TOWARDS BINARY-VALUED GATES FOR ROBUST LSTM TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Long Short-Term Memory (LSTM) is one of the most widely used recurrent structures in sequence modeling. Its goal is to use gates to control the information flow (e.g., whether to skip some information or transformation or not) in the recurrent computations, although its practical implementation based on soft gates only partially achieves this goal and is easy to overfit. In this paper, we propose a new way for LSTM training, which pushes the values of the gates towards 0 and 1. By doing so, we can (1) better control the information flow: the gates are mostly open or closed, instead of in a middle state; and (2) avoid overfitting to certain extent: the gates operate at their flat regions, which is shown to correspond to better generalization ability. However, learning towards discrete values of the gates is generally difficult. To tackle this challenge, we leverage the recently developed Gumbel-Softmax trick from the field of variational methods, and make the model trainable with standard backpropagation. Experimental results on language modelling and machine translation show that (1) the values of the gates generated by our method are more reasonable and intuitively interpretable, and (2) our proposed method generalizes better and achieves better accuracy on test sets of in all tasks. Moreover, the learnt models are robust to low-precision approximation and low-rank approximation of the gate parameters due to its flat loss surface.

1 INTRODUCTION

Recurrent neural networks (RNN) (Hochreiter, 1998) are widely used in sequence modeling tasks, such as language modeling (Kim et al., 2016; Jozefowicz et al., 2016), speech recognition (Zhang et al., 2016), time series prediction (Xingjian et al., 2015), machine translation (Wu et al., 2016; Britz et al., 2017), image captioning (Vinyals et al., 2015; Xu et al., 2015), and image generation (Villegas et al., 2017).

To address the long-term dependency and gradient vanishing problem of conventional RNN, long short-term memory (LSTM) (Gers et al., 1999; Hochreiter & Schmidhuber, 1997b) was proposed, which introduces *gates* to control the information in a recurrent unit: a *forget gate* to determine how much previous information should be excluded for the current step, an *input gate* to find relevant signals to absorb into the hidden context, and an *output gate* for prediction and decision making. For ease of optimization, in practical implementation, one usually uses element-wise sigmoid functions to mimic the gates, whose outputs are soft values between 0 and 1. By using such gates, LSTM usually performs much better than conventional RNN. However, the benefits come with the cost of introducing many more parameters in the gates, which makes the training of a LSTM model inefficient and easy to overfit(Krueger et al., 2016; Zaremba et al., 2014; Semeniuta et al., 2016).

In this paper, we explore a new way to train LSTM by pushing the values of the gates to the boundary of their ranges $(0, 1)$ [1]. Pushing the values of the gates to 0/1 has certain advantages. First, it well aligns with the original purpose of the development of gates: to get in or skip the information by "opening" or "closing" the gates during the recurrent computation. Second, training LSTM towards

[1] In practice, the output of a gate function is usually a vector, of which different dimensions correspond to different semantic aspects. For simplicity, in the paper, we usually say "pushing the output of the gate function to 0/1" when meaning "pushing each dimension of the output vector of the gate function to either 0 or 1". We sometimes also call each dimension of the output vector of the gate function as a gate, and say a gate is open/closed if its value is close to 1/0.

binary-valued gates can make the learned model generalize better. According to (Hochreiter & Schmidhuber, 1997a; Haussler et al., 1997; Keskar et al., 2016; Chaudhari et al., 2016), a model lying in a flat region of the loss surface is likely to generalize well, since any small perturbation to the model makes little fluctuation to the loss. Training LSTM towards binary-valued gates means seeking those parameters whose corresponding values of the gates is closely to zero or one, namely residing in the flat region of the sigmoid function. Simple deductions show that this also corresponds to the flat region of the overall loss surface.

Technically, pushing the outputs of the gates towards such discrete values is challenging. A straightforward approach is to sharpen the sigmoid function by a smaller temperature. However, this is equivalent to rescaling the input and cannot guarantee the value of the learned gate function to be 0 or 1. Another possible approach is reinforcement learning, in which the values of the gates in each recurrent step are viewed as *actions* $a \in \{0, 1\}^d$ (where d is the dimension of the hidden state of LSTM), and the loss is viewed as the cost in the terminal state. However, by doing so, one has to optimize the model with exponentially large number of actions which is usually hard and sample-inefficient.

To tackle this challenge, in this paper, we leverage the Gumbel-Softmax trick Jang et al. (2016); Maddison et al. (2016) recently developed for variantional methods. The trick aims to generate approximated samples for categorical latent variables in a stochastic computational graph, e.g., variational autoencoder, and brings convenience to using reparametrization tricks thus leads to efficient learning. Specifically, during training, we apply the Gumbel-Softmax trick to the gates to approximate the values sampled from the Bernoulli distribution given by the parameters, and train the LSTM model with standard backpropagation methods. We call this method Gumbel-Gate LSTM (G^2-LSTM). We did three experiments on two real-world tasks (i.e., language modelling and machine translation) to verify our proposed method. According to the experimental results, we have the following observations:

- Our model generalizes well: In all tasks, we achieved superior performance to baseline algorithms on the test sets, and the gap between training and test was effectively reduced.
- Our model is very robust due to its flat loss surface: We apply several model compression algorithms to the parameters in the gates, including low-precision approximation and low-rank approximation, and all results show that the models we learnt are better under different compression methods.
- The gates we learned are meaningful and intuitively interpretable after visualization. Furthermore, our model can automatically learn the hierarchical structure of sentences.

