{otherwise} \end{cases}$$

with $u \sim \text{Uniform}(0, 1)$ being a random number.

3.2 Optimization

Having computed the objective (5), we find the gradient via automatic differentiation, and optimize $\boldsymbol{\theta}, \boldsymbol{\phi}$ via Adam (Kingma and Ba, 2014)—a variant of stochastic gradient decent—with learning rate 0.07, batchsize 5, sentence per epoch 400, and L2 regularization. (These hyperparameters, along with the regularization coefficients ς and ξ from equation (5), were tuned on dev data (§4) for each language respectively.) We train the punctuation model for 30 epochs. The initial NoisyChannel parameters ($\boldsymbol{\phi}$) are drawn from $\mathcal{N}(0, 1)$, and the initial Attach parameters ($\boldsymbol{\theta}$) are drawn from $\mathcal{N}(0, 1)$ (with one minor exception described in Appendix A).

[11] We still construct the real matrix $\mathbf{M}_i(l)$ by *ordinary* matrix multiplication before pairing its elements with strings. This involves summation of real numbers: each element of the resulting real matrix is a marginal probability, which sums over possible PFST paths (edit sequences) that could map the underlying puncteme l to a certain substring of the surface slot x_i. Similarly for $\mathbf{M}_k(r)$.

4 Intrinsic Evaluation of the Model

Data. Throughout §§4–6, we will examine the punctuation model on a subset of the Universal Dependencies (UD) version 1.4 (Nivre et al., 2016)—a collection of dependency treebanks across 47 languages with unified POS-tag and dependency label sets. Each treebank has designated training, development, and test portions. We experiment on Arabic, English, Chinese, Hindi, and Spanish (Table 2)—languages with diverse punctuation vocabularies and punctuation interaction rules, not to mention script directionality. For each treebank, we use the tokenization provided by UD, and take the punctuation tokens (which may be multi-character, such as `...`) to be the tokens with the `PUNCT` tag. We replace each straight double quotation mark `"` with either ‶ or ″ as appropriate, and similarly for single quotation marks.[12] We split each non-punctuation token that ends in `.` (such as `etc.`) into a shorter non-punctuation token (`etc`) followed by a special punctuation token called the "abbreviation dot" (which is distinct from a period). We prepend a special punctuation mark ˆ to every sentence $\bar{\mathbf{x}}$, which can serve to absorb an initial comma, for example.[13] We then replace each token with the special symbol `UNK` if its type appeared fewer than 5 times in the training portion. This gives the surface sentences.

To estimate the vocabulary $\mathcal{V}$ of *underlying* punctemes, we simply collect all *surface* token sequences x_i that appear at any slot in the training portion of the processed treebank. This is a generous estimate. Similarly, we estimate $\mathcal{W}_d$ (§2.1) as all pairs $(l, r) \in \mathcal{V}^2$ that flank any d constituent.

Recall that our model generates surface punctuation given an unpunctuated dependency tree. We train it on each of the 5 languages independently. We evaluate on conditional perplexity, which will be low if the trained model successfully assigns a high probability to the actual surface punctuation in a held-out corpus of the same language.

Baselines. We compare our model against three baselines to show that its complexity is necessary. Our first baseline is an ablation study that does not use latent underlying punctuation, but generates the surface punctuation directly from the tree. (To implement this, we fix the parameters of the noisy channel so that the surface punctuation equals the underlying with probability 1.) If our full model performs significantly better, it will demonstrate the importance of a distinct underlying layer.

Our other two baselines ignore the tree structure, so if our full model performs significantly better, it will demonstrate that conditioning on explicit syntactic structure is useful. These baselines are based on previously published approaches that reduce the problem to tagging: Xu et al. (2016) use a BiLSTM-CRF tagger with bigram topology; Tilk and Alumäe (2016) use a BiGRU tagger with attention. In both approaches, the model is trained to tag each slot i with the correct string $x_i \in \mathcal{V}^*$ (possibly ϵ or ˆ). These are discriminative probabilistic models (in contrast to our generative one). Each gives a probability distribution over the taggings (conditioned on the unpunctuated sentence), so we can evaluate their perplexity.[14]

Language	Treebank	#Token	%Punct	#Omit	#Type
Arabic	ar	282K	7.9	255	18
Chinese	zh	123K	13.8	3	23
English	en	255K	11.7	40	35
	en_esl	97.7K	9.8	2	16
Hindi	hi	352K	6.7	21	15
Spanish	es_ancora	560K	11.7	25	16

Table 2: Statistics of our datasets. "Treebank" is the UD treebank identifier, "#Token" is the number of tokens, "%Punct" is the percentage of punctuation tokens, "#Omit" is the small number of sentences containing non-leaf punctuation tokens (see footnote 19), and "#Type" is the number of punctuation types after preprocessing. (Recall from §4 that preprocessing distinguishes between left and right quotation mark types, and between abbreviation dot and period dot types.)

Results. As shown in Table 3, our full model beats the baselines in perplexity in all 5 languages. Also, in 4 of 5 languages, allowing a trained NOISYCHANNEL (rather than the identity map) significantly improves the perplexity.

5 Analysis of the Learned Grammar

5.1 Rules Learned from the Noisy Channel

We study our learned probability distribution over noisy channel rules ($ab \mapsto b$, $ab \mapsto a$, $ab \mapsto ab$,

[12] For en and en_esl, ‶ and ″ are distinguished by language-specific part-of-speech tags. For the other 4 languages, we identify two `"` dependents of the same head word, replacing the left one with ‶ and the right one with ″.

[13] For symmetry, we should also have added a final mark.

[14] These methods learn word embeddings that optimize conditional log-likelihood on the punctuation restoration training data. They might do better if these embeddings were shared with other tasks, as multi-task learning might lead them to discover syntactic categories of words.

	Attn.	CRF	ATTACH	+NC	DIR
Arabic	1.4676	1.3016	1.2230	**1.1526**	L
Chinese	1.6850	1.4436	1.1921	**1.1464**	L
English	1.5737	**1.5247**	1.5636	**1.4276**	R
Hindi	1.1201	1.1032	1.0630	**1.0598**	L
Spanish	1.4397	**1.3198**	**1.2364**	**1.2103**	R

Table 3: Results of the conditional perplexity experiment (§4), reported as perplexity per punctuation slot, where an unpunctuated sentence of n words has $n+1$ slots. Column "Attn." is the BiGRU tagger with attention, and "CRF" stands for the BiLSTM-CRF tagger. "ATTACH" is the ablated version of our model where surface punctuation is directly attached to the nodes. Our full model "+NC" adds NOISYCHANNEL to transduce the attached punctuation into surface punctuation. DIR is the learned direction (§2.2) of our full model's noisy channel PFST: Left-to-right or Right-to-left. Our models are given oracle parse trees T. The best perplexity is **boldfaced**, along with all results that are not significantly worse (paired permutation test, $p < 0.05$).

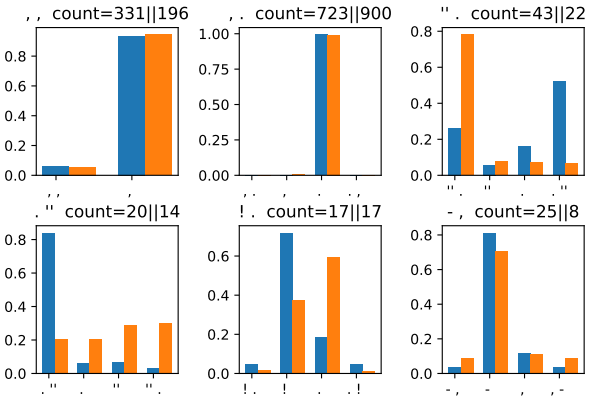


Figure 3: Rewrite probabilities learned for English, averaged over the last 4 epochs on en treebank (blue bars) or en_esl treebank (orange bars). The header above each figure is the underlying punctuation string (input to NOISYCHANNEL). The two counts in the figure headers are the number of occurrences of the underlying punctuation strings in the 1-best reconstruction of underlying punctuation sequences (by Algorithm 1) respectively in the en and en_esl treebank. Each bar represents one surface punctuation string (output of NOISYCHANNEL), its height giving the probability.

