Towards More Biologically Plausible Deep Learning and Visual Processing

by

Qianli Liao

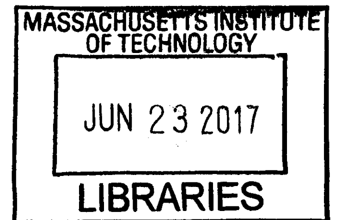
Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2017



ARCHIVES

© Qianli Liao, MMXVII. All rights reserved.

The author hereby grants to MIT permission to reproduce and to distribute publicly paper and electronic copies of this thesis document in whole or in part in any medium now known or hereafter created.

Signature redacted

Author ....
Department of Electrical Engineering and Computer Science
May 19, 2017

Signature redacted

Certified by.. ....
Tomaso A. Poggio
Eugene McDermott Professor in the Department of Brain and Cognitive Sciences and Principal Investigator of McGovern Institute for Brain Research and Computer Science and Artificial Intelligence Laboratory
Thesis Supervisor

Signature redacted

Accepted by ... .
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Towards More Biologically Plausible Deep Learning and Visual Processing

by

Qianli Liao

Submitted to the Department of Electrical Engineering and Computer Science
on May 19, 2017, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

Over the last decade, we have witnessed tremendous successes of Artificial Neural Networks (ANNs) on solving a wide range of AI tasks. However, there is considerably less development in understanding the biological neural networks in primate cortex. In this thesis, I try to bridge the gap between artificial and biological neural networks. I argue that it would be beneficial to build ANNs that are both biologically-plausible and well-performing, since they may serve as models for the brain and guide neuroscience research. On the other hand, developing a biology-compatible framework for ANNs makes it possible to borrow ideas from neuroscience to improve the performance of AI systems. I discuss several aspects of modern ANNs that can be made more consistent with biology: (1) the backpropagation learning algorithm (2) ultra-deep neural networks (e.g., ResNet, He et al., 2016) for visual processing (3) Batch Normalization (Ioffe and Szegedy, 2015). For each of the three aspects, I propose biologically-plausible modifications of the ANN models to make them more implementable by the brain while maintaining (or even improving) their performance.

Thesis Supervisor: Tomaso A. Poggio
Title: Eugene McDermott Professor in the Department of Brain and Cognitive Sciences and Principal Investigator of McGovern Institute for Brain Research and Computer Science and Artificial Intelligence Laboratory

Acknowledgments

I would like to thank my family, supervisor, co-authors, colleagues and friends for their support on this thesis.

Contents

1 Introduction **23**

2 Biologically Plausible Backpropagation **25**

2.1 Introduction 25

2.2 Asymmetric Backpropagations 27

2.3 Normalizations/stabilizations are necessary for "asymmetric" backpropagations 29

2.4 Related Work 31

2.5 Experiments 33

2.5.1 Method 33

2.5.2 Datasets 33

2.5.3 Training Details 34

2.5.4 Results 35

2.6 Discussion 38

3 A Biologically-Plausible Framework of Residual Learning, Recurrent Neural Networks and Visual Cortex **41**

3.1 Introduction 41

3.2 Equivalence of ResNet and a specific RNN 43

3.2.1 Intuition 43

3.2.2 Formulation in terms of Dynamical Systems 43

3.3 A Generalized RNN for Multi-stage Fully Recurrent Processing . . . 45

3.4 Related Work 49

3.5 Experiments 50
3.6 Discussion 53
3.7 Future Directions 54

4 Biologically-Plausible Normalizations For Recurrent and Online Learning 55

4.1 Introduction 55
4.2 Online and Batch Learning with "Decoupled Accumulation and Update" 57
4.3 A General Framework for Normalization 58
4.3.1 Sample Normalization 59
4.3.2 General Batch Normalization 61
4.3.3 Streaming Normalization 62
4.3.4 Lp Normalization: Calculating NormStats with Different Orders of Moments 66
4.3.5 Separate Learnable Bias and Gain Parameters 67
4.4 Generalization to Recurrent Learning 67
4.4.1 Recurrent Sample Normalization 68
4.4.2 Recurrent General Batch Normalization (RGBN) 68
4.4.3 Recurrent Streaming Normalization 68
4.5 Streaming Normalized RNN and GRU 69
4.6 Related Work 69
4.7 Experiments 70
4.7.1 CIFAR-10 architectures and Settings 70
4.7.2 Lp Normalization 71
4.7.3 Online Learning or Learning with Very Small Mini-batches . . 72
4.7.4 Evaluating Variants of Batch Normalization 73
4.7.5 More Experiments on Streaming Normalization 76
4.7.6 Recurrent Neural Networks for Character-level Language Modeling 76
4.8 Discussion 80
4.9 Appendix: Other Variants of Streaming Normalization 82

5 Conclusions and Future Work **83**

List of Figures

2-1 A simple illustration of backpropagation 28

3-1 A formal equivalence of a ResNet (A) with weight sharing and a RNN (B). I is the identity operator. K is an operator denoting the nonlinear transformation called f in the main text. x_t is the value of the input at time t. δ_t is a Kronecker delta function. 44

3-2 (A) Modeling the ventral stream of visual cortex using a multi-state fully recurrent neural network. (B) the model consists of a input, a recurrent part and a output. (C) Examples of the recurrent part of the model and corresponding transition matrices used in this work. "BN" denotes Batch Normalization and "Conv" denotes convolution. Deconvolution layer (denoted by "Deconv") is [88] used as a transition function from a spacially small state to a spacially large one. BRCx2/BRDx2 denotes a BN-ReLU-Conv/Deconv-BN-ReLU-Conv/Deconv pipeline (similar to a residual module [26]). There is always a 2x2 subsampling/upsampling between nearby states (e.g., V1/h1: 32x32, V2/h2: 16x16, V4/h3:8x8, IT:4x4). Stride 2 (convolution) or upsampling 2 (deconvolution) is used in transition functions to match the spacial sizes of input and output states. The intermediate feature sizes of transition function BRCx2/BRDx2 or BRCx3/BRDx3 are chosen to be the average feature size of input and output states. "+I" denotes a identity shortcut mapping. 46

3-3 The performance of 4-state and 3-state models. The state sizes of the 4-state model are: 32x32x8, 16x16x16, 8x8x32, 4x4x64. The state sizes of the 3-state model are: 32x32x64, 16x16x128, 8x8x256. Only small 4-state models were tried since they are very computationally heavy. The readout time is t=5 for both models. All models are time-invariant systems (i.e., weights are shared across time). 47

3-4 The transition matrices of all models are shown in Figure 3-2C. "#Param" denotes the number of parameters. For each model, the effective unrolling factors "xN" are determined by the readout time t (See Section 3.3 for the definition). Dashed lines are training errors. For multi-state ResNet, a downstream state only executes after receiving inputs from upstream states. For fully recurrent nets, all transitions execute concurrently. The state sizes are shown above. (A) A 2-state ResNet with the second state only unrolled once. This architecture works better with shared rather than non-shared weights on CIFAR-10. In general, we empirically observe that weight sharing in the 1st state of a multi-state system is often beneficial. One conjecture is that our transition function might match better low-level visual recurrent computations. (B) Standard 3-state ResNet. (C) 2-state fully recurrent NN. (D) Different readout time of (A). (E) A 2-state fully recurrent network with different readout time. There is consistent performance improvement as t increases. The number of parameters changes since at some t, some recurrent connections have not been contributing to the output and thus their number of parameters are subtracted from the total. We observe in (D) and (E) that error reduces as t increases, while #param. is kept the same. 51

3-5 (A) Training and testing with different readout time. A 2-state ResNet with shared weights is trained with readout time t=13 (i.e., 1st state unrolling factor N=t-3=10) and tested with t from 7 to 23. The model can generalize well to readout time that are not trained with. There even seems to be tiny performance improvement (on test set) when testing with slightly larger readout time than training. (B) Biological Plausibility of Time-specific Batch Normalization (TSBN): we show the normalization statistics of a batch normalization module in the model in part (B) (which was trained with t=23). Each line represents the statistics of a feature channel over time. TSBN seems to implement a simple decay, thus it may be biologically-plausible. 53

3-6 ImageNet experiment with a 4-state ResNet with non-shared vs. shared weights. Train: training error. Val: validation error. No scaling augmentation is done. Performances are all based on single-crop. We trained the models with 12 epochs and logarithmically decaying learning rates (i.e., logspace(-1, -4, N) in Matlab, where N=12 is the number of epochs). The logarithmic learning rates make the model converge much faster than the usual three stage learning rates (i.e., dividing by 10 every some epochs), thus the models have already converged well and the performance differences are representative. 53

4-1 A General Framework of Normalization. A: the input to convolutional layer is a 3D matrix consists of 3 dimensions: x (image width), y (image height) and features/channel. For fully-connected layers, x=y=1. B: training with decoupled accumulation and update. C: Sample Normalization. D: General Batch Normalization. E: Streaming Normalization 60

4-3 Lp Normalization. The architecture is a **feedforward and convolutional** network (shown in Figure 4-2 B). All statistical moments perform similarly well. L7 normalization is slightly worse. 71

4-2 Architectures for CIFAR-10. Note that C reduces to B when $k_1 = k_2 = 1$. 71

4-4 Plain Mini-batch vs. Decoupled Accumulation and Update (DAU). The architecture is a **feedforward and fully-connected** network (shown in Figure 4-2 A). S/B: Samples per Batch. B/U: Batches per Weight Update. We show there are significant performance differences between plain mini-batch (i.e., B/U=1) and Decoupled Accumulation and Update (DAU, i.e., B/U=n>1). DAU significantly improves the performance of BN with small number of samples per mini-batch (e.g., compare curve 1 with 3). 72

4-5 Different normalizations applied to a **feedforward and fully-connected** network (shown in Figure 4-2 A). The right two pannels are **zoomed-in versions** of the left two pannels. S/B: Samples per Batch. B/U: Batches per Weight Update. "Ours" refers to Streaming Normalization with "L1 norm" (Setting B with p=1 in Section 4.3.4) and $\alpha_1 = \beta_1 = 0.7$, $\alpha_2 = \beta_2 = 0.3$ and $\beta_3 = 0$ (see Section 4.3.3 for more details about hyperparameters). We show that our algorithm works with pure online learning (1 S/B) and tiny mini-batch (2 S/B), and it outperforms Layer Normalization. The choice of S/B does not matter for layer normalization since it processes samples independently. 73

4-6 Different normalizations applied to a **feedforward and convolutional** network (shown in Figure 4-2 B). All models were trained with 32 Samples per Batch (S/B), 1 Batch per Update (B/U). "Our approach" refers to Streaming Normalization with "L2 norm" (Setting A with p=2 in Section 4.3.4) and $\alpha_1 = \beta_1 = 0.5$, $\alpha_2 = \beta_2 = 0.5$ and $\beta_3 = 0$ (see Section 4.3.3 for more details about hyperparameters). LN: Layer Normalization. Sample Normalizations (including LN) seem to all work similarly. It seems beneficial to normalize each channel/feature map separately (e.g., compare BA3 with BA4), like what BN does. 74

4-7 Different normalizations applied to a **recurrent and convolutional** network (Figure 4-2 C with $k_1 = 5$ and $k_2 = 1$). All models were trained with 32 Samples per Batch (S/B), 1 Batch per Update (B/U). "Our approach" refers to Streaming Normalization with "L2 norm" (Setting A with p=2 in Section 4.3.4) and $\alpha_1 = \beta_1 = 0.5$, $\alpha_2 = \beta_2 = 0.5$ and $\beta_3 = 0$ (see Section 4.3.3 for more details about hyperparameters). LN: Layer Normalization. Sample Normalizations (including LN) seem to all work similarly. It seems beneficial to normalize each channel/feature map separately (e.g., compare BA3 with BA4), like what BN does. . 75

4-8 Time-specific Batch Normalization (TSBN) and Streaming Normalization applied to a **densely recurrent and convolutional** network (Figure 4-2 D with $k = 5$). "Ours" refers to Streaming Normalization with "L2 norm" (Setting B with p=2 in Section 4.3.4), 32 Samples per Batch (S/B) and 2 Batches per Update (B/U) and $\alpha_1 = \beta_1 = 0.7$, $\alpha_2 = 0.3, \beta_2 = 0$ and $\beta_3 = 0.3$ (see Section 4.3.3 for more details about hyperparameters). Sometimes for recurrent networks, B/U > 1 is preferred, since the first mini-batch collects NormStats from all timesteps so that the second mini-batch is normalized in a more stable way. TSBN was trained with 64 S/B, 1 B/U (32 S/B, 2 B/U would give similar performance, if not worse). Streaming Normalization has similar performance to TSBN but does not require storing different NormStats for each timestep. 76

4-9 Different normalizations applied to a **recurrent and convolutional** network (shown in Figure 4-2 C with unrolling parameters $k_1 = k_2 = 5$). The right two pannels are **zoomed-in versions** of the left two pannels. "Ours" refers to Streaming Normalization with "L2 norm" (Setting B with p=2 in Section 4.3.4), 32 Samples per Batch (S/B), 2 Batches per Update (B/U) and $\alpha_1 = \beta_1 = 0.7$, $\alpha_2 = 0.3, \beta_2 = 0$ and $\beta_3 = 0.3$ (see Section 4.3.3 for more details about hyperparameters). Time-specific Batch Normalization, original BN and Layer Normalization (LN) were trained with 64 S/B, 1 B/U (32 S/B, 2 B/U would give similar performance, if not worse). Streaming Normalization clearly outperforms other methods in training. Streaming Normalization converges **more than twice** as fast as LN. Note that 32 S/B 2 B/U and 64 S/B 1 B/U are equivalent to LN since it processes samples independently. Original BN fails on testing. 77

4-10 Evaluate different choices of hyperparameter β_1,β_2 and β_3. The architecture is a **recurrent and convolutional** network (shown in Figure 4-2 C with unrolling parameters $k_1 = k_2 = 5$). The right two pannels are **zoomed-in versions** of the left two pannels. The models are Streaming Normalization with "L2 norm" (Setting B with p=2 in Section 4.3.4), 32 Samples per Batch (S/B), 2 Batches per Update (B/U) and α_1, $\alpha_2 = 0.3, \kappa_1 = \kappa_3 = 0.7, \kappa_2 = \kappa_4 = 0.3$. The hyperparameters $(\beta_1, \beta_2, \beta_3)$ are shown in the figure. (0,0,0) means that the gradients of NormStats are ignored. (0,0,1) means only using NormStats gradients from the current mini-batch. (0,1,0) means only using NormStats gradients accumulated since the last weight update. Note that regardless the values of β, the gradients of NormStats are always accumulated (See Section 4.3.3). Using gradients from the previous weight update (i.e., 1,0,0) seems to work reasonably well. Some combinations (i.e., (0.7,0.0.3) or (0.7,0.0.3)) of previous and current gradients seem to give the best performances. This experiment indicates that streaming the gradients of NormStats is very important for performance. 78

4-11 Character-level language modeling with RNN on Shakespeare's work concatenated. The training (left) and validation (right) softmax losses are reported. "Ours" refers to Streaming Normalization with "L2 norm" (Setting B with p=2 in Section 4.3.4), 32 Samples per Batch (S/B), 2 Batches per Update (B/U) and $\alpha_1 = \beta_1 = 0.7$, $\alpha_2 = 0.3, \beta_2 = 0$ and $\beta_3 = 0.3$ (see Section 4.3.3 for more details about hyperparameters). TSBN: time-specific BN. LN: Layer Normalization. Both TSBN and Streaming Normalization (SN) converges faster than LN. SN reaches slightly lower loss than TSBN and LN. 79

4-12 Character-level language modeling with GRU on Shakespeare's work concatenated. The training (left) and validation (right) softmax losses are reported. "Ours" refers to Streaming Normalization with "L2 norm" (Setting B with p=2 in Section 4.3.4), 32 Samples per Batch (S/B), 2 Batches per Update (B/U) and $\alpha_1 = \beta_1 = 0.7$, $\alpha_2 = 0.3, \beta_2 = 0$ and $\beta_3 = 0.3$ (see Section 4.3.3 for more details about hyperparameters). TSBN: time-specific BN. LN: Layer Normalization. Streaming Normalization converges faster than LN and reaches lower loss than TSBN. 80

A1 We explore other variants of Streaming Normalization with different NormRef (e.g., SP1-SP5, BA1-BA6 in 4-1 C and D) within each minibatch. **-B** denotes the batch version. **-S** denotes the streaming version. The architecture is a **feedforward and convolutional** network (shown in Figure 4-2 B). Streaming normalization lowers training errors. . . 82

List of Tables

2.1 Network architectures used in the experiments: AxBxC/D means C feature maps of size AxB, with stride D. The CIFAR10&100 architecture has a 2 units zero-padding for every convolution layer and 1 unit right-bottom zero-padding for every pooling layer. The other models do not have paddings. "FC X" denotes Fully Connected layer of X feature maps. In the first model, the number of hidden units in FC layers are chosen according to the number of classes (denoted by "#Class") in the classification task. " max(256,#Class*3)" denotes 256 or #Class*3, whichever is larger. Rectified linear units (ReLU) are used as nonlinearities for all models. 32

2.2 **Experiment A:** The magnitudes of feedbacks do not matter. Sign concordant feedbacks can produce strong performance. Numbers are error rates (%). Yellow: performances worse than baseline(SGD) by 3% or more. **Blue**: performances better than baseline(SGD) by 3% or more. 34

2.3 **Experiment B Part 1 (left):** Feedbacks have random magnitudes, varing probability of having different signs (percentages in second row, column 3-7) from the feedforward ones. The M and S redrawn in each mini-batch. Numbers are error rates (%). Yellow: performances worse than baseline(SGD) by 3% or more. **Blue**: performances better than baseline(SGD) by 3% or more. **Experiment B Part 2 (right):** Same as part 1, but The M and S were fixed throughout each experiment. 36

2.4 **Experiment C1:** fixed random feedbacks. **Experiment C2:** (.)*Bottom*: The model's last layer is initialized randomly and clamped/frozen. All learning happens in the layers before the last layer. (.) *Top*: The model's layers before the last layer are initialized randomly and clamped/frozen. All learning happens in the last layer. Numbers are error rates (%). Yellow: performances worse than baseline(SGD) by 3% or more. **Blue**: performances better than baseline(SGD) by 3% or more. 36

2.5 Different settings of Batch Manhattan (as described in Section 2.3) seem to give similar performances. SGD: setting 0, BM1: setting 1, BM2: setting 2, BM3: setting 3. The interaction of BM with sign concordant feedback weights (uSF) and Batch Normalization are shown in "uSF+BN+(.)" entries. Numbers are error rates (%). Yellow: performances worse than baseline (SGD) by 3% or more. **Blue**: performances better than baseline(SGD) by 3% or more. 37

3.1 Compare our model (FRNN) with other best models (prior to our work) on CIFAR-10. All models were trained with simple translation and mirror augmentation. The depth is the number of weighted linear combination layers and pooling layers. All of our models were trained with 60 epochs for consistency, since we did not focus on the absolute performance. We expect better performance if more epochs are used (will report in the next revision). Nevertheless, these models are all at the level of human performance. We believe at this point biological consistency is more interesting than marginal performance differences. Also our models achieve high performance with small latency (i.e., depth), which supports rapid visual recognition and is crucial for organism's survival. 47

4.1 An overview of normalization techiques for different tasks. ✓: works well. ✗: does not work well. FF: Feedforward. Rec: Recurrent. FC: Fully-connected. Conv: convolutional. Limited: time-specific BN requires recording normalization statistics for each timestep and thus may not generalize to novel sequence length. *Layer normalization does not fail on these tasks but perform significantly worse than the best approaches. 56

Chapter 1

Introduction

During the last several years we saw a resurgence of interest in Artificial Neural Networks (ANNs). The big data and the unprecedented growth of computational power (mainly due to GPUs) has enabled new designs and applications of these classical models. ANNs nowadays are orders of magnitude larger than those in 1990s. In particular, there is a trend of increasing the number of layers in ANNs, leading to Deep Neural Networks (DNNs) and the field "Deep Learning". DNNs have achieved remarkable performance in many domains [39, 1, 27, 55, 80, 21, 74]. Interestingly, recent state-of-the-art models for visual processing are actually "ultra-deep" networks [24, 26] with hundreds to a thousand layers.