The organization of the paper is as follows. We introduce the related work in section 2 and propose our learning algorithm in section 3. Experiments are provided in section 4 and the future work is discussed in the last section.

2 Related Work

2.1 Relationship between loss surface and generalization

The concept of sharp and flat minimizers has been first discussed in Hochreiter & Schmidhuber (1997a); Haussler et al. (1997) . Intuitively, a flat minimizer x corresponds to the point for which the function varies slowly in a relatively large neighborhood of x. In contrast, a sharp minimizer x is such that the function increases rapidly in a small neighborhood of x. The large sensitivity of the training function at a sharp minimizer negatively impacts the ability of the trained model to generalize on new data. Recently, several papers discuss how to lead the training process to learn a model into the flat region thus obtain better generalization ability. In Keskar et al. (2016), the authors show by using batches with small batch size during training, the learnt model is more likely to converge to a flat region rather than a sharp one. In Chaudhari et al. (2016), the authors proposed a new objective function considering the local second-order information and push the model to be optimized towards a wide valley.

2.2 DROPOUT

Dropout is one of the most standard tricks used in deep learning to improve generalization ability. For recurrent neural networks, initially successful applications of dropout in RNNs (Zaremba et al., 2014) applied dropout to feed-forward connections. Several recent works (Semeniuta et al., 2016; Gal & Ghahramani, 2016; Krueger et al., 2016) propose methods that are not limited in this way. Semeniuta et al. (2016) apply recurrent dropout to the updates to LSTM memory cells. In Zoneout (Krueger et al., 2016), the values of the hidden states and memory cells randomly either maintain their previous value or are updated as usual. This introduces stochastic identity connections between subsequent time steps.

The main difference between dropout and our proposed method is that we never drop gradient and computations. Our benefit to generalization is coming from biasing the optimization process to a flatter region to avoid overfitting, but not to regularize the training of a deep neural network by dropping the gradient to prevent co-adaptations. Therefore, both methods can be jointly used to obtain better generalization ability.

3 THE ALGORITHM

In this section, we present our training algorithm for LSTM by learning towards binary-valued gates. In the first part, we introduce the technical background. After that, we propose our main algorithm.

3.1 BACKGROUND

Recurrent neural networks process data $x_1, x_2, \ldots, x_T$ sequentially and construct a corresponding sequence of representations $h_1, h_2, \ldots, h_T$. For a single-layer recurrent neural network, the hidden states $h_1, h_2, \ldots, h_T$ learnt could be used for prediction or decision makings. For deep (stacked) recurrent neural networks, the hidden states in layer k could be propagated as inputs to layer $k+1$.

In recurrent neural networks, each hidden state is trained (implicitly) to remember and emphasize all task-relevant aspects of the preceding inputs, and to incorporate new inputs via a transition operator, T, which converts the present hidden state and input into a new hidden state $h_t = T(h_{t-1}, x_t)$. Usually the function T is a neural network, e.g., $h_t = \tanh(W_h h_{t-1} + W_x x_t + b)$, where W_h, W_x and b are parameters.

Long short-term memory RNN (LSTM) (Hochreiter & Schmidhuber, 1997b) is a carefully designed recurrent structure. The hidden state of LSTM is divided into memory cell c_t, intended for internal long-term storage, and hidden state h_t, used as a transient representation of state at timestep t.

In the most widely used formulation of an LSTM, c_t and h_t are computed via a set of three gates. The forget gate, f_t, directly connects c_t to the memories of the previous timestep c_{t-1} via an element-wise multiplication. Large values of the forget gate cause the cell to remember most (not all) of its previous value. The other gates control the flow of information in input (i_t) and output (o_t) of the cell. Each gate has a weight matrix and a bias vector; we use subscripts f, i and o to denote parameters for forget gate, input gate and output gate respectively, e.g., the parameter for the forget gate is denoted by W_{xf}, W_{hf}, and b_f.

An LSTM is defined as follows:

$$
\begin{aligned}
i_t &= \sigma(W_{xi} x_t + W_{hi} h_{t-1} + b_i) && (1)\\
f_t &= \sigma(W_{xf} x_t + W_{hf} h_{t-1} + b_f) && (2)\\
o_t &= \sigma(W_{xo} x_t + W_{ho} h_{t-1} + b_o) && (3)\\
g_t &= \tanh(W_{xg} x_t + W_{hg} h_{t-1} + b_g) && (4)\\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t && (5)\\
h_t &= o_t \odot \tanh(c_t), && (6)
\end{aligned}
$$

where $\sigma(\cdot)$ represents the sigmoid function and $\odot$ is the element-wise product.

