SYNTHETIC GRADIENT METHODS WITH VIRTUAL FORWARD-BACKWARD NETWORKS

Takeru Miyato[1,2], Daisuke Okanohara[1], Shin-ichi Maeda[3], Koyama Masanori[4]
{miyato, hillbig}@preferred.jp,
ichi@sys.i.kyoto-u.ac.jp, mkoyama@fc.ritsumei.ac.jp
[1] Preferred Networks, Inc. [2] ATR Cognitive Mechanisms Laboratories
[3] Informatics, Kyoto University [4] Mathematics, Ritsumeikan University

ABSTRACT

The concept of *synthetic gradient* introduced by Jaderberg et al. (2016) provides an avant-garde framework for asynchronous learning of neural network. Their model, however, has a weakness in its construction, because the structure of their synthetic gradient has little relation to the objective function of the target task. In this paper we introduce virtual forward-backward networks (VFBN). VFBN is a model that produces synthetic gradient whose structure is analogous to the actual gradient of the objective function. VFBN is the first of its kind that succeeds in decoupling deep networks like ResNet-110 (He et al., 2016) without compromising its performance.

1 INTRODUCTION

So far, deep neural network has been successful in winning state of the art performances on various tasks, and numerous studies including Shazeer et al. (2017); He et al. (2016) support the belief of *the deeper the better, the larger the better*. While these recent advances are increasingly enabling the usage of deep and wide models in applications, some serious hardware problems remain. For outrageously large networks, one is forced to distribute the model parameters and data over multiple machines, and the computational efficiency is at the mercy of sheer communication-speed among hardwares. Asynchronous update scheme for neural network may provide a cure for resolving such bottleneck. One such scheme is the decoupled learning of neural network based on *synthetic gradient* introduced by (Jaderberg et al., 2016). If h is an output of some hidden layer in the network, the derivative of the final objective function $\delta(h) := \partial \ell / \partial h$ with respect to h requires the parameters and the outputs of the layers above. The idea of *synthetic gradient* is to boldly approximate $\delta(h)$ by much simpler, independent function $\hat{\delta}(h)$. Quoting their terminology, one can *decouple* the sets of independent $\hat{\delta}(h)$ function associated with each hidden layer. Unfortunately, however, as we will show later in this paper, their method fails in training some important families of deep neural networks. We believe that the weakness of the gradient synthesizer model used in lies in its structure. In fact, their construction of the synthetic gradient does not guarantee any structural similarity to the original gradient function that is computed with the standard forward and backward prop.

Our virtual forward-backward network (VFBN) is a model that synthesizes a gradient with respect to the variable in intermediate layer using a virtual network that mimics the actual subnetwork above the very layer. In order to evaluate the effectiveness of our VFBN, we applied our method to the Residual Network with 110 layers. We were able to decouple the original model without significantly compromising the performance when compared to the original model trained with the back-prop. This is a feat that could not be achieved by Jaderberg et al. (2016).

2 THE METHOD OF SYNTHETIC GRADIENT

In this section, we would like to use a label classifier as an example-model to describe the synthetic gradient method proposed by Jaderberg et al. (2016). We shall however note that our concept is widely applicable to many other tasks that uses neural network. Throughout, let us denote the input

by $x \in R^I$, where I is input dimension. Also, we would use $y \in \{0,1\}^K; \sum_i y_i = 1$ to denote one-hot vector, and use $\mathcal{D}_l = \{x^{(n)}, y^{(n)} | n = 1, \ldots, N\}$ to denote a labeled dataset. We train the model $p(Y|x,\theta)$ with $\mathcal{D}$. To make the discussion compact, we would like to assume that $p(Y|x,\theta)$ is a neural network with L hidden layers. That is, putting $\theta = \{W^l\}_{l=1}^{L+1}$, we can write

$$p(Y = y_i|x,\theta) := \text{softmax}(W_i^{L+1} h_L),\ h_l := f(a_l),\ a_l := W^l h_{l-1},\ l \in \{1,\ldots,L\}. \quad (1)$$

where $f(\cdot)$ is a nonlinear function, such as ReLU (Jarrett et al., 2009; Nair & Hinton, 2010; Glorot et al., 2011). Also, softmax$(\cdot)$ indicates a softmax function on $o_i \in R$; $\exp(o_i)/\sum_{j=1}^K \exp(o_j)$. We have omitted the bias parameters in each layer for simplicity.

