Towards Binary-Valued Gates for Robust LSTM Training

Zhuohan Li [1] **Di He** [1] **Fei Tian** [2] **Wei Chen** [2] **Tao Qin** [2] **Liwei Wang** [1 3] **Tie-Yan Liu** [2]

Abstract

Long Short-Term Memory (LSTM) is one of the most widely used recurrent structures in sequence modeling. It aims to use gates to control information flow (e.g., whether to skip some information or not) in the recurrent computations, although its practical implementation based on soft gates only partially achieves this goal. In this paper, we propose a new way for LSTM training, which pushes the output values of the gates towards 0 or 1. By doing so, we can better control the information flow: the gates are mostly open or closed, instead of in a middle state, which makes the results more interpretable. Empirical studies show that (1) Although it seems that we restrict the model capacity, there is no performance drop: we achieve better or comparable performances due to its better generalization ability; (2) The outputs of gates are not sensitive to their inputs: we can easily compress the LSTM unit in multiple ways, e.g., low-rank approximation and low-precision approximation. The compressed models are even better than the baseline models without compression.

1. Introduction

Recurrent neural networks (RNNs) (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; He et al., 2016), 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 RNNs, long short-term memory (LSTM) (Gers et al., 1999; Hochreiter & Schmidhuber, 1997b) networks were proposed, which introduce *gate functions* to control the information flow in a recurrent unit: a *forget gate function* to determine how much previous information should be excluded for the current step, an *input gate function* to find relevant signals to be absorbed into the hidden context, and an *output gate function* for prediction and decision making. For ease of optimization, in practical implementation, one usually uses the element-wise sigmoid function to mimic the gates, whose outputs are soft values between 0 and 1.

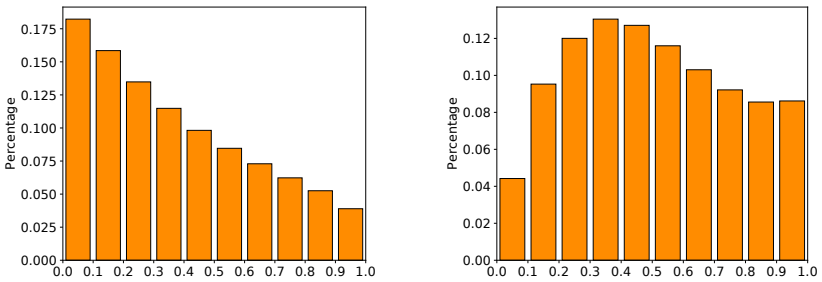
By using such gates with many more parameters, LSTM usually performs much better than conventional RNNs. However, when looking deep into the unit, we empirically find that the values of the gates are not that meaningful as the design logic. For example, in Figure 1, the distributions of the forget gate values and input gate values are not sharp and most of the values are in the middle state (around 0.5), meaning that most of the gate values are ambiguous in LSTM. This phenomenon contradicts the design of both gates: to control whether or not to take the information from the previous timesteps or the new inputs. At the same time, several works (Murdoch & Szlam, 2017; Karpathy et al., 2015) show that most cell coordinates of LSTM are hard to find particular meanings.

In this paper, we propose to push 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 the information in or skip by "opening" or "closing" the gates during the recurrent computation, which reflects more accurate and clear linguistic and structural information. Second, similar to BitNet in image classification (Courbariaux et al., 2016), by pushing the activation function to be binarized, we can learn a model that is ready for further compression. Third, training LSTM towards binary-valued gates enables better generation of the learned model. According

The work was done while the first author was visiting Microsoft Research Asia. [1]Key Laboratory of Machine Perception, MOE, School of EECS, Peking University [2]Microsoft Research [3]Center for Data Science, Peking University, Beijing Institute of Big Data Research. Correspondence to: Tao Qin <taoqin@microsoft.com>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

[1]The output of a gate function is usually a vector. For simplicity, in the paper, we 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 also say that each dimension of the output vector of the gate function is a gate, and say a gate is open/closed if its value is close to 1/0.



(a) Input gates in LSTM (b) Forget gates in LSTM

Figure 1. Histograms of gate value distributions in LSTM, based on the gate outputs of the first-layer LSTM in the decoder from 10000 sentence pairs IWSLT14 German→English training sets.

