Online Segment to Segment Neural Transduction

Lei Yu[1], **Jan Buys**[1] and **Phil Blunsom**[1,2]
[1]University of Oxford
[2]DeepMind
{lei.yu, jan.buys, phil.blunsom}@cs.ox.ac.uk

Abstract

We introduce an online neural sequence to sequence model that learns to alternate between encoding and decoding segments of the input as it is read. By independently tracking the encoding and decoding representations our algorithm permits exact polynomial marginalization of the latent segmentation during training, and during decoding beam search is employed to find the best alignment path together with the predicted output sequence. Our model tackles the bottleneck of vanilla encoder-decoders that have to read and memorize the entire input sequence in their fixed-length hidden states before producing any output. It is different from previous attentive models in that, instead of treating the attention weights as output of a deterministic function, our model assigns attention weights to a sequential latent variable which can be marginalized out and permits online generation. Experiments on abstractive sentence summarization and morphological inflection show significant performance gains over the baseline encoder-decoders.

1 Introduction

The problem of mapping from one sequence to another is an importance challenge of natural language processing. Common applications include machine translation and abstractive sentence summarisation. Traditionally this type of problem has been tackled by a combination of hand-crafted features, alignment models, segmentation heuristics, and language models, all of which are tuned separately.

The recently introduced encoder-decoder paradigm has proved very successful for machine translation, where an input sequence is encoded into a fixed-length vector and an output sequence is then decoded from said vector (Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014; Cho et al., 2014). This architecture is appealing, as it makes it possible to tackle the problem of sequence-to-sequence mapping by training a large neural network in an end-to-end fashion. However it is difficult for a fixed-length vector to memorize all the necessary information of an input sequence, especially for long sequences. Often a very large encoding needs to be employed in order to capture the longest sequences, which invariably wastes capacity and computation for short sequences. While the attention mechanism of Bahdanau et al. (2015) goes some way to address this issue, it still requires the full input to be seen before any output can be produced.

In this paper we propose an architecture to tackle the limitations of the vanilla encoder-decoder model, a segment to segment neural transduction model (SSNT) that learns to generate and align simultaneously. Our model is inspired by the HMM word alignment model proposed for statistical machine translation (Vogel et al., 1996; Tillmann et al., 1997); we impose a monotone restriction on the alignments but incorporate recurrent dependencies on the input which enable rich locally non-monotone alignments to be captured. This is similar to the sequence transduction model of Graves (2012), but we propose alignment distributions which are parameterised separately, making the model more flexible

Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pages 1307–1316,
Austin, Texas, November 1-5, 2016.

and allowing online inference.

Our model introduces a latent segmentation which determines correspondences between tokens of the input sequence and those of the output sequence. The aligned hidden states of the encoder and decoder are used to predict the next output token and to calculate the transition probability of the alignment. We carefully design the input and output RNNs such that they independently update their respective hidden states. This enables us to derive an exact dynamic programme to marginalize out the hidden segmentation during training and an efficient beam search to generate online the best alignment path together with the output sequence during decoding. Unlike previous recurrent segmentation models that only capture dependencies in the input (Graves et al., 2006; Kong et al., 2016), our segmentation model is able to capture unbounded dependencies in both the input and output sequences while still permitting polynomial inference.

While attentive models treat the attention weights as output of a deterministic function, our model assigns attention weights to a sequential latent variable which can be marginalized out. Our model is general and could be incorporated into any RNN-based encoder-decoder architecture, such as Neural Turing Machines (Graves et al., 2014), memory networks (Weston et al., 2015; Kumar et al., 2016) or stack-based networks (Grefenstette et al., 2015), enabling such models to process data online.

We conduct experiments on two different transduction tasks, abstractive sentence summarisation (sequence to sequence mapping at word level) and morphological inflection generation (sequence to sequence mapping at character level). We evaluate our proposed algorithms in both the online setting, where the input is encoded with a unidirectional LSTM, and where the whole input is available such that it can be encoded with a bidirectional network. The experimental results demonstrate the effectiveness of SSNT — it consistently output performs the baseline encoder-decoder approach while requiring significantly smaller hidden layers, thus showing that the segmentation model is able to learn to break one large transduction task into a series of smaller encodings and decodings. When bidirectional encodings are used the segmentation model outperforms an attention-based benchmark. Qualitative analysis shows that the alignments found by our model are highly intuitive and demonstrates that the model learns to read ahead the required number of tokens before producing output.