$ab \mapsto ba$) for English. The probability distributions corresponding to six of Nunberg's English rules are shown in Figure 3. By comparing the orange and blue bars, observe that the model trained on the en_cesl treebank learned different quotation rules from the one trained on the en treebank. This is because en_cesl follows British style, whereas en has American-style quote transposition.[15]

We now focus on the model learned from the en treebank. Nunberg's rules are deterministic, and our noisy channel indeed learned low-entropy rules, in the sense that for an input ab with underlying count ≥ 25,[16] at least one of the possible outputs (a, b, ab or ba) always has probability > 0.75. The one exception is $" . \mapsto . "$ for which the argmax output has probability ≈ 0.5, because writers do not apply this quote transposition rule consistently. As shown by the blue bars in Figure 3, the high-probability transduction rules are consistent with Nunberg's hand-crafted deterministic grammar in Table 1.

Our system has high precision when we look at the confident rules. Of the 24 learned edits with conditional probability > 0.75, Nunberg lists 20.

Our system also has good recall. Nunberg's hand-crafted schemata consider 16 punctuation types and generate a total of 192 edit rules, including the specimens in Table 1. That is, of the $16^2 = 256$ *possible* underlying punctuation bigrams ab, $\frac{3}{4}$ are supposed to undergo absorption or transposition. Our method achieves fairly high recall, in the sense that when Nunberg proposes $ab \mapsto \gamma$, our learned $p(\gamma \mid ab)$ usually ranks highly among all probabilities of the form $p(\gamma' \mid ab)$. 75 of Nunberg's rules got rank 1, 48 got rank 2, and the remaining 69 got rank > 2. The mean reciprocal rank was 0.621. Recall is quite high when we restrict to those Nunberg rules $ab \mapsto \gamma$ for which our model is confident how to rewrite ab, in the sense that some $p(\gamma' \mid ab) > 0.5$. (This tends to eliminate rare ab: see footnote 5.) Of these 55 Nunberg rules, 38 rules got rank 1, 15 got rank 2, and only 2 got rank worse than 2. The mean reciprocal rank was 0.836.

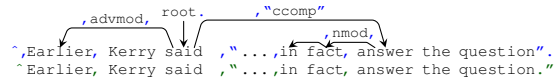
¿What about Spanish? Spanish uses inverted question marks ¿ and exclamation marks ¡, which form symmetric pairs with the regular question marks and exclamation marks. If we try to extrapolate to Spanish from Nunberg's English formalization, the English mark most analogous to ¿ is (. Our learned noisy channel for Spanish (not graphed here) includes the high-probability rules , ¿ $\mapsto$, ¿ and : ¿ $\mapsto$: ¿ and ¿ , $\mapsto$ ¿ which match Nunberg's treatment of (in English.

[15] American style places commas and periods inside the quotation marks, even if they are not logically in the quote. British style (more sensibly) places unquoted periods and commas in their logical place, sometimes outside the quotation marks if they are not part of the quote.

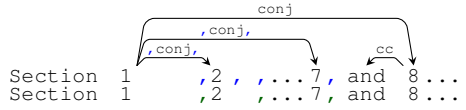
[16] For rarer underlying pairs ab, the estimated distributions sometimes have higher entropy due to undertraining.

5.2 Attachment Model

What does our model learn about how dependency relations are marked by underlying punctuation?



The above example[17] illustrates the use of specific puncteme pairs to set off the `advmod`, `ccomp`, and `nmod` relations. Notice that `said` takes a complement (`ccomp`) that is symmetrically quoted but also left delimited by a comma, which is indeed how direct speech is punctuated in English. This example also illustrates quotation transposition. The top five relations that are most likely to generate symmetric punctemes and their top (l, r) pairs are shown in Table 4.



The above example[18] shows how our model handles commas in conjunctions of 2 or more phrases. UD format dictates that each conjunct after the first is attached by the `conj` relation. As shown above, each such conjunct is surrounded by underlying commas (via the `N.,.,.conj` feature from Appendix A), except for the one that bears the conjunction `and` (via an even stronger weight on the $\texttt{C.}\epsilon\texttt{.}\epsilon\texttt{.}\overrightarrow{\texttt{conj}}\texttt{.cc}$ feature). Our learned feature weights indeed yield $p(\ell = \epsilon, r = \epsilon) > 0.5$ for the final conjunct in this example. Some writers omit the "Oxford comma" before the conjunction: this style can be achieved simply by changing "surrounded" to "preceded" (that is, changing the `N` feature to $\texttt{N.,.}\epsilon\texttt{.conj}$).

6 Performance on Extrinsic Tasks

We evaluate the trained punctuation model by using it in the following three tasks.

6.1 Punctuation Restoration

In this task, we are given a depunctuated sentence $\bar{\mathbf{d}}$[19] and must restore its (surface) punctuation. Our model supposes that the observed punctuated sentence $\bar{\mathbf{x}}$ would have arisen via the generative process (1). Thus, we try to find T, T', and $\bar{\mathbf{x}}$ that are consistent with $\bar{\mathbf{d}}$ (a partial observation of $\bar{\mathbf{x}}$).

parataxis 2.38		appos 2.29		list 1.33		advcl 0.77		ccomp 0.53	
, ,	**26.8**	, ,	**18.8**	ϵ ϵ	60.0	ϵ ϵ	73.8	ϵ ϵ	90.8
ϵ ϵ	20.1	: ϵ	18.1	, ,	**22.3**	, ,	**21.2**	`` ''	**2.4**
( )	**13.0**	– ϵ	15.9	, ϵ	5.3	ϵ ,	3.1	, ,	**2.4**
– ϵ	9.7	ϵ ϵ	14.4	< >	**3.0**	( )	**0.74**	:`` ''	0.9
: ϵ	8.1	( )	**13.1**	( )	**3.0**	ϵ –	0.21	`` ,''	0.8

Table 4: The top 5 relations that are most likely to generate symmetric punctemes, the entropy of their puncteme pair (row 2), and their top 5 puncteme pairs (rows 3–7) with their probabilities shown as percentages. The symmetric punctemes are in boldface.

The first step is to reconstruct T from $\bar{\mathbf{d}}$. This initial parsing step is intended to choose the T that maximizes $p_{\text{syn}}(T \mid \bar{\mathbf{d}})$.[20] This step depends only on p_{syn} and not on our punctuation model $(p_{\boldsymbol{\theta}}, p_{\boldsymbol{\phi}})$. In practice, we choose T via a dependency parser that has been trained on an unpunctuated treebank with examples of the form $(\bar{\mathbf{d}}, T)$.[21]

Equation (2) now defines a distribution over $(T', \mathbf{x})$ given this T. To obtain a single prediction for $\mathbf{x}$, we adopt the minimum Bayes risk (MBR) approach of choosing surface punctuation $\hat{\mathbf{x}}$ that minimizes the expected loss with respect to the unknown truth $\mathbf{x}^*$. Our loss function is the total edit distance over all slots (where edits operate on punctuation tokens). Finding $\hat{\mathbf{x}}$ exactly would be intractable, so we use a sampling-based approximation and draw $m = 1000$ samples from the posterior distribution over $(T', \mathbf{x})$. We then define

$$\hat{\mathbf{x}} = \operatorname*{argmin}_{\mathbf{x} \in S(T)} \sum_{\mathbf{x}^* \in S(T)} \hat{p}(\mathbf{x}^* | T) \cdot \text{loss}(\mathbf{x}, \mathbf{x}^*) \quad (10)$$

where $S(T)$ is the set of unique $\mathbf{x}$ values in the sample and $\hat{p}$ is the empirical distribution given by the sample. This can be evaluated in $O(m^2)$ time.

