A. Algorithms

Below are algorithms for the hard monotonic decoding process we used at test time (algorithm 1) and the approach for computing its expected output that we used to train the network (algorithm 2). Terminology matches the main text, except we use $\vec{0}$ to signify a vector of zeros.

Algorithm 1 Hard Monotonic Attention Process

Input: memory $\mathbf{h}$ of length T
State: $s_0 = \vec{0}, t_0 = 1, i = 1, y_0 = \text{StartOfSequence}$
while $y_{i-1} \neq \text{EndOfSequence}$ **do** *// Produce output tokens until end-of-sequence token is produced*
  $\text{finished} = 0$ *// Keep track of whether we chose a memory entry or not*
  for $j = t_{i-1}$ **to** T **do** *// Start inspecting memory entries h_j left-to-right from where we left off*
    $e_{i,j} = a(s_{i-1}, h_j)$ *// Compute attention energy for h_j*
    $p_{i,j} = \sigma(e_{i,j})$ *// Compute probability of choosing h_j*
    $z_{i,j} \sim \text{Bernoulli}(p_{i,j})$ *// Sample whether to ingest another memory entry or output new symbol*
    if $z_{i,j} = 1$ **then** *// If we sample 1, we stop scanning the memory*
      $c_i = h_j$ *// Set the context vector to the chosen memory entry*
      $t_i = j$ *// Remember where we left off for the next output timestep*
      $\text{finished} = 1$ *// Keep track of the fact that we chose a memory entry*
      break *// Stop scanning the memory*
    end if
  end for
  if $\text{finished} = 0$ **then**
    $c_i = \vec{0}$ *// If we scanned the entire memory without selecting anything, set c_i to a vector of zeros*
  end if
  $s_i = f(s_{i-1}, y_{i-1}, c_i)$ *// Update the state based on the new context vector using the RNN f*
  $y_i = g(s_i, c_i)$ *// Output a new symbol using the* softmax *layer g*
  $i = i + 1$
end while

Algorithm 2 Soft Monotonic Attention Decoder

Input: memory $\mathbf{h}$ of length T, target outputs $\hat{\mathbf{y}} = \{\text{StartOfSequence}, \hat{y}_1, \hat{y}_2, \ldots, \text{EndOfSequence}\}$
State: $s_0 = \vec{0}$, $i = 1$, $\alpha_{0,j} = \delta_j$ for $j \in \{1, \ldots, T\}$
while $\hat{y}_{i-1} \neq \text{EndOfSequence}$ **do** *// Produce output tokens until end of the target sequence*
  $p_{i,0} = 0, q_{i,0} = 0$ *// Special cases so that the recurrence relation matches eq. (9)*
  for $j = 1$ **to** T **do** *// Inspect all memory entries h_j*
    $e_{i,j} = a(s_{i-1}, h_j)$ *// Compute attention energy for h_j using eq. (16)*
    $e_{i,j} = e_{i,j} + \mathcal{N}(0, 1)$ *// Add pre-sigmoid noise to encourage $p_{i,j} \approx 0$ or $p_{i,j} \approx 1$*
    $p_{i,j} = \sigma(e_{i,j})$ *// Compute probability of choosing h_j*
    $q_{i,j} = (1 - p_{i,j-1})q_{i,j-1} + \alpha_{i-1,j}$ *// Iterate recurrence relation derived in eq. (10)*
    $\alpha_{i,j} = p_{i,j}q_{i,j}$ *// Compute the probability that $c_i = h_j$*
  end for
  $c_i = \sum_{j=1}^{T} \alpha_{i,j} h_j$ *// Compute weighted combination of memory for context vector*
  $s_i = f(s_{i-1}, y_{i-1}, c_i)$ *// Update the state based on the new context vector using the RNN f*
  $y_i = g(s_i, c_i)$ *// Compute predicted output for timestep i using the* softmax *layer g*
  $i = i + 1$
end while

B. Figures

Below are example hard monotonic and $\mathrm{softmax}$ attention alignments for each of the different tasks we included in our experiments. Attention matrices are displayed so that black corresponds to 1 and white corresponds to 0.

