Abstract

In this project, we seek to explore the integration of external memory into end-to-end neural architectures for Automatic Speech Recognition (ASR). Though memory-based networks have recently proven to be quite successful across a variety of tasks in other domains, to the best of our knowledge there have been no previous attempts to apply such models to the challenge of spoken language processing. We thus propose a novel neural system that explicitly incorporates external memory modules into an encoder-decoder framework, where we aim to predict character-level outputs from audio-level features over the Wall Street Journal (WSJ0), CHIME, and TIDIGITS corpora. Though our ultimate memory models do not quite surpass the performance of our pure attention-based models, our results are quite promising and perhaps indicate that with proper fine-tuning these networks could obtain superlative performance on ASR tasks. We believe there is much potential for future development and look forward to performing continued research in this area.

1 Introduction

The task of speech recognition remains a remarkably rich and nuanced problem within the domain of spoken language processing. Despite the increasing prevalence of highly developed techniques for Automatic Speech Recognition (ASR) across academic and industrial settings, modern methods still face several inherent challenges, such as being robust to speaker-unique features, handling accents/dialects, working with distant microphones, or functioning in noisy environments. Many of these models are also limited in their ability to learn from sparse amounts of data, with advanced models sometimes requiring several millions of utterances, which are often not readily available in many contexts, in order to effectively train.

Given the existing limitations of such models, searching for more effective and efficient formulations for speech recognition is key. In particular, a sophisticated model should be robust to noise, handle large variations in input, while also being able to learn meaningful information from limited quantities of inputs. Thus, in light of this and inspired by the recent success of memory-based models across several difficult natural language and vision tasks, we propose the integration of memory modules into neural frameworks for speech recognition. In particular, we devise an encoder-decoder model where at each step of encoding or decoding, the network has the opportunity to read from or write to an external memory bank. It is our belief that these memory networks could potentially enhance performance across ASR tasks and enable the training of a more data-efficient and generalizable model.

2 Previous Work

To the best of our knowledge, there are no published examples of memory-based models applied to automatic speech recognition – however, there still exists a significant amount of related work focusing individually on memory-based networks and on ASR as a whole. In this section, we will examine traditional non-neural/hybrid approaches for ASR, before shifting to end-to-end neural ASR systems, and finally concluding with a general discussion of memory-based neural models and their application to other domains.

2.1 HMM-based ASR Models

(Povey et al., 2011) demonstrate that hybrid models which combine Gaussian Mixture models with the traditional Hidden Markov models, given enough components, can represent the probabilities that input features are associated with each HMM state incredibly accurately, are relatively easy to fit, less computationally demanding, easily parallelizable, and have consistently proven to be very successful for all types of ASR tasks. However, despite their success, these GMM-HMM hybrids are intractable for modeling non-linearities because they require an exponential number of parameters but actual speech input often has a much lower dimensionality. This shortcoming, combined with the recent advances in compute power, has resulted in the development of Deep Neural Network - HMM hybrid models (DNN-HMMs). (Hinton et al., 2012) show that using a DNN-HMM generated output to perform the traditional Viterbi alignment out-performs highly tuned GMM-HMM systems on a variety of datasets of various vocabulary sizes. However, both of these GMM and DNN hybrid models suffer from the shortcomings inherent to any HMM system - they require prior do-
main knowledge, necessitate explicit, and not necessarily good, assumptions about the state space, and require pre-segmented inputs and post-processed outputs.

2.2 End-to-End Neural ASR Models

In contrast to some of the limitations described in the previous section, end-to-end neural models for ASR seek to use a single differentiable pipeline that replaces the HMM system with a neural framework that implicitly represents both the acoustic and language models. Thus these systems effectively leverage the discriminative power of neural networks in what is ultimately a discriminative task without requiring the design of complicated, manually tuned modules.

(Graves et al., 2006) proposed one of the earliest approaches towards constructing such end-to-end neural speech recognizers by defining Connectionist Temporal Classification (CTC) loss. At the time, recurrent neural networks could not be used to generate predictions/sequences of different length than the input sequence if the input itself was not pre-segmented, since there are exponentially many alignments that could correspond to a target labelling. Thus, they propose an application of the forward-backward algorithm to compute the probability of individual labelings by “marginalizing” out possible alignments, and then training the network to minimize cross-entropy loss over the labelings. Their results demonstrate an improvement over baseline HMM and hybrid DNN-HMM systems, and thus showed the practical potential of end-to-end neural systems.

