Character-Level Language Models Using Recurrent Neural Networks

Hosang Yoon

September 24, 2017

1 Introduction

The task of making computers work with inputs or outputs expressed in human languages is known as natural language processing. The traditional approach to this task is based on representing sentences as sequences of words where those words form the fundamental indivisible units of a sentence, including models based on n-grams [1] and word2vec [2]. While these models have been highly successful, one drawback of the word-based models is their difficulty in handling words outside of the training corpus. Recently, character-level language models have drawn attention as a potential alternative to the word-level models [3], which naturally avoids the out-of-vocabulary issue. This approach was not considered feasible until recently, but developments in recurrent neural networks (RNNs), such as long short-term memory (LSTM) [4] that can learn correlations in its inputs hundreds of times steps (characters) away, now make it possible to tackle such a problem directly. The wide availability of parallel GPU-based computation libraries for training these networks also helped in bringing down the prohibitively lengthy training times to a tractable level. In this report, we will explore a number of different RNN architectures for the character-level language modeling task enabled by these developments.

2 Problem definition

A character-level language model can be stated as a model that predicts the next character in a text given the text up to the current location in the text. More concretely, the model produces a probability distribution for time steps $t = 1, \cdots, T$ defined by $y_t : \mathcal{C} \to [0, 1]$, where $\mathcal{C}$ is the set of characters and $\sum_{x' \in \mathcal{C}} y_t(x') = 1$, which estimates the probability distribution for the next character given past and current input as

$$y_t(x'_{t+1}) = \Pr(x'_{t+1} | x_{\leq t}; \theta), \tag{1}$$

where x' is a character in $\mathcal{C}$, x is a character from the data, and θ represents model parameters. The estimated likelihood for the model to generate a given data sequence $\mathbf{x}$ is $\Pr(\mathbf{x}) = \prod_{t=0}^{T-1} \Pr(x_{t+1} | x_{\leq t}; \theta) = \prod_{t=0}^{T-1} y_t(x_{t+1})$. The objective of the problem is to configure θ to maximize the likelihood of this sequence, which is equivalent to minimizing the negative log likelihood given by [3]

$$\mathcal{L}(\mathbf{x}; \theta) = -\sum_{t=0}^{T-1} \log y_t(x_{t+1}). \tag{2}$$

To arrive at this goal, the output function of Eq. 1 is implemented using an RNN with a softmax output layer, which is trained using gradient descent with Eq. 2 as the loss function. This is schematically illustrated in Fig. 1. The objective of this report will be to implement, train, and evaluate such models. Furthermore, variations of the basic model will be considered in order to improve upon the baseline result.

3 Dataset

The text8 dataset [6] was used for training the model, which is a commonly used dataset for language modeling in the academic literature. The dataset consists of English sentences totaling to 100,000,000 characters

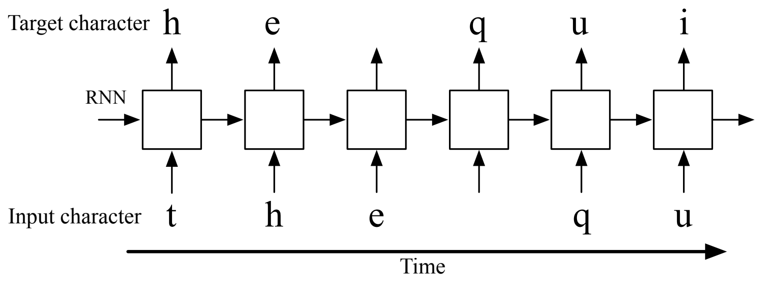

Figure 1: Training a character-level RNN language model. Negative log of the probability predicted for the target character is the loss for each time step. Adapted from [5].

```
 anarchism originated as a term of abuse first used against early working class radicals
including the diggers of the english revolution and the sans culottes of the french
revolution whilst the term is still used in a pejorative way to describe any act that
used violent means to destroy the organization of society it has also been taken up
as a positive label by self defined anarchists the word anarchism is derived from
the greek without archons ruler chief king anarchism as a political philosophy is the
belief that rulers are unnecessary and should be abolished although there are differing
interpretations of what this means anarchism also refers to related social movements that
advocate the elimination of authoritarian institutions particularly the state the word
anarchy as most anarchists use it does not imply chaos nihilism or anomie but rather a
harmonious anti authoritarian society in place of what are regarded as authoritarian
political structures and coercive economic institutions anarchists advocate
```

Figure 2: The first 1024 characters of the text8 dataset.

extracted from a dump of Wikipedia database. The text is subsequently preprocessed to remove markups, and then transformed so that only lowercase characters and non-repeating plain spaces are contained in the text. Fig. 2 shows the first 1024 characters of the text8 dataset, clearly showing that the text has been thoroughly cleaned. No further preprocessing of data was required, other than converting the characters to one-hot vectors of dimension 27 (representing ‘ ’, ‘a’, $\cdots$, ‘z’) to be used as the input and target for the RNN.

The text8 dataset is commonly split into 90% training, 5% validation, and 5% test sets starting from the beginning of the text, as suggested in [7]. To confirm the evenness of such splits, the occurrence probability of the 27 characters in each of the splits was plotted as shown in Fig. 3. It can be seen that the three sets exhibit very similar distributions of characters with ‘ ’, ‘e’, and ‘t’ as the three most common characters, as would be expected from a large English text. The entropy values of the three distributions ignoring inter-character correlations[1] are 4.124, 4.120, 4.124 bits per character, respectively, indicating that the training, validation, and test sets were split impartially in this measure.

4 Evaluation metrics

The standard measure of performance for character-level language models is bits-per-character (BPC) [9]. BPC is calculated as BPC $= \mathbb{E}[-\log_2 \Pr(x_{t+1}|x_{\leq t};\theta)] = \frac{1}{T\log 2}\mathcal{L}(\mathbf{x};\theta)$, which is the average cross entropy between the target and the output given an input sequence, expressed as a binary logarithm. BPC can be thought of as the average number of bits needed per character of output to encode the error between the target and the output given an input sequence.