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 a set of parameters to make the values of the gates approaching zero or one, namely residing in the flat region of the sigmoid function, which 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 small temperature. However, this is equivalent to rescaling the input and cannot guarantee the values of the learned gates to be close to 0 or 1. To tackle this challenge, in this paper, we leverage the Gumbel-Softmax estimator developed for variational methods (Jang et al., 2016; Maddison et al., 2016). The estimator generates approximated and differentiable samples for categorical latent variables in a stochastic computational graph, e.g., variational autoencoder. Specifically, during training, we apply the Gumbel-Softmax estimator 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 the learned model Gumbel-Gate LSTM (G^2-LSTM). We conduct experiments on language modeling and machine translation to verify our proposed method. We have the following observations from experimental results:

- Our method restricts the gate outputs to be close to the boundary, and thus reduces the representation power. Surprisingly, there is no performance drop. Furthermore, our model achieves better or comparable results compared to the baseline model.
- Our learned model is easy for further compression. We apply several model compression algorithms to the parameters in the gates, including low-precision approximation and low-rank approximation, and results show that our compressed model can be even better than the baseline model without compression.
- We investigate a set of samples and find that the gates in our learned model are meaningful and intuitively interpretable. We show our model can automatically learn the boundaries in the sentences.

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

2. Background

2.1. Gumbel-Softmax Estimator

Jang et al. (2016) and Maddison et al. (2016) develop 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 by Gumbel-Softmax distribution. In detail, given a probability distribution over k categories with parameter $\pi_1, \pi_2, \ldots, \pi_k$, the Gumbel-Softmax estimator gives an approximate one-hot sample y with

$$y_i = \frac{\exp((\log \pi_i + q_i)/\tau)}{\sum_{j=1}^{k} \exp((\log \pi_j + q_j)/\tau)} \quad \text{for } i = 1, \ldots, k, \quad (1)$$

where τ is the temperature and q_i is independently sampled from Gumbel distribution: $q_i = -\log(-\log U_i), U_i \sim \text{Uniform}(0, 1)$.

By using the Gumbel-Softmax estimator, we can generate sample $y = (y_1, ..., y_k)$ to approximate the categorical distribution. Furthermore, as the randomness q is independent of π (which is usually defined by a set of parameters), we can use reparameterization trick to optimize the model parameters using standard backpropagation algorithms. Gumbel-Softmax estimator has been adopted in

several applications such as variation autoencoder (Jang et al., 2016), generative adversarial network (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 estimator in LSTM for robust training purpose.

2.2. Loss surface and generalization

The concept of sharp and flat minima has been first discussed in (Hochreiter & Schmidhuber, 1997a; Haussler et al., 1997). Intuitively, a flat minimum x of a loss $f(\cdot)$ corresponds to the point for which the value of function f varies slowly in a relatively large neighborhood of x. In contrast, a sharp minimum x is such that the function f changes rapidly in a small neighborhood of x. The sensitivity of the loss function at sharp minima negatively impacts the generalization ability of a trained model on new data. Recently, several papers discuss how to modify the training process and to learn a model in a flat region so as to obtain better generalization ability. Keskar et al. (2016) show by using small-batch training, the learned model is more likely to converge to a flat region rather than a sharp one. Chaudhari et al. (2016) propose a new objective function considering the local entropy and push the model to be optimized towards a wide valley.

3. The Proposed Training Algorithm

In this section, we present a new and robust training algorithm for LSTM by learning towards binary-valued gates.

3.1. Long Short-Term Memory RNN

Recurrent neural networks process an input sequence $\{x_1, x_2, \ldots, x_T\}$ sequentially and construct a corresponding sequence of hidden states/representations $\{h_1, h_2, \ldots, h_T\}$. In single-layer recurrent neural networks, the hidden states $\{h_1, h_2, \ldots, h_T\}$ are used for prediction or decision making. In deep (stacked) recurrent neural networks, the hidden states in layer k are used as inputs to layer $k+1$.

In recurrent neural networks, each hidden state is trained (implicitly) to remember and emphasize task-relevant aspects of the preceding inputs, and to incorporate new inputs via a recurrent operator, T, which converts the previous hidden state and present input into a new hidden state, e.g.,

$$h_t = T(h_{t-1}, x_t) = \tanh(W_h h_{t-1} + W_x x_t + b),$$

where W_h, W_x and b are parameters.

Long short-term memory RNN (LSTM) (Hochreiter & Schmidhuber, 1997b) is a carefully designed recurrent structure. In addition to the hidden state h_t used as a transient representation of state at timestep t, LSTM introduces a memory cell c_t, intended for internal long-term storage. c_t and h_t are computed via three gate functions. The forget gate function f_t directly connects c_t to the memory cell c_{t-1} of the previous timestep via an element-wise multiplication. Large values of the forget gates cause the cell to remember most (if not all) of its previous values. The other gates control the flow of information in input (i_t) and output (o_t) of the cell. Each gate function has a weight matrix and a bias vector; we use subscripts f, i and o to denote parameters for the forget gate function, the input gate function and the output gate function respectively, e.g., the parameters for the forget gate function are denoted by W_{xf}, W_{hf}, and b_f.