That said, this approach had to take special effort to address the disparity in input/output lengths and non-aligned training data. A more intuitive approach is perhaps to utilize neural sequence-to-sequence models, which are naturally suited for handling input and output sequences of different lengths and which were first applied to great effect in machine translation (Sutskever et al., 2014). In this formulation, an encoder converts the input sequence into a fixed size representation while a decoder generates predictions based on this encoding.

One of the most successful attempts to apply this model towards ASR was in (Chan et al., 2015), which utilized a pyramidal encoder and a decoder with attention to produce character level outputs. This model bypassed the conditional independence assumptions found in the CTC model while also implicitly generating alignments with the attention mechanism. However, its performance, though remarkable, fell short of the then state-of-the-art model, and also required three million alignments/sequences of different length than the input sequence for the decoder component of one of our methods. All of these papers thus seem to indicate that incorporating some form of memory generally improves performance while also allowing the model to be more robust and data efficient. However, we must again note that there have been no approaches to apply these memory-based networks to ASR or related problems. In this project, we thus present the first attempt, to the best of our knowledge, to apply memory-enhanced models to the task of speech recognition.

3 Technical Approach

In this section, we now describe the various methodologies that we have applied to our task. Section 4 describes our experiments and evaluations, and Section 5 provides a more qualitative analysis of our results.

3.1 Dataset and Preprocessing

We train and evaluate our model on three individual datasets - TIDIGITS, CHiME2 Grid, and CSR-I (WSJ0). TIDIGITS is the simplest of the three datasets, from which we utilize a subset of 50192
continuous utterances of one to seven digit sequences randomly chosen from a set of eleven characters - \{1,2,3,4,5,6,7,8,9,z,o\} where numbers 1-9 map to their corresponding spoken digit and z and o map to two way to say the digit 0, "zero" and "oh", respectively. We divide the data into approximately 35000 pairs for training, 7500 for validation, and 7500 for testing.

The second dataset we examine is the "clean" subset of the CHiME2 Grid challenge data. This subset consists of 17000 simple continuous 6-word utterances spoken in a living room environment. For illustration, some example utterances are “bin blue at two now" or “set white by c zero now”. We split this dataset into approximately 12000 pairs for training (about 6 hours worth of audio), 2500 for validation, and 2500 for testing. All inputs strictly consist of alphabetical characters.

Finally, for our most challenging dataset, we utilize a subset of the Continuous Speech Recognition Wall Street Journal corpus, CSR-I (WSJ0) Sennheiser speech, which contains 35487 audio recordings of read speech, spontaneous dictation, and read spontaneous dictation, along with their corresponding transcriptions. Some sample transcriptions extracted from the dataset are shown below:

```
Specifically the union said it was proposing to purchase all of the
assets of United Airlines including planes gates facilities and landing
rights
```

```
[lip smack] In over the counter trading Friday the company's com-
mon closed at ten dollars and twenty five cents a share down fifty
cents
```

We split our data into approximately 25,000 pairs for training (about 49 hours worth of audio), 4000 for validation, and 4000 for testing. We then apply basic pre-processing to our data by removing all non alphanumeric characters and converting transcriptions to lower-case. Next, given their usefulness in speech processing-related tasks, across all datasets we extract Mel-Frequency Cepstral Coefficient (MFCC) features from the audio files at time intervals of 10ms, computing 39 features per timestep. We then normalize the features by subtracting off the training feature mean. Finally, all inputs and output are padded to a single respective length, determined by an examination of the data distribution (see WSJ0 example below) to allow efficient batch processing during training.

3.2 Models

As we can see, for the WSJ0 input sequences, nearly all of the examples have a length of at most 500, and for the target transcriptions, nearly all of the sequences have output lengths less than 200. Thus, we can keep all examples that fall in this range and remove the outliers, which increases computational tractability while not significantly affecting performance. For our inputs across all datasets, we append zero vectors until we reach the max input length, and for the transcriptions, we append <PAD> tokens after the end token of each sentence until we reach the maximum output length. We can then apply appropriate masking to make sure these padded values do not affect loss computation or optimization.