We evaluate on Arabic, English, Chinese, Hindi, and Spanish. For each language, we train both the parser and the punctuation model on the training split of that UD treebank (§4), and evaluate on held-out data. We compare to the BiLSTM-CRF baseline in §4 (Xu et al., 2016).[22]

[17] [en] `Earlier, Kerry said, "Just because you get an honorable discharge does not, in fact, answer that question."`

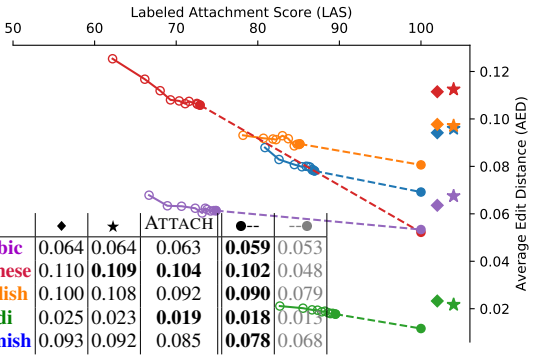
[18] [en] `Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.`

[19] To depunctuate a treebank sentence, we remove all tokens with POS-tag `PUNCT` or dependency relation `punct`. These are almost always leaves; else we omit the sentence.

[20] Ideally, rather than maximize, one would integrate over possible trees T, in practice by sampling many values T_k from $p_{\text{syn}}(\cdot \mid \bar{\mathbf{u}})$ and replacing $S(T)$ in (10) with $\bigcup_k S(T_k)$.

[21] Specifically, the Yara parser (Rasooli and Tetreault, 2015), a fast non-probabilistic transition-based parser that uses rich non-local features (Zhang and Nivre, 2011).

[22] We copied their architecture exactly but re-tuned the hyperparameters on our data. We also tried tripling the amount of training data by adding unannotated sentences (provided



	◆	★	ATTACH	●--	--●
Arabic	0.064	0.064	0.063	**0.059**	0.053
Chinese	0.110	**0.109**	**0.104**	**0.102**	0.048
English	0.100	0.108	0.092	**0.090**	0.079
Hindi	0.025	0.023	**0.019**	**0.018**	0.015
Spanish	0.093	0.092	0.085	**0.078**	0.068

Figure 4: Edit distance per slot (which we call average edit distance, or AED) for each of the 5 corpora. Lower is better. The table gives the final AED on the **test** data. Its first 3 columns show the baseline methods just as in Table 3: the trivial deterministic method, the BiLSTM-CRF, and the ATTACH ablation baseline that attaches the surface punctuation directly to the tree. Column 4 is our method that incorporates a noisy channel, and column 5 (in gray) is our method using oracle (gold) trees. We boldface the best non-oracle result as well as all that are not significantly worse (paired permutation test, $p < 0.05$). The curves show how our method's AED (on **dev** data) varies with the labeled attachment score (LAS) of the trees, where --● at $x = 100$ uses the oracle (gold) trees, ●-- at $x < 100$ uses trees from our parser trained on 100% of the training data, and the ○-- points at $x \ll 100$ use increasingly worse parsers. The ◆ and ★ at the right of the graph show the AED of the trivial deterministic baseline and the BiLSTM-CRF baseline, which do not use trees.

We also compare to a "trivial" deterministic baseline, which merely places a period at the end of the sentence (or a "|" in the case of Hindi) and adds no other punctuation. Because most slots do not in fact have punctuation, the trivial baseline already does very well; to improve on it, we must fix its errors without introducing new ones.

Our final comparison on test data is shown in the table in Figure 4. On all 5 languages, our method beats (usually significantly) its 3 competitors: the trivial deterministic baseline, the BiLSTM-CRF, and the ablated version of our model (ATTACH) that omits the noisy channel.

Of course, the success of our method depends on the quality of the parse trees T (which is particularly low for Chinese and Arabic). The graph in Figure 4 explores this relationship, by evaluating (on dev data) with noisier trees obtained from parsers that were variously trained on only the first 10%, 20%, ... of the training data. On all 5 languages, provided that the trees are at least 75% correct, our punctuation model beats both the trivial baseline and the BiLSTM-CRF (which do not use trees). It also beats the ATTACH ablation baseline at all levels of tree accuracy (these curves are omitted from the graph to avoid clutter). In all languages, better parses give better performance, and gold trees yield the best results.

6.2 Punctuation Correction

Our next goal is to correct punctuation errors in a learner corpus. Each sentence is drawn from the Cambridge Learner Corpus treebanks, which provide original (en_esl) and corrected (en_cesl) sentences. All kinds of errors are corrected, such as syntax errors, but we use only the 30% of sentences whose depunctuated trees T are isomorphic between en_esl and en_cesl. These en_cesl trees may correct word and/or punctuation errors in en_esl, as we wish to do automatically.

We assume that an English learner can make mistakes in both the attachment and the noisy channel steps. A common attachment mistake is the failure to surround a non-restrictive relative clause with commas. In the noisy channel step, mistakes in quote transposition are common.

Correction model. Based on the assumption about the two error sources, we develop a discriminative model for this task. Let $\bar{\mathbf{x}}_e$ denote the full input sentence, and let $\mathbf{x}_e$ and $\mathbf{x}_c$ denote the input (possibly errorful) and output (corrected) punctuation sequences. We model $p(\mathbf{x}_c \mid \bar{\mathbf{x}}_e) = \sum_T \sum_{T'_c} p_{\text{syn}}(T \mid \bar{\mathbf{x}}_e) \cdot p_{\boldsymbol{\theta}}(T'_c \mid T, \mathbf{x}_e) \cdot p_{\boldsymbol{\phi}}(\mathbf{x}_c \mid T'_c)$. Here T is the depunctuated parse tree, T'_c is the corrected underlying tree, T'_e is the error underlying tree, and we assume $p_{\boldsymbol{\theta}}(T'_c \mid T, \mathbf{x}_e) = \sum_{T'_e} p(T'_e \mid T, \mathbf{x}_e) \cdot p_{\boldsymbol{\theta}}(T'_c \mid T'_e)$.

In practice we use a 1-best pipeline rather than summing. Our first step is to reconstruct T from the error sentence $\bar{\mathbf{x}}_e$. We choose T that maximizes $p_{\text{syn}}(T \mid \bar{\mathbf{x}}_e)$ from a dependency parser trained on en_esl treebank examples $(\bar{\mathbf{x}}_e, T)$. The second step is to reconstruct T'_e based on our punctuation model trained on en_esl. We choose T'_e that maximizes $p(T'_e \mid T, \mathbf{x}_e)$. We then reconstruct T'_c by

along with the original annotated sentences by Ginter et al. (2017)), taking advantage of the fact that the BiLSTM-CRF does not require its training sentences to be annotated with trees. However, this actually hurt performance slightly, perhaps because the additional sentences were out-of-domain. We also tried the BiGRU-with-attention architecture of Tilk and Alumäe (2016), but it was also weaker than the BiLSTM-CRF (just as in Table 3). We omit all these results from Figure 4 to reduce clutter.

	◆	★	●--	parsed	gold	★-corr
AED	0.052	0.051	0.047	0.034	0.033	**0.005**
$F_{0.5}$	0.779	0.787	0.827	0.876	0.881	**0.984**

Table 5: AED and $F_{0.5}$ results on the test split of English-ESL data. Lower AED is better; higher $F_{0.5}$ is better. The first three columns (markers correspond to Figure 4) are the punctuation restoration baselines, which ignore the input punctuation. The fourth and fifth columns are our correction models, which use parsed and gold trees. The final column is the BiLSTM-CRF model tailored for the punctuation correction task.

$$p(T'_c \mid T'_e) = \prod_{w_e \in T'_e} p(l, r \mid w_e) \quad (11)$$

where w_e is the node in T'_e, and $p(l, r \mid w_e)$ is a similar log-linear model to equation (4) with additional features (Appendix C[4]) which look at w_e.

