Learning Neural Templates for Text Generation

The Harvard community has made this article openly available. Please share how this access benefits you. Your story matters

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Citable link</td>
<td><a href="http://nrs.harvard.edu/urn-3:HUL.InstRepos:40827359">http://nrs.harvard.edu/urn-3:HUL.InstRepos:40827359</a></td>
</tr>
<tr>
<td>Terms of Use</td>
<td>This article was downloaded from Harvard University’s DASH repository, and is made available under the terms and conditions applicable to Open Access Policy Articles, as set forth at <a href="http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#OAP">http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#OAP</a></td>
</tr>
</tbody>
</table>
Learning Neural Templates for Text Generation

Sam Wiseman  Stuart M. Shieber  Alexander M. Rush
School of Engineering and Applied Sciences
Harvard University
Cambridge, MA, USA
{swiseman,sieber,srush}@seas.harvard.edu

Abstract

While neural, encoder-decoder models have had significant empirical success in text generation, there remain several unaddressed problems with this style of generation. Encoder-decoder models are largely (a) uninterpretable, and (b) difficult to control in terms of their phrasing or content. This work proposes a neural generation system using a hidden semi-markov model (HSMM) decoder, which learns latent, discrete templates jointly with learning to generate. We show that this model learns useful templates, and that these templates make generation both more interpretable and controllable. Furthermore, we show that this approach scales to real data sets and achieves strong performance nearing that of encoder-decoder text generation models.

1 Introduction

With the continued success of encoder-decoder models for machine translation and related tasks, there has been great interest in extending these methods to build general-purpose, data-driven natural language generation (NLG) systems (Mei et al., 2016; Dušek and Jurcicek, 2016; Lebret et al., 2016; Chisholm et al., 2017; Wiseman et al., 2017). These encoder-decoder models (Sutskever et al., 2014; Cho et al., 2014; Bahdanau et al., 2015) use a neural encoder model to represent a source knowledge base, and a decoder model to emit a textual description word-by-word, conditioned on the source encoding. This style of generation contrasts with the more traditional division of labor in NLG, which famously emphasizes addressing the two questions of “what to say” and “how to say it” separately, and which leads to systems with explicit content selection, macro- and micro-planning, and surface realization components (Reiter and Dale, 1997; Jurafsky and Martin, 2014).

Encoder-decoder generation systems appear to have increased the fluency of NLG outputs, while reducing the manual effort required. However, due to the black-box nature of generic encoder-decoder models, these systems have also largely sacrificed two important desiderata that are often found in more traditional systems, namely (a) interpretable outputs that (b) can be easily controlled in terms of form and content.

This work considers building interpretable and controllable neural generation systems, and proposes a specific first step: a new data-driven generation model for learning discrete, template-like
structures for conditional text generation. The core system uses a novel, neural hidden semi-markov model (HSMM) decoder, which provides a principled approach to template-like text generation. We further describe efficient methods for training this model in an entirely data-driven way by backpropagation through inference. Generating with the template-like structures induced by the neural HSMM allows for the explicit representation of what the system intends to say (in the form of a learned template) and how it is attempting to say it (in the form of an instantiated template).

We show that we can achieve performance competitive with other neural NLG approaches, while making progress satisfying the above two desiderata. Concretely, our experiments indicate that we can induce explicit templates (as shown in Figure 1) while achieving competitive automatic scores, and that we can control and interpret our generations by manipulating these templates. Finally, while our experiments focus on the data-to-text regime, we believe the proposed methodology represents a compelling approach to learning discrete, latent-variable representations of conditional text.

2 Related Work

A core task of NLG is to generate textual descriptions of knowledge base records. A common approach is to use hand-engineered templates (Kulkich, 1983; McKeown, 1992; McRoy et al., 2000), but there has also been interest in creating templates in an automated manner. For instance, many authors induce templates by clustering sentences and then abstracting templated fields with hand-engineered rules (Angeli et al., 2010; Konstdadi et al., 2013; Howald et al., 2013), or with a pipeline of other automatic approaches (Wang and Cardie, 2013).

There has also been work in incorporating probabilistic notions of templates into generation models (Liang et al., 2009; Konstas and Lapata, 2013), which is similar to our approach. However, these approaches have always been conjoined with discriminative classifiers or rerankers in order to actually accomplish the generation (Angeli et al., 2010; Konstas and Lapata, 2013). In addition, these models explicitly model knowledge base field selection, whereas the model we present is fundamentally an end-to-end model over generation segments.

Recently, a new paradigm has emerged around neural text generation systems based on machine translation (Sutskever et al., 2014; Cho et al., 2014; Bahdanau et al., 2015). Most of this work has used unconstrained black-box encoder-decoder approaches. There has been some work on discrete variables in this context, including extracting representations (Shen et al., 2018), incorporating discrete latent variables in text modeling (Yang et al., 2018), and using non-HSMM segmental models for machine translation or summarization (Yu et al., 2016; Wang et al., 2017; Huang et al., 2018). Dai et al. (2017) develop an approximate inference scheme for a neural HSMM using RNNs for continuous emissions; in contrast we maximize the exact log-marginal, and use RNNs to parameterize a discrete emission distribution. Finally, there has also been much recent interest in segmental RNN models for non-generative tasks in NLP (Tang et al., 2016; Kong et al., 2016; Lu et al., 2016).

