Bridging HMMs and RNNs through Architectural Transformations

Jan Buys[1] **Yonatan Bisk**[1] **Yejin Choi**[1,2]
Paul G. Allen School of Computer Science & Engineering, University of Washington[1]
Allen Institute for Artificial Intelligence[2]
`{jbuys, ybisk, yejin}@cs.washington.edu`

Abstract

A distinct commonality between HMMs and RNNs is that they both learn hidden representations for sequential data. In addition, it has been noted that the backward computation of the Baum-Welch algorithm for HMMs is a special case of the back-propagation algorithm used for neural networks [6]. Do these observations suggest that, despite their many apparent differences, HMMs are a special case of RNNs? In this paper, we show that that is indeed the case, and investigate a series of architectural transformations between HMMs and RNNs, both through theoretical derivations and empirical hybridization. In particular, we investigate three key design factors—independence assumptions between the hidden states and the observation, the placement of softmaxes, and the use of non-linearities—in order to pin down their empirical effects. We present a comprehensive empirical study to provide insights into the interplay between expressivity and interpretability in this model family with respect to language modeling and parts-of-speech induction.

1 Introduction

The sequence is a common structure among many forms of naturally occurring data, including speech, text, video, and DNA. As such, sequence modeling has long been a core research problem across several fields of machine learning and AI. By far the most widely used approach for decades is Hidden Markov Models [2, 11], which assumes a sequence of *discrete* latent variables to generate a sequence of observed variables. When the latent variables are unobserved, unsupervised training of HMMs can be performed via the Baum-Welch algorithm (which, in turn, is based on the forward-backward algorithm), as a special case of Expectation-Maximization (EM) [5]. Importantly, the discrete nature of the latent variables has the benefit of interpretability, as they recover contextual clustering of the output variables. In contrast, Recurrent Neural Networks (RNNs) [12, 7] introduced later assume *continuous* latent representations. Their hidden states have no probabilistic interpretation, regardless of many different architectural variants, such as LSTMs [10], GRUs [4] and RANs [14].

Despite their many apparent differences, both HMMs and RNNs model hidden representations for sequential data. At the heart of both models are: a state at time t, a transition function $f : h_{t-1} \rightarrow h_t$ in latent space, and an emission function $g : h_t \rightarrow x_t$. In addition, it has been noted that the backward computation in the Baum-Welch algorithm is a special case of back-propagation for neural networks [6]. Therefore, a natural question arises as to the fundamental relationship between HMMs and RNNs. Might HMMs be a special case of RNNs?

In this paper, we investigate a series of architectural transformations between HMMs and RNNs—both through theoretical derivations and empirical hybridization. In particular, we demonstrate that forward marginal inference for an HMM—accumulating forward probabilities to compute the marginal emission and hidden state distributions at each time step—can be reformulated as equations for computing an RNN cell. In addition, we investigate three key design factors—independence

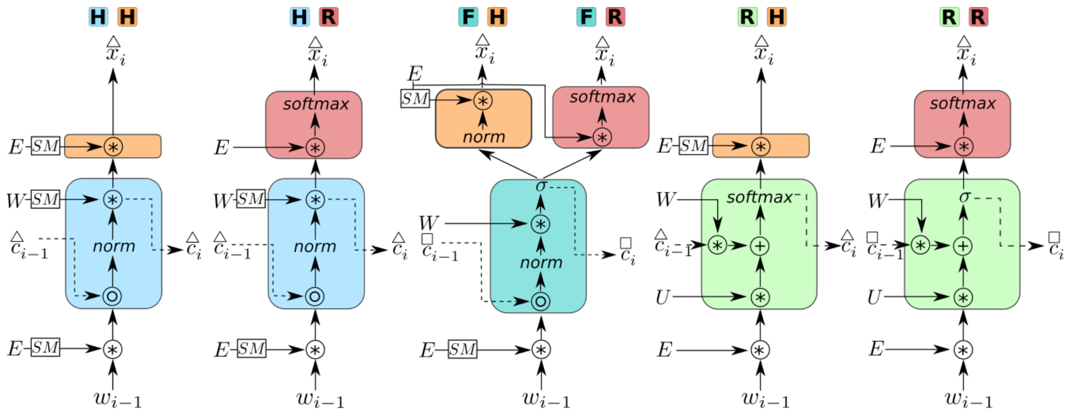

Figure 1: Above each of the models we indicate the type of transition and emission cells used. **H** for HMM, **R** for RNN/Elman and **F** is a novel Fusion defined in §3.3. It is particularly important to track when a vector is a distribution (resides in a simplex) versus in the unit cube (e.g. after a sigmoid non-linearity). These are indicated by $\overset{\triangle}{c}_i$ and $\overset{\square}{c}_i$, respectively. SM stands for *softmax rows*.

assumptions between the hidden states and observations, the placement of softmaxes, and the use of non-linearities—in order to pin down their empirical effects. While we focus on HMMs with discrete outputs, our analysis framework could be extended to HMMs over continuous observations.