Test set BPC values on the text8 dataset for character-level language models built from various RNN architectures exist in the literature [7, 9, 10], which can be used as a direct and objective comparison of the results obtained here to those of others. Some notable results are summarized in Table 1. As of 2017, the

[1]This inflates the true entropy of English text estimated to be 0.6–1.3 bits per character [8].

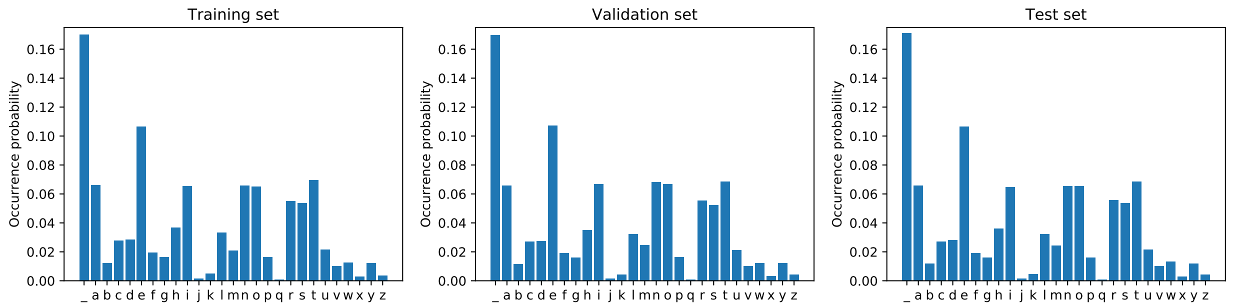

Figure 3: Character occurrence probability in the training, validation, and test set of the text8 dataset.

Model	BPC
td-LSTM (Zhang et al., 2016)	1.63
HF-MRNN (Mikolov et al., 2012)	1.54
Skipping-RNN (Pachitariu et al., 2013)	1.48
MI-LSTM (Wu et al., 2016)	1.44
BatchNorm LSTM (Coojimans et al., 2016)	1.36
LayerNorm HM-LSTM (Chung et al., 2016)	1.29
Recurrent Highway Network (Zilly et al., 2016)	1.29

Table 1: Test set BPC values of various RNN architectures on the text8 dataset. Table adapted from [9, 10].

state-of-the-art BPC values for the text8 dataset is around 1.3, and moderate-performance networks are in the range 1.4–1.5. The objective of this report will be to attempt to build a model reasonably close to these ranges, within reasonable training time and model complexity constraints.

In addition to calculating the error metrics relative to the test set, the model can be used as a generative model of text starting from an arbitrary input sequence, and then drawing the next input character repeatedly from the multinomial distribution described by Eq. 1. While qualitative, the generated sentences can be examined for naturalness to the human eye. Specifically, we may test the model's ability to respond reasonably to words outside of its training corpus, which was a rationale for studying the character-level language models.

5 Implementation

The RNN training and evaluation platform was built using Theano [11]. High-level libraries such as Keras [12] or Lasagne [13] were not appropriate for the purposes of this report, as the relatively new RNN architectures to be explored here are not included these libraries, and it is difficult to weave these types of features into existing frameworks. Theano also allowed the implementation of customized backpropagation through time algorithm to be discussed below.

5.1 RNN architectures

The main RNN architecture to be explored in this report is the LSTM [4] with forget gates [14] and peephole connections [15], defined by

$$\begin{aligned}
\mathbf{i}_t &= \sigma(W_{\mathrm{ix}}\mathbf{x}_t + W_{\mathrm{ih}}\mathbf{h}_{t-1} + \mathbf{w}_{\mathrm{ic}} \circ \mathbf{c}_{t-1} + \mathbf{b}_{\mathrm{i}}), \\
\mathbf{f}_t &= \sigma(W_{\mathrm{fx}}\mathbf{x}_t + W_{\mathrm{if}}\mathbf{h}_{t-1} + \mathbf{w}_{\mathrm{fc}} \circ \mathbf{c}_{t-1} + \mathbf{b}_{\mathrm{f}}), \\
\mathbf{c}_t &= \mathbf{i}_t \circ \tanh(W_{\mathrm{cx}}\mathbf{x}_t + W_{\mathrm{ch}}\mathbf{h}_{t-1} + \mathbf{b}_{\mathrm{c}}) + \mathbf{f}_t \circ \mathbf{c}_{t-1}, \\
\mathbf{o}_t &= \sigma(W_{\mathrm{ox}}\mathbf{x}_t + W_{\mathrm{oh}}\mathbf{h}_{t-1} + \mathbf{w}_{\mathrm{oc}} \circ \mathbf{c}_t + \mathbf{b}_{\mathrm{o}}), \\
\mathbf{h}_t &= \mathbf{o}_t \circ \tanh(\mathbf{c}_t),
\end{aligned} \tag{3}$$

where $\mathbf{x}_t$ is the input, $W_{\cdot}$ and $\mathbf{w}_{\cdot}$ are matrix and vector weights, respectively, $\mathbf{b}_{\cdot}$ are bias weights, $\mathbf{i}_t$, $\mathbf{f}_t$, and $\mathbf{o}_t$ are input, forget, and output gate levels, respectively, $\mathbf{c}_t$ is the memory cell state, $\mathbf{h}_t$ is the output activation state, $\sigma(\cdot)$ is the sigmoid function, and $\circ$ is the element-wise multiplication operator. LSTM networks have been extremely successful in a large number of sequential tasks, including the character-level language modeling task as can be seen in Table 1.