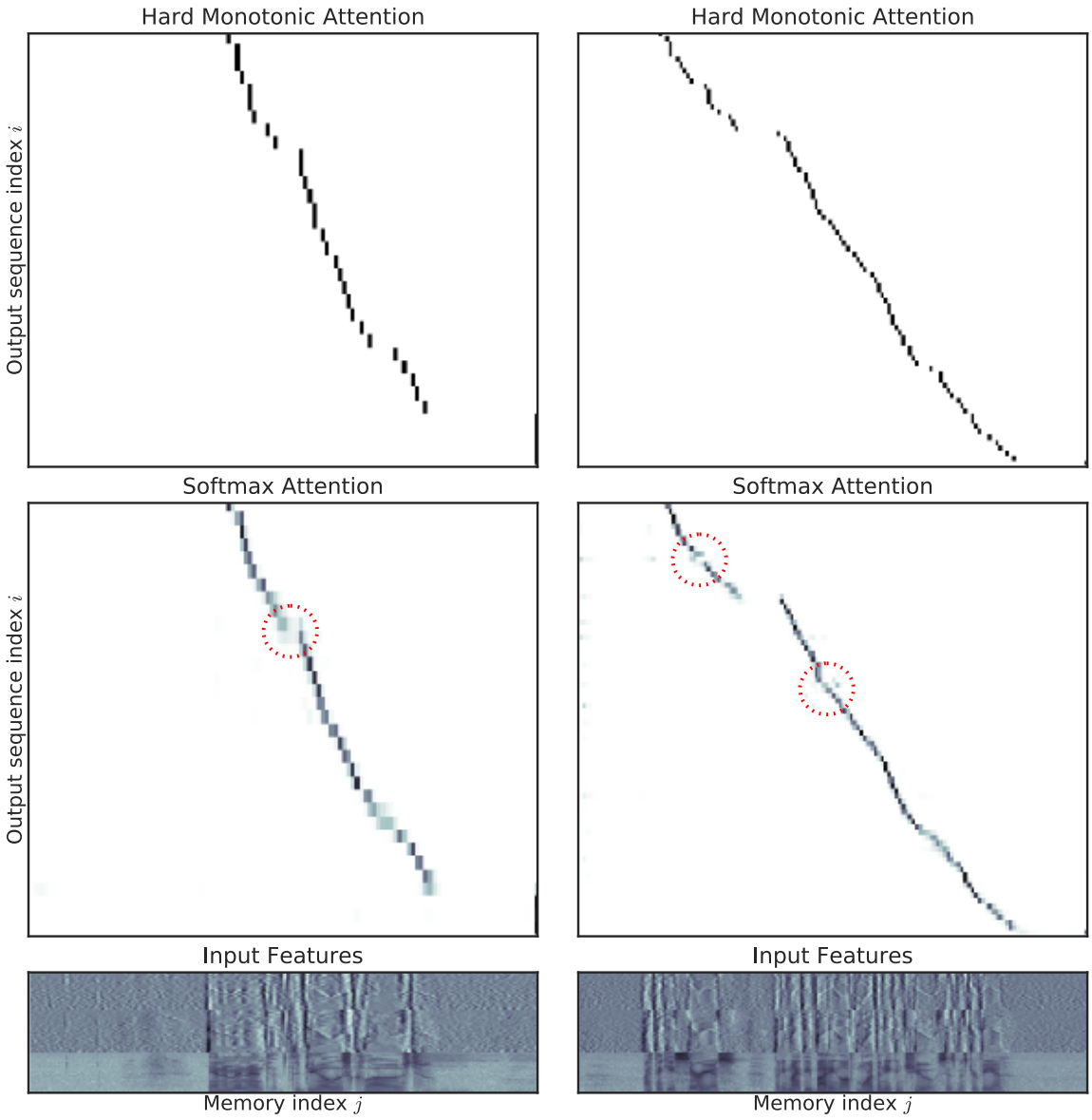

Figure 4. Attention alignments from hard monotonic attention and $\mathrm{softmax}$-based attention models for a two example speech utterances. From top to bottom, we show the hard monotonic alignment, the $\mathrm{softmax}$-attention alignment, and the utterance feature sequence. Differences in the alignments are highlighted with dashed red circles. Gaps in the alignment paths correspond to effectively ignoring silences and pauses in the speech utterances.