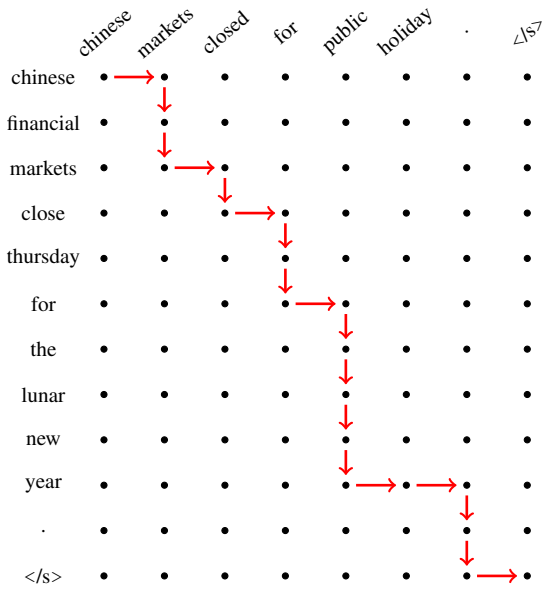

Figure 1: Example output of our recurrent segmentation model on the task of abstractive sentence summarisation. The path highlighted is the alignment found by the model during decoding.

2 Model

Let $\boldsymbol{x}_1^I$ be the input sequence of length I and $\boldsymbol{y}_1^J$ the output sequence of length J. Let y_j denote the j-th token of $\boldsymbol{y}$. Our goal is to model the conditional distribution

$$p(\boldsymbol{y}|\boldsymbol{x}) = \prod_{j=1}^{J} p(y_j|\boldsymbol{y}_1^{j-1}, \boldsymbol{x}). \quad (1)$$

We introduce a hidden alignment sequence $\boldsymbol{a}_1^J$ where each $a_j = i$ corresponds to an input position $i \in \{1, \ldots, I\}$ that we want to focus on when generating y_j. Then $p(\boldsymbol{y}|\boldsymbol{x})$ is calculated by marginalizing over all the hidden alignments,

$$
\begin{aligned}
& p(\boldsymbol{y}|\boldsymbol{x}) = \textstyle\sum_{\boldsymbol{a}} p(\boldsymbol{y}, \boldsymbol{a}|\boldsymbol{x}) \qquad (2)\\
\approx \quad & \textstyle\sum_{\boldsymbol{a}} \prod_{j=1}^{J} \underbrace{p(a_j|a_{j-1}, \boldsymbol{y}_1^{j-1}, \boldsymbol{x})}_{\text{transition probability}} \cdot \\
& \qquad \underbrace{p(y_j|\boldsymbol{y}_1^{j-1}, a_j, \boldsymbol{x})}_{\text{word prediction}}.
\end{aligned}
$$

Figure 1 illustrates the model graphically. Each path from the top left node to the right-most column in the graph corresponds to an alignment. We constrain the alignments to be monotone, i.e. only forward and downward transitions are permitted at each point in the grid. This constraint enables the model to learn to perform online generation. Additionally, the model learns to align input and output segments, which means that it can learn local reorderings by memorizing phrases. Another possible constraint on the alignments would be to ensure that the entire input sequence is consumed before last output word is emitted, i.e. all valid alignment paths have to end in the bottom right corner of the grid. However, we do not enforce this constraint in our setup.

The probability contributed by an alignment is obtained by accumulating the probability of word predictions at each point on the path and the transition probability between points. The transition probabilities and the word output probabilities are modeled by neural networks, which are described in detail in the following sub-sections.

2.1 Probabilities of Output Word Predictions

The input sentence $\boldsymbol{x}$ is encoded with a Recurrent Neural Network (RNN), in particular an LSTM (Hochreiter and Schmidhuber, 1997). The encoder can either be a unidirectional or bidirectional LSTM. If a unidirectional encoder is used the model is able to read input and generate output symbols online. The hidden state vectors are computed as

$$\mathbf{h}_i^{\rightarrow} = \text{RNN}(\mathbf{h}_{i-1}^{\rightarrow}, v^{(e)}(x_i)), \qquad (3)$$

$$\mathbf{h}_i^{\leftarrow} = \text{RNN}(\mathbf{h}_{i+1}^{\leftarrow}, v^{(e)}(x_i)), \qquad (4)$$

where $v^{(e)}(x_i)$ denotes the vector representation of the token x, and $\mathbf{h}_i^{\rightarrow}$ and $\mathbf{h}_i^{\leftarrow}$ are the forward and backward hidden states, respectively. For a bidirectional encoder, they are concatenated as $\mathbf{h}_i = [\mathbf{h}_i^{\rightarrow}; \mathbf{h}_i^{\leftarrow}]$; and for unidirectional encoder $\mathbf{h}_i = \mathbf{h}_i^{\rightarrow}$. The hidden state $\mathbf{s}_j$ of the RNN for the output sequence $\boldsymbol{y}$ is computed as

$$\mathbf{s}_j = \text{RNN}(\mathbf{s}_{j-1}, v^{(d)}(y_{j-1})), \qquad (5)$$

where $v^{(d)}(y_{j-1})$ is the encoded vector of the previously generated output word y_{j-1}. That is, $\mathbf{s}_j$ encodes $\boldsymbol{y}_1^{j-1}$.