With the above notations, an LSTM is formally defined as

$$\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i), && (2)\\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f), && (3)\\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o), && (4)\\
g_t &= \tanh(W_{xg}x_t + W_{hg}h_{t-1} + b_g), && (5)\\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t, && (6)\\
h_t &= o_t \odot \tanh(c_t), && (7)
\end{aligned}$$

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 parameters than the simple RNN unit. As we can see from Eqn. (2) - (7), a large percentage of the parameters are used to compute the gate (sigmoid) functions. If we can push the outputs of the gates to the saturation area of the sigmoid function (i.e., towards 0 or 1), the loss function with respect to the parameters in the gates will be flat: if the parameters in the gates perturb, the change to the output of the gates is small due to the sigmoid operator (see Figure 2), and then the change to the loss is little, which means the flat region of the loss. First, as such model is robust to small parameter changes, it is robust to different model compression methods, e.g., low-precision compression or low-rank compression. Second, as discussed in (Chaudhari et al., 2016), minima in a flat region is more likely to generalize better, and thus toward binary-valued gates may lead to better test performance.

However, the task of training towards binary-valued gates is quite challenging. One straightforward idea is to sharpen the sigmoid function by using a smaller temperature, i.e., $f_{W,b}(x) = \sigma((Wx+b)/\tau)$, where $\tau < 1$ is the temperature. However, it is computationally equivalent to $f_{W',b'}(x) = \sigma(W'x + b')$ by setting $W' = W/\tau$ and $b' = b/\tau$. Then using a small temperature is equivalent to rescale the initial parameters as well as the gradients to a larger range. Usually, using an initial point in a large range with a large learning rate will harm the optimization process, and apparently cannot guarantee the outputs to be close to the boundary after training.

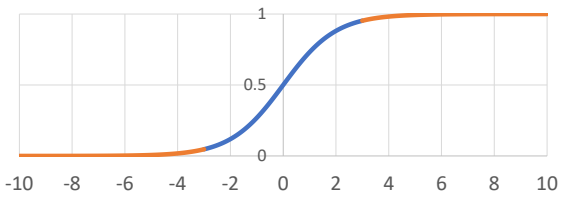


Figure 2. The orange parts correspond to the saturation area of the sigmoid function.

In this work, we leverage the recently developed Gumbel-Softmax trick. This trick is efficient in approximating discrete distributions, and is one of the widely used methods to learn discrete random variables in stochastic computational graphs. We first provide a proposition about the approximation ability of this trick for Bernoulli distribution, which will be used in our proposed algorithm.

Proposition 3.1. *Assume $\sigma(\cdot)$ is the sigmoid function. Given $\alpha \in \mathbb{R}$ and temperature $\tau > 0$, we define random variable $D_\alpha \sim B(\sigma(\alpha))$ where $B(\sigma(\alpha))$ is the Bernoulli distribution with parameter $\sigma(\alpha)$, and define $G(\alpha, \tau) = \sigma\left(\frac{\alpha+\log U-\log(1-U)}{\tau}\right)$ where $U \sim$ Uniform$(0,1)$. Then the following inequalities hold for arbitrary $\epsilon \in (0, 1/2)$,*

$$\begin{aligned} P(D_\alpha = 1) - (\tau/4)\log(1/\epsilon) &\le P(G(\alpha,\tau) \ge 1-\epsilon) \\ &\le P(D_\alpha = 1), \qquad (8) \\ P(D_\alpha = 0) - (\tau/4)\log(1/\epsilon) &\le P(G(\alpha,\tau) \le \epsilon) \\ &\le P(D_\alpha = 0). \qquad (9) \end{aligned}$$

Proof. Since $\sigma^{-1}(x) = \log\left(\frac{x}{1-x}\right)$, we have

$$\begin{aligned} & P(G(\alpha,\tau) \ge 1-\epsilon) \\ &= P\left(\frac{\alpha+\log U-\log(1-U)}{\tau} \ge \log(1/\epsilon - 1)\right) \\ &= P(e^{\alpha-\tau\log(1/\epsilon-1)} \ge (1-U)/U) \\ &= P\left(U \ge \frac{1}{1+e^{\alpha-\tau\log(1/\epsilon-1)}}\right) \\ &= \sigma(\alpha - \tau\log(1/\epsilon - 1)). \end{aligned}$$

Considering that sigmoid function is $(1/4)$-Lipschitz continuous and morotonically increasing, we have

$$\begin{aligned} & P(D_\alpha = 1) - P(G(\alpha,\tau) \ge 1-\epsilon) \\ &= \sigma(\alpha) - \sigma(\alpha - \tau\log(1/\epsilon - 1)) \\ &\le (\tau/4)\log(1/\epsilon - 1) \le (\tau/4)\log(1/\epsilon) \end{aligned}$$

and $P(D_\alpha = 1) - P(G(\alpha,\tau) \ge 1-\epsilon) \ge 0$. We omit the proof for Eqn. (9) as it is almost identical to the proof of Eqn. (8). □