The neural text generation community has also recently been interested in “controllable” text generation (Hu et al., 2017), where various aspects of the text (often sentiment) are manipulated or transferred (Shen et al., 2017; Zhao et al., 2018; Li et al., 2018). In contrast, here we focus on controlling either the content of a generation or the way it is expressed by manipulating the (latent) template used in realizing the generation.

3 Overview: Data-Driven NLG

Our focus is on generating a textual description of a knowledge base or meaning representation. Following standard notation (Liang et al., 2009; Wiseman et al., 2017), let \( x = \{ r_1 \ldots r_J \} \) be a collection of records. A record is made up of a type \( r.t \), an entity \( r.e \), and a value \( r.m \). For example, a knowledge base of restaurants might have a record with \( r.t = \text{Cuisine}, r.e = \text{Denny’s}, \) and \( r.m = \text{American} \). The aim is to generate an adequate and fluent text description \( \hat{y}_{1:T} = \hat{y}_1, \ldots, \hat{y}_T \) of \( x \). Concretely, we consider the E2E Dataset (Novikova et al., 2017) and the WikiBio Dataset (Lebret et al., 2016). We show an example E2E knowledge base \( x \) in the top of Figure 1. The top of Figure 2 shows an example knowledge base \( x \) from the WikiBio dataset, where it is paired with a reference text \( y = y_{1:T} \) at the bottom.

The dominant approach in neural NLG has been
to use an encoder network over $x$ and then a conditional decoder network to generate $y$, training the whole system in an end-to-end manner. To generate a description for a given example, a black-box network (such as an RNN) is used to produce a distribution over the next word, from which a choice is made and fed back into the system. The entire distribution is driven by the internal states of the neural network.

While effective, relying on a neural decoder makes it difficult to understand what aspects of $x$ are correlated with a particular system output. This leads to problems both in controlling fine-grained aspects of the generation process and in interpreting model mistakes.

As an example of why controllability is important, consider the records in Figure 1. Given these inputs an end-user might want to generate an output meeting specific constraints, such as not mentioning any information relating to customer rating. Under a standard encoder-decoder style model, one could filter out this information either from the encoder or decoder, but in practice this would lead to unexpected changes in output that might propagate through the whole system.

As an example of the difficulty of interpreting mistakes, consider the following actual generation from an encoder-decoder style system for the records in Figure 2: “Frederick Parker-Rhodes (21 November 1914 - 2 March 1987) was an English linguist, plant pathologist, computer scientist, mathematician, mystic, and mycologist.” In addition to not being fluent, it is unclear what the end of this sentence will be attempted to convey: it may be attempting to convey a fact not actually in the knowledge base (e.g., Parker-Rhodes studied), or perhaps it is simply failing to fluently realize information that is in the knowledge base (e.g., Parker-Rhodes’s country of residence).

Traditional NLG systems (Kukich, 1983; McKeeown, 1992; Belz, 2008; Gatt and Reiter, 2009), in contrast, largely avoid these problems. Since they typically employ an explicit planning component, which decides which knowledge base records to focus on, and a surface realization component, which realizes the chosen records, the intent of the system is always explicit, and it may be modified to meet constraints.

The goal of this work is to propose an approach to neural NLG that addresses these issues in a principled way. We target this goal by proposing a new model that generates with template-like objects induced by a neural HSMM (see Figure 1). Templates are useful here because they represent a fixed plan for the generation’s content, and because they make it clear what part of the generation is associated with which record in the knowledge base.

## 4 Background: Semi-Markov Models

What does it mean to learn a template? It is natural to think of a template as a sequence of typed text-segments, perhaps with some segments acting as the template’s “backbone” (Wang and Cardie, 2013), and the remaining segments filled in from the knowledge base.

A natural probabilistic model conforming with this intuition is the hidden semi-markov model (HSMM) (Gales and Young, 1993; Ostendorf et al., 1996), which models latent segmentations in an output sequence. Informally, an HSMM is much like an HMM, except emissions may last multiple time-steps, and multi-step emissions need not be independent of each other conditioned on the state.

We briefly review HSMMs following Murphy (2002). Assume we have a sequence of observed tokens $y_1 \ldots y_T$ and a discrete, latent state $z_t \in \{1, \ldots, K\}$ for each timestep. We addition-
ally use two per-timestep variables to model multi-step segments: a length variable \( l_t \in \{1, \ldots, L \} \) specifying the length of the current segment, and a deterministic binary variable \( f_t \) indicating whether a segment finishes at time \( t \). We will consider in particular conditional HSMMs, which condition on a source \( x \), essentially giving us an HSMM decoder.

An HSMM specifies a joint distribution on the observations and latent segmentations. Letting \( \theta \) denote all the parameters of the model, and using the variables introduced above, we can write the corresponding joint-likelihood as follows

\[
p(y, z, l, f \mid x; \theta) = \prod_{t=0}^{T-1} p(z_{t+1}, l_{t+1} \mid z_t, l_t, x)^{f_t} \times \prod_{t=1}^{T} p(y_{t-l_t+1:t} \mid z_t, l_t, x)^{f_t},
\]

where we take \( z_0 \) to be a distinguished start-state, and the deterministic \( f_t \) variables are used for excluding non-segment log probabilities. We further assume \( p(z_{t+1}, l_{t+1} \mid z_t, l_t, x) \) factors as \( p(z_{t+1} \mid z_t, x) \times p(l_{t+1} \mid z_{t+1}) \). Thus, the likelihood is given by the product of the probabilities of each discrete state transition made, the probability of the length of each segment given its discrete state, and the probability of the observations in each segment, given its state and length.