3.2 TRAINING LSTM GATES TOWARDS BINARY VALUES

The LSTM unit requires much more number of parameters than simple RNN unit, which makes it hard to generalize. As a clear comparison with vanilla RNN, a large percentage of the parameters are contributed to the computation of the gate values[2]. If we can regularize the parameters in the gates or lower down the corresponding complexity, we will obtain a model that generalize better. As we usually use sigmoid function in the gates, if we can push the outputs of the gates towards 0 and 1, the loss function with respect to the gate parameters will be flat: if the parameters in the gates perturbs, the change to the output of the gates is much smaller due to the sigmoid operator (see in Appendix B), then the change to the loss is not that much, which leads to better generalization. [3]

However, the task is quite challenging. One straight forward idea is to sharpen sigmoid function by using a smaller temperature. However, this equals to rescale the parameters in the gates which cannot guarantee the outputs to be closed to the boundary after training. Another solution is to consider the 0/1 choice of each gate as discrete actions, define the problem by reinforcement learning and use policy gradient method to optimize the trajectories, however, this way of solving the problem is usually costly and sample-inefficient.

In this work, we leverage the recent developed Gumbel-Softmax trick, which is one of the widely used method to learn discrete random variables in stochastic computational graphs. Due to the space restriction, we leave the discussion and its literature review in Appendix A. We first provide a proposition about the general technique to approximate Bernoulli distribution and its error bound. Then we will discuss why and how to use it in our problem.

Proposition 1. *Assume $\sigma(\cdot)$ is the sigmoid function. Given $\alpha \in \mathbb{R}$ and $\tau > 0$, we define random variable $D_\alpha \sim B(\sigma(\alpha))$ where $B(\sigma(\alpha))$ is the Bernoulli distribution with parameter $\sigma(\alpha)$ and $G_{\alpha,\tau} = \sigma(\frac{\alpha+\log U-\log(1-U)}{\tau})$ where $U \sim$ Uniform$(0,1)$. Then the following inequalities hold for arbitrary $\epsilon \in (0, \frac{1}{2})$,*

$$P(D_\alpha = 1) - \frac{\tau}{4}\log(\frac{1}{\epsilon}) \leq \quad P(G_{\alpha,\tau} \geq 1-\epsilon) \quad \leq P(D_\alpha = 1), \tag{7}$$

$$P(D_\alpha = 0) - \frac{\tau}{4}\log(\frac{1}{\epsilon}) \leq \qquad P(G_{\alpha,\tau} \leq \epsilon) \qquad \leq P(D_\alpha = 0). \tag{8}$$

Proof. Since $\sigma^{-1}(x) = \log(\frac{x}{1-x})$, we have $P(G_{\alpha,\tau} \geq 1-\epsilon) = P(\frac{\alpha+\log(U)-\log(1-U)}{\tau} \geq \log(\frac{1}{\epsilon} - 1)) = P(e^{\alpha-\tau\log(\frac{1}{\epsilon}-1)} \geq \frac{1-U}{U}) = P(U \geq \frac{1}{1+e^{\alpha-\tau\log(\frac{1}{\epsilon}-1)}}) = \sigma(\alpha - \tau\log(\frac{1}{\epsilon}-1))$. Considering that sigmoid function is monotonically increasing and $\frac{1}{4}$-Lipschitz continuous, we have $P(D_\alpha = 1) - P(G_{\alpha,\tau} \geq 1-\epsilon) = \sigma(\alpha) - \sigma(\alpha - \tau\log(\frac{1}{\epsilon}-1)) \geq 0$ and $P(D_\alpha = 1) - P(G_{\alpha,\tau} \geq 1-\epsilon) = \sigma(\alpha) - \sigma(\alpha - \tau\log(\frac{1}{\epsilon}-1)) \leq \frac{\tau}{4}\log(\frac{1}{\epsilon}-1) \leq \frac{\tau}{4}\log(\frac{1}{\epsilon})$. We omit the proof for (8) as it is almost identical to the proof of (7). □

We can see from the above proposition, the distribution of $G_{\alpha,\tau}$ can be considered as an approximation of Bernoulli distribution $B(\sigma(\alpha))$. The rate of convergence is characterized by (7) and (8). When τ approaches positive zero, we directly obtain the following property which is also proved by Maddison et al. (2016),

$$P(\lim_{\tau\to 0^+} G_{\alpha,\tau} = 1) = P(D_\alpha = 1), P(\lim_{\tau\to 0^+} G_{\alpha,\tau} = 0) = P(D_\alpha = 0). \tag{9}$$

We can also see from the proposition that the noise term U is independent of α when we compute G. As α is usually computed by parameters that needs to be optimized, this technique makes the standard backpropagation work and is generally referred as reparameterization trick.

We use such trick to train the parameters in the gates. Imagine a specific gate with a specific training example and assume it prefers *zero* much more than *one*. By repeatedly sampling from G, the model

[2] In practice, either for language modelling or machine translation, the number of parameters in the gates is usually around 10^8 or even larger.

[3] The benefit on generalization can be viewed from another perspective: If we can ideally push the outputs of the gates to binary value, e.g., by changing the gate function from sigmoid operator to sign operator, we actually lower down the representation capacity of the model thus the new architecture deserves better generalization ability.

will produce a larger loss when the sampled gate value is closed to one, and a smaller loss when the sampled gate value is closed to zero. During the training, it will push its parameters to output values closed to zero by gradient decent thus we can learn the binary-valued gates.