Finally, we reconstruct $\mathbf{x}_c$ based on the noisy channel $p_{\phi}(\mathbf{x}_c \mid T'_c)$ in §2.2. During training, ϕ is regularized to be close to the noisy channel parameters in the punctuation model trained on en_cesl.

We use the same MBR decoder as in §6.1 to choose the best action. We evaluate using AED as in §6.1. As a second metric, we use the script from the CoNLL 2014 Shared Task on Grammatical Error Correction (Ng et al., 2014): it computes the $F_{0.5}$-measure of the set of edits found by the system, relative to the true set of edits.

As shown in Table 5, our method achieves better performance than the punctuation restoration baselines (which ignore input punctuation). On the other hand, it is soundly beaten by a new BiLSTM-CRF that we trained specifically for the task of punctuation correction. This is the same as the BiLSTM-CRF in the previous section, except that the BiLSTM now reads a *punctuated* input sentence (with possibly erroneous punctuation). To be precise, at step $0 \leq i \leq n$, the BiLSTM reads a concatenation of the embedding of word i (or BOS if $i = 0$) with an embedding of the punctuation token sequence x_i. The BiLSTM-CRF wins because it is a discriminative model tailored for this task: the BiLSTM can extract arbitrary contextual features of slot i that are correlated with whether x_i is correct in context.

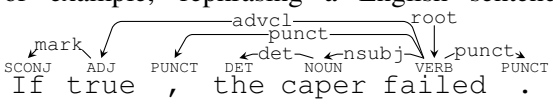
6.3 Sentential Rephrasing

We suspect that syntactic transformations on a sentence should often preserve the underlying punctuation attached to its tree. The surface punctuation can then be regenerated from the transformed tree. Such transformations include edits that are suggested by a writing assistance tool (Heidorn, 2000), or subtree deletions in compressive summarization (Knight and Marcu, 2002).

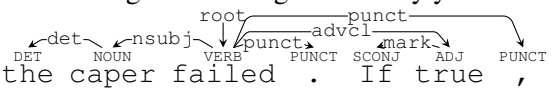
For our experiment, we evaluate an interesting case of syntactic transformation. Wang and Eisner (2016) consider a systematic rephrasing procedure by rearranging the order of dependent subtrees within a UD treebank, in order to synthesize new languages with different word order that can then be used to help train multi-lingual systems (i.e., data augmentation with synthetic data).

As Wang and Eisner acknowledge (2016, footnote 9), their permutations treat surface punctuation tokens like ordinary words, which can result in synthetic sentences whose punctuation is quite unlike that of real languages.

In our experiment, we use Wang and Eisner's (2016) "self-permutation" setting, where the dependents of each noun and verb are stochastically reordered, but according to a dependent ordering model that has been trained on the same language. For example, rephrasing a English sentence

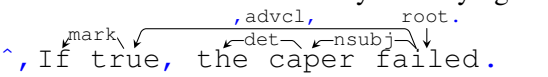


under an English ordering model may yield

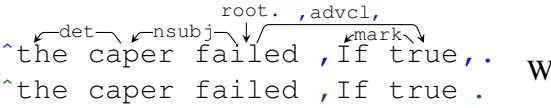


which is still grammatical except that , and . are wrongly swapped (after all, they have the same POS tag and relation type). Worse, permutation may yield bizarre punctuation such as , , at the start of a sentence.

Our punctuation model gives a straightforward remedy—instead of permuting the tree directly, we first discover its most likely underlying tree



by the maximizing variant of Algorithm 1 (§3.1). Then, we permute the underlying tree and sample the surface punctuation from the distribution modeled by the trained PFST, yielding



We leave the handling of capitalization to future work.

We test the naturalness of the permuted sentences by asking how well a word trigram language model trained on them could predict the original sentences.[23] As shown in Table 6, our per-

[23] So the two approaches to permutation yield different

	Punctuation			All		
	Base	Half	Full	Base	Half	Full
Arabic	**156.0**	231.3	186.1	**540.8**	590.3	553.4
Chinese	165.2	110.0	**61.4**	205.0	174.4	**78.7**
English	98.4	74.5	**51.0**	140.9	131.4	**75.4**
Hindi	10.8	11.0	**9.7**	118.4	118.8	**91.8**
Spanish	266.2	259.2	**194.5**	346.3	343.4	**239.3**

Table 6: Perplexity (evaluated on the train split to avoid evaluating generalization) of a trigram language model trained (with add-0.001 smoothing) on different versions of *rephrased training sentences*. "Punctuation" only evaluates perplexity on the trigrams that have punctuation. "All" evaluates on all the trigrams. "Base" permutes all surface dependents including punctuation (Wang and Eisner, 2016). "Full" is our full approach: recover underlying punctuation, permute remaining dependents, regenerate surface punctuation. "Half" is like "Full" but it permutes the non-punctuation tokens identically to "Base." The permutation model is trained on surface trees or recovered underlying trees T', respectively. In each 3-way comparison, we boldface the best result (always significant under a paired permutation test over per-sentence log-probabilities, $p < 0.05$).

mutation approach reduces the perplexity over the baseline on 4 of the 5 languages, often dramatically.

7 Related Work

Punctuation can aid syntactic analysis, since it signals phrase boundaries and sentence structure. Briscoe (1994) and White and Rajkumar (2008) parse punctuated sentences using hand-crafted constraint-based grammars that implement Nunberg's approach in a declarative way. These grammars treat *surface* punctuation symbols as ordinary words, but annotate the nonterminal categories so as to effectively keep track of the *underlying* punctuation. This is tantamount to crafting a grammar for underlyingly punctuated sentences and composing it with a finite-state noisy channel.

The parser of Ma et al. (2014) takes a different approach and treats punctuation marks as features of their neighboring words. Zhang et al. (2013) use a generative model for punctuated sentences, leting them restore punctuation marks during transition-based parsing of unpunctuated sentences. Li et al. (2005) use punctuation marks to segment a sentence: this "divide and rule" strategy reduces ambiguity in parsing of long Chinese sentences. Punctuation can similarly be used to constrain syntactic structure during grammar induction (Spitkovsky et al., 2011).

Punctuation restoration (§6.1) is useful for transcribing text from unpunctuated speech. The task is usually treated by tagging each slot with zero or more punctuation tokens, using a traditional sequence labeling method: conditional random fields (Lui and Wang, 2013; Lu and Ng, 2010), recurrent neural networks (Tilk and Alumäe, 2016), or transition-based systems (Ballesteros and Wanner, 2016).

8 Conclusion and Future Work

We have provided a new computational approach to modeling punctuation. In our model, syntactic constituents stochastically generate latent underlying left and right punctemes. Surface punctuation marks are not directly attached to the syntax tree, but are generated from sequences of adjacent punctemes by a (stochastic) finite-state string rewriting process . Our model is inspired by Nunberg's (1990) formal grammar for English punctuation, but is probabilistic and trainable. We give exact algorithms for training and inference.

We trained Nunberg-like models for 5 languages and L2 English. We compared the English model to Nunberg's, and showed how the trained models can be used across languages for punctuation restoration, correction, and adjustment.

