Prescient Language Models: Multitask Learning for Long-term Planning in LSTM's

Joyce Xu
External Collaborators: Akhila Yerukola, John Hewitt, Abi See
Project Mentor: Abi See
jexu@stanford.edu

Abstract

Many applications of language models, such as those in natural language generation, are notoriously susceptible to undesirable and difficult-to-control local behavior such as repetition, truncation, or rapid topic changes. We hypothesize that attending to *future* context in the form of "planning" may be key to stronger, more robust models, and propose a multi-task language model architecture that jointly predicts the next word in a sequence along with a vector representation of future content. Our analysis reveals that (a) certain internal states within the standard encoder-decoder stacked LSTM architecture contain more long-term information than others, (b) hidden states consistently hold more information on future content than embedding-based representations of the input context, and (c) LSTM's *are* able to form a general syntactic "plan" for at least a few upcoming tokens. We conclude by exploring and benchmarking how multitask learning with future predictions can impact existing language modeling behavior.

1 Introduction

Recent efforts in deep learning-based language modeling have often focused on capturing long-term dependencies in text. However, beyond the model's ability to successfully maximize the probability of the next word, we have little way of evaluating *how* the model is using or interpreting these dependencies. In particular, we have yet to benchmark and analyze a model's ability to use context for long-term *planning*.

The presence and reliability of long-term planning may be critical to a language model's performance. Especially as sentence length and complexity increases, having a sense of downstream content could help the model predict the correct tokens and sub-phrases leading up to that content. More concretely, on a local level, having a hypothesis about soon-upcoming words can help the model select grammatically or syntactically appropriate token(s) to transition to those words.

We attempt to answer the question of whether or not RNN's "plan ahead" by probing the hidden states of a stacked LSTM. Our hypothesis is that if RNN's do actively plan for downstream content, some representation of that downstream content should be captured in (and easily extractable from) the model's hidden state. We proceed with a series of experiments to benchmark an LSTM's ability to predict a longer-term future, and additionally introduce a multi-task language model that simultaneously attempts to predict future content in order to encourage this planning behavior.

2 Related Work

Our work draws on a growing body of literature around the analysis of LSTM's, how they learn, and what they understand. [2] conducted a widespread hyperparameter search and analysis of eight different LSTM variants, which toggled the presence of various gates and activation functions within

a LSTM to determine how each component contributes to the model's performance. Both [6] and [7] proposed novel approaches to interpreting the hidden state's "memories" through visual analysis, and [6] in particular was able to show that even small LSTM's capture grammatical functions of words in their hidden states.

[3] ran a similar experiment to ours: they investigated how language models use prior context (and how much context is used), and demonstrated that the model is able to use up to 200 tokens of context, but word order and sequential structure is only retained in the most recent 50 tokens. The analysis component of our work investigates language models' understanding of *future* content in a parallel manner to [3], but we do so by attempting to extract information from hidden states and comparing the relative contribution of each layer in the stacked LSTM.

Our work is also inspired by recent interest and success in multitask learning in natural language processing. Works such as the "Natural Language Decathlon" in [4] have shown that training more general models on multiple tasks can outperform task-specific models on their target task, due to transfer across tasks. In an approach similar to ours, [8] attempt to improve the ability of RNN's to capture long-term dependencies by adding an *auxiliary task* of reconstructing either past events or upcoming events from a random hidden state in the prior sequence. They successfully demonstrate that this auxiliary loss helps the model hold on to long-term dependencies and achieve comparable performance to strong LSTM and transformer baselines.

Our work extends existing language modeling architectures to use auxiliary losses as well, but differs from [8] in two key ways. Firstly, our auxiliary losses are not based in *reconstruction* of the past, but rather in *prediction* of the future. Secondly, we feed our predictions back into the model, and allow it to incorporate its secondary task into executing its primary one.

3 Approach

In an attempt to understand and improve language models' ability to plan for long-term downstream content, we augment existing language models with an additional "future prediction" task. We begin by proposing two different formulations of this future prediction task. Then, for each formulation, our contribution is two-fold: we firstly run a series of analysis experiments examining how well this auxiliary task can be learned on its own and which parts of the base language model architecture (i.e. which layer) contains the most salient information about future content, and secondly demonstrate how multitask learning with this task can contribute to the primary goal of language modeling.