Nevertheless, the success of ANNs and Deep Learning does not immediately deliver better understanding or modeling of biological neural networks and the brain. I argue that the "biological plausibility" of DNN models and algorithms is an important but under-investigated question. In particular, increasingly deep designs of neural networks pose several questions regarding biological plausibility:

- First, How to train efficiently a deep neural network in a biologically-plausible manner? Traditional training algorithm backpropagation has several biologically unrealistic requirements.

- Second, Is "ultra-deep" networks [24, 26] biologically-plausible? One motivation for the field "Deep Learning" is that the brain is organized in a hierarchical way.

However, the "ultra-deep" networks for visual processing are one to two orders of magnitude deeper than the ventral stream of visual cortex.

- Third, activation normalization algorithms (e.g., Batch Normalization [35]) are crucial for training deep networks. Yet the current algorithms cannot handle biologically realistic training settings such as recurrent learning and online learning. If biological neural normalizations exist in the brain, how should them support these training scenarios?

Each of the following three chapters discusses one of the above questions, where novel biologically-plausible models are proposed to solve corresponding problems.

Chapter 2

Biologically Plausible Backpropagation

2.1 Introduction

Deep Neural Networks (DNNs) have achieved remarkable performance in many domains [39, 1, 27, 55, 80, 21, 74]. The simple gradient backpropagation (BP) algorithm has been the essential "learning engine" powering most of this work.

Deep neural networks are universal function approximators [31]. Thus it is not surprising that solutions to real-world problems exist within their configuration space. Rather, the real surprise is that such configurations can actually be discovered by gradient backpropagation.

The human brain may also be some form of DNN. Since BP is the most effective known method of adapting DNN parameters to large datasets, it becomes a priority to answer: could the brain somehow be implementing BP? Or some approximation to it?

For most of the past three decades since the invention of BP, it was generally believed that it could not be implemented by the brain [12, 52, 61, 8, 5]. BP seems to have three biologically implausible requirements: (1) feedback weights must be the same as feedforward weights (2) forward and backward passes require different computations, and (3) error gradients must somehow be stored separately from activations.

One biologically plausible way to satisfy requirements (2) and (3) is to posit a distinct "error network" with the same topology as the main (forward) network but used only for backpropagation of error signals. The main problem with such a model is that it makes requirement (1) implausible. There is no known biological way for the error network to know precisely the weights of the original network. This is known as the "weight transport problem" [23]. In this work we call it the "weight symmetry problem". It is arguably the crux of BP's biological implausibility.

In this chapter, we systematically relax BP's weight symmetry requirement by manipulating the feedback weights. We find that some natural and biologically plausible schemes along these lines lead to exploding or vanishing gradients and render learning impossible. However, useful learning is restored if a simple and indeed *more* biologically plausible rule called Batch Manhattan (BM) is used to compute the weight updates. Another technique, called Batch Normalization (BN) [35], is also shown effective. When combined together, these two techniques seem complementary and significantly improve the performance of our asymmetric version of backpropagation.

The results are somewhat surprising: if the aforementioned BM and/or BN operations are applied, the magnitudes of feedback weights turn out not to be important. A much-relaxed *sign-concordance* property is all that is needed to attain comparable performance to mini-batch SGD on a large number of tasks.

Furthermore, we tried going beyond sign concordant feedback. We systematically reduced the probability of feedforward and feedback weights having the same sign (the *sign concordance probability*). We found that the effectiveness of backpropagation is strongly dependent on high sign concordance probability. That said, completely random and fixed feedback still outperforms chance e.g., as in the recent work of Lillicrap et al. [51].

Our results demonstrate that the perfect forward-backward weight symmetry requirement of backpropagation can be significantly relaxed and strong performance can still be achieved. To summarize, we have the following conclusions:

(I) The magnitudes of feedback weights do not matter to performance. This surprising result suggests that our theoretical understanding of why backpropagation

works may be far from complete.

(II) Magnitudes of the weight updates also do not matter.

(III) Normalization / stabilization methods such as Batch Normalization and Batch Manhattan are necessary for these asymmetric backpropagation algorithms to work. Note that this result was missed by previous work on random feedback weights [51].

(IV) Asymmetric backpropagation algorithms evade the weight transport problem. Thus it is plausible that the brain could implement them.

(V) These results indicate that sign-concordance is very important for achieving strong performance. However, even fixed random feedback weights with Batch Normalization significantly outperforms chance. This is intriguing and motivates further research.

(VI) Additionally, we find Batch Manhattan to be a very simple but useful technique in general. When used with Batch Normalization, it often improves the performance. This is especially true for smaller training sets.

2.2 Asymmetric Backpropagations

A schematic representation of backpropagation is shown in Fig. 2-1. Let E be the objective function. Let W and V denote the feedforward and feedback weight matrices respectively. Let X denote the inputs and Y the outputs. W_{ij} and V_{ij} are the feedforward and feedback connections between the j-th output Y_j and the i-th input X_i, respectively. $f(.)$ and $f'(.)$ are the transfer function and its derivative. Let the derivative of the i-th input with respect to the objective function be $\frac{\partial E}{\partial X_i}$, the formulations of forward and back propagation are as follows:

$$Y_j = f(N_j), where\ N_j = \sum_i W_{ij} X_i \tag{2.1}$$

$$\frac{\partial E}{\partial X_i} = \sum_j V_{ij} f'(N_j) \frac{\partial E}{\partial Y_j} \tag{2.2}$$

The standard BP algorithm requires $V = W$. We call that case *symmetric backpropagation.* In this work we systematically explore the case of *asymmetric*

backpropagation where $V \neq W$.

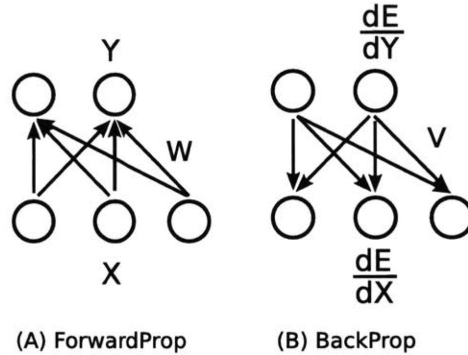


Figure 2-1: A simple illustration of backpropagation

By varying V, one can test various asymmetric BPs. Let $sign()$ denote the function that takes the sign (-1 or 1) of each element. Let $\circ$ indicate element-wise multiplication. M, S are matrices of the same size as W. M is a matrix of uniform random numbers $\in [0, 1]$ and S_p is a matrix where each element is either 1 with probability $1 - p$ or -1 with probability p. We explored the following choices of feedback weights V in this work:

1. **Uniform Sign-concordant Feedbacks (uSF)**:
$V = sign(W)$

2. **Batchwise Random Magnitude Sign-concordant Feedbacks (brSF)**:
$V = M \circ sign(W)$, where M is redrawn after each update of W (i.e., each mini-batch).

3. **Fixed Random Magnitude Sign-concordant Feedbacks (frSF)**:
$V = M \circ sign(W)$, where M is initialized once and fixed throughout each experiment.

4. **Batchwise Random Magnitude p-percent-sign-concordant Feedbacks (brSF-p)**:
$V = M \circ sign(W) \circ S_p$, where M and S_p is redrawn after each update of W (i.e., each mini-batch).

5. **Fixed Random Magnitude p-percent-sign-concordant Feedbacks (frSF-p)**:
$V = M \circ sign(W) \circ S_p$, where M and S_p is initialized once and fixed throughout each experiment.

6. **Fixed Random Feedbacks (RndF)**:

Each feedback weight is drawn from a zero-mean gaussian distribution and fixed throughout each experiment: $V \sim \mathcal{N}(0, \sigma^2)$, where σ was chosen to be 0.05 in all experiments.

The results are summarized in the Section 2.5. The performances of 1, 2 and 3, which we call **strict sign-concordance** cases, are shown in Experiment A. The performances of 4 and 5 with different choices of p, which we call **partial sign-concordance** cases, are shown in Experiment B. The performances and control experiments about setting 6, which we call **no concordance** cases, are shown in Experiments C1 and C2.

2.3 Normalizations/stabilizations are necessary for "asymmetric" backpropagations

Batch Normalization (BN)

Batch Normalization (BN) is a recent technique proposed by [35] to reduce "internal covariate shift" [35]. The technique consists of element-wise normalization to zero mean and unit standard deviation. Means and standard deviations are separately computed for each batch. Note that in [35], the authors proposed the use of additional learnable parameters after the whitening. We found the effect of this operation to be negligible in most cases. Except for the "BN" and "BN+BM" entries (e.g., in Table 2.2), we did not use the learnable parameters of BN. Note that batch normalization may be related to the homeostatic plasticity mechanisms (e.g., Synaptic Scaling) in the brain [83, 79, 82].

Batch Manhattan (BM)

We were first motivated by looking at how BP could tolerate noisy operations that could be seen as more easily implementable by the brain. We tried relaxing the weight updates by discarding the magnitudes of the gradients. Let the weight at time t be $w(t)$, the update rule is:

$$w(t+1) = w(t) + \eta * \tau(t) \tag{2.3}$$

where η is the learning rate.

We tested several settings of $\tau(t)$ as follows:

Setting 0 (SGD): $\tau(t) = -\sum_b \frac{\partial E}{\partial w} + m * \tau(t-1) - d * w(t)$

Setting 1: $\tau(t) = -sign(\sum_b \frac{\partial E}{\partial w}) + m * \tau(t-1) - d * w(t)$

Setting 2: $\tau(t) = sign(-sign(\sum_b \frac{\partial E}{\partial w}) + m * \tau(t-1) - d * w(t))$

Setting 3: $\tau(t) = sign(\kappa(t))$

where $\kappa(t) = -sign(\sum_b \frac{\partial E}{\partial w}) + m * \kappa(t-1) - d * w(t)$

where m and d are momentum and weight decay rates respectively. $sign()$ means taking the sign (-1 or 1), E is the objective function, and b denotes the indices of samples in the mini-batch. Setting 0 is the SGD algorithm (by "SGD" in this work, we always refer to the mini-batch version with momentum and weight decay). Setting 1 is same as 0 but rounding the accumulated gradients in a batch to its sign. Setting 2 takes an extra final sign after adding the gradient term with momentum and weight decay terms. Setting 3 is something in between 1 and 2, where an final sign is taken, but not accumulated in the momentum term.

We found these techniques to be surprisingly powerful in the sense that they did not lower performance in most cases (as long as learning rates were reasonable). In fact, sometimes they improved performance. This was especially true for smaller training sets. Recall that asymmetric BPs tend to have exploding/vanishing gradients, these techniques are immune to such problems since the magnitudes of gradients are discarded.

We also found that the performance of this technique was influenced by batch size on some experiments. In the cases of very small batch sizes, discarding the magnitudes of the weight updates was sometimes detrimental to performance.

This class of update rule is very similar to a technique called the Manhattan update rule, which can be considered as a simplified version of Rprop [67]. We suggest calling

it "Batch Manhattan" (BM) to distinguish it from the stochastic version [87]. By default, we used setting 1 for BM throughout the Experiments A, B, C1 and C2. The "miscellaneous experiment" at the end of the Section 2.5.4 demonstrates that settings 1, 2 and 3 give similar performances, so the conclusions we draw broadly apply to all of them.

2.4 Related Work

Since the invention of backpropagation (BP) [69], its biological plausibility has been a long-standing controversy. Several authors have argued that BP is not biologically plausible [12, 52, 61, 8, 5]. Various biologically plausible modifications have been proposed. Most involve bidirectional connections e.g. Restricted Boltzmann Machines [29, 75] and so-called recirculation algorithms [28, 61] which despite their name provided, in the case of an autoencoder, an elegant early demonstration that adaptive backwards weight can work without being identical to the forward ones. Recently, there have also been BP-free auto-encoders [4] based on "target propagation" [45].

The most relevant work to ours is a recent paper by Lillicrap et al. [51] of which we became aware after most of this work was done. Lillicrap et al. showed that fixed random feedback weights can support the learning of good representations for several simple tasks: (i) approximating a linear function, (ii) digit recognition on MNIST and (iii) approximating the outputs of a random 3 or 4 layer nonlinear neural network. Our work is very similar in spirit but rather different and perhaps complementary in its results, since we conclude that signs must be concordant between feedforward and corresponding feedback connections for consistent good performance, whereas the magnitudes do not matter, unlike Lilicrap et al. experiments in which both signs and magnitudes were random (but fixed). To explain the difference in our conclusions, it is useful to consider the following points:

1. We systematically explored performance of the algorithms using 15 different datasets because simple tasks like MNIST by themselves do not always reveal differences between algorithms.

	All others	MNIST	CIFAR10&100	SVHN	TIMIT-80
InputSize	119x119x3	28x28x1	32x32x3	32x32x3	1845x1x1
1	Conv 9x9x48/2	Conv 5x5x20/1	Conv 5x5x32/1	Conv 5x5x20/1	FC 512
2	Max-Pool 2x2/2	Max-Pool 2x2/2	Max-Pool 3x3/2	Max-Pool 2x2/2	FC 256
3	Conv 5x5x128/1	Conv 5x5x50/1	Conv 5x5x64/1	Conv 7x7x512/1	FC 80
4	Max-Pool 2x2/2	Max-Pool 2x2/2	Avg-Pool 3x3/2	Max-Pool 2x2/2	
5	FC max(256,#Class*3)	FC 500	Conv 5x5x64/1	FC 40	
6	FC #Class*2	FC 10	Avg-Pool 3x3/2	FC 10	
7	FC #Class		FC 128		
8			FC 10/100		

Table 2.1: Network architectures used in the experiments: AxBxC/D means C feature maps of size AxB, with stride D. The CIFAR10&100 architecture has a 2 units zero-padding for every convolution layer and 1 unit right-bottom zero-padding for every pooling layer. The other models do not have paddings. "FC X" denotes Fully Connected layer of X feature maps. In the first model, the number of hidden units in FC layers are chosen according to the number of classes (denoted by "#Class") in the classification task. " max(256,#Class*3)" denotes 256 or #Class*3, whichever is larger. Rectified linear units (ReLU) are used as nonlinearities for all models.

2. We tested deeper networks, since the accuracy of asymmetric BP's credit assignment may critically attenuate with depth (for task (i) and (ii) Lillicrap et al. used a 3-layer (1 hidden layer) fully-connected network, and for task (iii) they used a 3 or 4 layer fully-connected network, whereas in most of our experiments, we use deeper and larger CNNs as shown in Table 2.1).

3. We found that local normalizations/stabilizations is critical for making asymmetric BP algorithms work. As shown by our results in Table 2.4, the random feedbacks scheme (i.e. the "RndF" column) suggested by Lillicrap et al. seem to work well only on one or two tasks, performing close to chance on most of them. Only when combined with Batch Normalization ("RndF+BN" or "RndF+BN+BM" in Table 2.4), it appears to become competitive.

4. We investigated several variants of asymmetric BPs such as sign-concordance (Table 2.2 and 2.3), batchwise-random vs. fixed-random feedbacks (Table 2.3) and learning with clamped layers (Table 2.4 Exp. C2).

2.5 Experiments

2.5.1 Method

We were interested in relative differences between algorithms, not absolute performance. Thus we used common values for most parameters across all datasets to facilitate comparison. Key to our approach was the development of software allowing us to easily evaluate the "cartesian product" of models (experimental conditions) and datasets (tasks). Each experiment was a {model,dataset} pair, which was run 5 times using different learning rates (reporting the best performance).

2.5.2 Datasets

We extensively test our algorithms on 15 datasets of 5 Categories as described below. No data augmentation (e.g., cropping, flip, etc.) is used in any of the experiments.
Machine learning tasks: MNIST [46], CIFAR-10 [40], CIFAR-100 [40], SVHN[57], STL10 [10]. Standard training and testing splits were used.
Basic-level categorization tasks: Caltech101 [17]: 102 classes, 30 training and 10 testing samples per class. Caltech256-101 [22]: we train/test on a subset of randomly sampled 102 classes. 30 training and 10 testing per class. iCubWorld dataset [16]: We followed the standard categorization protocol of this dataset.
Fine-grained recognition tasks: Flowers17 [59], Flowers102 [60]. Standard training and testing splits were used.
Face Identification: Pubfig83-ID [64], SUFR-W-ID [47], LFW-ID [33] We did not follow the usual (verification) protocol of these datasets. Instead, we performed a 80-way face identification task on each dataset, where the 80 identities (IDs) were randomly sampled. Pubfig83: 85 training and 15 testing samples per ID. SUFR-W: 10 training and 5 testing per ID. LFW: 10 training and 5 testing per ID.
Scene recognition: MIT-indoor67 [65]: 67 classes, 80 training and 20 testing per class
Non-visual task: TIMIT-80 [19]: Phoneme recognition using a fully-connected

Experiment A	SGD	BM	BN	BN+BM	uSF	NuSF	uSF +BM	uSF +BN	uSF +BN +BM	brSF +BN +BM	frSF +BN +BM
MNIST	0.67	0.99	0.52	0.73	88.65	0.60	0.95	0.55	0.83	0.80	0.91
CIFAR	22.73	23.98	**16.75**	**17.94**	90.00	40.60	26.25	**19.48**	**19.29**	**18.44**	**19.02**
CIFAR100	55.15	58.44	**49.44**	**51.45**	99.00	71.51	65.28	57.19	53.12	**50.74**	52.25
SVHN	9.06	10.77	7.50	9.88	80.41	14.55	9.78	8.73	9.67	9.95	10.16
STL10	48.01	**44.14**	45.19	**43.19**	90.00	56.53	46.41	48.49	**41.55**	**42.74**	**42.68**
Cal101	74.08	**66.70**	**66.07**	**61.75**	98.95	**70.50**	75.24	**63.33**	**60.70**	**59.54**	**60.27**
Cal256-101	87.06	**83.43**	**82.94**	**81.96**	99.02	85.98	86.37	**82.16**	**80.78**	**78.92**	**80.59**
iCub	57.62	55.57	**46.43**	**37.08**	89.96	66.57	70.61	61.37	**48.38**	**47.33**	**46.08**
Flowers17	35.29	**31.76**	36.76	32.35	94.12	42.65	38.24	35.29	32.65	**29.41**	**31.47**
Flowers102	77.30	77.57	75.78	74.92	99.67	77.92	79.25	**71.74**	**73.20**	**73.31**	**73.57**
PubFig83-ID	63.25	**54.42**	**51.08**	**41.33**	98.75	78.58	65.83	**54.58**	**40.67**	**42.67**	**40.33**
SUFR-W-ID	80.00	**74.25**	**75.00**	**65.00**	98.75	83.50	79.50	**72.00**	**65.75**	**66.25**	**66.50**
LFW-ID	79.25	**74.25**	**73.75**	**55.75**	98.75	85.75	80.75	**73.75**	**56.25**	**57.25**	**55.75**
Scene67	87.16	85.37	86.04	**82.46**	98.51	88.21	87.09	87.09	**81.87**	**82.31**	**81.79**
TIMIT80	23.04	25.92	23.92	24.40	23.60	29.28	25.84	25.04	25.12	25.24	24.92

Table 2.2: **Experiment A:** The magnitudes of feedbacks do not matter. Sign concordant feedbacks can produce strong performance. Numbers are error rates (%). Yellow: performances worse than baseline(SGD) by 3% or more. **Blue**: performances better than baseline(SGD) by 3% or more.

network. There are 80 classes, 400 training and 100 testing samples per class.

2.5.3 Training Details

The network architectures for various experiments are listed in Table 2.1. The input sizes of networks are shown in the second row of the table. All images are resized to fit the network if necessary.

