Faster Training of Very Deep Networks Via p-Norm Gates

Trang Pham, Truyen Tran, Dinh Phung, Svetha Venkatesh
Center for Pattern Recognition and Data Analytics
Deakin University, Geelong Australia
Email: {phtra, truyen.tran, dinh.phung, svetha.venkatesh}@deakin.edu.au

***Abstract*—A major contributing factor to the recent advances in deep neural networks is structural units that let sensory information and gradients to propagate easily. Gating is one such structure that acts as a flow control. Gates are employed in many recent state-of-the-art recurrent models such as LSTM and GRU, and feedforward models such as Residual Nets and Highway Networks. This enables learning in very deep networks with hundred layers and helps achieve record-breaking results in vision (e.g., ImageNet with Residual Nets) and NLP (e.g., machine translation with GRU). However, there is limited work in analysing the role of gating in the learning process. In this paper, we propose a flexible p-norm gating scheme, which allows user-controllable flow and as a consequence, improve the learning speed. This scheme subsumes other existing gating schemes, including those in GRU, Highway Networks and Residual Nets as special cases. Experiments on large sequence and vector datasets demonstrate that the proposed gating scheme helps improve the learning speed significantly without extra overhead.**

I. Introduction

Deep neural networks are becoming the method of choice in vision [1], speech recognition [2] and NLP [3], [4]. Deep nets represent complex data more efficiently than shallow ones [5]. With more non-linear hidden layers, deep networks can theoretically model functions with higher complexity and nonlinearity [6]. However, learning standard feedforward networks with many hidden layers is notoriously difficult [7]. Likewise, standard recurrent networks suffer from vanishing gradients for long sequences [8] making gradient-based learning ineffective. A major reason is that many layers of non-linear transformation prevent the data signals and gradients from flowing easily through the network. In the forward direction from data to outcome, a change in data signals may not lead to any change in outcome, leading to the poor credit assignment problem. In the backward direction, a large error gradient at the outcome may not be propagated back to the data signals. As a result, learning stops prematurely without returning an informative mapping from data to outcome.

There have been several effective methods to tackle the problem. The first line of work is to use non-saturated non-linear transforms such as rectified linear units (ReLUs) [9], [1], [10], whose gradients are non-zero for a large portion of the input space. Another approach that also increases the level of linearity of the information propagation is through *gating* [11], [12]. The gates are extra control neural units that let part of information pass through a channel. They are learnable and have played an important role in state-of-the-art feedforward architectures such as Highway Networks [13] and Residual Networks [14], and recurrent architectures such as such as Long Short-Term Memory (LSTM) [11], [12] and Gated Recurrent Unit (GRU) [15].

Although the details of these architectures differ, they share a common gating scheme. More specifically, let $\boldsymbol{h}_t$ be the activation vector of size K (or memory cells, in the case of LSTM) at computational step t, where t can be the index of the hidden layer in feedforward networks, or the time step in recurrent networks. The updating of $\boldsymbol{h}_t$ follows the following rule:

$$\boldsymbol{h}_t \leftarrow \boldsymbol{\alpha}_1 * \tilde{\boldsymbol{h}}_t + \boldsymbol{\alpha}_2 * \boldsymbol{h}_{t-1} \quad (1)$$

where $\tilde{\boldsymbol{h}}_t$ is the non-linear transformation of $\boldsymbol{h}_t$ (and the input at t if given), $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \in [0,1]^k$ are gates and $*$ is point-wise multiplication. When $\boldsymbol{\alpha}_2 > \mathbf{0}$, a part of the previous activation vector is copied into the new vector. Thus the update has a nonlinear part (controlled by $\boldsymbol{\alpha}_1$) and a linear part (controlled by $\boldsymbol{\alpha}_2$). The nonlinear part keeps transforming the input to more complex output, whilst the linear part retains a part of input to pass across layers much easier. The linear part effectively prevents the gradient from vanishing even if there are hundreds of layers. For example, Highway Networks can be trained with more than 1000 layers [13], which were previous impossible for feedforward networks.