Another well-known RNN architecture is the gated recurrent unit (GRU) [16], defined by

$$\begin{aligned}
\mathbf{r}_t &= \sigma(W_{\mathrm{rx}}\mathbf{x}_t + W_{\mathrm{rh}}\mathbf{h}_{t-1} + \mathbf{b}_{\mathrm{r}}), \\
\mathbf{u}_t &= \sigma(W_{\mathrm{ux}}\mathbf{x}_t + W_{\mathrm{uh}}\mathbf{h}_{t-1} + \mathbf{b}_{\mathrm{u}}), \\
\mathbf{c}_t &= \tanh(W_{\mathrm{cx}}\mathbf{x}_t + \mathbf{r}_t \circ (W_{\mathrm{ch}}\mathbf{h}_{t-1}) + \mathbf{b}_{\mathrm{c}}), \\
\mathbf{h}_t &= (1 - \mathbf{u}_t) \circ \mathbf{h}_{t-1} + \mathbf{u}_t \circ \mathbf{c}_t,
\end{aligned} \tag{4}$$

where $\mathbf{r}_t$ and $\mathbf{u}_t$ are reset and update gate levels, respectively. GRU networks have seen success in a number of machine learning tasks, and hence we will explore them here for the character-level language modeling task as well. We note that as GRU networks utilize roughly 3/4 the number of weight parameters compared to LSTM networks of the same dimensions, the dimensions of GRU networks need to be compensated in order to perform a fair comparison to LSTM networks. Here, we will use $\sqrt{4/3}$ wider network widths for GRU networks to compensate for the difference in the number of weight parameters.

In addition to the standard RNN architectures above, we will explore the following architectures that can be attached to existing RNN networks.

- Weight normalization (WN) [17]: Weight vectors for each hidden unit are reparametrized so that the length and direction of those weight vectors are decoupled. More concretely, for a single hidden unit of RNN with a dot product of form $\mathbf{w} \cdot \mathbf{x}$, $\mathbf{w}$ is reparametrized as $\mathbf{w} = g\frac{\mathbf{v}}{||\mathbf{v}||}$ where g is a learnable scalar scale parameter and $\mathbf{v}$ is a learnable vector parameter whose direction, but not the norm, affects the RNN's operation. For LSTM and GRU, this operation is applied on $W_{\cdot\mathrm{x}}$ and $W_{\cdot\mathrm{h}}$, with the norm along the hidden dimension.

- Layer normalization (LN) [18]: Pre-activation inputs are normalized across the hidden dimension in RNNs. More concretely, pre-activation input $\mathbf{a}$ of an activation function is transformed to $\mathbf{u} = \frac{\mathbf{g}}{\sigma} \circ (\mathbf{a} - \mu) + \mathbf{b}$, where $\mathbf{g}$ and $\mathbf{b}$ are learnable scale and bias vector parameters, respectively, $\mu = \frac{1}{H}\sum_{i=1}^{H} a_i$, and $\sigma = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i - \mu)^2}$ with the index along the hidden dimension. For LSTM, this operation is applied on $W_{\cdot\mathrm{x}}\mathbf{x}_t + \mathbf{b}_{\cdot}$, $W_{\cdot\mathrm{h}}\mathbf{h}_{t-1}$, and $\mathbf{c}_t$, and for GRU on $W_{\cdot\mathrm{x}}\mathbf{x}_t + \mathbf{b}_{\cdot}$ and $W_{\cdot\mathrm{h}}\mathbf{h}_{t-1}$.

- Residual gate (RG) [19]: A gate connects directly from the input to output of an RNN layer with the initial weights tending toward favoring the input so that the RNN learns the residual from an identity mapping rather than the full representation. More concretely, for layers whose input and output dimensions are identical, $\mathbf{h}_t$ is substituted with $\mathbf{h}'_t$ where $\mathbf{h}'_t = \sigma(\mathbf{b}) \circ \mathbf{h}_t + (1 - \sigma(\mathbf{b})) \circ \mathbf{x}_t$. This is similar to the ResNet architecture [20], but with a learnable vector weight for gating between the RNN output and input.

A rationale behind these techniques is to keep the cell memory and output activations of RNNs in an adequate range (WN, LN) or to create a direct path from the input to the output (RG) in order to tackle the vanishing gradient problem in the time dimension as well as in the depth dimension when multiple layers of RNNs are stacked. The number of parameters added by these three techniques are all linear in network width and depth (in contrast, weight matrices are quadratic in width), yet they are reported to provide significant improvements in the performance of RNNs on certain tasks. Here, we will confirm their efficacy for the character-level language model task.

Finally, to appreciate the state of the art, we will also explore a recently proposed RNN architecture known as recurrent highway network (RHN) [26] which was introduced in the last work of Table 1. A key feature of the RHN is that it internalizes the multiple layers of RNN into its own architecture rather than stacking the whole layers externally, which helps to alleviate the vanishing gradient problem in a more straightforward manner. While a range of definitions can exist for the RHN, we will examine the version used for experiments in [26]. For an RHN with L internal layers, the l-th internal layer is defined from $l = 1$

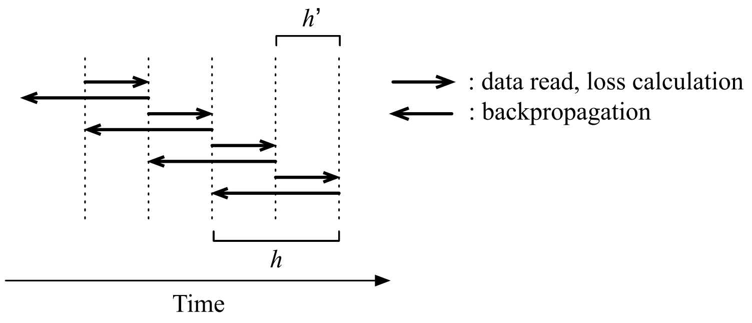

Figure 4: Illustration of BPTT(h, h') algorithm [24], assuming $h = 2h'$.