Momentum was used with hyperparameter 0.9 (a conventional setting). All experiments were run for 65 epochs. The base learning rate: 1 to 50 epochs $5 * 10^{-4}$, 51 to 60 epochs $5 * 10^{-5}$, and 61 to 65 epochs $5 * 10^{-6}$. All models were run 5 times on each dataset with base learning rate multiplied by 100, 10, 1, 0.1, 0.01 respectively. This is because different learning algorithms favor different magnitudes of learning rates. The best validation error among all epochs of 5 runs was recorded as each model's final performance. The batch sizes were all set to 100 unless stated otherwise. All experiments used a softmax for classification and the cross-entropy loss function. For testing with batch normalization, we compute exponential moving averages (alpha=0.05) of training means and standard deviations over 20 mini batches after each training epoch.

2.5.4 Results

Experiment A: sign-concordant Feedback

In this experiment, we show the performances of setting 1, 2 and 3 in Section 2.2, which we call **strict sign-concordance** cases: while keeping the signs of feedbacks the same as feedforward ones, the magnitudes of feedbacks are either randomized or set to uniform. The results are shown in Table 2.2 : Plus sign (+) denotes combination of methods. For example, uSF+BM means Batch Manhattan with uniform sign-concordant feedback. **SGD:** Stochastic gradient descent, the baseline algorithm. **BM:** SGD + Batch Manhattan. **BN:** SGD + Batch Normalization. **BN+BM:** SGD + Batch Normalization + Batch Manhattan. **uSF:** Uniform sign-concordant feedback. This condition often had exploding gradients. **NuSF:** same as uSF, but with feedback weights normalized by dividing the number of inputs of the feedforward filter (filter width * filter height * input feature number). This scheme avoids the exploding gradients but still suffers from vanishing gradients. **uSF+BM:** this setting is somewhat unstable for small batch sizes. Two training procedures were explored: (1) batch size 100 for all epochs (2) batch size 100 for 3 epochs and then batch size 500 for the remaining epochs. The best performance was reported. While this gives a little advantage to this model since more settings were tried, we believe it is informative to isolate the stability issue and show what can be achieved if the model converges well. Note that uSF+BM is the only entry with slightly different training procedures. All other models share exactly the same training procedure & parameters. **uSF+BN, uSF+BN+BM, brSF+BN+BM, frSF+BN+BM:** These are some combinations of uSF, brSF, frSF, BN and BM. The last three are the most robust, well-performing ,and biologically-plausible algorithms.

Experiment B: Violating Sign-Concordance with probability p

In this experiment, we test the effect of **partial sign-concordance**. That is, we test settings 4 and 5 as described in Section 2.2. In these cases, the feedback weight

	Baseline	Part 1: Random M and S every batch					Part 2: Fixed random M and S				
Experiment B	SGD	100%	75%	50%	25%	Same Sign (brSF +BN+BM)	100%	75%	50%	25%	Same Sign (frSF +BN+BM)
MNIST	0.67	7.87	8.34	7.54	1.01	0.80	4.25	3.56	1.37	0.84	0.91
CIFAR	22.73	71.18	75.87	70.60	19.98	**18.44**	71.41	68.49	31.54	20.85	**19.02**
CIFAR100	55.15	93.72	96.23	94.58	68.98	**50.74**	96.26	95.22	71.98	56.02	52.25
SVHN	9.06	75.25	77.91	74.64	10.94	9.95	37.78	32.72	15.02	11.50	10.16
STL10	48.01	69.65	72.50	72.10	**44.82**	**42.74**	72.46	68.96	50.54	**43.85**	**42.68**
Cal101	74.08	91.46	93.99	91.36	**67.65**	**59.54**	94.20	87.57	**68.60**	**63.12**	**60.27**
Cal256-101	87.06	93.24	94.02	92.75	**82.75**	**78.92**	95.98	90.88	84.22	**83.04**	**80.59**
iCub	57.62	75.56	79.36	83.61	**52.42**	**47.33**	68.97	63.82	64.62	**51.22**	**46.08**
Flowers17	35.29	70.29	82.06	73.82	35.00	**29.41**	79.12	61.76	44.12	35.59	**31.47**
Flowers102	77.30	90.54	92.58	89.56	**73.18**	**73.31**	93.69	93.77	77.62	77.85	**73.57**
PubFig83-ID	63.25	95.33	95.67	94.17	61.25	**42.67**	94.25	89.25	64.33	**47.17**	**40.33**
SUFR-W-ID	80.00	95.00	95.75	92.50	**71.00**	**66.25**	95.50	95.50	77.25	**68.00**	**66.50**
LFW-ID	79.25	95.25	96.00	94.50	**64.50**	**57.25**	95.75	95.25	**75.00**	**59.75**	**55.75**
Scene67	87.16	94.48	94.78	93.43	**83.13**	**82.31**	95.37	92.69	88.36	84.40	**81.79**
TIMIT80	23.04	55.32	61.36	55.28	26.48	25.24	62.60	33.48	27.56	26.08	24.92

Table 2.3: **Experiment B Part 1 (left):** Feedbacks have random magnitudes, varing probability of having different signs (percentages in second row, column 3-7) from the feedforward ones. The M and S redrawn in each mini-batch. Numbers are error rates (%). **Yellow**: performances worse than baseline(SGD) by 3% or more. **Blue**: performances better than baseline(SGD) by 3% or more. **Experiment B Part 2 (right):** Same as part 1, but The M and S were fixed throughout each experiment.

Experiment C1	SGD	RndF	NuSF	BN	RndF+BN	RndF+BM	RndF+BN+BM	uSF+BN+BM
MNIST	0.67	1.81	0.60	0.52	0.83	1.89	1.07	0.83
CIFAR	22.73	42.69	40.60	**16.75**	24.35	62.71	25.75	**19.29**
CIFAR100	55.15	90.88	71.51	**49.44**	60.83	97.11	64.69	53.12
SVHN	9.06	12.35	14.55	7.50	12.63	13.63	12.79	9.67
STL10	48.01	57.80	56.53	45.19	51.60	80.39	47.39	**41.55**
Cal101	74.08	88.51	**70.50**	**66.07**	72.81	98.42	**67.12**	**60.70**
Cal256-101	87.06	94.12	85.98	**82.94**	85.49	98.14	**83.63**	**80.78**
iCub	57.62	67.87	66.57	**46.43**	58.82	84.41	59.02	**48.38**
Flowers17	35.29	69.41	42.65	36.76	43.53	91.18	38.24	32.65
Flowers102	77.30	92.31	77.92	75.78	81.22	96.13	78.99	**73.20**
PubFig83	63.25	95.42	78.58	**51.08**	67.00	97.67	**55.25**	**40.67**
SUFR-W-ID	80.00	94.75	83.50	**75.00**	77.75	97.25	**69.00**	**65.75**
LFW-ID	79.25	95.75	85.75	**73.75**	79.25	97.75	**67.50**	**56.25**
Scene67	87.16	95.75	88.21	86.04	87.84	97.69	87.09	**81.87**
TIMIT80	23.04	26.76	29.28	23.92	26.52	33.12	26.32	25.12
Experiment C2	SGD *Bottom*	SGD *Top*	RndF *Bottom*	RndF *Top*	RndF+BN+BM *Bottom*	RndF+BN+BM *Top*	uSF+BN+BM *Bottom*	uSF+BN+BM *Top*
MNIST	0.65	3.85	85.50	3.81	86.74	3.81	0.66	3.85
CIFAR	23.12	56.80	89.71	56.77	78.90	58.54	**16.72**	57.84
CIFAR100	59.49	80.71	98.79	80.65	98.69	84.34	61.61	84.10
SVHN	8.31	75.22	82.86	75.12	84.84	69.99	10.96	71.89
STL10	49.96	74.69	88.36	72.44	81.31	76.11	52.18	76.09
Cal101	71.97	82.72	98.63	79.14	98.21	80.40	**63.86**	79.98
Cal256-101	86.08	89.71	98.43	88.92	98.14	89.02	**82.84**	89.12
iCub	**46.73**	83.96	87.56	83.26	80.36	84.31	**49.33**	84.16
Flowers17	38.24	70.59	92.35	70.00	87.35	77.06	45.00	77.06
Flowers102	78.99	86.57	97.89	86.84	98.11	84.24	78.09	84.57
PubFig83	66.75	89.58	97.67	89.58	97.67	89.67	**43.83**	89.50
SUFR-W-ID	80.50	90.50	97.50	90.50	97.50	89.50	**71.50**	89.50
LFW-ID	79.75	92.50	98.25	93.00	97.00	89.50	**65.00**	89.50
Scene67	88.73	91.57	97.84	91.49	97.54	91.19	85.97	91.04
TIMIT80	23.52	46.20	95.00	46.20	93.00	39.76	24.96	40.16

Table 2.4: **Experiment C1:** fixed random feedbacks. **Experiment C2:** (.)*Bottom*: The model's last layer is initialized randomly and clamped/frozen. All learning happens in the layers before the last layer. (.)*Top*: The model's layers before the last layer are initialized randomly and clamped/frozen. All learning happens in the last layer. Numbers are error rates (%). **Yellow**: performances worse than baseline(SGD) by 3% or more. **Blue**: performances better than baseline(SGD) by 3% or more.

	SGD	BM1	BM2	BM3	uSF+BN	uSF+BN+BM1	uSF+BN+BM2	uSF+BN+BM3
MNIST	0.67	0.99	1.30	1.09	0.55	0.83	0.72	0.61
CIFAR	22.73	23.98	23.09	20.47	**19.48**	**19.29**	**18.87**	**18.38**
CIFAR100	55.15	**58.44**	**58.81**	52.82	57.19	53.12	52.38	54.68
SVHN	9.06	10.77	**12.31**	**12.23**	8.73	9.67	10.54	9.20
STL10	48.01	**44.14**	**44.74**	45.23	48.49	**41.55**	47.71	46.45
Cal101	74.08	**66.70**	**65.96**	**70.28**	**63.33**	**60.70**	**64.38**	**62.59**

Table 2.5: Different settings of Batch Manhattan (as described in Section 2.3) seem to give similar performances. SGD: setting 0, BM1: setting 1, BM2: setting 2, BM3: setting 3. The interaction of BM with sign concordant feedback weights (uSF) and Batch Normalization are shown in "uSF+BN+(.)" entries. Numbers are error rates (%). Yellow: performances worse than baseline (SGD) by 3% or more. **Blue**: performances better than baseline(SGD) by 3% or more.

magnitudes were all random. Strict sign-concordance was relaxed by manipulating the probability p of concordance between feedforward and feedback weight signs. Feedback weights $V = M \circ sign(W) \circ S_p$ depend on the matrix S_p as defined in Section 2.2. Table 2.3 Part 1 reports results from setting 4, the case where a new M and S_p is sampled for each batch. Table 2.3 Part 2 reports results of setting 5, the case where M and S_p are fixed.

Experiment C1: Fixed Random Feedback

Results are shown in Table 2.4: **RndF:** fixed random feedbacks. **RndF+BN, RndF+BN+BM:** some combinations of RndF, BN and BM. **uSF+BN+BM:** one of our best algorithms, for reference. The "RndF" setting is the same as the one proposed by [51]. Apparently it does not perform well on most datasets. However, combining it with Batch Normalization makes it significantly better. Although it remains somewhat worse than its sign concordant counterpart. Another observation is that random feedback does not work well with BM alone (but better with BN+BM).

Experiment C2: Control experiments for Experiment C1

The fact that random feedback weights can learn meaningful representations is somewhat surprising. We explore this phenomenon by running some control experiments, where we run two control models for each model of interest: **1.** (.)*Bottom*: The

model's last layer is initialized randomly and clamped/frozen. All learning happens in the layers before the last layer. **2.** (.) *Top*: The model's layers before the last layer are initialized randomly and clamped/frozen. All learning happens in the last layer. Results are shown in Table 2.4.

There are some observations: **(i)** When only the last layer was allowed to adapt, all models behaved similarly. This was expected since the models only differed in the way they backpropagate errors. **(ii)** With the last layer is clamped, random feedback cannot learn meaningful representations. **(iii)** In contrast, the models with sign concordant feedback can learn surprisingly good representations even with the last layer locked. We can draw the following conclusions from such observations: sign concordant feedback ensures that meaningful error signals reach lower layers by itself, while random feedback is not sufficient. If all layers can learn, random feedback seems to work via a "mysterious co-adaptation" between the last layer and the layers before it. This "mysterious co-adaptation" was first observed by [51], where it was called "feedback alignment" and some analyses were given. Note that our Experiment C shows that the effect is more significant if Batch Normalization is applied.

Miscellaneous Experiment: different settings of Batch Manhattan

We show that the 3 settings of BM (as described in Section 2.3) produce similar performances. This is the case for both symmetric and asymmetric BPs. Results are in Table 2.5.

2.6 Discussion

This work aims to establish that there exist variants of the gradient backpropagation algorithm that could plausibly be implemented in the brain. To that end we considered the question: how important is weight symmetry to backpropagation? Through a series of experiments with increasingly asymmetric backpropagation algorithms, our work complements a recent demonstration[51] that perfect weight symmetry can be significantly relaxed while still retaining strong performance.

These results show that Batch Normalization and/or Batch Manhattan are crucial for asymmetric backpropagation to work. Furthermore, they are complementary operations that are better used together than individually. These results highlight the importance of sign-concordance to asymmetric backpropagation by systematically exploring how performance declines with its relaxation.

Finally, let us return to our original motivation. How does all this relate to the brain? Based on current neuroscientific understanding of cortical feedback, we cannot make any claim about whether such asymmetric BP algorithms are actually implemented by the brain. Nevertheless, this work shows that asymmetric BPs, while having less constraints, are not computationally inferior to standard BP. So if the brain were to implement one of them, it could obtain most of the benefits of the standard algorithm.

This work suggests a hypothesis that could be checked by empirical neuroscience research: if the brain does indeed implement an asymmetric BP algorithm, then there is likely to be a high degree of sign-concordance in cortical forward-backward connections.

These empirical observations concerning Batch Manhattan updating may shed light on the general issue of how synaptic plasticity may implement learning algorithms. They show that changes of synaptic strength can be rather noisy. That is, the *sign* of a long term accumulation of synaptic potentiation or depression, rather than precise magnitude, is what is important. This scheme seems biologically implementable.

Chapter 3

A Biologically-Plausible Framework of Residual Learning, Recurrent Neural Networks and Visual Cortex

3.1 Introduction

Residual learning [24], a novel deep learning scheme characterized by ultra-deep architectures has recently achieved state-of-the-art performance on several popular vision benchmarks. The most recent incarnation of this idea [26] with hundreds of layers demonstrate consistent performance improvement over shallower networks. The 3.57% top-5 error achieved by residual networks on the ImageNet test set arguably rivals human performance.

Because of recent claims [86] that networks of the AlexNet[41] type successfully predict properties of neurons in visual cortex, one natural question arises: how similar is an ultra-deep residual network to the primate cortex? A notable difference is the depth. While a residual network has as many as 1202 layers[24], biological systems seem to have one or two orders of magnitude less, if we make the customary assumption that a weighted linear combination layer (plus a nonlinearity) in the NN architecture corresponds to a cortical area. In fact, there are about half a dozen areas in the

ventral stream of visual cortex from the retina to the Inferior Temporal cortex. Notice that it takes in the order of 10ms for neural activity to propagate from one neuron to another one (remember that spiking activity of cortical neurons is usually well below 100 Hz). The evolutionary advantage of having fewer layers is apparent: it supports rapid (100msec from image onset to meaningful information in IT neural population) visual recognition, which is a key ability of human and non-human primates [81, 72].

It is intriguingly possible to account for this discrepancy by taking into account recurrent connections within each visual area. Areas in visual cortex comprise six different layers with lateral and feedback connections [42], which are believed to mediate some attentional effects [6, 42, 36, 66, 34] and even learning (such as backpropagation [49]). "Unrolling" in time the recurrent computations carried out by the visual cortex provides an equivalent "ultra-deep" feedforward network, which might represent a more appropriate comparison with the state-of-the-art computer vision models.

In addition, we conjecture that the effectiveness of recent "ultra-deep" neural networks primarily come from the fact they can efficiently model the recurrent computations that are required by the recognition task. We show evidence for this conjecture by demonstrating that 1. a deep residual network is formally equivalent to a specific shallow RNN [1]; 2. such a RNN with weight sharing, thus with orders of magnitude less parameters (depending on the unrolling depth), can retain most of the performance of the corresponding deep residual network.

Furthermore, we generalize such a RNN to a class of more biologically plausible models of visual cortex to account for multi-stage recurrent processing and show their effectiveness on the CIFAR-10 and ImageNet dataset.

Another minor contribution of this work is that, we propose using *time-specific* batch normalization (TSBN, defined in Section 3.3) in hidden-to-hidden transitions of RNN. We are also the first to show when TSBN is used, there is no difficulty training RNNs (even multi-state fully recurrent ones) with ReLUs. This was previously believed to be difficult without careful weight initializations [44].

[1] We use the term "RNN" to broadly refer to any neural network with recurrent activations, not the "vanilla/plain RNN" (the baseline model people often use to compare with LSTMs).

3.2 Equivalence of ResNet and a specific RNN

3.2.1 Intuition

We discuss here a very simple observation: a Residual Network (ResNet) approximates a specific, standard Recurrent Neural Network (RNN) implementing the discrete dynamical system described by

$$h_t = K \circ (h_{t-1}) + h_{t-1} \tag{3.1}$$

where h_t is the activity of the neural layer at time t and K is a nonlinear operator. Such a dynamical systems corresponds to the feedback system of Figure 3-1 (B). Figure 3-1 (A) shows that unrolling in (discrete) time the feedback system gives a deep residual network with the same (that is, shared) weights among the layers. The number of layers in the unrolled network corresponds to the discrete time iterations of the dynamical system. The identity shortcut mapping that characterizes residual learning appears in the figure.

From a biological perspective, a neuron's current activation, characterized by Excitatory Postsynaptic Potential (EPSP) of soma (i.e., activation before nonlinearity) is roughly the sum of inputs plus the residual EPSP from the previous time step. The latter corresponds naturally to the identity shortcut mapping (especially the pre-activation scheme in [26], which we adopted in this work).

3.2.2 Formulation in terms of Dynamical Systems

We frame recurrent and residual neural networks in the language of dynamical systems. We consider here dynamical systems in discrete time, though most of the definitions carry over to continuous time. A neural network (that we assume for simplicity to have a single layer with n neurons) can be a dynamical system with a dynamics defined as

$$h_{t+1} = f(h_t; w_t) + x_t \tag{3.2}$$

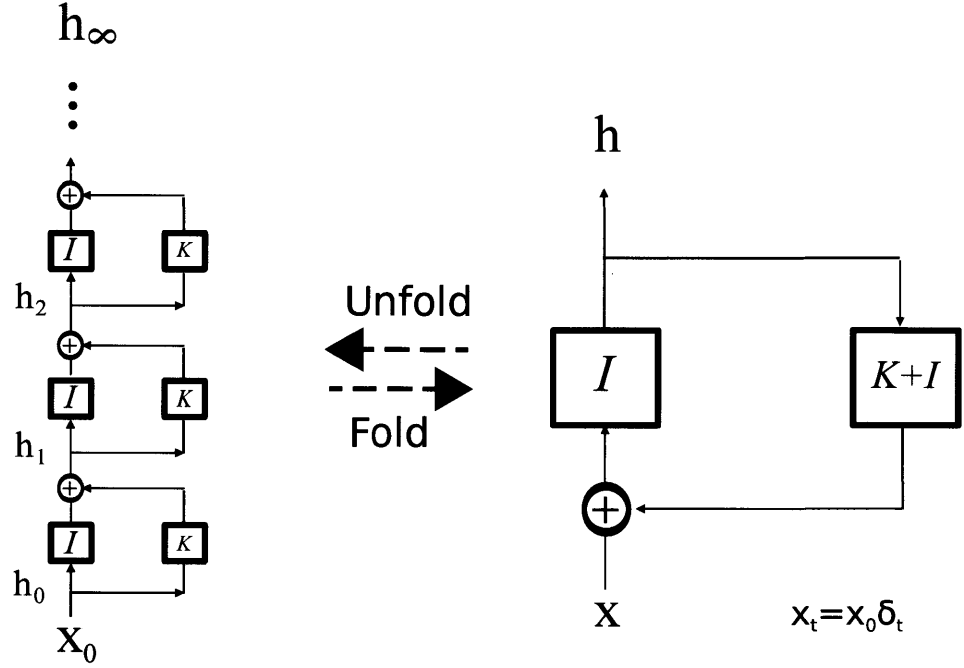


Figure 3-1: A formal equivalence of a ResNet (A) with weight sharing and a RNN (B). I is the identity operator. K is an operator denoting the nonlinear transformation called f in the main text. x_t is the value of the input at time t. δ_t is a Kronecker delta function.

where $h_t \in \mathbb{R}^n$ is the activity of the n neurons in the layer at time t and $f : \mathbb{R}^n \to \mathbb{R}^n$ is a continuous, bounded function parametrized by the vector of weights w_t. In a typical neural network, f is synthesized by the following relation between the activity y_t of a single neuron and its inputs x_{t-1}:

$$y_t = \sigma(\langle w, x_{t-1} \rangle + b), \tag{3.3}$$

where σ is a nonlinear function such as the linear rectifier $\sigma(\cdot) = |\cdot|_+$.