3.1 Future Prediction Task Formulation

We explore two primary formulations of the "future prediction" task. This first is predicting single-word targets downstream in the sequence. In particular, if the input sequence to the language model at a given timestep is $x_{t-c}, ..., x_{t-2}, x_{t-1}$, and the language modeling target would have been the next word x_t, our future prediction model instead attempts to predict *one* of the next 5 words in the sequence. Concretely, for a "+1" auxiliary task, we predict x_{t+1}; for a "+2" task, we predict x_{t+2}; and so forth. In our analysis, we compare model behavior and performance across the different choices of n for "+n" predictions.

The second formulation we explore is predicting an *average word embedding* of an upcoming window of words. Similarly to our single-word prediction task, we also experiment with a variety of window sizes, specifically 1, 3, and 10. With a window size of 3, for example, our target "future prediction" vector is the average of the word embeddings for x_{t+1}, x_{t+2}, and x_{t+3}. We refer to this as our average BOW (bag-of-words) model, and also provide analysis for differing performance across window sizes.

3.2 Part 1: Standalone Future Prediction Models

Before attempting any multitask learning, we design a series of experiments to determine how well these standalone future prediction tasks can be achieved on their own.

Note that for all experiments, we utilize a base language model that consists of three stacked, unidirectional LSTM's with weight tying in the encoder and decoder. The model is trained using Salesforce's open-source AWD Language Model Toolkit, with the hyperparameters suggested by [5].

3.2.1 Baselines

We implement two primary baselines for the standalone prediction of "+n" words. The first baseline is simply the unigram perplexity of the validation dataset, which is calculated purely as a function of individual word counts. The second baseline is a shallow feed-forward neural net which attempts to predict the target future word based on v_s, a frequency-weighted average of the word embeddings in the current sequence propose by [1]:

$$v_s = \frac{1}{|s|} \sum_{w \in s} \frac{a}{a + p(w)} v_w$$

In the expression above, a is a hyperparameter controlling the "smoothing" factor, $p(w)$ is the count-based probability of a word in the training set, and v_w is the word embedding.

The second baseline is similar to our standalone prediction models, except that our models leverage LSTM hidden states instead of word embeddings. Our experiments were, in part, motivated by intuition that a well-trained LSTM language model should contain more information about downstream content than simple word embeddings of the context (since the LSTM already has access to those embeddings).

3.2.2 Our Models

For both formulations of future prediction tasks, we construct a shallow (1-layer) neural net that attempts to predict the target future vector from the final state of a hidden layer in our trained base language model. In these experiments, we freeze the weights of our base language model and train only our future prediction model (and thus ignore the primary language modeling task and loss). We repeat this experiment using inputs from either the first, second, or third hidden layers in the stacked LSTM language model, which we will henceforth refer to as $h1$, $h2$, and $h3$ respectively.

The model architecture for "+n" target word predictions can be found in Figure 1a. Note that these "+n" prediction models use the same decoder matrix as the base language model and are trained using a cross-entropy loss with the target word. The average BOW prediction models, however, do not use the decoder, since their target is not necessarily a word at all. Thus, they do not use a cross-entropy loss. Instead, the average BOW model is trained on the cosine distance to the true average vector.

3.3 Part 2: Multitask Language Modeling

Finally, we propose a method and model architecture for incorporating these auxiliary tasks into training for the primary language modeling task.

In the case of "+n" future word predictions, consider the case of predicting x_{t+1}. The loss for our multitask model becomes a weighted sum of the primary LM loss and the future prediction loss:

$$l_{+n}(\theta_{LM}, \theta_F) = -\frac{1}{T} \sum_{t=1}^{T} \log(p_{x_t} | x_{t-c}, ..., x_{t-1}; \theta_{LM}) + \lambda \log(q_{x_{t+n}} | x_{t-c}, ..., x_{t-1}; \theta_F),$$

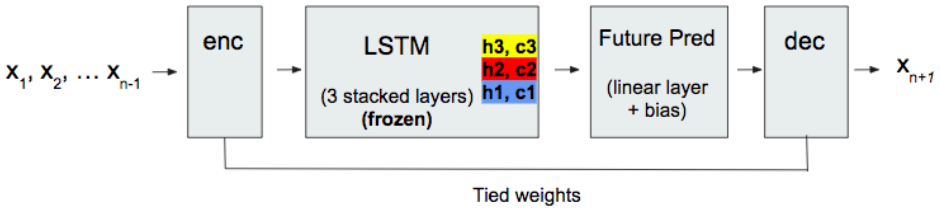
where p and q are the respective predicted probability distributions over words for the main LM and the future prediction model, θ_{LM} and θ_F are the models' respective weights, and λ is a hyperparameter controlling the relative weighting of the two losses.