In the future, we would like to study the usefulness of the recovered underlying trees on tasks such as syntactically sensitive sentiment analysis (Tai et al., 2015), machine translation (Cowan et al., 2006), relation extraction (Culotta and Sorensen, 2004), and coreference resolution (Kong et al., 2010). We would also like to investigate how underlying punctuation could aid parsing. For discriminative parsing, features for scoring the tree could refer to the underlying punctuation, not just the surface punctuation. For generative parsing (§3), we could follow the scheme in equation (1). For example, the p_{syn} factor in equation (1) might be a standard recurrent neural network grammar (RNNG) (Dyer et al., 2016); when a subtree of T is completed by the REDUCE operation of p_{syn}, the punctuation-augmented RNNG (1) would stochastically attach subtree-external left and right punctemes with $p_{\boldsymbol{\theta}}$ and transduce the subtree-internal slots with $p_{\boldsymbol{\phi}}$.

In the future, we are also interested in enriching the T' representation and making it more differ-

training data, but are compared fairly on the same test data.

ent from T, to underlyingly account for other phenomena in T such as capitalization, spacing, morphology, and non-projectivity (via reordering).

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant Nos. 1423276 and 1718846, including a REU supplement to the first author. We are grateful to the state of Maryland for the Maryland Advanced Research Computing Center, a crucial resource. We thank Xiaochen Li for early discussion, Argo lab members for further discussion, and the three reviewers for quality comments.

References

Apoorv Agarwal, Boyi Xie, Ilia Vovsha, Owen Rambow, and Rebecca Passonneau. 2011. Sentiment analysis of Twitter data. In *Proceedings of the Workshop on Language in Social Media (LSM 2011)*, pages 30–38.

Miguel Ballesteros and Leo Wanner. 2016. A neural network architecture for multilingual punctuation generation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1048–1053.

Yehoshua Bar-Hillel, M. Perles, and E. Shamir. 1961. On formal properties of simple phrase structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung*, 14:143–172. Reprinted in Y. Bar-Hillel (1964), *Language and Information: Selected Essays on their Theory and Application*, Addison-Wesley 1964, pages 116–150.

Jean Berstel and Christophe Reutenauer. 1988. *Rational Series and their Languages*. Springer-Verlag.

Ann Bies, Mark Ferguson, Karen Katz, Robert MacIntyre, Victoria Tredinnick, Grace Kim, Mary Ann Marcinkiewicz, and Britta Schasberger. 1995. Bracketing guidelines for Treebank II style: Penn Treebank project. Technical Report MS-CIS-95-06, University of Pennsylvania.

Ted Briscoe. 1994. Parsing (with) punctuation, etc. Technical report, Xerox European Research Laboratory.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750.

Noam Chomsky and Morris Halle. 1968. *The Sound Pattern of English*. Harper and Row, New York.

Ryan Cotterell, Nanyun Peng, and Jason Eisner. 2014. Stochastic contextual edit distance and probabilistic FSTs. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 625–630.

Ryan Cotterell, Nanyun Peng, and Jason Eisner. 2015. Modeling word forms using latent underlying morphs and phonology. *Transactions of the Association for Computational Linguistics (TACL)*, 3:433–447.

Ryan Cotterell, Tim Vieira, and Hinrich Schütze. 2016. A joint model of orthography and morphological segmentation. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 664–669.

Brooke Cowan, Ivona Kučerová, and Michael Collins. 2006. A discriminative model for tree-to-tree translation. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 232–241.

Aron Culotta and Jeffrey Sorensen. 2004. Dependency tree kernels for relation extraction. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*.

Timothy Dozat and Christopher Manning. 2017. Efficient third-order dependency parsers. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational*

Linguistics: Human Language Technologies (NAACL-HLT), pages 199–209.

Jason Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING)*, pages 340–345.

Jason Eisner. 2016. Inside-outside and forward-backward algorithms are just backprop. In *Proceedings of the EMNLP Workshop on Structured Prediction for NLP*.

A. Caracciolo di Forino. 1968. String processing languages and generalized Markov algorithms. In D. G. Bobrow, editor, *Symbol Manipulation Languages and Techniques*, pages 191–206. North-Holland Publishing Company, Amsterdam.

Kuzman Ganchev, Jennifer Gillenwater, and Ben Taskar. 2010. Posterior regularization for structured latent variable models. *Journal of Machine Learning Research*, 11:2001–2049.

Filip Ginter, Jan Hajič, Juhani Luotolahti, Milan Straka, and Daniel Zeman. 2017. CoNLL 2017 shared task - automatically annotated raw texts and word embeddings. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Yoav Goldberg and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*, pages 742–750.

Joshua Goodman. 1999. Semiring parsing. *Computational Linguistics*, 25(4):573–605.

George Heidorn. 2000. Intelligent writing assistance. In Robert Dale, Herman Moisl, and Harold Somers, editors, *Handbook of Natural Language Processing*, pages 181–207. Marcel Dekker, New York.

C. Douglas Johnson. 1972. *Formal Aspects of Phonological Description*. Mouton.

Bernard E. M. Jones. 1994. Exploring the role of punctuation in parsing natural text. In *COLING 1994 Volume 1: The 15th International Conference on Computational Linguistics*.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics (TACL)*, 4:313–327.

Kevin Knight and Daniel Marcu. 2002. Summarization beyond sentence extraction: A probabilistic approach to sentence compression. *Artificial Intelligence*, 139(1):91–107.

Fang Kong, Guodong Zhou, Longhua Qian, and Qiaoming Zhu. 2010. Dependency-driven anaphoricity determination for coreference resolution. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING)*, pages 599–607.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1–11.

Albert E. Krahn. 2014. *A New Paradigm for Punctuation*. Ph.D. thesis, The University of Wisconsin-Milwaukee.

Tao Lei, Yu Xin, Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. 2014. Low-rank tensors for scoring dependency structures. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1381–1391.

Roger Levy. 2008. A noisy-channel model of human sentence comprehension under uncertain input. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 234–243.

Xing Li, Chengqing Zong, and Rile Hu. 2005. A hierarchical parsing approach with punctuation processing for long Chinese sentences. In *Proceedings of the International Joint Conference on Natural Language Processing (IJCNLP)*.

Zhifei Li and Jason Eisner. 2009. First- and second-order expectation semirings with applications to minimum-risk training on translation forests. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 40–51.

Wei Lu and Hwee Tou Ng. 2010. Better punctuation prediction with dynamic conditional random fields. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 177–186.

Marco Lui and Li Wang. 2013. Recovering casing and punctuation using conditional random fields. In *Proceedings of the Australasian Language Technology Association Workshop (ALTA)*, pages 137–141.

Ji Ma, Yue Zhang, and Jingbo Zhu. 2014. Punctuation processing for projective dependency parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 791–796.

Gideon S. Mann and Andrew McCallum. 2010. Generalized expectation criteria for semi-supervised learning with weakly labeled data. *Journal of Machine Learning Research*, 11:955–984.

Andrey Andreevich Markov. 1960. The theory of algorithms. *American Mathematical Society Translations*, series 2(15):1–14.

Ilia Markov, Vivi Nastase, and Carlo Strapparava. 2018. Punctuation as native language interference. In *Proceedings of the 27th International Conference on Computational Linguistics (COLING)*, pages 3456–3466.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *Computing Research Repository (CoRR)*, arXiv:1301.3781.

Mark-Jan Nederhof and Giorgio Satta. 2003. Probabilistic parsing as intersection. In *8th International Workshop on Parsing Technologies (IWPT)*, pages 137–148.

Hwee Tou Ng, Siew Mei Wu, Ted Briscoe, Christian Hadiwinoto, Raymond Hendy Susanto, and Christopher Bryant. 2014. The CoNLL-2014 shared task on grammatical error correction. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, pages 1–14.