Our work builds on earlier work that have also noted the connection between RNNs and HMMs [24, 26] (see §7). Our contribution is to provide the first thorough theoretical investigation into the model variants, carefully controlling for every design choices, along with comprehensive empirical analysis over the spectrum of possible hybridization between HMMs and RNNs.

We find that the key elements to better performance of the HMMs are the use of a sigmoid instead of softmax linearity in the recurrent cell, and the use of an unnormalized output distribution matrix in the emission computation. On the other hand, multiplicative integration of the previous hidden state and input embedding, and intermediate normalizations in the cell computation are less consequential. We also find that HMMs outperform other RNNs variants for unsupervised prediction of the next POS tag, demonstrating the advantages of discrete bottlenecks for increased interpretability.

The paper is structured as follows. First, we present the derivation of HMM marginal inference as a special case of RNN computation (§2). Next we explore a gradual transformation of HMMs into RNNs (§3), followed by the reverse transformation of Elman RNNs back to HMMs (§4). Finally we provide empirical analysis in §5 and §6 to pin point the effects of varying design choices over possible hybridizations between HMMs and RNNs.

2 Formulating HMMs as Recurrent Neural Networks

We start by defining HMMs as discrete sequence models, together with the forward-backward algorithm which is used for inference. Then we show that, by rewriting the forward algorithm, the computation can be viewed as updating a hidden state at each time step by feeding the previous word prediction, and then computing the next word distribution, similar to the way RNNs are structured. The resulting architecture corresponds to the first cell in Figure 1.

2.1 Model definition

Let $\mathbf{x}^{(1:n)} = \{\mathrm{x}^{(1)}, \ldots, \mathrm{x}^{(n)}\}$ be a sequence of random variables, where each x is drawn from a vocabulary $\mathbb{V}$ of size v, and an instance x is represented as an integer w or a one-hot vector $\boldsymbol{e}^{(w)}$, where w corresponds to an index in $\mathbb{V}$. [1] We also define a corresponding sequence of hidden variables $\mathbf{h}^{(1:n)} = \{\mathrm{h}^{(1)}, \ldots, \mathrm{h}^{(n)}\}$, where $\mathrm{h} \in \{1, 2, \ldots m\}$. The distribution $P(\mathbf{x})$ is defined by

[1]Our notation follows [9]. Sequences are notated as $\mathbf{x}$ or $\boldsymbol{w}$.

marginalizing over $\mathbf{h}$, and factorizes as follows:

$$P(\mathbf{x}) = \sum_{\mathbf{h}} P(\mathbf{x}, \mathbf{h}) = \sum_{\mathbf{h}} P(\mathrm{h}^{(1)}) p(\mathrm{x}^{(1)}|\mathrm{h}^{(1)}) \prod_{i=2}^{n} P(\mathrm{h}^{(i)}|\mathrm{h}^{(i-1)}) P(\mathrm{x}^{(i)}|\mathrm{h}^{(i)}) \tag{1}$$

We define the hidden state distribution, referred to as the *transition* distribution, and the the *emission* (output) distribution as

$$P(\mathrm{h}^{(i)}|\mathrm{h}^{(i-1)} = l) = \mathrm{softmax}(\boldsymbol{W}_{l,:} + \boldsymbol{b}), \boldsymbol{W} \in \mathbb{R}^{m \times m}, \boldsymbol{b} \in \mathbb{R}^{m} \tag{2}$$

$$P(\mathrm{h}^{(1)}) = \mathrm{softmax}(\sum_{l} \boldsymbol{W}_{l,:}^{\top} + \boldsymbol{b}), \tag{3}$$

$$p(\mathrm{x}^{(i)}|\mathrm{h}^{(i)} = k) = \mathrm{softmax}(\boldsymbol{E}_{k,:} + \boldsymbol{d}), \boldsymbol{E} \in \mathbb{R}^{m \times v}, \boldsymbol{d} \in \mathbb{R}^{v}. \tag{4}$$

2.2 Inference

Inference for HMMs (marginalizing over the hidden states to compute the observed sequence probabilities) is performed with the forward-backward algorithm. The backward algorithm is equivalent to automatically differentiating the forward algorithm [6]. Therefore, while traditional HMM implementations had to implement both the forward and backward algorithm, and train the model with the EM algorithm, we only implement the forward algorithm in standard deep learning software, and perform end-to-end minibatched SGD training, efficiently parallelized on a GPU.