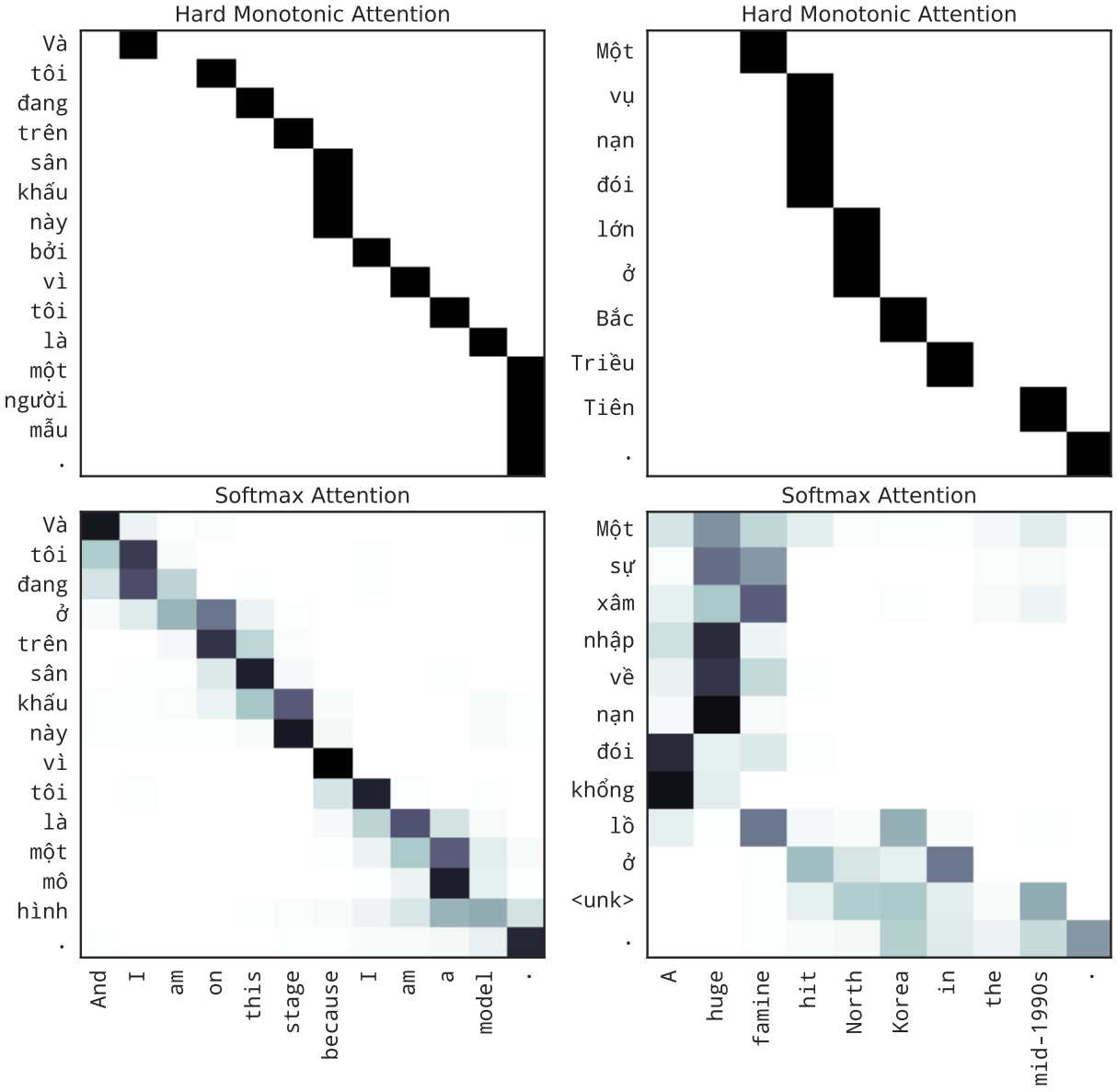

Figure 5. English sentences, predicted Vietnamese sentences, and input-output alignments for our proposed hard monotonic alignment model and the baseline model of (Luong & Manning, 2015). The Vietnamese model outputs for the left example can be translated back to English as "And I on this stage because I am a model." (monotonic) and "And I am on this stage because I am a structure." (softmax). The input word "model" can mean either a person or a thing; the monotonic alignment model correctly chose the former while the softmax alignment model chose the latter. The monotonic alignment model erroneously skipped the first verb in the sentence. For the right example, translations of the model outputs back to English are "A large famine in North Korea." (monotonic) and "An invasion of a huge famine in <unk>." (softmax). The monotonic alignment model managed to translate the proper noun North Korea, while the softmax alignment model produced <unk>. Both models skipped the phrase "mid-1990s"; this type of error is common in neural machine translation systems.

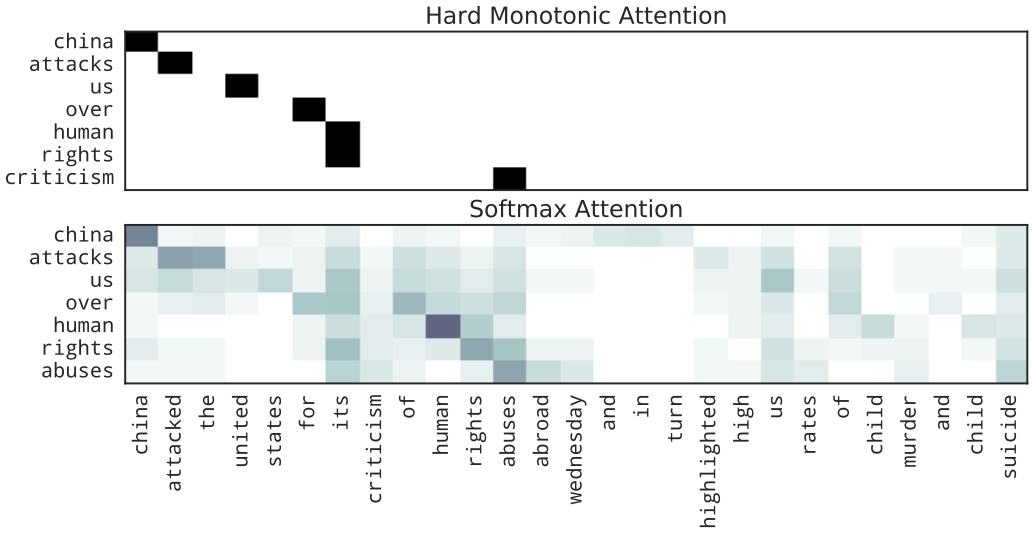

Figure 6. Additional example sentence-summary pair and attention alignment matrices for our hard monotonic model and the softmax-based attention model of (Liu & Pan, 2016). The ground-truth summary is "china attacks us human rights".