As the gate function is vector-valued function, we extend the notation into a general form: Given $\alpha \in \mathbb{R}^d$ and $\tau > 0$, we define $G_{\alpha,\tau} = \sigma(\frac{\alpha+\log U-\log(1-U)}{\tau})$, where each element u_i in U is independently sampled from Uniform$(0,1)$, $i = 1, 2, \ldots, d$.

In this paper, we push the input gate and forget gate to binary values, but didn't apply the strategy to the output gate due to two reasons: first, the output gate is usually used for the prediction or decision making, it is not that reasonable to let the gate skip information to make decisions. Second, the experimental result dropped quite a lot if we jointly train the three gate together using the proposed method. We call our proposed learning method Gumbel-Gate LSTM (G^2-LSTM), which is as follows:

$$i_t = G(W_{xi}x_t + W_{hi}h_{t-1} + b_i, \tau) \quad (10)$$
$$f_t = G(W_{xf}x_t + W_{hf}h_{t-1} + b_f, \tau) \quad (11)$$
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \quad (12)$$
$$g_t = \tanh(W_{xg}x_t + W_{hg}h_{t-1} + b_g) \quad (13)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \quad (14)$$
$$h_t = o_t \odot \tanh(c_t). \quad (15)$$

The temperature is a hyperparameter which characters the strength to push the gate function to be binarized. The smaller the temperature is, the stronger it is to push the outputs to binary values. During training, in the forward pass, we use (Eqn (10) (15)) in the LSTM units and calculate the loss, e.g., using log likelihood function. In the backward pass, as the loss to the parameters is continuous and fully differentiable, we can use any gradient-based method to train the model parameters.

4 Experiments

4.1 Experimental Design

We conduct three experiments of two tasks – language modelling and machine translation – to verify our proposed method.

4.1.1 Language Modelling

Language modelling is one of the most typical tasks using LSTM. In this paper, we use The Penn Treebank language model corpus which contains about 1 million words. The model is trained to predict the next word and evaluated according to perplexity.

We did word-level language modelling based on the open source code Zaremba et al. (2014), which achieves the best single-model performance. We follow to use a stacked two-layer LSTM with 1500 units, with weights initialized uniformly in $[-0.04, +0.04]$. The model is trained for 14 epochs with learning rate 1.0 using standard stochastic gradient decent, after which the learning rate is reduced by a factor of 1.15 at the beginning of each epoch. Gradient norms are clipped at 10.0. For G^2-LSTM, we found the temperature is not very sensitive. We set the temperature to 1.0 and didnt change during training.

4.1.2 Machine Translation

For the neural machine translation (NMT) experiment, we compare the performances of different models on two tasks: (1) IWSLT2014 German→English translation task (Cettolo et al., 2014), widely adopted in machine learning community (Bahdanau et al., 2016; Wiseman & Rush, 2016; Ranzato et al., 2015). The training/dev/test set respectively contains about $153k/7k/7k$ sentence pairs, with words pre-processed into sub-word units with byte pair encoding (BPE) (Sennrich et al., 2016). We choose $25k$ most frequent sub-word units as vocabulary for both German and English. (2) English→German translation in WMT'14, which is commonly used as a benchmark task to evaluate different NMT models. (Bahdanau et al., 2014; Wu et al., 2016; Gehring et al., 2017). The

Table 1: Performance comparison on language model (perplexity)

Model	Perplexity
Unregularzed LSTM	114.5
NR-dropout (Zaremba et al., 2014)	78.4
Zoneout (Krueger et al., 2016)	77.4
G^2-LSTM	**77.3**

Table 2: Performance comparison on machine translation (BLEU)

English→German task	BLEU	German→English task	BLEU
Existing end-to-end system			
RNNSearch-LV (Jean et al., 2015)	19.40	NMPT (Huang et al.)	28.96
Global-att (Luong et al., 2015)	20.90	NMPT+LM (Huang et al.)	29.16
GNMT (Wu et al., 2016)	**24.61**	ActorCritic (Bahdanau et al., 2016)	28.53
Our end-to-end system			
Baseline	21.89	-	31.00
G^2-LSTM	22.43	-	**31.95**

training data comprises 4.5M English→German sentence pairs. The models are evaluated on the WMT14 test set Newstest 2014, and the concatenation of Newstest 2012 and Newstest2013 is used as development set. Similarly, BPE is used to form a vocabulary of most frequent $30k$ sub-word units for both language. In both tasks, we remove the sentences with more than 64 sub-word units in training.

For German→English task, we adopt a stacked two-layer encoder-decoder framework. We set the size of word embedding and hidden state to 256. As amount of data in the English→German task is much larger, we adopt a stacked three-layer encoder-decoder framework and set the size of word embedding and hidden state to 512 and 1024 respectively. The first layer of the encoder is bi-directional. We also use dropout in training stacked LSTM as in Zaremba et al. (2014), with dropout value determined via dev set performance. For both experiments, we set the temperature used in G^2-LSTM to 0.9. We adapted mini-batch training with batch size to be 32/64 for German→English/English→German tasks respectively. All models were trained with AdaDelta (Zeiler, 2012) on one M40 GPU. Both gradient clipping norm are set to 2.0. We use tokenized case-sensitive BLEU (Papineni et al., 2002)[4] as evaluation measure.