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 Eqn. (8) and (9). When the temperature τ approaches positive zero, we directly obtain the following property, which is also proved by Maddison et al. (2016),

$$\begin{aligned} P\left(\lim_{\tau\to 0^+} G(\alpha,\tau) = 1\right) &= P(D_\alpha = 1), \\ P\left(\lim_{\tau\to 0^+} G(\alpha,\tau) = 0\right) &= P(D_\alpha = 0). \qquad (10) \end{aligned}$$

We apply this method into the computation of the gates. Imagine a one-dimensional gate $\sigma(\alpha(\theta))$ where α is a scalar parameterized by θ, and assume the model will produce a larger loss if the output of the gate is close to one, and produce a smaller loss if the gate value is close to zero. If we can repeatedly sample the output of the gate using $G(\alpha(\theta),\tau) = \sigma\left(\frac{\alpha(\theta)+\log U-\log(1-U)}{\tau}\right)$ and estimate the loss, any gradient-based algorithm will push the parameter θ such that the output value of the gate is close to zero in order to minimize the expected loss. By this way, we can optimize towards the binary-valued gates.

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

In particular, we only push the outputs of input gates and forget gates towards binary values as the output gates usually need fine-granularity information for decision making which makes binary values less desirable. To justify this, we conducted similar experiments and observed a performance drop when pushing the output gates to 0/1 together with the input gates and the forget gates.

We call our proposed learning method Gumbel-Gate LSTM (G^2-LSTM), which works as follows during training:

$$\begin{aligned} i_t &= G(W_{xi}x_t + W_{hi}h_{t-1} + b_i, \tau) \qquad (11) \\ f_t &= G(W_{xf}x_t + W_{hf}h_{t-1} + b_f, \tau) \qquad (12) \\ o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \qquad (13) \\ g_t &= \tanh(W_{xg}x_t + W_{hg}h_{t-1} + b_g) \qquad (14) \\ c_t &= f_t \odot c_{t-1} + i_t \odot g_t \qquad (15) \\ h_t &= o_t \odot \tanh(c_t). \qquad (16) \end{aligned}$$

In the forward pass, we first independently sample values for U in each time step, then update G^2-LSTMs using Eqn. (11) - (16) and calculate the loss, e.g., negative log likelihood loss. In the backward pass, as G is continuous and differentiable with respect to the parameters and the loss is continuous and differentiable with respect to G, we can use any standard gradient-based method to update the model parameters.

Table 1. Performance comparison on language model (perplexity)

Model	Size	Valid	Test
Existing results			
Unregularzed LSTM	7M	120.7	114.5
NR-dropout (Zaremba et al., 2014)	66M	82.2	78.4
Zoneout (Krueger et al., 2016)	66M	-	77.4
Variational LSTM (Gal & Ghahramani, 2016)	19M	-	73.4
CharCNN (Kim et al., 2016)	21M	72.4	78.9
Pointer Sentinel-LSTM (Merity et al., 2016)	51M	-	70.9
LSTM + continuous cache pointer (Grave et al., 2016)	-	-	72.1
Variational LSTM + augmented loss (Inan et al., 2016)	51M	71.1	68.5
Variational RHN (Zilly et al., 2016)	23M	67.9	65.4
NAS Cell (Zoph & Le, 2016)	54M	-	62.4
4-layer skip connection LSTM (Melis et al., 2017)	24M	60.9	58.3
AWD-LSTM w/o finetune (Merity et al., 2017)	24M	60.7	58.8
AWD-LSTM (Baseline) (Merity et al., 2017)	24M	60.0	57.3
Our system			
Sharpened Sigmoid AWD-LSTM w/o finetune	24M	61.6	59.4
Sharpened Sigmoid AWD-LSTM	24M	59.9	57.5
G^2-LSTM w/o finetune	24M	**60.4**	**58.2**
G^2-LSTM	24M	**58.5**	**56.1**
+continuous cache pointer			
AWD-LSTM + continuous cache pointer (Merity et al., 2017)	24M	53.9	52.8
Sharpened Sigmoid AWD-LSTM + continuous cache pointer	24M	53.9	53.2
G^2-LSTM + continuous cache pointer	24M	**52.9**	**52.1**

4. Experiments

4.1. Settings

We tested the proposed training algorithm on two tasks – language modeling and machine translation.[2]

4.1.1. Language Modeling

Language modeling is a very basic task for LSTM. We used the Penn Treebank corpus that contains about 1 million words. The task is to train an LSTM model to correctly predict the next word conditioned on previous words. A model is evaluated by the prediction perplexity: smaller the perplexity, better the prediction.