For the average-BOW prediction model, the loss of the future prediction term is a cosine distance loss instead of a cross-entropy loss:

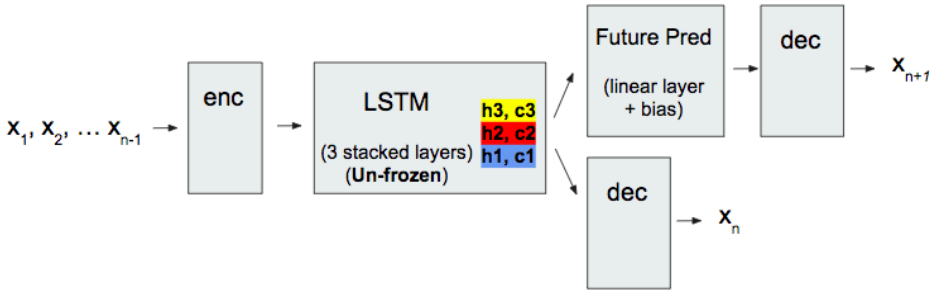
$$\hat{x}_{bow} = \text{FutureModel}(x_{t-c}, ..., x_{t-1}; \theta_F)$$

$$l_{BOW}(\theta_{LM}, \theta_F) = -\frac{1}{T} \sum_{t=1}^{T} \log(p_{x_t} | x_{t-c}, ..., x_{t-1}; \theta_{LM}) + \lambda(1 - \text{cos\_sim}(x_{bow}, \hat{x}_{bow})).$$

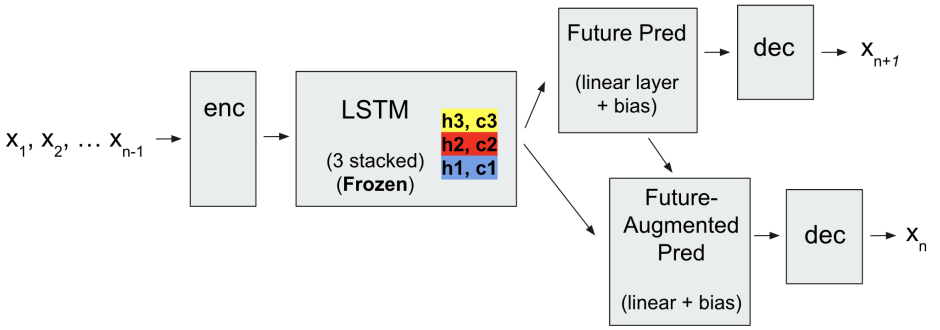
For our multitask language model, we also run two variations of experiments. In the first variation, which is shown in Figure 1b, we un-freeze the weights to our base language model and fine-tune it with both the language modeling loss and the auxiliary future loss.



(a) Standalone "+1" future prediction model



(b) Multitask LM augmented with a +1 future prediction task



(c) Multitask LM with a +1 future prediction AND a "future-augmented" combination layer

Figure 1: Network architectures for +1 prediction task: standalone and auxiliary

In the second variation, we keep the weights of the base language model frozen, but we add another layer that concatenates the LSTM's final hidden state with our predicted future vector, and pass the output of this layer through the decoder to get our modified language modeling prediction. The full model is still trained with both losses. During training of the final layer, however, we begin by feeding in the "gold" embedding of the future, and as training progresses, gradually mix-in and feed our own predictions from the future prediction module. This gives our future prediction module time to train and learn relatively reasonable predictions before passing them on to the last layer, which helps stabilize training.

4 Experiments

4.1 Data

Our models are evaluated on the WikiText language modeling dataset. We use WikiText-2, which contains around 2.5 million word-level tokens and 33,278 unique vocabulary words, to train and evaluate our primary language model. To train our standalone future prediction model, we use a 2-million word subset of the Wikitext-103 dataset.