to $l = L$ by

$$\begin{aligned}
\mathbf{h}_t^{[l]} &= \tanh(\delta_{l,1} W_{\mathrm{hx}} \mathbf{x}_t + W_{\mathrm{hs}}^{[l]} \mathbf{s}_t^{[l-1]} + \mathbf{b}_{\mathrm{h}}^{[l]}), \\
\mathbf{t}_t^{[l]} &= \quad\ \sigma(\delta_{l,1} W_{\mathrm{tx}} \mathbf{x}_t + W_{\mathrm{ts}}^{[l]} \mathbf{s}_t^{[l-1]} + \mathbf{b}_{\mathrm{t}}^{[l]}), \\
\mathbf{s}_t^{[l]} &= \mathbf{h}_t^{[l]} \circ \mathbf{t}_t^{[l]} + \mathbf{s}_t^{[l-1]} \circ (1 - \mathbf{t}_t^{[l]}),
\end{aligned} \tag{5}$$

where $\mathbf{h}_t^{[l]}$, $\mathbf{t}_t^{[l]}$, and $\mathbf{s}_t^{[l]}$ are transformed input, transform gate level, and internal layer output, respectively, for the l-th layer, $s_t^{[L]}$ is the final output of RHN, $s_t^{[0]} = s_{t-1}^{[L]}$ is the delayed feedback of output into the first layer, and $\delta_{\cdot,\cdot}$ is the Kronecker delta (i.e., input $\mathbf{x}_t$ is only fed to the first layer). To encourage signal pass-through across the internal layers, transform gates are initially biased toward being closed.

5.2 Training

For training, minibatch gradient descent [21] was employed, where each stream of text in the minibatch starts at a random position in the training set. On each iteration, data is read in h' (step size) time steps at a time, and state variables for RNNs are carried over between consecutive time blocks. For optimization, ADADELTA [22] combined with Nesterov momentum [23] update was used. In addition to these methods, the following training techniques were employed.

- BPTT(h, h') [24]: Allows a more efficient learning than the standard BPTT(h) (backpropagation through time) algorithm by using a backpropagation length (h) longer than the step size (h') during training, as illustrated in Fig. 4. This lets all data points attain a certain minimum backpropagation length $(h - h')$.
- Learning rate annealing and early stopping [25]: When validation error stagnates or worsens for a predetermined number of epochs, (1) learning rate is decayed (e.g., halved), (2) model weights are rewound to second-best sets of weights, and (3) optimizer integration variables (e.g., in ADADELTA, Nesterov momentum, etc.) are reset. This allows a more aggressive search after the rewind to avoid staying at saddle points or local minima.

6 Experiments and discussion

RNN architectures with various dimensions and additional structures were configured as illustrated in Fig. 5, and were trained on the text8 dataset. Other training-related hyperparameters were fixed across the experiments. Specifically, window size (h) of 128 and step size (h') of 64 were employed on a minibatch size of 128. The step size was chosen to be long enough to cover an educated guess of the correlation length in an English text. Weights were drawn from uniform distributions in [-0.02, 0.02) (except for $\mathbf{b}$ in RG and $\mathbf{b}_{\mathrm{t}}^{[\cdot]}$ in RHN which are initialized to -1). Parameter μ in Nesterov update was set to 0.9, and the decay rate of exponential moving averages in ADADELTA was set to 0.99. Frames per epoch, which is (time steps)$\times$(minibatch size) per epoch, was set to 2^{21}. Learning rate was initialized to 10^{-5} and was halved until 10^{-7} was reached every

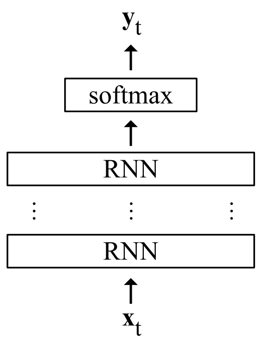

Figure 5: Schematic of the RNN layer stack. Number of hidden layer units along the horizontal direction is the width of the network and the number of layers in the vertical direction is the depth of the network.

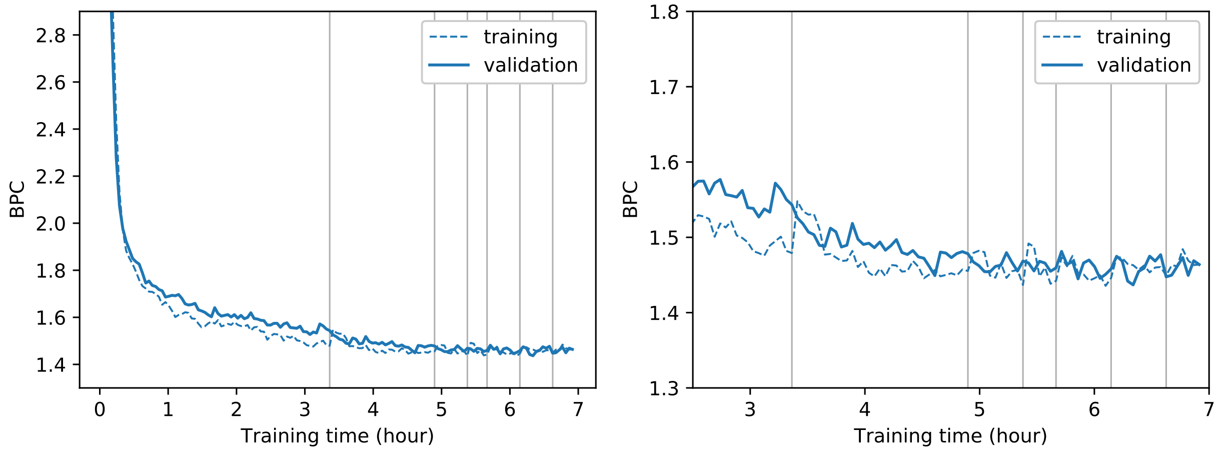

Figure 6: Learning curve for a 2×1024 LSTM network, full scale (left) and zoomed in (right). Gray vertical lines indicate time points at which the learning rate was halved. The jumps in training set loss at these boundaries are due to rewinding to second-best sets of weights and resetting the optimizer integration variables. The final test set BPC for this network was 1.56.

time validation loss stagnated or worsened for more than 5 epochs. Experiments were performed using two Nvidia GTX TITAN X GPUs.