Baseline Encoder-Decoder Models

The simplest of models that we could implement that nevertheless retains the advantages of an end-to-end neural model is a vanilla encoder-decoder network. In particular, for this model, we utilize a single-layer Gated Recurrent Unit (GRU) (Cho et al., 2014) for both the encoder and the decoder.

The encoder accepts the raw MFCC features extracted from the audio file, and encodes it into a single fixed-size embedding (represented by the final hidden state of the network). The decoder initializes its first hidden state to this encoded representation, and at each timestep during training, accepts a ground-truth
token as input and attempts to predict the next token of the sequence. We utilize cross-entropy loss at each timestep, and use randomly initialized and trainable embeddings to represent the output characters. During testing/evaluation, the only difference is that the previously generated character is now fed in as input to the next timestep, where at each timestep we greedily output the most probable character.

Note that this model is extremely simple (e.g. one layer, no attention, no beam search, etc.), so it is certainly not expected to achieve superlative results – however, at its core, it possesses the same structure of our more advanced models, and thus serves as a useful lower-bound on future performance.

For a second baseline, an additional enhancement we can make to our vanilla model is to use a bi-directional network for the encoder – the intuition behind this choice is that, as shown in many machine translation papers, such as (Bahdanau et al., 2014), reversing the input sequence before running the encoder is helpful since the models have the tendency to focus on information in the final timesteps of the encoder, and at the start of the decoding process, we are presumably trying to generate outputs that correspond to the beginning of the input sequence. A bidirectional encoder builds on this intuition by encapsulating information extracted from both sides of the input before presenting it to the decoder – we thus expect that adding this network will result in a more meaningful and informative encoded representation. We thus concatenate the forward and backward final hidden states of the encoder to produce our final encoded representation.

**Attention-Based Encoder-Decoder**

One of the central limitations of the previous two approaches is that the model is forced to encode all the information it needs to know about the input sequence into a fixed-size representation that is then used during decoding. This clearly may not be realistic when input sequences are long, or the decodings are based on nuanced functions of the inputs. Thus, to address these shortcomings, for our next model we have incorporated a global attention mechanism, an elegant technique that has recently gained popularity in natural language processing tasks due to its ability to handle long input sequences and its recent success in machine translation.

The idea, as described in (Luong et al., 2015), is that at every timestep $t$ of the decoder, we extract the decoder’s hidden state $h_t$ and compute an attention map across the entire set of hidden states of the encoder; we then use this mapping to obtain a context vector that is better able to inform our predictions – thus, the decoder can potentially shift its “focus” to any point in the input sequence. More formally, for a given decoder timestep $t$ and corresponding hidden vector $h_t$, we compute a variable-sized alignment vector $a_t$ by comparing $h_t$ with each source (encoder) hidden vector $h_s$. Thus, the $s$-th element of $a_t$ is given by

$$a_t(s) = \text{align}(h_t, \tilde{h}_s) = \frac{\exp(\text{score}(h_t, \tilde{h}_s))}{\sum_s \exp(\text{score}(h_t, h_s'))}$$

where $\tilde{h}_s$ is the encoder’s hidden vector at source timestep $s$. The alignment weights $a_t$ are then used to compute the context vector $c_t$, which is the weighted average of all the source hidden states in the encoder. The context $c_t$ is in turn used to derive another vector $\tilde{h}_t$, where

$$\tilde{h}_t = \tanh(W_c[c_t; h_t])$$

which is finally input into a linear layer with cross-entropy loss to predict the next symbol during decoding. As an aside, we note that there are many possible options we could choose for the scoring function used in computing $a_t$ – in our case, however, we use a simple dot product so that $\text{score}(h_t, \tilde{h}_s) = h_t^T \tilde{h}_s$, which is also what was originally suggested in (Luong et al., 2015). Figure 2 shows the general structure of our global-attention network.

### 3.2.1 CNN-RNN Encoder

While we expect the attention-based model to be well-suited to the task of speech recognition (and indeed we see that is the case in (Chan et al., 2015)), we note that a particular challenge associated with our task is that the input sequences for speech recognition can frequently be quite long – for instance, our pre-processed Wall Street Journal corpus contains inputs of up to 500 timesteps in length. This in turn causes the memory used in attention to increase in size and thus effectively dilutes the focus of the attention mechanism, since the network now must make a decision over a larger number of positions regarding where to attend. While we could
conceivably perform a downsampling strategy (e.g. examining only every other feature) to produce a smaller amount of encoded memory, we expect that this drops valuable information about the input in the process.