This updating rule opens room to study relationship between the two gates $\boldsymbol{\alpha}_1$ and $\boldsymbol{\alpha}_2$, and there has been a limited work in this direction. Existing work includes Residual Networks with $\boldsymbol{\alpha}_1 = \boldsymbol{\alpha}_2 = \mathbf{1}$, hence $\tilde{\boldsymbol{h}}_t$ plays the role of the residual. For the LSTM, there is no explicit relation between the two gates. The GRU and the work reported in [13] use $\boldsymbol{\alpha}_1 + \boldsymbol{\alpha}_2 = \mathbf{1}$, which leads to less parameters compared to the LSTM. This paper focuses on the later, and aims to address the inherent drawback in this linear relationship. In particular, when $\boldsymbol{\alpha}_1$ approaches $\mathbf{1}$ with rate λ, $\boldsymbol{\alpha}_2$ approaches $\mathbf{0}$ with the same rate, and this may prevent information from passing too early. To this end we propose a more flexible p-norm gating scheme, where the following relationship holds: $(\boldsymbol{\alpha}_1^p + \boldsymbol{\alpha}_2^p)^{1/p} = \mathbf{1}$ for $p > 0$ and the norm is applied element-wise. This introduces just one an extra controlling hyperparameter p. When $p = 1$, the scheme returns to original gating in Highway Networks and GRUs.

We evaluate this p-norm gating scheme on two settings: the traditional classification of vector data under Highway

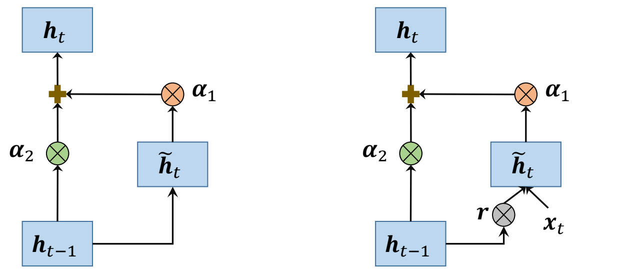
Networks and sequential language models under the GRUs. Extensive experiments demonstrate that with $p > 1$, the learning speed is significantly higher than existing gating schemes with $p = 1$. Compared with the original gating, learning with $p > 1$ is 2 to 3 times faster for vector data and more than 15% faster for sequential data.

The paper is organized as follows. Section 2 presents the Highway Networks, GRUs and the p-norm gating mechanism. Experiments and results with the two models are reported in Section 3. Finally, Section 4 discusses the findings further and concludes the paper.

II. Methods

In this section, we propose our p-norm gating scheme. To aid the exposition, we first briefly review the two state-of-the-art models that use gating mechanisms: Highway Networks [13] (a feedforward architecture for vector-to-vector mapping) and Gated Recurrent Units [15] (a recurrent architecture for sequence-to-sequence mapping) in Sections II-A and II-B, respectively.

Notational convention : We use bold lowercase letters for vectors and capital letters for matrices. The sigmoid function of a scalar x is defined as $\sigma(x) = [1 + \exp(-x)]^{-1}$, $x \in \mathbb{R}$. With a slight abuse of notation, we use $\sigma(\boldsymbol{x})$, where $\boldsymbol{x} = (x_1, x_2, ..., x_n)$ is a vector, to denote a vector $(\sigma(x_1), ..., \sigma(x_n))$. The same rule applies to other function of vector $g(\boldsymbol{x})$. The operator $*$ is used to denote element-wise multiplication. For both the feedforward networks and recurrent networks, we use index t to denote the *computational steps*, and it can be layers in feedforward networks or time step in recurrent networks. As shown from Fig. 1, the two architectures are quite similar except for when extra input $\boldsymbol{x}_t$ is available at each step.



(a) Highway Network layer (b) Gated Recurrent Unit

Figure 1. Gating mechanisms in Highway Networks and GRUs. The current hidden state $\boldsymbol{h}_t$ is the sum of candidate hidden state $\tilde{\boldsymbol{h}}_t$ moderated by $\boldsymbol{\alpha}_1$ and the previous hidden state $\boldsymbol{h}_{t-1}$ moderated by $\boldsymbol{\alpha}_2$

A. Highway Networks

A Highway Network is a feedforward neural network which maps an input vector $\boldsymbol{x}$ to an outcome y. A standard feedforward network consists of T hidden layers where the activation $\boldsymbol{h}_t \in \mathbb{R}^{k_t}$ at the t^{th} layer $(t = 1, .., T)$ is a non-linear function of its lower layer:

$$\boldsymbol{h}_t = g\left(W_t \boldsymbol{h}_{t-1} + \boldsymbol{b}_t\right)$$

where W_t and $\boldsymbol{b}_t$ are the parameter matrix and bias vector at the t^{th} layer, and $g(\cdot)$ is the element-wise non-linear transform. At the bottom layer, $\boldsymbol{h}_0$ is the input $\boldsymbol{x}$. The top hidden layer $\boldsymbol{h}_T$ is connected to the outcome.