A standard classification of dynamical systems defines the system as

1. *homogeneous* if $x_t = 0, \forall t > 0$ (alternatively the equation reads as $h_{t+1} = f(h_t; w_t)$ with the inital condition $h_0 = x_0$)

2. *time invariant* if $w_t = w$.

Residual networks with weight sharing thus correspond to *homogeneous, time-invariant* systems which in turn correspond to a feedback system (see Figure 3-1) with an input which is non-zero only at time $t = 0$ ($x_{t=0} = x_0, x_t = 0 \ \forall t > 0$) and with $f(z) = (K + I) \circ z$:

$$h_n = f(h_t; w_t) = (K + I)^n \circ x_0 \tag{3.4}$$

"Normal" residual networks correspond to *homogeneous, time-variant* systems.

3.3 A Generalized RNN for Multi-stage Fully Recurrent Processing

As shown in the previous section, the recurrent form of a ResNet is actually shallow (if we ignore the possible depth of the operator K). In this section, we generalize it into a moderately deep RNN that reflects the multi-stage processing in the primate visual cortex.

Multi-state Graph We propose a general formulation that can capture the computations performed by a multi-stage processing hierarchy with full recurrent connections. Such a hierarchy can be characterized by a directed (cyclic) graph G with vertices V and edges E: G = {V,E}. where vertices V is a set contains all the processing stages (i.e., we also call them states). Take the ventral stream of visual cortex for example, $V = \{LGN, V1, V2, V4, IT\}$. Note that retina is not listed since there is no known feedback from primate cortex to the retina. The edges E are a set that contains all the connections (i.e., transition functions) between all vertices/states, e.g., V1-V2, V1-V4, V2-IT, etc. One example of such a graph is in Figure 3-2 (A).

Pre-net and Post-net The multi-state fully recurrent system does not have to receive raw inputs. Rather, a (deep) neural network can serve as a preprocesser. We call the preprocesser a "pre-net" as shown in Figure 3-2 (B). On the other hand, one also needs a "post-net" as a postprocessor and provide supervisory signals to the recurrent system and the pre-net. The pre-net, recurrent system and post-net are

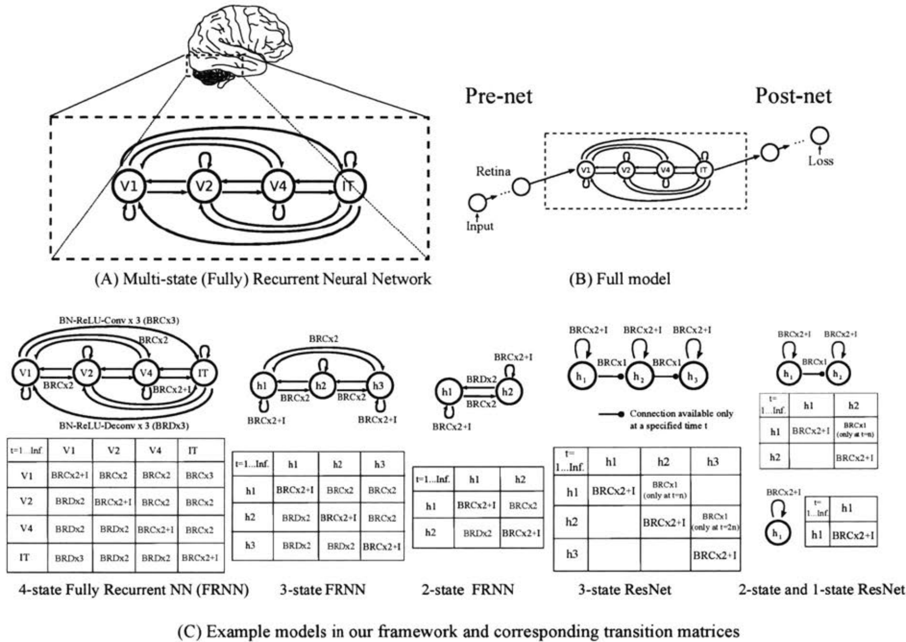


Figure 3-2: (A) Modeling the ventral stream of visual cortex using a multi-state fully recurrent neural network. (B) the model consists of a input, a recurrent part and a output. (C) Examples of the recurrent part of the model and corresponding transition matrices used in this work. "BN" denotes Batch Normalization and "Conv" denotes convolution. Deconvolution layer (denoted by "Deconv") is [88] used as a transition function from a spacially small state to a spacially large one. BRCx2/BRDx2 denotes a BN-ReLU-Conv/Deconv-BN-ReLU-Conv/Deconv pipeline (similar to a residual module [26]). There is always a 2x2 subsampling/upsampling between nearby states (e.g., V1/h1: 32x32, V2/h2: 16x16, V4/h3:8x8, IT:4x4). Stride 2 (convolution) or upsampling 2 (deconvolution) is used in transition functions to match the spacial sizes of input and output states. The intermediate feature sizes of transition function BRCx2/BRDx2 or BRCx3/BRDx3 are chosen to be the average feature size of input and output states. "+I" denotes a identity shortcut mapping.

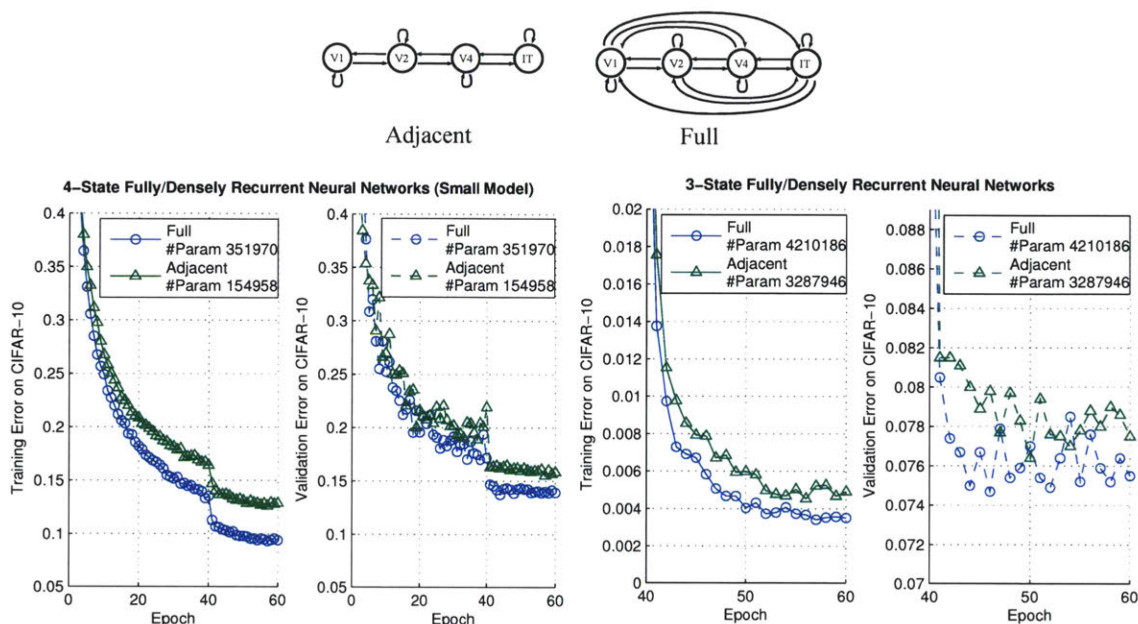


Figure 3-3: The performance of 4-state and 3-state models. The state sizes of the 4-state model are: 32x32x8, 16x16x16, 8x8x32, 4x4x64. The state sizes of the 3-state model are: 32x32x64, 16x16x128, 8x8x256. Only small 4-state models were tried since they are very computationally heavy. The readout time is t=5 for both models. All models are time-invariant systems (i.e., weights are shared across time).

Model	Error (%)	Depth	Training Epochs
All-CNN [77]	7.25%	11	350
Highway Network [78]	7.72%	19	400
ResNet-110 [26]	6.61%	110	200
ResNet-164 ResNet-1001 [26]	5.46% 4.69%	164 1001	200 200
Human[37]	≈ 6%	Recurrent	-
Same FRNN with different readout time 3-state FRNN (readout t=5) 3-state FRNN (readout t=10)	 7.44% 6.86%	Recurrent 13 (unrolled) 23 (unrolled)	 60 60

Table 3.1: Compare our model (FRNN) with other best models (prior to our work) on CIFAR-10. All models were trained with simple translation and mirror augmentation. The depth is the number of weighted linear combination layers and pooling layers. All of our models were trained with 60 epochs for consistency, since we did not focus on the absolute performance. We expect better performance if more epochs are used (will report in the next revision). Nevertheless, these models are all at the level of human performance. We believe at this point biological consistency is more interesting than marginal performance differences. Also our models achieve high performance with small latency (i.e., depth), which supports rapid visual recognition and is crucial for organism's survival.

trained in an end-to-end fashion with backpropagation.

For most models in this work, unless stated otherwise, the pre-net is a simple 3x3 convolutional layer and the post-net is a pipeline of a batch normalization, a ReLU, a global average pooling and a fully connected layer (or a 1x1 convolution, we use these terms interchangeably).

Take primate visual system for instance, the retina is a part of the "pre-net". It does not receive any feedback from the cortex and thus can be separated from the recurrent system for simplicity. In Section 3.5, we also tried 3 layers of 3x3 convolutions as an pre-net, which might be more similar to a retina, and observed slightly better performance.

Transition Matrix The connections between states over time can be represented by a 3-D matrix where each element (i,j,t) represents the *transition function* from state i to state j at time t. This formulation is flexible in the sense that transition functions can vary over time (e.g., being blocked from time t_1 to time t_2, etc.). This formulation allows us to design a system where multiple locally recurrent systems are connected sequentially: a downstream recurrent system only receives inputs when its upstream recurrent system finishes, similar to recurrent convolutional neural networks (e.g., [48]). This system with non-shared weights can also represent exactly the state-of-the-art ResNet. Example transition matrices used in this work are shown in Figure 3-2 (C). When there are multiple transition functions to a state, their outputs are summed (for ResNet) or averaged (for our fully recurrent neural networks (FRNN)). Averaging gives slightly better performance for FRNN (about 1%).

Weight sharing Given an unrolled network, a weight sharing configuration can be described as a set S, whose element is a set of tied weights $s = \{W_{i_1,j_1,t_1}, ..., W_{i_m,j_m,t_m}\}$, where W_{i_m,j_m,t_m} denotes the weight of the transition functions from state i_m to j_m at time t_m. This requires: 1. all weights $W_{i_m,j_m,t_m} \in s$ have the same initial values. 2. the actual gradients used for updating each element of s is the sum of the gradients of all elements in s: $\forall W \in s$, $(\frac{\partial E}{\partial W})_{used} = \sum_{W' \in s}(\frac{\partial E}{\partial W'})_{original}$, where E is the training objective.

Notations: Unrolling Depth vs. Readout Time The meaning of "unrolling

depth" may vary in different RNN models since "unrolling" a cyclic graph is not well defined. In this work, we adopt a biologically-plausible definition: we simulate the time after the onset of the visual stimuli assuming each transition function takes constant time 1. We use the term "readout time" to refer to the time the post-net reads the data from the last state. Regarding the initial values, at $t = 0$ all states are empty except that the first state has some data received from the pre-net. We only start simulate a transition function when its input state is populated.

This definition in principle allows one to have quantitive comparisons with biological systems. e.g., for a model with readout time t, the wall clock time can be estimated to be $20t$ to $50t$ ms, considering the latency of a single layer of biological neurons.

Sequential vs. Static Inputs/Outputs As a RNN, our model supports sequential data processing and in principle all other tasks supported by traditional RNNs.

Batch Normalizations for RNNs As an additional observation, we found that it generally hurts performance when the normalization statistics (i.e., average, standard deviation) in batch normalization are shared across time. This may explain the difficulties observed in [43]. However, good performance is restored if we apply a procedure we call a "time-specific normalization": mean and standard deviation are calculated independently for every t (using training set). In CIFAR-10 experiments (except Table 3.1) we do not use the learnable parameters of BN. In ImageNet experiments, we do used the learnable parameters (shared over time) since they noticeably improves performance.

3.4 Related Work

Deep Recurrent Neural Networks: Our final model is deep and similar to a stacked RNN [71, 15, 20] with several main differences: 1. our model has feedback transitions between hidden layers and self-transition from each hidden layer to itself. 2. our model has identity shortcut mappings inspired by residual learning. 3. our transition functions are deep and convolutional. As suggested by [62], the term depth

in RNN could also refer to input-to-hidden, hidden-to-hidden or hidden-to-output connections. Our model is deep in all of these senses. See Section 3.3.

Recursive Neural Networks and Convolutional Recurrent Neural Networks: When unfolding RNN into a feedforward network, the weights of many layers are tied. This is reminiscent of Recursive Neural Networks (Recursive NN), first proposed by [76]. Recursive NN are characterized by applying same operations recursively on a structure. The convolutional version was first studied by [14]. Subsequent related work includes [63] and [48]. One characteristic distinguishes our model and residual learning from Recursive NN and convolutional recurrent NN is whether there are identity shortcut mappings. This discrepancy seems to account for the superior performance of residual learning and of our model over the latters.

Our work may explain the seemingly surprising observation of "stochastic depth" [32] that randomly replacing a subset of layers of ResNet by identity mappings during training does not hurt performance — if they are similar refinements performed repeatedly, dropping a few of them should not be catastrophic. A recent report [7] we became aware of after we finished this work discusses the idea of imitating cortical feedback by introducing loops into neural networks. A Highway Network [78] is a feedforward network inspired by Long Short Term Memory [30] featuring gated shortcut mappings (instead of hardwired identity mappings used by ResNet).

3.5 Experiments

Dataset and training details We test most models on the standard CIFAR-10 [40] dataset. All images are 32x32 pixels with color. Data augmentation is performed in the same way as [24]. Momentum was used with hyperparameter 0.9. Experiments were run for 60 epochs with batchsize 64 unless stated otherwise. The learning rates are 0.01 for the first 40 epochs, 0.001 for epoch 41 to 50 and 0.0001 for the last 10 epochs. All experiments used the cross-entropy loss function and softmax for classification. Batch Normalization (BN) [35] is used for all experiments. But the learnable scaling and shifting parameters are not used in the recurrent parts of the model unless stated

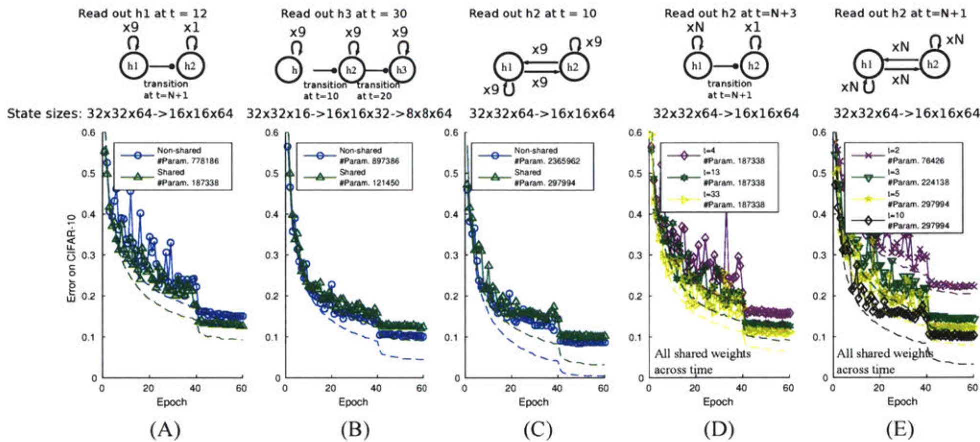


Figure 3-4: The transition matrices of all models are shown in Figure 3-2C. "#Param" denotes the number of parameters. For each model, the effective unrolling factors "xN" are determined by the readout time t (See Section 3.3 for the definition). Dashed lines are training errors. For multi-state ResNet, a downstream state only executes after receiving inputs from upstream states. For fully recurrent nets, all transitions execute concurrently. The state sizes are shown above. (A) A 2-state ResNet with the second state only unrolled once. This architecture works better with shared rather than non-shared weights on CIFAR-10. In general, we empirically observe that weight sharing in the 1st state of a multi-state system is often beneficial. One conjecture is that our transition function might match better low-level visual recurrent computations. (B) Standard 3-state ResNet. (C) 2-state fully recurrent NN. (D) Different readout time of (A). (E) A 2-state fully recurrent network with different readout time. There is consistent performance improvement as t increases. The number of parameters changes since at some t, some recurrent connections have not been contributing to the output and thus their number of parameters are subtracted from the total. We observe in (D) and (E) that error reduces as t increases, while #param. is kept the same.

otherwise. Network weights were initialized with the method described in [25]. The implementations are based on MatConvNet[85].

ResNet with Shared Weights Across Time We conjecture that the effectiveness of ResNet mainly comes from the fact that it efficiently models the recurrent computations required by the recognition task. If this is the case, one should be able to reinterpret ResNet as a RNN with weight sharing and achieve comparable performance to the original version. We demonstrate various incarnations of this idea and show it is indeed the case (See Figure 3-4 and Figure 3-6).

Fully Recurrent Neural Networks with Shared and Non-shared Weights Although a RNN is usually implemented with shared weights across time, it is however possible to unshare the weights and use an independent set of weights at every time t. For practical applications, whenever one can have a initial $t = 0$ and enumerate all possible ts, a RNN with non-shared weights should be feasible, similar to the time-specific batch normalization described in 3.3. The results of 2-state fully recurrent neural networks with shared and non-shared weights are shown in Figure 3-4 (C).

The Effect of Readout Time In visual cortex, useful information increases as time proceeds from the onset of the visual stimuli. This suggests that recurrent system might have better representational power as more time is allowed. We tried training and testing 2-state ResNet and FRNN with various readout time (i.e., unrolling depth, see Section 3.3) and observe similar effects. Results are shown in Figure 3-4 (D) and (E). The difference from Figure 3-5 (A) is that training and testing readout times are the same in this experiment.

Larger Models With More States

The results of 3-state and 4-state FRNNs are shown in Figure 3-3 and Table 3.1. 3-state models seem to generally outperform 2-state ones. This is expected since more parameters are introduced. We compare the 3-state models with existing best models on CIFAR-10 in Table 3.1.Next, for computational efficiency, we tried only allowing each state to have transitions to adjacent states and to itself by disabling bypass connections (e.g., Figure 3-3 top). In this case, the number of transitions scales linearly as the number of states increases, instead of quadratically. This setting performs well with 3-state networks and slightly less well with 4-state networks (perhaps as a result of small feature/parameter sizes). Finally, for 4-state fully recurrent networks, the models tend to become overly computationally heavy if we train it with large t or large number of feature maps. With small t and feature maps, we have not achieved better performance than 3-state networks. For experiments in this subsection, we choose a moderately deep pre-net of three 3x3 convolutional layers to model the layers between retina and V1: Conv-BN-ReLU-Conv-BN-ReLU-Conv. This is not essential but outperforms shallow pre-net slightly (within 1% validation error).