Let $\boldsymbol{w} = \{w^{(1)}, \ldots, w^{(n)}\}$ be the observed sequence, and $\boldsymbol{w}^{(i)}$ the one-hot representation of $w^{(i)}$. The forward probabilities $\boldsymbol{a}$ are defined recurrently (i.e., sequentially recursively) as

$$a_k^{(i)} = P(\mathrm{h}^{(i)} = k, \mathbf{x}^{(1:i)} = \boldsymbol{w}^{(1:i)}), \tag{5}$$

$$= P(\mathrm{x}^{(i)} = w^{(i)}|\mathrm{h}^{(i)} = k) \sum_{l=1}^{m} a_l^{(i-1)} P(\mathrm{h}^{(i)} = k|\mathrm{h}^{(i-1)} = l). \tag{6}$$

This can be rewritten by defining

$$\boldsymbol{c}^{(i)} = P(\mathrm{h}^{(i)}|\mathbf{x}^{(1:i-1)} = \boldsymbol{w}^{(1:i-1)}), \tag{7}$$

$$\boldsymbol{s}^{(i)} = P(\mathrm{h}^{(i)}|\mathbf{x}^{(1:i)} = \boldsymbol{w}^{(1:i)}), \tag{8}$$

$$x^{(i)} = P(\mathrm{x}^{(i)} = w^{(i)}|\mathbf{x}^{(1:i-1)} = \boldsymbol{w}^{(1:i)}), \tag{9}$$

and substituting $\boldsymbol{a}$, so that equation 6 is rewritten as left below, or expressed directly in terms of the parameters used to define the distributions with vectorized computations (right below):

$$c_k^{(i)} = \sum_{l=1}^{m} s_l^{(i-1)} P(\mathrm{h}^{(i)} = k|\mathrm{h}^{(i-1)} = l), \qquad \boldsymbol{c}^{(i)} = \mathrm{softmax}_{\text{rows}}(\boldsymbol{W})^{\top} \boldsymbol{s}^{(i-1)}, \tag{10}$$

$$\boldsymbol{e}^{(i)} = \mathrm{softmax}_{\text{rows}}(\boldsymbol{E}) \boldsymbol{w}^{(i)}, \tag{11}$$

$$x^{(i)} = \sum_{k=1}^{m} P(\mathrm{x}^{(i)} = w^{(i)}|\mathrm{h}^{(i)} = k) c_k^{(i)}, \qquad x^{(i)} = \boldsymbol{e}^{(i)^{\top}} \boldsymbol{c}^{(i)}, \tag{12}$$

$$s_k^{(i)} = \frac{1}{x^{(i)}} P(\mathrm{x}^{(i)} = w^{(i)}|\mathrm{h}^{(i)} = k) c_k^{(i)}. \qquad \boldsymbol{s}^{(i)} = \frac{1}{x^{(i)}} \boldsymbol{e}^{(i)} \circ \boldsymbol{c}^{(i)}. \tag{13}$$

Here $\boldsymbol{w}^{(i)}$ is used as a one-hot vector, and the bias vectors $\boldsymbol{b}$ and $\boldsymbol{d}$ are omitted for clarity. Note that the computation of $\boldsymbol{s}^{(i)}$ can be delayed until time step $i + 1$. The computation step can therefore be rewritten to let $\boldsymbol{c}$ be the recurrent vector (equivalent logspace formulations presented on the right): [2]

$$\boxed{\begin{aligned}
\boldsymbol{e}^{(i-1)} &= \text{softmax}_{\text{rows}}(\boldsymbol{E})\boldsymbol{w}^{(i-1)}, && = \text{logsoftmax}_{\text{rows}}(\boldsymbol{E})\boldsymbol{w}^{(i-1)}, && (14)\\
\boldsymbol{s}^{(i-1)} &= \text{normalize}(\boldsymbol{e}^{(i-1)} \circ \boldsymbol{c}^{(i-1)}), && = \text{softmax}(\boldsymbol{e}^{(i-1)} + \boldsymbol{c}^{(i-1)}), && (15)\\
\boldsymbol{c}^{(i)} &= \text{softmax}_{\text{rows}}(\boldsymbol{W})^\top \boldsymbol{s}^{(i-1)}, && = \log(\text{softmax}_{\text{rows}}(\boldsymbol{W})^\top \boldsymbol{s}^{(i-1)}), && (16)\\
\boldsymbol{e}^{(i)} &= \text{softmax}_{\text{rows}}(\boldsymbol{E})\boldsymbol{w}^{(i)}, && = \text{logsoftmax}_{\text{rows}}(\boldsymbol{E})\boldsymbol{w}^{(i)}, && (17)\\
x^{(i)} &= \boldsymbol{e}^{(i)\top} \boldsymbol{c}^{(i)}, && = \text{logsumexp}(\boldsymbol{e}^{(i)} + \boldsymbol{c}^{(i)}). && (18)
\end{aligned}}$$

This can be viewed as a step of a recurrent neural network with tied input and output embeddings: Equation 14 embeds the previous prediction, equations 15 and 16, the *transition* step, updates the hidden state $\boldsymbol{c}$, corresponding to the cell of a RNN, and equations 17 and 18, the *emission* step, computes the output next word probability.