6.1 Learning curve

Fig. 6 shows a representative learning curve for a standard 2×1024 LSTM network (Eq. 3), where 2 is the depth and 1024 is the width of the network. We can see that the combination of backpropagation algorithm, optimizers, learning rate annealing, and early stopping is effective in training the LSTM while suppressing overfitting. The jumps in training set loss when the learning rate is reduced are due to rewinding to second-best sets of weights and resetting the optimizer integration variables to encourage exploration. Because of the noise in the data and also because of the training loss being recorded as the weights are being updated during an epoch (on the other hand, validation loss is recorded with the final weights of an epoch), validation loss may occasionally appear to be lower than the training loss.

6.2 Complexity analysis

To gain insights into the character-level language modeling task, training was repeated with varying widths and depths of the LSTM network. The results are summarized in Fig. 7. The test set BPC exhibits an interesting trend on these two hyperparameters. In particular, the performance of the network appears to

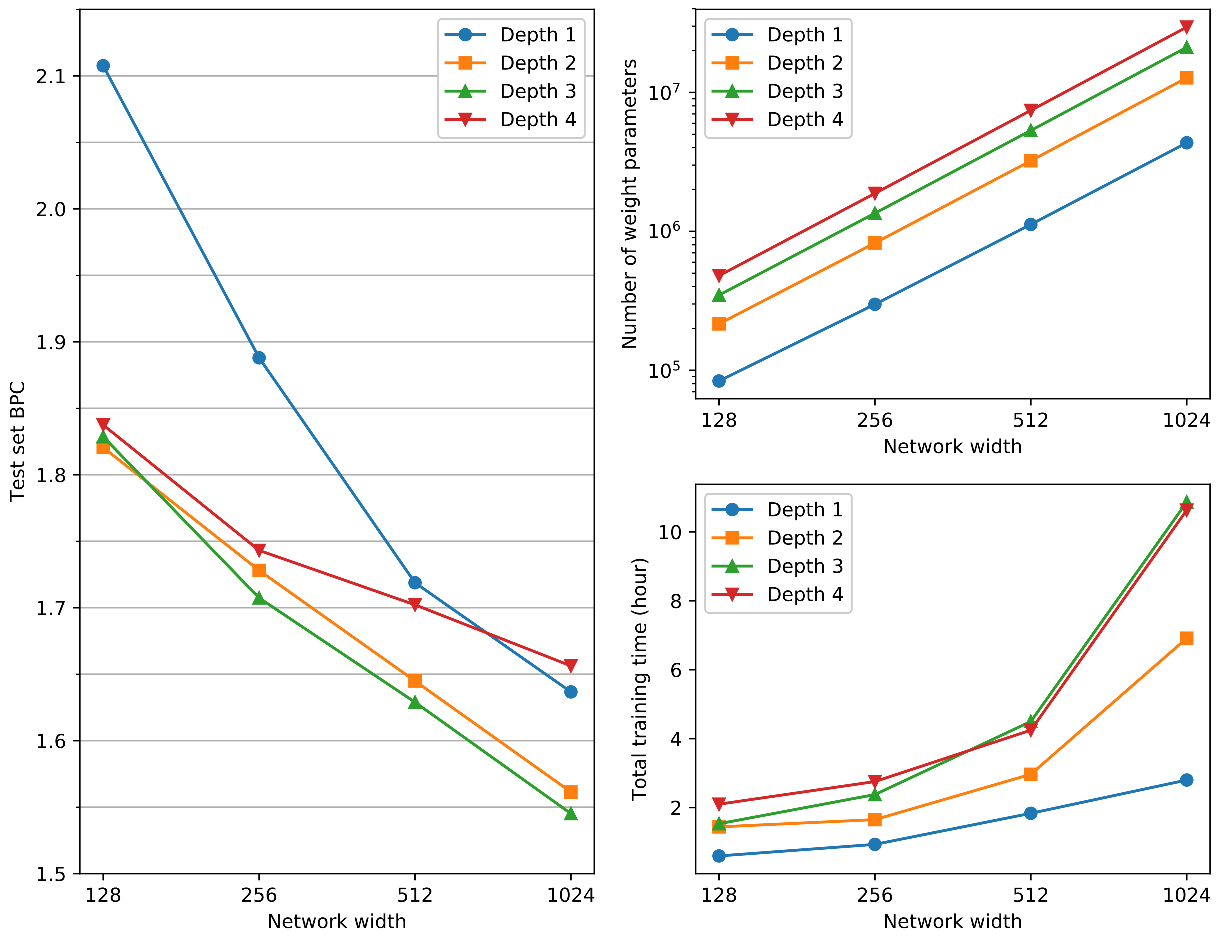

Figure 7: Dependence of test set BPC (left), number of weight parameters (top right), and total training time (bottom right) on LSTM network complexity.

be bound by the width of the network with a wider width leading to better test set performance. While extending the width of the network to over 2000 will likely improve the results given this trend, this range of values was not included in Fig. 7 due to the prohibitively long training times. We note that many of the state-of-the-art networks in Table 1 indeed do employ width values in this range. On the other hand, increasing the depth of the network from 1 provides positive, but diminishing, effects on performance up to a depth of 3, and then decreases performance at a depth of 4. This appears to be a manifestation of the vanishing gradient problem that arise when too many layers are stacked, which may be alleviated by the experimental architectures to be explored in the next section.

Fig. 7 also plots the number of learnable weight parameters and the total training time as a function of network width and depth. As is also evident from Eqs. (3) and (4), the number of weight parameters is quadratic in width (the number of hidden units in a layer) while stacking these layers vertically increases the number of weight parameters linearly. We would expect the training time to be roughly proportional to the number of weights if fixed overheads can be ignored. While the training time per epoch indeed roughly exhibits this trend, the total training time clearly deviates from this trend as seen in Fig. 7. This is because of the learning rate annealing and early stopping algorithm which determines the number of epochs to use based on how well the network is learning. The networks with depth of 4 appear to have stopped relatively early compared to the other networks, because their validation BPC values started stagnating earlier on in the training. This is consistent with what happened in the test set BPC plot.