Generalization Across Readout Time As a RNN, our model supports training and testing with different readout times. Based on our theoretical analyses in Section 3.2.2, the representation is usually not guaranteed to converge when running a model with time $t \to \infty$. Nevertheless, the model exhibits good generalization over time. Results are shown in Figure 3-5.

Experiments on ImageNet We also evaluated weight sharing on ImageNet 1000-way classification task. The results are shown in Figure 3-6.

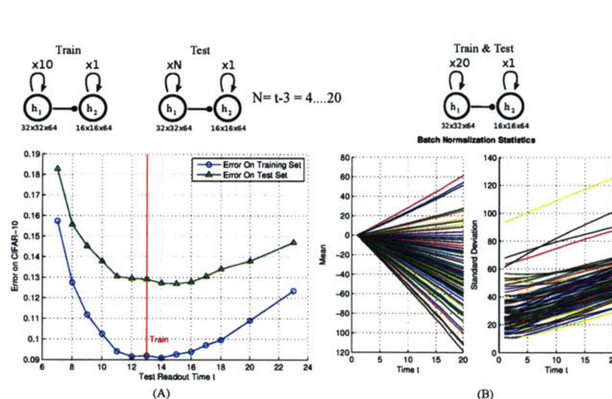
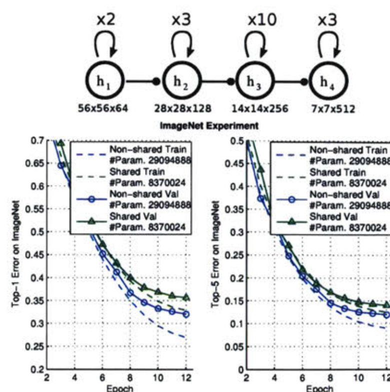



Figure 3-5: (A) Training and testing with different readout time. A 2-state ResNet with shared weights is trained with readout time t=13 (i.e., 1st state unrolling factor N=t-3=10) and tested with t from 7 to 23. The model can generalize well to readout time that are not trained with. There even seems to be tiny performance improvement (on test set) when testing with slightly larger readout time than training. (B) Biological Plausibility of Time-specific Batch Normalization (TSBN): we show the normalization statistics of a batch normalization module in the model in part (B) (which was trained with t=23). Each line represents the statistics of a feature channel over time. TSBN seems to implement a simple decay, thus it may be biologically-plausible.

Figure 3-6: ImageNet experiment with a 4-state ResNet with non-shared vs. shared weights. Train: training error. Val: validation error. No scaling augmentation is done. Performances are all based on single-crop. We trained the models with 12 epochs and logarithmically decaying learning rates (i.e., logspace(-1, -4, N) in Matlab, where N=12 is the number of epochs). The logarithmic learning rates make the model converge much faster than the usual three stage learning rates (i.e., dividing by 10 every some epochs), thus the models have already converged well and the performance differences are representative.

3.6 Discussion

When is Weight Sharing Good/Bad? We observe that (e.g., in Figure 3-4 A), sharing weights over time in the first stage of ResNet always improves performance. However, sharing weights in the later stages does sometimes slightly lower performance. We conjecture that whenever our prior — the transition function (e.g., BN-ReLU-Conv x2) is "similar" to the underlying recurrent function used by visual cortex, weight sharing will be beneficial. This would imply that our transition function happen to better model recurrent computations in early visual areas than later stages.

Psychophysics Support: After our work, [13] reported that while higher layers of deep networks enjoy better performance on a visual recognition task, only intermediate (instead of last) layers agree best with human rapid predictions (t= 50 to 500ms). Given that a deep network's output layer is trained to match human predictions at $t = \infty$. It is interesting to see an intermediate layer matches best an intermediate time, consistent with what we predict.

3.7 Future Directions

1. The dark secret of Deep Networks: trying to imitate Recurrent Shallow Networks? The results of this work lead to the following conjecture: the effectiveness of most of the deep feedforward neural networks, including but not limited to ResNet, can be attributed to their ability to approximate recurrent computations that are prevalent in most tasks with larger t than shallow feedforward networks. This may offer a new perspective on the theoretical pursuit of the long-standing question "why is deep better than shallow" [56, 53].

2. Conjecture about Cortex and Recurrent Computations in Cortical Areas: Most of the models of cortex that led to the Deep Convolutional architectures and followed them – such as the Neocognitron [18], HMAX [68] and more recent models [86] – have neglected the layering in each cortical area and the feedforward and recurrent connections within each area and between them. They also neglected the time evolution of selectivity and invariance in each of the areas. Our proposal makes several interesting predictions. Each cortical area would correspond to a recurrent network and thus to a system with a temporal dynamics even for flashed inputs; with increasing time one expects asymptotically better performance; masking with a mask an input image flashed briefly should disrupt recurrent computations in each area; performance should increase with time even without a mask for briefly flashed images; the cortex and each of its component areas are RNNs and, unlike relatively shallow feedforward networks, are computationally as powerful as universal Turing machine [73].

Chapter 4

Biologically-Plausible Normalizations For Recurrent and Online Learning

4.1 Introduction

Batch Normalization [35] (BN) is a highly effective technique for speeding up convergence of feedforward neural networks. It enabled recent development of ultra-deep networks [24] and some biologically-plausible variants of backpropagation [49]. However, despite its success, there are two major learning scenarios that cannot be handled by BN: (1) online learning and (2) recurrent learning.

For the second scenario recurrent learning, previous chapter (i.e., [50]) and [11] independently proposed "time-specific batch normalization": different normalization statistics are used for different timesteps of an RNN. Although this approach works well in many experiments in the previous chapter (i.e., [50]) and [11], it is far from perfect due to the following reasons: First, it does not work with small mini-batch or online learning. This is similar to the original batch normalization, where enough samples are needed to compute good estimates of statistical moments. Second, it requires sufficient training samples for every timestep of an RNN. It is not clear how to generalize the model to unseen timesteps. Finally, it is not biologically-plausible. While homeostatic plasticity mechanisms (e.g., Synaptic Scaling) [83, 79, 82] are good biological candidates for BN, it is hard to imagine how such normalizations can behave

Approach	FF & FC	FF & Conv	Rec & FC	Rec & Conv	Online Learning	Small Batch	All Combined
Original Batch Normalization(BN)	✓	✓	✗	✗	✗	Suboptimal	✗
Time-specific BN	✓	✓	Limited	Limited	✗	Suboptimal	✗
Layer Normalization	✓	✗*	✓	✗*	✓	✓	✗*
Streaming Normalization	✓	✓	✓	✓	✓	✓	✓

Table 4.1: An overview of normalization techiques for different tasks. ✓: works well. ✗: does not work well. FF: Feedforward. Rec: Recurrent. FC: Fully-connected. Conv: convolutional. Limited: time-specific BN requires recording normalization statistics for each timestep and thus may not generalize to novel sequence length. *Layer normalization does not fail on these tasks but perform significantly worse than the best approaches.

differently for each timestep. Recently, Layer Normalization (LN) [3] was introduced to solve some of these issues. It performs well in feedforward and recurrent settings when only fully-connected layers are used. However, it does not work well with convolutional networks. A summary of normalization approaches in various learning scenarios is shown in Table 4.1.

We note that different normalization methods like BN and LN can be described in the same framework detailed in Section 4.3. This framework introduces Sample Normalization and General Batch Normalization (GBN) as generalizations of LN and BN, respectively. As their names imply, they either collect normalization statistics from a single sample or from a mini-batch. We explored many variants of these models in the experiment section.

A natural and biologically-inspired extension of these methods would be Streaming Normalization: normalization statistics are collected in an online fashion from all previously seen training samples (and all timesteps if recurrent). We found numerous advantages associated with this approach: 1. it naturally supports pure online learning or learning with small mini-batches. 2. for recurrent learning, it is more biologically-plausible since a unique set of normalization statistics is maintained for all timesteps. 3. it performs well out of the box in all learning scenarios (e.g., online learning, batch learning, fully-connected, convolutional, feedforward, recurrent and mixed — recurrent and convolutional). 4. it offers a new direction of designing normalization algorithms,

since the idea of maintaining online estimates of normalization statistics is independent from other design choices, and as a result any existing algorithm (e.g., in Figure 4-1 C and D) can be extended to a "streaming" setting.

We also propose Lp normalization: instead of normalizing by the second moment like BN, one can normalize by the p-th root of the p-th absolute moment (See Section 4.3.4 for details). In particular, L1 normalization works as well as the conventional approach (i.e., L2) in almost all learning scenarios. Furthermore, L1 normalization is easier to implement: it is simply the average of absolute values. We believe it can be used to simplify and speed up BN and our Streaming Normalization in GPGPU, dedicated hardware or embedded systems. L1 normalization may also be more biologically-plausible since the gradient of the absolute value is trivial to implement, even for biological neurons.

In the following section, we introduce a simple training scheme, a minor but necessary component of our formulation.

4.2 Online and Batch Learning with "Decoupled Accumulation and Update"

Although it is not our main result, we discuss a simple but to the best of our knowledge less explored[1] training scheme we call a "Decoupled Accumulation and Update" (DAU).

Conventionally, the weights of a neural network are updated every mini-batch. So the accumulation of gradients and weights update are coupled. We note that a general formulation would be that while for every mini-batch the gradients are still accumulated, one does not necessarily update the weights. Instead, the weights are updated every n mini-batches. The gradients are cleared after each weight update. Two parameters characterize this procedure: Samples per Batch (S/B) m and Batch per Update (B/U) n. In conventional training, $n = 1$.

Note that **this procedure is similar to but distinct from simply training larger mini-batches** since every mini-batch arrives in a purely online fashion so one

[1] We are not aware of this approach in the literature. If there is, please inform us.

cannot look back into any previous mini-batches. For example, if batch normalization is present, performing this procedure with m S/B and n B/U is different from that with $m * n$ S/B and 1 B/U.

If $m = 1$, it reduces to a pure online setting where one sample arrives at a time. The key advantage of this proposal over conventional training is that we explicitly require less frequent (but more robust) weight updates. The memory requirement (in additional to storing the network weights) is storing m samples and related activations.

This training scheme generalizes the conventional approach, and we found that merely applying this approach greatly mitigated (although not completely solved) the catastrophic failure of training batch normalization with small mini-batches (See Figure 4-4). Therefore, we will use this formulation throughout this work.

We also expect this scheme to benefit learning sequential recurrent networks with varying input sequence lengths. Sometimes it is more efficient to pack in a mini-batch training samples with the same sequence length. If this is the case, our approach predicts that it would be desirable to process multiple such mini-batches with varying sequence lengths before a weight update. Accumulating gradients from training different sequence lengths should provide a more robust update that works for different sequence lengths, thus better approximating the true gradient of the dataset.

In Streaming Normalization with recurrent networks, it is often beneficial to learn with $n > 1$ (i.e., more than one batch per update). The first mini-batch collects all the normalization statistics from all timesteps so that later mini-batches are normalized in a more stable way.

4.3 A General Framework for Normalization

We propose a general framework to describe different normalization algorithms. A normalization can be thought of as a process of modifying the activation of a neuron using some statistics collected from some reference activations. We adopt three terms to characterize this process: A **Normalization Operation (NormOP)** is a function $N(x_i, s_i)$ that is applied to each neuron i to modify its value from x_i to $N(x_i, s_i)$, where

s_i is the **Normalization Statistics (NormStats)** for this neuron. **NormStats** is any data required to perform NormOP, collected using some function $s_i = S(R_i)$, where R_i is a set of activations (could include x_i) called **Normalization Reference (NormRef)**. Different neurons may or may not share NormRef and NormStats.

This framework captures many previous normalizations algorithms as special cases. For example, the original Batch Normalization (BN) [35] can be described as follows: the NormRef for each neuron i is all activations in the same channel of the same layer and same batch (See Figure 4-1 C for illustration). The NormStats are $S_i = \{\mu_i, \sigma_i\}$ — the mean and standard deviation of the activations in NormRef. The NormOp is $\mathrm{N}(x_i, \{\mu_i, \sigma_i\}) = \frac{x_i - \mu_i}{\sigma_i}$

For the recent Layer Normalization [3], NormRef is all activations in the same layer (See Figure 4-1 C). The NormStats and NormOp are the same as BN.

We group normalization algorithms into three categories:

1. Sample Normalization (Figure 4-1 C): NormStats are collected from one sample.
2. General Batch Normalization (Figure 4-1 D): NormStats are collected from all samples in a mini-batch.
3. Streaming Normalization: NormStats are collected in an online fashion from all pass training samples.

In the following sections, we detail each algorithm and provide pseudocode.

4.3.1 Sample Normalization

Sample normalization is the simplest category among the three. NormStats are collected only using the activations in the current layer of current sample. The computation is the same at training and test times. It handles online learning naturally. Layer Normalization [3] is an example of this category. More examples are shown in Figure 4-1 C.

The pseudocode of forward and backpropagation is show in Algorithm 1 and Algorithm 2.

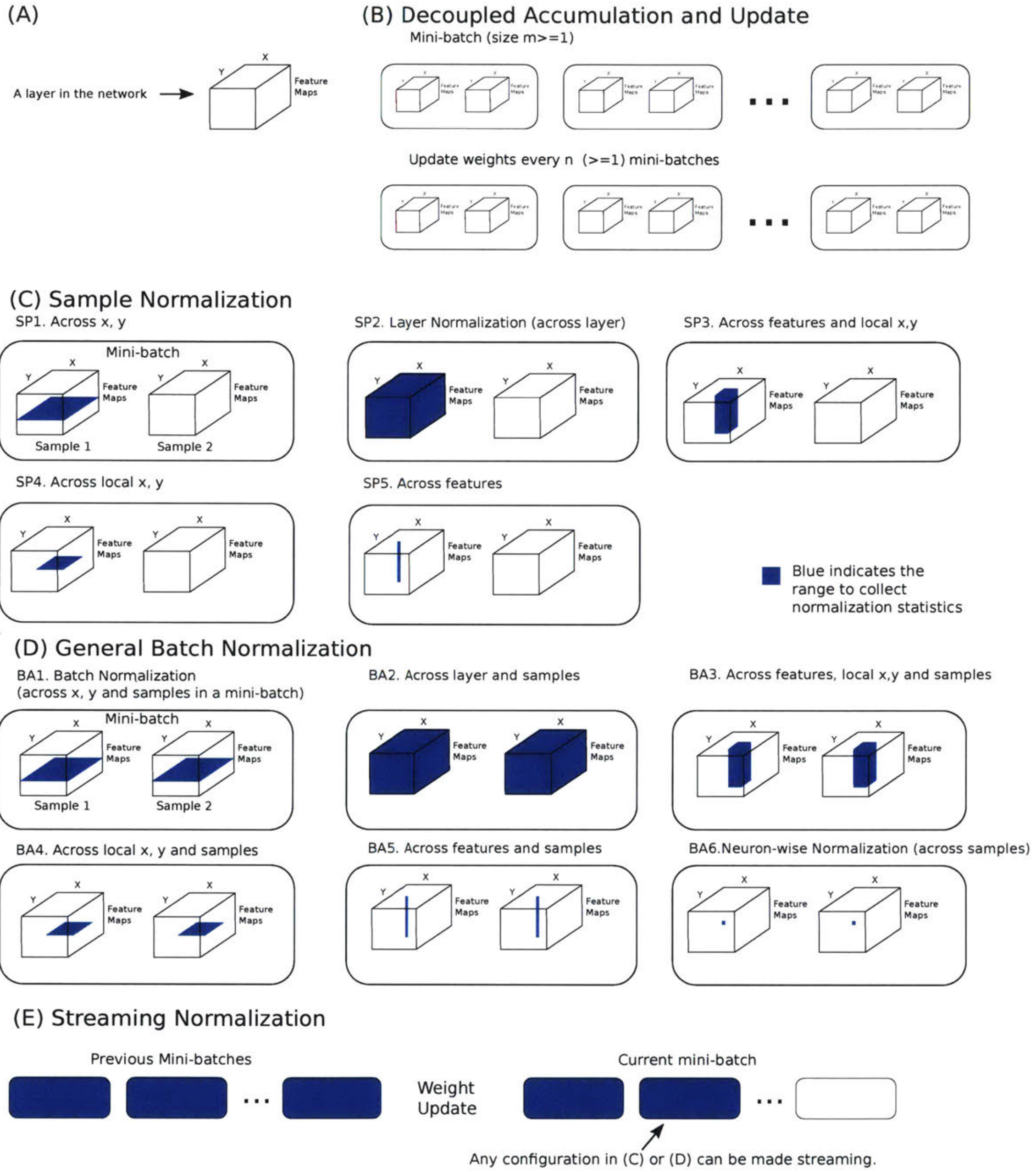


Figure 4-1: A General Framework of Normalization. A: the input to convolutional layer is a 3D matrix consists of 3 dimensions: x (image width), y (image height) and features/channel. For fully-connected layers, x=y=1. B: training with decoupled accumulation and update. C: Sample Normalization. D: General Batch Normalization. E: Streaming Normalization

Algorithm 1 Sample Normalization Layer: Forward

Require: layer input **x** (a sample), NormOP N(.,.), function S(.) to compute NormStats for every element of **x**

Ensure: layer output **y** (a sample)

s = S(x)

y = N(x,s)

Algorithm 2 Sample Normalization Layer: Backpropagation

Require: $\frac{\partial E}{\partial y}$ (a sample) where E is objective, layer input **x**, NormOP N(.,.), function S(.) to compute NormStats for every element of **x**

Ensure: $\frac{\partial E}{\partial x}$ (a sample)

$\frac{\partial E}{\partial x}$ can be calculated using chain rule. Detail omitted.

4.3.2 General Batch Normalization

In General Batch Normalization (GBN), NormStats are collected in some way using the activations from all samples in a training mini-batch. Note that one cannot really compute the batch NormStats at test time since test samples should be handled independently from each other, instead in a batch. To overcome this, one can simply compute running estimates of training NormStats and use them for testing (e.g., the original Batch Normalization [35] computes moving averages).

More examples of GBN are shown in Figure 4-1 D. The pseudocode is shown in Algorithm 3 and 4.

Algorithm 4 General Batch Normalization Layer: Backpropagation

Require: $\frac{\partial E}{\partial y}$ (a mini-batch) where E is objective, layer input **x** (a mini-batch), NormOP N(.,.), function S(.) to compute NormStats for every element of **x**, running estimates of NormStats $\hat{s}$, function F to update $\hat{s}$.

Ensure: $\frac{\partial E}{\partial x}$ (a mini-batch)

$\frac{\partial E}{\partial x}$ can be calculated using chain rule. Detail omitted.

Algorithm 3 General Batch Normalization Layer: Forward

Require: layer input $\mathbf{x}$ (a mini-batch), NormOP N(.,.), function S(.) to compute NormStats for every element of $\mathbf{x}$, running estimates of NormStats $\hat{s}$, function F to update $\hat{s}$.
Ensure: layer output $\mathbf{y}$ (a mini-batch), running estimates of NormStats $\hat{s}$
if *training* **then**
  s = S(x)
  $\hat{s}$ = F($\hat{s}$,s)
  y = N(x,s)
else {*testing*}
  y = N(x,$\hat{s}$)
end if

4.3.3 Streaming Normalization

Finally, we present the main results of this work. We propose Streaming Normalization: NormStats are collected in an online fashion from all past training samples. The main challenge of this approach is that it introduces infinitely long dependencies throughout the neuron's activation history — every neuron's current activation depends on all previously seen training samples.