4.2 Experimental Results

For all tasks, we compare our proposed algorithm with several baselines and list them in Table 2. First we compare our training algorithm with the baseline algorithm (which uses conventional gates without Gumbel-Softmax trick during the forward pass) in the same setting. From the results, we can see that our learnt models are better than the baseline models. In language modelling task, we outperform the baseline algorithm for 1.1 points using perplexity. For machine translation, we outperform the baseline for 0.95 and 0.56 points using BLEU score for German→English and English→German tasks respectively. Note that the advantages are got only based on a new training method on LSTM units but no architecture changes. Training-validation curves of the experiments can be found in Appendix C, which also show that our models have lower generalization errors.

Besides this, we also list the performance of previous works in literature with different model architectures or settings. For language modelling, we obtain competitive results compared to Zoneout (Krueger et al., 2016). For German→English task, the two-layer stacked encoder-decoder model we learnt outperforms all previous works. For English→German task, our result is slightly worse than GNMT (Wu et al., 2016) as they used a stacked eight-layer LSTM encoder-decoder model while we only use a three-layer one.

[4]Calculated by the script at `https://github.com/moses-smt/mosesdecoder/blob/master/scripts/generic/multi-bleu.perl`

Table 3: Model compression results on Penn Tree Bank dataset.

	Original	Round	Round & clip	SVD ($rank = 128$)	SVD ($rank = 64$)
LSTM	78.4	78.5(+0.1)	78.9(+0.5)	81.4 (+3.0)	94.7 (+16.3)
G^2-LSTM	**77.3**	**77.3 (+0.0)**	**77.4 (+0.1)**	**78.3 (+1.0)**	**80.4 (+3.1)**

Table 4: Model compression results on IWSLT German→English dataset.

	Original	Round	Round & clip	SVD ($rank = 32$)	SVD ($rank = 16$)
LSTM	31.00	28.65 (-2.35)	21.97 (-9.03)	30.52 (-0.48)	29.56 (-1.44)
G^2-LSTM	**31.95**	**31.44 (-0.51)**	**31.44 (-0.51)**	**31.62 (-0.33)**	**31.28 (-0.67)**

Table 5: Model compression results on WMT English→German dataset.

	Original	Round	Round & clip	SVD ($rank = 32$)	SVD ($rank = 16$)
LSTM	21.89	16.22 (-5.67)	16.03 (-5.86)	21.15 (-0.74)	19.99 (-1.90)
G^2-LSTM	**22.43**	**20.15 (-2.28)**	**20.29 (-2.14)**	**22.16 (-0.27)**	**21.84 (-0.51)**

4.3 Robustness Analysis

We also did a set of experiments to show the learnt language model and NMT models by our proposed method are very robust compared to the baseline models. To be more specific, we use two ways to compress the models and check the performance.

Gate Parameter Compression with Low Precision We compress parameters in the input/forget gate to lower precision. By doing this, the model can be compressed to a relatively small size. In particular, we test our model by round and clip operations.

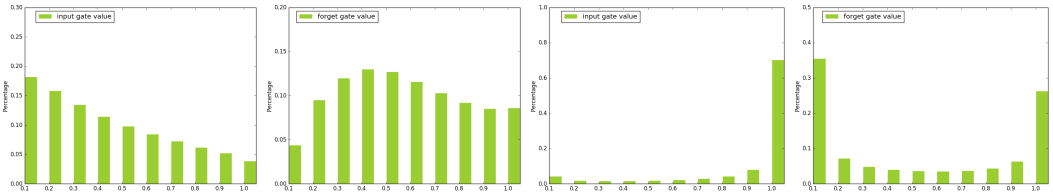
$$\text{round}_r(x) = \text{round}(x/r) * r \tag{16}$$

$$\text{clip}_c(x) = \text{clip}(x, -c, c) \tag{17}$$

We test two settings. In the first setting (*Round*), we round the parameter using (16). In this way, we can reduce the support set of the value of parameters. In the second setting (*Round & Clip*), we further clip the rounded value to a fixed range thus restrict the number of different values to constant. For language model/machine translation task, we select clip value $c = 0.2/1.0$ and round parameter $r = 0.1/0.5$ which performs better. As a result, parameters of input gate and forget gate of each layer in language models and machine translation models can only take five values $(0.0, \pm 0.1, \pm 0.2)$ and $(0.0, \pm 0.5, \pm 1.0)$ respectively.

Gate Parameter Compression with Low Rank We compress parameter matrices in the input/forget gate to lower rank. By doing this, the model can be compressed to a relatively small size and relatively faster in the matrix multiplication. In particular, we test different low rank precision ($rank = 16/32/64/128$) using standard SVD algorithm.