We followed the practice in (Merity et al., 2017) to set up the model architecture for LSTM: a stacked three-layer LSTM with drop-connect (Wan et al., 2013) on recurrent weights and a variant of averaged stochastic gradient descent (ASGD) (Polyak & Juditsky, 1992) for optimization, with a 500-epoch training phase and a 500-epoch finetune phase. Our training code for G^2-LSTM was also based on the code released by Merity et al. (2017). Since the temperature τ in G^2-LSTM does not have significant effects on the results, we set it to 0.9 and followed all other configurations in Merity et al. (2017). We added neural cache model (Grave et al., 2016) on the top of our trained language model to further improve the perplexity.

4.1.2. Machine Translation

We used two datasets for experiments on neural machine translation (NMT): (1) IWSLT'14 German→English translation dataset (Cettolo et al., 2014), which is widely adopted in machine learning community (Bahdanau et al., 2016; Wiseman & Rush, 2016a; Ranzato et al., 2015). The training/validation/test sets contain about 153K/7K/7K sentence pairs respectively, with words pre-processed into sub-word units using byte pair encoding (BPE) (Sennrich et al., 2016). We chose 25K most frequent sub-word units as the vocabulary for both German and English. (2) English→German translation dataset in WMT'14, which is also commonly used as a benchmark task to evaluate different NMT models (Bahdanau et al., 2014; Wu et al., 2016; Gehring et al., 2017; He et al., 2017). The training set contains 4.5M English→German sentence pairs, Newstest2014 is used as the test set, and the concatenation of Newstest2012 and Newstest2013 is used as the validation set. Similarly, BPE was used to form a vocabulary of most frequent 30K sub-word units for both languages. In both datasets, we removed

[2]Codes for the experiments are available at https://github.com/zhuohan123/g2-lstm

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	BSO (Wiseman & Rush, 2016b)	26.36
MRT (Shen et al., 2015)	20.45	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
Sharpened Sigmoid	21.64	-	29.73
G^2-LSTM	22.43	-	**31.95**

the sentences with more than 64 sub-word units in training.

For the German→English dataset, we adopted a stacked two-layer encoder-decoder framework. We set the size of word embedding and hidden state to 256. As the amount of data in the English→German dataset is much larger, we adopted 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 was bi-directional. We also used dropout in training stacked LSTM as in (Zaremba et al., 2014), with dropout value determined via validation set performance. For both experiments, we set the temperature τ for G^2-LSTM to 0.9, the same as language modeling task. The mini-batch size was 32/64 for German→English/English→German respectively. All models were trained with AdaDelta (Zeiler, 2012) on one M40 GPU. Both gradient clipping norms were set to 2.0. We used tokenized case-insensitive and case-sensitive BLEU as evaluation measure for German→English/English→German respectively, following common practice.[3] The beam size is set to 5 during the inference step.

4.2. Experimental Results

The experimental results are shown in Table 1 and 2. We compare our training method with two algorithms. For the first algorithm (we call it *Baseline*), we remove the Gumble-Softmax trick and train the model using standard optimization methods. For the second algorithm (we call it *Sharpened Sigmoid*), we use a sharpened sigmoid function as described in Section 3.2 by setting $\tau = 0.2$ and check whether such trick can bring better performance.

From the results, we can see that our learned models are competitive or better than all baseline models. In language modeling task, we outperform the baseline algorithms for 0.7/1.1 points (1.2/1.4 points without continuous cache pointer) in terms of test perplexity. For machine translation, we outperform the baselines for 0.95/2.22 and 0.54/0.79 points in terms of BLEU score for German→English and English→German dataset respectively. Note that the only difference between G^2-LSTM and the baselines is the training algorithm, while they adopt the same model structure. Thus, better results of G^2-LSTM demonstrate the effectiveness of our proposed training method. This shows that restricting the outputs of the gates towards binary values doesn't bring performance drop at all. On the contrary, the performances are even better. We conclude that such benefit may come from the better generalization ability.

We also list the performance of previous works in literature, which may adopt different model architectures or settings. For language modeling, we obtain better performance results compared to the previous works listed in the table. For German→English translation, the two-layer stacked encoder-decoder model we learned outperforms all previous works. For English→German translation, our result is worse than GNMT (Wu et al., 2016) as they used a stacked eight-layer model while we only used a three-layer one.

4.3. Sensitivity Analysis

We conducted a set of experiments to test how sensitive our learned models were when compressing their gate parameters. We considered two ways of compression as follows.