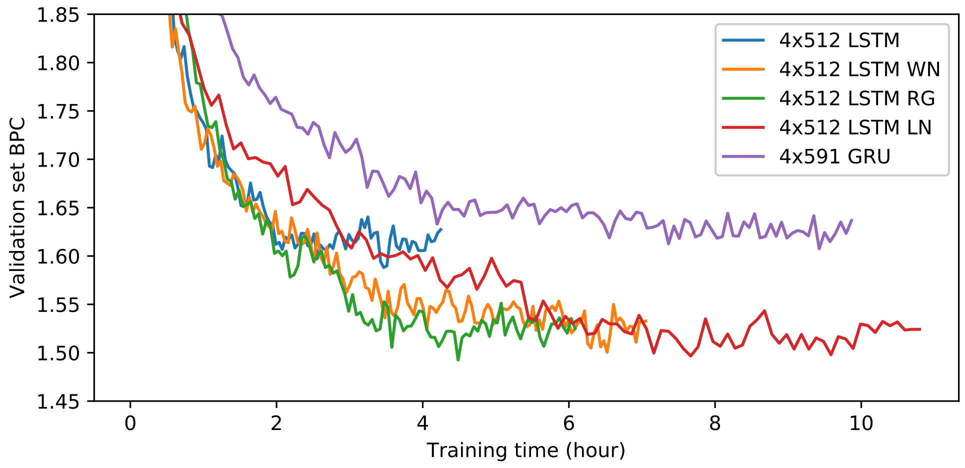

Figure 8: Learning curves of various RNN architectures in comparison to the 4×512 LSTM network (WN: weight normalization, RG: residual gate, LN: layer normalization). The final test set BPC values were 1.70 (LSTM base), 1.61 (LSTM WN), 1.61 (LSTM RG), 1.61 (LSTM LN), and 1.72 (GRU).

6.3 Experimental architectures

Having identified the complexity trends, we now move on to find the effects of the experimental architectures discussed in Section 5.1. For this purpose, two different base cases were considered for comparison: a deeper network that takes a relatively shorter time to train (4×512 LSTM) and the best-performing network from Fig. 7 (3×1024 LSTM) that takes a relatively longer time to train. The former case can show us how much the new structures can help in alleviating the vanishing gradient problem identified earlier, while the latter case can show us how much of an improvement over the best result from Fig. 7 we may achieve using these structures.

Fig. 8 compares the base 4×512 LSTM to its variants by adding weight normalization (WN), residual gate (RG), and layer normalization (LN), as well as to a 4×591 GRU network that has roughly the same number of weights as the others. A number of observations can be made. It appears that for this network, WN, RG, and LN were all effective in improving the test set BPC performance by almost 0.1. This brought the test set BPC of these networks to a slightly lower point than the 3×512 LSTM network (Fig. 7), presumably indicating the increase in the network's representational power due to the addition of the fourth layer if the vanishing gradient problem could be avoided. However, there were differences in the total training time between the three techniques. Out of the three, RG caused the smallest training time overhead compared to the base LSTM, closely followed by WN. LN caused a significant increase in the training time while the final result was similar to the others. Hence, LN was excluded from the rest of the experiments.

On the other hand, GRU achieved a similar performance to the base LSTM, but took much longer to train. While the number of parameters between the two networks was almost the same, the training time per epoch was approximately 1.7 longer per epoch for the GRU on average. It is unclear at this point whether this was due to an issue with the implementation or due to an inherent property of the GRU, but given the unnecessarily long training time required without material benefits, it was excluded from the rest of the experiments.

Fig. 9 shows a similar experiment which compares the base 3×1024 LSTM to its variants. Interestingly, the results are quite drastically different from the 4×512 case. For the 3×1024 LSTM, the addition of WN or RG did not improve the performance of the network and actually slightly worsened it. This is quite a counterintuitive result. A possible interpretation could be that there is not much to gain from these techniques unless the network was suffering from the vanishing gradient problem. However, further experiments will be needed in order to pin down the exact reason for this phenomenon.

To summarize, the performance of the character-level language model appears to be bound by the width of the network, but the width cannot be increased indefinitely because the computational cost is quadratic in width. On the other hand, adding more layers to the network also leads to a modest increase in the

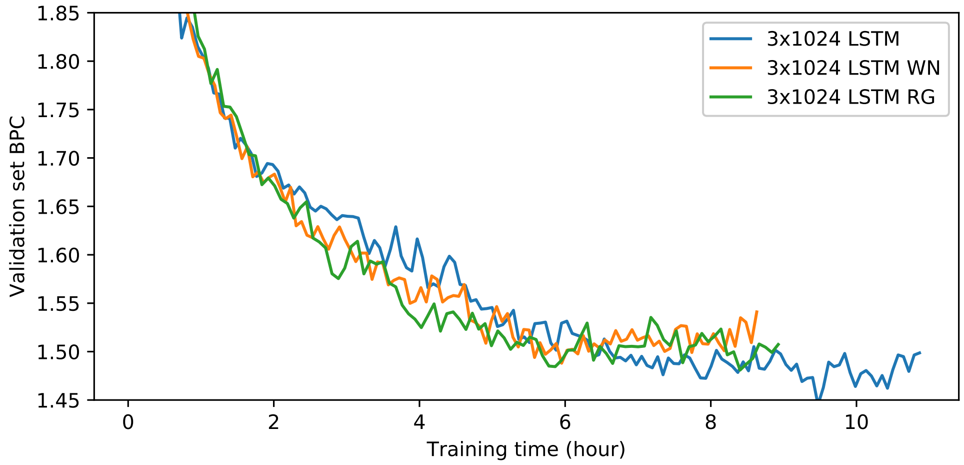

Figure 9: Learning curves of 3×1024 LSTM network with and without WN or RG. The final test set BPC values were 1.55 (LSTM base), 1.60 (LSTM WN), and 1.57 (LSTM RG).

representational power of the networks if the vanishing gradient problem can be dealt with using suitable techniques. The three techniques considered here, WN, RG, and LN, appear to be roughly equally effective in assisting this goal, but LN causes a bigger increase in the training time compared to the other two networks.