We summarize the results in Table 3, Table 4 and Table 5. More results can be found in Appendix D. All results show that the proposed method is much more robust compared to the baseline. For language modelling task, we can see that the models don't differ much according to compression with low precision. But our model significantly outperforms the baseline with low rank approximation. Even when we set $rank = 64$ (roughly 12x compression rate to the gates), we still get 80.4 perplexity. For machine translation, it can be seen from the table, in all cases, our proposed method is better in any compression with low precision and low rank. Even we set the $rank = 16$ (roughly 8x/32x compression rate to the gates for German→English and English→German tasks respectively), we still roughly comparable to the baseline model with full parameters. All results show the models trained with the proposed method is more robust than the baselines.



(a) Input gate in LSTM (b) Forget gate in LSTM (c) Input gate in G^2-LSTM (d) Forget gate in G^2-LSTM

Figure 1: The value distribution of the input/forget gates in German→English task.

4.4 GATE VISUALIZATION

To well verify the effectiveness of our proposed G^2-LSTM, we did a set of experiments to show the values of gates we have learnt are near the boundary and are reasonable, based on the model learnt from German→English translation task. We show the value distribution of the gates trained using classic LSTM and G^2-LSTM. To achieve this, we sample 10k sentence pairs from training data and feed the data into the model. By doing this, we get the output value vectors of the input/forget gates during encoding and decoding. We record the value of each element in the vector for each word, and get the value distribution in the range of $[0, 1]$, which is shown in Figure 1.

From the figure, we can see that although both LSTM and G^2-LSTM work in practice, the output values of the gates are far different. In LSTM, the distribution of the values in input gate and forget gate are flat. However, in G^2-LSTM, the outputs of input gate are mostly concentrated into larger value, this suggests that the model we trained didn't try to drop any information of the current input in the each step. While in the forget gate, the outputs are mostly concentrated to the boundaries. This observation shows that our training algorithm works as we expect. It also suggests the model we trained using G^2-LSTM tries to totally remember/forget each semantic aspect during the recurrent step.

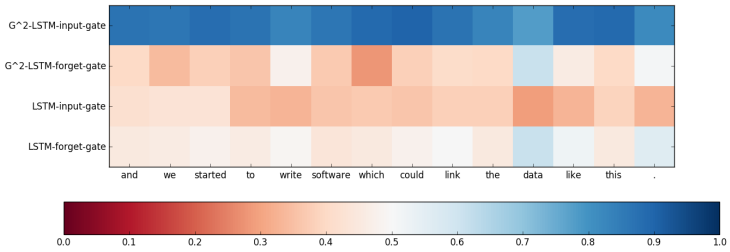


Figure 2: The gate value visualization in German→English task.

We provide a case study for the sampled data. As it is hard to go through any specific dimension of hidden state, we just calculate the averaged output values of the gate functions. In particular, for each word, we focus on the average value of input/forget gate functions in the first layer and check whether the averaged value is reasonable by checking the word in the sentence. We plot the heatmap of the English sentence part in Figure 2. More visualizations can be found in Appendix E. First, we can see that our G^2-LSTM never drop information in the input gate, the averaged gate value is relatively large. However, for LSTM, the values of input gate are small. Even some of the words with small value are very meaningful. As those words are not included into LSTM, they cannot be summarized and abstracted during encoding and decoding, which is not good for machine translation. Second, we can see that most of the words where its forget gate value is small are conjunctions or punctuations. We conjecture this is how the G^2-LSTM rearranges the sentence: it learns the hierarchical structure during the recurrent steps.

5 CONCLUSION AND FUTURE WORK

In this paper, we leverage the recent developed Gumbel-Softmax trick in the field of variantional method. We apply the trick to the gate function to approximate the Bernoulli distribution, and train the model with standard backpropagation. Experimental results show the proposed method can achieve better performances in several benchmark sequence modelling tasks, including language modelling and machine translation. Moreover, we show that the learnt models are much more robust with low-precision approximation and low-rank approximation to the parameters. All of these results indicate that the proposed algorithm can bring a better and much more flexible model.

REFERENCES

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron Courville, and Yoshua Bengio. An actor-critic algorithm for sequence prediction. *arXiv preprint arXiv:1607.07086*, 2016.

Denny Britz, Anna Goldie, Thang Luong, and Quoc Le. Massive exploration of neural machine translation architectures. *arXiv preprint arXiv:1703.03906*, 2017.

Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, and Marcello Federico. Report on the 11th iwslt evaluation campaign, iwslt 2014. In *Proceedings of the International Workshop on Spoken Language Translation, Hanoi, Vietnam*, 2014.

Pratik Chaudhari, Anna Choromanska, Stefano Soatto, and Yann LeCun. Entropy-sgd: Biasing gradient descent into wide valleys. *arXiv preprint arXiv:1611.01838*, 2016.

Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, pp. 1019–1027, 2016.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122*, 2017.

Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999.

David Haussler, Manfred Opper, et al. Mutual information, metric entropy and cumulative relative entropy risk. *The Annals of Statistics*, 25(6):2451–2492, 1997.

Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02): 107–116, 1998.

Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997a.