Table 1: "+n" Standalone Prediction Perplexities

+n	h1	h2	h3	weighted-ave	unigram
1	408.2	**243.67**	262.67	793.77	1001.87
2	627.88	**418.58**	445.64	841.71	1001.87
3	739.73	**529.24**	572.14	966.45	1001.87
4	794.68	**595.89**	640.08	973.25	1001.87
5	848.06	**631.22**	705.56	929.81	1001.87

Table 2: Average-BOW Standalone Prediction Cosine Similarities

window size	cosine similarity
1	0.251
3	0.516
10	0.712

We choose to use a separate (but similar) set of documents to train the future prediction model in order to avoid the possibility of the model simply memorizing sequences seen in training during evaluation. Thus, we sample documents from Wikitext-103 that do not overlap with Wikitext-2 to form our future prediction training set, and evaluate that model on our original Wikitext-2 validation set.

4.2 Experimental Details

For our future prediction models, we froze the main language model weights and trained the future model for 200 epochs (with early stopping), with a batch size of 80 and a learning rate of 2.0.

For the multitask prediction model in which we un-freeze the base language model, we train for 750 epochs at a learning rate of 2.0. In the variation with a frozen base language model but an additional layer that incorporates predictions from the future module, we train for 250 epochs with a learning rate of 1.0.

4.3 Results

4.3.1 Standalone Predictions

In Table 1 and 2, we present our experimental results for the standalone prediction models, in which the only task and loss is the future prediction. Table 1 shows results from training our "+n" prediction model on $h1$, $h2$, and $h3$, as well as our unigram baseline and weighted average word embedding baseline. Note that the unigram baseline is the same for all values of "+n", because the unigram distribution for words is independent of our target prediction task.

For "+n" tasks, all of our future prediction models outperform both baselines, as expected. Furthermore, the prediction models that used hidden states from the second layer ($h2$) also consistently perform the best across all values of "+n", which was a surprising result for us. We had assumed that $h3$ would contain the most information about future content, as it is the highest layer in the LSTM, and conventional wisdom points to higher layers being responsible for more abstract reasoning. However, we reason that because the original language model encourages the outputs of $h3$ to be close in the word embedding space to the next word, perhaps $h3$ is more responsible for specificity in prediction of the immediate upcoming word, and more temporally abstracted, "long-term" reasoning is actually occurring in the layer before it. This suggests that **multi-layered (stacked) LSTM's may do general long-term content planning *before* next-word prediction** in the forward pass.

For the average BOW tasks, we do not have any baselines to compare against, because it is unclear what a baseline for the average future word embeddings would look like. However, in the results we do observe a clear trend of increasing cosine similarity as the window size increases. This result is consistent with our intuition: an average of a large upcoming window of words likely has much less variance than that of a small window, and as a result is perhaps easier to predict.

Table 3: Multitask Language Model Perplexities

Model	Perplexity
Base LM	71.73
Base LM + fine-tuning	*69.51*
Multitask (n1h2): aux loss	**69.25**
Multitask (n1h2): aux loss + feed-in	65.97
Multitask (win1): aux loss + feed-in	65.84
Multitask (win3): aux loss + feed-in	**65.6**
Multitask (rand): feed-in	*66.6*

4.3.2 Multitask Models

As shown in experimental results in Table 3, the multitask language model only improves upon the base language model by a maximum of 1 perplexity point.

With the first variation of our multitask language modeling task, in which we simply train the base language model with an additional, auxiliary loss signal from the "+1" prediction task, we can improve upon the original language model's loss by about 2 perplexity points, dropping from 71.73 to 69.25. However, simply fine-tuning the original language model for the same number of epochs also achieves a perplexity of 69.51, so most gains can be attributed to further training of the base language model.

In our second variation, we not only train with an auxiliary future loss, but also feed this predicted "future vector" back in to the modified language model to help it make its next-word prediction. Our best result is achieved with an "average-BOW" prediction module that uses a target window size of 3, with which we obtain a validation perplexity of 65.6.

While this is a large gain over the original perplexity of 71.73, we note that this model adds a significant number of parameters: the original language model has 33,556,078 parameters, and the modified, multitask language model (which includes the future prediction module as well as the final combination layer) has a total of 60,672,156 parameters. We run a control experiment with the same architecture, but train with a random "future vector" and without a future prediction loss; this control achieves a validation perplexity of 66.6, which is only one point higher than our multitask model. Again, most gains can be attributed to increase in model capacity as opposed to utility of future predictions.