Training very deep feedforward networks remains difficult for several reasons. First, the number of parameters grows with the depth of the network, which leads to overfitting. Second, the stack of multiple non-linear functions makes it difficult for the information and the gradients to pass through.

In Highway Networks, there are two modifications that resolve these problems: (i) Parameters are shared between layers leading to a compact model, and (ii) The activation function is modified by adding sigmoid gates that let information from lower layers pass linearly through. Fig. 1(a) illustrates a Highway Network layer. The first modification requires that all the hidden layers to have the same hidden units k. The bottom layer is identical to that of standard feedforward networks. The second modification defines a candidate hidden state $\tilde{\boldsymbol{h}}_t \in \mathbb{R}^k$ as the usual non-linear transform:

$$\tilde{\boldsymbol{h}}_t = g\left(W \boldsymbol{h}_{t-1} + \boldsymbol{b}\right)$$

where W and $\boldsymbol{b}$ are parameter matrix and bias vector that shared among all hidden layers. Finally the hidden state is gated by two gates $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \in [0,1]^k$ as follows:

$$\boldsymbol{h}_t = \boldsymbol{\alpha}_1 * \tilde{\boldsymbol{h}}_t + \boldsymbol{\alpha}_2 * \boldsymbol{h}_{t-1} \qquad (2)$$

for $t \geq 2$. The two gates $\boldsymbol{\alpha}_1$ and $\boldsymbol{\alpha}_2$ are sigmoid functions and can be independent, where $\boldsymbol{\alpha}_1 = \sigma\left(U_1 \boldsymbol{h}_{t-1} + \boldsymbol{c}_1\right)$ and $\boldsymbol{\alpha}_2 = \sigma\left(U_2 \boldsymbol{h}_{t-1} + \boldsymbol{c}_2\right)$ or summed to unit element-wise, e.g., $\boldsymbol{1} = \boldsymbol{\alpha}_1 + \boldsymbol{\alpha}_2$. The latter option was used in the paper of Highway Networks [13].

The part $\boldsymbol{\alpha}_2 * \boldsymbol{h}_{t-1}$, which is called *carry behavior*, makes the information from layers below pass easily through the network. This behavior also allows the back-propagation to compute the gradient more directly to the input. The net effect is that the networks can be very deep (up to thousand of layers).

B. Gated Recurrent Units

Recurrent neural networks: A recurrent neural network (RNN) is an extension of feedforward networks to map a variable-length input sequence $\boldsymbol{x}_1, ..., \boldsymbol{x}_T$ to a output sequence $y_1, ..., y_T$. An RNN allows self-loop connections and shared parameters across all steps of the sequence. For vanilla RNNs, the activation (which is also called hidden state) $\boldsymbol{h}_t$ is a function of the current input and the previous hidden state $\boldsymbol{h}_{t-1}$:

$$\boldsymbol{h}_t = g\left(W \boldsymbol{x}_t + U \boldsymbol{h}_{t-1} + \boldsymbol{b}\right) \qquad (3)$$

where W, U and $\boldsymbol{b}$ are parameters shared among all steps. However, vanilla RNNs suffer from vanishing gradient for large T, thus preventing the use for long sequences [8].

Gated Recurrent Units: A Gated Recurrent Unit (GRU) [16], [15] is an extension of vanilla RNNs (see Fig. 1(b) for an illustration) that does not suffer from the vanishing gradient problem. At each step t, we fist compute a candidate hidden state $\tilde{\boldsymbol{h}}_t$ as follows:

$$\begin{aligned} \boldsymbol{r}_t &= \sigma\left(W_r\boldsymbol{x}_t + U_r\boldsymbol{h}_{t-1} + \boldsymbol{b}_r\right) \\ \tilde{\boldsymbol{h}}_t &= \tanh\left(W_h\boldsymbol{x}_t + U_h\left(\boldsymbol{r}_t * \boldsymbol{h}_{t-1}\right) + \boldsymbol{b}_h\right) \end{aligned}$$

where $\boldsymbol{r}_t$ is a reset gate that controls the information flow of the previous state to the candidate hidden state. When $\boldsymbol{r}_t$ is close to $\mathbf{0}$, the previous hidden state is ignored and the candidate hidden state is reset with the current input.

GRUs then update the hidden state $\boldsymbol{h}_t$ using the same rule as in Eq. (2). The difference is in the gate function: $\boldsymbol{\alpha}_1 = \sigma\left(W_{\boldsymbol{\alpha}}\boldsymbol{x}_t + U_{\boldsymbol{\alpha}}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{\boldsymbol{\alpha}}\right)$, where current input $\boldsymbol{x}_t$ is used. The linear relationship between the two gates is assumed: $\boldsymbol{\alpha}_2 = 1-\boldsymbol{\alpha}_1$. This relationship enables the hidden state from previous step to be copied partly to the current step. Hence $\boldsymbol{h}_t$ is a linear interpolation of the candidate hidden state and all previous hidden states. This prevents the gradients from vanishing and captures longer dependencies in the input sequence.