C. Monotonic Attention Distribution

Recall that our goal is to compute the expected value of c_i under the stochastic process defined by eqs. (6) to (8). To achieve this, we will derive an expression for the probability that $c_i = h_j$ for $j \in \{1, \dots, T\}$, which in accordance with eq. (2) we denote $\alpha_{i,j}$. For $i = 1$, $\alpha_{1,j}$ is the probability that memory element j was chosen ($p_{1,j}$) multiplied by the probability that memory elements $k \in \{1, 2, \dots, j-1\}$ were not chosen ($(1 - p_{i,k})$), giving

$$\alpha_{1,j} = p_{1,j} \prod_{k=1}^{j-1} (1 - p_{1,k}) \tag{18}$$

For $i > 0$, in order for $c_i = h_j$ we must have that $c_{i-1} = h_k$ for some $k \in \{1, \dots, j\}$ (which occurs with probability $\alpha_{i-1,k}$) and that none of $h_k, \dots, h_{j-1}$ were chosen. Summing over possible values of k, we have

$$\alpha_{i,j} = p_{i,j} \sum_{k=1}^{j} \left(\alpha_{i-1,k} \prod_{l=k}^{j-1} (1 - p_{i,l}) \right) \tag{19}$$

where for convenience we define $\prod_n^m x = 1$ when $n > m$. We provide a schematic and explanation of eq. (19) in fig. 7. Note that we can recover eq. (18) from eq. (19) by defining the special case $\alpha_{0,j} = \delta_j$ (i.e. $\alpha_{0,1} = 1$ and $\alpha_{0,j} = 0$ for $j \in \{2, \dots, T\}$). Expanding eq. (19) reveals we can compute $\alpha_{i,j}$ directly given $\alpha_{i-1,j}$ and $\alpha_{i,j-1}$:

$$\alpha_{i,j} = p_{i,j} \left(\sum_{k=1}^{j-1} \left(\alpha_{i-1,k} \prod_{l=k}^{j-1} (1 - p_{i,l}) \right) + \alpha_{i-1,j} \right) \tag{20}$$

$$= p_{i,j} \left((1 - p_{i,j-1}) \sum_{k=1}^{j-1} \left(\alpha_{i-1,k} \prod_{l=k}^{j-2} (1 - p_{i,l}) \right) + \alpha_{i-1,j} \right) \tag{21}$$

$$= p_{i,j} \left((1 - p_{i,j-1}) \frac{\alpha_{i,j-1}}{p_{i,j-1}} + \alpha_{i-1,j} \right) \tag{22}$$

Defining $q_{i,j} = \alpha_{i,j} / p_{i,j}$ produces eqs. (13) and (14). Equation (22) also has an intuitive interpretation: The expression $(1 - p_{i,j-1}) \alpha_{i,j-1} / p_{i,j-1}$ represents the probability that the model attended to memory item $j - 1$ at output timestep i, adjusted for the fact that memory item $j - 1$ was not chosen by multiplying $(1 - p_{i,j-1})$ and dividing $p_{i,j-1}$. Adding $\alpha_{i-1,j}$ reflects the additional possibility that the model attended to memory item j at the previous output timestep, and multiplying by $p_{i,j}$ enforces that memory item j was chosen at the current output timestep i.

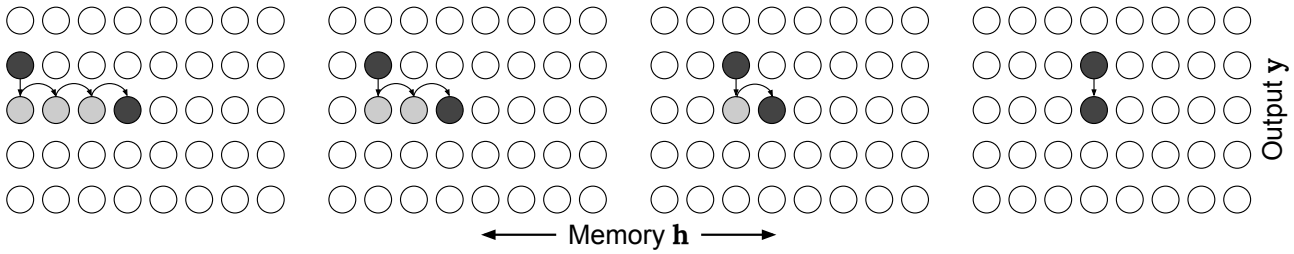

Figure 7. Visualization of eq. (19). In this example, we are showing the computation of $\alpha_{3,4}$. Each grid shows each of the four terms in the summation, corresponding to the possibilities that we attended to memory item $k = 1, 2, 3, 4$ at the previous output timestep $i - 1 = 2$. Gray nodes with curved arrows represent the probability of not selecting to the lth memory entry $(1 - p_{i,l})$. The black nodes represent the possibility of attending to memory item j at timestep i.