5 A Neural HSMM Decoder

We use a novel, neural parameterization of an HSMM to specify the probabilities in the likelihood above. This full model, sketched out in Figure 3, allows us to incorporate the modeling components, such as LSTM and attention, that make neural text generation effective, while maintaining the HSMM structure.

5.1 Parameterization

Since our model must condition on \( x \), let \( r_j \in \mathbb{R}^d \) represent a real embedding of record \( r_j \in x \), and let \( x_u \in \mathbb{R}^d \) represent a real embedding of the entire knowledge base \( x \), obtained by max-pooling coordinate-wise over all the \( r_j \). It is also useful to have a representation of just the unique types of records that appear in \( x \), and so we also define \( x_u \in \mathbb{R}^d \) to be the sum of the embeddings of the unique types appearing in \( x \), plus a bias vector and followed by a ReLU nonlinearity.

\[\text{Figure 3: HSMM factor graph (under a known segmentation) to illustrate parameters. Here we assume } z_1 \text{ is in the “red” state (out of } K \text{ possibilities), and transitions to the “blue” state after emitting three words. The transition model, shown as } T, \text{ is a function of the two states and the neural encoded source } x. \text{ The emission model is a function of a “red” RNN model (with copy attention over } x \text{) that generates words 1, 2 and 3. After transitioning, the next word } y_4 \text{ is generated by the “blue” RNN, but independently of the previous words.}\]

**Transition Distribution** The transition distribution \( p(z_{t+1} \mid z_t, x) \) may be viewed as a \( K \times K \) matrix of probabilities, where each row sums to 1. We define this matrix to be

\[
p(z_{t+1} \mid z_t, x) \propto AB + C(x_u)D(x_u),
\]

where \( A \in \mathbb{R}^{K \times m_1}, B \in \mathbb{R}^{m_1 \times K} \) are state embeddings, and where \( C : \mathbb{R}^d \rightarrow \mathbb{R}^{K \times m_2} \) and \( D : \mathbb{R}^d \rightarrow \mathbb{R}^{m_2 \times K} \) are parameterized non-linear functions of \( x_u \). We apply a row-wise softmax to the resulting matrix to obtain the desired probabilities.

**Length Distribution** We simply fix all length probabilities \( p(l_{t+1} \mid z_{t+1}) \) to be uniform up to a maximum length \( L \).

**Emission Distribution** The emission model models the generation of a text segment conditioned on a latent state and source information, and so requires a richer parameterization. Inspired by the models used for neural NLG, we base this model on an RNN decoder, and write a segment’s probability as a product over token-level probabilities,

\[
p(y_{t-l_t+1:t} \mid z_t = k, l_t = l, x) = \prod_{i=1}^{l_t} p(y_{t-l_t+i:t-l_t+i-1} \mid y_{t-l_t+1:t-l_t+i-1}, z_t = k, x) \times p(\langle \text{seg} \rangle \mid y_{t-l_t+1:t}, z_t = k, x) \times 1_{\{l_t = l\}},
\]

\[\text{We experimented with parameterizing the length distribution, but found that it led to inferior performance. Forcing the length probabilities to be uniform encourages the model to cluster together functionally similar emissions of different lengths, while parameterizing them can lead to states that specialize to specific emission lengths.}\]
where \( \langle \text{seg} \rangle \) is an end of segment token. The RNN decoder uses attention and copy-attention over the embedded records \( r_j \), and is conditioned on \( z_t = k \) by concatenating an embedding corresponding to the \( k \)'th latent state to the RNN’s input; the RNN is also conditioned on the entire \( x \) by initializing its hidden state with \( x_0 \).

More concretely, let \( h^k_{t-1} \in \mathbb{R}^d \) be the state of an RNN conditioned on \( x \) and \( z_t = k \) (as above) run over the sequence \( y_{t-l_t+1:t-l_t+i-1} \). We let the model attend over records \( r_j \) using \( h^k_{t-1} \) (in the style of Luong et al. (2015)), producing a context vector \( c^k_{t-1} \). We may then obtain scores \( v_{i-1} \) for each word in the output vocabulary,

\[
v_{i-1} = W \tan\hskip1pt h^k_{t-1} \circ [h^k_{t-1}, c^k_{t-1}],
\]

with parameters \( g^k \in \mathbb{R}^{2d} \) and \( W \in \mathbb{R}^{V \times 2d} \). Note that there is a \( g^k \) vector for each of \( K \) discrete states. To additionally implement a kind of slot filling, we allow emissions to be directly copied from the value portion of the records \( r_j \) using copy attention ( Gülçehre et al., 2016; Gu et al., 2016; Yang et al., 2016). Define copy scores,

\[
\rho_j = r^j_T \tan\hskip1pt (g^k \circ h^k_{t-1}),
\]

where \( g^k \in \mathbb{R}^d \). We then normalize the output-vocabulary and copy scores together, to arrive at

\[
\tilde{v}_{i-1} = \text{softmax}(v_{i-1}, \rho_1, \ldots, \rho_J),
\]

and thus

\[
p(y_{t-l_t+i} = w | y_{t-l_t+1:t-l_t+i-1}, z_t = k, x) = \tilde{v}_{i-1,w} + \sum_{j : r_j,m = w} \tilde{v}_{i-1,V+j},
\]

### An Autoregressive Variant

The model as specified assumes segments are independent conditioned on the associated latent state and \( x \). While this assumption still allows for reasonable performance, we can tractably allow interdependence between tokens (but not segments) by having each next-token distribution depend on all the previously generated tokens, giving us an autoregressive HSMM. For this model, we will in fact use \( p(y_{t-l_t+i} = w | y_{t-l_t+1:t-l_t+i-1}, z_t = k, x) \) in defining our emission model, which is easily implemented by using an additional RNN run over all the preceding tokens. We will report scores for both non-autoregressive and autoregressive HSMM decoders below.