To address these concerns, inspired by the pyramidal architecture of (Chan et al., 2015), we propose a hybrid CNN-RNN encoder, where we will incorporate 1-dimensional convolutional layers that can provide an effective reduction in sequence length while also learning to appropriately compose information across the time series. In particular, we will integrate these convolutional layers together with our standard bi-directional recurrent networks in order to also encode temporal relationships between the features. To formalize this more concretely, our model definition is as follows: we begin by using a bidirectional GRU to transform the input sequence into another sequence of equivalent length (which is obtained by extracting the hidden state of the network at every timestep). We then convolve 100 filters (which is obtained by extracting the hidden state of the CNN by using a bidirectional GRU to transform the input) of the CNN layer with width network at every timestep). We then convolve 100 filters (which is obtained by extracting the hidden state of the CNN by using a bidirectional GRU to transform the input).

3.2.2 Memory-Augmented Decoder

In this section, we now introduce the formulation of our first external-memory based model. We propose a novel decoder structure that is a synthesis of Neural Turing Machines and the standard RNN decoders used in sequence-to-sequence tasks. Inspired by the work of (Wang et al., 2016), which utilized memory-based decoders in machine translation, we construct our decoder so that at every timestep $t$ our decoder will have the opportunity to read or write from an external memory bank $M^B_t$ in a differentiable manner. Intuitively, we expect this addition to be a valuable enhancement to the network architecture since it allows the model to effectively propagate more information during the decoding process and thus capture long-range dependencies. We can in fact think of the memory bank as a "notepad" that the model can use to keep track of information it may require at subsequent points during decoding.

To express this more formally, let $x_t$ be the input to the decoder at timestep $t$, let $M^B_t$ be an $n \times d$ matrix that is the current state of the memory bank at time $t$ (where $n$ is the number of cells and $d$ is the depth of each cell), and let $s_{t-1}$ be the output produced by the previous timestep of the network. We first use a content-based addressing mechanism to perform a READ operation of the memory bank $M^B_t$ using the previously generated output as a key:

$$r_t = \text{READ}(s_{t-1}, M^B_t)$$

The resultant vector is then concatenated with the current input $x_t$ and fed into a (potentially multi-layer) GRU cell to produce an output $h_t$:

$$h_t = \text{GRU}(\begin{bmatrix} x_t \\ r_t \end{bmatrix})$$

As with our standard attention based-models, this vector $h_t$ is used to perform Luong-style attention over the source hidden states to produce context vector $c_t$ that represents a weighted sum of the encoder hidden states. Then, to produce our final output vector for the current timestep, we compute

$$s_t = \tanh(W_c \begin{bmatrix} c_t \\ h_t \\ r_t \end{bmatrix})$$

Finally, the computed output vector is used to write to the memory bank using content-based addressing:

$$M^B_{t+1} = \text{WRITE}(s_t, M^B_t)$$

The general structure of the memory-based decoder cell is shown in Figure 4. In the following sections we describe in greater detail how the reading and writing mechanisms are implemented.

**Reading**

To perform reading over the memory bank $M^B_t$ with a key $s_{t-1}$, we take an alternative approach to the method described in (Wang et al., 2016) and (Graves et al., 2014) by using a simple dot-product to serve as the scoring function between memory cells and the key vector, and then perform softmax over the scores to produce an alignment vector. This reduces the required number of

![Figure 3: CNN-RNN Encoder Model](image)
parameters needed to compute similarity while nevertheless maintaining performance, as was demonstrated for general attention mechanisms in (Luong et al., 2015) – though it necessarily constrains the dimension of each cell to be identical to that of the decoder’s output. That is, first we compute an alignment vector \( \tilde{a}_t^i \) whose \( i \)-th element is

\[
\tilde{a}_t^i = \frac{\exp(s^T_{t-1} M^B_t[i])}{\sum_j \exp(s^T_{t-1} M^B_t[j])}
\]

where \( M^B_t[i] \) is the \( i \)-th memory cell of the memory bank. In contrast to standard attention, however, we also incorporate the influence of the previous alignment vector by using a gating mechanism, as proposed in (Wang et al., 2016), so that the final alignment vector is

\[
a_t^i = a_t^i \tilde{a}_{t-1}^i + (1 - a_t^i) \tilde{a}_t^i
\]

where \( a_t^i \) is a scalar given by \( \sigma(w_g^r s_{t-1}) \), with \( w_g^r \) being a learnable parameter. Intuitively, we want to bound how quickly the read head can change focus between consecutive timesteps, and the gating ensures that we incorporate enough information of what was read during the previous timestep into our current calculations. This final alignment vector is then used to compute a weighted sum over the memory cells, which produces the final output of the READ mechanism.