5 Analysis

5.1 Standalone Future Predictions

We examine the actual word predictions made on the validation set by our "$+1$" model, which achieved the lowest perplexity (243.67) of all the $+n$ tasks. The vast majority of top predictions are common, generic words such as "and", "to", "which", etc.; the model is usually unable to extract or confidently predict rarer, more specific upcoming words from the hidden state. However, the part of speech, word type, and usage of the top "$+1$" predictions are generally correct, even if they do not always match the exact upcoming ground truth in validation.

The first two examples in Table 4 are fairly representative of the average behavior of the "$+1$" prediction module. We see in the first example that the majority of the top predicted x_{t+1} words are prepositions (e.g. to, in, from, by), which is the the correct part of speech for the upcoming token. In the second example, several top x_{t+1} predictions are numbers (e.g. "four," "three," "two"), which makes perfect semantic sense in context: "The first pair of <unk> is armed with *three* large ...". The consistent appearance of these sorts of predictions suggests that **the language model forms a rough *syntactic* plan for at least a few upcoming words** at each timestep.

Table 4: "+1" Standalone Prediction Examples

gold context | gold x_n x_{n+1} x_{n+2}	**Base LM preds (top 5)**	**Multitask LM preds (top 5)**	**+1 Future preds (top 10)**
and parts of the Black Sea . It is closely | related to the	found, related, distributed, classified, associated	related, distributed, found, considered, endemic	to, in, „ the, from, and, that, by, with, about
animals . <eos> The first pair of <unk> is armed | with a large	with, in, „ by, .	with, in, by, at, on	the, a, <unk>, four, two, three, with, The, which, and
wrestlers <unk> chose . <unk> 's four wrestlers were James | Storm , Matt	<unk>, „ and, Hardy, Moore	<unk>, „ ", and, R.	„ and, <unk>, ., who, 's, (, from, in, Williams
<unk> chose . <unk> 's four wrestlers were James Storm | , Matt Morgan	„ and, (, ., who	„ and, (, ., <unk>	<unk>, and, the, „ who, a, but, The, Williams, David
chose . <unk> 's four wrestlers were James Storm , | Matt Morgan ,	who, <unk>, the, a, Kevin	<unk>, who, the, a, and	<unk>, and, „ the, of, was, Ryan, had, @-@, Michael
. <unk> 's four wrestlers were James Storm , Matt | Morgan , Robert	<unk>, and, „ Lee, Hardy	<unk>, „ Hardy, and, Lee	„ and, <unk>, (, from, @-@, of, 's, on, was
to 60 centimetres (24 in) and weighing up | to 5 -	to, in, and, with, by	to, with, by, at, and	4, 2, 1, 60, 150, 230, 0, 8, 12, 70
60 centimetres (24 in) and weighing up to | 5 - 6	2, 1, 3, 5, 12	2, 5, 3, 4, 12	@.@, cm, mm, %, centimetres, feet, @,@, meters, inches, kilograms

More impressively, the "+1" prediction module is able to learn structure such as lists and number/unit patterns. The next four examples in the table show the language model and future predictions as we slide across a list of names, i.e. "James Storm, Matt Morgan, Robert Roode." In each case, the "+1" prediction module is able to anticipate the correct placement and number of <unk>'s to represent the named entities, as well as predict a comma in the correct places.

The "+1" module also appears to make sensible anticipations about upcoming numbers and units of measurements. Even when the predictions differ from the ground truth, such as the last example of Table 4 (in which the ground truth went on to describe a range of values whereas the "+1" module predicted units of measurement), the "number + units" structure again suggests the presence of a reasonable syntactic *plan* for the future.

5.2 Base vs. Multitask Language Model

As we saw in our experimental results, there is not a significant difference in perplexity between the base language model and the various future prediction-augmented language models. Table 4 already supports these results: in general, the top predictions for the base LM and the multitask LM are incredibly similar. As noted in Table 5, only in 21.71% of the validation set did the two models differ in their predictions of the single most likely word, and even then, the models typically had each other's top predictions as their own second- or third- most likely word.

Still, there are a few specific scenarios in which being trained to jointly predict the future helps the language modeling prediction itself.