### 5.2 Learning

The model requires fitting a large set of neural network parameters. Since we assume \( z, l, \) and \( f \) are unobserved, we marginalize over these variables to maximize the log marginal-likelihood of the observed tokens \( y \) given \( x \). The HSMM marginal-likelihood calculation can be carried out efficiently with a dynamic program analogous to either the forward- or backward-algorithm familiar from HMMs (Rabiner, 1989).

It is actually more convenient to use the backward-algorithm formulation when using RNNs to parameterize the emission distributions, and we briefly review the backward recurrences here, again following Murphy (2002). We have:

\[
\beta_t(j) = p(y_{t+1:T} | z_t = j, f_t = 1, x) = \sum_{k=1}^K \beta^*_t(k) p(z_{t+1} = k | z_t = j)
\]

\[
\beta^*_t(k) = p(y_{t+1:T} | z_{t+1} = k, f_t = 1, x) = \sum_{l=1}^L \left[ \beta_{t+1}(k) p(l_{t+1} = l | z_{t+1} = k) \right],
\]

with base case \( \beta_T(j) = 1 \). We can now obtain the marginal probability of \( y \) as \( p(y | x) = \sum_{k=1}^K \beta^*_0(k) p(z_1 = k) \), where we have used the fact that \( f_0 \) must be 1, and we therefore train to maximize the log-marginal likelihood of the observed \( y \):

\[
\ln p(y | x; \theta) = \ln \sum_{k=1}^K \beta^*_0(k) p(z_1 = k).
\]  

Since the quantities in (1) are obtained from a dynamic program, which is itself differentiable, we may simply maximize with respect to the parameters \( \theta \) by back-propagating through the dynamic program; this is easily accomplished with automatic differentiation packages, and we use pytorch (Paszke et al., 2017) in all experiments.

### 5.3 Extracting Templates and Generating

After training, we could simply condition on a new database and generate with beam search, as is standard with encoder-decoder models. However, the structured approach we have developed allows us to generate in a more template-like way, giving us more interpretable and controllable generations.
[The Golden Palace]$_{55}$ [is a]$_{59}$ [coffee shop]$_{12}$ [providing]$_{13}$ [Indian]$_{30}$ [food]$_{1}$ [in the]$_{17}$ [20-25]$_{26}$ [price range]$_{16}$ [. ]$_{12}$ [It is]$_{8}$ [located in the]$_{25}$ [riverside]$_{40}$ [. ]$_{53}$ [Its customer rating is]$_{19}$ [high]$_{23}$ [. ]$_{12}$

Figure 4: A sample Viterbi segmentation of a training text; subscripted numbers indicate the corresponding latent state. From this we can extract a template with $S = 17$ segments; compare with the template used at the bottom of Figure 1.

First, note that given a database $x$ and reference generation $y$ we can obtain the MAP assignment to the variables $z$, $t$, and $f$ with a dynamic program similar to the Viterbi algorithm familiar from HMMs. These assignments will give us a typed segmentation of $y$, and we show an example Viterbi segmentation of some training text in Figure 4. Computing MAP segmentations allows us to associate text-segments (i.e., phrases) with the discrete labels $z_i$ that frequently generate them. These MAP segmentations can be used in an exploratory way, as a sort of dimensionality reduction of the generations in the corpus. More importantly for us, however, they can also be used to guide generation.

In particular, since each MAP segmentation implies a sequence of hidden states $z$, we may run a template extraction step, where we collect the most common “templates” (i.e., sequences of hidden states) seen in the training data. Each “template” $z^{(i)}$ consists of a sequence of latent states, with $z^{(i)} = z_1^{(i)}, \ldots, z_S^{(i)}$ representing the $S$ distinct segments in the $i$’th extracted template (recall that we will technically have a $z_i$ for each time-step, and so $z^{(i)}$ is obtained by collapsing adjacent $z_i$’s with the same value); see Figure 4 for an example template (with $S = 17$) that can be extracted from the E2E corpus. The bottom of Figure 1 shows a visualization of this extracted template, where discrete states are replaced by the phrases they frequently generate in the training data.

With our templates $z^{(i)}$ in hand, we can then restrict the model to using (one of) them during generation. In particular, given a new input $x$, we may generate by computing

$$\hat{y}^{(i)} = \arg \max_{y'} p(y', z^{(i)} | x),$$

(2)

which gives us a generation $\hat{y}^{(i)}$ for each extracted template $z^{(i)}$. For example, the generation in Figure 1 is obtained by maximizing (2) with $x$ set to the database in Figure 1 and $z^{(i)}$ set to the template extracted in Figure 4. In practice, the $\arg \max$ in (2) will be intractable to calculate exactly due to the use of RNNs in defining the emission distribution, and so we approximate it with a constrained beam search. This beam search looks very similar to that typically used with RNN decoders, except the search occurs only over a segment, for a particular latent state $k$.