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

Po-Sen Huang, Chong Wang, Dengyong Zhou, and Li Deng. Toward neural phrase-based machine translation.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. On using very large target vocabulary for neural machine translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 1–10, Beijing, China, July 2015. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P15-1001`.

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. In *AAAI*, pp. 2741–2749, 2016.

David Krueger, Tegan Maharaj, Janos Kramar, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Aaron Courville, and Christopher Pal. Zoneout: Regularizing rnns by randomly preserving hidden activations. 2016.

Matt J Kusner and José Miguel Hernández-Lobato. Gans for sequences of discrete elements with the gumbel-softmax distribution. *arXiv preprint arXiv:1611.04051*, 2016.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pp. 311–318. Association for Computational Linguistics, 2002.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*, 2015.

Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. Recurrent dropout without memory loss. *arXiv preprint arXiv:1603.05118*, 2016.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *ACL*, 2016.

Sandeep Subramanian, Sai Rajeswar, Francis Dutil, Christopher Pal, and Aaron Courville. Adversarial generation of natural language. *ACL 2017*, pp. 241, 2017.

Ruben Villegas, Jimei Yang, Yuliang Zou, Sungryull Sohn, Xunyu Lin, and Honglak Lee. Learning to generate long-term future via hierarchical prediction. *arXiv preprint arXiv:1704.05831*, 2017.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3156–3164, 2015.

Sam Wiseman and Alexander M. Rush. Sequence-to-sequence learning as beam-search optimization. In *EMNLP*, November 2016.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Shi Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems*, pp. 802–810, 2015.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, pp. 2048–2057, 2015.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Yu Zhang, Guoguo Chen, Dong Yu, Kaisheng Yaco, Sanjeev Khudanpur, and James Glass. Highway long short-term memory rnns for distant speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pp. 5755–5759. IEEE, 2016.

A GUMBEL-SOFTMAX TRICK

The Gumbel-softmax trick proposed in Jang et al. (2016); Maddison et al. (2016) developed a continuous relaxation of discrete random variables in stochastic computational graphs. The main idea of the method is that the multinomial distribution can be represented according to Gumbel-Max trick, thus can be approximated as Gumbel-softmax distribution. In detail, given a probability distribution over k categories with parameter $\pi_1, \pi_2 \cdots \pi_k$. The Gumbel-softmax trick approximately samples the categorical variable according to:

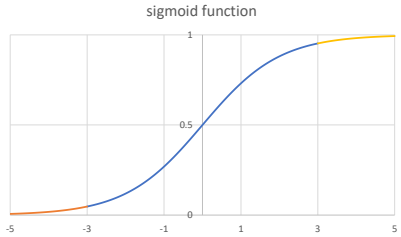
$$y_i = \frac{\exp((\log(\pi_i) + g_i)/\tau}{\sum_{j=1}^{k} \exp((\log(\pi_j) + g_j)/\tau} \qquad \text{for } i = 1, \ldots, k, \tag{18}$$

where τ is the temperature and g_i is independent and identically sampled according to Gumbel distribution: $u_i \sim \text{Uniform}(0, 1), g_i = -\log(-\log(u))$.

By using the Gumbel-softmax trick, we can generate sample $y = (y_1, ..., y_k)$ approximated the categorical distribution. As the randomness g is independent of π (which is usually the model), we can use reparameterization trick and optimize the model parameters by standard gradient backpropagation. Gumbel-softmax trick has been adopted in several applications such as variation autoencoder (Jang et al., 2016), generative adversarial net (Kusner & Hernández-Lobato, 2016), and language generation (Subramanian et al., 2017). To the best of our knowledge, this is the first work to introduce the Gumbel-Softmax trick in LSTM for robust training purpose.

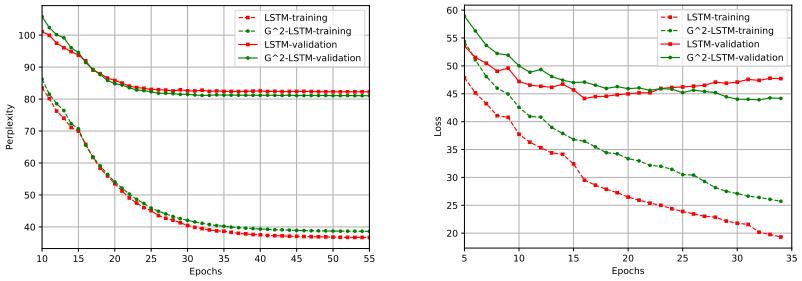
B FLATNESS OF SIGMOID FUNCTION

In the sigmoid function shown in the figure, the yellow and orange lines indicates the flat region while the blue line corresponds the sharp region. The output of the sigmoid function will only slightly change when the input perturbs a little if it is in the flat region.



C TRAINING/VALIDATION CURVE

Here we summarize the training and validation curves of the experiments between our proposed model and the baseline model. As shown in Figure 3, our proposed model converges slower during the training process. At last, our proposed model reaches lower validation error at a point that has higher training error, which shows our model generalizes better.



(a) Penn Tree Bank dataset (b) German→English IWSLT dataset

Figure 3: Training/validation curves of language modelling and machine translation tasks.

