Hierarchical Multiscale Gated Recurrent Unit

Alex Pine
alex.pine@nyu.edu

Sean D Rosario
seandrosario@nyu.edu

Abstract

The hierarchical multiscale long-short term memory recurrent neural network (HM-LSTM) was recently introduced as a model that can learn the semantics of sequence data at multiple time scales simultaneously without the need for explicit boundary information. We introduce an adaptation of this model, the hierarchical multiscale gated recurrent unit (HM-GRU), that incorporates the gated recurrent unit into its internal structure. We evaluate its performance as a language model on the Penn treebank dataset, and a novel dataset composed of lyrics from popular music. We find that HM-GRU performs similarly well to HM-LSTM, suggesting that the hierarchical multiscale architecture is not tied to the LSTM cell design.

1 Introduction

Hierarchical multiscale long-short term memory recurrent neural networks (HM-LSTMs) have been recently introduced by Chung et al. (2016) as an improvement on traditional LSTMs (Hochreiter & Schmidhuber, 1997). Like regular LSTMs, HM-LSTMs are recurrent neural networks designed to process variable-length sequences of data, most commonly written text. Both models may also be stacked in several layers, with the outputs of one layer being fed to the layer above in addition to the next time step of the same layer (El Hihi & Bengio 1995).

HM-LSTMs differ from regular LSTMs in three important ways: 1) higher levels of the model send their output to lower levels and 2) the connections between layers and timesteps are controlled by hard (i.e. discrete) binary gate neurons, and 3) the formulas used to compute the output of the model are conditionally controlled by the values of the input binary gate neurons.

In this paper, we introduce a variation on the HM-LSTM: the hierarchical multiscale gated recurrent united (HM-GRU). This model retains the binary gates and layer connections of the HM-LSTM, but replaces the soft gates and conditional update rules with ones inspired by the gated recurrent unit (GRU) (Cho et al., 2014). The GRU has similar performance characteristics as the LSTM, but requires fewer parameters (Chung et al., 2014). Mirroring the tests of HM-LSTM in Chung et al. (2016), we evaluate the ability of HM-GRU as a language model on the Penn tree bank dataset (Marcus et al., 1993), and compare it to the HM-LSTM, GRU, and LSTM.

The different layers of the HM-LSTM model have been shown to be able to learn the semantics of their input at different time scales, with the higher levels learning longer-term semantics (Chung et al. 2016). We have created a new dataset consisting of the lyrics of over 13,000 songs to further test this assertion. Song lyrics have a simple, regular, hierarchical structure that should be easily learned by the different layers of the model, and the examples are short enough that they can easily be fit into memory. We evaluate the performance of HM-GRU and HM-LSTM on this new dataset and compare their performance to their counterparts, LSTM and GRU, respectively.

2 Related Work

HM-GRU is a direct variant of HM-LSTM, as stated earlier. HM-LSTM can be seen as a synthesis between two different RNN models, the original LSTM, and the original hierarchical RNN proposed by El Hihi & Bengio (1995), enhanced with the conditional computation enabled by binary stochastic neurons (Bengio et al., 2013). The LSTM is the defacto standard for RNN cells, and hierarchical RNNs have been shown to reduce classification errors on longer input sequences (El Hihi & Bengio, 1995).

The novel aspect about the HM-LSTM is its application of stochastic binary neurons to dynamically control which layers should be connected to others, and how cell outputs are computed, based on the inputs to the model. Previous models updated levels either at a pre-set schedule, like the clockwork RNN (Koutnk et al., 2014), or using a-priori knowledge of the structure of the data, as in Ling et al. (2015). HM-LSTM is the first model to learn a hierarchical structure from its training data, and adapt it dynamically at inference time. This technique was made viable by using the straight through estimator first suggested by Hinton (2012) and developed by Bengio et al. (2013) for computing the gradient of the stochastic binary neurons, coupled with the new slope annealing trick (Chung et al. 2016).

Song lyrics have only recently been used in conjunction with neural networks in academic context. Malmi et al., (2015) created a model called DopeLearning which is trained to find a line from any song corpus of song lyrics that best matches an input line of text according to rhyme and rhythm. It can then be run iteratively to generate a new song that forms a lyrically sensible composition of lines from different songs. Potash et al. (2015) created a model called Ghostwriter that trains an LSTM on hip-hop lyrics, and examines the quality of its output, finding that the model was often able to generate lyrics with plausible rhyme schemes.