5.4 Discussion

Returning to the discussion of controllability and interpretability, we note that with the proposed model (a) it is possible to explicitly force the generation to use a chosen template $z^{(i)}$, which is itself automatically learned from training data, and (b) that every segment in the generated $\hat{y}^{(i)}$ is typed by its corresponding latent variable. We explore these issues empirically in Section 7.1.

We also note that these properties may be useful for other text applications, and that they offer an additional perspective on how to approach latent variable modeling for text. Whereas there has been much recent interest in learning continuous latent variable representations for text (see Section 2), it has been somewhat unclear what the latent variables to be learned are intended to capture. On the other hand, the latent, template-like structures we induce here represent a plausible, probabilistic latent variable story, and allow for a more controllable method of generation.

Finally, we highlight one significant possible issue with this model – the assumption that segments are independent of each other given the corresponding latent variable and $x$. Here we note that the fact that we are allowed to condition on $x$ is quite powerful. Indeed, a clever encoder could capture much of the necessary interdependence between the segments to be generated (e.g., the correct determiner for an upcoming noun phrase) in its encoding, allowing the segments themselves to be decoded more or less independently, given $x$.

6 Data and Methods

Our experiments apply the approach outlined above to two recent, data-driven NLG tasks.

6.1 Datasets

Experiments use the E2E (Novikova et al., 2017) and WikiBio (Lebret et al., 2016) datasets, examples of which are shown in Figures 1 and 2, respectively. The former dataset, used for the
2018 E2E-Gen Shared Task, contains approximately 50K total examples, and uses 945 distinct word types, and the latter dataset contains approximately 500K examples and uses approximately 400K word types. Because our emission model uses a word-level copy mechanism, any record with a phrase consisting of $n$ words as its value is replaced with $n$ word values, following the preprocessing of Lebret et al. (2016). For example, “type[coffee shop]” in Figure 1 becomes “type-1[coffee]” and “type-2[shop].”

For both datasets we compare with published encoder-decoder models, as well as with direct template-style baselines. The E2E task is evaluated in terms of BLEU (Papineni et al., 2002), NIST (Belz and Reiter, 2006), ROUGE (Lin, 2004), CIDEr (Vedantam et al., 2015), and METEOR (Banerjee and Lavie, 2005). The benchmark system for the task is an encoder-decoder style system followed by a reranker, proposed by Dušek and Jurčícek (2016). We compare to this baseline, as well as to a simple but competitive non-parametric template-like baseline (“SUB” in tables), which selects a training sentence with records that maximally overlap (without including extraneous records) the unseen set of records we wish to generate from; ties are broken at random. Then, word-spans in the chosen training sentence are aligned with records by string-match, and replaced with the corresponding fields of the new set of records.\(^3\)

The WikiBio dataset is evaluated in terms of BLEU, NIST, and ROUGE, and we compare with the systems and baselines implemented by Lebret et al. (2016), which include two neural, encoder-decoder style models, as well as a Kneser-Ney, templated baseline.

6.2 Model and Training Details

We first emphasize two additional methodological details important for obtaining good performance.

Constraining Learning We were able to learn more plausible segmentations of $y$ by constraining the model to respect word spans $y_{l+1:t+l}$ that appear in some record $r_j \in x$. We accomplish this by giving zero probability (within the backward re-

Increasing the Number of Hidden States While a larger $K$ allows for a more expressive latent model, computing $K$ emission distributions over the vocabulary can be prohibitively expensive. We therefore tie the emission distribution between multiple states, while allowing them to have a different transition distributions.

We give additional architectural details of our model in the Supplemental Material; here we note that we use an MLP to embed $r_j \in \mathbb{R}^d$, and a 1-layer LSTM (Hochreiter and Schmidhuber, 1997) in defining our emission distributions. In order to reduce the amount of memory used, we restrict our output vocabulary (and thus the height of the matrix $W$ in Section 5) to only contain words in $y$ that are not present in $x$; any word in $y$ present in $x$ is assumed to be copied. In the case where a word $y_t$ appears in a record $r_j$ (and could therefore have been copied), the input to the LSTM at time $t+1$ is computed using information from $r_j$; if there are multiple $r_j$ from which $y_t$ could have been copied, the computed representations are simply averaged.

For all experiments, we set $d = 300$ and $L = 4$. At generation time, we select the 100 most common templates $z^{(i)}$, perform beam search with a beam of size 5, and select the generation with the highest overall joint probability.

For our E2E experiments, our best non-autoregressive model has 55 “base” states, duplicated 5 times, for a total of $K = 275$ states, and our best autoregressive model uses $K = 60$ states, without any duplication. For our WikiBio experiments, both our best non-autoregressive and autoregressive models uses 45 base states duplicated 3 times, for a total of $K = 135$ states. In all cases, $K$ was chosen based on BLEU performance on held-out validation data. Code implementing our models is available at https://github.com/harvardnlp/neural-template-gen.

7 Results

Our results on automatic metrics are shown in Tables 1 and 2. In general, we find that the templated baselines underperform neural models, whereas our proposed model is fairly competitive with neural models, and sometimes even out-

\(^2\)We use the official E2E NLG Challenge scoring scripts at https://github.com/tuetschek/e2e-metrics.