We can now compare this formulation against the definition of an Elman RNN with tied embeddings and a sigmoid non-linearity. These equations correspond to the first and last cells in Figure 1. The Elman RNN has the same parameters, except for an additional input matrix $\boldsymbol{U} \in \mathbb{R}^{m \times m}$.

$$\boxed{\begin{aligned}
\boldsymbol{e}^{(i-1)} &= \boldsymbol{E}\boldsymbol{w}^{(i-1)}, && (19)\\
\boldsymbol{c}^{(i)} &= \sigma(\boldsymbol{W}\boldsymbol{c}^{(i-1)} + \boldsymbol{U}\boldsymbol{e}^{(i-1)}), && (20)\\
x^{(i)} &= \text{softmax}(\boldsymbol{E}\boldsymbol{c}^{(i)})\boldsymbol{w}^{(i)}. && (21)
\end{aligned}}$$

3 Transforming an HMM towards an RNN

Having established the relation between HMMs and RNNs, we propose a number of models that we hypothesize have intermediate expressiveness between HMMs and RNNs. The architecture transformations can be seen in the first 3 cells in Figure 1. We will evaluate these model variants empirically (§5), and investigate their interpretability (§6).

3.1 Conditioning transition distribution on previous word

By relaxing the independence assumption of the HMM transition probability distribution we can increase the expressiveness of the HMM "cell" by modelling more complex interactions between the fed word and the hidden state. These model variants are non-homogeneous HMMs.

Tensor-based feeding:
Following [22] we define the transition distribution as

$$P(\text{h}^{(i)} | \text{h}^{(i-1)} = l, \text{x}^{(i-1)} = w) = \text{softmax}(\mathbf{W}_{l,:}\boldsymbol{e}^{(i-1)} + \boldsymbol{B}_{l,:}), \tag{22}$$

where $\mathbf{W} \in \mathbb{R}^{m \times m \times m}, \boldsymbol{B} \in \mathbb{R}^{m \times m}$.

Addition-based feeding:
As tensor-based feeding increases the number of parameters considerably, we also propose an additive version:

$$P(\text{h}^{(i)} | \text{h}^{(i-1)} = l, \text{x}^{(i-1)} = w) = \text{softmax}(\boldsymbol{W}_{l,:} + \boldsymbol{U}\boldsymbol{e}^{(i-1)} + \boldsymbol{b}), \tag{23}$$

where $\boldsymbol{W} \in \mathbb{R}^{m \times m}, \boldsymbol{U} \in \mathbb{R}^{m \times m}, \boldsymbol{b} \in \mathbb{R}^{m}$.

Gating-based feeding:
Finally we propose a more expressive model where interaction is controlled via a gating mechanism and the feeding step uses unnormalized embeddings (this does not violate the HMM factorization):

$$\boldsymbol{e}^{'(i-1)} = \boldsymbol{E}\boldsymbol{w}^{(i-1)}, \tag{24}$$

$$\boldsymbol{f}^{i} = \sigma(\boldsymbol{U}\boldsymbol{e}^{'(i-1)} + \boldsymbol{b}), \tag{25}$$

$$P(\text{h}^{(i)} | \text{h}^{(i-1)} = l, \text{x}^{(i-1)} = w) = \text{softmax}(\boldsymbol{W}_{l,:} \circ \boldsymbol{f}^{(i)}), \tag{26}$$

where $\boldsymbol{U} \in \mathbb{R}^{m \times m}, \boldsymbol{b} \in \mathbb{R}^{m}, \boldsymbol{W} \in \mathbb{R}^{m \times m}$.

[2] where $\text{normalize}(\boldsymbol{y}) = \frac{\boldsymbol{y}}{\sum_i y_i}$.

3.2 Delayed softmaxes

Another way to make HMMs more expressive is to relax their independence assumptions through delaying when vectors are normalized to probability distributions by applying the softmax function.