D EXTRA EXPERIMENTS ON ROBUSTNESS

We did an extra set of experiments on language modeling to show our models are more robust than the baseline model, no matter what the parameters (c, r in low precision compression, $rank$ in low rank compression) are. The results are shown in Table 6 and Table 7.

Table 6: Low precision compression results on Penn Tree Bank dataset.

	Original	$c = 0.20, r = 0.10$	$c = 0.16, r = 0.08$	$c = 0.08, r = 0.04$
LSTM	78.4	78.9 (+0.5)	85.6 (+7.2)	158.3 (+79.9)
G^2-LSTM	**77.3**	**77.4** (**+0.1**)	**78.2** (**+0.9**)	**97.1** (**+19.8**)

Table 7: Low rank compression results on Penn Tree Bank dataset.

	Original	$rank = 128$	$rank = 64$	$rank = 32$	$rank = 16$
LSTM	78.4	81.4 (+3.0)	94.7 (+16.3)	163.6 (+85.2)	328.7 (+250.3)
G^2-LSTM	**77.3**	**78.3** (**+1.0**)	**80.4** (**+3.1**)	**89.6** (**+12.3**)	**114.8** (**+37.5**)

E EXAMPLES

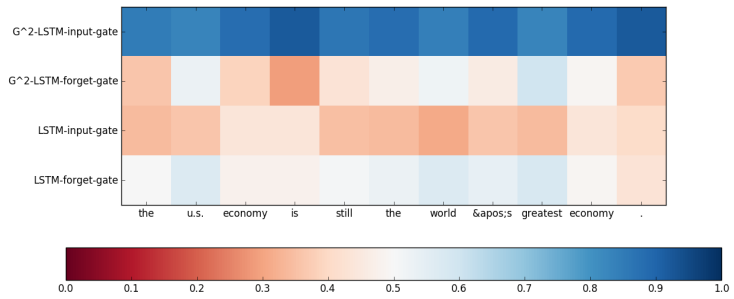


Figure 4: The gate value visualization in German→English task.

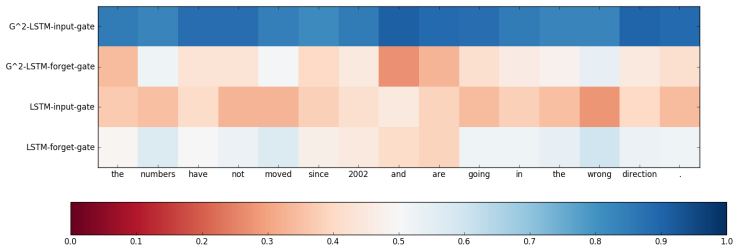


Figure 5: The gate value visualization in German→English task.

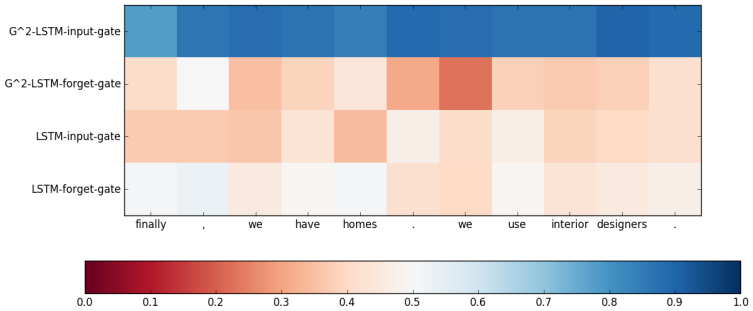


Figure 6: The gate value visualization in German→English task.

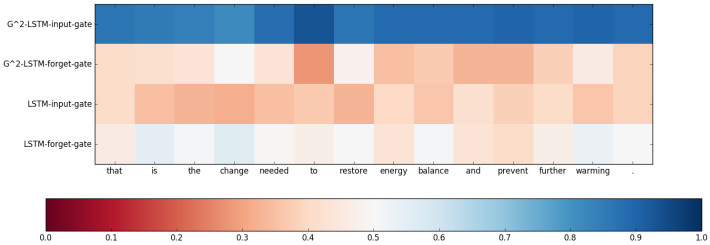


Figure 7: The gate value visualization in German→English task.

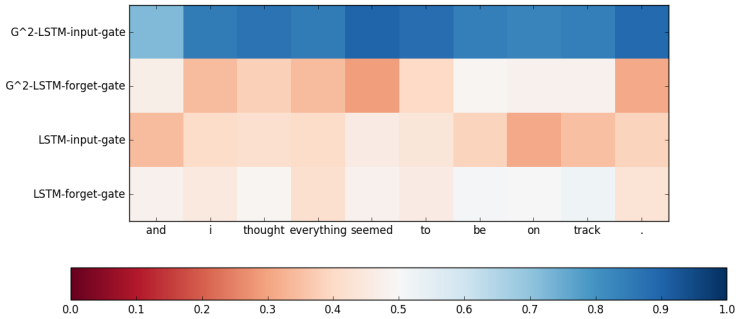


Figure 8: The gate value visualization in German→English task.