In usual training of the neural network, the update of the parameter in the lower layer must wait for the update of the layer above and work in *synchrony*. To train each layer asynchronously, Jaderberg et al. (2016) proposed *synthetic gradient* method that can *decouple* the training of each layer in the network, the precise meaning of which we would explain momentarily. In their work, they use

$$\delta_l \approx \hat{\delta}_l(h_l, c) \quad (2)$$

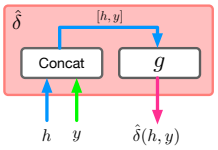
to denote the approximation of δ_l, where h_l is the output from l-th layer and c is some auxiliary contextual input available. Note that this approximator does not depend on the computations in the layers above. The approximator $\hat{\delta}_l(h_l, c)$ is trained by minimizing

$$e_l(\hat{\delta}_l) := \|\delta_l - \hat{\delta}_l\|_2^2, \quad (3)$$

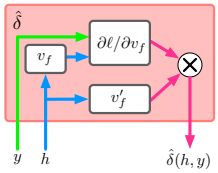
wherein δ_l can in turn be approximated as well. For the gradient synthesizer in their experiment, they used the label information y for the contextual information c, and used a model that takes the concatenated vector $[h, y]$ as the input (Figure 1a):

$$\hat{\delta}(h, y) := g([h, y]). \quad (4)$$

In our paper, we refer to this model as Jaderberg's model (see Appendix for more detail). This paper sprouted from our suspicion that this family of function does not really capture the relationship between forward and backward prop in the original network, and might fall short in its ability to approximate the gradient with high accuracy.



(a) The Jaderberg's model



(b) VFBN model

Figure 1: The models for synthesizing gradient.

3 VIRTUAL FORWARD-BACKWARD NETWORKS (VFBN)

Intuition dictates that a *good* form of $\hat{\delta}$ should somewhat trace the gradient that is computed with the forward and backward prop in standard routine. We shall there aim to synthesize a function that behaves like the forward prop and a function that behaves like the back-prop. With simple chain rule, we can see that the gradient of the original objective function ℓ with respect to a hidden variable h given a label y can be written using recursive definition of δ as:

$$\delta_l(h, y) := \frac{\partial \ell(y, fwd(h))}{\partial h} = bwd(h) \times (\partial_{fwd(h)} \ell(y, fwd(h))) \quad (5)$$

where fwd represents the forward function about h that is derived from $p(Y|x,\theta)$, and bwd represents its derivative. Replacing the fwd with *virtual* approximator v_f in eq.(5), we get our VFBN gradient synthesizer:

$$\hat{\delta}_l(h, y)_{\text{VFBN}} := v'_f(h) \times \partial_{v_f(h)} \ell(y, v_f(h)) \quad (6)$$

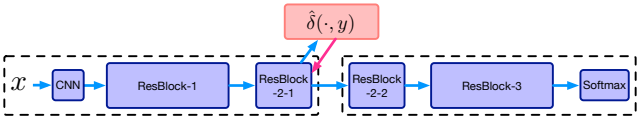


Figure 2: Decoupling of ResNet-110.

where v_f is an arbitrary function of user's choice that is to be trained during the training, and v'_f is its derivate with respect to h (Figure 1b). We can rightly refer to v_f and v'_f as *virtual forward network* and *virtual backward network*, respectively. The essence of our method is to use the approximator of this form for the gradient synthesizer. We trained v_f by minimizing Eq.(3) with respect to the relevant parameters. Structure-wise, v_f can be thought of as a simplified version of the subnetwork above the layer containing the variable h (see appendix for the the architecture of v_f we used in our experiments).