Delayed transition softmax
The computation of the recurrent vector $\boldsymbol{c}^{(i)} = P(\mathrm{h}^{(i)}|\mathbf{x}^{(1:i-1)})$ is replaced with

$$\boldsymbol{c}^{(i)} = \mathrm{softmax}(\boldsymbol{W}\boldsymbol{s}^{(i-1)}). \tag{27}$$

Both $\boldsymbol{c}$ and $\boldsymbol{s}$ are still valid probability distributions, but the independence assumption in the distribution over $\mathrm{h}^{(i)}$ no longer holds.

Delayed emission softmax
A further transformation is to delay the emission softmax until after multiplication with the hidden vector. This effectively replaces the HMM's emission computation with that of the RNN:

$$x^{(i)} = \mathrm{softmax}(\boldsymbol{E}\boldsymbol{c}^{(i)})\boldsymbol{w}^{(i)}. \tag{28}$$

This formulation breaks the independence assumption that the output distribution is only conditioned on the hidden state assignment. Instead it can be viewed as taking the expectation over the (unnormalized) embeddings with respect to the state distribution $\boldsymbol{c}$, then softmaxed (**H R** in Fig 1).

3.3 Sigmoid non-linearity

We can go further towards RNNs and replace the softmax in the transition by a sigmoid non-linearity. The sigmoid is placed in the same position as the delayed softmax. The recurrent state $\boldsymbol{c}$ is no longer a distribution so the output has to be renormalized so the emission still computes a distribution:

$$\boldsymbol{c}^{(i)} = \mathrm{sigmoid}(\boldsymbol{W}\boldsymbol{s}^{(i-1)}), \tag{29}$$

$$x^{(i)} = \boldsymbol{e}^{(i)\top}\mathrm{normalize}(\boldsymbol{c}^{(i)}). \tag{30}$$

This model could also be combined with a delayed emission softmax - which we'll see makes it closer to an Elman RNN. This model is indicated as **F** (fusion) in Figure 1.

4 Transforming an RNN towards an HMM

Analogously to making the HMM more similar to Elman RNNs, we can make Elman networks more similar to HMMs. Examples of these transformations can be seen in the last 2 cells in Figure 1.

4.1 HMM emission

First, we use the Elman cell with an HMM emission function. This requires the hidden state be a distribution. We consider two options: One is to replace the sigmoid non-linearity with the softmax function (**R H** in Figure 1):

$$\boldsymbol{c}^{(i)} = \mathrm{softmax}(\boldsymbol{W}\boldsymbol{c}^{(i-1)} + \boldsymbol{U}\boldsymbol{e}^{(i-1)}) \tag{31}$$

$$x^{(i)} = (\mathrm{softmax}(\boldsymbol{E})\boldsymbol{w}^{(i)})^\top \boldsymbol{c}^{(i)}. \tag{32}$$

The second formulation is to keep the sigmoid non-linearity, but normalize the hidden state output inside the emission computation:

$$\boldsymbol{c}^{(i)} = \sigma(\boldsymbol{W}\boldsymbol{c}^{(i-1)} + \boldsymbol{U}\boldsymbol{e}^{(i-1)}) \tag{33}$$

$$x^{(i)} = (\mathrm{softmax}(\boldsymbol{E})\boldsymbol{w}^{(i)})^\top \mathrm{normalize}(\boldsymbol{c}^{(i)}). \tag{34}$$

4.2 Softmax non-linearity

Second, we experiment with replacing the sigmoid non-linearity with a softmax:

$$\boldsymbol{c}^{(i)} = \mathrm{softmax}(\boldsymbol{W}\boldsymbol{c}^{(i-1)} + \boldsymbol{U}\boldsymbol{e}^{(i-1)}) \tag{35}$$

As a more flexible variant, the softmax is applied only to compute the emission distribution, while the sigmoid non-linearity is still applied to recurrent state:

$$\boldsymbol{c}^{(i)} = (\boldsymbol{W}\sigma(\boldsymbol{c}^{(i-1)}) + \boldsymbol{U}\boldsymbol{e}^{(i-1)}) \tag{36}$$

$$x^{(i)} = \mathrm{softmax}(\boldsymbol{E}\mathrm{softmax}(\boldsymbol{c}^{(i)}))\boldsymbol{w}^{(i)}. \tag{37}$$

4.3 Multiplicative integration

In the HMM cell, the integration of the previous recurrent state and the input embedding is modelled through an element-wise product instead of adding affine transformations of the two vectors. We can modify the Elman cell to do a similar multiplicative integration:[3]

$$\boldsymbol{c}^{(i)} = \sigma((\boldsymbol{W}\boldsymbol{c}^{(i-1)}) \circ (\boldsymbol{U}\boldsymbol{e}^{(i-1)}))) \tag{38}$$

Or, using a single transformation matrix:

$$\boldsymbol{c}^{(i)} = \sigma(\boldsymbol{W}(\boldsymbol{c}^{(i-1)} \circ \boldsymbol{e}^{(i-1)})) \tag{39}$$

5 Language Modeling Experiments

Our formulations investigate a series of small architectural changes to HMMs and Elman cells. In particular, these changes raise questions about the expressivity and importance of (1) normalization within the recurrence and (2) independence assumptions during emission. In this section, we analyze the effects of these changes quantitatively via a standard language modeling benchmark.