3 The Model

The HM-GRU model is a straightforward adaptation of the HM-LSTM. Like a typical RNN cell, HM-GRU cells are intended to be stacked on top of each other in several layers, with the input data at the lowest layer. Like an HM-LSTM cell, an HM-GRU cell takes as its input a discrete binary gate from the layer below and the previous time step on the same layer, in addition to the hidden vectors from the same sources. Also like HM-LSTM cell, it takes as one of its inputs the hidden state of the cell at one layer above at the previous time step. A single HM-LSTM cell at layer $l \in L$ layers at time step t has the following functional form:

$$h_t^l, z_t^l = f_{\text{HM-GRU}}^l(h_{t-1}^l, h_t^{l-1}, h_{t-1}^{l+1}, z_t^l, z_t^{l-1})$$

Here, h_t^l represents the hidden vector of the HM-GRU, while z_t^l values represents the binary scalar boundary detector states which determine how h_t^l is computed:

$$h_t^l = \begin{cases} (1 - u_t^l) \odot h_{t-1}^l + u_t^l \odot g_t^l \\ \text{if } z_{t-1}^l = 0 \text{ and } z_t^{l-1} = 1 \text{ (UPDATE)} \\ \\ h_{t-1}^l \\ \text{if } z_{t-1}^l = 0 \text{ and } z_t^{l-1} = 0 \text{ (COPY)} \\ \\ u_t^l \odot g_t^l \\ \text{if } z_{t-1}^l = 1 \text{ (FLUSH)} \end{cases}$$

These three equations are adapted from the corresponding UPDATE, COPY, and FLUSH equations in HM-LSTM (Chung et al., 2016). In HM-LSTM, the UPDATE operation corresponds to the normal LSTM update rule. The FLUSH operation is similar, but leaves out the term with the previous time step's c_t^l vector. The COPY operation outputs the vectors from the previous time step completely unchanged.

HM-GRU implements the ideas of these three operations equivalently with the update rules from the GRU. Like GRU, HM-GRU only has one hidden vector h_t^l, compared to HM-LSTM's two vectors, c_t^l and h_t^l. Its UPDATE operation corresponds exactly to that of the normal GRU cell, while its FLUSH operation is the UPDATE operation with the previous time step's hidden vector omitted. The COPY operation just copies the previous hidden vector, exactly like HM-LSTM does.

The r_t^l and u_t^l terms in these equations refer to the reset and update gates, which correspond directly to the gates in the canonical GRU update formula (Cho et al., 2014). These gates, along with the candidate next hidden vector g_t^l, and the parameter to the boundary detector state, $\tilde{z}_t^l$, are computed in a similar fashion to how HM-LSTM computes its gates and candidate output vectors. Each one is computed as a composition of the output from the previous time steps at the same layer and the one above, and the output of the layer below:

$$r_t^l = sigm\big(U_{l,r}^l h_{t-l}^l + z_{t-l}^l U_{l+1,r}^l h_{t-l}^{l+1} + z_t^{l-1} W_{l-1,r}^l h_t^{l-1} + b_r^l\big)$$

$$u_t^l = sigm\big(U_{l,u}^l h_{t-l}^l + z_{t-l}^l U_{l+1,u}^l h_{t-l}^{l+1} + z_t^{l-1} W_{l-1,u}^l h_t^{l-1} + b_u^l\big)$$

$$W_{i,r}^j, W_{i,u}^j \in \mathbb{R}^{dim(h^l)\times dim(h^{l-1})}$$
$$U_{i,r}^j, U_{i,u}^j \in \mathbb{R}^{dim(h^l)\times dim(h^l)}$$

Just as in GRU, we compute the candidate output hidden state using $tanh$ instead of $sigm$, and $r \odot h$ instead of h:

$$g_t^l = tanh\big(U_{l,g}^l (r_t^l \odot h_{t-l}^l) + z_{t-l}^l U_{l+1,g}^l (r_t^l \odot h_{t-l}^{l+1}) + z_t^{l-1} W_{l-1,g}^l (r_t^l \odot h_t^{l-1}) + b_g^l\big)$$

$$W_{i,g}^j \in \mathbb{R}^{dim(h^l)\times dim(h^{l-1})}$$
$$U_{i,g}^j \in \mathbb{R}^{dim(h^l)\times dim(h^l)}$$

Note that in the original GRU formula, the update gate is denoted with z_t^l, while I denote it with a u_t^l here. This is done to prevent confusion between it and the boundary detector state z_t^l defined earlier.