4 Experiment : Decoupling of ResNet-110 on CIFAR-10

In order to verify the efficacy of the VFBN, we tested our model to decouple the training on 110-layered Residual Networks (ResNet-110) (He et al., 2016) (see Appndix B) and compared its performance against that of Jaderberg's model[1]. Figure 2 is a graphical representation of our decoupling procedure. We partitioned the residual network into two large subnetworks consisting of 55 layers each, and decoupled the trainings on these to substructures.

Figure 3 compares the learning curve of two synthetic gradient methods against the standard training with the back prop(BP)[2]. Note that the learning curve of Jaderberg's models fall significantly behind the BP, while our VFBN keeps its pace with the BP throughout. Table 1 tabulates the test errors of the model trained with comparative methods. With 12 epochs, the test error rate of Jaderberg's Linear model, VFBN ($\alpha_{gs} = 1e$-3) and BP were respectively 15.56%, 5.51% and 5.15% .

We shall also note that, against our intuition, VFBN was able to produce a network with an error as low as 5.73% with untrained v_f ($\alpha_{gs} = 0$), which is slightly better than the result with the bottom half of the network alone. We can justify this outcome with the structure of VFBN. If the parameters of VFBN remain fixed, our VFBN keeps returning a derivative to the layers below the synthesizer so that the output of the lower layers transformed by the fixed v_f will become closely correlated with the label information. But of course, trained VFBN can capture better information than the untrained VFBN because it gains information from the layer above, and the generalization performance is in fact better with $\alpha_{gs} = 1e$-3. The result indicates that VFBN can successfully learn the gradients for deep network and draw out the potential of the deep network.

	Test error (%)
BackProp	5.15
Bottom half with back prop	5.76
Layer-wise supervised loss	5.71
(Synthetic Gradient models)	
Jaderberg's small-ResNet	20.45
Jaderberg's Linear	15.56
VFBN (ours, $\alpha_{gs} = 0$)	5.73
VFBN (ours, $\alpha_{gs} = 1e$-3)	5.51

Table 1: Test error rates on CIFAR-10

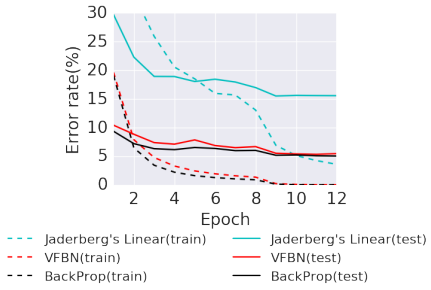


Figure 3: Learning curves on CIFAR-10

[1]We shall confess that, in our experiment, we simply replaced the back-propagated gradient with synthetic gradient in usual synchronous training, and that our result has not been tested on distributional environment.

[2]We augmented the dataset prior to the experiment ($50K$ training images$\rightarrow$ $1.8M$ training images) and each epoch in our experiment corresponds to the learning with the entire augmented dataset. As such, our 1 epoch corresponds to 36 epochs ($= 1.8M/50K$) in the standard study.

ACKNOWLEDGMENTS

We would like to thank the members of Preferred Networks, Inc., particularly Seiya Tokui, for insightful advices and comments.

REFERENCES

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *AISTATS*, 2011.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. *arXiv preprint arXiv:1608.05343*, 2016.

Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *ICCV*, 2009.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. Chainer: a next-generation open source framework for deep learning. In *Proceedings of workshop on machine learning systems (LearningSys) in the twenty-ninth annual conference on neural information processing systems (NIPS)*, 2015.

A TWO VERSIONS OF THE GRADIENT SYNTHESIZER

A.1 JADERBERG'S MODEL

In their experiment, Jaderberg et al used *Linear model*, given by:

$$\hat{\delta}(h, y) := W[h, y] \tag{7}$$

where [a, b] indicates concatenation of the two vectors a and b.