5.1 Setup

We follow the standard PTB language modeling setup [3, 17]. We work with one-layer models to enable a direct comparison between RNNs and HMMs and a budget of 10 million parameters (typically corresponding to hidden state sizes of around 900). Models are trained with batched backpropagation through time (35 steps). Input and output embeddings are tied in all models. Models are optimized with a grid search over optimizer parameters for two strategies: SGD[4] and AMSProp. AMSProp is based on the optimization setup proposed in [15].[5]

5.2 Results

We see from the results in Table 1 (also depicted in Figure 2) that the HMM models perform significantly worse than the Elman network, as expected. Interestingly, many of the HMM variants that in principle have more expressivity or weaker independence assumptions do not perform better than the vanilla HMM. This includes delaying the transition or emission softmax, and most of the feeding models. The exception is the gated feeding model, which does substantially better, showing that gating is an effective way of incorporating more context into the transition matrix. Using

[3]This is related to the architecture proposed in [26], which shows that models with multiplicative integration can obtain competitive performance.

[4]Choose a learning rate from $\{5, 10, 20\}$ and decayed by 4.0 after each epoch where validation did not improve. Batch size was 20, and embedding parameters are initialized from the uniform distribution in the range $(-0.1, 0.1)$. These mostly follow the settings of `https://github.com/pytorch/examples/tree/master/word_language_model`.

[5] We use AMSGrad [20] (instead of Adam [13]) with $\beta_1 = 0$, making it more similar to RMSProp. We pick an initial learning rate from $\{0.001, 0.002\}$, and a l_2 weight decay rate from $\{0, 1e-4, 1e-5, 1e-6, 1e-7\}$. Batch size is 32, embedding parameters are initialized from the uniform distribution in the range $(-0, 8, 0.8)$.

Model	dev ppl	Model	dev ppl
HMM		**Elman**	
– Vanilla	284.59	– Softmax, HMM emission	313.84
– Tensor feeding	288.15	– HMM emission (normalize)	312.63
– Addition feeding	288.62	– Softmax non-linearity	207.95
– Gated feeding	243.51	– Softmax normalize before emit	225.36
– Delayed transition softmax	284.59	– Multiplicative (single matrix)	107.45
– Delayed emission softmax	287.00	– Multiplicative	100.71
– Delayed transition and emission softmax	293.72	– Vanilla	87.27
– Sigmoid non-linearity	240.91		
– Sigmoid non-linearity, delayed emission softmax	142.31	**LSTM**	80.61

Table 1: Language Modeling Perplexity for our baseline and transformed models.

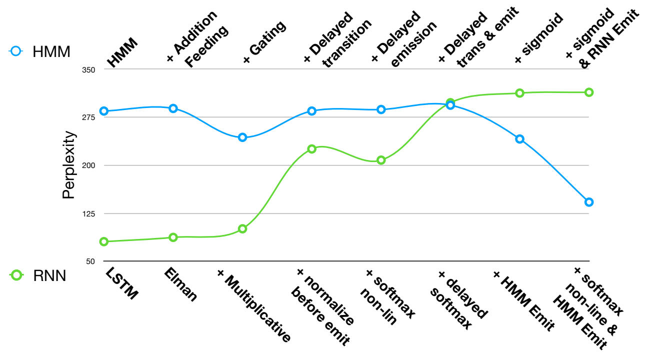

Figure 2: Perplexities under our architectural transformations of HMM and RNN variants.

a sigmoid non-linearity before the output of the HMM cell (instead of a softmax) does improve performance (by 44 ppl), and combining that with delaying the emission softmax gives a substantial improvement (almost another 100 ppl), making it much closer to some of the RNN variants.

We also evaluate variants of Elman RNNs: Just replacing the sigmoid non-linearity with the softmax function leads to a substantial drop in performance (120 ppl), although it still performs better than the HMM variants where the recurrent state is a distribution. Another way to investigate the effect of the softmax is to normalize the hidden state output just before applying the emission function, while keeping the sigmoid non-linearity: This performs somewhat worse than the softmax non-linearity, which indicates that it is significant whether the input to the emission function is normalized or softmaxed before multiplying with the (emission) embedding matrix. As a comparison for how much the softmax non-linearity acts as a bottleneck, a neural bigram model outperforms these approaches, obtaining 177 validation perplexity on this same setup.