C.1. Recurrence Relation Solution

While eqs. (10) and (22) allow us to compute $\alpha_{i,j}$ directly from $\alpha_{i-1,j}$ and $\alpha_{i,j-1}$, the dependence on $\alpha_{i,j-1}$ means that we must compute the terms $\alpha_{i,1}, \alpha_{i,2}, \ldots, \alpha_{i,T}$ sequentially. This is in contrast to $\mathrm{softmax}$ attention, where these terms can be computed in parallel because they are independent. Fortunately, there is a solution to the recurrence relation of eq. (10) which allows the terms of α_i to be computed directly via parallelizable cumulative sum and cumulative product operations. Using eq. (13) which substitutes $q_{i,j} = \alpha_{i,j}/p_{i,j}$, we have

$$q_{i,j} = (1 - p_{i,j-1})q_{i,j-1} + \alpha_{i-1,j} \tag{23}$$

$$q_{i,j} - (1 - p_{i,j-1})q_{i,j-1} = \alpha_{i-1,j} \tag{24}$$

$$\frac{q_{i,j}}{\prod_{k=1}^{j}(1 - p_{i,k-1})} - \frac{(1 - p_{i,j-1})q_{i,j-1}}{\prod_{k=1}^{j}(1 - p_{i,k-1})} = \frac{\alpha_{i-1,j}}{\prod_{k=1}^{j}(1 - p_{i,k-1})} \tag{25}$$

$$\frac{q_{i,j}}{\prod_{k=1}^{j}(1 - p_{i,k-1})} - \frac{q_{i,j-1}}{\prod_{k=1}^{j-1}(1 - p_{i,k-1})} = \frac{\alpha_{i-1,j}}{\prod_{k=1}^{j}(1 - p_{i,k-1})} \tag{26}$$

$$\sum_{l=1}^{j}\left(\frac{q_{i,l}}{\prod_{k=1}^{l}(1 - p_{i,k-1})} - \frac{q_{i,l-1}}{\prod_{k=1}^{l-1}(1 - p_{i,k-1})}\right) = \sum_{l=1}^{j}\frac{\alpha_{i-1,l}}{\prod_{k=1}^{l}(1 - p_{i,k-1})} \tag{27}$$

$$\frac{q_{i,j}}{\prod_{k=1}^{j}(1 - p_{i,k-1})} - q_{i,0} = \sum_{l=1}^{j}\frac{\alpha_{i-1,l}}{\prod_{k=1}^{l}(1 - p_{i,k-1})} \tag{28}$$

$$q_{i,j} = \left(\prod_{k=1}^{j}(1 - p_{i,k-1})\right)\left(\sum_{l=1}^{j}\frac{\alpha_{i-1,l}}{\prod_{k=1}^{l}(1 - p_{i,k-1})}\right) \tag{29}$$

$$\Rightarrow q_i = \mathtt{cumprod}(1 - p_i)\mathtt{cumsum}\left(\frac{\alpha_{i-1}}{\mathtt{cumprod}(1 - p_i)}\right) \tag{30}$$

where $\mathtt{cumprod}(x) = [1, x_1, x_1x_2, \ldots, \prod_i^{|x|-1} x_i]$ and $\mathtt{cumsum}(x) = [x_1, x_1 + x_2, \ldots, \sum_i^{|x|} x_i]$. Note that we use the "exclusive" variant of $\mathtt{cumprod}$[4] in keeping with our defined special case $p_{i,0} = 0$. Unlike the recurrence relation of eq. (10), these operations can be computed efficiently in parallel (Ladner & Fischer, 1980). The primary disadvantage of this approach is that the product in the denominator of eq. (29) can cause numerical instabilities; we address this in appendix G.

[4]This can be computed e.g. in Tensorflow via `tf.cumprod(x, exclusive=True)`

D. Experiment Details

In this section, we give further details into the models and training procedures used in section 4. Any further questions about implementation details should be directed to the corresponding author. All models were implemented with TensorFlow (Abadi et al., 2016).

D.1. Speech Recognition

D.1.1. TIMIT

Mel filterbank features were standardized to have zero mean and unit variance across feature dimensions according to their training set statistics and were fed directly into an RNN encoder with three unidirectional LSTM layers, each with 512 hidden units. After the first and second LSTM layers, we downsampled hidden state sequences by skipping every other state before feeding into the subsequent layer. For the decoder, we used a single unidirectional LSTM layer with 256 units, fed directly into the output $\mathrm{softmax}$ layer. All weight matrices were initialized uniformly from $[-0.075, 0.075]$. The output tokens were embedded via a learned embedding matrix with dimensionality 30, initialized uniformly from $[-\sqrt{3/30}, \sqrt{3/30}]$. Our decoder attention energy function used a hidden dimensionality of 512, with the scalar bias r initialized to -1. The model was regularized by adding weight noise with a standard deviation of 0.5 after 2,000 training updates. L2 weight regularization was also applied with a weight of 10^{-6}.

We trained the network using Adam (Kingma & Ba, 2014), with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-6}$. Utterances were fed to the network with a minibatch size of 4. Our initial learning rate was 10^{-4}, which we halved after 40,000 training steps. We clipped gradients when their global norm exceeded 2. We used three training replicas. Beam search decoding was used to produce output sequences with a beam width of 10.