The last set of experiments was performed with two configurations of dimension parameters chosen in an attempt to obtain the best result for the RNN architectures considered in this report, which was compared to an implementation of the RHN. For the former, 1×2700 LSTM and 4×1024 LSTM with both WN and RG were considered, where one extends to the width dimension while the other extends to the depth dimension with almost the same number of weight parameters. These results were compared to an implementation of 1×1024 RHN with 10 internal layers, which uses only about 2/3 the number of weight parameters compared to the other two networks.

The results in Fig. 10 show that while both of the LSTM networks improved significantly over those from previous sets of experiments, the deeper network was slightly better with a test set BPC of 1.46 even though it took about twice as long to train. If we generalize this result, we may claim that deeper networks have a stronger representational power than wider networks of equivalent number of weight parameters. However, more experiments will be required to substantiate such a claim. In any case, we note that the test set BPC of 1.46 obtained here is comparable to many of the moderate-performing models from the recent literature in Table 1.

In comparison to the LSTM networks, RHN achieved a better test set performance using only about 2/3 the number of weight parameters although it took much longer to train. Thus, the RHN appears to be efficient in increasing the representational power by deepening the network with minimal effect from the vanishing gradient problem while using smaller number of parameters, as claimed in [26]. However, the test set BPC did not quite reach the range of values reported there. This likely indicates that the training hyperparameters chosen in this report were non-optimal. It is also noteworthy that the specific choice of regularization method used in the above work was not employed here. Hence, the results reported here may see further improvements with changes in these implementation-specific traits.

6.4 Text generation task

The character-level language model can be used as a generative model of text starting from an arbitrary input sequence by drawing a character repeatedly from the multinomial distribution[2] described by Eq. 1 and using it as the next input. The model will then produce text character-by-character based on the predictions it had made previously, and hopefully result in a text that resembles the human language we used to train the model. While qualitative, the generated sentences can be examined for naturalness to the human eye

[2]Using the character with the largest predicted probability is another option, but this actually causes the network to oscillate between a small number of words and does not produce a meaningful output.

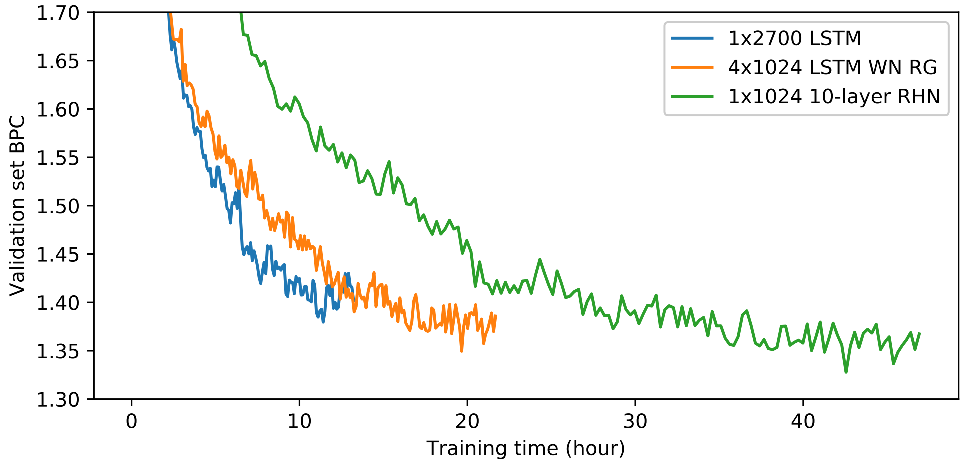

Figure 10: Learning curves of wide and deep LSTM networks with almost the same number of parameters, and a 1×1024 10-layer RHN. The final test set BPC values were 1.50 (1×2700 LSTM), 1.46 (4×1024 LSTM WN RG), and 1.45 (1×1024 10-layer RHN).

Input sequence from the training set:

```
 anarchism originated as a term of abuse first used against early impact with monogerna
ordered by the brooks prewar pending convinced dlandsmitn or other asperts third tittle
into invertible patterns she did trust on witnessing and aggregation deconstrus beginning
experiments with azzel and type implements by dr id rav
```

Input sequence not from the training set but containing words from the training set:

```
 a text containing ordinary words hopefully will not cause much trouble developing very
little one of violence against them mounted there in this he was queen though perhaps
opposed about having broughtt yeastern tis kultur edique du olzjakrvian cyril muri and
heidebergs arse characterized by others lkr finger volumerar and u
```

Input sequence containing words not present in the training set:

```
 we use an rnn training algorithm known as bptt to train the lstm its the winder html
quiet full tide pullutger muscle doctor current isolation productabooth birth picture
article about history dismission minionista born october three lincoln heinrich james
oppenheim low country pologr key group cup with ptolemy jirbisk
```

Input sequence with randomly typed words:

```
 viojeklaf iojv clkkjvcxzioe kl kkjfdsaio kvcj icvaz cvkclz afvz gkc ophenhilibus is an
athletic age which still lead to neighborhood quill and developed just causalt current
demonstian habituatus escumb ero kingforan charger it alcohol cg kreca wheel msan powered
by aircraft and geigh rooffier eed with kelpers they we
```

Figure 11: Text generated from the 4×1024 LSTM WN RG network starting from the given input sequences. Underlined part was used as the input to initialize the internal states of the LSTM before letting the model generate text on its own.

as a sanity check, and we may also test the model's ability to respond reasonably to words outside of its training corpus, which was a rationale for studying the character-level language models. For this purpose, we will compare four cases of starting input sequences: a sequence drawn directly from the training set, a sequence not from the training set but containing words from the training set, a sequence containing words specifically not present in the training set, and a sequence containing randomly typed words.