Low-Precision Compression We compressed parameters in the input and forget gates to lower precision. Doing so the model can be compressed to a relatively small size. In particular, we applied round and clip operations to the parameters of the input and forget gates:

$$\text{round}_r(x) = \text{round}(x/r) * r, \quad (17)$$

$$\text{clip}_c(x) = \text{clip}(x, -c, c). \quad (18)$$

We tested two settings of low-precision compression. In the first setting (named as *Round*), we rounded the parameters using Eqn. (17). In this way, we reduced the support set of the parameters in the gates. In the second setting (named as *Round & Clip*), we further clipped the rounded value to a

[3] 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 ($rank = 64$)
Baseline	52.8	53.2 (+0.4)	53.6 (+0.8)	56.6 (+3.8)	65.5 (+12.7)
Sharpened Sigmoid	53.2	53.5 (+0.3)	53.6 (**+0.4**)	54.6 (+1.4)	60.0 (+6.8)
G^2-LSTM	**52.1**	**52.2** (**+0.1**)	**52.8** (+0.7)	**53.3** (**+1.2**)	**56.0** (**+3.9**)

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

	Original	Round	Round & clip	SVD ($rank = 32$)	SVD ($rank = 16$)
Baseline	31.00	28.65 (-2.35)	21.97 (-9.03)	30.52 (-0.48)	29.56 (-1.44)
Sharpened Sigmoid	29.73	27.08 (-2.65)	25.14 (-4.59)	29.17 (-0.53)	28.82 (-0.91)
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 ($rank = 16$)
Baseline	21.89	16.22 (-5.67)	16.03 (-5.86)	21.15 (-0.74)	19.99 (-1.90)
Sharpened Sigmoid	21.64	16.85 (-4.79)	16.72 (-4.92)	20.98 (-0.66)	19.87 (-1.77)
G^2-LSTM	**22.43**	**20.15** (**-2.28**)	**20.29** (**-2.14**)	**22.16** (**-0.27**)	**21.84** (**-0.51**)

fixed range using Eqn. (18) and thus restricted the number of different values. As the two tasks are far different, we set the round parameter $r = 0.2$ and the clip parameter $c = 0.4$ for the task of language modeling, and set $c = 1.0$ and $r = 0.5$ for neural machine translation. As a result, parameters of input gates and forget gates in language modeling can only take values from $(0.0, \pm 0.2, \pm 0.4)$, and $(0.0, \pm 0.5, \pm 1.0)$ for machine translation.

Low-Rank Compression We compressed parameter matrices of the input/forget gates to lower-rank matrices through singular value decomposition, which can reduce the model size and lead to faster matrix multiplication. Given that the hidden states of the task of language modeling were of much larger dimension than that of neural machine translation, we set $rank = 64/128$ for language modeling and $rank = 16/32$ for neural machine translation.

We summarize the results in Table 3-5. From Table 3, we can see that for language modeling both the baseline and our learned model are quite robust to low-precision compression, but our model is much more robust and significantly outperforms the baseline with low-rank approximation. Even setting $rank = 64$ (roughly $12\times$ compression rate of the gates), we still get 56.0 perplexity, while the perplexity of the baseline model increases from 52.8 to 65.5, i.e., becoming 24% worse. For machine translation, our proposed method is always better than the baseline model, no matter for low-precision or low-rank compression. Even if setting $rank = 16$ (roughly $8\times/32\times$ compression rate of the gates for German→English and English→German respectively), we still get roughly comparable translation accuracy to the baseline model with full parameters. All results show that the models trained with our proposed method are less sensitive to parameter compression.

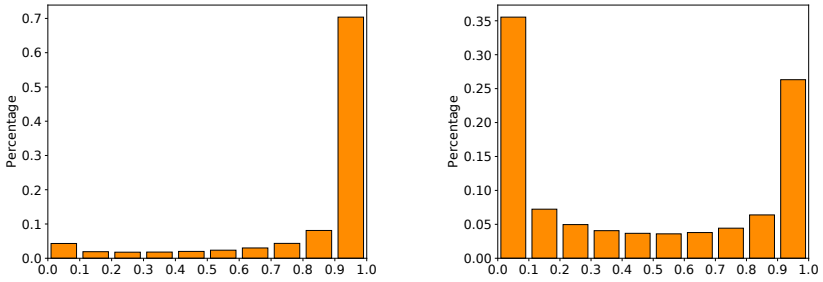
4.4. Visualization of the Gates

In addition to comparing the final performances, we further looked inside the learned models and checked the gates.