**Writing**

Writing follows a similar procedure to that of reading – in particular, we first compute an alignment vector \( \tilde{a}_t^i \) given key \( s_t \) with the same process defined in reading, but utilizing a distinct set of parameters. Once this alignment vector is computed, an ERASE operation followed by an ADD operation is performed.

For each \( i \)-th cell of the memory bank, the ERASE mechanism computes a modified value of

\[
\tilde{M}_t^B[i] = M_t^B[i] \odot (1 - \alpha_t w^e_t \cdot e_t^{ERS})
\]

where \( e_t^{ERS} \) is a vector given by \( \sigma(W^{ERS} s_t) \), and where \( W^{ERS} \) is a learnable parameter. The ADD operation finally computes the new value of the \( i \)-th cell as

\[
M_t^B[i] = \tilde{M}_t^B[i] + a_t^i \tilde{a}_t^i \cdot e_t^{ADD}
\]

where \( e_t^{ADD} \) is a vector given by \( \sigma(W^{ADD} s_t) \), and where \( W^{ADD} \) is a learnable parameter.

### 3.2.3 Memory-Augmented Encoder

One of the challenges posed by the previous approach is deriving an appropriate initialization for the decoder’s memory bank. In particular, there must be enough asymmetry among the cells of the bank to allow diverse learning to take place, and, ideally, it would also be beneficial to ensure that the memory bank is not completely independent of the encoded sequence when decoding begins.

(Wang et al., 2016) cope with this by initialing every cell to a value computed from the encoder hidden states, and then adding random noise to each cell. This approach however yields non-determinism across identical inputs, and in addition it might be difficult for short output sequences to effectively overcome the starting amount of noise present in the memory bank.

To derive a solution to these limitations, we propose a novel augmentation of the encoder with an external memory module which the network will write to during encoding. Once the encoding process has concluded, the resultant memory bank is used to initialize the memory bank of the decoder. This ensures that the initial values for the decoder’s memory bank are reasonable and also allows the encoder to “decide” what initial message to pass on to the decoder, yielding much more flexibility.

More concretely, the encoder memory model is a simplified version of the decoder, where only writes are performed and no separate attention mechanism is involved. At time \( t \) with input \( x_t \), the encoder first computes \( h_t \) as \( \text{GRU}(x_t) \). \( h_t \) is then used as the key to write to the encoder’s memory bank: \( M_t^E = \text{WRITE}(h_t, M_t^E) \). Finally, if there are \( T \) timesteps in the input sequence, then \( M_t^E \) is used as the initialization of the decoder memory. The initial state of the encoder memory itself is treated as a parameter to be learned during training. One subtlety to observe is if we are using a bidirectional GRU – in this case, the forward and reverse RNN each utilize a separate memory bank, and the final memory bank state is produced by concatenating the memory banks, doubling the dimension of each cell.

Note that we do not consider reading during encoding since the memory bank is intuitively designated for decoder use and we only want the encoder to “scribble” information on it, but encoder cells with READs might prove a worthwhile extension to investigate.
We likewise see similar behavior for the bidirectional model. What we can conclude from this and several other examples is that the model often correctly predicts the first character of the test sequence, but then just defaults to generating a sentence seen in the training set that is consistent with that first character instead of learning a meaningful mapping from input audio to features to output transcriptions. All of the other models, in contrast, seem to perform significantly better, obtaining near-perfect performance on the CHiME and TIDIGITS corpora, which follows our intuition given that they represent easier ASR tasks.

In contrast, on the much more challenging WSJ dataset, we see that there is a larger performance spread—in particular, our deep attention model obtains a superlative 25.8% character error rate on the test set (in fact, comparing our results to those obtained by Graves et al. across comparable models in (Graves and Jaitly, 2014), we obtain a lower error rate than with half as much data). Likewise, comparing to, whereas remarkably our CNN-RNN encoder model obtains only 43%. We believe that this drop in performance for the CNN model cannot necessarily be attributed to the 1D-CNN layers, which have typically been used to great effect to model time-series. Instead, further reorganization of the network structure and tweaking of hyperparameters could prove useful in improving overall performance.