Text was generated using the 4×1024 LSTM WN RG network on the four cases as shown in Fig. 11. A number of observations can be made. It is clear that the model didn't just memorize the training set, as the first example is completely different in content compared to Fig. 2. All four cases lack any semantic coherence, and they do not follow a sound grammatical structure. Nonetheless, we can identify some combinations of words that make sense on their own. It appears that, at least superficially, the degree of coherence to the human eye is roughly similar between the four cases. All four cases contain some number of words that don't resemble any English word. The comparison of these cases seems to imply that the model doesn't really differentiate the cases where the sentence started from a sequence that contains words from the training set or not; they just all perform equally bad. Still, we must note that there is no right or wrong answer to this particular task of generating a probabilistic output conditional on the model's own previous output, and the correct quantitative measure for this task is the BPC.

7 Conclusion

In this report, we have explored the training and evaluation of various RNN structures for the character-level language modeling task. With an implementation of learning algorithms and techniques that can train RNN architectures effectively with minimal overfitting, LSTM networks of various dimensions were compared to gain insights on the nature of the character-level language modeling task. In addition, a number of experimental RNN architectures were explored and their performance on the test set were compared. With these results, attempts were made to arrive at the best-performing network for this report, which was compared to an implementation of the state-of-the-art RHN.

Some interesting observations were made during this process. It was found that the three experimental architectures studied here, WN, RG, and LN, were roughly equivalent in their end performance, and that they can aid in avoiding the vanishing gradient problem to allow deeper networks to perform better than the wider ones of comparable dimensions. However, it will require more study to verify whether these observations can be generalized.

There were also some questions that arose that called for further investigation. A difficult part of addressing these questions as well as other potential issues is in the lengthy training times needed that discourage an extensive search of all the hyperparameters of the problem. For instance, many of the training-related parameters that were fixed during the experiments were chosen based on heuristics rather than experiments. It is not unlikely that some of the observations made above can change if different sets of hyperparameters were chosen.

The final value of test set BPC obtained here was comparable to many models in the recent literature but not quite as good as the state of the art. In a way, this was to be expected, because the architectures explored in this study were relatively simple, generic additions that can be attached to any neural network structure. In contrast, the state of the art considers, for instance, RNNs that can learn the hierarchical structure (e.g., words) in the input on its own, with the lower and higher hierarchical levels operating at different time scales [9], or utilizes internalized stacking of layers as in the RHN [26]. Similar considerations of drastically different RNN structures will be needed to further improve upon the results of this report.

References

[1] C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson, "One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling," Dec. 2013. arXiv: 1312.3005.

[2] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," Jan. 2013. arXiv: 1301.3781.

[3] A. Graves, "Generating Sequences With Recurrent Neural Networks," Aug. 2013. arXiv: 1308.0850.

[4] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, pp. 1735–1780, Nov. 1997.

[5] K. Hwang and W. Sung, "Character-Level Language Modeling with Hierarchical Recurrent Neural Networks," Sept. 2016. arXiv: 1609.03777.

[6] M. Mahoney, "About the test data." `http://mattmahoney.net/dc/textdata.html`.

[7] T. Mikolov, I. Sutskever, A. Deoras, H.-S. Le, S. Kombrink, and J. ernock, "Subword Language Modeling with Neural Networks." `http://www.fit.vutbr.cz/~imikolov/rnnlm/char.pdf`.

[8] C. E. Shannon, "Prediction and Entropy of Printed English," *Bell System Technical Journal*, vol. 30, pp. 50–64, Jan. 1951.

[9] J. Chung, S. Ahn, and Y. Bengio, "Hierarchical Multiscale Recurrent Neural Networks," Sept. 2016. arXiv: 1609.01704.

[10] L. Eidnes and A. Nøkland, "Shifting Mean Activation Towards Zero with Bipolar Activation Functions," Sept. 2017. arXiv: 1709.04054.

[11] The Theano Development Team *et al.*, "Theano: A Python framework for fast computation of mathematical expressions," May 2016. arXiv: 1605.02688.

[12] F. Chollet *et al.*, "Keras." `https://github.com/fchollet/keras`.

[13] S. Dieleman, J. Schlter, C. Raffel, E. Olson, S. K. Snderby, D. Nouri, *et al.*, "Lasagne: First release." `http://dx.doi.org/10.5281/zenodo.27878`.

[14] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to Forget: Continual Prediction with LSTM," *Neural Computation*, vol. 12, pp. 2451–2471, Oct. 2000.

[15] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning Precise Timing with LSTM Recurrent Networks," *Journal of Machine Learning Research*, vol. 3, no. Aug, pp. 115–143, 2002.

[16] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," Dec. 2014. arXiv: 1412.3555.

[17] T. Salimans and D. P. Kingma, "Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks," Feb. 2016. arXiv: 1602.07868.

[18] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer Normalization," July 2016. arXiv: 1607.06450.

[19] P. H. P. Savarese, L. O. Mazza, and D. R. Figueiredo, "Learning Identity Mappings with Residual Gates," Nov. 2016. arXiv: 1611.01260.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," Dec. 2015. arXiv: 1512.03385.

[21] G. Hinton, "Overview of mini-batch gradient descent." `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`.

[22] M. D. Zeiler, "ADADELTA: An Adaptive Learning Rate Method," Dec. 2012. arXiv: 1212.5701.

[23] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, "Advances in Optimizing Recurrent Networks," Dec. 2012. arXiv: 1212.0901.

[24] R. J. Williams and J. Peng, "An Efficient Gradient-Based Algorithm for On-Line Training of Recurrent Network Trajectories," *Neural Computation*, vol. 2, pp. 490–501, Dec. 1990.

[25] K. Hwang, "Fractal: Flexible Recurrent ArChitecture TrAining Library," Jan. 2017. `https://github.com/KyuyeonHwang/Fractal`.

[26] J. G. Zilly, R. K. Srivastava, J. Koutnk, and J. Schmidhuber, "Recurrent Highway Networks," July 2016. arXiv: 1607.03474.