To well verify the effectiveness of our proposed G^2-LSTM, we did a set of experiments to show the values of gates learned by G^2-LSTM are near the boundary and reasonable, based on the model learned 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 sampled 10000 sentence pairs from the training set and fed them into the learned models. We got the output value vectors of the input/forget gates in the first layer of the decoder. We recorded the value of each element in the output vectors and plotted the distributions in Figure 1 and 3.

From the figures, we can see that although both LSTM and G^2-LSTM work reasonably well in practice, the output values of the gates are very different. In LSTM, the distributions of the gate values are relatively uniform and have no clear concentration. In contrast, the values of the input gates of G^2-LSTM are concentrated in the region close to 1, which suggests that our learned model tries to keep most information from the input words; the values of the forget gates are concentrated in the boundary regions (i.e., either the region close to 0 or the region close to 1). This observation shows that our training algorithm meets our expectation and successfully pushes the gates to 0/1.

Besides the overall distribution of gate values over a sam-



(a) Input gates in G^2-LSTM (b) Forget gates in G^2-LSTM

Figure 3. Histograms of gate value distributions in G^2-LSTM, from the same data as Figure 1.

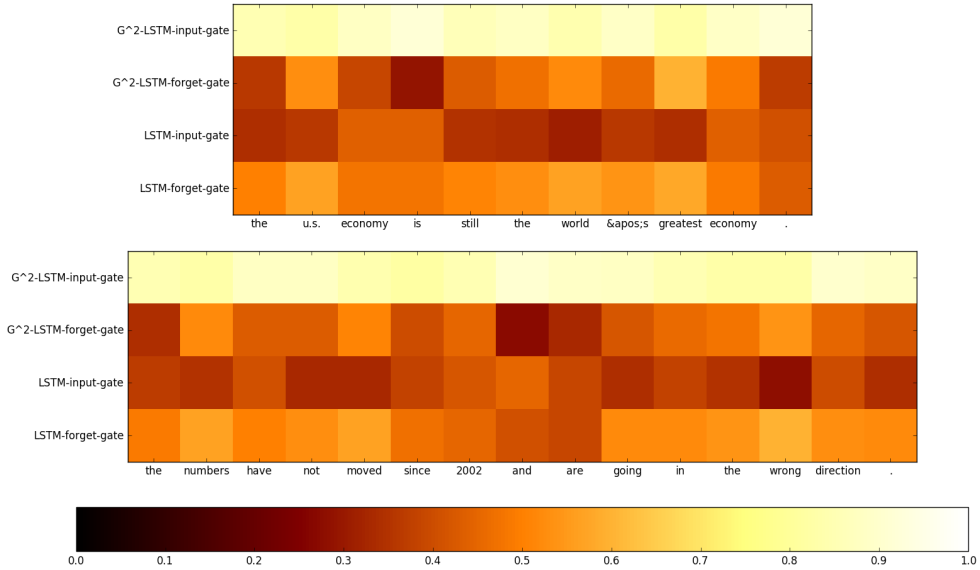


Figure 4. Visualization of average gate value at each timestep in LSTM and G^2-LSTM, from the same model as Figure 1.

pled set of training data, here we provide a case study for sampled sentences. We calculated the average value of the output vector of the input and forget gate functions for each word. In particular, we focused on the average value of the input/forget gate functions in the first layer and check whether the averages are reasonable. We plot the heatmap of the English sentence part in Figure 4.

First, we can see that our G^2-LSTM does not drop information in the input gate function since the average values are relatively large for all words. In contrast, the average values of the input gates of LSTM are sometimes small (less than 0.5), even for the meaningful word like "wrong". As those words are not included into LSTM, they cannot be effectively encoded and decoded, and thus lead to bad translation results. Second, for G^2-LSTM, most of the words with small values for forget gates are function words (e.g., conjunctions and punctuations) or the boundaries in clauses. That is, our training algorithm indeed ensures the model to forget information on the boundaries in the sentences, and reset the hidden states with new inputs.

5. Conclusion and Future Work

In this paper, we designed a new training algorithm for LSTM by leveraging the recently developed Gumbel-Softmax estimator. Our training algorithm can push the values of the input and forget gates to 0 or 1, leading to robust LSTM models. Experiments on language modeling and machine translation demonstrated the effectiveness of the proposed training algorithm.

We will explore following directions in the future. First, we will apply our algorithm to deeper models (e.g., 8+ layers) and test on larger datasets. Second, we have considered the tasks of language modeling and machine translation. We will study more applications such as question answering and text summarization.

Acknowledgements

This work was partially supported by National Basic Research Program of China (973 Program) (grant no. 2015CB352502), NSFC (61573026). We would like to thank Chen Xing and Qizhe Xie for helpful discussions, and the anonymous reviewers for their valuable comments on our paper.