Replacing the RNN emission function with that of an HMM leads to even worse performance than the HMM; a softmax non-linearity or a sigmoid followed by normalization does not make a significant difference. Multiplicative integration leads to only a small drop in performance compared to a vanilla Elman RNN, and doing so with a single transformation matrix (more comparable to the HMM) leads to only a small further drop. In contrast, preliminary experiments showed that the second transformation matrix is crucial in the performance of the vanilla Elman network.

To put our results in context, we also compare against an LSTM baseline with the same number of parameters, using the same regularization and hyperparameter search strategy as for our other models. While more extensive hyperparameter tuning [15] or more sophisticated optimization and regularization techniques [16] would improve performance, the goal here is just to do a fair comparison within the computational resources we had available to optimize all models, not to compete with state-of-the-art performance.

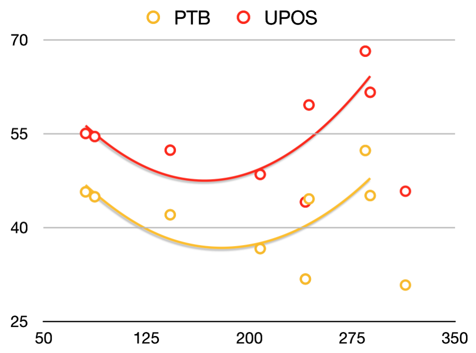

Figure 3: Tagging accuracies (right) are plotted against perplexities from Table 1. We see a somewhat quadratic relationship.

Model	PTB	UPOS
HMM		
– Vanilla	52.36	68.23
– Tensor feeding	45.16	61.66
– Gated feeding	44.62	59.64
– Sigmoid non-linearity	31.82	44.13
– Sigmoid non-linearity with delayed emission softmax	42.09	52.41
Elman		
– Softmax, HMM emission	30.86	45.85
– Softmax non-linearity	36.68	48.54
– Vanilla	44.97	54.59
LSTM	45.75	55.08

Table 2: Tagging accuracies for representative models. Accuracy is calculated by converting $p(w)$ to $p(t)$ according to WSJ tag distributions.

6 Syntactic Evaluation

A strength of HMM bottlenecks is forcing the model to produce an interpretable hidden representation. A classic example of this property in language modeling is part-of-speech tag induction. It is therefore natural to ask whether changes in the architecture of our models correlate with their ability to discover syntactic properties. We evaluate this by analyzing the models' implicitly predicted tag distributions at each time step. Specifically we hypothesize that the HMMs will preserve basic tag-tag patterns of the language, and that this may not be true for RNNs. We test this by computing the accuracy of predicting the tag of the next word in the sequence out of the next word distribution. None of the models were trained to perform this task.

We estimate the model's distribution over POS tags at each time step, $p(t)$, by marginalizing over the model's output word distribution $p(w)$ and the context-independent tag distribution $p(t|w)$ for every word in the training portion of the PTB. We compare the most likely marginal tag against the ground truth to compute a tagging accuracy. This evaluation rewards models which place their emission probability mass predominantly on words of the correct part-of-speech. We compute this metric across both the full PTB tagset and universal tags (UPOS) [19].

Viterbi decoding in HMMs enable us to compute the tag distribution conditioned on the highest scoring (Viterbi) state at each time step. This leads to better performance than marginalizing over hidden state values, showing that the states encode meaningful word clusterings. In contrast, Elman models perform best when conditioned on the full hidden state rather than the maximum dimension only. Results are shown in Table 2 and plotted against perplexity in Figure 3.

7 Related Work

A number of recent papers have identified variants of gated RNNs which are simpler than LSTMs but perform competitively or satisfy properties that LSTMs lack. These variants include RNNs without recurrent non-linearities to improve interpretability [8], gated RNN variants with type constraints [1], and a class of RNNs called rational recurrences, in which the hidden states can be computed by WFSAs [18]. Our goal was instead to compare RNNs against HMMs, which while clearly less expressive can provide complementary insights into the strengths of RNNs.

Another strand of recent work proposed neural models that learn discrete, interpretable structure: [27] introduced a mixture of softmaxes model where the output distribution is conditioned on discrete latent variable. Other work includes language modeling that jointly learns unsupervised syntactic (tree) structure [21] and neural hidden Markov models for Part-of-Speech induction [22]. Models of segmental structure over sequences [25, 23] and neural transduction models with discrete latent alignments [28] have also been proposed.