To calculate the probability of the next word, we concatenate the aligned hidden state vectors $\mathbf{s}_j$ and $\mathbf{h}_{a_j}$ and feed the result into a softmax layer,

$$
\begin{aligned}
& p(y_j = l|\boldsymbol{y}_1^{j-1}, a_j, \boldsymbol{x}) \\
&= p(y_j = l|\mathbf{h}_{a_j}, \mathbf{s}_j) \qquad (6)\\
&= \text{softmax}(\mathbf{W}_w[\mathbf{h}_{a_j}; \mathbf{s}_j] + \mathbf{b}_w)_l.
\end{aligned}
$$

The word output distribution in Graves (2012) is parameterised in similar way.

Figure 2 illustrates the model structure. Note that the hidden states of the input and output decoders are kept independent to permit tractable inference, while the output distributions are conditionally dependent on both.

2.2 Transition Probabilities

As the alignments are constrained to be monotone, we can treat the transition from timestep j to $j+1$ as a sequence of `shift` and `emit` operations. Specifically, at each input position, a decision of `shift` or `emit` is made by the model; if the operation is `emit` then the next output word is generated; otherwise, the model will `shift` to the next input word. While the multinomial distribution is an alternative for parameterising alignments, the shift/emit parameterisation does not place an upper limit on the jump size, as a multinomial distribution would, and biases the model towards shorter jump sizes, which a multinomial model would have to learn.

We describe two methods for modelling the alignment transition probability. The first approach is independent of the input or output words. To parameterise the alignment distribution in terms of shift and emit operations we use a geometric distribution,

$$p(a_j|a_{j-1}) = (1-e)^{a_j - a_{j-1}} e, \qquad (7)$$

where e is the emission probability. This transition probability only has one parameter e, which can be

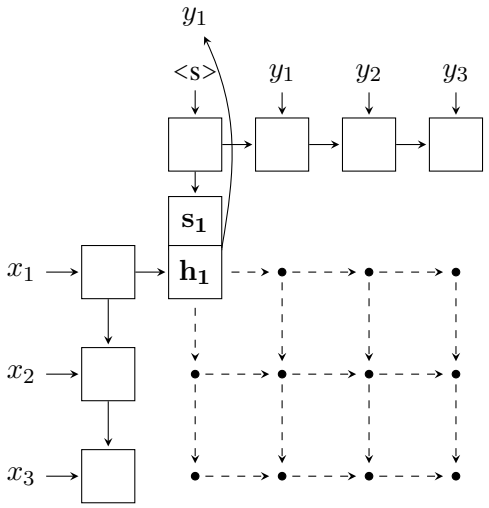

Figure 2: The structure of our model. (x_1, x_2, x_3) and (y_1, y_2, y_3) denote the input and output sequences, respectively. The points, e.g. (i, j), in the grid represent an alignment between x_i and y_j. For each column j, the concatenation of the hidden states $[\mathbf{h}_i, \mathbf{s}_j]$ is used to predict y_j.

estimated directly by maximum likelihood as

$$e = \frac{\sum_n J_n}{\sum_n I_n + \sum_n J_n}, \tag{8}$$

where I_n and J_n are the lengths of the input and output sequences of training example n, respectively.

For the second method we model the transition probability with a neural network,

$$\begin{aligned} p(a_1 = i) &= \prod_{d=1}^{i-1}(1 - p(e_{d,1}))p(e_{i,1}), \\ p(a_j = i | a_{j-1} = k) &= \prod_{d=k}^{i-1}(1 - p(e_{d,j}))p(e_{i,j}), \end{aligned} \tag{9}$$

where $p(e_{i,j})$ denotes the probability of `emit` for the alignment $a_j = i$. This probability is obtained by feeding $[\mathbf{h}_i; \boldsymbol{s}_j]$ into a feed forward neural network,

$$p(e_{i,j}) = \sigma(\text{MLP}(\mathbf{W}_t[\mathbf{h}_i; \mathbf{s}_j] + b_t)). \tag{10}$$

For simplicity, $p(a_j = i | a_{j-1} = k, \mathbf{s}_j, \mathbf{h}_k^i)$ is abbreviated as $p(a_j = i | a_{j-1} = k)$.

3 Training and Decoding

Since there are an exponential number of possible alignments, it is computationally intractable to explicitly calculate every $p(\boldsymbol{y}, \boldsymbol{a}|\boldsymbol{x})$ and then sum them to get the conditional probability $p(\boldsymbol{y}|\boldsymbol{x})$. We instead approach the problem using a dynamic-programming algorithm similar to the forward-backward algorithm for HMMs (Rabiner, 1989).