D.1.2. Wall Street Journal

The input 80 mel filterbank / delta / delta-delta features were organized as a $T \times 80 \times 3$ tensor, i.e. raw features, deltas, and delta-deltas are concatenated along the "depth" dimension. This was passed into a stack of two convolutional layers with ReLU activations, each consisting of 32 $3 \times 3 \times$ depth kernels in time $\times$ frequency. These were both strided by 2×2 in order to downsample the sequence in time, minimizing the computation performed in the following layers. Batch normalization (Ioffe & Szegedy, 2015) was applied prior to the ReLU activation in each layer. All encoder weight matrices and filters were initialized via a truncated Gaussian with zero mean and a standard deviation of 0.1.

This downsampled feature sequence was then passed into a single unidirectional convolutional LSTM layer using 1x3 filter (i.e. only convolving across the frequency dimension within each timestep). Finally, this was passed into a stack of three unidirectional LSTM layers of size 256, interleaved with a 256 dimensional linear projection, following by batch normalization, and a ReLU activation. Decoder weight matrices were initialized uniformly at random from $[-0.1, 0.1]$.

The decoder input is created by concatenating a 64 dimensional embedding corresponding to the symbol emitted at the previous timestep, and the 256 dimensional attention context vector. The embedding was initialized uniformly from $[-1, 1]$. This was passed into a single unidirectional LSTM layer with 256 units. We used an attention energy function hidden dimensionality of 128 and initialized the bias scalar r to -4. Finally the concatenation of the attention context and LSTM output is passed into the $\mathrm{softmax}$ output layer.

We applied label smoothing (Chorowski & Jaitly, 2017), replacing $\hat{y}_t$, the target at time t, with $(0.015\hat{y}_{t-2} + 0.035\hat{y}_{t-1} + \hat{y}_t + 0.035\hat{y}_{t+1} + 0.015\hat{y}_{t+2})/1.1$. We used beam search decoding at test time with rank pruning at 8 hypotheses and a pruning threshold of 3.

The network was trained using teacher forcing on minibatches of 8 input utterances, optimized using Adam (Kingma & Ba, 2014) with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-6}$. Gradients were clipped to a maximum global norm of 1. We set the initial learning rate to 0.0002 and decayed by a factor of 10 after 700,000, 1,000,000, and 1,300,000 training steps. L2 weight decay is used with a weight of 10^{-6}, and, beginning from step 20,000, Gaussian weight noise with standard deviation of 0.075 was added to weights for all LSTM layers and decoder embeddings. We trained using 16 replicas.

D.2. Sentence Summarization

For data preparation, we used the same Gigaword data processing scripts provided in (Rush et al., 2015) and tokenized into words by splitting on spaces. The vocabulary was determined by selecting the most frequent 200,000 tokens. Only the tokens of the first sentence of the article were used as input to the model. An embedding layer was used to embed tokens into a 200 dimensional space; embeddings were initialized using random normal distribution with mean 0 and standard deviation 10^{-4}.

We used a 4-layer bidirectional LSTM encoder with 4 layers and a single-layer unidirectional LSTM decoder. All LSTMs, and the attention energy function, had a hidden dimensionality of 256. The decoder LSTM was fed directly into the $\mathrm{softmax}$ output layer. All weights were initialized

uniform-randomly between -0.1 and 0.1. In our monotonic alignment decoder, we initialized r to -4. At test time, we used a beam search over possible label sequences with a beam width of 4.

A batch size of 64 was used and the model was trained to minimize the sampled-$\mathrm{softmax}$ cross-entropy loss with 4096 negative samples. The Adam optimizer (Kingma & Ba, 2014) was used with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-4}$, and an initial learning rate of 10^{-3}; an exponential decay was applied by multiplying the initial learning rate by $.98^{n/30000}$ where n is the current training step. Gradients were clipped to have a maximum global norm of 2. Early stopping was used with respect to validation loss and took about 300,000 steps for the baseline model, and 180,000 steps for the monotonic model. Training was conducted on 16 machines with 4 GPUs each. We reported ROUGE scores computed over the test set of (Rush et al., 2015).

D.3. Machine Translation

Overall, we followed the model of (Luong & Manning, 2015) closely; our hyperparameters are largely the same: Words were mapped to 512-dimensional embeddings, which were learned during training. We passed sentences to the network in minibatches of size 128. As mentioned in the text, we used two unidirectional LSTM layers in both the encoder and decoder. All LSTM layers, and the attention energy function, had a hidden dimensionality of 512. We trained with a single replica for 40 epochs using Adam (Kingma & Ba, 2014) with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. We performed grid searches over initial learning rate and decay schedules separately for models using each of the two energy functions eq. (16) and eq. (17). For the model using eq. (16), we used an initial learning rate of 0.0005, and after 10 epochs we multiplied the learning rate by 0.8 each epoch; for eq. (17) we started at 0.001 and multiplied by 0.8 each epoch starting at the eighth epoch. Parameters were uniformly initialized in range $[-0.1, 0.1]$. Gradients were scaled whenever their norms exceeded 5. We used dropout with probability 0.3 as described in (Pham et al., 2014). Unlike (Luong & Manning, 2015), we did not reverse source sentences in our monotonic attention experiments. We set $r = -2$ for the attention energy function bias scalar for both eq. (16) and eq. (17). We used greedy decoding (i.e. no beam search) at test time.