Joakim Nivre, Željko Agić, Lars Ahrenberg, Maria Jesus Aranzabe, Masayuki Asahara, Aitziber Atutxa, Miguel Ballesteros, John Bauer, Kepa Bengoetxea, Yevgeni Berzak, Riyaz Ahmad Bhat, Eckhard Bick, Carl Börstell, Cristina Bosco, Gosse Bouma, Sam Bowman, Gülşen Cebiroğlu Eryiğit, Giuseppe G. A. Celano, Fabricio Chalub, Çağrı Çöltekin, Miriam Connor, Elizabeth Davidson, Marie-Catherine de Marneffe, Arantza Diaz de Ilarraza, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Puneet Dwivedi, Marhaba Eli, Tomaž Erjavec, Richárd Farkas, Jennifer Foster, Claudia Freitas, Katarína Gajdošová, Daniel Galbraith, Marcos Garcia, Moa Gärdenfors, Sebastian Garza, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Memduh Gökırmak, Yoav Goldberg, Xavier Gómez Guinovart, Berta Gonzáles Saavedra, Matias Grioni, Normunds Grūzītis, Bruno Guillaume, Jan Hajič, Linh Hà Mỹ, Dag Haug, Barbora Hladká, Radu Ion, Elena Irimia, Anders Johannsen, Fredrik Jørgensen, Hüner Kaşıkara, Hiroshi Kanayama, Jenna Kanerva, Boris Katz, Jessica Kenney, Natalia Kotsyba, Simon Krek, Veronika Laippala, Lucia Lam, Phuong Lê Hồng, Alessandro Lenci, Nikola Ljubešić, Olga Lyashevskaya, Teresa Lynn, Aibek Makazhanov, Christopher Manning, Cătălina Mărănduc, David Mareček, Héctor Martínez Alonso, André Martins, Jan Mašek, Yuji Matsumoto, Ryan McDonald, Anna Missilä, Verginica Mititelu, Yusuke Miyao, Simonetta Montemagni, Keiko Sophie Mori, Shunsuke Mori, Bohdan Moskalevskyi, Kadri Muischnek, Nina Mustafina, Kaili Müürisep, Luong Nguyễn Thị, Huyền Nguyễn Thị Minh, Vitaly Nikolaev, Hanna Nurmi, Petya Osenova, Robert Östling, Lilja Øvrelid, Valeria Paiva, Elena Pascual, Marco Passarotti, Cenel-Augusto Perez, Slav Petrov, Jussi Piitulainen, Barbara Plank, Martin Popel, Lauma Pretkalniņa, Prokopis Prokopidis, Tiina Puolakainen, Sampo Pyysalo, Alexandre Rademaker, Loganathan Ramasamy, Livy Real, Laura Rituma, Rudolf Rosa, Shadi Saleh, Baiba Saulīte, Sebastian Schuster, Wolfgang Seeker, Mojgan Seraji, Lena Shakurova,

Mo Shen, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Mária Šimková, Kiril Simov, Aaron Smith, Carolyn Spadine, Alane Suhr, Umut Sulubacak, Zsolt Szántó, Takaaki Tanaka, Reut Tsarfaty, Francis Tyers, Sumire Uematsu, Larraitz Uria, Gertjan van Noord, Viktor Varga, Veronika Vincze, Lars Wallin, Jing Xian Wang, Jonathan North Washington, Mats Wirén, Zdeněk Žabokrtský, Amir Zeldes, Daniel Zeman, and Hanzhi Zhu. 2016. Universal Dependencies 1.4. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University. Data available at `http://universaldependencies.org`.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007a. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 915–932.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülşen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007b. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135.

Joakim Nivre et al. 2018. Universal dependencies annotation guidelines. Available at `universaldependencies.org`.

Geoffrey Nunberg. 1990. *The Linguistics of Punctuation*. Number 18 in CSLI Lecture Notes. Center for the Study of Language and Information.

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up? Sentiment classification using machine learning techniques. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Fernando C. N. Pereira and Michael D. Riley. 1996. Speech recognition by composition of weighted finite automata. *Computing Research Repository (CoRR)*, arXiv:cmp-lg/9603001.

Mohammad Sadegh Rasooli and Joel R. Tetreault. 2015. Yara parser: A fast and accurate dependency parser. *Computing Research Repository*, arXiv:1503.06733 (version 2).

Radim Řehůřek and Petr Sojka. 2010. Software framework for topic modelling with large corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50.

Valentin I. Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky. 2011. Punctuation: Making a point in unsupervised dependency parsing. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning*, CoNLL '11, pages 19–28.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (ACL-COLING)*, pages 1556–1566.

Ottokar Tilk and Tanel Alumäe. 2016. Bidirectional recurrent neural network with attention mechanism for punctuation restoration. In *Interspeech*, pages 3047–3051.

Ke M. Tran, Yonatan Bisk, Ashish Vaswani, Daniel Marcu, and Kevin Knight. 2016. Unsupervised neural hidden Markov models. In *Proceedings of the Workshop on Structured Prediction for NLP*, pages 63–71.

University of Chicago. 2010. *The Chicago Manual of Style*. University of Chicago Press.

Dingquan Wang and Jason Eisner. 2016. The Galactic Dependencies treebanks: Getting more data by synthesizing new languages. *Transactions of the Association for Computational Linguistics (TACL)*, 4:491–505.

Michael White and Rajakrishnan Rajkumar. 2008. A more precise analysis of punctuation for broad-coverage surface realization with CCG.

In *Proceedings of the COLING 2008 Workshop on Grammar Engineering Across Frameworks*, pages 17–24.

K. Xu, L. Xie, and K. Yao. 2016. Investigating LSTM for punctuation prediction. In *2016 10th International Symposium on Chinese Spoken Language Processing (ISCSLP)*, pages 1–5.

Richard Zens, Franz Josef Och, and Hermann Ney. 2002. Phrase-based statistical machine translation. In *Annual Conference on Artificial Intelligence*, pages 18–32.

Dongdong Zhang, Shuangzhi Wu, Nan Yang, and Mu Li. 2013. Punctuation prediction with transition-based parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 752–760.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 188–193.

Supplementary Reference Material: Details for Replicability

A Feature Templates for ATTACH

Below, we provide **feature templates** for the features used by the ATTACH model in §2.1. To illustrate, Table 7 lists all the non-backoff features that fire on a particular node in Figure 1 of the main paper. Specifically, Table 7 lists the nonzero features in the feature vector $\mathbf{f}(l, r, w)$ where w is the tree node that dominates the subject `Dale` and $(l, r) = (\text{``}, \text{''})$ says to surround that subject with quotation marks.

In general, the feature vector $\mathbf{f}(l, r, w)$ assigns nonzero values (1 values unless otherwise stated) to the features that are named by the following tuples. (We use dots here to separate the elements of a tuple.)

- `N`.l.r.g.$\bar{d}$, `N`.l.r.g.d, `N`.l.r.g, `N`.l.r.$\bar{d}$, and `N`.l.r.d, where g is the POS-tag of the word at w, d is the dependency relation that labels the edge of T that points to w, and $\bar{d} = \overleftarrow{\text{d}}$ or $\overrightarrow{\text{d}}$ according to the direction of that edge. The first feature name is most specific, while the remaining 4 features are **backoff features**.

  For example, such features can be used to say that an appositive ($d =$ `appos`) headed by a noun ($g =$ `NOUN`) likes to be surrounded by commas ($l = r =$ `,`).

  To make training faster and perhaps avoid local optima, we initialize the weight of feature `N`.l.r.d to its log-count in training data.

- `W`.h.l.r.g.$\bar{d}$, `W`.h.l.r.g.d, `W`.h.l.r.g, `W`.h.l.r.$\bar{d}$, and `W`.h.l.r.d, where h measures the length of the constituent headed by w: $h = 1$ for a short constituent (1–2 words), $h = 2$ for a medium constituent (3–5 words), and $h = 3$ for a long constituent (≥ 6 words).

  For example, a positive weight on `W.3.,.,.advcl` says that long subordinate clauses ($h = 3$, $d =$ `advcl`) are likely to be surrounded by commas.