3.1 Training

For an input $\boldsymbol{x}$ and output $\boldsymbol{y}$, the forward variable $\alpha(i, j) = p(a_j = i, \boldsymbol{y}_1^j | \boldsymbol{x})$. The value of $\alpha(i, j)$ is computed by summing over the probabilities of every path that could lead to this cell. Formally, $\alpha(i, j)$ is defined as follows:

For $i \in [1, I]$:

$$\alpha(i, 1) = p(a_1 = i) p(y_1 | \mathbf{h}_i, \mathbf{s}_1). \tag{11}$$

For $j \in [2, J]$, $i \in [1, I]$:

$$\begin{aligned} \alpha(i, j) = p(y_j | \mathbf{h}_i, \mathbf{s}_j) \cdot \\ \sum_{k=1}^{i} \alpha(k, j-1) p(a_j = i | a_{j-1} = k). \end{aligned} \tag{12}$$

The backward variables, defined as $\beta(i, j) = p(\boldsymbol{y}_{j+1}^J | a_j = i, \boldsymbol{y}_1^j, \boldsymbol{x})$, are computed as:

For $i \in [1, I]$:

$$\beta(i, J) = 1. \tag{13}$$

For $j \in [1, J-1]$, $i \in [1, I]$:

$$\begin{aligned} \beta(i, j) = \sum_{k=i}^{I} p(a_{j+1} = k | a_j = i) \beta(k, j+1) \cdot \\ p(y_{j+1} | \mathbf{h}_k, \mathbf{s}_{j+1}). \end{aligned} \tag{14}$$

During training we estimate the parameters by minimizing the negative log likelihood of the training set S:

$$\begin{aligned} \mathcal{L}(\boldsymbol{\theta}) &= -\sum_{(\boldsymbol{x}, \boldsymbol{y}) \in S} \log p(\boldsymbol{y} | \boldsymbol{x}; \boldsymbol{\theta}) \\ &= -\sum_{(\boldsymbol{x}, \boldsymbol{y}) \in S} \log \sum_{i=1}^{I} \alpha(i, J). \end{aligned} \tag{15}$$

Let $\boldsymbol{\theta_j}$ be the neural network parameters w.r.t. the model output at position j. The gradient is computed as:

$$\frac{\partial \log p(\boldsymbol{y} | \boldsymbol{x}; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_{j=1}^{J} \sum_{i=1}^{I} \frac{\partial \log p(\boldsymbol{y} | \boldsymbol{x}; \boldsymbol{\theta})}{\partial \alpha(i, j)} \frac{\partial \alpha(i, j)}{\partial \boldsymbol{\theta_j}}. \tag{16}$$

The derivative w.r.t. the forward weights is

$$\frac{\partial \log p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})}{\partial \alpha(i,j)} = \frac{\beta(i,j)}{p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})}. \quad (17)$$

The derivative of the forward weights w.r.t. the model parameters at position j is

$$\frac{\partial \alpha(i,j)}{\partial \boldsymbol{\theta_j}} = \frac{\partial p(y_j|\mathbf{h}_i,\mathbf{s}_j)}{\partial \boldsymbol{\theta_j}} \frac{\alpha(i,j)}{p(y_j|\mathbf{h}_i,\mathbf{s}_j)} + p(y_j|\mathbf{h}_i,\mathbf{s}_j) \sum_{k=1}^{i} \alpha(j-1,k) \frac{\partial}{\partial \boldsymbol{\theta_j}} p(a_j{=}i|a_{j-1}{=}k). \quad (18)$$

For the geometric distribution transition probability model $\frac{\partial}{\partial \boldsymbol{\theta_j}} p(a_j = i|a_{j-1} = k) = 0$.

3.2 Decoding

Algorithm 1 DP search algorithm

Input: source sentence $\boldsymbol{x}$
Output: best output sentence $\boldsymbol{y*}$
Initialization: $Q \in \mathbb{R}^{I \times J_{\text{max}}}$, bp $\in \mathbb{N}^{I \times J_{\text{max}}}$, $W \in \mathbb{N}^{I \times J_{\text{max}}}$, I_{end}, J_{end}.
for $i \in [1, I]$ **do**
  $Q[i,1] \leftarrow \max_{y \in \mathcal{V}} p(a_1 = i) p(y|\mathbf{h}_i, \mathbf{s}_1)$
  $bp[i,1] \leftarrow 0$
  $W[i,1] \leftarrow \arg\max_{y \in \mathcal{V}} p(a_1 = i) p(y|\mathbf{h}_i, \mathbf{s}_1)$
end for
for $j \in [2, J_{\text{max}}]$ **do**
  for $i \in [1, I]$ **do**
    $Q[i,j] \leftarrow \max_{y \in \mathcal{V}, k \in [1,i]} Q[k, j-1] \cdot p(a_j = i|a_{j-1} = k) p(y|\mathbf{h}_i, \mathbf{s}_j)$
    $bp[i,j], W[i,j] \leftarrow \arg\max_{y \in \mathcal{V}, k \in [1,i]} \cdot Q[k, j-1] p(a_j = i|a_{j-1} = k) p(y|\mathbf{h}_i, \mathbf{s}_j)$
  end for
  $I_{\text{end}} \leftarrow \arg\max_i Q[i,j]$
  if $W[I_{\text{end}}, j] = \text{EOS}$ **then**
    $J_{\text{end}} \leftarrow j$
    break
  end if
end for
return a sequence of words stored in W by following backpointers starting from $(I_{\text{end}}, J_{\text{end}})$.