It is intractable to perform exact backpropagation on this dependency graph for the following reasons: there is no point backpropagating beyond the last weight update (if any) since one cannot redo the weight update. This, on the other hand, would imply that one cannot update the weights until having seen all future samples. Even backpropagating within a weight update (i.e., the interval between two weight updates, consisting of n mini-batches) turns out to be problematic: one is usually not allowed to backpropagate to the previous several mini-batches since they are discarded in many practical settings.

For the above reasons, we abandoned the idea of performing exact backpropagation for streaming normalization. Instead, we propose two simple heuristics: Streaming NormStats and Streaming Gradients. They are discussed below and the pseudocode is shown in Algorithm 5 and Algorithm 6.

Streaming NormStats

Streaming NormStats is a natural requirement of Streaming Normalization, since NormStats are collected from all previously seen training samples. We maintain a structure/table H_1 to keep all the information needed to generate a good estimate of NormStats. and update it using a function F everytime we encounter a new training sample. Function F also generates the current estimate of NormStats called $\hat{s}$. We use $\hat{s}$ to normalize instead of s. See Algorithm 5 for details.

There could be many potential designs for F and H_1, and in our experiments we explored a particular version: we compute two sets of running estimates to keep track of the long-term and short-term NormStats: $H_1 = \{\hat{s}_{long}, \hat{s}_{short}, counter\}$.

- Short-term NormStats $\hat{s}_{short}$ is the **exact average** of NormStats s since **the last weight update**. The *counter* keeps track of the number of times a different s is encountered to compute the exact average of s.

- Long-term NormStats $\hat{s}_{long}$ is an **exponential average** of $\hat{s}_{short}$ since **the beginning of training**.

Whenever the weights of the network is updated: $\hat{s}_{long} = \kappa_1 * \hat{s}_{long} + \kappa_2 * \hat{s}_{short}$, *counter* is reset to 0 and $\hat{s}_{short}$ is set to empty. In our experiments, $\kappa_1 + \kappa_2 = 1$ so that an exponential average of $\hat{s}_{short}$ is maintained in $\hat{s}_{long}$.

In our implementation, before testing the model, the last weight update is NOT performed, and $\hat{s}_{short}$ is also NOT cleared, since $\hat{s}_{short}$ is needed for testing [2]

In addition to updating H_1 in the way described above, the function F also computes $\hat{s} = \alpha_1 \hat{s}_{long} + \alpha_2 \hat{s}_{short}$. Finally, $\hat{s}$ is used for normalization.

Streaming Gradients

We maintain a structure/table H_2 to keep all the information needed to generate a good estimate of gradients of NormStats and update it using a function G everytime

[2] It also possible to simply store the last $\hat{s}$ computed in training for testing, instead of storing $\hat{s}_{short}$ and re-computing $\hat{s}$ in testing. These two options are mathematically equivalent.

backpropagation reaches this layer. Function G also generates the current estimate of gradients of NormStats called $\widehat{\frac{\partial E}{\partial \hat{s}}}$. We use $\widehat{\frac{\partial E}{\partial \hat{s}}}$ for further backpropagation instead of $\frac{\partial E}{\partial \hat{s}}$. See Algorithm 6 for details.

Again, there could be many potential designs for G and H_2, and we explored a particular version: we compute two sets of running estimates to keep track of the long-term and short-term gradients of NormStats: $H_2 = \{\hat{g}_{long}, \hat{g}_{short}, counter\}$.

- Short-term Gradients of NormStats $\hat{g}_{short}$ is the **exact average** of gradients of NormStats $\frac{\partial E}{\partial \hat{s}}$ since **the last weight update**. The *counter* keeps track of the number of times a different $\frac{\partial E}{\partial \hat{s}}$ is encountered to compute the exact average of $\frac{\partial E}{\partial \hat{s}}$.

- Long-term Gradients of NormStats $\hat{g}_{long}$ is an **exponential average** of $\hat{g}_{short}$ since **the beginning of training**.

Whenever the weights of network is updated: $\hat{g}_{long} = \kappa_3 * \hat{g}_{long} + \kappa_4 * \hat{g}_{short}$, $counter$ is reset to 0 and $\hat{g}_{short}$ is set to empty. In our experiments, $\kappa_3 + \kappa_4 = 1$ so that an exponential average of $\hat{g}_{short}$ is maintained in $\hat{g}_{long}$.

In addition to updating H_2 in the way described above, the function G also computes $\widehat{\frac{\partial E}{\partial \hat{s}}} = \beta_1 \hat{g}_{long} + \beta_2 \hat{g}_{short} + \beta_3 \frac{\partial E}{\partial \hat{s}}$. Finally, $\widehat{\frac{\partial E}{\partial \hat{s}}}$ is used for further backpropagation.

Algorithm 5 Streaming Normalization Layer: Forward

Require: layer input **x** (a mini-batch), NormOP N(.,.), function S(.) to compute NormStats for every element of **x**, running estimates of NormStats and/or related information packed in a structure/table H_1, function F to update H_1 and generate current estimate of NormStats ŝ.

Ensure: layer output **y** (a mini-batch) and H_1 (it is stored in this layer, instead of feeding to other layers), always maintain the latest ŝ in case of testing

if *training* **then**
  s = S(x)
  $\{H_1$,ŝ$\}$ = F(H_1,s)
  y = N(x,ŝ)
else $\{testing\}$
  y = N(x,ŝ)
end if

Algorithm 6 Streaming Normalization Layer: Backpropagation

Require: $\frac{\partial E}{\partial y}$ (a mini-batch) where E is objective, layer input $\mathbf{x}$ (a mini-batch), NormOP N(.,.), function S(.) to compute NormStats for every element of $\mathbf{x}$, running estimates of NormStats $\hat{\mathrm{s}}$, running estimates of gradients and/or related information packed in a structure/table H_2, function G to update H_2 and generate the current estimates of gradients of NormStats $\widehat{\frac{\partial E}{\partial \hat{s}}}$.

Ensure: $\frac{\partial E}{\partial x}$ (a mini-batch) and H_2 (it is stored in this layer, instead of feeding to other layers)

$\frac{\partial E}{\partial \hat{s}}$ is calculated using chain rule.

$\{H_2,\widehat{\frac{\partial E}{\partial \hat{s}}}\} = \mathrm{G}(H_2,\frac{\partial E}{\partial \hat{s}})$

Use $\widehat{\frac{\partial E}{\partial \hat{s}}}$ for further backpropagation, instead of $\frac{\partial E}{\partial \hat{s}}$

$\frac{\partial E}{\partial x}$ is calculated using chain rule.

A Summary of Streaming Normalization Design and Hyperparameters

The NormOp used in this work is $\mathrm{N}(x_i,\{\mu_i,\sigma_i\}) = \frac{x_i-\mu_i}{\sigma_i}$, the same as what is used by BN. The NormStats are collected using one of the Lp normalization schemes described in Section 4.3.4.

With our particular choices of F, G, H_1 and H_2, the following hyperparameters uniquely characterize a Streaming Normalization algorithm: α_1, α_2, $\beta_1, \beta_2, \beta_3$, $\kappa_1, \kappa_2, \kappa_3, \kappa_4$, samples per batch m, batches per update n and a choice of mini-batch NormRef (i.e., SP1-SP5, BA1-BA6 in Figure 4-1 C and D).

Unless mentioned otherwise, we use BA1 in Figure 4-1 D as the NormRef throughout this work. We also demonstrate the use of other NormRefs (BA4 and BA6) in the Appendix Figure A1.

An important special case: If $n = 1, \alpha_1 = 0, \alpha_2 = 1, \beta_1 = 0, \beta_2 = 0, \beta_3 = 1$, we ignore all NormStats and gradients beyond the current mini-batch. **The algorithm reduces to exactly the GBN algorithm.** So Streaming Normalization is strictly a generalization of GBN and thus also captures the original BN [35] as a special case. Recall from Section 4.3.3 that $\hat{s}_{short}$ is NOT cleared before testing the model. Thus for testing, NormStats are inherited from the last training mini-batch. It works well in practice.

Unless mentioned otherwise, we set $\alpha_1 = \kappa_1 = \kappa_3$, $\alpha_2 = \kappa_2 = \kappa_4$, $\kappa_1 + \kappa_2 = 1$, $\kappa_3 + \kappa_4 = 1$, $\alpha_1 + \alpha_2 = 1$, $\beta_1 + \beta_2 + \beta_3 = 1$. We leave it to future research to explore

different choices of hyperparameters (and perhaps other F and G).

Implementation Notes

One minor drawback of not performing exact backpropagation is that it may break the gradient check of the entire model. One solution coulde be: (1) perform gradient check of the model without SN and then add a correctly implemented SN. (2) make sure the SN layer is correctly coded (e.g., by reducing it to standard BN using the hyperparameters discussed above and then perform gradient check).

4.3.4 Lp Normalization: Calculating NormStats with Different Orders of Moments

Let us discuss the function $S(.)$ for calculating NormStats. There are several choices for this function. We propose **Lp normalization**. It captures the previous mean-and-standard-deviation normalization as a special case.

First, mean μ is always calculated the same way — the average of the activations in NormRef. The **divisive factor** σ, however, can be calculated in several different ways. In **Lp Normalization**, σ is chosen to be the p-th root of the p-th **Absolute Moment**.

Here the **Absolute Moment** of a distribution $P(x)$ about a point c is:

$$\int |x - c|^p P(x)dx \tag{4.1}$$

and the discrete form is:

$$\frac{1}{N}\sum_{i=1}^{N} |x_i - c|^p \tag{4.2}$$

Lp Normalization can be performed with three settings:

- **Setting A:** σ is the p-th root of the p-th absolute moment of all activations in NormRef with c being the mean μ of NormRef.
- **Setting B:** σ is the p-th root of the p-th absolute moment of all activations in NormRef with c being the running estimate $\hat{\mu}$ of the average.

- **Setting C:** σ is the p-th root of the p-th absolute moment of all activations in NormRef with c being 0.

Most of these variants have similar performance but some are better in some situations.

We call it Lp normalization since it is similar to the norm in the L^p space.

Setting B and C are better for online learning since A will give degenerate result (i.e., $\sigma = 0$) when there is only one sample in NormRef. Empirically, when there are enough samples in a mini-batch, A and B perform similarly.

We discuss several important special cases:

Special Case A-2: setting A with n=2, σ is the standard deviation (square root of the 2nd moment) of all activations in NormRef. This setting is what is used by Batch Normalization [35] and Layer Normalization [3].

Special Case p=1: Whenever p=1, σ is simply the average of absolute values of activations. This setting works virtually the same as p=2, but is much simpler to implement and faster to run. It might also be more biologically-plausible, since the gradient computations are much simpler.

4.3.5 Separate Learnable Bias and Gain Parameters

The original Batch Normalization [35] also learns a bias and a gain (i.e., shift and scaling) parameter for each feature map. Although not usually done, but clearly these shift and scaling operations can be completely separated from the normalization layer. We implemented them as a separate layer following each normalization layer. These parameters are learned in the same way for all normalization schemes evaluated in this work.

4.4 Generalization to Recurrent Learning

In this section, we generalize Sample Normalization, General Batch Normalization and Streaming Normalization to recurrent learning. The difference between recurrent learning and feedforward learning is that for each training sample, every hidden layer of the network receives t activations $h_1, , ..., h_t$, instead of only one.

4.4.1 Recurrent Sample Normalization

Sample Normalization naturally generalizes to recurrent learning since all NormStats are collected from the current layer of the current timestep. Training and testing algorithms remain the same.

4.4.2 Recurrent General Batch Normalization (RGBN)

The generalization of GBN to recurrent learning is the same as what was proposed by the previous chapter [50] and [11]. The training procedure is exactly the same as before (Algorithm 3 and Algorithm 4). For testing, one set of running estimates of NormStats is maintained for each timestep.

Another way of viewing this algorithm is that the same GBN layers described in Algorithm 3 and 4 are used in the **unrolled** recurrent network. Each GBN layer in the unrolled network uses a different memory storage for NormStats.

4.4.3 Recurrent Streaming Normalization

Extending Streaming Normalization to recurrent learning is straightforward – we not only stream through all the past samples, but also through all past timesteps. Thus, we maintain a unique set of running estimates of NormStats for all timesteps. This is more biologically-plausible and memory efficient than the above approach (RGBN).

Again, another way of viewing this algorithm is that the same Streaming Normalization layers described in Algorithm 5 and 6 are used in the **original (instead of the unrolled)** recurrent network. All unrolled versions of the same layer share running estimates of NormStats and other related data.

One caveat is that as time proceeds, the running estimates of NormStats are slightly modified. Thus when backpropagation reaches the same layer again, the NormStats are slightly different from the ones originally used for normalization. Empirically, it seems to not cause any problem on the performance. Training with "decoupled accumulation update" with Batches per Update (B/U) > 1 [3] can also mitigate this

[3] B/U=2 is often enough

problem, since it makes NormStats more stable over time.

4.5 Streaming Normalized RNN and GRU

In our character-level language modeling task, we tried Normalized Recurrent Neural Network (RNN) and Normalized Gated Recurrent Unit (GRU) [9]. Let us use Norm(.) to denote a normalization, which can be either Sample Normalization, General Batch Normalization or Streaming Normalization. A bias and gain parameter is also learned for each neuron. We use NonLinear to denote a nonlinear function. We used hyperbolic tangent (tanh) nonlinearity in our experiments. But we observed ReLU also works. h_t is the hidden activation at time t. x_t is the network input at time t. $W_.$ denotes the weights. $\odot$ denotes elementwise multiplication.

Normalized RNN

$$h_t = NonLinear(Norm(W_x * x_t) + Norm(W_h * h_{t-1})) \tag{4.3}$$

Normalized GRU

$$g_r = Sigmoid(Norm(W_{xr} * x_t) + Norm(W_{hr} * h_{t-1}))) \tag{4.4}$$

$$g_z = Sigmoid(Norm(W_{xz} * x_t) + Norm(W_{hz} * h_{t-1}))) \tag{4.5}$$

$$h_{new} = NonLinear(Norm(W_{xh} * x_t) + Norm(W_{hh} * (h_{t-1} \odot g_r))) \tag{4.6}$$

$$h_t = g_z \odot h_{new} + (1 - g_z) \odot h_{t-1} \tag{4.7}$$

4.6 Related Work

[43] and [2] used Batch Normalization (BN) in stacked recurrent networks, where BN was only applied to the feedforward part (i.e., "vertical" connections, input to each RNN), but not the recurrent part (i.e., "horizontal", hidden-to-hidden connections between timesteps). Previous chapter (i.e., [50]) and [11] independently proposed applying BN in recurrent/hidden-to-hidden connections of recurrent networks, but

separate normalization statistics must be maintained for each timestep. Previous chapter (i.e., [50]) demonstrated this idea with deep multi-stage fully recurrent (and convolutional) neural networks with ReLU nonlinearities and residual connections. [11] demonstrated this idea with LSTMs on language processing tasks and sequential MNIST. [3] proposed Layer Normalization (LN) as a simple normalization technique for online and recurrent learning. But they observed that LN does not work well with convolutional networks. [70] and [58] studied normalization using weight reparameterizations. An early work by [84] mathematically analyzed a form of online normalization for visual perception and adaptation.

4.7 Experiments

4.7.1 CIFAR-10 architectures and Settings

We evaluated the normalization techniques on CIFAR-10 dataset using feedforward fully-connected networks, feedforward convolutional network and a class of convolutional recurrent networks proposed by the previous chapter (i.e., [50]). The architectural details are shown in Figure 4-2. We train all models with learning rate 0.1 for 25 epochs and 0.01 for 5 epochs. Momentum 0.9 is used. We used MatConvNet [85] to implement our models.

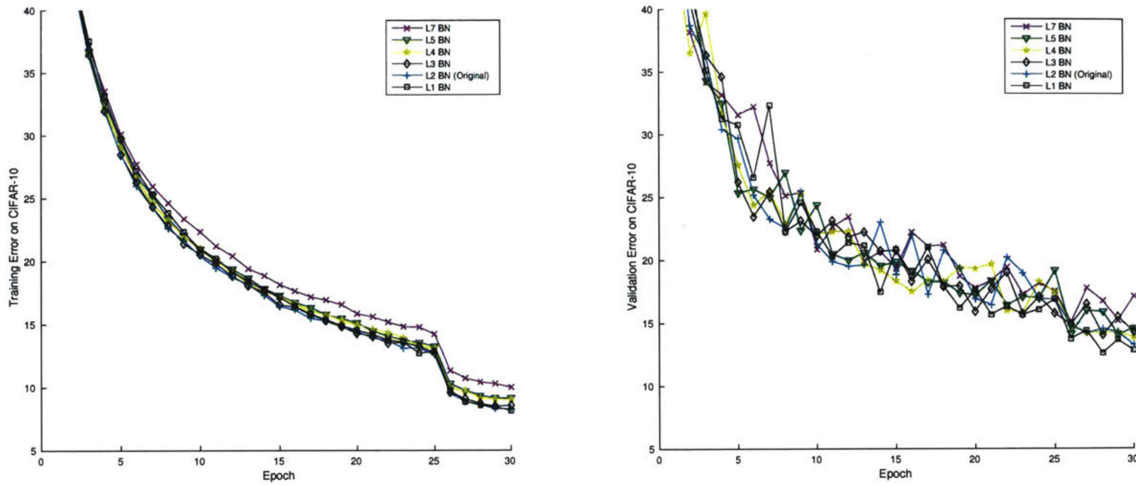


Figure 4-3: Lp Normalization. The architecture is a **feedforward and convolutional** network (shown in Figure 4-2 B). All statistical moments perform similarly well. L7 normalization is slightly worse.

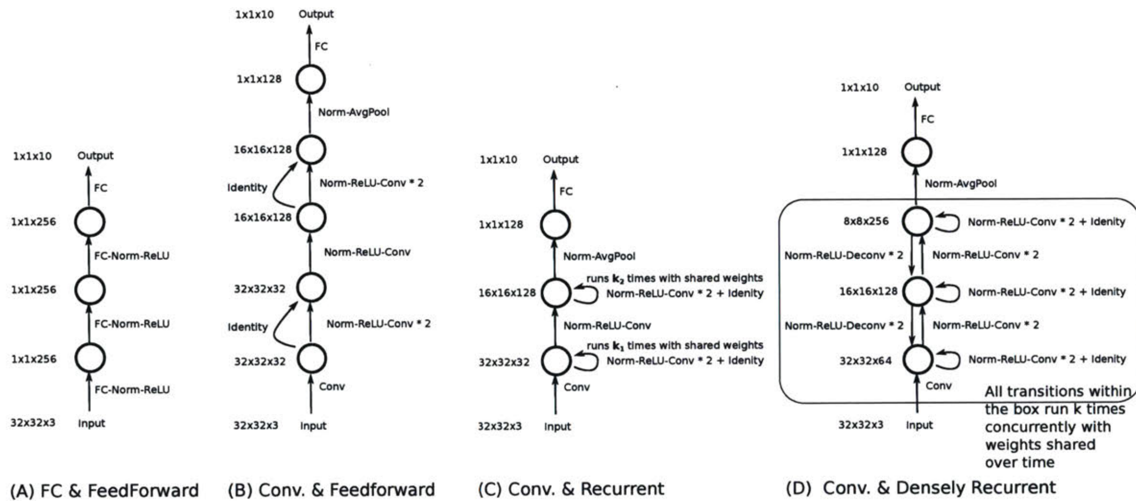


Figure 4-2: Architectures for CIFAR-10. Note that C reduces to B when $k_1 = k_2 = 1$.

4.7.2 Lp Normalization

We show BN with Lp normalization in Figure 4-3. Note that Lp normalization can be applied to Layer Normalization and all other normalizations show in 4-1 C and D. L1 normalization works as well as L2 while being simpler to implement and faster to compute.

4.7.3 Online Learning or Learning with Very Small Mini-batches

We perform online learning or learning with small mini-batches using architecture A in Figure 4-2.