Remark: Both Highway Networks and GRUs can be considered as simplified versions of Long Short-Term Memory [11]. With the linear interpolation between consecutive states, the GRUs have less parameters. Empirical experiments revealed that GRUs are comparable to LSTM and more efficient in training [17].

C. *p-norm Gates*

As described in the two sections above, the gates of the non-linear and linear parts in both Highway Networks (the version empirically validated in [13]) and GRUs use the same linear constraint:

$$\boldsymbol{\alpha}_1 + \boldsymbol{\alpha}_2 = \mathbf{1}, \quad \text{s.t.} \quad \boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \in (0,1)^k$$

where $\boldsymbol{\alpha}_1$ plays the updating role and $\boldsymbol{\alpha}_2$ plays the forgetting role in the computational sequence. Since the relationship is linear, when $\boldsymbol{\alpha}_1$ gets closer to $\mathbf{1}$, $\boldsymbol{\alpha}_2$ will get closer to $\mathbf{0}$ at the same rate. During learning, the gates might become more specialized and discriminative, this same-rate convergence may block the information from the lower layer passing through at a high rate. The learning speed may suffer as a result.

We propose to relax this scheme by using the following the following p-norm scheme:

$$\left(\boldsymbol{\alpha}_1^p + \boldsymbol{\alpha}_2^p\right)^{\frac{1}{p}} = \mathbf{1}, \quad \text{equivalently:} \quad \boldsymbol{\alpha}_2 = \left(\mathbf{1} - \boldsymbol{\alpha}_1^p\right)^{\frac{1}{p}} \qquad (4)$$

for $p > 0$, where the norm is applied element-wise.

The dynamics of the relationship of the two gates as a function of p is interesting. For $p > 1$ we have $\boldsymbol{\alpha}_1 + \boldsymbol{\alpha}_2 > \mathbf{1}$. This increases the amount of information passing for the linear part. To be more concrete, let $\boldsymbol{\alpha}_1 = \mathbf{0.9}$. For the linear gates relationship with $p = 1$, there is a portion of $\boldsymbol{\alpha}_2 = \mathbf{0.1}$ of old information passing through each step. But for $p = 2$, the passing portion is $\boldsymbol{\alpha}_2 = \mathbf{0.4359}$, and for $p = 5$, it is $\boldsymbol{\alpha}_2 = \mathbf{0.865}$. When $p \to \infty$, $\boldsymbol{\alpha}_2 \to \mathbf{1}$, regardless of $\boldsymbol{\alpha}_1$ as long as $\boldsymbol{\alpha}_1 < \mathbf{1}$. This is achievable since $\boldsymbol{\alpha}_1$ is often modelled as a logistic function. When $p \to \infty$, $\boldsymbol{\alpha}_2 \to \mathbf{1}$, the activation of the final hidden layer loads all the information of the past without forgetting. Note that the ImageNet winner 2015, the Residual Network [14], is a special case with $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \to \mathbf{1}$.

On the other hand, $p < 1$ implies $\boldsymbol{\alpha}_1 + \boldsymbol{\alpha}_2 < \mathbf{1}$, the linearity gates are closed at a faster rate, which may prevent information and gradient flow passing easily through layers.

III. Experiments

In this section, we empirically study the behavior of the p-norm gates in feedforward networks (in particular, Highway Networks presented in Section. II-A) and recurrent networks (Gated Recurrent Unit in Section. II-B).

A. *Vector Data with Highway Network*

We used vector-data classification tasks to evaluate the Highway Networks under p-norm gates. We used 10 hidden layers of 50 dimensions each. The models were trained using standard stochastic gradient descent (SGD) for 100 epochs with mini-batch of 20.

Datasets: We used two large UCI datasets: MiniBooNE particle identification (MiniBoo) [1] and Sensorless Drive Diagnosis (Sensorless) [2]. The first is a binary classification task where data were taken from the MiniBooNE experiment and used to classify electron neutrinos (signal) from muon neutrinos (background). The second dataset was extracted from motor current with 11 different classes. Table 1 reports the data statistics.

Table I
Datasets for testing Highway Networks.