For decoding, we aim to find the best output sequence $\boldsymbol{y}^*$ for a given input sequence $\boldsymbol{x}$:

$$\boldsymbol{y}^* = \arg\max_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x}) \quad (19)$$

The search algorithm is based on dynamic programming (Tillmann et al., 1997). The main idea is to create a path probability matrix Q, and fill each cell $Q[i,j]$ by recursively taking the most probable path that could lead to this cell. We present the greedy search algorithm in Algorithm 1. We also implemented a beam search that tracks the k best partial sequences at position (i,j). The notation bp refers to backpointers, W stores words to be predicted, $\mathcal{V}$ denotes the output vocabulary, J_{max} is the maximum length of the output sequences that the model is allowed to predict.

4 Experiments

We evaluate the effectiveness of our model on two representative natural language processing tasks, sentence compression and morphological inflection. The primary aim of this evaluation is to assess whether our proposed architecture is able to outperform the baseline encoder-decoder model by overcoming its encoding bottleneck. We further benchmark our results against an attention model in order to determine whether our alternative alignment strategy is able to provide similar benefits while processing the input online.

4.1 Abstractive Sentence Summarisation

Sentence summarisation is the task of generating a condensed version of a sentence while preserving its meaning. In abstractive sentence summarisation, summaries are generated from the given vocabulary without the constraint of copying words in the input sentence. Rush et al. (2015) compiled a data set for this task from the annotated Gigaword data set (Graff et al., 2003; Napoles et al., 2012), where sentence-summary pairs are obtained by pairing the headline of each article with its first sentence. Rush et al. (2015) use the splits of 3.8m/190k/381k for training, validation and testing. In previous work on this dataset, Rush et al. (2015) proposed an attention-based model with feed-forward neural networks, and Chopra et al. (2016) proposed an attention-based recurrent encoder-decoder, similar to one of our baselines.

Due to computational constraints we place the following restrictions on the training and validation set:

1. The maximum lengths for the input sentences

Model	ROUGE-1	ROUGE-2	ROUGE-L
Seq2seq	25.16	9.09	23.06
Attention	29.25	12.85	27.32
uniSSNT	26.96	10.54	24.59
biSSNT	27.05	10.62	24.64
uniSSNT+	30.15	13.59	27.88
biSSNT+	**30.27**	**13.68**	**27.91**

Table 1: ROUGE F1 scores on the sentence summarisation test set. Seq2seq refers to the vanilla encoder-decoder and attention denotes the attention-based model. SSNT denotes our model with alignment transition probability modelled as geometric distribution. SSNT+ refers to our model with transition probability modelled using neural networks. The prefixes uni- and bi- denote using unidirectional and bidirectional encoder LSTMs, respectively.

Model	Configuration	Perplexity
Seq2seq	$H = 128, L = 1$	48.5
	$H = 256, L = 1$	35.6
	$H = 256, L = 2$	32.1
	$H = 256, L = 3$	31.0
biSSNT+	$H = 128, L = 1$	26.7
	$H = 256, L = 1$	**22.6**

Table 2: Perplexity on the validation set with 172k sentence-summary pairs.

and summaries are 50 and 25, respectively.

2. For each sentence-summary pair, the product of the input and output lengths should be no greater than 500.

We use the filtered 172k pairs for validation and sample 1m pairs for training. While this training set is smaller than that used in previous work (and therefore our results cannot be compared directly against reported results), it serves our purpose for evaluating our algorithm against sequence to sequence and attention-based approaches under identical data conditions. Following from previous work (Rush et al., 2015; Chopra et al., 2016; Gülçehre et al., 2016), we report results on a randomly sampled test set of 2000 sentence-summary pairs. The quality of the generated summaries are evaluated by three versions of ROUGE for different match lengths, namely ROUGE-1 (unigrams), ROUGE-2 (bigrams), and ROUGE-L (longest-common substring).

For training, we use Adam (Kingma and Ba, 2015) for optimization, with an initial learning rate of 0.001. The mini-batch size is set to 32. The number of hidden units H is set to 256 for both our model and the baseline models, and dropout of 0.2 is applied to the input of LSTMs. All hyperparameters were optimised via grid search on the perplexity of the validation set. We use greedy decoding to generate summaries.

Table 1 displays the ROUGE-F1 scores of our models on the test set, together with baseline models, including the attention-based model. Our models achieve significantly better results than the vanilla encoder-decoder and outperform the attention-based model. The fact that SSNT+ performs better is in line with our expectations, as the neural network-parameterised alignment model is more expressive than that modelled by geometric distribution.