E. Future Work

We believe there are a variety of promising extensions of our monotonic attention mechanism, which we outline briefly below.

- The primary drawback of training in expectation is that it retains the quadratic complexity during training. One idea would be to replace the cumulative product in eq. (9) with the thresholded remainder method of (Graves, 2016) and (Grefenstette et al., 2015), but in preliminary experiments we were unable to successfully learn alignments with this approach. Alternatively, we could further our investigation into gradient estimators for discrete decisions (such as REINFORCE or straight-through) instead of training in expectation (Bengio et al., 2013).
- As we point out in section 2.4, our method can fail when the attention energies $e_{i,j}$ are poorly scaled. This primarily stems from the strict enforcement of monotonicity. One possibility to mitigate this would be to instead regularize the model with a soft penalty which discourages non-monotonic alignments, instead of preventing them outright.
- In some problems, the input-output alignment is non-monotonic only in small regions. A simple modification to our approach which would allow this would be to subtract a constant integer from t_{i-1} between output timesteps. Alternatively, utilizing multiple monotonic attention mechanisms in parallel would allow the model to attend to disparate memory locations at each output timestep (effectively allowing for non-monotonic alignments) while still maintaining linear-time decoding.
- To facilitate comparison, we sought to modify the standard $\mathrm{softmax}$-based attention framework as little as possible. As a result, we have thus far not fully taken advantage of the fact that the decoding process is much more efficient. Specifically, the attention energy function of eq. (15) was primarily motivated by the fact that it is trivially parallelizable so that its repeated application is inexpensive. We could instead use a recurrent attention energy function, whose output depends on both the attention energies for prior memory items and those at the previous output timestep.

F. How much faster is linear-time decoding?

Throughout this paper, we have emphasized that one advantage of our approach is that it allows for linear-time decoding, i.e. the decoder RNN only makes a single pass over the memory in the course of producing the output sequence. However, we have thus far not attempted to quantify how much of a speedup this incurs in practice. Towards this end, we conducted an additional experiment to measure the speed of efficiently-implemented $\mathrm{softmax}$-based and hard monotonic attention mechanisms. We chose to focus solely on the speed of the attention mechanisms rather than an entire RNN sequence-to-sequence model because models using these attention mechanisms are otherwise equivalent.

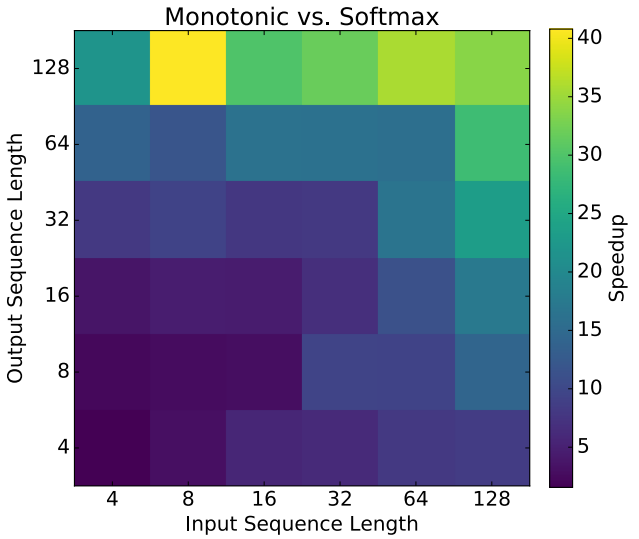

Figure 8. Speedup of hard monotonic attention mechanism compared to softmax attention on a synthetic benchmark.

Measuring the speed of the attention mechanisms alone allows us to isolate the difference in computational cost between the two approaches.

Specifically, we implemented both attention mechanisms using the highly-efficient C++ linear algebra package Eigen (Guennebaud et al., 2010). We set entries of the memory $\mathbf{h}$ and the decoder hidden states s_i to random vectors with entries sampled uniformly in the range $[-1, 1]$. We then computed context vectors following eqs. (2) and (3) for the softmax attention mechanism and following algorithm 1 for hard monotonic attention. We varied the input and output sequence lengths and averaged the time to produce all of the corresponding context vectors over 100 trials for each setting.

The speedup of the monotonic attention mechanism compared to softmax attention is visualized in fig. 8. We found monotonic attention to be about $4 - 40\times$ faster depending on the input and output sequence lengths. The most prominent difference occurred for short input sequences and long output sequences; in these cases the monotonic attention mechanism finishes processing the input sequence before it finishes producing the output sequence and therefore is able to stop computing context vectors. We emphasize that these numbers represent the best-case speedup from our approach; a more general insight is simply that our proposed hard monotonic attention mechanism has the potential to make decoding significantly more efficient for long sequences. Additionally, this advantage is distinct from the fact that our hard monotonic attention mechanism can be used for online sequence-to-sequence problems. We also emphasize that at training time, we expect our soft monotonic attention approach to have roughly the computational cost as standard softmax attention, thanks to the fact that we can compute the resulting attention distribution in parallel as described in appendix C.1. The code used for this benchmark is available in the repository for this paper.[5]