Dataset	Dimens.	Classes	Train. set	Valid. set
MiniBoo	50	2	48,700	12,200
Sensorless	48	11	39,000	9,800

Training curves: Fig. 2 shows the training curves on training sets. The loss function is measured by negative-log likelihood. The training costs with $p = 2$ and $p = 3$ decrease and converge much faster than ones with $p = 0.8$ and $p = 1$. In the MiniBoo dataset, training with $p = 2$ and $p = 3$ only needs 20 epochs to reach 0.3 nats, while $p = 1$ needs nearly 100 epochs and $p = 0.8$ does not reach that value. The pattern is similar in the Sensorless dataset, the training loss for $p = 1$ is 0.023 after 100 epochs, while for $p = 2$ and $p = 3$, the losses reach that value after 53 and 44 epochs, respectively. The training for $p = 0.5$ was largely unsuccessful so we do not report here.

Prediction: The prediction results on the validation sets are reported in Table II. To evaluate the learning speed, we report the number of training epochs to reach a certain benchmark with different values of p. We also report the results after 100 epochs. For the MiniBoo dataset (Table II(a)), $p = 0.8$ does not reach the benchmark of 89% of F1-score, $p = 1$ needs 94

[1]https://archive.ics.uci.edu/ml/datasets/MiniBooNE+particle+identification
[2]https://archive.ics.uci.edu/ml/datasets/Dataset+for+Sensorless+Drive+Diagnosis

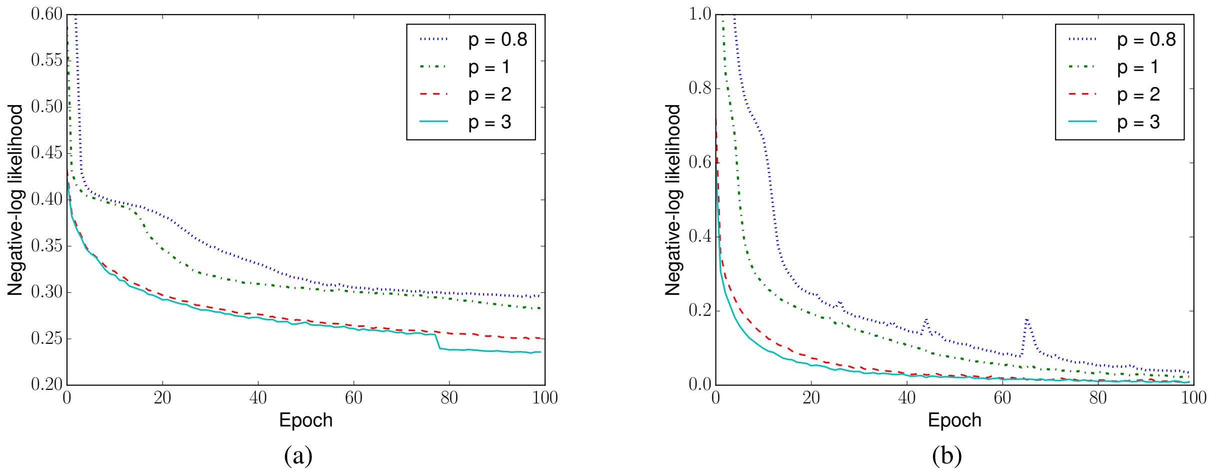


Figure 2. Learning curves on training sets. (a) MiniBoo dataset. (b) Sensorless dataset.

epochs while both $p = 2$ and $p = 3$ need 33 epochs, nearly 3 times faster. For the Sensorless dataset (Table II(b)), $p = 3$ has the best result and needs only 35 epochs to achieve 99% of macro F1-score while $p = 1$ and $p = 2$ need 77 and 41 epochs, respectively.

Visualization: Fig. 3 illustrates how 50 channels of the two gates open through 10 layers with different value of p for a randomly chosen data instance in the test set of MiniBoo. Recall from Sec. II-C that $\boldsymbol{\alpha}_1$ and $\boldsymbol{\alpha}_2$ control the amount of information in the non-linearity part and the linearity part, respectively and $\boldsymbol{\alpha}_1^p + \boldsymbol{\alpha}_2^p = \mathbf{1}$. It is clear that with the larger value of p, the more the two gates are open. Interestingly, the values of most channels in the gate $\boldsymbol{\alpha}_2$ are larger than ones in the gate $\boldsymbol{\alpha}_1$ for all values of p. The model seems to prefer the linearity part. More interestingly, there is a gradual change in gates over the layers, although gates between layers are not directly linked. At the lower layers, gates are more uniform, but get more informative near the top (which is closer to the outcome).