We also note that the mistakes made by both of these aforementioned models make sense; for example, some errors respectively for both models are

- Predicted: *people will soon leave what go undraying from sacrami- nator joseporate or cisco period*
  - Real: *people will soon be able to go on the train from sacramento to san francisco period*

- Predicted: *the professor of chinese politics have university washing- ton that of head under few Real: a professor of chinese politics at the university of washington though had another view*

As we can see, the model is making mistakes that sound close to the actual ground truth, which shows that it is learning some form of intuitive mapping. Now, we discuss the performance of our memory models—as we can see, the decoder-only memory model does not fare so well on the WSJ0 dataset. This actually follows our expectation, since due to the ambiguities in initialization the model can easily become skewed during training and not generalize well to unseen data. Indeed, we see by examining a few error cases, the model does suffer from periodically repeating itself, as it seems to do by inserting “comma” which indicates the memory may be improperly initialized:

- but this time indefed that the election state allowing comma in
- he last comma he indicates he clear to incorporated period

In contrast, the model with memory applied to both the encoder and decoder significantly improves performance on the WSJ dataset, outperformed only by the
Table 1: Character error rates

<table>
<thead>
<tr>
<th></th>
<th>TIDIGITS</th>
<th>CHiME</th>
<th>WSJ0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Train</td>
<td>Val</td>
<td>Test</td>
</tr>
<tr>
<td>Baseline</td>
<td>0.0110</td>
<td>0.0478</td>
<td>0.0475</td>
</tr>
<tr>
<td></td>
<td>0.0588</td>
<td>0.1908</td>
<td>0.1928</td>
</tr>
<tr>
<td></td>
<td>0.0944</td>
<td>0.6898</td>
<td>0.6919</td>
</tr>
<tr>
<td>Bidirectional Encoder</td>
<td>0.0302</td>
<td>0.0992</td>
<td>0.1006</td>
</tr>
<tr>
<td></td>
<td>0.0588</td>
<td>0.1908</td>
<td>0.1928</td>
</tr>
<tr>
<td></td>
<td>0.1928</td>
<td>0.6919</td>
<td>0.7555</td>
</tr>
<tr>
<td>Bidir. Encoder w/ Attention</td>
<td>0.0019</td>
<td>0.0046</td>
<td>0.0056</td>
</tr>
<tr>
<td></td>
<td>0.0019</td>
<td>0.0466</td>
<td>0.0555</td>
</tr>
<tr>
<td></td>
<td>0.0019</td>
<td>0.0466</td>
<td>0.0555</td>
</tr>
<tr>
<td>CNN-RNN</td>
<td>0.0012</td>
<td>0.0576</td>
<td>0.0578</td>
</tr>
<tr>
<td></td>
<td>0.0180</td>
<td>0.0089</td>
<td>0.0089</td>
</tr>
<tr>
<td></td>
<td>0.0180</td>
<td>0.0089</td>
<td>0.0089</td>
</tr>
<tr>
<td>MemDec</td>
<td>0.0029</td>
<td>0.0030</td>
<td>0.0019</td>
</tr>
<tr>
<td></td>
<td>0.0081</td>
<td>0.0078</td>
<td>0.0078</td>
</tr>
<tr>
<td></td>
<td>0.0081</td>
<td>0.0078</td>
<td>0.0078</td>
</tr>
<tr>
<td>MemEncDec</td>
<td>0.0110</td>
<td>0.0116</td>
<td>0.0116</td>
</tr>
<tr>
<td></td>
<td>0.0011</td>
<td>0.0111</td>
<td>0.0111</td>
</tr>
<tr>
<td></td>
<td>0.0011</td>
<td>0.0111</td>
<td>0.0111</td>
</tr>
</tbody>
</table>