To make further comparison, we experimented with different sizes of hidden units and adding more layers to the baseline encoder-decoder. Table 2 lists the configurations of different models and their corresponding perplexities on the validation set. We can see that the vanilla encoder-decoder tends to get better results by adding more hidden units and stacking more layers. This is due to the limitation of compressing information into a fixed-size vector. It has to use larger vectors and deeper structure in order to memorize more information. By contrast, our model can do well with smaller networks. In fact, even with 1 layer and 128 hidden units, our model works much better than the vanilla encoder-decoder with 3 layers and 256 hidden units per layer.

4.2 Morphological Inflection

Morphological inflection generation is the task of predicting the inflected form of a given lexical item based on a morphological attribute. The transformation from a base form to an inflected form usually includes concatenating it with a prefix or a suffix and substituting some characters. For example, the inflected form of a German stem *abgang* is *abgängen* when the case is dative and the number is plural.

In our experiments, we use the same dataset as

Model	Avg. accuracy
Seq2Seq	79.08
Seq2Seq w/ Attention	95.64
Adapted-seq2seq (FTND16)	**96.20**
uniSSNT+	87.85
biSSNT+	95.32

Table 3: Average accuracy over all the morphological inflection datasets. The baseline results for Seq2Seq variants are taken from (Faruqui et al., 2016).

Dataset	DDN13	NCK15	FTND16	biSSNT+
de-N	88.31	**88.60**	88.12	87.50
de-V	94.76	97.50	**97.72**	92.11
es-V	99.61	99.80	**99.81**	99.52
fi-NA	92.14	93.00	95.44	**95.48**
fi-V	97.23	98.10	97.81	**98.10**
fr-V	98.80	**99.20**	98.82	98.65
nl-V	90.50	96.10	96.71	95.90
Avg.	94.47	96.04	**96.20**	95.32

Table 4: Comparison of the performance of our model (biSSNT+) against the previous state-of-the-art on each morphological inflection dataset.

Faruqui et al. (2016). This dataset was originally created by Durrett and DeNero (2013) from Wiktionary, containing inflections for German nouns (de-N), German verbs (de-V), Spanish verbs (es-V), Finnish noun and adjective (fi-NA), and Finnish verbs (fi-V). It was further expanded by Nicolai et al. (2015) by adding Dutch verbs (nl-V) and French verbs (fr-V). The number of inflection types for each language ranges from 8 to 57. The number of base forms, i.e. the number of instances in each dataset, ranges from 2000 to 11200. The predefined split is 200/200 for dev and test sets, and the rest of the data for training.

Our model is trained separately for each type of inflection, the same setting as the factored model described in Faruqui et al. (2016). The model is trained to predict the character sequence of the inflected form given that of the stem. The output is evaluated by accuracies of string matching. For all the experiments on this task we use 128 hidden units for the LSTMs and apply dropout of 0.5 on the input and output of the LSTMs. We use Adam (Kingma and Ba, 2015) for optimisation with initial learning rate of 0.001. During decoding, beam search is employed with beam size of 30.

Table 3 gives the average accuracy of the uniSSNT+, biSSNT+, vanilla encoder-decoder, and attention-based models. The model with the best previous average result — denoted as adapted-seq2seq (FTND16) (Faruqui et al., 2016) — is also included for comparison. Our biSSNT+ model outperforms the vanilla encoder-decoder by a large margin and almost matches the state-of-the-art result on this task. As mentioned earlier, a characteristic of these datasets is that the stems and their corresponding inflected forms mostly overlap. Compare to the vanilla encoder-decoder, our model is better at copying and finding correspondences between prefix, stem and suffix segments.

Table 4 compares the results of biSSNT+ and previous models on each individual dataset. DDN13 and NCK15 denote the models of Durrett and DeNero (2013) and Nicolai et al. (2015), respectively. Both models tackle the task by feature engineering. FTND16 (Faruqui et al., 2016) adapted the vanilla encoder-decoder by feeding the i-th character of the encoded string as an extra input into the i-th position of the decoder. It can be considered as a special case of our model by forcing a fixed diagonal alignment between input and output sequences. Our model achieves comparable results to these models on all the datasets. Notably it outperforms other models on the Finnish noun and adjective, and verbs datasets, whose stems and inflected forms are the longest.

5 Alignment Quality

Figure 3 presents visualisations of segment alignments generated by our model for sample instances from both tasks. We see that the model is able to learn the correct correspondences between segments of the input and output sequences. For instance, the alignment follows a nearly diagonal path for the example in Figure 3c, where the input and output sequences are identical. In Figure 3b, it learns to add the prefix 'ge' at the start of the sequence and replace 'en' with 't' after copying 'zock'. We observe that the model is robust on long phrasal mappings. As