3.1 Computing the Binary Boundary Detector

The binary boundary detector state z_t^l can be effectively computed in several possible ways from the deterministically computed $\tilde{z}_t^l$ (Bengio et al., 2013), just as it can be for HM-LSTM. We compute the $\tilde{z}_t^l$ the same way Chung et al. (2016) do, except that use the regular sigmoid function instead of the hard sigmoid function:

$$\tilde{z}_t^l = sigm\big(U_{l,\tilde{z}_t^l}^l h_{t-l}^l + z_{t-l}^l U_{l+1,\tilde{z}_t^l}^l h_{t-l}^{l+1} + z_t^{l-1} W_{l-1,\tilde{z}_t^l}^l h_t^{l-1} + b_{\tilde{z}_t^l}^l\big)$$

$$W_{i,\tilde{z}_t^l}^j \in \mathbb{R}^{1\times dim(h^{l-1})}$$
$$U_{i,\tilde{z}_t^l}^j \in \mathbb{R}^{1\times dim(h^l)}$$

We compute the binary boundary detector, z_t^l, from $\tilde{z}_t^l$ as a stochastic binary neuron:

$$z_t^l \sim \text{Bernoulli}(\tilde{z}_t^l)$$

Chung et al. (2016) mention this method as a valid choice, but choose to compute $\tilde{z}_t^l$ with the "hard sigmoid" function coupled with the slope annealing trick, in which the slope of the hard sigmoid function is increased as training progresses. They compute z_t^l with the deterministic step function:

$$z_t^l = \begin{cases} 1 \text{ if } \tilde{z}_t^l > 0.5 \\ 0 \text{ otherwise} \end{cases}$$

This difference between their model and ours was made simply because it was easier to implement, not because we believe it improves performance. HM-GRU could just as easily compute $\tilde{z}_t^l$ using the same techniques HM-LSTM chose. Like HM-LSTM, the gradient of $\tilde{z}_t^l$ is computed using the straight-through estimator, where the incoming gradient remains unchanged if z_t^l, and is set to zero otherwise.

4 Language Modeling Experiments

We tested the viability of the HM-GRU model by adapting it to do language modeling in a similar fashion to how HM-LSTM was tested. Like the language model built for HM-LSTM in Chung et al. (2016), we built a three layer model with an input embedding layer, and a two-layer fully-connected output module that takes in a weighted combination of all three layers. Each time step tries to predict the next value in the sequence, with negative log-likelihood as the cost function. Gradient vectors were clipped with a maximum norm of 5 to prevent exploding gradients, and Dropout (Hinton et al., 2012) is used minimize the risk of overfitting.

However, unlike the character-level model used by Chung et al. (2016) to evaluate HM-LSTM, we use a word-level model with a vocabulary size of 10,000 words, as done in Graves et al. (2013). We chose a word-level model instead of a character-level one because we wanted maximize the likelihood that the model would learn the longest semantic dependencies possible by removing the need to understand character-level semantics.

We also do not use LayerNorm (Ba et al., 2016) as was done for HM-LSTM (Chung et al., 2013), but only because we could not find a publicly available implementation in Tensorflow. Also unlike Chung et al. (2016), we trained the model with stochastic gradient descent in mini-batches, instead of using the Adam optimizer (Kingma & Ba, 2014), again, because it was more convenient to implement. Lastly, we used average word-level perplexity (Graves et al. 2013) as our evaluation metric instead of the closely related bits-per-character metric used by Chung et al. (2016).

4.1 Penn Tree Bank

We trained our HM-GRU model on the Penn treebank data set in a similar fashion to Graves et al. (2013). During training, the model was unrolled 35 time steps, well above the 21 words in the average length sentence in the data set. The initial learning rate was set to 1.0, although we found an initial rate of 0.7 resulted in lower perplexity for the GRU models.

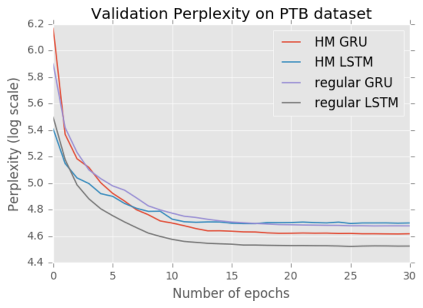

Figure 1: Validation Perplexity on the PTB dataset

From table 1, we can see the regular LSTM outperformed all other models in both perplexity and training time. HM-LSTM and regular GRU performed nearly equivalently, but GRU trained in nearly half the time. HM-GRU performed a few perplexity points better than HM-LSTM in roughly the same amount of time.