A.2 VFBN MODEL

If we use the log-likelihood for the objective function, we may for instance use linear v_f, in which case we obtain

$$\hat{\delta}(h, y) := W_v^{\mathrm{T}}(y - softmax(W_v h)) \tag{8}$$

where W_f is the weight matrix on v_f. If we use one hidden layer neural network for v_f, our VFBN becomes

$$\begin{aligned}\hat{\delta}(h, y) :=& W_v^{1\mathrm{T}} f'(W_v^1 h) W_v^{2\mathrm{T}} \\ & \times (y - softmax(W_v^2 f(W_v^1 h)))\end{aligned} \tag{9}$$

B DETAILS OF EXPERIMENTS

B.1 LEARNING WITH RESNET-110 ON CIFAR-10

All experiments used Chainer (Tokui et al., 2015) on GPU. CIFAR-10 is a well known benchmark dataset for the image recognition tasks. For the experiments in this paper, we used augmented CIFAR-10 dataset consisting of (1) 24×24 dimensional images randomly cropped from the original 32×32 dimensional images, (2) 24×24 dimensional images constructed by compressing 28×28 dimensional images cropped from the original images and (3) making left-right reflection of the output. The whole augmented dataset consists of $1,800,000$ images as opposed to the original $50,000$. For the evaluation of test error, we produced augmented dataset for each test sample, fed the resulting augmented dataset to the model and averaged the output. For the original model on which to apply our VFBN, we used essentially the same network as the ResNet-110 used in He et al. (2016) for the task on CIFAR-10.

Figure 2 shows the architecture of our version of ResNet-110. In this network, $24 \times 24 \times 3$ dimensional input x(three channels) is first transformed to $22 \times 22 \times 16$ dimensional input with convolution without padding. The output is then fed to ResBlock-1,2,3 consisting of 36 convolutional layers, each using resolution preserving padding with 3×3 kernel width. Skip connections occur at every 2 layers in this network. The resolutions and the number of feature maps at these three blocks are 22×22, 11×11, 6×6 and $32, 64, 64$, respectively. We are using the stride of size 1 everywhere, with exceptions on the connections over which the resolution changes, wherein we use the stride of size 2. Each convolution layers is followed by Batch Normalization layer (Ioffe & Szegedy, 2015) and ReLU activation. The global average pooling is conducted at ResBlock-3, and the outcome is fed to softmax layer, producing the final probability output. For further details, please consult the actual source code provided at `https://github.com/mitmul/chainer-cifar10/`.

For the experiment with synthetic gradient, we constructed gradient synthesizer right at the middle of ResBlock-2.

For our experiment with the Jaderberg's model (Jaderberg et al., 2016), we followed their procedure and fed a concatenation of the intermediate output h and one hot vector y to their synthesizer. Output of the layer at which we attached the synthetic gradient network consists of 64 feature maps with 11×11 resolution. Thus, the concatenated input $[h, y]$ to be fed to the synthesizer consists of 74 feature maps with 11×11 resolution. We experimented on Jaderberg's model with two families of g (Eq.(4)). We first tested their model with linear g (Jaderberg's Linear in Table 1) representing one layer convoltion of 5×5 kernel width. We also tested their model with g being a ResNet (Jaderberg's small-ResNet in Table 1). We would articulate on the architecture of this ResNet synthesizer.

Our version of Jaderberg's ResNet synthesizer first transforms $[h, y]$ into 64 feature maps via convolution layer with resolution preserving padding with 3×3 kernel width. The network then feeds the output to 4 convolution layers using resolution preserving padding with 3×3 kernel width. For the latter 4 layers, the skip connection occurs every 2 layers. The final output of the synthesizer is produced by feeding the output again to the last layer of convolution using resolution preserving padding with 3×3 kernel width. Except for the last layer, every convolution in the synthesizer network is followed by batch normalization layer and ReLU.

Finally, we would describe the architecture of v_f in our VFBN, which is a ResNet on its own. Our v_f first feeds the input to 4 convolutional layers using resolution preserving padding with kernel width 3×3, admitting 2 skip connection at each layer. Our v_f then mimics the original network and applies global average pooling and softmax transformation in the end. As in the original network, each convolution in v_f is also followed by BatchNormalization Layer and ReLU.