G. Practitioner's Guide

Because we are proposing a novel attention mechanism, we share here some insights gained from applying it in various settings in order to help practitioners try it on their own problems:

- The recursive structure of computing $\alpha_{i,j}$ in eq. (9) can result in exploding gradients. We found it vital to apply gradient clipping in all of our experiments, as described in appendix D.
- Many automatic differentiation packages can produce numerically unstable gradients when using their cumulative product function.[6][7] Our simple solution was to compute the product in log-space, i.e. replacing $\prod_n x_n = \exp(\sum_i \log(x_n))$.
- In addition, the product in the denominator of eq. (29) can become negligibly small because the terms $(1 - p_{i,k-1})$ all fall in the range $[0, 1]$. The simplest way to prevent the resulting numerical instabilities is to clip the range of the denominator to be within $[\epsilon, 1]$ where ϵ is a small constant (we used $\epsilon = 10^{-10}$). This can result in incorrect values for $\alpha_{i,j}$ particularly when some $p_{i,j}$ are close to 1, but we encountered no discernible effect on our results.
- Alternatively, we found in preliminary experiments that simply setting the denominator to 1 still produced good results. This can be explained by the observation that when all $p_{i,j} \in \{0, 1\}$ (which we encourage during training), eq. (29) is equivalent to the recurrence relation of eq. (10) even when the denominator is 1.
- As we mention in the experiment details of the previous section, we ended up using a small range of values for the initial energy function scalar bias r. In general, performance was not very sensitive to this parameter, but we found small performance gains from using values in $\{-5, -4, -3, -2, -1\}$ for different problems.
- More broadly, while the attention energy function modifications described in section 2.4 allowed models using our mechanism to be effectively trained on all tasks we

[5] https://github.com/craffel/mad

[6] https://github.com/tensorflow/tensorflow/issues/3862

[7] https://github.com/Theano/Theano/issues/5197

tried, they were not always necessary for convergence. Specifically, in speech recognition experiments the performance of our model was the same using eq. (15) and eq. (16), but for summarization experiments the models were unable to learn to utilize attention when using eq. (15). For ease of implementation, we recommend starting with the standard attention energy function of eq. (15) and then applying the modifications of eq. (16) if the model fails to utilize attention.

- It is occasionally recommended to reverse the input sequence prior to feeding it into sequence-to-sequence models (Sutskever et al., 2014). This violates our assumption that the input should be processed in a left-to-right manner when computing attention, so should be avoided.
- Finally, we highly recommend visualizing the attention alignments $\alpha_{i,j}$ over the course of training. Attention provides valuable insight into the model's behavior, and failure modes can be quickly spotted (e.g. if $\alpha_{i,j} = 0$ for all i and j).

With the above factors in mind, on all problems we studied, we were able to replace $\mathrm{softmax}$-based attention with our novel attention and immediately achieve competitive performance.

References

Abadi, Martin, Barham, Paul, Chen, Jianmin, Chen, Zhifeng, Davis, Andy, Dean, Jeffrey, Devin, Matthieu, Ghemawat, Sanjay, Irving, Geoffrey, Isard, Michael, Kudlur, Manjunath, Levenberg, Josh, Monga, Rajat, Moore, Sherry, Murray, Derek G., Steiner, Benoit, Tucker, Paul, Vasudevan, Vijay, Warden, Pete, Wicke, Martin, Yu, Yuan, and Zheng, Xiaoqiang. TensorFlow: A system for large-scale machine learning. In *Operating Systems Design and Implementation*, 2016.

Bengio, Yoshua, Léonard, Nicholas, and Courville, Aaron. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Chorowski, Jan and Jaitly, Navdeep. Towards better decoding and language model integration in sequence to sequence models. *arXiv preprint arXiv:1612.02695*, 2017.

Graves, Alex. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Grefenstette, Edward, Hermann, Karl Moritz, Suleyman, Mustafa, and Blunsom, Phil. Learning to transduce with unbounded memory. In *Advances in Neural Information Processing Systems*, 2015.

Guennebaud, Gaël, Jacob, Benoıt, Avery, Philip, Bachrach, Abraham, Barthelemy, Sebastien, et al. Eigen v3. `http://eigen.tuxfamily.org`, 2010.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Ladner, Richard E. and Fischer, Michael J. Parallel prefix computation. *Journal of the ACM (JACM)*, 27(4):831–838, 1980.

Liu, Peter J. and Pan, Xin. Text summarization with TensorFlow. `http://goo.gl/16RNEu`, 2016.

Luong, Minh-Thang and Manning, Christopher D. Stanford neural machine translation systems for spoken language domain. In *International Workshop on Spoken Language Translation*, 2015.

Pham, Vu, Bluche, Théodore, Kermorvant, Christopher, and Louradour, Jérôme. Dropout improves recurrent neural networks for handwriting recognition. In *International Conference on Frontiers in Handwriting Recognition*, 2014.

Rush, Alexander M., Chopra, Sumit, and Weston, Jason. A neural attention model for abstractive sentence summarization. In *Conference on Empirical Methods in Natural Language Processing*, 2015.

Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc V. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, 2014.