\(^3\)For categorical records, like “familyFriendly”, which cannot easily be aligned with a phrase, we simply select only candidate training sentences with the same categorical value.
Table 1: Comparison of the two best neural systems of Dušek and Jurčícek (2016), which forms the baseline for the E2E challenge, a non-parametric, substitution-based baseline (see text), and our HSMM models (denoted “NTemp” and “NTemp+AR” for the non-autoregressive and autoregressive versions, resp.) on the validation and test portions of the E2E dataset. “ROUGE” is ROUGE-L. Models are evaluated using the official E2E NLG Challenge scoring scripts.

<table>
<thead>
<tr>
<th></th>
<th>BLEU</th>
<th>NIST</th>
<th>ROUGE</th>
<th>CIDEr</th>
<th>METEOR</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Validation</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D&amp;J</td>
<td>69.25</td>
<td>8.48</td>
<td>72.57</td>
<td>2.40</td>
<td>47.03</td>
</tr>
<tr>
<td>SUB</td>
<td>43.71</td>
<td>6.72</td>
<td>55.35</td>
<td>1.41</td>
<td>37.87</td>
</tr>
<tr>
<td>NTemp</td>
<td>64.53</td>
<td>7.66</td>
<td>68.18</td>
<td>1.82</td>
<td>42.46</td>
</tr>
<tr>
<td>NTemp+AR</td>
<td>67.07</td>
<td>7.98</td>
<td>69.50</td>
<td>2.29</td>
<td>43.07</td>
</tr>
<tr>
<td><strong>Test</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D&amp;J</td>
<td>65.93</td>
<td>8.59</td>
<td>68.50</td>
<td>2.23</td>
<td>44.83</td>
</tr>
<tr>
<td>SUB</td>
<td>43.78</td>
<td>6.88</td>
<td>54.64</td>
<td>1.39</td>
<td>37.35</td>
</tr>
<tr>
<td>NTemp</td>
<td>55.17</td>
<td>7.14</td>
<td>65.70</td>
<td>1.70</td>
<td>41.91</td>
</tr>
<tr>
<td>NTemp+AR</td>
<td>59.80</td>
<td>7.56</td>
<td>65.01</td>
<td>1.95</td>
<td>38.75</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>BLEU</th>
<th>NIST</th>
<th>ROUGE</th>
<th>CIDEr</th>
<th>METEOR</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Validation</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D&amp;J</td>
<td>69.25</td>
<td>8.48</td>
<td>72.57</td>
<td>2.40</td>
<td>47.03</td>
</tr>
<tr>
<td>SUB</td>
<td>43.71</td>
<td>6.72</td>
<td>55.35</td>
<td>1.41</td>
<td>37.87</td>
</tr>
<tr>
<td>NTemp</td>
<td>64.53</td>
<td>7.66</td>
<td>68.18</td>
<td>1.82</td>
<td>42.46</td>
</tr>
<tr>
<td>NTemp+AR</td>
<td>67.07</td>
<td>7.98</td>
<td>69.50</td>
<td>2.29</td>
<td>43.07</td>
</tr>
<tr>
<td><strong>Test</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D&amp;J</td>
<td>65.93</td>
<td>8.59</td>
<td>68.50</td>
<td>2.23</td>
<td>44.83</td>
</tr>
<tr>
<td>SUB</td>
<td>43.78</td>
<td>6.88</td>
<td>54.64</td>
<td>1.39</td>
<td>37.35</td>
</tr>
<tr>
<td>NTemp</td>
<td>55.17</td>
<td>7.14</td>
<td>65.70</td>
<td>1.70</td>
<td>41.91</td>
</tr>
<tr>
<td>NTemp+AR</td>
<td>59.80</td>
<td>7.56</td>
<td>65.01</td>
<td>1.95</td>
<td>38.75</td>
</tr>
</tbody>
</table>

Table 2: Top: comparison of the two best neural systems of Lebret et al. (2016), their templated baseline, and our HSMM models (denoted “NTemp” and “NTemp+AR” for the non-autoregressive and autoregressive versions, resp.) on the test portion of the WikiBio dataset. Models marked with a † are from Lebret et al. (2016), and following their methodology we use ROUGE-4. Bottom: state-of-the-art seq2seq-style results from Liu et al. (2018).

<table>
<thead>
<tr>
<th></th>
<th>BLEU</th>
<th>NIST</th>
<th>ROUGE-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Template KN †</td>
<td>19.8</td>
<td>5.19</td>
<td>10.7</td>
</tr>
<tr>
<td>NNLM (field) †</td>
<td>33.4</td>
<td>7.52</td>
<td>23.9</td>
</tr>
<tr>
<td>NNLM (field &amp; word) †</td>
<td>34.7</td>
<td>7.98</td>
<td>25.8</td>
</tr>
<tr>
<td>NTemp</td>
<td>34.2</td>
<td>7.94</td>
<td>35.9</td>
</tr>
<tr>
<td>NTemp+AR</td>
<td>34.8</td>
<td>7.59</td>
<td>38.6</td>
</tr>
<tr>
<td>Seq2seq (Liu et al., 2018)</td>
<td>43.65</td>
<td>-</td>
<td>40.32</td>
</tr>
</tbody>
</table>