B. Sequential Data with GRUs

To evaluate the p-norm with GRUs, we compare the results with different values of p in the task of language modeling at character-level [12], [18], [19]. This modeling level has attracted a great interest recently due to their generalizability over language usage, a very compact state space (which equals the alphabet size, hence the learning speed), and the ability to capture sub-word structures. More specifically, given a sequence of characters $c_1, ..., c_{t-1}$, the GRU models the probability of the next character c_t: $P\left(c_t \mid c_{t-1}, ..., c_1\right)$. The quality of model is measured by *bits-per-character* in the test set which is $-\log_2 P\left(c_t \mid c_{t-1}, ..., c_1\right)$. The model is trained by maximizing the log-likelihood: $\sum_{t=1}^{T} \log P\left(c_t \mid c_{t-1}, ..., c_1\right)$.

Dataset: We used the UCI Reuters dataset which contains articles of 50 authors in online Writeprint [3]. We randomly chose 10,000 sentences for training and 4,000 sentences for validation. For sentences with length longer than 100 characters, we only used the first 100 characters. The model is trained with 400 hidden units, 50 epochs and 32 sentences each mini-batch.

Results: Fig. 4 reports (a) training curves and (b) results on validation set through epochs. It is clear from the two figures that the model with $p = 3$ performs best among the choices $p \in (0.5, 1, 2, 3)$, both in learning speed and model quality. To give a more concrete example, as indicated by the horizontal lines in Figs. 4(a,b), learning with $p = 3$ reaches 1.5 nats after 34 epoch, while learning $p = 1$ reaches that training loss after 43. For model quality on test data, model with $p = 3$ achieves 2.0 bits-per-character after 41 epochs, faster than model with $p = 1$ after 50 epochs.

C. Evaluating the Effectiveness of p

We have demonstrated in both Highway Nets and GRUs that training is faster with $p > 1$. However, we question whether the larger value of p always implies the better results and faster training? For example, as $p \to \infty$, we have $\boldsymbol{\alpha}_2 \to 1$ and the activation at the final hidden layer contains a copy of the first layer and all other candidate states: $\boldsymbol{h}_T = \boldsymbol{h}_1 + \sum_{t=2}^{T} \boldsymbol{\alpha}_t \tilde{\boldsymbol{h}}_t$. This makes the magnitude of hidden states not properly controllable in very deeper networks. To evaluate the effectiveness of p, we conducted experiments on the MiniBoo dataset with $p = 0.8, 1, 2, ..., 8$ and networks with depths of 10, 20, 30. We found that the model works very well for all values of p with 10 hidden layers. When the number of layers increase (let say 20 or 30), the model only works well with $p = 2$ and $p = 3$. This suggests that a proper control of the hidden state norms may be needed for very deep networks and widely open gates.

IV. Discussion and Conclusion

A. Discussion

Gating is a method for controlling the linearity of the functions approximated by deep networks. Another method is

[3]https://archive.ics.uci.edu/ml/datasets/Reuter_50_50

Table II

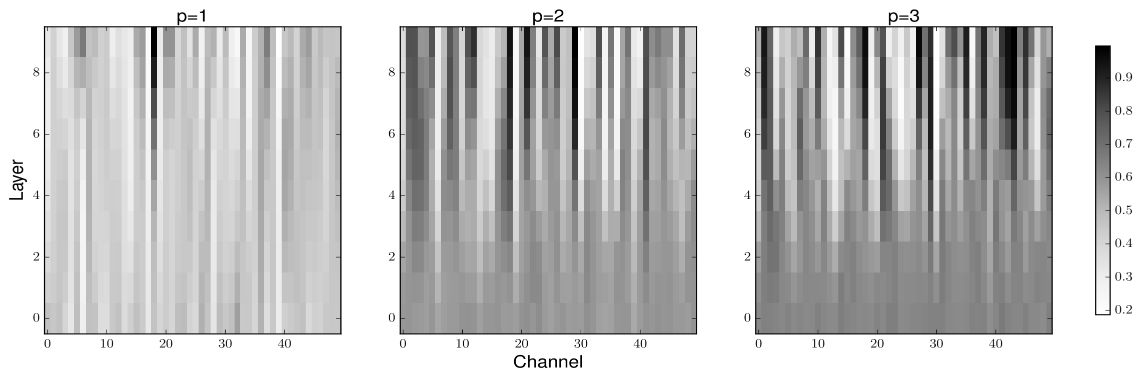
RESULTS ON VALIDATION SETS. THE SECOND COLUMN IS THE NUMBER OF EPOCHS TO REACH A BENCHMARK MEASURED BY F1-SCORE (%) FOR MINIBOO AND MACRO F1-SCORE (%) FOR SENSORLESS. THE THIRD COLUMN IS THE RESULTS AFTER RUNNING 100 EPOCHS.