- `A`.l.r.g.$\bar{d}$.d', `A`.l.r.g.d.d', `A`.l.r.g.d', `A`.l.r.$\bar{d}$.d', and `A`.l.r.d.d', for each dependency relation d' that occurs along the path from the root of T to the parent of w. (Here l, r, and g are properties of w as before, whereas d' refers to an ancestor of w.) The value of this feature is the number of times that d' appears along the path. Notice that if $d =$ `root`, the path is empty, so none of the `A` features fire.

  For example, such features might cause a subordinate clause to be punctuated differently depending on whether it is attached to the main verb or a more deeply nested verb.

- `C`.l.r.g.$\bar{d}$.d', `C`.l.r.g.d.d', `C`.l.r.g.d', `C`.l.r.$\bar{d}$.d', and `C`.l.r.d.d', for each dependency relation d' that appears on an edge from w to a child of w. The value of this feature is the number of such edges. Notice that if w is a leaf, it has no children, so none of the `C` features fire.

  For example, such features could be used to say that a relative clause that contains a subject ($d' =$ `subj`), such as an object-relative clause, likes to be surrounded by commas.

- `L`.l.g_{-1}.g_{+1} and `R`.r.g_{-1}.g_{+1}, where g_{-1} and g_{+1} are the POS-tags surrounding the slot where l or r (respectively) is generated. We use $g_{-1} =$ `BOS` or $g_{+1} =$ `EOS` if the slot is at the beginning or the end of the sentence (respectively).

- `S`.g.$\bar{d}$, `S`.g.d, `S`.g, `S`.$\bar{d}$ and `S`.d, provided that l and r are **symmetric punctemes**. Symmetry is determined by simultaneously scanning l from left to right and r from right to left, and checking whether the punctuation marks at each position form one of the following pairs:[24] `{}` `[]` `()` `“”` `<>` `¿?` `¡!` 《》 〈〉 【】 〖〗 「」 „ `--` . If l and r are both empty strings, they are not considered symmetric.

- c.l.r.g.$\bar{d}$, c.l.r.g.d, c.l.r.g, c.l.r.$\bar{d}$ and c.l.r.d, for each punctuation token c that appears at least once as *surface* punctuation within the constituent dominated by w. (That is, if w's constituent stretches from slot i to slot k, its **internal slots** are $j = i + 1, \ldots, k - 1$, and c must appear in x_j for some such j.)

[24] A more complete list could be compiled from Unicode's opening/closing punctuation pairs, but this list is sufficient for the experiments in this paper.

These features make it possible to implement punctuation marks of different precedence. For example, a conjunct is ordinarily delimited by commas (§5.2), but a conjunct that already contains internal commas ($c =$ `,`) may be delimited by semicolons instead, as shown below.[25] Similarly, an appositive that already contains internal commas may be delimited by dashes instead of commas.

```
There are two ways to read newspapers:
     in print, which is costly;
       or in digital, which is free.
```

Some of these features are not edge-local. They look at entire paths or constituents, or the surface punctuation of a constituent. However, they do admit tractable exact algorithms, similarly to a neural HMM (Tran et al., 2016). How?

During training, line 13 of Algorithm 1 is able to compute each feature vector $\mathbf{f}(l, r, w)$ given the observed input tree T and surface punctuation $\mathbf{x}$.

§6.2 and §6.3 both need to find the 1-best underlying tree T' that corresponds to the given T and $\mathbf{x}$ of a treebank sentence, so that it can correct or permute that sentence. As discussed at the end of §3.1, this makes use of the same feature vectors $\mathbf{f}(l, r, w)$, and merely replaces the inside algorithm with a Viterbi decoding algorithm.

The situation is slightly more difficult at test time, when T is still observed, but the surface punctuation is not observed and must be sampled (§6.1). However, we can still do exact joint sampling of T' and $\mathbf{x}$ by traversing T *bottom-up*. That is, after we have processed the child nodes of w, we can process w by sampling x_j at the internal slots between its children (using NOISYCHANNEL) and *then* sampling (ℓ, r) at its external slots (using ATTACH, which may depend on the x_j values via the c features).

B Posterior Regularization

Equation (5) includes the expectation of $c(T')$, which counts the nodes in T' whose l and r punctemes contain any unmatched punctuation tokens.

We define a criterion to decide whether l and r are unmatched, based on this list of matched symmetric tokens: `{} [] () "" <> ¿? ¡! 《》 〈〉 【】 〖〗 「」` . This is the same list used by the `S` feature in Appendix A, except that it omits the pairs where the two tokens are equal (namely `--` and `„`).

First, we modify l and r to filter out tokens that do not appear in the list above. We then check whether the modified l and r are symmetric punctemes in the sense of the `S` feature (Appendix A). If not, we count the node as having unmatched punctuation.

[25] Unfortunately, this feature does not explain why all *other* conjuncts in the same conjunction (including the final conjunct) also switch to semicolons.

Feature Type	Name	Value
$\texttt{N}.l.r.g.\bar{d}$	$\texttt{N.".".NOUN.}\overleftarrow{\texttt{nsubj}}$	1
$\texttt{W}.h.l.r.g.\bar{d}$	$\texttt{W.1.".".NOUN.}\overleftarrow{\texttt{nsubj}}$	1
$\texttt{S}.g.\bar{d}$	$\texttt{S.NOUN.}\overleftarrow{\texttt{nsubj}}$	1
$\texttt{A}.l.r.g.\bar{d}.d'$	$\texttt{A.".".NOUN.}\overleftarrow{\texttt{nsubj}}\texttt{.root}$	1
$\texttt{L}.l.g_{-1}.g_{+1}$	L.".BOS.NOUN	1
$\texttt{R}.r.g_{-1}.g_{+1}$	R."".NOUN.VERB	1

Table 7: A subset of the features that fire on the node with `nsubj` in Figure 1.

C Correction Feature Templates

For the correction model (§6.2), recall that we first find the 1-best underlyingly punctuated tree T'_e that explains a tree T along with its possibly erroneous or non-standard surface punctuation $\mathbf{x}_e$.

We then use ATTACH to generate corrected punctuation to attach to T. At this step, it may be beneficial to condition on knowledge of the reconstructed underlying punctuation that we reconstructed in T'_e. Thus, we add the following 2 feature templates, which are extended versions of the `N` and `W` features in Appendix A. In these templates for evaluating $\mathbf{f}(l, r, w)$ in a proposed T', l' and r' denote the left and right underlying punctemes attached to the corresponding node w_e in T'_e.

- $\texttt{N}.l.r.g.\bar{d}.l'.r'$, $\texttt{N}.l.r.\bar{d}.l'.r'$, $\texttt{N}.l.r.g.l'.r'$, $\texttt{N}.l.r.l'.r'$
- $\texttt{W}.h.l.r.g.\bar{d}.l'.r'$,$\texttt{W}.h.l.r.\bar{d}.l'.r'$, $\texttt{W}.h.l.r.g.l'.r'$, $\texttt{W}.h.l.r.l'.r'$

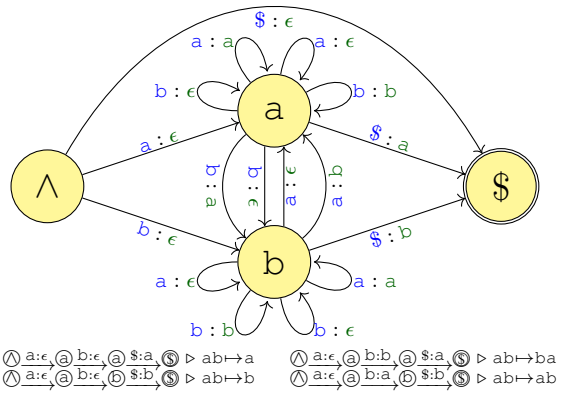


Figure 5: An example of our PFST on vocabulary $\Sigma = \{\mathtt{a}, \mathtt{b}\}$. The input (underlying punctuation tokens) is colored in blue and the output (surface punctuation tokens) is colored in green. All arc probabilities are suppressed for readability. (∧) is the start state, ($) is the final state, ϵ denotes the empty string, and \$ denotes a special end-of-input token. The four rewriting rules at the bottom of the figure are illustrated as different paths in the PFST.