Plain Mini-batch vs. Decoupled Accumulation and Update (DAU): We show in Figure 4-4 comparisons between conventional mini-batch training and Decoupled Accumulation and Update (DAU).

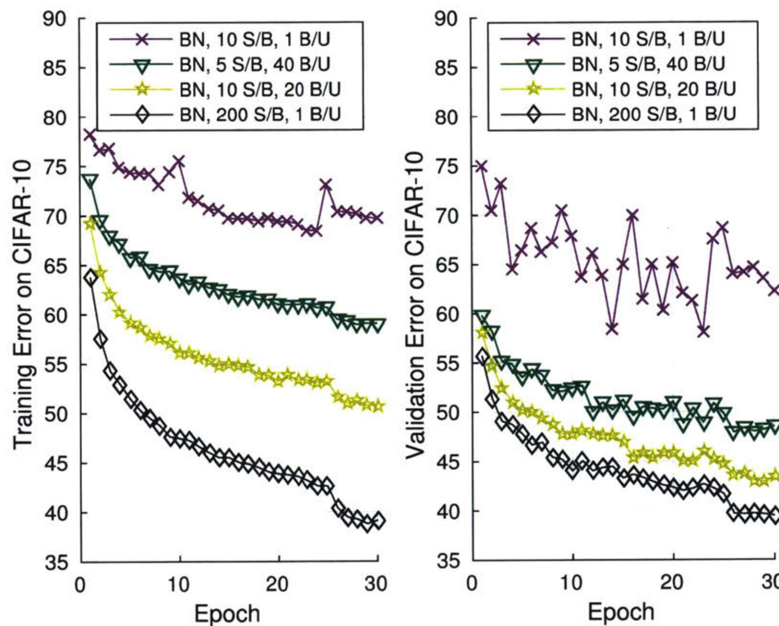


Figure 4-4: Plain Mini-batch vs. Decoupled Accumulation and Update (DAU). The architecture is a **feedforward and fully-connected** network (shown in Figure 4-2 A). S/B: Samples per Batch. B/U: Batches per Weight Update. We show there are significant performance differences between plain mini-batch (i.e., B/U=1) and Decoupled Accumulation and Update (DAU, i.e., B/U=n>1). DAU significantly improves the performance of BN with small number of samples per mini-batch (e.g., compare curve 1 with 3).

Layer Normalization vs. Batch Normalization vs. Streaming Normalization: We compare in Figure 4-5 Layer Normalization, Batch Normalization and Streaming Normalization with different choices of S/B and B/U.

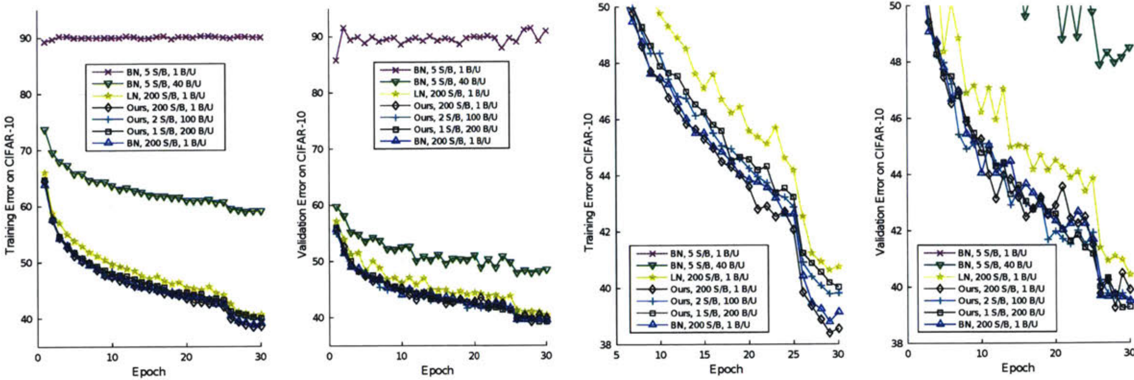


Figure 4-5: Different normalizations applied to a **feedforward and fully-connected** network (shown in Figure 4-2 A). The right two pannels are **zoomed-in versions** of the left two pannels. S/B: Samples per Batch. B/U: Batches per Weight Update. "Ours" refers to Streaming Normalization with "L1 norm" (Setting B with p=1 in Section 4.3.4) and $\alpha_1 = \beta_1 = 0.7$, $\alpha_2 = \beta_2 = 0.3$ and $\beta_3 = 0$ (see Section 4.3.3 for more details about hyperparameters). We show that our algorithm works with pure online learning (1 S/B) and tiny mini-batch (2 S/B), and it outperforms Layer Normalization. The choice of S/B does not matter for layer normalization since it processes samples independently.

4.7.4 Evaluating Variants of Batch Normalization

Feedforward Convolutional Networks: In Figure 4-6, we tested algorithms shown in Figure 4-1 C and D using the architecture B in Figure 4-2. We also show the performance of our Streaming Normalization for reference.

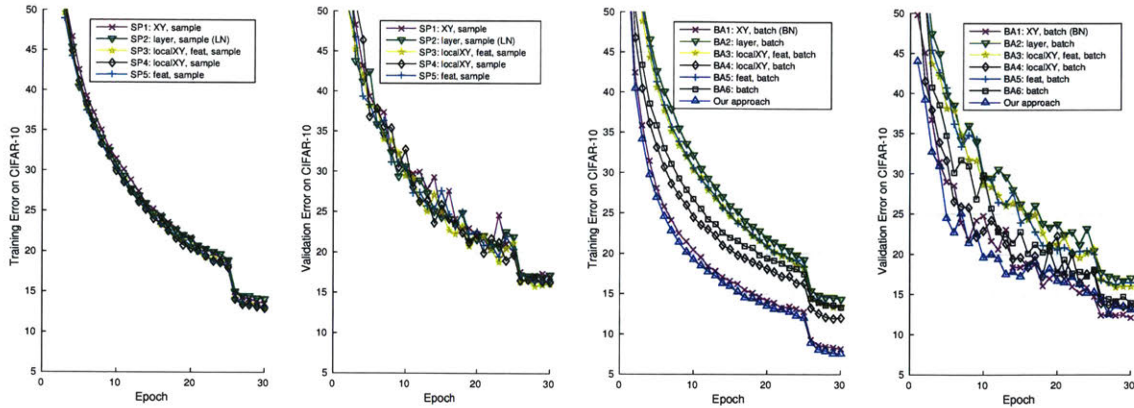


Figure 4-6: Different normalizations applied to a **feedforward and convolutional** network (shown in Figure 4-2 B). All models were trained with 32 Samples per Batch (S/B), 1 Batch per Update (B/U). "Our approach" refers to Streaming Normalization with "L2 norm" (Setting A with p=2 in Section 4.3.4) and $\alpha_1 = \beta_1 = 0.5$, $\alpha_2 = \beta_2 = 0.5$ and $\beta_3 = 0$ (see Section 4.3.3 for more details about hyperparameters). LN: Layer Normalization. Sample Normalizations (including LN) seem to all work similarly. It seems beneficial to normalize each channel/feature map separately (e.g., compare BA3 with BA4), like what BN does.

ResNet-like convolutional RNN: In Figure 4-7, we tested algorithms shown in Figure 4-1 C and D using the architecture C in Figure 4-2. We also show the performance of our Streaming Normalization for reference.

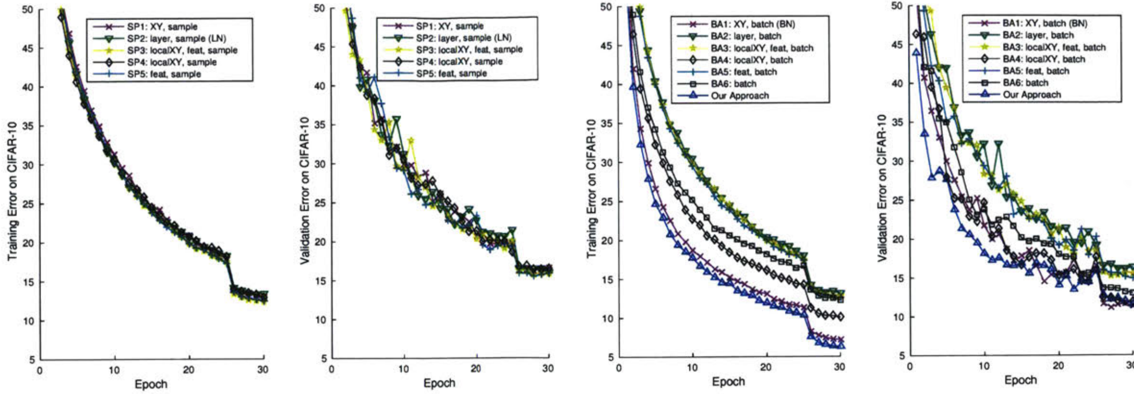


Figure 4-7: Different normalizations applied to a **recurrent and convolutional** network (Figure 4-2 C with $k_1 = 5$ and $k_2 = 1$). All models were trained with 32 Samples per Batch (S/B), 1 Batch per Update (B/U). "Our approach" refers to Streaming Normalization with "L2 norm" (Setting A with p=2 in Section 4.3.4) and $\alpha_1 = \beta_1 = 0.5$, $\alpha_2 = \beta_2 = 0.5$ and $\beta_3 = 0$ (see Section 4.3.3 for more details about hyperparameters). LN: Layer Normalization. Sample Normalizations (including LN) seem to all work similarly. It seems beneficial to normalize each channel/feature map separately (e.g., compare BA3 with BA4), like what BN does.

Densely Recurrent Convolutional Network: In Figure 4-8, we tested Time-Specific Batch Normalization and Streaming Normalization on the architecture D in Figure 4-2.

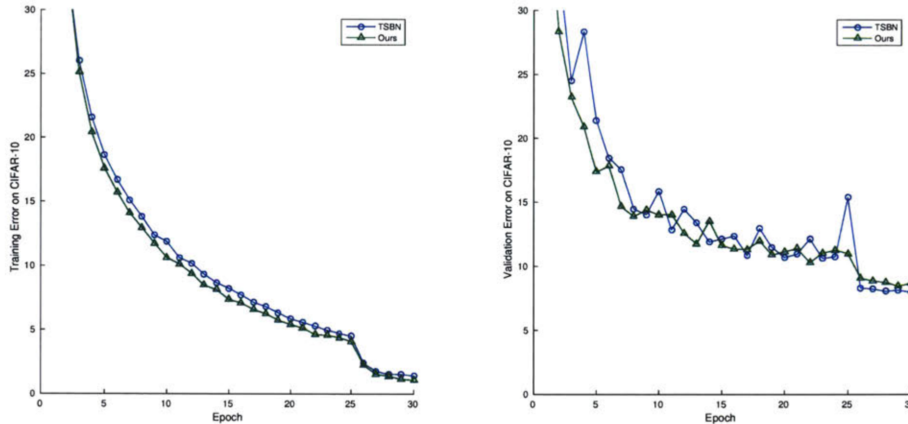


Figure 4-8: Time-specific Batch Normalization (TSBN) and Streaming Normalization applied to a **densely recurrent and convolutional** network (Figure 4-2 D with $k = 5$). "Ours" refers to Streaming Normalization with "L2 norm" (Setting B with p=2 in Section 4.3.4), 32 Samples per Batch (S/B) and 2 Batches per Update (B/U) and $\alpha_1 = \beta_1 = 0.7$, $\alpha_2 = 0.3, \beta_2 = 0$ and $\beta_3 = 0.3$ (see Section 4.3.3 for more details about hyperparameters). Sometimes for recurrent networks, B/U > 1 is preferred, since the first mini-batch collects NormStats from all timesteps so that the second mini-batch is normalized in a more stable way. TSBN was trained with 64 S/B, 1 B/U (32 S/B, 2 B/U would give similar performance, if not worse). Streaming Normalization has similar performance to TSBN but does not require storing different NormStats for each timestep.

4.7.5 More Experiments on Streaming Normalization

In Figure 4-9, we compare the performances of original BN (i.e., NormStats shared over time), time-specific BN, layer normalization and streaming normalization on a recurrent and convolutional network shown in Figure 4-2 C.

We evaluated different choices of hyperparameter β_1,β_2 and β_3 in Figure 4-10.

4.7.6 Recurrent Neural Networks for Character-level Language Modeling

We tried our simple implementations of vanilla RNN and GRU described in Section 4.5. The RNN and GRU both have 1 hidden layer with 100 units. Weights are updated using the simple Manhattan update rule described in [49]. The models were

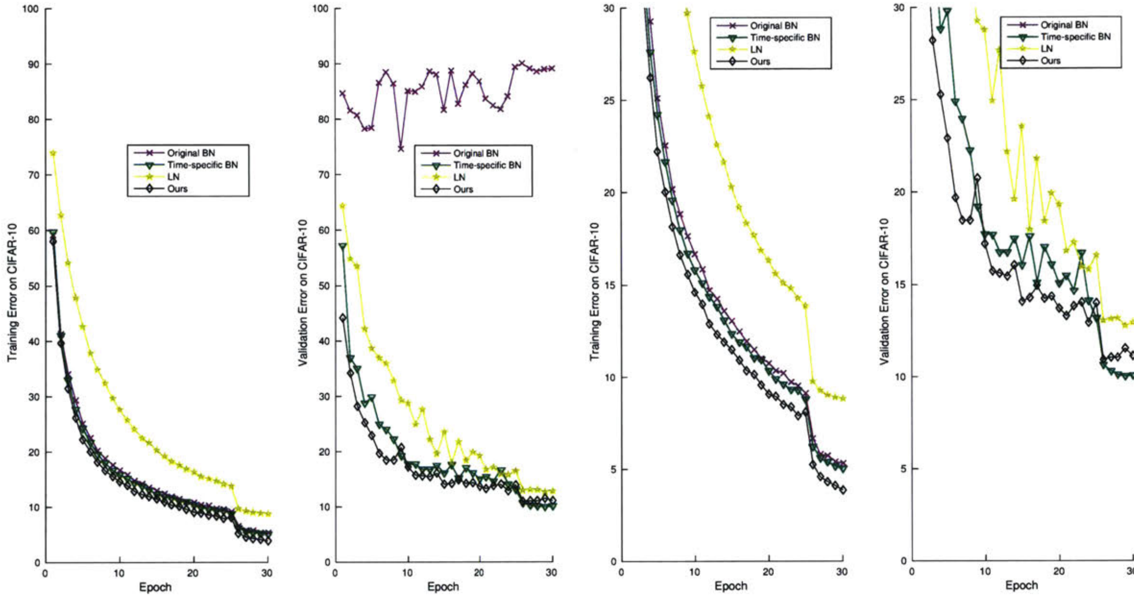


Figure 4-9: Different normalizations applied to a **recurrent and convolutional** network (shown in Figure 4-2 C with unrolling parameters $k_1 = k_2 = 5$). The right two pannels are **zoomed-in versions** of the left two pannels. "Ours" refers to Streaming Normalization with "L2 norm" (Setting B with p=2 in Section 4.3.4), 32 Samples per Batch (S/B), 2 Batches per Update (B/U) and $\alpha_1 = \beta_1 = 0.7$, $\alpha_2 = 0.3, \beta_2 = 0$ and $\beta_3 = 0.3$ (see Section 4.3.3 for more details about hyperparameters). Time-specific Batch Normalization, original BN and Layer Normalization (LN) were trained with 64 S/B, 1 B/U (32 S/B, 2 B/U would give similar performance, if not worse). Streaming Normalization clearly outperforms other methods in training. Streaming Normalization converges **more than twice** as fast as LN. Note that 32 S/B 2 B/U and 64 S/B 1 B/U are equivalent to LN since it processes samples independently. Original BN fails on testing.

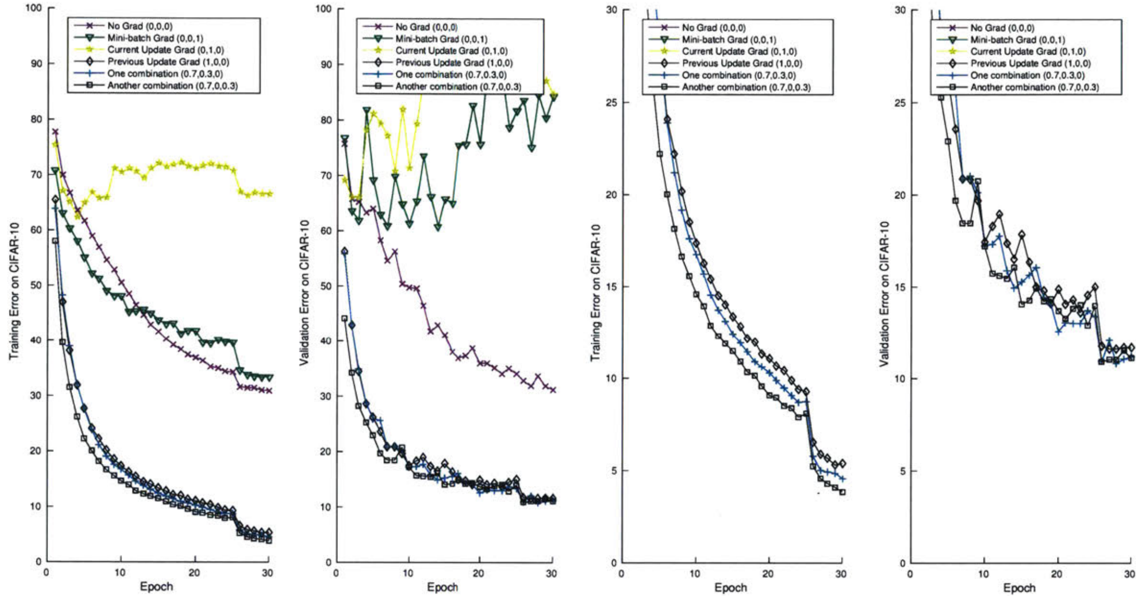


Figure 4-10: Evaluate different choices of hyperparameter β_1,β_2 and β_3. The architecture is a **recurrent and convolutional** network (shown in Figure 4-2 C with unrolling parameters $k_1 = k_2 = 5$). The right two pannels are **zoomed-in versions** of the left two pannels. The models are Streaming Normalization with "L2 norm" (Setting B with p=2 in Section 4.3.4), 32 Samples per Batch (S/B), 2 Batches per Update (B/U) and α_1, $\alpha_2 = 0.3, \kappa_1 = \kappa_3 = 0.7, \kappa_2 = \kappa_4 = 0.3$. The hyperparameters $(\beta_1, \beta_2, \beta_3)$ are shown in the figure. (0,0,0) means that the gradients of NormStats are ignored. (0,0,1) means only using NormStats gradients from the current mini-batch. (0,1,0) means only using NormStats gradients accumulated since the last weight update. Note that regardless the values of β, the gradients of NormStats are always accumulated (See Section 4.3.3). Using gradients from the previous weight update (i.e., 1,0,0) seems to work reasonably well. Some combinations (i.e., (0.7,0.0.3) or (0.7,0.0.3)) of previous and current gradients seem to give the best performances. This experiment indicates that streaming the gradients of NormStats is very important for performance.

trained with learning rate 0.01 for 2 epochs and 0.001 for 1 epoch on a text file of all Shakespeare's work concatenated. We use 99% the text file for training and 1% for validation. The training and validation softmax losses are reported. Training losses are from mini-batches so they are noisy, and we smoothed them using moving averages of 50 neighbors (using the Matlab *smooth* function). The test loss on the entire validation set is evaluated and recorded every 20 mini-batches. We show in Figure 4-11 and 4-12 the performances of Time-specific Batch Normalization, Layer Normalization and Streaming Normalization. Truncated BPTT was performed with 100 timesteps.