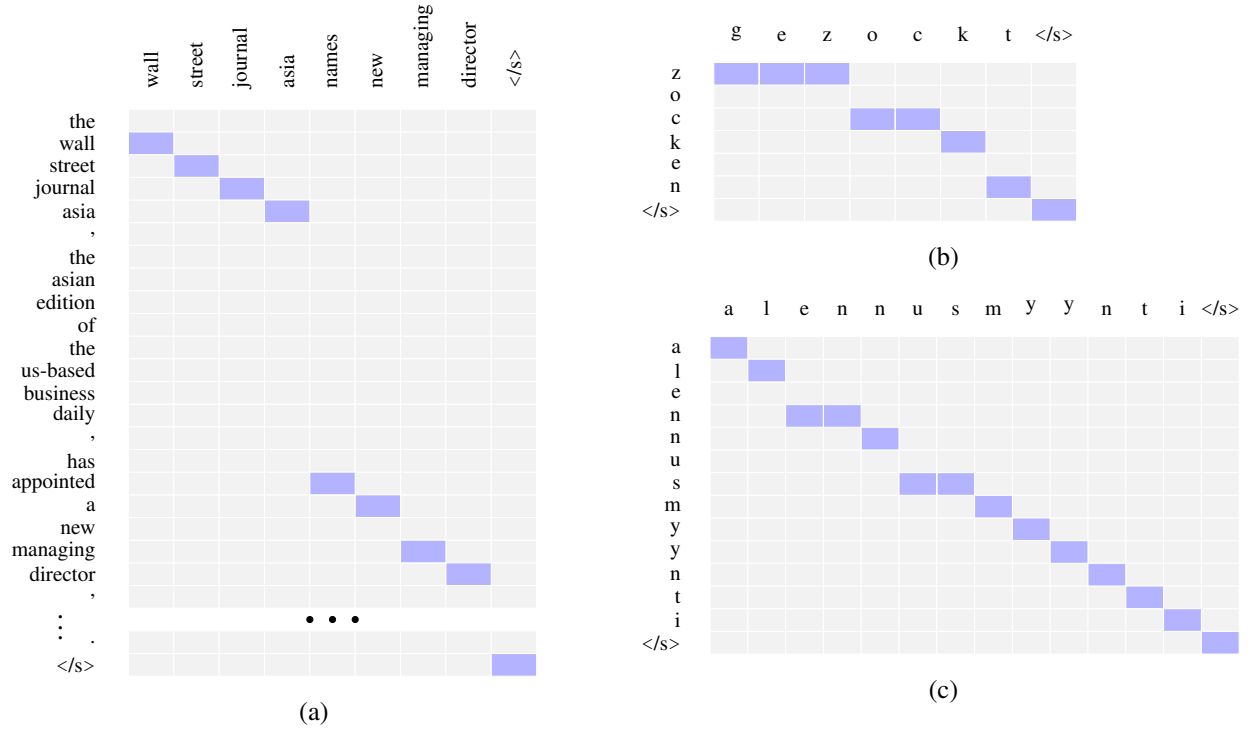

Figure 3: Example alignments found by BiSSNT+. Highlighted grid cells represent the correspondence between the input and output tokens.

shown in Figure 3a, the mapping between 'the wall street journal asia, the asian edition of the us-based business daily' and 'wall street journal asia' demonstrates that our model learns to ignore phrasal modifiers containing additional information. We also find some examples of word reordering, e.g., the phrase 'industrial production in france' is reordered as 'france industrial output' in the model's predicted output.

6 Related Work

Our work is inspired by the seminal HMM alignment model (Vogel et al., 1996; Tillmann et al., 1997) proposed for machine translation. In contrast to that work, when predicting a target word we additionally condition on all previously generated words, which is enabled by the recurrent neural models. This means that the model also functions as a conditional language model. It can therefore be applied directly, while traditional models have to be combined with a language model through a noisy channel in order to be effective. Additionally, instead of EM training on the most likely alignments at each iteration, our model is trained with direct gradient descent, marginalizing over all the alignments.

Latent variables have been employed in neural network-based models for sequence labelling tasks in the past. Examples include connectionist temporal classification (CTC) (Graves et al., 2006) for speech recognition and the more recent segmental recurrent neural networks (SRNNs) (Kong et al., 2016), with applications on handwriting recognition and part-of-speech tagging. Weighted finite-state transducers (WFSTs) have also been augmented to encode input sequences with bidirectional LSTMs (Rastogi et al., 2016), permitting exact inference over all possible output strings. While these models have been shown to achieve appealing performance on different applications, they have common limitations in terms of modelling dependencies between labels. It is not possible for CTCs to model explicit dependencies. SRNNs and neural WFSTs model fixed-length dependencies, making it is difficult to carry out effective inference as the dependencies become longer.

Our model shares the property of the sequence

transduction model of Graves (2012) in being able to model unbounded dependencies between output tokens via an output RNN. This property makes it possible to apply our model to tasks like summarisation and machine translation that require the tokens in the output sequence to be modelled highly dependently. Graves (2012) models the joint distribution over outputs and alignments by inserting null symbols (representing shift operations) into the output sequence. During training the model uses dynamic programming to marginalize over permutations of the null symbols, while beam search is employed during decoding. In contrast our model defines a separate latent alignment variable, which adds flexibility to the way the alignment distribution can be defined (as a geometric distribution or parameterised by a neural network) and how the alignments can be constrained, without redefining the dynamic program. In addition to marginalizing during training, our decoding algorithm also makes use of dynamic programming, allowing us to use either no beam or small beam sizes.