Table 2: Word error rates

<table>
<thead>
<tr>
<th></th>
<th>TIDIGITS</th>
<th>CHiME</th>
<th>WSJ0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Train</td>
<td>Val</td>
<td>Test</td>
</tr>
<tr>
<td>Baseline</td>
<td>0.0388</td>
<td>0.1580</td>
<td>0.1590</td>
</tr>
<tr>
<td></td>
<td>0.1223</td>
<td>0.4742</td>
<td>0.4731</td>
</tr>
<tr>
<td></td>
<td>0.8920</td>
<td>0.8816</td>
<td>0.9442</td>
</tr>
<tr>
<td>Bidirectional Encoder</td>
<td>0.0885</td>
<td>0.2714</td>
<td>0.2775</td>
</tr>
<tr>
<td></td>
<td>0.1319</td>
<td>0.3582</td>
<td>0.3623</td>
</tr>
<tr>
<td></td>
<td>0.8920</td>
<td>0.8816</td>
<td>0.9442</td>
</tr>
<tr>
<td>Bidir. Encoder w/ Attention</td>
<td>0.0092</td>
<td>0.0095</td>
<td>0.0137</td>
</tr>
<tr>
<td></td>
<td>0.0092</td>
<td>0.0095</td>
<td>0.0134</td>
</tr>
<tr>
<td></td>
<td>0.0092</td>
<td>0.0095</td>
<td>0.0134</td>
</tr>
<tr>
<td>CNN-RNN</td>
<td>0.0071</td>
<td>0.0072</td>
<td>0.0072</td>
</tr>
<tr>
<td></td>
<td>0.0230</td>
<td>0.0230</td>
<td>0.0230</td>
</tr>
<tr>
<td></td>
<td>0.0230</td>
<td>0.0230</td>
<td>0.0230</td>
</tr>
<tr>
<td>MemDec</td>
<td>0.0148</td>
<td>0.0135</td>
<td>0.0118</td>
</tr>
<tr>
<td></td>
<td>0.0207</td>
<td>0.0191</td>
<td>0.0191</td>
</tr>
<tr>
<td></td>
<td>0.0207</td>
<td>0.0191</td>
<td>0.0191</td>
</tr>
<tr>
<td>MemEncDec</td>
<td>0.0110</td>
<td>0.0102</td>
<td>0.0102</td>
</tr>
<tr>
<td></td>
<td>0.0277</td>
<td>0.0267</td>
<td>0.0267</td>
</tr>
<tr>
<td></td>
<td>0.0277</td>
<td>0.0267</td>
<td>0.0267</td>
</tr>
</tbody>
</table>

Although the model does not exceed the deep attention model’s performance, we once again believe that our model is intuitive. In particular, what we notice is that the model is able to correctly match every character on several inputs over the WSJ test set, but it occasionally makes more extreme mistakes which lower its average score, which might happen if the encoder does not correctly write sufficient information to the notepad – perhaps it thus requires a few minor changes to the forward pass in order to improve performance.

6 Conclusion

In this paper, we have presented a novel approach towards performing automatic speech recognition with end-to-end neural models. In particular, we have proposed the first model for ASR to incorporate external memory reserves and have compared its performance to a variety of other sequence-to-sequence model across three different datasets. Though our pure memory-based decoder alone does not produce reasonable results, incorporating memory into both the encoder and decoder significantly improves the quality of outputs. Our ultimate findings are promising with our memory-based encoder-decoder obtaining close to the best results among our models on the WSJ dataset, and also shows signs that the model is learning in an intuitive way.

7 Future Work

There are several directions that we could pursue for the continuation of this project. Most immediately, we would like to tune the structure and hyper-parameters of our memory based model to achieve even more sophisticated results, as it has often been noted that memory-based models are extremely sensitive to training conditions, initializations, and hyper-parameter settings.

Next, we note that since we were interested purely in deriving a powerful neural model for ASR, we have not integrated any form of additional knowledge or outside information – in particular, a technique which is commonly utilized in ASR systems to improve quality of responses is to add a language model. In particular, we see that our current model, since it is generating characters, does not consider whether the eventual tokens that are produced actually form realistic words. Once we achieve superlative performance with our set of models, we could likely produce even better results by thus incorporating information from the language model.

Finally, it would be an interesting experiment to examine the effects of using reinforcement learning based approaches to yield more fine-grained look-ups over the memory bank, akin to how local attention might be used in sequence-to-sequence models.

In any case, we believe that there is much potential for further interesting developments in this field, and we look forward to performing continued research in this domain.
8 Acknowledgements

The authors would like to thank Professor Andrew Maas, Jiwei Li, and the 224S staff, for their continued support and advice on this project.

References