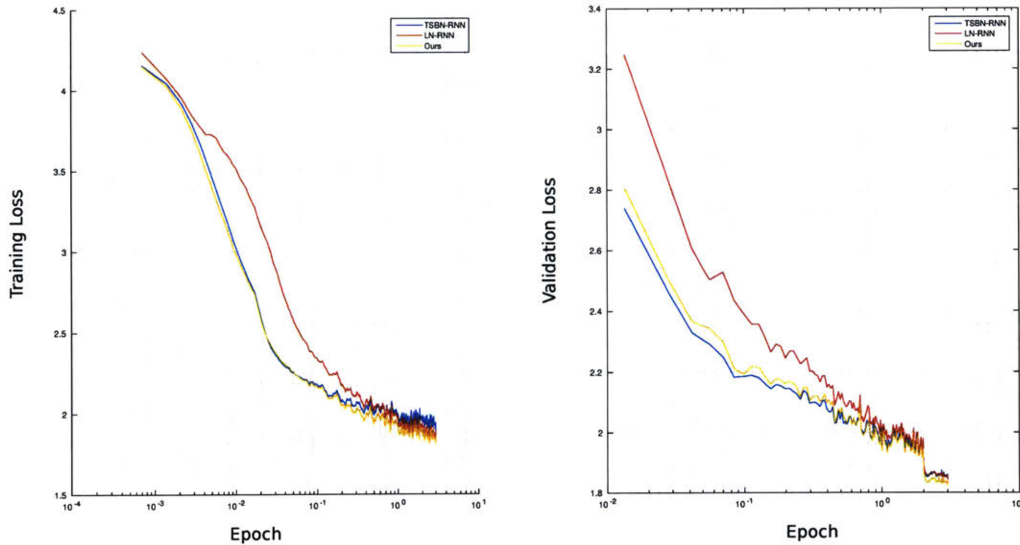


Figure 4-11: Character-level language modeling with RNN on Shakespeare's work concatenated. The training (left) and validation (right) softmax losses are reported. "Ours" refers to Streaming Normalization with "L2 norm" (Setting B with p=2 in Section 4.3.4), 32 Samples per Batch (S/B), 2 Batches per Update (B/U) and $\alpha_1 = \beta_1 = 0.7$, $\alpha_2 = 0.3, \beta_2 = 0$ and $\beta_3 = 0.3$ (see Section 4.3.3 for more details about hyperparameters). TSBN: time-specific BN. LN: Layer Normalization. Both TSBN and Streaming Normalization (SN) converges faster than LN. SN reaches slightly lower loss than TSBN and LN.

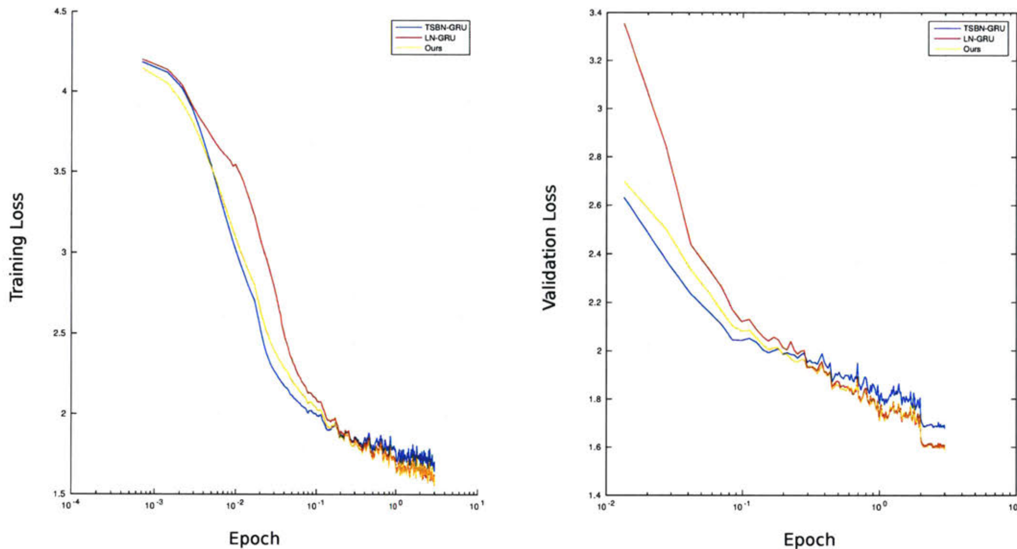


Figure 4-12: Character-level language modeling with GRU on Shakespeare's work concatenated. The training (left) and validation (right) softmax losses are reported. "Ours" refers to Streaming Normalization with "L2 norm" (Setting B with p=2 in Section 4.3.4), 32 Samples per Batch (S/B), 2 Batches per Update (B/U) and $\alpha_1 = \beta_1 = 0.7$, $\alpha_2 = 0.3, \beta_2 = 0$ and $\beta_3 = 0.3$ (see Section 4.3.3 for more details about hyperparameters). TSBN: time-specific BN. LN: Layer Normalization. Streaming Normalization converges faster than LN and reaches lower loss than TSBN.

4.8 Discussion

Biological Plausibility

We found that the simple "Neuron-wise normalization" (BA6 in Figure 4-1 D) performs very well (Figure 4-6 and 4-7). This setting does not require collecting normalization statistics from any other neurons. We show the streaming version of neuron-wise normalization in Figure A1, and the performance is again competitive. In neuron-wise normalization, each neuron simply maintains running estimates of its own mean and variance (and related gradients), and all the information is maintained locally. This approach may serve as a baseline model for biological homeostatic plasticity mechanisms (e.g., Synaptic Scaling) [83, 79, 82], where each neuron internally maintains some normalization/scaling factors that depend on neuron's firing history and can be applied and updated in a pure online fashion.

Lp Normalization

Our observations about Lp normalization have several biological implications: First, we show that most Lp normalizations work similarly, which suggests that there might exist a large class of statistics that can be used for normalization. Biological systems could implement any of these methods to get the same level of performance. Second, L1 normalization is particularly interesting, since its gradient computations are much easier for biological neurons to implement.

As an orthogonal direction of research, it would also be interesting to study the relations between our Lp normalization (standardizing the average Lp norm of activations) and Lp regularization (discounting the the Lp norm of weights, e.g., L1 weight decay).

Theoretical Understanding

Although normalization methods have been empirically shown to significantly improve the performance of deep learning models, there is not enough theoretical understanding about them. Activation-normalized neurons behave more similarly to biological neurons whose activations are constrained into a certain range: is it a blessing or a curse? Does it affect approximation bounds of shallow and deep networks [53, 54]? It would also be interesting to see if certain normalization methods can mitigate the problems of poor local minima and saddle points, as the problems have been analysed without normalization [38].

Internal Covariate Shift in Recurrent Networks

Note that our approach (and perhaps the brain's "synaptic scaling") does not normalize differently for each timestep. Thus, it does not naturally handle internal covariate shift [35] (more precisely, covariate shift over time) in recurrent networks, which was the main motivation of the original Batch Normalization and Layer Normalization. Our results seem to suggest that internal covariate shift is not as hazardous as previously believed as long as the entire network's activations are normalized to a good range. But more research is needed to answer this question.

4.9 Appendix: Other Variants of Streaming Normalization

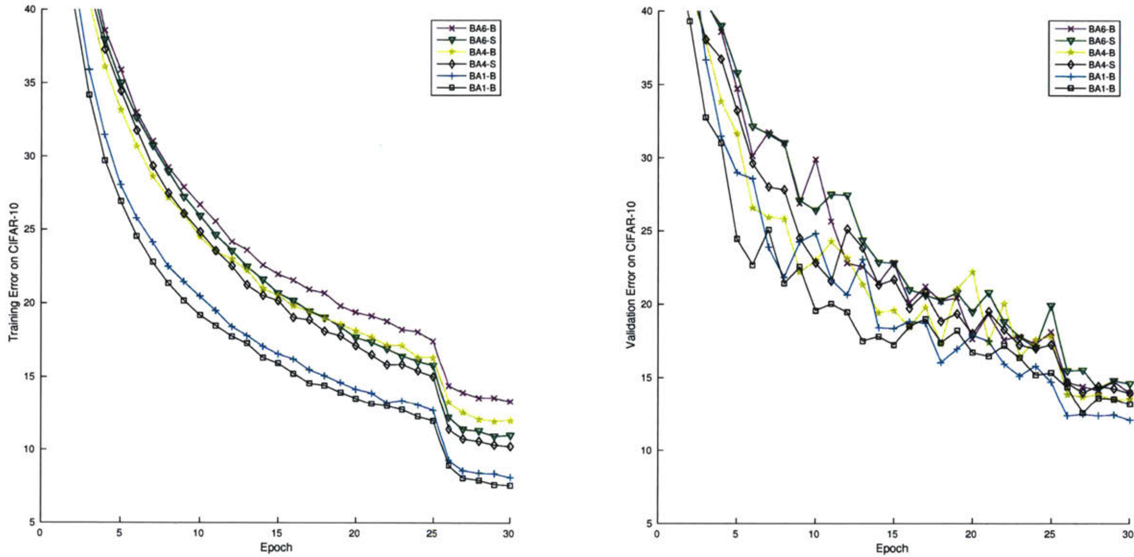


Figure A1: We explore other variants of Streaming Normalization with different NormRef (e.g., SP1-SP5, BA1-BA6 in 4-1 C and D) within each mini-batch. **-B** denotes the batch version. **-S** denotes the streaming version. The architecture is a **feedforward and convolutional** network (shown in Figure 4-2 B). Streaming normalization lowers training errors.

Chapter 5

Conclusions and Future Work

In this thesis I discussed three aspects of Deep Neural Networks (DNNs) that can be made more biologically-plausible. I propose models and algorithms that are both biologically implementable and well-performing.

In contrast to traditional DNNs, the proposed systems and algorithms better model the brain and may provide some guidance to future experimental research in neuroscience. On the other hand, developing biology-compatible frameworks for ANNs facilitates the design of biologically inspired AI systems.

There are, of course, more to be done in this emerging area of research, since biological plausibility of Deep Learning is an under-investigated problem. For example, future work should answer the following important open questions:

- How does the brain implement generative models? e.g., for visual imagery.

- How to train recurrent networks in a biologically plausible manner?

Bibliography

[1] O Abdel-Hamid, A Mohamed, H Jiang, and G Penn. Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4277–4280, 2012.

[2] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. *arXiv preprint arXiv:1512.02595*, 2015.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[4] Yoshua Bengio. How auto-encoders could provide credit assignment in deep networks via target propagation. *arXiv preprint arXiv:1407.7906*, 2014.

[5] Yoshua Bengio, Dong-Hyun Lee, Jorg Bornschein, and Zhouhan Lin. Towards biologically plausible deep learning. *arXiv preprint arXiv:1502.04156*, 2015.

[6] Christian Büchel and KJ Friston. Modulation of connectivity in visual pathways by attention: cortical interactions evaluated with structural equation modelling and fmri. *Cerebral cortex*, 7(8):768–778, 1997.

[7] Isaac Caswell, Chuanqi Shen, and Lisa Wang. Loopy neural nets: Imitating feedback loops in the human brain. *CS231n Report, Stanford*, 2016, `http://cs231n.stanford.edu/reports2016/110_Report.pdf`. Google Scholar time stamp: March 25th, 2016.

[8] Lakshminarayan V Chinta and Douglas B Tweed. Adaptive optimal control without weight transport. *Neural computation*, 24(6):1487–1518, 2012.

[9] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[10] Adam Coates, Andrew Y Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *International conference on artificial intelligence and statistics*, pages 215–223, 2011.

[11] Tim Cooijmans, Nicolas Ballas, César Laurent, and Aaron Courville. Recurrent batch normalization. *arXiv preprint arXiv:1603.09025*, 2016.

[12] Francis Crick. The recent excitement about neural networks. *Nature*, 337(6203):129–132, 1989.

[13] Sven Eberhardt, Jonah Cader, and Thomas Serre. How deep is the feature analysis underlying rapid visual categorization? *arXiv preprint arXiv:1606.01167*, 2016.

[14] David Eigen, Jason Rolfe, Rob Fergus, and Yann LeCun. Understanding deep architectures using a recursive convolutional network. *arXiv preprint arXiv:1312.1847*, 2013.

[15] Salah El Hihi and Yoshua Bengio. Hierarchical recurrent neural networks for long-term dependencies. Citeseer.

[16] Sean Ryan Fanello, Carlo Ciliberto, Maurizio Santoro, Lorenzo Natale, Giorgio Metta, Lorenzo Rosasco, and Francesca Odone. icub world: Friendly robots help building good vision data-sets. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2013 IEEE Conference on*, pages 700–705. IEEE, 2013.

[17] Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer Vision and Image Understanding*, 106(1):59–70, 2007.

[18] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, April 1980.

[19] John Garofolo, Lori Lamel, William Fisher, Jonathan Fiscus, David Pallett, Nancy Dahlgren, and Victor Zue. Timit acoustic-phonetic continuous speech corpus.

[20] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[21] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[22] Gregory Griffin, Alex Holub, and Pietro Perona. Caltech-256 object category dataset. 2007.

[23] Stephen Grossberg. Competitive learning: From interactive activation to adaptive resonance. *Cognitive science*, 11(1):23–63, 1987.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. *arXiv preprint arXiv:1603.05027*, 2016.

[27] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

[28] Geoffrey E Hinton and James L McClelland. Learning representations by recirculation. In *Neural information processing systems*, pages 358–366. New York: American Institute of Physics, 1988.

[29] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

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

[31] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[32] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Weinberger. Deep networks with stochastic depth. *arXiv preprint arXiv:1603.09382*, 2016.

[33] Gary B. Huang, Marwan Mattar, Tamara Berg, and Eric Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. In *Workshop on faces in real-life images: Detection, alignment and recognition (ECCV)*, Marseille, Fr, 2008.

[34] JM Hupe, AC James, BR Payne, SG Lomber, P Girard, and J Bullier. Cortical feedback improves discrimination between figure and background by v1, v2 and v3 neurons. *Nature*, 394(6695):784–787, 1998.

[35] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[36] Minami Ito and Charles D Gilbert. Attention modulates contextual influences in the primary visual cortex of alert monkeys. *Neuron*, 22(3):593–604, 1999.

[37] Andrej Karpathy. Lessons learned from manually classifying cifar-10. 2011.

[38] Kenji Kawaguchi. Deep learning without poor local minima. In *Advances in Neural Information Processing Systems (NIPS)*, 2016. to appear.

[39] A Krizhevsky, I Sutskever, and G Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, Lake Tahoe, CA, 2012.

[40] Alex Krizhevsky. Learning multiple layers of features from tiny images, 2009.

[41] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[42] Victor AF Lamme, Hans Super, and Henk Spekreijse. Feedforward, horizontal, and feedback processing in the visual cortex. *Current opinion in neurobiology*, 8(4):529–535, 1998.

[43] César Laurent, Gabriel Pereyra, Philémon Brakel, Ying Zhang, and Yoshua Bengio. Batch normalized recurrent neural networks. *arXiv preprint arXiv:1510.01378*, 2015.

[44] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

[45] Yann Le Cun. Learning process in an asymmetric threshold network. In *Disordered systems and biological organization*, pages 233–240. Springer, 1986.

[46] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The mnist database.

[47] Joel Z. Leibo, Qianli Liao, and Tomaso Poggio. Subtasks of Unconstrained Face Recognition. In *International Joint Conference on Computer Vision, Imaging and Computer Graphics, VISIGRAPP*, Lisbon, Portugal, 2014.

[48] Ming Liang and Xiaolin Hu. Recurrent convolutional neural network for object recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3367–3375, 2015.

[49] Qianli Liao, Joel Z Leibo, and Tomaso Poggio. How important is weight symmetry in backpropagation? *arXiv preprint arXiv:1510.05067*, 2015.

[50] Qianli Liao and Tomaso Poggio. Bridging the gaps between residual learning, recurrent neural networks and visual cortex. *arXiv preprint arXiv:1604.03640*, 2016.

[51] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random feedback weights support learning in deep neural networks. *arXiv preprint arXiv:1411.0247*, 2014.

[52] Pietro Mazzoni, Richard A Andersen, and Michael I Jordan. A more biologically plausible learning rule for neural networks. *Proceedings of the National Academy of Sciences*, 88(10):4433–4437, 1991.

[53] Hrushikesh Mhaskar, Qianli Liao, and Tomaso Poggio. Learning real and boolean functions: When is deep better than shallow. *arXiv preprint arXiv:1603.00988*, 2016.

[54] Hrushikesh Mhaskar and Tomaso Poggio. Deep vs. shallow networks: An approximation theory perspective. *arXiv preprint arXiv:1608.03287*, 2016.

[55] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3111–3119, 2013.

[56] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*, pages 2924–2932, 2014.

[57] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 5, 2011.

[58] Behnam Neyshabur, Ruslan R Salakhutdinov, and Nati Srebro. Path-sgd: Path-normalized optimization in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2422–2430, 2015.

[59] Maria-Elena Nilsback and Andrew Zisserman. A visual vocabulary for flower classification. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*. IEEE, 2006.

[60] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Computer Vision, Graphics & Image Processing, 2008. ICVGIP'08. Sixth Indian Conference on*. IEEE, 2008.

[61] Randall C O'Reilly. Biologically plausible error-driven learning using local activation differences: The generalized recirculation algorithm. *Neural computation*, 8(5):895–938, 1996.

[62] Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*, 2013.

[63] Pedro HO Pinheiro and Ronan Collobert. Recurrent convolutional neural networks for scene parsing. *arXiv preprint arXiv:1306.2795*, 2013.

[64] Nicolas Pinto, Zak Stone, Todd Zickler, and David Cox. Scaling up biologically-inspired computer vision: A case study in unconstrained face recognition on facebook. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*, pages 35–42. IEEE, 2011.

[65] Ariadna Quattoni and Antonio Torralba. Recognizing indoor scenes. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 413–420. IEEE, 2009.

[66] Rajesh PN Rao and Dana H Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature neuroscience*, 2(1):79–87, 1999.

[67] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *Neural Networks, 1993., IEEE International Conference on*, pages 586–591. IEEE, 1993.

[68] M Riesenhuber and T Poggio. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2(11):1019–1025, November 1999.

[69] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 1988.

[70] Tim Salimans and Diederik P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *arXiv preprint arXiv:1602.07868*, 2016.

[71] Jürgen Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242, 1992.

[72] Thomas Serre, Aude Oliva, and Tomaso Poggio. A feedforward architecture accounts for rapid categorization. *Proceedings of the National Academy of Sciences of the United States of America*, 104(15):6424–6429, 2007.

[73] Hava T Siegelmann and Eduardo D Sontag. On the computational power of neural nets. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 440–449. ACM, 1992.

[74] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[75] Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. 1986.

[76] Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 129–136, 2011.

[77] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[78] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[79] David Stellwagen and Robert C Malenka. Synaptic scaling mediated by glial tnf-α. *Nature*, 440(7087):1054–1059, 2006.

[80] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lars Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1701–1708. IEEE, 2014.

[81] Simon Thorpe, Denis Fize, Catherine Marlot, et al. Speed of processing in the human visual system. *nature*, 381(6582):520–522, 1996.

[82] Gina G Turrigiano. The self-tuning neuron: synaptic scaling of excitatory synapses. *Cell*, 135(3):422–435, 2008.

[83] Gina G. Turrigiano and Sacha B. Nelson. Homeostatic plasticity in the developing nervous system. *Nature Reviews Neuroscience*, 2004.

[84] S Ullman and G Schechtman. Adaptation and gain normalization. *Proceedings of the Royal Society of London B: Biological Sciences*, 216(1204):299–313, 1982.

[85] Andrea Vedaldi and Karel Lenc. Matconvnet: Convolutional neural networks for matlab. In *Proceedings of the 23rd Annual ACM Conference on Multimedia Conference*, pages 689–692. ACM, 2015.

[86] Daniel LK Yamins and James J DiCarlo. Using goal-driven deep learning models to understand sensory cortex. *Nature neuroscience*, 19(3):356–365, 2016.

[87] Elham Zamanidoost, Farnood M Bayat, Dmitri Strukov, and Irina Kataeva. Manhattan rule training for memristive crossbar circuit pattern classifiers. 2015.

[88] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Rob Fergus. Deconvolutional networks. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2528–2535. IEEE, 2010.