(a) MiniBoo dataset

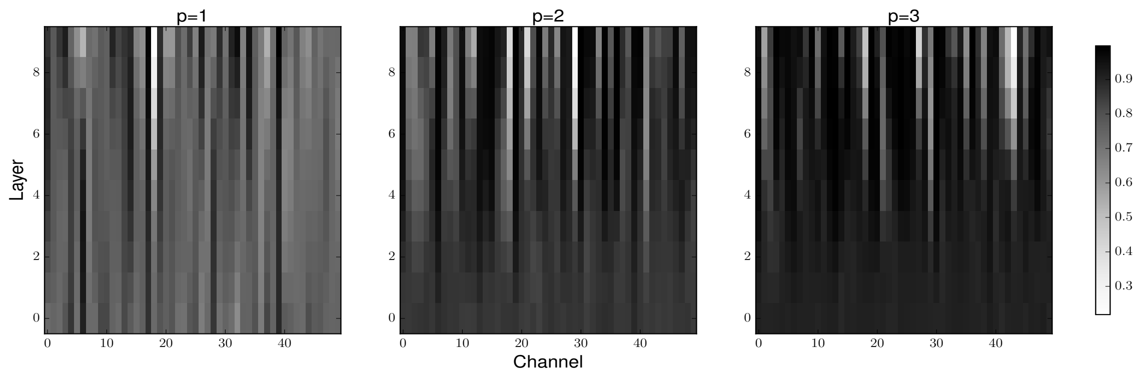
p	epochs to 89% F1-score	F1-score (%)
0.8	N/A	88.5
1	94	89.1
2	**33**	90.2
3	**33**	**90.4**

(b) Sensorless dataset

p	epochs to 99% F1-score	macro F1-score (%)
0.8	92	99.1
1	77	99.4
2	41	99.7
3	**35**	**99.7**

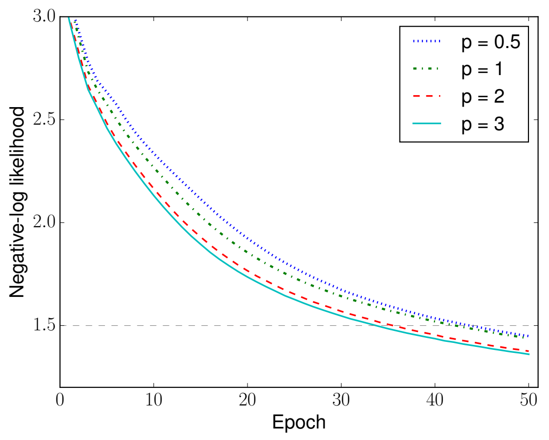


(a) The gate $\boldsymbol{\alpha}_1$

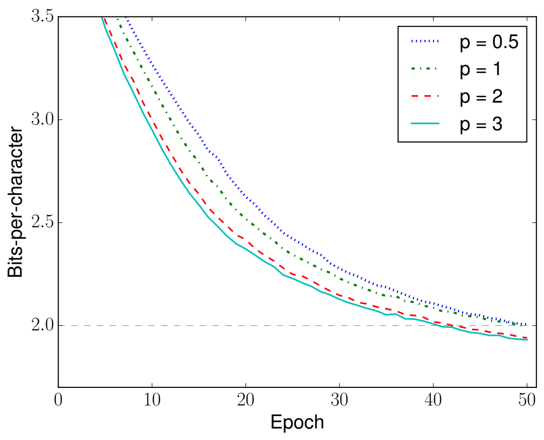


(b) The gate $\boldsymbol{\alpha}_2$

Figure 3. The dynamics of 50 channels of the two gates through 10 layers with different p.