Model	Test Perplexity	Training time (hours)
HM-GRU	96.05	6.64
HM-LSTM	103.23	6.78
GRU	103.82	3.25
LSTM	88.67	2.61

Table 1: Perplexity results on the PTB test set

The hierarchical multiscale models require more than double the amount of time to train compared to their canonical counterparts. This is likely due to the stochasticity introduced by the boundary detector states, as predicted by Bengio et al. (2013). Had we used the deterministic indicator function used by Chung et al. (2016) for their boundary detector states, the model would have likely trained more quickly.

We account for the relatively poor performance of our hierarchical models in comparison to that of the HM-LSTM in Chung et al. (2016) to the lack of Layer Normalization, and not using the slope annealing trick to tune the gradients of the stochastic boundary detector gates. Despite, this, the superior performance of HM-GRU compared to HM-LSTM here proves that the new model architecture is at least a viable one, and is possibly an improvement over HM-LSTM.

4.2 Tagged Song Lyrics

Hierarchical multiscale RNNs (HM-RNNs) are designed to learn long-term semantic dependencies more effectively than traditional RNNs. However, the Penn tree bank dataset has only 21 words per example on average, so there are not any long-term dependencies for the HM-RNN to learn.

The other dataset used by Chung et al. (2016), the Hutter Wikipedia Prize corpus of Wikipedia articles (Hutter, 2012), is better suited for this purpose, since it consists of one long document consisting of several other long, highly-structured sub-documents. It still is not ideal for distinguishing HM-RNNs, however, since most of its sub-documents are too long to be input all at once to an RNN during training. Additionally, the sub-documents are not clearly delineated. As a result, the document is typically broken up without regard to its larger structure during training, diminishing the ability of the model to learn the long-term structure.

We wanted to build our own text corpus that we hoped would illustrate the power of HM-RNNs compared to traditional RNNs, so we collected a corpus of song lyrics from the internet. Song lyrics are ideal for this task because song lyrics (in our dataset) have an average of 360 words per song, with simple hierarchical structure that runs throughout each document. This is just short enough for an RNN can be unrolled on an entire song during training, but long enough to contain several layers of hierarchical structure.

The HM-RNN paper states that the higher layers of the hidden states are updated less frequently, and that the higher layers focus on modeling long term dependencies. We hoped that the different layers of our model would each learn the corresponding layers of organization of each song. For example, the first (bottom) hidden state might learn the line breaks, the second hidden layer might learn the structure of the stanzas, and the topmost layer could learn stanza structure, e.g., verse-chorus-verse.

4.3 Corpus Compilation

We obtained lyrics of English songs from the website Songlyrics.com, because its lyrics pages are simple and have a consistent structure, which makes it simple to scrape. The URL structure is http://songlyrics.com/artist-name/song-title-lyrics/, e.g. http://www.songlyrics.com/drake/hotline-bling-lyrics/. We limited the corpus to contemporary pop songs. The reasoning was that contemporary pop has consistent structural properties, such as alternating verses and choruses, similar stanza and line lengths, as well as repeating lines.

We used BeautifulSoup to scrape the HTML. We extracted 13,491 songs in total. We used 70% of the songs for the training set, 20% for the validation set, and the remaining 10% on the test set. Using HTML tags, we identified the beginning of each stanza. We wrote a Python script that annotates the lyrics to indicate start and end of each line with ⟨l⟩ and ⟨/l⟩ respectively, and start and end of each stanza with ⟨s⟩ and ⟨/s⟩ respectively. We added a ⟨start⟩ token to the beginning of each example as well. This is to aid the model in learning the structure better, as used in the GhostWriter model by Peter Potash et al. (2015).

4.4 Song Lyrics Experiments

We compared the model against an HM-LSTM model with the same structure and hyperparameters, and also against three-layer instantiations of the regular LSTM and GRU models. For each experiment, we varied the number of hidden units in each cell, the initial learning rate, and number of epochs trained.

We also tried unrolling each model to a larger number of time steps, to give the model an opportunity to learn longer-term structure of the data. We tried 90, 180 and 360 time steps. For those experiments, the resulting test perplexities were surprisingly within 0.2 of each other. It seems that, on this dataset, increasing the number of time steps to unroll for training has a negligible effect on the model's performance. Table 2, shown below, contains the best results for each of the four models.