Firstly, note from Table 5 that in the cases where the base and multitask models *do* differ in predictions, the multitask language model makes the "correct" future prediction on the validation set with a greater frequency. This trend holds across the $+1$, $+2$, and $+3$ modules, although the gap between how frequently the two models are correct closes with larger $+n$. Since the future prediction perplexities

Table 5: Base vs. Multitask LM Differences Across "+n" Models

	+1 pred module	**+2 pred module**	**+3 pred module**
% cases where base and multitask LM differ in single-word top prediction	21.71	19.44	19.75
% of differing cases where base LM is correct	9.86	11.97	11.75
% of differing cases where multitask LM is correct	12.78	12.50	12.77

Table 6: Base LM vs. "+1"-Augmented Multitask LM Examples

gold context | gold x_t x_{t+1} x_{t+2}	**Base LM preds (top 5)**	**Multitask LM preds (top 5)**	**+1 Future preds (top 10)**
was bombed by the Japanese for the first time on | 26 January 1942	the, 20, 23, 18, 24	20, 18, 1, 19, 24	November, April, February, January, December, October, June, March, May, August
<eos> = = Early life = = <eos> <eos> <unk> | was born in	<unk>, ,, was, (, is	was, (, is, <unk>, ,	born, a, was, the, in, <unk>, an, on, ,, is

increase with larger $+n$, these predictions become less accurate and less useful to the multitask model, which explains the closing gap.

In Table 6, the first example demonstrates that the future module, which successfully predicted a month as the "+1" word, helped the multitask model plan for introducing a properly formatted date in its upcoming tokens.

The second example highlights that jointly predicting the future can help models learn where and how to initialize new continuous sequences. Language in the real world is not uniformly continuous; in the Wikitext dataset, documents are split into subsections through token-deliminated headings. It is a much more difficult task to get language models to properly initialize and sustain a reasonable introduction after such breaks, because there is less syntactic/semantic direction provided by its context. In this case, jointly predicting upcoming tokens (and using that prediction) can help the multitask model identify a stronger, more sensible starting sequence.

While a few such examples of the future prediction module proving useful to language modeling do exist, it is critical to note that in general, this module does *not* appear to make a significant difference to the original language modeling task. Because most outputs of the future prediction module are — albeit reasonable in part of speech or usage — *not* particularly specific or semantically informative, **it remains rare that these future predictions sway the base language model's original choice of words.** In cases where future predictions *are* specific or topically relevant, such as the example in which the "+1" module predicted upcoming months, they do appear useful to the multitask model. However, these cases are few and far enough in between that we do not see a significant qualitative or quantitative difference.

6 Conclusion

Our research has contributed what is, to our knowledge, the first benchmarks for the long-term planning capacity of LSTM's. We have found that, contrary to conventional wisdom and intuition, planning does not actually occur in the final, most abstracted layer of the LSTM, but rather *before* that last layer and before next-word predictions are made. We also show that LSTM's have the capacity to form general syntactic plans for upcoming words, and can learn and predict structures such as lists and date formats. However, multitask learning with an additional "future prediction" loss does not improve upon base language modeling abilities significantly, since in cases where the base language model has an initially *incorrect* next-word prediction, it is rare that these future predictions are specific and informative enough to correct that choice.

References

[1] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. 2017.

[2] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *CoRR*, abs/1503.04069, 2015.

[3] Urvashi Khandelwal, He He, Peng Qi, and Dan Jurafsky. Sharp nearby, fuzzy far away: How neural language models use context. *CoRR*, abs/1805.04623, 2018.

[4] Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. The natural language decathlon: Multitask learning as question answering. *CoRR*, abs/1806.08730, 2018.

[5] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. An Analysis of Neural Language Modeling at Multiple Scales. *arXiv preprint arXiv:1803.08240*, 2018.

[6] Yao Ming, Shaozu Cao, Ruixiang Zhang, Zhen Li, Yuanzhe Chen, Yangqiu Song, and Huamin Qu. Understanding hidden memories of recurrent neural networks. *CoRR*, abs/1710.10777, 2017.

[7] Hendrik Strobelt, Sebastian Gehrmann, Bernd Huber, Hanspeter Pfister, and Alexander M. Rush. Visual analysis of hidden state dynamics in recurrent neural networks. *CoRR*, abs/1606.07461, 2016.

[8] Trieu H. Trinh, Andrew M. Dai, Thang Luong, and Quoc V. Le. Learning longer-term dependencies in rnns with auxiliary losses. *CoRR*, abs/1803.00144, 2018.