(a)



(b)

Figure 4. (a). Learning curve on training set. (b). Results on validation set measured by Bits-per-character

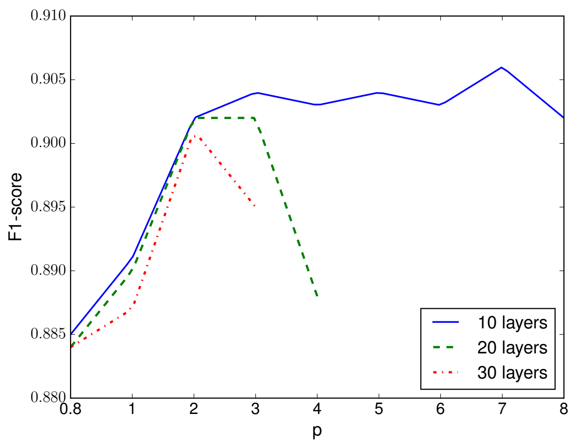


Figure 5. Results on Miniboo dataset with different p and different number of layers

to use piece-wise linear units, such as those in ReLU family [9], [1], [10]. Still, partial or piece-wise linearity has desirable nonlinearity for complex functions. At the same time it helps to prevent activation units from being saturated and the gradients from being vanishing, making gradient-based learning possible for very deep networks [13], [11]. The main idea of p-norm gates is to allow a greater flow of data signals and gradients through many computational steps. This leads to faster learning, as we have demonstrated through experiments.

It remains less clear about the dynamics of the relationship between the linearity gate $\boldsymbol{\alpha}_2$ and nonlinearity gate $\boldsymbol{\alpha}_1$. We hypothesize that, at least during the earlier stage of the learning, larger gates help to improve the credit assignment by allowing easier gradient communication from the outcome error to each unit. Since the gates are learnable, the amount of linearity in the function approximator is controlled automatically.

B. Conclusion

In this paper, we have introduced p-norm gates, a flexible gating scheme that relaxes the relationship between nonlinearity and linearity gates in state-of-the-art deep networks such as Highway Networks, Residual Networks and GRUs. The p-norm gates make the gates generally wider for larger p, and thus increase the amount of information and gradient flow passing through the networks. We have demonstrated the p-norm gates on two major settings: vector classification tasks with Highway Networks and sequence modelling with GRUs. The extensive experiments consistently demonstrated that faster learning is caused by $p > 1$.

There may be other ways to control linearity through the relationship between the linearity gate $\boldsymbol{\alpha}_2$ and nonlinearity gate $\boldsymbol{\alpha}_1$. A possible scheme could be a monotonic relationship between the two gates so as $\boldsymbol{\alpha}_1 \to 0$ then $\boldsymbol{\alpha}_2 \to 1$ and $\boldsymbol{\alpha}_1 \to 1$ then $\boldsymbol{\alpha}_2 \to 0$. It also remains open to validate this idea on LSTM memory cells, which may lead to a more compact model with less than one gate parameter set. The other open direction is to modify the internal working of gates to make them more informative [20], and to assist in regularizing the hidden states, following the findings in Sec. III-C and also in a recent work [21].

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1026–1034.

[2] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *Signal Processing Magazine, IEEE*, vol. 29, no. 6, pp. 82–97, 2012.

[3] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems*, 2014, pp. 3104–3112.

[4] A. Kumar, O. Irsoy, J. Su, J. Bradbury, R. English, B. Pierce, P. Ondruska, I. Gulrajani, and R. Socher, "Ask me anything: Dynamic memory networks for natural language processing," *arXiv preprint arXiv:1506.07285*, 2015.

[5] Y. Bengio, "Learning deep architectures for AI," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[6] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun, "The loss surfaces of multilayer networks," in *International Conference on Artificial Intelligence and Statistics*, 2015, pp. 192–204.

[7] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin, "Exploring strategies for training deep neural networks," *The Journal of Machine Learning Research*, vol. 10, pp. 1–40, 2009.

[8] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," A field guide to dynamical recurrent neural networks. IEEE Press, 2001.

[9] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier networks," in *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics. JMLR W&CP Volume*, vol. 15, 2011, pp. 315–323.

[10] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," in *Proceedings of The 30th International Conference on Machine Learning*, 2013, pp. 1319–1327.

[11] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[12] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[13] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training very deep networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 2368–2376.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[15] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[16] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *EMNLP*, 2014, pp. 1724–1734.

[17] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[18] T. Mikolov, I. Sutskever, A. Deoras, H.-S. Le, S. Kombrink, and J. Cernocky, "Subword language modeling with neural networks," *preprint (http://www. fit. vutbr. cz/imikolov/rnnlm/char. pdf)*, 2012.

[19] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush, "Character-aware neural language models," *arXiv preprint arXiv:1508.06615*, 2015.

[20] T. Pham, T. Tran, D. Phung, and S. Venkatesh, "Deepcare: A deep dynamic memory model for predictive medicine," *arXiv preprint arXiv:1602.00357*, 2016.

[21] D. Krueger and R. Memisevic, "Regularizing RNNs by Stabilizing Activations," *arXiv preprint arXiv:1511.08400*, 2015.