Table 3: Impact of varying the template $z(i)$ for a single $x$ from the E2E validation data; generations are annotated with the segmentations of the chosen $z(i)$. Results were obtained using the NTemp+AR model from Table 1.

```latex
\begin{align*}
\text{Travellers Rest Beefeater} & \\
\{\text{name}\} & \{\text{Travellers Rest Beefeater}\}, \{\text{customerRating}\} & \{3 \text{out of 5}\}, \{\text{area}\} & \{\text{riverside}\}, \{\text{near}\} & \{\text{Raja Indian Cuisine}\} & \\
1. \{\text{Travellers Rest Beefeater}\} & \{3 \text{star}\} & \{\text{restaurant}\} & \{1\} & \{\text{located near}\} & \{\text{Raja Indian Cuisine}\} & \{40 \text{ out of 5}\} & \{\text{rated}\} & \{\text{located near}\} & \{\text{Raja Indian Cuisine}\} & \{40 \text{ out of 5}\} & \\
2. \{\text{Near}\} & \{\text{riverside}\} & \{3\} & \{\text{Travellers Rest Beefeater}\} & \{3 \text{star}\} & \{\text{food}\} & \{1\} & \{2\} & \{\text{It is}\} & \{\text{near}\} & \{\text{Raja Indian Cuisine}\} & \{40 \text{ out of 5}\} & \{\text{rated}\} & \{\text{located near}\} & \{\text{Raja Indian Cuisine}\} & \{40 \text{ out of 5}\} & \\
3. \{\text{Travellers Rest Beefeater}\} & \{3 \text{star}\} & \{\text{restaurant}\} & \{1\} & \{\text{located near}\} & \{\text{Raja Indian Cuisine}\} & \{40 \text{ out of 5}\} & \{\text{rated}\} & \{\text{located near}\} & \{\text{Raja Indian Cuisine}\} & \{40 \text{ out of 5}\} & \\
4. \{\text{Travellers Rest Beefeater}\} & \{3 \text{star}\} & \{\text{place to eat}\} & \{12\} & \{\text{located near}\} & \{\text{Raja Indian Cuisine}\} & \{40 \text{ out of 5}\} & \{\text{rated}\} & \{\text{located near}\} & \{\text{Raja Indian Cuisine}\} & \{40 \text{ out of 5}\} & \\
5. \{\text{Travellers Rest Beefeater}\} & \{3 \text{star}\} & \{\text{restaurant}\} & \{1\} & \{\text{located near}\} & \{\text{Raja Indian Cuisine}\} & \{40 \text{ out of 5}\} & \{\text{rated}\} & \{\text{located near}\} & \{\text{Raja Indian Cuisine}\} & \{40 \text{ out of 5}\} & \\
\end{align*}
```

7.1 Qualitative Evaluation

We now qualitatively demonstrate that our generations are controllable and interpretable.

**Controllable Diversity** One of the powerful aspects of the proposed approach to generation is that we can manipulate the template $z(i)$ while leaving the database $x$ constant, which allows for easily controlling aspects of the generation. In Table 3 we show the generations produced by our model for five different neural template sequences $z(i)$, while fixing $x$. There, the segments in each generation are annotated with the latent states determined by the corresponding $z(i)$. We see that these templates can be used to affect the word-ordering, as well as which fields are mentioned in the generated text. Moreover, because the discrete states align with particular fields (see below), it is generally simple to automatically infer to which fields particular latent states correspond, allowing users to choose which template best meets their requirements. We emphasize that this level of controllability is much harder to obtain for encoder-decoder models, since, at best, a large amount of sampling would be required to avoid generating around a particular mode in the conditional distribution, and even then it would be difficult to control the sort of generations obtained.
Table 4 shows the impact of varying the template produced by the system, since each segment is explicitly typed by the current hidden state of the model. Table 4 shows the impact of varying the template produced by the system, since each segment is explicitly typed by the current hidden state of the model.

Table 4: Impact of varying the template \( z^{(i)} \) for a single \( x \) from the WikiBio validation data; generations are annotated with the segmentations of the chosen \( z^{(i)} \). Results were obtained using the NTemp model from Table 2.

<table>
<thead>
<tr>
<th></th>
<th>NTemp</th>
<th>NTemp+AR</th>
</tr>
</thead>
<tbody>
<tr>
<td>E2E</td>
<td>89.2 (17.4)</td>
<td>85.4 (18.6)</td>
</tr>
<tr>
<td>WikiBio</td>
<td>43.2 (19.7)</td>
<td>39.9 (17.9)</td>
</tr>
</tbody>
</table>

Table 5: Empirical analysis of the average purity of discrete states learned on the E2E and WikiBio datasets, for the NTemp and NTemp+AR models. Average purities are given as percents, and standard deviations follow in parentheses. See the text for full description of this calculation.

8 Conclusion and Future Work

We have developed a neural, template-like generation model based on an HSMM decoder, which can be learned tractably by backpropagating through a dynamic program. The method allows us to extract template-like latent objects in a principled way in the form of state sequences, and then generate with them. This approach scales to large-scale text datasets and is nearly competitive with encoder-decoder models. More importantly, this approach allows for controlling the diversity of generation and for producing interpretable states during generation. We view this work both as the first step towards learning discrete latent variable template models for more difficult generation tasks, as well as a different perspective on learning latent variable text models in general. Future work will examine encouraging the model to learn maximally different (or minimal) templates, which our objective does not explicitly encourage, templates of larger textual phenomena, such as paragraphs and documents, and hierarchical templates.

Acknowledgments

SW gratefully acknowledges the support of a Siebel Scholars award. AMR gratefully acknowledges the support of NSF CCF-1704834, Intel Research, and Amazon AWS Research grants.
References


Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In ICLR.