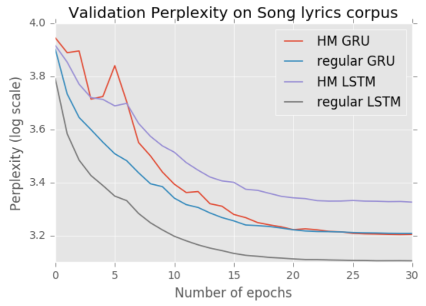

Figure 2: Validation Perplexity on the song lyrics corpus

Model	Test Perplexity	Training time (hours)
HM-GRU	24.562	16.103
HM-LSTM	27.471	19.732
GRU	24.682	8.569
LSTM	22.124	8.357

Table 2: Perplexity on the test set of song lyrics corpus

From table 2, we can see the regular LSTM outperformed all other models in both perplexity and training time. HM-GRU slightly outperforms GRU, but trains in twice the time. HM-LSTM does the worst, and takes the longest to train. Again, the slow training time of the HM-RNNs is likely due to the stochasiticity of the boundary dectector.

5 Conclusion

Our experiments demonstrate the viability of HM-GRU as an RNN for sequential data. It performs similarly well, and sometimes better, than our implementation of HM-LSTM in language modeling tasks. We also constructed a song lyrics corpus, with the hope that it would would illustrate the comparative advantage of HM-RNNs. The HM-RNNs achieved a similar level of performance on this dataset to their traditional counterparts, but unfortunately did not outperform them as hoped. It is not clear why this was the case, but we suspect that the lyrics were not long or semantically complicated enough to go stretch the capabilities of the LSTM and GRU cells.

Appendix

All the data and Tensorflow code used in this project is available at http://github.com/pinesol/grumpy/.

Statement of Collaboration

Alex designed and implemented HM-GRU and HM-LSTM, and adapted the Google RNN tutorial code to train them. Sean scraped, cleaned, pre-processed and tagged the lyrics data, visualized perplexity curves. Both of us trained multiple models and recorded the results of the training. Alex and Sean both wrote the paper together.

Acknowledgments

We would like to acknowledge the Google Tensorflow team for proving much of the code we used (Google, 2016) to train our models. We would also like to thank the R2RT blog (R2RT, 2016) for providing the code we used to implement stochastic binary neurons in our models. We would like to thank the TA, Tian Wang, for helping us understand the model. We would also like to thank Sam Bowman for his guidance in implementing our model.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *CoRR*, 1607.06450.

Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. 2016. Hierarchical multiscale recurrent neural networks. *CoRR*, abs/1609.01704.

Salah El Hihi and Yoshua Bengio. 1995. Hierarchical recurrent neural networks for long-term dependencies. In *Proceedings of the 8th International Conference on Neural Information Processing Systems*, NIPS'95, pages 493–499, Cambridge, MA, USA. MIT Press.

Alex Graves. 2013. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580.

G. Hinton. 2012. Neural networks for machine learning. *Coursera, video lectures.*

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

Marcus Hutter. 2012. The human knowledge compression contest. *http://prize.hutter1.net/.*

Google Inc. 2016. Recurrent neural networks. *https://www.tensorflow.org/tutorials/recurrent/.*

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Jan Koutník, Klaus Greff, Faustino J. Gomez, and Jürgen Schmidhuber. 2014. A clockwork RNN. *CoRR*, abs/1402.3511.

Wang Ling, Isabel Trancoso, Chris Dyer, and Alan W. Black. 2015. Character-based neural machine translation. *CoRR*, abs/1511.04586.

Eric Malmi, Pyry Takala, Hannu Toivonen, Tapani Raiko, and Aristides Gionis. 2015. Dopelearning: A computational approach to rap lyrics generation. *CoRR*, abs/1505.04771.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313–330, June.

Lluís Màrquez, Chris Callison-Burch, Jian Su, Daniele Pighin, and Yuval Marton, editors. 2015. *Proceedings of the 2015 Conference on Empirical Methods in*

Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015. The Association for Computational Linguistics.

Peter Potash, Alexey Romanov, and Anna Rumshisky. 2015. Ghostwriter: Using an LSTM for automatic rap lyric generation. In Màrquez et al. (Màrquez et al., 2015), pages 1919–1924.

R2RT. 2016. Binary stochastic neurons in tensorflow - r2rt. *http://r2rt.com/binary-stochastic-neurons-in-tensorflow.html*.