Our work is also related to the attention-based models first introduced for machine translation (Bahdanau et al., 2015). Luong et al. (2015) proposed two alternative attention mechanisms: a global method that attends all words in the input sentence, and a local one that points to parts of the input words. Another variation on this theme are pointer networks (Vinyals et al., 2015), where the outputs are pointers to elements of the variable-length input, predicted by the attention distribution. Jaitly et al. (2016) propose an online sequence to sequence model with attention that conditions on fixed-sized blocks of the input sequence and emits output tokens corresponding to each block. The model is trained with alignment information to generate supervised segmentations.

Although our model shares the same idea of joint training and aligning with the attention-based models, our design has fundamental differences and advantages. While attention-based models treat the attention weights as output of a deterministic function (soft-alignment), in our model the attention weights correspond to a hidden variable, that can be marginalized out using dynamic programming. Further, our model's inherent online nature permits it the flexibility to use its capacity to chose how much input to encode before decoding each segment.

7 Conclusion

We have proposed a novel segment to segment neural transduction model that tackles the limitations of vanilla encoder-decoders that have to read and memorize an entire input sequence in a fixed-length context vector before producing any output. By introducing a latent segmentation that determines correspondences between tokens of the input and output sequences, our model learns to generate and align jointly. During training, the hidden alignment is marginalized out using dynamic programming, and during decoding the best alignment path is generated alongside the predicted output sequence. By employing a unidirectional LSTM as encoder, our model is capable of doing online generation. Experiments on two representative natural language processing tasks, abstractive sentence summarisation and morphological inflection generation, showed that our model significantly outperforms encoder-decoder baselines while requiring much smaller hidden layers. For future work we would like to incorporate attention-based models to our framework to enable such models to process data online.

Acknowledgments

We thank Chris Dyer, Karl Moritz Hermann, Edward Grefenstette, Tomáš Kočiský, Gabor Melis, Yishu Miao and many others for their helpful comments. The first author is funded by EPSRC.

References

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

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of EMNLP*.

Sumit Chopra, Michael Auli, and Alexander M. Rush. 2016. Abstractive sentence summarization with attentive recurrent neural networks. In *Proceedings of NAACL*.

Greg Durrett and John DeNero. 2013. Supervised learning of complete morphological paradigms. In *Proceedings of HLT-NAACL*.

Manaal Faruqui, Yulia Tsvetkov, Graham Neubig, and Chris Dyer. 2016. Morphological inflection generation using character sequence to sequence learning. In *Proceedings of NAACL*.

David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. 2003. English gigaword. *Linguistic Data Consortium, Philadelphia*.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. 2006. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of ICML*.

Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural turing machines. *CoRR*, abs/1410.5401.

Alex Graves. 2012. Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711*.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. 2015. Learning to transduce with unbounded memory. In *Proceedings of NIPS*, pages 1819–1827.

Çaglar Gülçehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. 2016. Pointing the unknown words. *CoRR*, abs/1603.08148.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Navdeep Jaitly, David Sussillo, Quoc V. Le, Oriol Vinyals, Ilya Sutskever, and Samy Bengio. 2016. A neural transducer. In *Proceedings of NIPS*.

Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent continuous translation models. In *Proceedings of EMNLP*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of ICIR*.

Lingpeng Kong, Chris Dyer, and Noah A Smith. 2016. Segmental recurrent neural networks. In *Proceedings of ICLR*.

Ankit Kumar, Ozan Irsoy, Jonathan Su, James Bradbury, Robert English, Brian Pierce, Peter Ondruska, Ishaan Gulrajani, and Richard Socher. 2016. Ask me anything: Dynamic memory networks for natural language processing. In *Proceedings of ICML*.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of EMNLP*.

Courtney Napoles, Matthew Gormley, and Benjamin Van Durme. 2012. Annotated gigaword. In *Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction*.

Garrett Nicolai, Colin Cherry, and Grzegorz Kondrak. 2015. Inflection generation as discriminative string transduction. In *Proceedings of NAACL*.

Lawrence R Rabiner. 1989. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

Pushpendre Rastogi, Ryan Cotterell, and Jason Eisner. 2016. Weighting finite-state transductions with neural context. In *Proceedings of NAACL*.

Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. In *Proceedings of EMNLP*.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Proceedings of NIPS*.

Christoph Tillmann, Stephan Vogel, Hermann Ney, and Alex Zubiaga. 1997. A DP-based search using monotone alignments in statistical translation. In *Proceedings of EACL*.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Proceedings of NIPS*.

Stephan Vogel, Hermann Ney, and Christoph Tillmann. 1996. HMM-based word alignment in statistical translation. In *Proceedings of COLING*.

Jason Weston, Sumit Chopra, and Antoine Bordes. 2015. Memory networks. In *Proceedings of ICLR*.