8 Conclusion

In this work, we presented a theoretical and empirical investigation into model variants over the spectrum of possible hybridization between HMMs and RNNs. By carefully controlling all design choices, we provide new insights into several factors including independence assumptions, the placement of softmax, and the use of nonlinearities. Comprehensive empirical results demonstrate that the key elements to better performance of the RNN are the use of a sigmoid instead of softmax linearity in the recurrent cell, and the use of an unnormalized output distribution matrix in the emission computation. Multiplicative integration of the previous hidden state and input embedding, and intermediate normalizations in the cell computation are less consequential. HMMs outperforms other RNNs variants in a next POS tag prediction task, which demonstrates the advantages of models with discrete bottlenecks in increased interpretability.

References

[1] David Balduzzi and Muhammad Ghifary. Strongly-typed recurrent neural networks. *CoRR*, abs/1602.02218, 2016.

[2] Leonard E. Baum and J. A. Eagon. An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology. *Bulletin of the American Mathematical Society*, 73(3):360 – 363, 1967.

[3] Ciprian Chelba and Frederick Jelinek. Exploiting syntactic structure for language modeling. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 1*, pages 225–231, Montreal, Quebec, Canada, August 1998. Association for Computational Linguistics.

[4] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.

[5] A Dempster, N Laird, and D Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[6] Jason Eisner. Inside-outside and forward-backward algorithms are just backprop (tutorial paper). In *Proceedings of the Workshop on Structured Prediction for NLP*, pages 1–17, Austin, TX, November 2016. Association for Computational Linguistics.

[7] J Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 06 1990.

[8] Jakob N Foerster, Justin Gilmer, Jascha Sohl-Dickstein, Jan Chorowski, and David Sussillo. Input switched affine networks: An rnn architecture designed for interpretability. In *International Conference on Machine Learning*, pages 1136–1145, 2017.

[9] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016.

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

[11] Frederick Jelinek, Lalit R. Bahl, and Robert L Mercer. Design of a linguistic statistical decoder for the recognition of continuous speech. *IEEE Transactions on Information Theory*, 21(3):250–256, May 1975.

[12] Michael I Jordan. Serial order: A parallel distributed processsing approach. Technical report, University of California, San Diego, 1986.

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

[14] Kenton Lee, Omer Levy, and Luke S Zettlemoyer. Recurrent additive networks. *arXiv preprint arXiv:1705.07393*, 05 2017.

[15] Gábor Melis, Chris Dyer, and Phil Blunsom. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*, 2017.

[16] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing LSTM language models. *CoRR*, abs/1708.02182, 2017.

[17] Tomáš Mikolov, Anoop Deoras, Stefan Kombrink, Lukáš Burget, and Jan Černockỳ. Empirical evaluation and combination of advanced language modeling techniques. In *Twelfth Annual Conference of the International Speech Communication Association*, 2011.

[18] Hao Peng, Roy Schwartz, Sam Thomson, and Noah A. Smith. Rational recurrences. In *Proc. of EMNLP*, 2018.

[19] Slav Petrov, Dipanjan Das, and Ryan McDonald. A universal part-of-speech tagset. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012)*, pages 2089–2096, Istanbul, Turkey, 05 2012.

[20] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. In *ICLR*, 2018.

[21] Yikang Shen, Zhouhan Lin, Chin-Wei Huang, and Aaron C. Courville. Neural language modeling by jointly learning syntax and lexicon. *CoRR*, abs/1711.02013, 2017.

[22] Ke M. Tran, Yonatan Bisk, Ashish Vaswani, Daniel Marcu, and Kevin Knight. Unsupervised neural hidden markov models. In *Proceedings of the Workshop on Structured Prediction for NLP*, pages 63–71, Austin, TX, November 2016. Association for Computational Linguistics.

[23] Chong Wang, Yining Wang, Po-Sen Huang, Abdelrahman Mohamed, Dengyong Zhou, and Li Deng. Sequence modeling via segmentations. *arXiv preprint arXiv:1702.07463*, 2017.

[24] T Wessels and Christian W Omlin. Refining hidden markov models with recurrent neural networks. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks*, volume 2, pages 271–276. IEEE, 2000.

[25] Sam Wiseman, Stuart M Schieber, and Alexander M Rush. Learning neural templates for text generation. *arXiv preprint arXiv:1808.10122*, 08 2018.

[26] Yuhuai Wu, Saizheng Zhang, Ying Zhang, Yoshua Bengio, and Ruslan R Salakhutdinov. On multiplicative integration with recurrent neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2856–2864. Curran Associates, Inc., 2016.

[27] Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W Cohen. Breaking the softmax bottleneck: A high-rank rnn language model. *arXiv preprint arXiv:1711.03953*, 2017.

[28] Lei Yu, Jan Buys, and Phil Blunsom. Online segment to segment neural transduction. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1307–1316, Austin, Texas, November 2016. Association for Computational Linguistics.