D PFST implementation

Construct the PFST Recall from §2.2 and Figure 2 that our noisy channel is supposed to slide a 2-token window over the string of punctuation tokens, stochastically editing them as it goes.

In our PFST implementation, each arc has the form $(a) \xrightarrow{b:c} (d)$, which transitions from state a to state d while reading an underlying punctuation token b and generating a surface punctuation token c. Here the state label a represents the first token in the current sliding window, and the underlying token b provides the second token in that window. All surface tokens preceding a have already been output by the PFST. a has not yet been output by the PFST, because it will not necessarily be part of the surface string—it might still be deleted or transposed.

Choosing to traverse this arc corresponds to choosing a particular edit to the current window contents ab. After this edit, the new state d will reflect the first token in the *new* position of the sliding window.[26]

Recall from §2.2 that there are 4 possible edits to ab. These correspond to different choices of c and d in $(a) \xrightarrow{b:c} (d)$:

- To allow $ab \mapsto ab$ (no change), we include an arc with $c = a$ and $d = b$. This outputs the delayed token a, and then slides the window rightward so that b is now the first token.
- To allow $ab \mapsto b$ (left absorption), we include an arc with $c = \epsilon$ and $d = b$. This is identical to the previous case, except that it chooses to skip outputting a, so a has been deleted.
- To allow $ab \mapsto a$ (right absorption), we include an arc with $c = \epsilon$ and $d = a$. This is identical to the previous case, except that it is now b that it skips outputting. The first token in the sliding window therefore remains a.
- To allow $ab \mapsto ba$ (transposition), we include an arc with $c = b$ and $d = a$. This is identical to the previous case, except that it outputs b *before* the delayed token a. We still have not output a, so the first token in the sliding window remains a.

The probabilities of these 4 arcs are specified by the noisy channel parameters ϕ. They must sum to 1 because our noisy channel model will choose exactly one of the 4 edits for the current sliding window ab. This fact helps to ensure that our automaton is indeed a PFST, whose definition requires that the possible transitions from a given state a on a given input token b must have total probability of 1 (Cotterell et al., 2014).

We must also deal with boundary conditions, using boundary tokens ∧ and \$ at the start and end (respectively) of the underlying string.

- The PFST starts in the special state (∧), meaning that the sliding window is *before* the left edge of the string. The arcs from (∧) have the form $(\wedge) \xrightarrow{a:\epsilon} (a)$ (with probability 1), which effectively edits the boundary window $\wedge a$ by left absorption of the ∧. In effect, taking the arc simply slides the window rightward to the first "real" position of the sliding window, discovering that its first character will be the first underlying token a.
- We append the terminal token \$ to the underlying string.[27] Thus, the sliding window's final position has the form $a\$$. The arcs that consume this token have the form

[26] Specifically, the new window contents will be de, where e is the underlying token e that follows b. That token e will be read by the *next* arc—the arc taken from the new state d.

[27] In contrast, we did not prepend the initial token ∧ to the underlying string, but rather initialized in a state (∧) that pretended that ∧ had previously been read.

$(a) \xrightarrow{\$:a} (\$)$ (with probability 1), which effectively edits the boundary window $a\$$ by right absorption of the \$, but with the modification that it actually emits the delayed character a (which cannot undergo any further changes) and halts.

Let Σ be the vocabulary of punctuation types; our PFST F has $|\Sigma|+2$ states. There is a start state $(\wedge)$, a final state $(\$)$, and the remaining $|\Sigma|$ states each represents a punctuation type. An edge $s \xrightarrow{a:b} t$ denotes a transition from state s to t upon reading an underlying punctuation token a and generating a surface punctuation token b. The weight of this edge is the probability of such a transtion, which is

The set of edges in our PFST could be enumerated as follows:

- $(a) \xrightarrow{b:b} (a), (a) \xrightarrow{b:\epsilon} (b), (a) \xrightarrow{b:\epsilon} (a), (a) \xrightarrow{b:a} (b)$, for all distinct $a, b \in \Sigma$
- $(\wedge) \xrightarrow{a:\epsilon} (a), (a) \xrightarrow{\$:a} (\$), (a) \xrightarrow{a:\epsilon} (a), (a) \xrightarrow{a:a} (a)$, for all $a \in \Sigma$
- $(\wedge) \xrightarrow{\$:\epsilon} (\$)$ (same as the first case above but where $a = \$$ instead of $a \in \Sigma$)

Figure 5 illustrates the topology of our PFST with a toy vocabulary $\Sigma = \{\mathtt{a}, \mathtt{b}\}$. The PFST is locally normalized, because the weights of edges from a given state on the same input sum up to 1. (See Cotterell et al. (2014) for a full discussion of locally normalized PFSTs.)

From PFST to WFSA In §3.1, we construct a weighted finite-state acceptor (WFSA) for each slot, which describes all possible underlying strings u_i that can be rewritten as the surface string x_i that was observed in that slot. We will explain how to obtain this WFSA. The method is a detailed explanation of line 3 in Algorithm 1, already sketched in footnote 10.

First, we construct the composition $F \circ x_i$, where F is the PFST as shown in yellow in Figure 5. This composition extracts just the paths of F that would output the given surface string x_i. To perform this composition, we must represent the string x_i as an unweighted straight-line FSA with one arc per token of x_i. We show this FSA in green: $(0) \xrightarrow{x_i[1]} (1) \xrightarrow{x_i[2]} (2) \cdots \xrightarrow{x_i[|x|]} (x)$.

The composition $F \circ x_i$ is illustrated in Figure 6. Each state in the composition has the form (y,z), where y is some yellow state identifier in F and

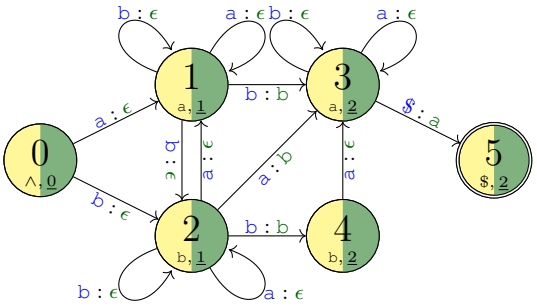


Figure 6: The WFST obtained by composing the yellow PFST F in Figure 5 with the green straight-line FSA $(0) \xrightarrow{\mathtt{b}} (1) \xrightarrow{\mathtt{a}} (2)$ that accepts $x_i = \mathtt{ba}$. The states are indexed from 0 (the initial state) to 5 (the final state). The bottom of each state shows the identifiers of the yellow and green states that it combines. Each arc is copied, along with its labels and weight, from a corresponding arc in Figure 5. Only states that are accessible from the initial state are shown; arc weights are suppressed for readability.

z is some green state identifier in the straight-line FSA for x_i. Thus, we depict it in Figure 6 as a yellow/green state. In other words, the state space of $F \circ x_i$ consists of the Cartesian product of the PFST states and the straight-line FSA states. The edge $(y,z) \xrightarrow{s} (y',z')$ exists if and only if $(y) \xrightarrow{s:t} (y')$ exists in F and $(z) \xrightarrow{t} (z')$ exists in x, with the edge weight inherited from the former. Note that the result of composition is a WFST rather than a PFST, since the arc weights are no longer guaranteed to be locally normalized.

Finally, to obtain the desired WFSA that describes the possible underlying strings u_i that could have yielded x_i, we project the WFST onto its domain (input). This is a simple matter of dropping the output (which follows the colon) from each arc in the WFST of Figure 6. The weights are retained.