References

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Bahdanau, D., Brakel, P., Xu, K., Goyal, A., Lowe, R., Pineau, J., Courville, A., and Bengio, Y. An actor-critic algorithm for sequence prediction. *arXiv preprint arXiv:1607.07086*, 2016.

Britz, D., Goldie, A., Luong, T., and Le, Q. Massive exploration of neural machine translation architectures. *arXiv preprint arXiv:1703.03906*, 2017.

Cettolo, M., Niehues, J., Stüker, S., Bentivogli, L., and Federico, M. Report on the 11th iwslt evaluation campaign, iwslt 2014. In *Proceedings of the International Workshop on Spoken Language Translation, Hanoi, Vietnam*, 2014.

Chaudhari, P., Choromanska, A., Soatto, S., and LeCun, Y. Entropy-sgd: Biasing gradient descent into wide valleys. *arXiv preprint arXiv:1611.01838*, 2016.

Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., and Bengio, Y. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

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

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122*, 2017.

Gers, F. A., Schmidhuber, J., and Cummins, F. Learning to forget: Continual prediction with lstm. 1999.

Grave, E., Joulin, A., and Usunier, N. Improving neural language models with a continuous cache. *arXiv preprint arXiv:1612.04426*, 2016.

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

He, D., Xia, Y., Qin, T., Wang, L., Yu, N., Liu, T., and Ma, W.-Y. Dual learning for machine translation. In *Advances in Neural Information Processing Systems*, pp. 820–828, 2016.

He, D., Lu, H., Xia, Y., Qin, T., Wang, L., and Liu, T. Decoding with value networks for neural machine translation. In *Advances in Neural Information Processing Systems*, pp. 177–186, 2017.

Hochreiter, S. 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.

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

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

Huang, P.-S., Wang, C., Zhou, D., and Deng, L. Toward neural phrase-based machine translation.

Inan, H., Khosravi, K., and Socher, R. Tying word vectors and word classifiers: A loss framework for language modeling. *arXiv preprint arXiv:1611.01462*, 2016.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Jean, S., Cho, K., Memisevic, R., and Bengio, Y. 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.

Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., and Wu, Y. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

Karpathy, A., Johnson, J., and Fei-Fei, L. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

Kim, Y., Jernite, Y., Sontag, D., and Rush, A. M. Character-aware neural language models. In *AAAI*, pp. 2741–2749, 2016.

Krueger, D., Maharaj, T., Kramar, J., Pezeshki, M., Ballas, N., Ke, N. R., Goyal, A., Bengio, Y., Courville, A., and Pal, C. Zoneout: Regularizing rnns by randomly preserving hidden activations. 2016.

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

Luong, M.-T., Pham, H., and Manning, C. D. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

Maddison, C. J., Mnih, A., and Teh, Y. W. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Melis, G., Dyer, C., and Blunsom, P. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*, 2017.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Merity, S., Keskar, N. S., and Socher, R. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*, 2017.

Murdoch, W. J. and Szlam, A. Automatic rule extraction from long short term memory networks. *arXiv preprint arXiv:1702.02540*, 2017.

Polyak, B. T. and Juditsky, A. B. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

Ranzato, M., Chopra, S., Auli, M., and Zaremba, W. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*, 2015.

Sennrich, R., Haddow, B., and Birch, A. Neural machine translation of rare words with subword units. In *ACL*, 2016.

Shen, S., Cheng, Y., He, Z., He, W., Wu, H., Sun, M., and Liu, Y. Minimum risk training for neural machine translation. *arXiv preprint arXiv:1512.02433*, 2015.

Subramanian, S., Rajeswar, S., Dutil, F., Pal, C., and Courville, A. Adversarial generation of natural language. *ACL 2017*, pp. 241, 2017.

Villegas, R., Yang, J., Zou, Y., Sohn, S., Lin, X., and Lee, H. Learning to generate long-term future via hierarchical prediction. *arXiv preprint arXiv:1704.05831*, 2017.

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

Wan, L., Zeiler, M., Zhang, S., Le Cun, Y., and Fergus, R. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pp. 1058–1066, 2013.

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

Wiseman, S. and Rush, A. M. Sequence-to-sequence learning as beam-search optimization. *arXiv preprint arXiv:1606.02960*, 2016b.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Xingjian, S., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-K., and Woo, W.-c. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems*, pp. 802–810, 2015.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., and Bengio, Y. Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, pp. 2048–2057, 2015.

Zaremba, W., Sutskever, I., and Vinyals, O. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

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

Zhang, Y., Chen, G., Yu, D., Yaco, K., Khudanpur, S., and Glass, J. 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.

Zilly, J. G., Srivastava, R. K., Koutník, J., and Schmidhuber, J. Recurrent highway networks. *arXiv preprint arXiv:1607.03474*, 2016.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.