Dan Jurafsky and James H Martin. 2014. Speech and language processing. Pearson London.


A Supplemental Material

A.1 Additional Model and Training Details

Computing $r_j$ A record $r_j$ is represented by embedding a feature for its type, its position, and its word value in $\mathbb{R}^d$, and applying an MLP with ReLU nonlinearity \cite{NairH10} to form $r_j \in \mathbb{R}^d$, similar to Yang et al. \cite{YangBDL16} and Wiseman et al. \cite{WisemanBD17}.

LSTM Details The initial cell and hidden-state values for the decoder LSTM are given by $Q_1 x_a$ and $\tanh(Q_2 x_a)$, respectively, where $Q_1, Q_2 \in \mathbb{R}^{d \times d}$.

When a word $y_t$ appears in a record $r_j$, the input to the LSTM at time $t+1$ is computed using an MLP with ReLU nonlinearity over the concatenation of the embeddings for $r_j$’s record type, word value, position, and a feature for whether it is the final position for the type. If there are multiple $r_j$ from which $y_t$ could have been copied, the computed representations are averaged. At test time, we use the MAP $r_j$ to compute the input, even if there are multiple matches. For $y_t$ which could not have been copied, the input to the LSTM at time $t+1$ is computed using the same MLP over $y_t$ and three dummy features.

For the autoregressive HSMM, an additional 1-layer LSTM with $d$ hidden units is used. We experimented with having the autoregressive HSMM consume either tokens $y_{1:t}$ in predicting $y_{t+1}$, or the average embedding of the field types corresponding to copied tokens in $y_{1:t}$. The former worked slightly better for the WikiBio dataset (where field types are more ambiguous), while the latter worked slightly better for the E2E dataset.

Transition Distribution The function $C(x_u)$, which produces hidden state embeddings conditional on the source, is defined as $C(x_u) = U_2(\text{ReLU}(U_1 x_u))$, where $U_1 \in \mathbb{R}^{m_3 \times d}$ and $U_2 \in \mathbb{R}^{K \times m_2 \times m_3}$; $D(x)$ is defined analogously. For all experiments, $m_1 = 64$, $m_2 = 32$, and $m_3 = 64$.

Optimization We train with SGD, using a learning rate of 0.5 and decaying by 0.5 each epoch after the first epoch in which validation log-likelihood fails to increase. When using an autoregressive HSMM, the additional LSTM is optimized only after the learning rate has been decayed. We regularize with Dropout \cite{SrivastavaHKMS14}.

A.2 Additional Learned Templates

In Tables 6 and 7 we show visualizations of additional templates learned on the E2E and WikiBio data, respectively, by both the non-autoregressive and autoregressive HSMM models presented in the paper. For each model, we select a set of five dissimilar templates in an iterative way by greedily selecting the next template (out of the 200 most frequent) that has the highest percentage of states that do not appear in the previously selected templates; ties are broken randomly. Individual states within a template are visualized using the three most common segments they generate.
<table>
<thead>
<tr>
<th>Template</th>
<th>E2E Model (NTemp)</th>
<th>NTemp+AR Model (bottom)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>The Waterman</td>
<td>The Golden Palace</td>
</tr>
<tr>
<td></td>
<td>is a pub</td>
<td>is an English pub</td>
</tr>
<tr>
<td></td>
<td>located north of the city centre</td>
<td>located on the centre of the city</td>
</tr>
<tr>
<td></td>
<td>that serves</td>
<td>that provides</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fast food</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sushi</td>
</tr>
<tr>
<td></td>
<td></td>
<td>take-away deliveries</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>There is a cheap restaurant</td>
<td>There is an Italian restaurant</td>
</tr>
<tr>
<td></td>
<td>French restaurant</td>
<td>located in the centre of the city</td>
</tr>
<tr>
<td></td>
<td>located in the Twenty Two</td>
<td>that serves</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fast food</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sushi</td>
</tr>
<tr>
<td></td>
<td></td>
<td>take-away deliveries</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>The Olive Grove</td>
<td>The Cambridge Blue</td>
</tr>
<tr>
<td></td>
<td>restaurant</td>
<td>serves</td>
</tr>
<tr>
<td></td>
<td>pub</td>
<td>fast food</td>
</tr>
<tr>
<td></td>
<td></td>
<td>take-away deliveries</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>The Child friendly restaurant</td>
<td>The average priced restaurant</td>
</tr>
<tr>
<td></td>
<td>French restaurant</td>
<td>located near</td>
</tr>
<tr>
<td></td>
<td></td>
<td>near</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>The Strada Alimentum</td>
<td>The Dumping Tree</td>
</tr>
<tr>
<td></td>
<td>serves</td>
<td>provides</td>
</tr>
<tr>
<td></td>
<td>Indian</td>
<td>Chinese</td>
</tr>
<tr>
<td></td>
<td>food in the city centre</td>
<td>English</td>
</tr>
<tr>
<td></td>
<td>and has a</td>
<td>food</td>
</tr>
<tr>
<td></td>
<td>customer rating of 1 out of 5</td>
<td>price range of average 5 out of 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6: Five templates extracted from the E2E data with the NTemp model (top) and the NTemp+AR model (bottom).
Table 7: Five templates extracted from the WikiBio data with the NTemp model (top) and the Ntemp+AR model (bottom).